การควบคุมเครื่องใช้ไฟฟ้าผ่านบอร์ดอีเธอร์เนต ไอ/โอ

CONTROLLING ELECTRONIC DEVICES VIA ETHERNET I/O

โดย

นาย นรินทร์ ลีลาวิวัฒน์

นาย นฤศันส์ ธันวารชร

นาย ศุภเวทย์ วิชชาวุธ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

การควบคุมเครื่องใช้ไฟฟ้าผ่านบอร์ดอีเธอร์เนต ไอ/โอ

CONTROLLING ELECTRONIC DEVICES VIA ETHERNET I/O

โดย

นาย นรินทร์ ลีลาวิวัฒน์ 45010379

นาย นฤศันส์ ธันวารชร 45010382

นาย ศุภเวทย์ วิชชาวุธ 45010787

อาจารย์ที่ปรึกษา

ผศ. นภัทร สระเอี่ยม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชา วิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมเครื่องใช้ไฟฟ้าผ่านบอร์ดอีเธอร์เนต ไอ/โอ

CONTROLLING ELECTRONIC DEVICES VIA ETHERNET I/O

โดย นาย นรินทร์ ลีลาวิวัฒน์ 45010379

นาย นฤศันส์ ธันวารชร 45010382

นาย ศุภเวทย์ วิชชาวุธ 45010787

.....อาจารย์ที่ปรึกษา

(ผศ. นภัทร สระเอี่ยม)

ควบคุมเครื่องใช้ไฟฟ้าผ่านบอร์ดอีเธอร์เนตไอ/โอ

CONTROLLING ELECTRONICS DEVICES VIA ETHERNET I/O

โดย นาย นรินทร์ ลีลาวิวัฒน์ 45010379
นาย นฤสันต์ ธันวารชร 45010382
นาย ศุภเวทย์ วิชชาวุธ 45010787

อาจารย์ที่ปรึกษา ผศ. นภัทร สระเอี่ยม

บทคัดย่อ

โครงงานนี้เป็นโครงงานที่ใช้ในการควบคุมการปิดปิดไฟฟ้าภายในอาคารหรือสำนักงานผ่านทางคอมพิวเตอร์โดยมีบอร์ดอีเธอร์เนต ไอ/โอ เป็นตัวควบคุม ระบบนี้จะมีประโยชน์เพื่ออำนวยความสะดวกต่อผู้ใช้ในการควบคุมเครื่องใช้ไฟฟ้าภายในอาคารสำนักงานผ่านทางระบบเครือข่าย ของบริษัทโดยที่ไม่ต้องการคอมพิวเตอร์ในการรับส่งข้อมูลควบคุม

Abstract

This Project is used to control the electric device in the office via computer network by using, embedded network controller, Ethernet I/O. This system is comfortable for user to turn on or turn off the electronic devices remotely and in local area network.

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูปภาพ

บทที่ 2 ทฤษฎีและหลักการ

บทที่ 3 การคำนวณและการสร้าง

สารบัญรูปภาพ(ต่อ)

สารบัญตาราง

บทที่ 2 ทฤษฎีและหลักการ

บทที่ 4 การทดลองและผลการทดลอง

บทที่ 1 บทนำ

1.1 ความเป็นมาและเหตุจูงใจในการวิจัย

ปัจจุบันเทคโนโลยีในการควบคุมอุปกรณ์ต่างได้พัฒนาขึ้นอย่างมาก ไม่ว่าจะเป็นการควบคุมโดยใช้คอมพิวเตอร์ การควบคุมด้วยรีโมตคอนโทรล การควบคุมผ่านระบบโทรศัพท์ การที่สามารถควบคุมอุปกรณ์ต่างๆได้ในระยะไกลนั้นทำให้สะดวกสบายเป็นอย่างมาก

การควบคุมอุปกรณ์ต่างๆผ่านระบบ LAN โดยใช้คอมพิวเตอร์ควบคุมนั้นได้มีการใช้กันอย่างแพร่หลายแต่ยังต้องใช้เครื่องคอมพิวเตอร์ในการควบคุม 2 เครื่องทำหน้าที่เป็นแบบ Client – Server ซึ่งทำให้สูญเสียทรัพยากรเป็นอย่างมากจึงได้มีการพัฒนาอุปกรณ์ในการควบคุมผ่านระบบ LAN โดยไม่ต้องใช้คอมพิวเตอร์ขึ้น

1.2 วัตถุประสงค์ของปริญญานิพนธ์

โครงงานนี้ได้ทำออกแบบและควบคุมบอร์ดอีเธอร์เนตไอโอซึ่งเป็นอุปกรณ์ในการควบคุมผ่านระบบ LAN ให้มีประสิทธิภาพและใช้งานได้จริง

1.3 ขอบเขตของรายงาน

รายงานนี้จะกล่าวถึง การควบคุมและควบคุมบอร์ดอีเธอร์เนตไอโอจากเครื่องคอมพิวเตอร์ผ่านระบบLAN โดยการควบคุมอุปกรณ์ไฟฟ้าจากบอร์ดอีเธอร์เนตไอโอโดยตรง หรือจะเป็นการที่บอร์ดอีเธอร์เนตไอโอส่งข้อมูลควบคุมผ่านไมโครคอนโทรลเลอร์ไปควบคุมอุปกรณ์ไฟฟ้าอีกทีหนึ่งโดยใช้การรับ-ส่งข้อมูลผ่านพอร์ตอนุกรม RS-232 นอกจากนี้ยังสามารถรับข้อมูลของสภาพแวดล้อมกลับมายังเครื่องคอมพิวเตอร์ได้อีกด้วย

1.4 ขั้นตอนการศึกษา

สำหรับการรายงานฉบับนี้นี้เริ่มจากการศึกษาโครงสร้างและการทำงานของบอร์ดอีเธอร์เนตไอโอ โดยใช้โปรแกรม Visual C++ Version 6.0 ในการเขียนโปรแกรมควบคุมการทำงานของบอร์ดอีเธอร์เนตไอโอ จากนั้นทำการศึกษาโครงสร้างและการทำงานของไมโครคอนโทรเลอร์ MCS-51 โปรแกรม C/C++ เพื่อนำไปใช้เขียนโปรแกรมควบคุมการทำงานของไมโครคอนโทรเลอร์ MCS-51 การรับส่งข้อมูลระหว่างคอมพิวเตอร์และบอร์ดอีเธอร์เนตไอโอ การรับส่งข้อมูลระหว่างบอร์ดอีเธอร์เนตไอโอและไมโครคอนโทรเลอร์ การรับส่งข้อมูลระหว่างไมโครคอนโทรเลอร์และอุปกรณ์ที่ถูกควบคุม หลังจากนั้นจะทำการศึกษาส่วนต่างๆ ที่จะใช้เป็นองค์ประกอบในการควบคุมเช่น การทำงานของการวัดอุณหภูมิของสภาพแวดล้อม การกำหนดการเปิดและปิดอุปกรณ์ไฟฟ้า

บทที่2 ทฤษฎีและหลักการ

2.1 อุปกรณ์ควบคุมผ่านระบบเครือข่าย

Embedded Network Controller คืออุปกรณ์ที่มีความสามรถในการเชื่อมต่อเข้ากับ LAN ได้ด้วยตัวของมันเองโดยมี Embedded Network Controller ป้อนคำสั่งใช้งานเพื่อใช้ในการเชื่อมต่อเข้ากับระบบ LAN ภายในตัวมันเอง ซึ่งตัว Embedded Network Controller นี้จะสามารถใช้งานแทนคอมพิวเตอร์ในการรับส่งข้อมูลเพื่อทำการควบคุมอุปกรณ์ต่างๆผ่านระบบเครือข่ายได้

ซึ่งข้อมูลจะอยู่ในรูปแบบของเฟรมข้อมูลซึ่งประกอบด้วยแพคเกจข้อมูลโดยใช้การรับส่งข้อมูลลักษณะแบบเป็นชั้นๆ เรียกการับส่งข้อมูลลักษณะนี้ว่า TCP/IP Protocol Stack ซึ่งตัวโปรโตคอลเลเยอร์นี้จะทำการจัดเตรียมและตรวจสอบข้อมูลในลักษณะเป็นชั้นๆ เพื่อทำการตรวจสอบและจัดการรับส่งข้อมูลว่าถูกต้องหรือไม่ในการรับส่งข้อมูลผ่าน LAN

2.2 บอร์ดอีเธอร์เนตไอโอ

เป็นบอร์ด Embedded System ที่สำหรับใช้งานควบคุมผ่านระบบ LAN โดยสั่งควบคุมจากคอมพิวเตอร์ที่เชื่อมต่ออยู่กับ LAN วงเดียวกันกับบอร์ดอีเธอร์เนตไอโอ โดยใช้โปรโตคอล TCP/IP ในการรับส่งข้อมูล

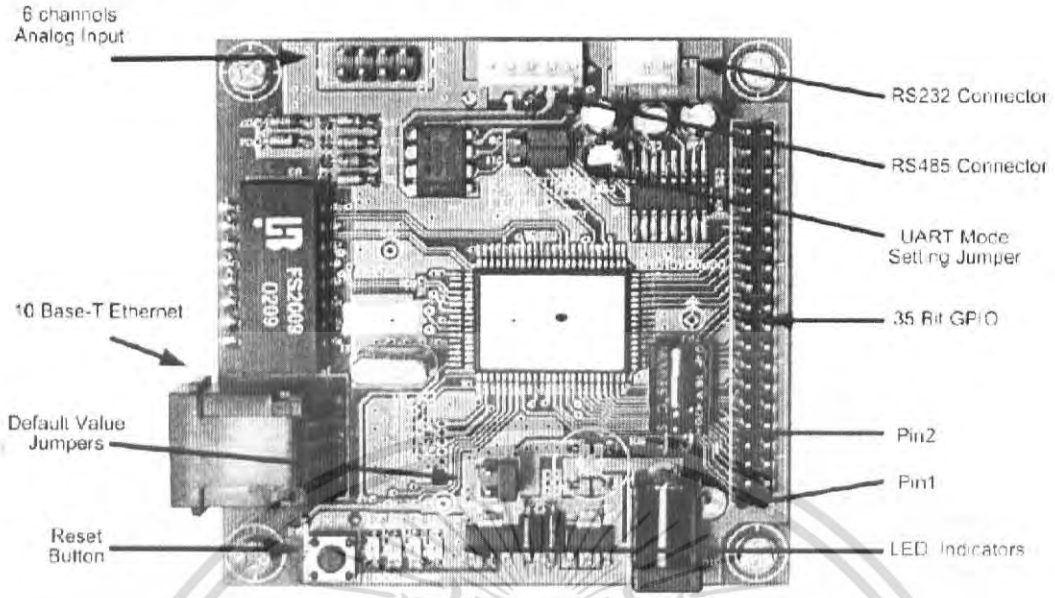
2.2.1 คุณสมบัติของบอร์ดอีเธอร์เนตไอโอ

1. สามารถติดต่อได้กับอีเธอร์เนตแบบ 10Base-T
2. มีหน่วยความจำ 64 กิโลไบต์
3. 16 กิโลไบต์ SRAM Program
4. 4 กิโลไบต์ SRAM Data
5. มี พอร์ต ทั้งแบบ RS232 และ RS 485 ด้วยความเร็ว 115200 บิตต่อวินาที
6. มี 35 บิตในการใช้งานทั่วไป ใช้ไฟ 5 โวลต์ เป็นอินพุตและให้ไฟ 2.5 โวลต์เป็นเอาท์พุต
7. 10 บิต, 6 ช่องสัญญาณ ความถี่การสุ่มสัญญาณสูงสุด 48 Hz
8. มีความเร็วในการส่งข้อมูลแบบ ขนาน 200 บิตต่อวินาที

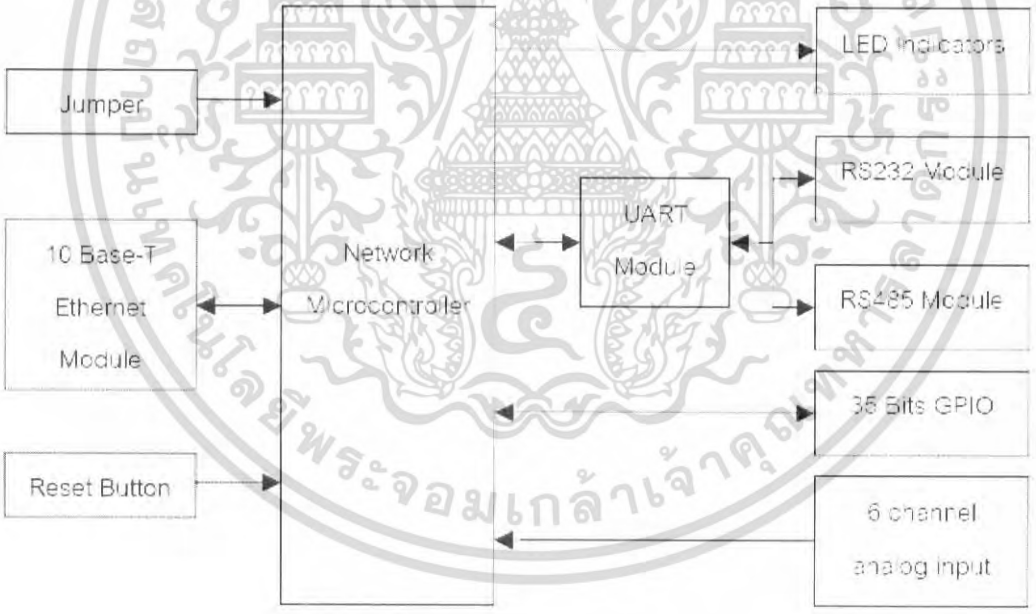
2.2.2 ระบบ ปฏิบัติงานที่ต้องการ

1. ระบบ ปฏิบัติการ Windows 2000 หรือ Windows XP
2. พื้นที่การใช้งาน เมกะบิต
3. ความต้องการ ของ RAM ที่โปรแกรมต้องการคือ 64 เมกะบิตสำหรับ Windows 2000 และ 128 เมกะบิตสำหรับ Windows XP
4. 10/100 เมกะบิตต่อวินาที สำหรับ LAN Card
5. โปรแกรม Microsoft Visual C++ 6.0

2.3 ลักษณะและรายละเอียดของ บอร์ดอีเธอร์เนต ไอโอ



รูปที่ 2.1 บอร์ดอีเธอร์เนต ไอ / โอ



รูปที่ 2.2 บล็อคไดอะแกรมของบอร์ดอีเธอร์เนต ไอ / โอ

2.3.1 รายละเอียดต่างๆของบอร์ด อีเธอร์เนตไอโอ

2.3.1.1 ส่วนที่ติดใช้ติดต่อกับอีเธอร์เนต แบบ 10 Base-T

2.3.1.2 ส่วนที่ใช้ติดต่อในส่วนของ UART

โดยในส่วนนี้นั้นสามารถใช้ติดต่อกับส่วนของ RS232 หรือ RS485 ได้โดยที่สามรถเลือกใช้ได้เพียงอย่างใดอย่างหนึ่ง ไม่สามารใช้พร้อมกันทั้งสองตัวได้ โดยการเลือกใช้งานตัวใดตัวหนึ่งนั้นสามารถตั้งค่าได้ที่ Jumper Module

UART Mode	J8		J11	
	1-2	2-3	1-2	2-3
RS232	Closed	Open	Closed	Open
RS485	Open	Closed	Open	Closed

ตารางที่ 2.1 ตาราง UART Settings

2.3.1.3 ส่วนที่สื่อสารผ่าน RS485

ซึ่งโดยปกติแล้วการสื่อสารผ่านทาง RS485 นั้น สามารถสื่อสารได้ระยะทางที่ไกลมากถึงประมาณ 1000 เมตรโดยมีตารางในการใช้งานดังตารางที่ 2.2

J9 Pin#	ชื่อของสัญญาณ	คำอธิบาย
1	TX+	Noninverting Driver Output
2	TX-	Inverting Driver Output
3	Gnd	Ground
4	RX+	Noninverting Receiver Input
5	RX-	Inverting Receiver input

ตารางที่ 2.2 การใช้งานการสื่อสารผ่าน RS485

2.3.1.4 ส่วนที่สื่อสารผ่านทาง RS232

ซึ่งในส่วนนี้อาไว้ใช้ในการสื่อสารระหว่างตัวบอร์ดกับส่วนของคอมพิวเตอร์ PC โดยมีการใช้งานภายในบอร์ดตามตารางที่ 2.3

J13 Pin#	ชื่อของสัญญาณ	คำอธิบาย
1	TX	Transmitted data from Ethernet IO board
2	RX	Received data to Ethernet IO board
3	Gnd	Ground

ตารางที่ 2.3 การใช้งานการสื่อสารโดยใช้ RS232

2.3.1.5 6 ช่องสัญญาณอนาลอกอินพุท

ครอบคลุมถึง 6 ช่องสัญญาณ เปลี่ยนสัญญาณอนาลอกเป็นดิจิตอลโดยใช้บิต 10 บิต มี Sampling rate สูงสุดคือ 48 kHz มีค่าแรงดันกระแสไฟฟ้าอ้างอิงเป็น +2.5 โวลต์ แรงดันสูงสุดที่อ่านได้คือ 0x3FF ดังนั้นแรงดันกระแสไฟฟ้าอ้างอิง (+2.5 โวลต์) จะมีค่าเท่ากับ 0x3FE และ ACDs resolution is 10 bits ตารางที่2. 4 แสดงค่าของสัญญาณที่แปลงจากอนาลอกเป็นดิจิตอลที่ขอบเขตบนและขอบเขตล่างของแรงดันไฟฟ้าอินพุท และตาราง 2.5 แสดงขาการต่ออินพุท

แรงดันไฟฟ้าอินพุท (อนาลอก)	ค่าเอาท์พุท (ดิจิตอล)
0	0x000
2.5 โวลต์/0x3FE	0x001
2.5 โวลต์	0x3FE
2.5 โวลต์ + (2.5 โวลต์ / 0x3FE)	0x3FF

ตารางที่ 2.4 การแปลงไฟฟ้าจากอนาลบอกเป็นดิจิตอล

J3 ขาที่	ชื่อของสัญญาณ	คำอธิบาย
1	AIN0	อนาลอกอินพุทช่อง 0
2	AIN1	อนาลอกอินพุทช่อง 1
3	AIN2	อนาลอกอินพุทช่อง 2
4	AIN3	อนาลอกอินพุทช่อง 3
5	AIN4	อนาลอกอินพุทช่อง 4
6	AIN5	อนาลอกอินพุทช่อง 5
7	AGND	อนาลอกกราวด์
8	AGND	อนาลอกกราวด์

ตารางที่ 2.5 ขาของอนาลอกอินพุท

2.3.1.6 35 บิต อินพุทเอาท์พุท

บอร์ด อีเธอร์เนตไอโอมีช่องสัญญาณสำหรับควบคุมอุปกรณ์โดยใช้บิตได้ถึง 35 บิตในการควบคุมและสามารถอ่านและเขียนข้อมูล 16 บิตได้ด้วย ตารางที่ 2.6 แสดงถึงหน้าที่ของขาต่างๆ

ขาที่	ชื่อสัญญาณ	ชื่อพอร์ท	ลักษณะ	กระแสไฟ(mA)		รายละเอียด
				Sink	Source	
1	+5 V	-	-	-	-	Power +5V
2	GPIO1	RA3	I/O	24	24	พอร์ท อินพุทเอาท์พุท
3	A0	RB0	I/O	8	8	Address 0
4	A1	RB1	I/O	8	8	Address 1
5	GPIO2	RB2	I/O	8	8	พอร์ท อินพุทเอาท์พุท
6	GPIO3	RB3	I/O	8	8	พอร์ท อินพุทเอาท์พุท
7	Wr	RB4	I/O	8	8	เขียนข้อมูล(ทำงานที่ 0)
8	Rd	RB5	I/O	8	8	อ่านข้อมูล(ทำงานที่ 0)
9	GPIO4	RB6	I/O	8	8	พอร์ท อินพุทเอาท์พุท
10	GPIO5	RB7	I/O	8	8	พอร์ท อินพุทเอาท์พุท
11	D8	RC0	I/O	4	4	ข้อมูล 8
12	D9	RC1	I/O	4	4	ข้อมูล 9
13	D10	RC2	I/O	4	4	ข้อมูล 10
14	D11	RC3	I/O	4	4	ข้อมูล 11
15	D12	RC4	I/O	4	4	ข้อมูล 12
16	D13	RC5	I/O	4	4	ข้อมูล 13
17	D14	RC6	I/O	4	4	ข้อมูล 14
18	D15	RC7	I/O	4	4	ข้อมูล 15
19	Gnd	-	-	-	-	กราวด์
20	Gnd	-	-	-	-	กราวด์
21	D0	RD0	I/O	4	4	ข้อมูล 0
22	D1	RD1	I/O	4	4	ข้อมูล 1
23	D2	RD2	I/O	4	4	ข้อมูล 2
24	D3	RD3	I/O	4	4	ข้อมูล 3
25	D4	RD4	I/O	4	4	ข้อมูล 4
26	D5	RD5	I/O	4	4	ข้อมูล 5
27	D6	RD6	I/O	4	4	ข้อมูล 6

28	D7	RD7	I/O	4	4	ข้อมูล 7
29	GPIO6	RE0	I/O	8	8	พอร์ท อินพุทเอาท์พุท
30	GPIO7	RE1	I/O	8	8	พอร์ท อินพุทเอาท์พุท
31	GPIO8	RE2	I/O	8	8	พอร์ท อินพุทเอาท์พุท
32	GPIO9	RE3	I/O	8	8	พอร์ท อินพุทเอาท์พุท
33	GPIO10	RE4	I/O	24	24	พอร์ท อินพุทเอาท์พุท
34	GPIO11	RE5	I/O	8	8	พอร์ท อินพุทเอาท์พุท
35	GPIO12	RF4	I/O	8	8	พอร์ท อินพุทเอาท์พุท
36	GPIO13	RF5	I/O	8	8	พอร์ท อินพุทเอาท์พุท
37	GPIO14	RF6	I/O	8	8	พอร์ท อินพุทเอาท์พุท
38	GPIO15	RF7	I/O	8	8	พอร์ท อินพุทเอาท์พุท
39	Gnd	-	-	-	-	กราวด์
40	Gnd	-	-	-	-	กราวด์

ตารางที่ 2.6 แสดงขาต่างๆของอินพุดเอาท์พุดพอร์ท

2.3.1.7 ปุ่มรีเซต

สำหรับทำการรีเซตบอร์ดให้ปิดและเปิดใหม่อัตโนมัติ

2.3.1.8 จัมพ์เปอร์

บอร์ดจะมี IP Address และ Mac Address เริ่มต้นมาให้โดยมีจัมพ์เปอร์ 4 เป็นตัวกำหนดการใช้งาน IP Address และ Mac Address เริ่มต้น ถ้าต้องการกำหนด IP Address และ Mac Address เองต้องนำตัวเชื่อมต่อจัมพ์เปอร์ 4 ออก

ทั้งนี้ยังมีจัมพ์เปอร์ 6 ซึ่งผู้ใช้สามารถกำหนดหน้าที่เองได้ ตาราง 2.7 แสดงหน้าที่ของจัมพ์เปอร์ในเงื่อนไขต่างๆ

จัมพ์เปอร์	ชื่อพอร์ท	สภาวะ	ลอจิก	คำอธิบาย
JP4	RB2	ปิด	ต่ำ	ใช้ IP Address และ Mac Address เริ่มต้น
		เปิด	สูง	ใช้ IP Address และ Mac Address กำหนดเอง
JP6	RB3	ปิด	ต่ำ	ผู้ใช้กำหนดเอง
		เปิด	สูง	ผู้ใช้กำหนดเอง

ตารางที่ 2.7 แสดงการกำหนดการใช้งาน

2.3.1.9 หลอด LED

แสดงสถานะของบอร์ดโดยแสดงดังตารางที่ 2.8

LED	สัญญาณ	คำอธิบาย
D11	ไฟเข้า	สว่างเมื่อมีกระแสไฟฟ้าเข้าบอร์ด
D2	เชื่อมต่อ	สว่างเมื่อบอร์ดมีการเชื่อมต่อกับโครงข่าย
D3	รับทราบ	สว่างเมื่อส่งหรือรับข้อมูลจากโครงข่าย
D4	การชน	สว่างเมื่อการชนกันของข้อมูลเกิดขึ้น

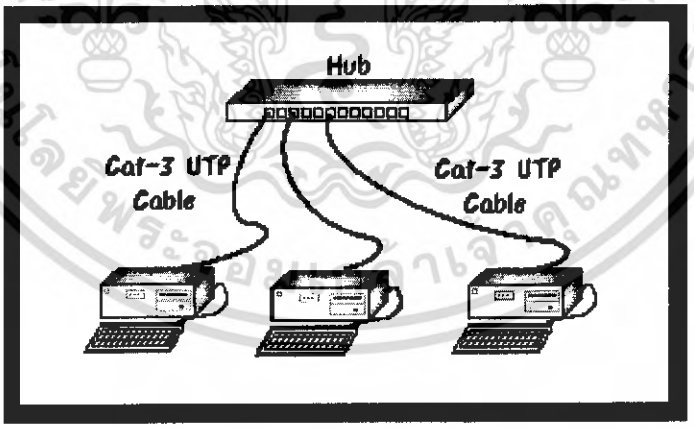
ตารางที่ 2.8 แสดงสถานะของบอร์ดทางหลอดไฟ

2.4 อีเธอร์เนต แบบ 10 Base-T

โดยปกติแล้วนั้นระบบ Ethernet ได้ถูกออกแบบมาเพื่อใช้งานกับสาย โคแอกเชี่ยล ซึ่งโดยปกติโดยทั่วไปแล้วนั้นตึกส่วนใหญ่จะเดินสายไว้เป็นแบบ twisted-pair ซึ่งไม่สามารถที่จะทำการวางระบบของ Ethernet ได้จึงทำให้เป็นการยุ่งยากในการเดินสายใหม่ เราจึงมีระบบที่เรียกว่า 10 base-T ขึ้นมาเพื่อ เพิ่มความยืดหยุ่นในการวางระบบร่วมกับสายเคเบิ้ลแบบ Twisted-Pair

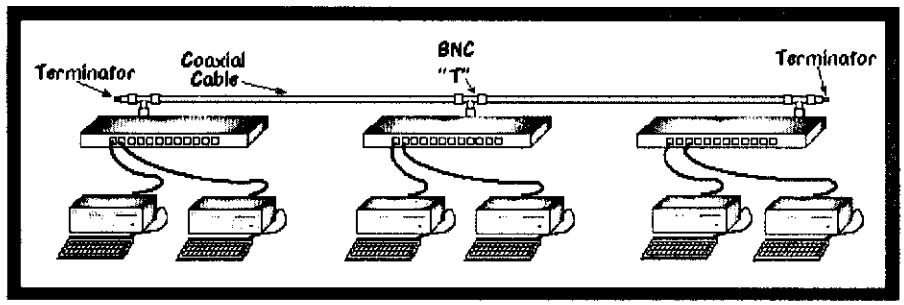
2.4.1 โทโปโลยี (Topology and Cabling)

การเดินสายของระบบ 10 Base-T นั้นใช้การเดินสายแบบ Unshielded Twisted pair (UTP) โดยที่แต่ละ node ของระบบนี้จะมีการเดินสายเคเบิ้ลของตัวเองเพื่อเข้าสู่ Hub โดยที่แต่ละสายของเคเบิ้ลนั้นสามารถเดินสายไปได้ไกลถึง 100 เมตร โดยสามารถแสดงได้ดังรูป 2.3



รูปที่2.3 ระบบโครงข่ายแบบ 10 Base-T

โดยที่การติดต่อสื่อสารหรือการทำเชื่อมโยงเครือข่ายของ 10-BASE-T นั้นสามารถทำการติดต่อได้เพียงเลเยอร์เดียวแต่สามารถใช้เชื่อมต่อกับระบบอื่นได้เช่นดังรูปที่ 2.4 โดยจากรูปเป็นการเชื่อต่อกันของระบบ 10 Base-T และ 10 Base-2



รูปที่ 2.4 การเชื่อต่อระบบ 10 Base-T กับระบบอื่น

2.4.2 ข้อได้เปรียบของระบบ 10-Base-T

1. มีความคงทนต่อความผิดพลาด เนื่องจากระบบนี้นั้นมีการเชื่อมต่อจากโหนดไปยัง ฮับ ด้วยสายสื่อสารข้อมูลแยกกัน และเมื่อเกิดความเสียหายของโหนดใดโหนดหนึ่งนั้น ทาง ฮับจะมีการตัดส่วนของโหนดนั้นออกไปจากโครงข่าย จนกระทั่งโหนดนั้นจะสามารถแก้ปัญหาของตัวเองได้
2. เนื่องจากมีการแยกการทำงาน กันของโหนดแต่ละโหนด เมื่อมีปัญหาเกิดขึ้นนั้นก็จะสามารถตรวจสอบปัญหาได้ตรงจุด เนื่องจากมีการแยกการทำงานของโหนดได้อย่างชัดเจน
3. สามารถเคลื่อนย้ายหรือเปลี่ยนจุดการติดตั้งได้ง่าย เนื่องจาก เมื่อถอดโหนดใดโหนดหนึ่งออกจากระบบแต่ก็จะไม่ส่งผลกระทบต่อ ระบบโดยรวม

2.4.3 ข้อเสียเปรียบของระบบ10-Base-T

1. มีความยาวของสายที่เชื่อมต่อระบบนั้นมีความยาวที่ จำกัดที่ประมาณ 100 เมตร เพราะฉะนั้นระบบที่ต้องการการเชื่อมต่อของสายที่มีความยาวมากจะไม่สามารถใช้ระบบ 10-Base-T ได้
2. เนื่องจากระบบนั้นใช้สายที่เชื่อมต่อแบบ Unshielded Twisted Pair นั้น จะมีความไวต่อสัญญาณรบกวนทำให้อาจเกิดการสูญเสียที่เกิดขึ้นง่ายกว่าระบบอื่น

2.5 การเชื่อมต่ออีเธอร์เน็ต

2.5.1 ซ็อกเก็ต (Socket)

ซ็อกเก็ตถูกกำหนดหรือนิยามไว้ว่า เป็นคู่ของการสื่อสาร หรือคู่ของโพรเซส (หรือเธรด) โดยที่การสื่อสารบนเน็ตเวิร์คใช้คู่ของซ็อกเก็ตสำหรับแต่ละโพรเซส ในซ็อกเก็ตหนึ่งประกอบด้วย ไอพีแอดเดรส (IP Address) ซึ่งมีขนาด 32 บิต (ในเวอร์ชัน IP4) กับหมายเลข Port (port number) ซึ่งจะมีขนาด 16 บิต

โดยทั่วไป ซ็อกเก็ตใช้สถาปัตยกรรมไคลเอนท์เซิร์ฟเวอร์ เซิร์ฟเวอร์จะรอการเข้ามาตามการขอร้องของไคลเอนท์โดยการฟังที่ พอร์ต (port) เฉพาะเมื่อการร้องขอได้รับ เซิร์ฟเวอร์ก็จะยอมรับการเชื่อมต่อจากซ็อกเก็ตไคลเอนท์เพื่อให้สมบูรณ์ในการเชื่อมต่อ

เซิร์ฟเวอร์ที่สร้างการบริการเฉพาะ เช่น telnet , ftp , mail และ http จะฟัง (listen) ที่พอร์ตมีชื่อ เช่น เซิร์ฟเวอร์ telnet จะฟังที่พอร์ต 23, เซิร์ฟเวอร์ ftp จะฟังที่พอร์ต 21 หรือเซิร์ฟเวอร์ http จะฟังที่พอร์ต 80 เป็นต้น

หมายเลขพอร์ตทั้งหมดที่ต่ำกว่า 1,024 จะถูกพิจารณาว่าเป็นพอร์ตที่ได้รับการมอบหมายหน้าที่ไว้รียบร้อยแล้ว เราสามารถใช้พอร์ตเหล่านี้เพื่อสร้างการบริการตามาตรฐานได้

2.5.2 ชนิดของซ็อกเก็ต

ชนิดของซ็อกเก็ตมีอยู่ 3 ชนิดคือ

2.5.2.1 Connection-Oriented Socket

เป็นซ็อกเก็ตการเชื่อมต่อแบบต่อเนื่องที่อนุญาตให้โพรเซสเชื่อมต่อกับโพรเซสระยะไกล (Remote) ซึ่งใช้โปรโตคอล TCP (Transmission Control Protocol) ดังนั้นด้วยวิธีการนี้ทำให้ข้อมูลเชื่อถือได้ เมื่อการเชื่อมต่อได้เกิดขึ้นโพรเซสก็จะมีการส่งข้อมูลกลับไปจนกระทั่งฝั่งได้ฝั่งหนึ่งหรืออื่นๆ มีการปิดการเชื่อมต่อ ชนิดของซ็อกเก็ตนี้ บางครั้งเรียกว่า สตรีมซ็อกเก็ต (Stream Socket) ทั้ง ftp และ http ใช้ซ็อกเก็ตแบบนี้ในการสื่อสาร

2.5.2.2 Connectionless Socket

หรือเรียกอีกอย่างว่า ดาต้าแกรม (Datagram) เป็นซ็อกเก็ตแบบไม่ต่อเนื่องและนำมาใช้เป็นประโยชน์ในการส่งเมสเสจสั้นๆ ซึ่งไม่สามารถสนับสนุนส่วนหัว ดังนั้นจึงพิจารณาการเชื่อมต่อประเภทนี้เป็นแบบเชื่อถือไม่ได้ ซึ่งก็คือ การไม่รับประกันข้อมูลที่ถูกส่งออกไป ไม่เหมือนกับซ็อกเก็ตการเชื่อมต่อแบบต่อเนื่องที่ซ็อกเก็ตปลายทางถูกตรวจสอบเมื่อแพ็กแก็ตถูกส่งออกไป ซ็อกเก็ตแบบไม่ต่อเนื่อง เปรียบเสมือนกับการบริการของไปรษณีย์ที่ผู้ส่งจดหมายไปตามที่อยู่แล้วใส่ใน กล่องรับจดหมาย ผู้ส่งจะไม่ทราบว่าผู้รับได้รับจดหมายหรือไม่ ซ็อกเก็ตแบบนี้นิยมใช้กันในเซิร์ฟเวอร์ DNS (Domain Name System) ที่ใช้ซ็อกเก็ตดาต้าแกรมในการตอบสนองต่อการขอร้องที่เข้ามามากๆ

นอกจากนี้จะใช้ดาต้าแกรมซ็อกเก็ตในการกระจาย (Broadcast) เมสเสจ หรือ Multicast เพื่อไปยังปลายทางหลายๆ แห่งพร้อมกัน ซึ่งเหมือนกันการกระจายเสียงวิทยุ หรือ โทรทัศน์

2.5.2.3 Raw Socket

เป็นซ็อกเก็ตที่อนุญาตให้การเข้าถึงโปรโตคอล Transport Raw Socket และสามารถนำมาใช้เพื่อจัดการข้อมูลส่วนหัวไอพี (IP Header) โดยไม่ใช้ระบบของ Windows นอกจากนี้แล้วการใช้ซ็อกเก็ตชนิดนี้ ต้องการความรู้อย่างมากของโครงสร้างโปรโตคอลพื้นฐาน

ยังมีซ็อกเก็ตหนึ่งเป็น infrared Socket หรือ Irsock ซึ่งเป็นเทคโนโลยีตัวใหม่ที่แนะนำในวินโดว์ CE โดยที่ซ็อกเก็ต Infrared อนุญาตให้เครื่องคอมพิวเตอร์สองเครื่องติดต่อสื่อสารซึ่งกันและกัน

ตลอด โดยพอร์ตอนุกรม ซ็อกเก็ตชนิดนี้ใช้โปรโตคอลแบบต่อเนื่อง (Connection Oriented) ซึ่งเป็นโปรโตคอลที่เชื่อถือได้

2.5.3 Winsock

WinSock เป็น API เปิดด้านเน็ตเวิร์คที่เป็นมาตรฐาน โดยที่ WinSock ถูกออกแบบมาครั้งแรกเพื่อสร้างการโปรแกรมอินเตอร์เฟสที่เป็นมาตรฐานสำหรับ TCP/IP ในทุกเวอร์ชันของระบบปฏิบัติการ Windows รวมทั้ง Windows XP , Windows 2000 , Windows NT และ Windows 98 ซึ่งจะเป็น WinSock เวอร์ชัน 2.2 แต่ถ้าเป็นระบบปฏิบัติการวินโดวส์รุ่นดั้งเดิม เช่น Windows 95 และ Windows CE นั้นจะใช้ WinSock เวอร์ชัน 1.1

WinSock เป็นเน็ตเวิร์คแอพพลิเคชันโปรแกรมมิ่งอินเตอร์เฟสไม่ใช่โปรโตคอล ซึ่ง WinSock นั้นมารกฐานเดียวกับ Socket ของยูนิกซ์ตระกูล BSD (Berkeley Software Distribution) เวอร์ชัน 4.3 จากมหาวิทยาลัยแคลิฟอร์เนีย วิทยาเขต Berkeley รายละเอียด (Specification) ได้รวมทั้งรูทีนซ็อกเก็ตสไตล์ BSC และ การขยายสเปคมาใช้กับ Windows

การใช้ WinSock อนุญาตให้แอพพลิเคชันของเราทำการติดต่อสื่อสารข้ามเน็ตเวิร์คใดๆ ก็ได้ที่กระทำกับ WinSock API แพลตฟอร์ม Win32 ซึ่ง WinSock ได้ให้เธรดที่ปลอดภัย (thread safety) รวมทั้งมีอีกสองเหตุผลหลักในการใช้ WinSock คือการคอนโทรล และความมีประสิทธิภาพ

คลาส MFC สนับสนุนการโปรแกรม WinSock API โดยการใช้คลาส CAsyncSocket และ CSocket โดยที่คลาส CAsyncSocket ห้อมล้อมด้วยวินโดวส์ซ็อกเก็ต API ที่ระดับล่าง ซึ่งต้องมีความรู้เกี่ยวกับการสื่อสารข้อมูลผ่านเน็ตเวิร์ค แต่ถ้าเราต้องการอินเตอร์เฟสที่ง่ายกว่าคลาสนี้แล้วควรจะใช้คลาส CSocket

คลาส CSocket สืบทอดจากคลาส CAsyncSocket และรวมทั้งสืบทอดการห้อมล้อมของ Winsock API

ออบเจ็กต์ CSocket แสดงในระดับสูง (High level) กว่าออบเจ็กต์ CAsyncSocket ออบเจ็กต์ CSocket ทำงานร่วมกับคลาส CSocketFile และคลาส CArchive เพื่อจัดการการส่งและการรับข้อมูล

2.6 โปรโตคอล TCP/IP (RFC1180)

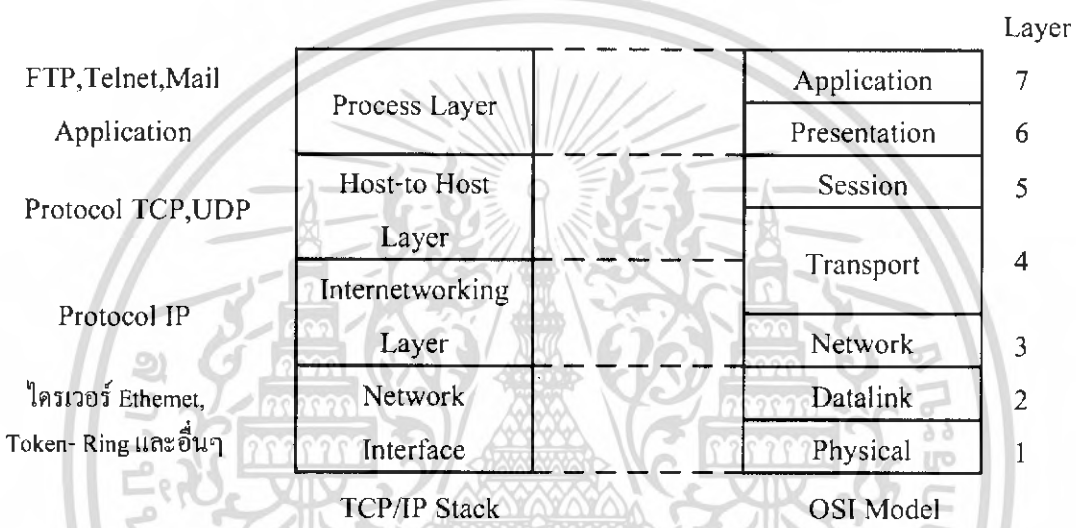
โปรโตคอล TCP/IP เป็นชื่อเรียกของชุดโปรโตคอลที่สำคัญ มีการใช้งานกันอย่างแพร่หลายตามการขยายของอินเตอร์เน็ต/อินทราเน็ต ความจริงแล้วโปรโตคอล TCP/IP เป็นกลุ่มของโปรโตคอลหลายตัวที่ประกอบกันเป็นชุดให้ใช้งาน โดยมีคำเต็มว่า Transmission Control Protocol/Internet Protocol ซึ่งจากชื่อเต็มทำให้เราเห็นว่า อย่างน้อยก็มีโปรโตคอลประกอบกันทำงานร่วมกัน 2 โปรโตคอล คือ TCP และ IP

2.6.1 โครงสร้างของโปรโตคอล TCP/IP

โปรโตคอล TCP/IP มีการจัดการแบ่งกลไกการทำงานออกเป็นชั้นๆ หรือ Layer เหมือนกับมาตรฐาน OSI Model และสามารถเทียบเคียงกับมาตรฐานของ OSI Model ได้ ซึ่งในแต่ละ Layer ของโปรโตคอล TCP/IP จะประกอบด้วย

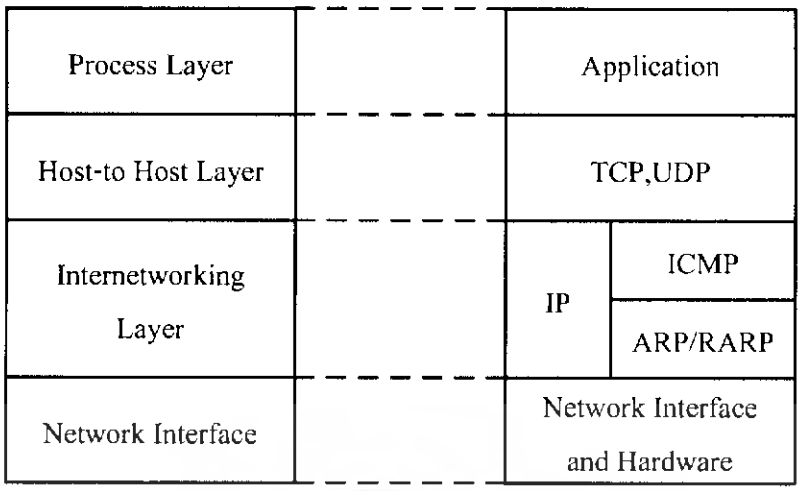
1. Process layer หรือ Application Layer
2. Host-to-Host layer หรือ Transport Layer
3. Inter network layer
4. Network Interface layer

โดยเมื่อเทียบกับมาตรฐาน OSI model แล้วจะเป็นดังรูปที่ 2.5 ซึ่งเราจะเห็นว่าบาง Layer ของโปรโตคอล TCP/IP เทียบได้กับมาตรฐาน OSI model ถึงสอง Layer และบาง Layer ก็จะทำงานคาบเกี่ยวกับหลายๆ Layer ของ OSI model ตัวอย่างเช่นในส่วนที่ Network Interface layer ของ โปรโตคอล TCP/IPจะเทียบได้กับการนำเอา Data link Layer และ Physical layer ของมาตรฐาน OSI model มารวมกัน เป็นต้น



รูปที่ 2.5 แสดงค่า TCP/IP Stack เปรียบเทียบกับมาตรฐาน OSI

ในแต่ละกลไกของโปรโตคอล TCP/IP โปรโตคอลอื่นๆในชุดของ TCP/IP ร่วมทำงานอยู่ด้วย ซึ่งทำให้เป็นที่มาของชื่อเรียก Protocol Stack เนื่องจากมีโปรโตคอลซ้อนทับกันอยู่ เพื่อช่วยกันทำงานดังรูปต่อไปนี้



รูปที่ 2.6 แสดงความสัมพันธ์ระหว่างโปรโตคอลต่างๆ ของ TCP/IP

จากรูปจะเห็นได้ว่า มี โปรโตคอล ในแต่ละระดับซ้อนทับกันอยู่หลายตัวด้วยกัน ซึ่งเราจะพูดกันในรายละเอียดต่อไป การซ้อนกันเป็นชั้นๆ หรือแต่ละ Layer นี้หากเป็น OSI model จะมีข้อบังคับให้แต่ละชั้นติดต่อได้กับเฉพาะชั้นที่ติดต่อกับตนเองเท่านั้นแต่สำหรับ TCP/IP Stack แล้วจะเห็นว่าบางชั้นสามารถละเลย หรือข้ามไปติดต่อกับชั้นที่ไม่ติดกับตนได้

2.6.2 การทำงานในแต่ละเลเยอร์ของโปรโตคอล TCP/IP

2.6.2.1 Process layer หรือ Application layer

ชั้นการทำงานของโปรโตคอล TCP/IP เทียบกับมาตรฐาน OSI model นั้น ในชั้นบนสุดเรียกว่า Process layer ทำงาน 2 หน้าที่เมื่อเทียบกับมาตรฐาน OSI model คือ Application layer และ Presentation layer ในชั้นนี้จะรองรับการทำงานของแอพพลิเคชั่นต่างๆ ที่ทำงานเป็นโปรเซสอยู่ในเครื่องเซิร์ฟเวอร์ที่ให้บริการ และเครื่องที่ขอใช้บริการหรือไคลแอนด์ (client) ซึ่งจะติดต่อผ่านโปรโตคอลเฉพาะแอพพลิเคชั่นอีกทีหนึ่ง และยังสามารถรองรับให้โปรโตคอลอื่นทำงานได้หลายโปรเซสและหลายโปรโตคอลพร้อมๆกัน

2.6.2.2 Host-to-Host Layer

การทำงานที่ชั้นของ Host-to-Host layer นี้จะมีบทบาทในการจัดการต่อจาก Process layer บางครั้งเราเรียกชั้น Host-to-Host ว่าเป็น Transport layer ซึ่งไม่ใช่ชั้นของ Transport layer ในมาตรฐาน OSI model การทำงานของ Host-to-Host layer นี้จะมีการสร้าง connection หรือการเชื่อมต่อกันระหว่างแอพพลิเคชั่นกับ Host-to-Host layer โดยจุดที่เชื่อมกันเพื่อรับส่งข้อมูลนี้เรียกว่า port หรือ socket และในแต่ละแอพพลิเคชั่นก็จะสร้างการเชื่อมต่อผ่าน port ได้พร้อมกันหลายแอพพลิเคชั่น ซึ่งการใช้งาน port ของแต่ละแอพพลิเคชั่นที่อยู่ในชั้น process layer จะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโปรโตคอลจะมีการใช้งาน port หมายเลขต่างๆไม่ซ้ำกัน

2.6.2.3 Internetwork Layer

ในระดับล่างต่อมาในชั้น Internetwork Layer มีหน้าที่ส่งผ่านข้อมูลในระดับเครือข่าย โดยมีโปรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใดๆ บนอินเตอร์เน็ต คือ โปรโตคอล IP (Internet Protocol) นอกจากนี้ในชั้น Internetwork Layer ยังมีโปรโตคอลทำงานอยู่ด้วยอีก 2 ชนิดคือ โปรโตคอล Internet Control Message Protocol (ICMP) และโปรโตคอล Address Resolution Protocol (ARP)

2.6.2.4 Network Interface Layer

การทำงานระดับล่างสุดต่อจาก Internetwork layer จะเป็นการแปลงข้อมูล IP datagram ให้อยู่ในรูปที่เหมาะสม และแปลงเป็นสัญญาณไฟฟ้าส่งไปยังเครือข่างต่อไป ซึ่งในชั้น Network Interface layer นี้ เมื่อเปรียบเทียบกับมาตรฐาน OSI Model แล้ว จะเป็นการรวม 2 layer เข้าด้วยกัน คือ data link layer และ Physical layer กล่าวโดยสรุปก็คือ การทำงานในชั้นต่างๆ ตามโครงสร้างของ โปรโตคอล TCP/IP

กล่าวโดยสรุปคือ โปรโตคอล TCP/IP ทำงานโดยแบ่งเป็นชั้นเทียบกับ OSI Model ได้กลไกในการทำงานของโปรโตคอล TCP/IP มี 4 ชั้น ซึ่งในชั้นแรกคือ Process Layer ทำหน้าที่ติดต่อกับแอพพลิเคชั่น และโปรโตคอล ที่แอพพลิเคชั่นนั้นๆ ใช้งาน

เราจะเห็นได้ว่าในแต่ละชั้นของโครงสร้าง TCP/IP Stack มีการใช้งานโปรโตคอลต่างๆ อยู่หนึ่งโปรโตคอลหรือมากกว่า ในแต่ละโปรโตคอลเหล่านี้ก็จะรับผิดชอบทำหน้าที่ของตน เพื่อส่งผ่านข้อมูลลงไปยังระดับล่าง และออกสู่เครือข่ายอินเตอร์เน็ตในที่สุด

2.6.3 โปรโตคอล TCP (Transmission Control Protocol)

การรับส่งข้อมูลของโปรโตคอล TCP จะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไปแต่จะแบ่งข้อมูลเป็นส่วนย่อยๆก่อนแล้วจึงส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในระหว่างการรับส่งข้อมูลนี้ จะมีขบวนการตรวจทานข้อมูลเพื่อให้ข้อมูลมีความถูกต้องไม่ผิดพลาดไปจากเดิม ในการติดต่อระหว่างกันเป็นแบบ Connection-Oriented แล้วจึงทำการรับส่งข้อมูลได้พร้อมกันผ่านทางพอร์ทต่างๆ

2.6.3 โปรโตคอล IP (Internet Protocol)

IP เป็นโปรโตคอลที่ทำหน้าที่รับภาระในการนำข้อมูลจาก Host-to-Host Layer ไปส่งยังจุดหมายปลายทางไม่ว่าที่ใดๆในอินเทอร์เนตได้อย่างถูกต้อง โปรโตคอลต่างๆใน TCP/IP ทั้ง TCP , UDP , ICMP ต่างก็ต้องอาศัยระบบนี้ทั้งหมด โปรโตคอล IP มีกลไกในการรับส่งข้อมูลทำโดยโปรโตคอล IPจะสร้าง IP datagram สำหรับการรับส่งขึ้นมาจากข้อมูลของ Host-to-Host Layer และพิจารณาว่าปลายทางของข้อมูลที่ต้องการส่งนี้อยู่ในเครือข่ายเดียวกันหรือเครือข่ายอื่นโดยการตรวจสอบค่า IP address ปลายทางในส่วนที่เป็นค่าหมายเลขของเครือข่าย (Network address)

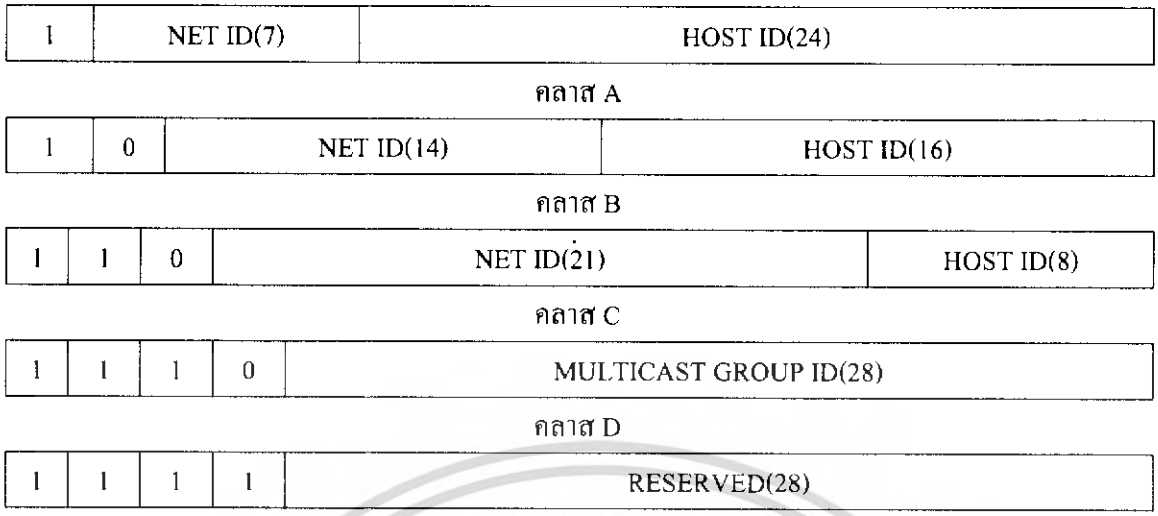
2.6.4 IP Addressing

IP Address ถูกกำหนดขึ้นมาให้เป็นหมายเลขอ้างอิงประจำตัวของอุปกรณ์ต่างๆที่เชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ต โดยการกำหนด IP Address ให้แต่ละเครื่องหรือแต่ละอุปกรณ์จะต้องไม่ซ้ำกัน ซึ่ง IP Addressนี้จะไม่ถูกผูกติดกับตัวฮาร์ดแวร์แต่อย่างใด จึงสามารถกำหนดใหม่หรือแก้ไข

เปลี่ยนแปลงได้เมื่อเปลี่ยนตัวฮาร์ดแวร์ ทั้งนี้เรื่องจากเป็นการกำหนดด้วยซอฟต์แวร์แตกต่างกับหมายเลข MAC address (Media Access Control Address)ซึ่งเป็นหมายเลขประจำตัวของอุปกรณ์ที่ต่อยู่ในเครือข่าย ค่าของ MAC address จะถูกกำหนดจากบริษัทผู้ผลิตอุปกรณ์ตั้งแต่เริ่มผลิตเช่น อุปกรณ์ Network Interface Card (NIC) จะมีค่า MAC address ประจำตัวที่ไม่ซ้ำกันและไม่สามารถแก้ไขได้ ค่า MAC address เป็นการระบุค่าอ้างอิงของอุปกรณ์ฮาร์ดแวร์ในระดับล่างสุด (Physical Layer) ของกลไกการรับส่งข้อมูลภายในเครื่อง ถ้าจะใช้หมายเลข MAC address สำหรับระบุอ้างอิงกันในเครือข่ายแล้วจะเกิดปัญหามาก เมื่อมีการเปลี่ยนหรือย้ายเครื่อง ต้องการทำการกำหนดระบบเครือข่ายใหม่ (configuration) นอกจากนี้ยังจดจำได้ยากกว่า การที่ IP Address ถูกใช้อ้างอิงในการติดต่อกันด้วยโปรโตคอล TCP/IP เพราะการใช้งาน IP Address จะยืดหยุ่นและคล่องตัวกว่า

ทุกอินเตอร์เฟสที่ต่ออยู่บนอินเทอร์เนตนั้นจะต้องมีหมายเลขประจำตัว เพื่อใช้ในการสื่อสารข้อมูลเรียกว่าอินเทอร์เนตแอดเดรส หรือเรียกย่อๆว่า IP Address โดยค่า IP Address นี้จะมีค่าเป็นหมายเลขจำนวน 32 บิต แต่แทนที่จะใช้เลขจำนวน 32 บิตนั้นนับเรียงกันต่อไปก็จะทำการแบ่งเลขนี้ออกเป็นกลุ่มๆ เป็นเลข 8 บิตจำนวน 4 ชุดและคั่นด้วยจุด นอกจากนี้ในส่วนของ IP Address จะแบ่งได้เป็น 2 ส่วน คือส่วนที่เป็น Network ID และส่วนของ Host ID ซึ่งส่วนนี้ของ IP นั้นจะถูกใช้สำหรับการค้นหาข้อมูลเส้นทางของ IP ในการขนส่งข้อมูลจากต้นทางไปยังปลายทาง โดยเพื่อความสะดวกในการใช้งานนั้น การจัด IP Address ได้ทำการแบ่งช่วงออกเป็นคลาสเพื่อทำการจัดสรร IP Address ได้อย่างเหมาะสมดังตารางที่ 2.9 และคุณสมบัติของแต่ละคลาสมีดังนี้

1. Class A เป็น IP Address ที่เริ่มต้นตั้งแต่ 0-127 จะกำหนดให้กับเครือข่ายที่มีขนาดใหญ่ เพราะ 1 เครือข่ายสามารถมีเครื่องลูกข่ายได้กว่า 18 ล้านเครื่อง IP Address จะเป็นลักษณะ net.host.host.host
2. Class B เป็น IP Address ที่เริ่มตั้งแต่ 128-191 จะกำหนดให้กับเครือข่ายที่มีขนาดใหญ่เช่นกันแต่เล็กกว่า Class A ในแต่ละเครือข่ายของ Class B สามารถมีเครื่องลูกข่ายได้ 2^{16} จำนวนแอดเดรสที่ใช้ควบคุมระบบเครือข่าย = 64,516 เครื่อง IP Address จะเป็นลักษณะ net.net.host.host
3. Class C เป็น IP Address ที่เริ่มตั้งแต่ 192-223 จะกำหนดให้กับเครือข่ายที่เป็นองค์กรทั่วไป ซึ่งส่วนใหญ่จะเป็นองค์กรขนาดกลางถึงเล็ก ในแต่ละกำหนดให้กับเครือข่ายมีเครื่องลูกข่ายไม่เกิน 2^8 จำนวนแอดเดรสที่ใช้ควบคุมระบบเครือข่ายเท่ากับ 254 เครื่อง IP Addressจะเป็นลักษณะ net.net.net.host
4. Class D เป็น IP Addressสำรองไว้สำหรับส่งข้อมูลแบบ Multicast ซึ่งจะไม่มีการแจกจ่ายให้ใช้งานทั่วไป
5. Class E เป็น IP Address พิเศษที่ใช้สำหรับงานทดสอบและพัฒนา ไม่มีการกำหนดให้ใช้งานทั่วไป



รูปที่ 2.7 การแบ่งคลาสของโปรโตคอล IP

คลาส	IP Range
A	1.0.0.0-121.255.255.255
B	128.0.0.0-191.255.255.255
C	192.0.0.0-233.255.255.255
D	224.0.0.0-239.255.255.255
E	240.0.0.0-255.255.255.255

ตารางที่ 2.9 แสดงช่วงของ IP Address แต่ละคลาส

2.7 ไมโครคอนโทรลเลอร์ P89C51RD2

2.7.1 การทำงานของ 8051

ไมโครคอนโทรลเลอร์ 8051 จะทำงานได้ก็ต้องประกอบด้วยส่วนสำคัญ 2 ส่วนคือ ส่วนของอุปกรณ์ (hardware) และอีกส่วนคือ ส่วนของโปรแกรม (software) โดยก่อนจะนำไอซี 8051 ไปใช้งานก็จะต้องมีการเขียนโปรแกรมคำสั่งลงไปในส่วนเก็บโปรแกรมที่เรียกสั้นๆว่า รอม โดยโปรแกรมที่เขียนขึ้นจะต้องนำไปทำการคอมไพล์ให้เป็นภาษาเครื่องเสียก่อน โดยโปรแกรมหลังจากคอมไพล์แล้วก็จะมีลักษณะของข้อมูลเป็นเลขฐาน 16 จึงจะนำโปรแกรมนั้นมาใส่ในหน่วยความจำ (ที่เรียกว่าการเบิร์นไอซีนั้นเอง) ของไอซีเพื่อนำไปใช้งานได้ เมื่อทำการป้อนไฟเข้ามาเลี้ยงวงจร ไอซี 8051 จะทำการเซทค่าก่อนเริ่มทำงาน ทุกครั้ง เพื่อให้เข้าใจถึงหลักการทำงานของ MCS-51 ได้ดียิ่งขึ้นควรพิจารณารูปในส่วนของโครงสร้างภายใน MCS-51 ประกอบไปด้วยหลังการการรีเซทค่าของวงจร การทำงานจะเริ่มจากในส่วนของโปรแกรมเคาท์เตอร์ (program counter) ซึ่งจะทำหน้าที่เป็นวงจรนับ และจะทำการส่งคือตำแหน่งของ

หน่วยความจำของโปรแกรมไปเก็บไว้ยังโปรแกรมเอดีดีอาร์รีจิสเตอร์ (program ADDR register) โดยส่งค่าผ่านบัสขนาด 16 บิต ในส่วนของโปรแกรมเอดีดีอาร์รีจิสเตอร์ ทำหน้าที่เป็นวงจรที่ทำการค้างค่าตำแหน่งของแอดเดรส ที่รับเข้ามาเอาไว้ซึ่งเป็นค่าตำแหน่งของหน่วยความจำที่รับมาจากโปรแกรมเคาท์เตอร์ มีขนาด 16 บิต ในสภาวะเริ่มทำงานเมื่อวงจรทำการรีเซท ค่าที่รับได้ที่ตำแหน่งนี้จะมีค่าเป็น 0000H หน่วยความจำโปรแกรมที่ได้จะเป็นค่าของหน่วยความจำรอมภายนอก หรือภายใน จะรู้ได้จากการตรวจสอบสภาวะลอจิกที่ขา เอเนเบิ้ลแอดเซส ที่ต่ออยู่กับวงจรส่วนของไทม์มิ่งแอนคอนโทรล (timing and control) โดยจะทำการสร้างสัญญาณและส่งค่าแอดเดรสไปยังรอมโดยผ่านทางบัส จากนั้นรอมจะส่งค่าของข้อมูลที่อยู่ใน แอดเดรส ที่รับค่ามาจากไทม์มิ่งแอนคอนโทรลส่งข้อมูลไปพักไว้ที่ อินสตรัคชั่นรีจิสเตอร์ (instruction register) โดยผ่านบัสข้อมูลภายใน (internal data bus) ในกรณีที่เป็นการติดต่อกับรอมภายในชิพ อินสตรัคชั่นรีจิสเตอร์ จะทำหน้าที่เป็นวงจรแลทช์ค่าของข้อมูลเอาไว้เพื่อส่งกลับไปยัง ไทม์มิ่งแอนคอนโทรลเพื่อทำการถอดรหัส และนำไปควบคุมการทำงานในส่วนอื่นๆต่อไป

ในอุปกรณ์ที่ติดต่อกับ รอม ภายนอกชิพ 8051 จะต้องกำหนดค่าลอจิก "1" ป้อนให้กับขา แอเนเบิ้ลแอดเซส วงจรไทม์มิ่งแอนคอนโทรล จะส่งค่าตำแหน่งแอดเดรสไปติดต่อกับรอมภายนอกจากนั้นรอมจะส่งข้อมูลที่อยู่ในตำแหน่งที่ถูกชี้โดยค่า แอดเดรส กลับมาพักไว้ใน อินสตรัคชั่นรีจิสเตอร์ และส่งกลับไปถอดรหัสที่วงจร ไทม์มิ่งแอนคอนโทรลเช่นเดียวกับการติดต่อกับรอม ภายในการทำงานในช่วงที่ ไทม์มิ่งแอนคอนโทรล ส่งค่า แอดเดรส ไปยังรอมเพื่ออ่านข้อมูลที่เป็นคำสั่งการทำงานกลับไปเก็บไว้ยังอินสตรัคชั่นรีจิสเตอร์ เรียกช่วงเวลาการทำงานนี้ว่า เฟทไซเคิ้ล (fetch cycle) หลังจากนั้นจะเป็นช่วงการนำข้อมูลที่รับเข้ามาไปถอดรหัส และส่งสัญญาณไปควบคุมการทำงานตามคำสั่งที่มีการโปรแกรมเอาไว้ และประมวลผลจนได้ผลลัพธ์ออกมา เรียกการทำงานในช่วงนี้ว่าเอ็กซ์ซีคิวไซเคิ้ล (execute cycle) ก็จะจบการทำงานใน 1 ไซเคิ้ล (cycle) การทำงานจะวนลูปการทำงานเช่นนี้ไปเรื่อยๆ

2.7.2 คุณสมบัติของไมโครคอนโทรลเลอร์ P89C51RD2

เป็นไมโครคอนโทรลเลอร์ตระกูล MCS-51 ซึ่งหน่วยความจำภายในเป็นแบบแฟลช (flash memory) ของ Philips Semicoconductor ในอนุกรม P89C51RD2 ซึ่งมีด้วยกัน 2 เบอร์คือ P89C51RD2-Hbp และ P89C51RD2BN สำหรับในหนังสือเล่มนี้จะอ้างถึงเบอร์ P89C51RD2BN มีคุณสมบัติทางเทคนิคที่โดดเด่นดังนี้

1. เป็นไมโครคอนโทรลเลอร์ 8 บิต ที่เข้ากันได้กับไมโครคอนโทรลเลอร์ MCS-51 พื้นฐาน
2. หน่วยความจำโปรแกรมภายในตัวไมโครคอนโทรลเลอร์เป็นแบบแฟลช ทำให้สามารถลบและเขียนใหม่ได้ถึงหนึ่งหมื่นครั้ง จึงสามารถใช้งานในรูปแบบไมโครคอนโทรลเลอร์ชิปเดี่ยวไม่ต้องใช้หน่วยความจำภายนอก ส่งผลให้สามารถใช้งานพอร์ตอินพุตของไมโครคอนโทรลเลอร์ได้อย่างเต็มประสิทธิภาพ ขนาดของหน่วยความจำโปรแกรมสูงถึง 64 กิโลไบต์
3. หน่วยความจำข้อมูลแรมภายในมีขนาด 1 กิโลไบต์

4. สามารถโปรแกรมข้อมูลขนาดลงในหน่วยความจำโปรแกรมแบบในวงจร หรือในระบบ (ISP : In-system programming)
5. ความถี่สัญญาณนาฬิกาสูงสุด 33 MHz ในกรณีทำงานด้วยสัญญาณนาฬิกาภายใน 12 ลูกต่อ แมชชีนไซเกิลและ 20 MHz ในกรณีทำงานด้วยสัญญาณนาฬิกาภายใน 6 ลูกต่อแมชชีนไซเกิล P89C51RD2BN ได้รับการกำหนดให้ทำงานเบื้องต้นในโหมดสัญญาณนาฬิกา 12 ลูก ต่อแมชชีนไซเกิล สามารถเลือกเปลี่ยนเป็น 6 สัญญาณนาฬิกาต่อแมชชีนไซเกิลได้ โดยที่สามารถเปลี่ยนกลับไปกลับมาได้
6. ขาพอร์ต 8 บิต 4 พอร์ต แบบกึ่งทิศทาง (quisi-bifirectional) เป็นได้ทั้งอินพุตและเอาต์พุต
7. อุปกรณ์เพอริเฟลรัลภายในไมโครคอนโทรลเลอร์สามารถทำงานด้วยความเร็ว 12 สัญญาณนาฬิกาต่อแมชชีนไซเกิลได้แม้ว่าซีพียูจะทำงานด้วยความเร็ว 6 สัญญาณนาฬิกาภายในต่อแมชชีนไซเกิลเป็นคุณสมบัติที่เพิ่มเติมเข้ามาในเบอร์ P89C51RD2BN
8. มีวงจรสื่อสารอนุกรมแบบฟูลดูเพล็กซ์
9. ไทเมอร์เคาน์ชขนาด 16 บิต 3 ตัว (ไทเมอร์ 0,1และ 2)
10. มีรีจิสเตอร์ตัวชี้ตำแหน่งข้อมูลหรือ DPTR 2 ตัว
11. สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ได้ 7 ประเภท
12. กำหนดนับสำคัญของการตอบสนองอินเตอร์รัปต์ได้ 4 ระดับ
13. สามารถติดต่อหน่วยความจำภายนอกได้สูงสุด 64 กิโลไบต์
14. มีวอตซ์ดือกไทเมอร์
15. มีโมดูลวงจรนับโปรแกรมได้ (PCA-Programmable Counter Array) ซึ่งบรรจุวงจรตรวจจับสัญญาณ (capture) เปรียบเทียบสัญญาณ (Compare) วงจรมอดูเลชั่นทางความกว้างพัลส์ (PWM) และวอตช์ไทเมอร์ (watchdog timer)

2.8 การสื่อสารข้อมูลผ่านพอร์ตอนุกรม ของไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษา C

ในบทนี้นำเสนอการเขียนโปรแกรมภาษา C เพื่อให้ไมโครคอนโทรลเลอร์ MCS-51 ติดต่อกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม ซึ่งในไมโครคอนโทรเลอร์ MCS-51 สามารถสื่อสารข้อมูลได้ทั้งรับและส่งในเวลาเดียวกัน หรือที่เรียกว่าเป็นแบบฟูลดูเพล็กซ์ (Full Duplex) โดยใช้ขา P3.0 ทำหน้าที่รับข้อมูล (RxD) และขา P3.1 ทำหน้าที่ส่งข้อมูล (TxD) รูปแบบการสื่อสารข้อมูลจะต้องมีรูปแบบเดียวกันทั้งคู่ อันประกอบด้วยอัตราในการถ่ายทอดข้อมูลหรืออัตราบอด ท(baudrate) บิตเริ่มต้น (Start bit) บิดข้อมูล (data bit) , บิตตรวจสอบพาริตี้ (Parity bot) และบิตหยุด (Stop bit)

2.8.1 รีจิสเตอร์ที่เกี่ยวข้องในการสื่อสารข้อมูลอนุกรม

2.8.1.1 รีจิสเตอร์ควบคุมการทำงานของพอร์ตอนุกรมหรือ SCON (Serial port Control Register)

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

รูปที่ 2.8 แสดงบิตการเชื่อมต่อของไมโครคอนโทรลเลอร์

1. SM0/FE (Serial port mode bit 0/Framing error bit) :

ปกติจะใช้ร่วมกับบิต SM1 เพื่อกำหนดโหมดการทำงานของพอร์ตอนุกรม การเข้าถึงบิตนี้จะเกิดขึ้นได้ก็ต่อเมื่อมีการเคลียร์บิต SMOD ซึ่งคือ บิต 6 ของรีจิสเตอร์ PCON ในกรณีที่ใช้ความสามารถตรวจจับความผิดพลาดของเฟรมข้อมูล บิตนี้จะใช้แจ้งความผิดที่เกิดขึ้น โดยจะเซตเป็น "1" ทันทีเมื่อพบว่า ไม่สามารถตรวจจับบิตหยุดหรือบิตปิดท้ายของข้อมูลของพอร์ตอนุกรมได้ การเอ็นแอเบิลความสามารถนี้ทำได้โดยการเซตบิต SMOD ในรีจิสเตอร์ PCON การเคลียร์บิตนี้ต้องกระทำทางซอฟต์แวร์เท่านั้น

2. SM1 (Serial port mode bit 1)

ใช้ร่วมกับบิต SMO ในการกำหนดโหมดการทำงานของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ P89C51RD2 ดังรายละเอียดต่อไปนี้

SM0	SM1	โหมด	รายละเอียด	อัตราบอด
0	0	0	ชิฟต์รีจิสเตอร์	ความถี่สัญญาณนาฬิกา /6 (ในโหมด 6 ไซเกิลสัญญาณนาฬิกา)
0	1	1	UART 8 บิต	ปรับค่าได้
0	0	2	UART 9 บิต	ความถี่สัญญาณนาฬิกา/32 (ในโหมด 6 ไซเกิลสัญญาณนาฬิกา)
1	1	3	UART 9 บิต	ปรับค่าได้

ตาราง 2.10 แสดงโหมดการเชื่อมต่อของไมโครคอนโทรลเลอร์

3. SM2 (Serial port mode bit 2) :

ใช้ในการเอ็นเคเบิลความสามารถการรับรู้แอดเดรสในการติดต่ออัตโนมัติ เมื่อมีการเชื่อมต่อไมโครคอนโทรลเลอร์หลายตัวด้วยกัน โดยความสามารถนี้จะใช้ได้ก็ต่อเมื่อวงจรพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ทำงานในโหมด 2 หรือ 3 ถ้าบิต SM2 เป็น "1" บิต R1 จะไม่เซต เว้นแต่ข้อมูลบิตที่ 9 ที่รับเข้ามาเป็น "1" เป็นแจ้งว่า สามารถติดต่อได้และข้อมูลที่รับเข้ามาคือ ค่า แอดเดรสที่ต้องการติดต่อด้วย

ในกรณีที่พอร์ตอนุกรมทำงานอยู่ในโหมด 1 ถ้าบิต SM2 เซต บิต R1 จะไม่เปลี่ยนแปลงจนกว่าจะได้รับข้อมูลบิตหยุดหรือบิตปิดท้าย และข้อมูลที่ได้รับจะเป็นข้อมูลแอดเดรสที่ต้องการติดต่อด้วย

ในกรณีที่วงจรพอร์ตอนุกรมทำงานในโหมด 0 บิต SM2 นี้เป็น "0"

4. REN (Receiverd enable bit) :

ใช้เอ็นเอเบิลความสามารถในการรับข้อมูลของวงจรพอร์ตอนุกรม "1" เอ็นเอเบิลการรับข้อมูล "0" ดิสเอเบิลการรับข้อมูล การเซตหรือเคลียร์บิตนี้ต้องกระทำด้วยกระบวนการทาง ซอฟแวร์

5. TB8 (Transmit data bit 8) :

ใช้เก็บข้อมูลบิต 8 หรือบิตที่ 9 ที่ต้องการออกทางพอร์ตอนุกรม เมื่อทำงานในโหมด 2และ 3

6. RB8 (Receive data bit 8) :

ใช้เก็บข้อมูลบิต 8 ของข้อมูลที่รับเข้ามาของพอร์ตอนุกรมเมื่อทำงานในโหมด 2 และ 3 ในโหมด 1 ถ้าบิต SM2 = 0 ข้อมูลของบิตหยุดจะเก็บไว้ที่บิตนี้ ไม่ใช้งานบิตนี้ในโหมด 0

7. TI (Transmit interrupt flag) :

บิตแสดงการเกิดอินเตอร์รรัปต์จากการส่งข้อมูลออกทางพอร์ตอนุกรมเมื่อทำงานในโหมด 0 บิตนี้จะเซตเมื่อมีการส่งข้อมูลบิต 7 หรือ บิต 8 ออกไป แต่ถ้าทำงานในโหมดอื่น บิตนี้จะเซตเมื่อมีการเริ่มต้นส่งบิตหยุดหรือบิตปิดท้าย การเคลียร์บิตนี้ต้องกระทำด้วยกระบวนการทางซอฟต์แวร์

8. RI (Receive interrupt flag) :

บิตแสดงการเกิดอินเตอร์รัปต์เนื่องจากการรับข้อมูลเข้ามาของพอร์ตอนุกรม เมื่อทำงานในโหมด 0 บิตนี้จะเซตเมื่อรับข้อมูลบิต 7 หรือบิตที่ 8 เสร็จสมบูรณ์ แต่ถ้าทำงานในโหมดอื่น บิตนี้จะเซตเมื่อการบิตปิดท้ายดำเนินไปได้ครึ่งทาง นอกจากนี้การเซตบิตนี้ยังมีเงื่อนไขที่กำหนดโดยบิต SM2ร่วมด้วย การเคลียร์บิตนี้ต้องกระทำด้วยกระบวนการทางซอฟต์แวร์

โดยส่วนใหญ่แล้วมักจะตั้งค่าของ SCON ดังนี้

SCON = 0x40 ; → พอร์ตอนุกรมทำงานในโหมด 1 สามารถส่งข้อมูลได้อย่างเดียว

SCON = 0x50 ; → พอร์ตอนุกรมทำงานในโหมด 1 สามารถส่งข้อมูลและรับได้

2.8.2 การเขียนโปรแกรมภาษา C ติดต่อกับพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MSC-51

2.8.2.1 การเขียนโปรแกรมแบบตรวจสอบบิตแฟลก

เป็นวิธีที่ให้โปรแกรมตรวจสอบบิตแฟลกที่เกี่ยวข้องกับการรับส่งข้อมูล ซึ่งได้แก่

1. บิต RI เป็นบิตที่ใช้ตรวจสอบการรับข้อมูลเข้ามาทางพอร์ต ถ้า RI = "1" หมายความว่ามีข้อมูลเข้ามาทางพอร์ตอนุกรมแล้ว และข้อมูลถูกเก็บอยู่ที่รีจิสเตอร์ SBUF ถ้าต้องการนำข้อมูลไปใช้งานต้องเคลียร์บิตแฟลก RI โดยเขียนคำสั่ง ; แล้วอ่านข้อมูลจาก SBUF มาเก็บไว้ที่ตัวแปรที่ต้องการ

2. บิต TI เป็นบิตที่ใช้ตรวจสอบการรับข้อมูลเข้ามาทางพอร์ต ถ้า TI = "1" หมายความว่าส่งข้อมูล1 ไบต์ออกทางพอร์ตอนุกรมเรียบร้อยแล้ว หลังจากนั้นทำการเคลียร์บิตแฟลก TI ถ้าด้วยคำสั่ง TI = 0 ; เพื่อเตรียมความพร้อมในการส่งข้อมูลไบต์ต่อไป โดยในการส่งข้อมูลออกนั้นจะต้องเขียนข้อมูลให้กับรีจิสเตอร์ SBUF

การเขียนโปรแกรมวิธีนี้มีข้อดีที่ไม่ซับซ้อนและเหมาะกับขนาดของข้อมูลจำนวนไม่มาก

2.8.2.2 เขียนโปรแกรมโดยใช้อินเตอร์รัปต์

เนื่องจากในการทำงานของพอร์ตอนุกรมสามารถสร้างสัญญาณอินเตอร์รัปต์ได้ โดยมีตำแหน่งอินเตอร์รัปเวกเตอร์อยู่ในระดับ 4 (interrupt 4) แต่จะมีข้อยุ่งยากเล็กน้อยตรงที่ การเกิดอินเตอร์รัปต์ของพอร์ตอนุกรมอาจจะเกิดจากการเซตบิตแฟลก RI เมื่อมีการรับข้อมูล หรือ จากบิต TI เมื่อส่งข้อมูล 1 ไบต์เสร็จสิ้นก็ได้ ซึ่งบิตแฟลกทั้ง 2 ตัวนี้จะทำหน้าที่เป็นแฟลกของการอินเตอร์รัปต์ในลำดับ 4 ทันที ในส่วนของโปรแกรมบริการอินเตอร์รัปต์นั้นจะต้องมีการตรวจสอบบอตแฟลกทั้ง 2 ตัวเลยว่า อินเตอร์รัปต์เกิดขึ้นจากการเซตของบิตแฟลกตัวใด จากนั้นทำการเคลียร์แฟลกตัวนั้นด้วย เพื่อให้เตรียมพร้อมรับหรือส่งข้อมูลในครั้งต่อไป

2.9 ระบบบัส I^2C

I^2C ย่อมาจาก Inter-IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซี โดยบัส I^2C ได้รับการพัฒนาขึ้นโดย ฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือ โมดูลสามารถติดต่อ สั่งงาน และควบคุมภายในสายสัญญาณเพียงสองเส้น เส้นหนึ่งคือสายข้อมูล อีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I^2C เรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock line) ในการอธิบายต่อไปนี้จะเรียกสายสัญญาณทั้งสองว่า สาย SDA และ SCL

2.9.1 คุณสมบัติโดยทั่วไปของบัส I^2C

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-direction line) ต้องมีการต่อตัวต้านทานพูลอัพกับแรงดัน 5 โวลต์ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาท์พุต ของอุปกรณ์ที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (open-drain) หรือคอลเล็กเตอร์เปิด (open-collector)

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติและสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูงอุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่

เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่าคือ 7 บิต (7-bit addressing) หรือ 10 บิต (10-bit addressing)

2.9.2 หลักการของบัส I^2C

บัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้นคือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส เพื่อให้ผู้ใช้งานทราบว่าขานี้ อุปกรณ์ใดติดต่อกันอยู่ และอุปกรณ์ตัวใดเป็นตัวรับหรือตัวส่ง ต่อไปนี้จะขออธิบายลักษณะ หน้าที่และนิยามของอุปกรณ์ที่ต่อยู่บนบัส I^2C เพื่อเป็นข้อตกลงก่อนอธิบายการทำงานของบัส I^2C ต่อไป

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง (transmitter)

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

อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบันบัส I^2C เรียกว่า มาสเตอร์ (master)

อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ

1. การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น
2. ในระหว่างกาถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะเป็นลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลงความหมายเป็นสัญญาณควบคุมแทน

2.9.3 การเขียนโปรแกรมภาษา C เพื่อติดต่อกับอุปกรณ์บนระบบบัส I^2C

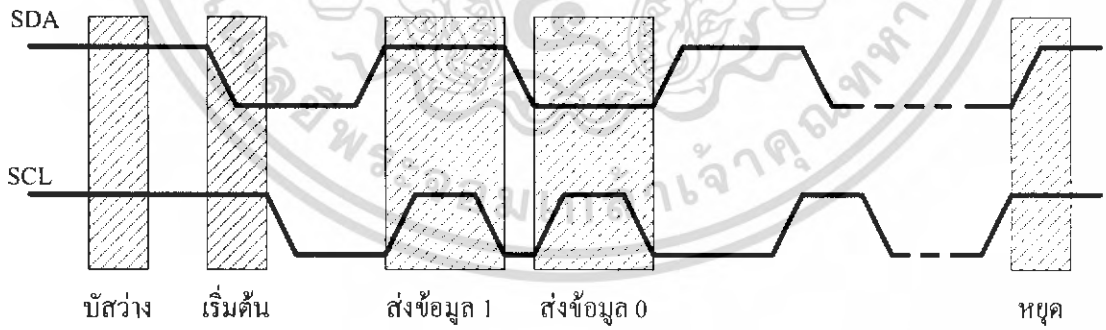
อุปกรณ์ระบบบัส I^2C ตัวแรกที่ขอแนะนำในการเขียนโปรแกรมภาษา C เพื่อเชื่อมต่อไมโครคอนโทรลเลอร์ MCS-51 กับอุปกรณ์ระบบบัส I^2C คือ PCF8574A ซึ่งเป็นไอซีขยายพอร์ตอินพุตเอาต์พุตขนาด 8 บิต โดยกำหนดให้ไมโครคอนโทรลเลอร์ MCS-51 เป็นอุปกรณ์มาสเตอร์ และ PCF8574A เป็นอุกปรณ์สเลฟ โดยเริ่มต้นด้วยการเขียนโปรแกรมสร้างสภาวะของบัส I^2C และโปรแกรมติดต่อกับอุปกรณ์ จากนั้นนำข้อมลทั้งหมดมาสร้างเป็นไลบรารีของฟังก์ชั่นระบบบัส I^2C ซึ่งสามารถนำไปใช้กับอุปกรณ์ระบบบัส I^2C ตัวอื่นๆ ในการทดลองถัดไป

2.9.3.1 ฟังก์ชั่นโปรแกรมภาษา C ของการติดต่อระบบบัส I^2C

เพื่อช่วยให้การเขียนโปรแกรมภาษา C สำหรับการติดต่อกับอุปกรณ์ระบบบัส I^2C ได้อย่างสะดวกขึ้น จึงมีการสร้างฟังก์ชั่นสำหรับติดต่อกับอุปกรณ์ระบบบัส I^2C ไว้ใช้งานซึ่งมีรายละเอียดในข้อมูลที่สำคัญของบัส I^2C สภาวะที่เกิดขึ้นบนบัส I^2C มีด้วยกัน 5 สภาวะ ดังนี้

1. บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทังคู่ นั่นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได

2. เริ่มต้นการถ่ายทอดข้อมูล (start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะเริ่มต้น (start)
3. ข้อมูลดำรงอยู่บนบัส (data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่าเป็น "0" หรือ "1" ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูงหากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น
4. รับรู้ข้อมูล (acknowledge) เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่า บิตรับรู้ มีสถานะเป็นลอจิกสูง หลังจากส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกา อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้ที่มีสถานะลอจิกต่ำเพื่อตอบสนองให้ทราบว่าได้รับข้อมูลเรียบร้อยแล้ว
5. หยุดการถ่ายทอดข้อมูล (stop data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูงในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะหยุด (stop)



รูปที่ 2.9 แสดงเวลาในสถานะต่างๆของบัส I^2C

2.9.4 การทำงานบนบัส I^2C

เริ่มต้นด้วยการเข้าถึงอุปกรณ์เสียก่อน โดยการเข้าถึงอุปกรณ์บนบิต I^2C นั้นจะใช้การเข้าถึงแบบ7 หรือ 10 บิต ในกรณีที่มีอุปกรณ์ต่ออยู่บนบัสไม่มาก ใช้การเข้าถึงแบบ 7 บิต ก็เพียงพอ แต่ในบาง

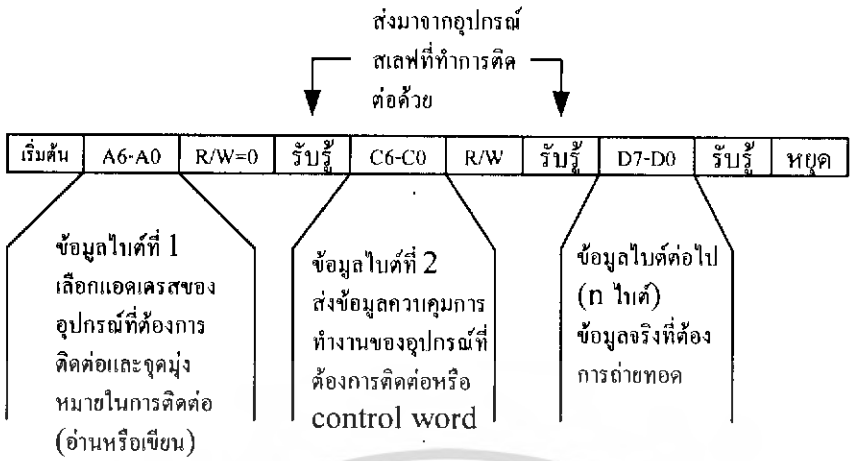
อุปกรณ์ต้องใช้การเข้าถึงแบบ 10 บิต หนังจากที่ติดต่ออุปกรณ์แต่ละตัวได้เรียบร้อยแล้ว ก็จะเริ่มต้อนการถ่ายทอดข้อมูลกันต่อไป

2.9.4.1 การเข้าถึงแบบ 7 บิต (7-bit addressing)

1. ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้นคือ ข้อมูลที่ใช้ในการอ้างถึงอุปกรณ์ที่ต้องการติดต่อ โดยมีรูปแบบแสดงในรูปที่ 2.9 ใน 7 บิตรวมทั้งบิต MSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็น บิตกำหนดแอดเดรสคงที่ (fixed address bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิต ไม่สามารถเปลี่ยนแปลงแก้ได้ ถัดมากอีก 3 บิตเป็นบิตกำหนดแอดเดรสที่สามารถโปรแกรมได้ (programmable address bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แต่ขา A0-A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I^2C ส่วนในบิต LSB เป็นบิตที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวนั้นๆ หากบิต LSB เป็น "0" หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ
2. ข้อมูลในไบต์ต่อมาคือ ข้อมูลควบคุม (control byte) ในอุปกรณ์แต่ละตัวมีการกำหนดข้อมูลควบคุมที่แตกต่างกันไป ยกตัวอย่าง ไอซีขยายพอร์ตมีข้อมูลควบคุมที่ใช้กำหนดว่าบิตใดเป็นอินพุต บิตใดเป็นเอาต์พุต ในขณะที่ไอซี ADC/DAC ต้องการข้อมูลควบคุมเพื่อกำหนดให้ทำงานเป็นวงจร ADC หรือ DAC เป็นต้น
3. ข้อมูลในไบต์ต่อมาคือ ข้อมูลที่ทำการถ่ายทอดจริง (data)

หลังจากถ่ายทอดข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง

รูปที่ 2.10 รูปแบบข้อมูลในการอ้างแอดเดรส

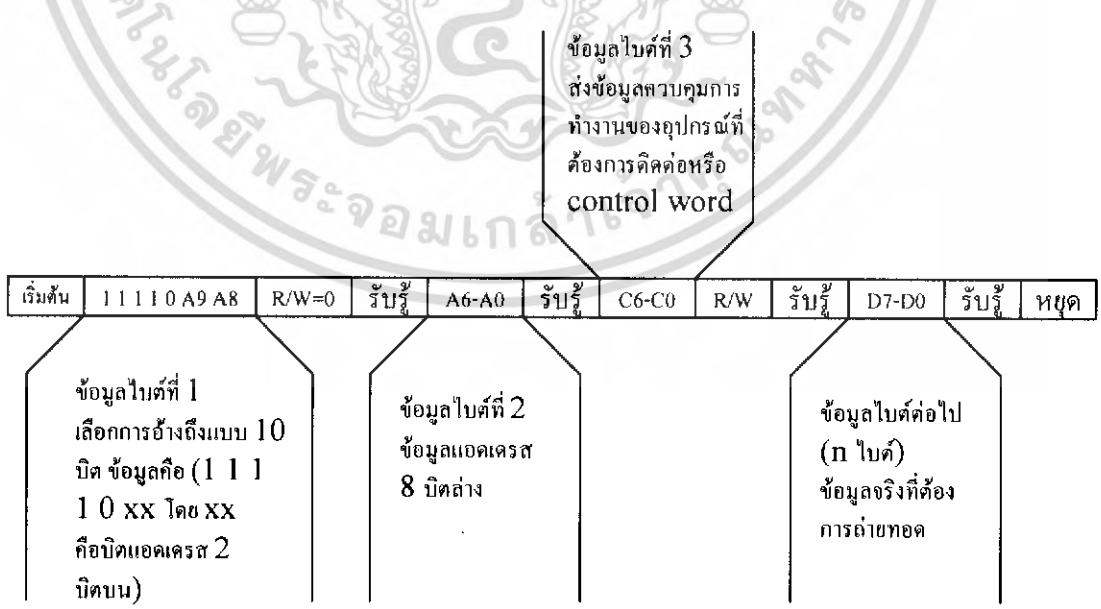


รูปที่ 2.11 รูปแบบข้อมูลในการเข้าถึงแบบ 7 บิต

2.9.4.2 การเข้าถึงแบบ 10 บิต

จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อยโดยในไบต์แรกหลังจากเกิดสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของอุปกรณ์ที่ต้องการติดต่อในบิต LSB ของข้อมูลไบต์แรกยังคงเป็นการกำหนดว่า ต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบต์ต่อมาเป็นข้อมูลแอดเดรสในไบต์ที่ 2 ของอุปกรณ์ที่ต้องการติดต่อด้วย ข้อมูลไบต์ถัดไปจึงเป็นข้อมูลควบคุมข้อมูลหลังจากนั้นก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อ

เช่นเดียวกับการเข้าถึงแบบ 7 บิต หลังจาถ่ายทอดข้อมูลครบทุกไบต์ ต้องมีสภาวะรับรู้เกิดขึ้นเพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้



รูปที่ 2.12 รูปแบบข้อมูลในการเข้าถึงแบบ 10 บิต

2.10 ระบบบัส 1 สาย ของไมโครคอนโทรลเลอร์ MCS-51

ระบบบัส 1 สาย(1-wire bus) เป็นระบบสื่อสารข้อมูลอนุกรมที่มีชาญฉลาดและใช่จำนวนสายเพียง 1 เส้นเท่านั้น โดยไม่ต้องมีสายสัญญาณนาฬิกามาควบคุมจังหวะการถ่ายทอดข้อมูลเหมือนกับระบบสื่อสารข้อมูลอนุกรมในแบบอื่นๆเนื่องจากสายข้อมูลนั้นจะทำหน้าที่เสมือนเป็นสายสัญญาณนาฬิกาในตัว ส่วนค่าของข้อมูลจะพิจารณาจากลักษณะของรูปสัญญาณที่ปรากฏบนสายสัญญาณในแต่ละช่องของเวลาหรือต่อไปนี้จะขอเรียกว่า ไทม์สล็อต (time-slot) โดยคาบเวลาต่ำสุดและสูงสุดของสถานะต่างๆ ในการสื่อสารข้อมูลในแต่ละไทม์สล๊อต มีการกำหนดขอบเขตไว้อย่างชัดเจน การถ่ายทอดข้อมูลจะเกิดขึ้นในแต่ละไทม์สล็อตนั้น รูปแบบการถ่ายทอดข้อมูลเป็นแบบอะซิงโครนัสในระดับบิต ไม่มีการกำหนดความยาวของข้อมูลเป็นระดับไบต์

การสื่อสารข้อมูลแบบนี้ผู้ค้นคิดคือ ดัลลัสเซมิคอนดักเตอร์ ดังนั้นในบางครั้งจังเรียกระบบสื่อสารข้อมูลแบบนี้ว่า ระบบสื่อสารข้อมูลดัลลัสเซมิคอนดักเตอร์ ดังนั้นในบางครั้งจึงเรียกระบบสื่อสารข้อมูลแบบนี้ว่า ระบบสื่อสารข้อมูลดัลลัสหนึ่งสาย (The Dallas 1 Wire Bus)

2.10.1 คุณสมบัติของไทม์สล็อต

อุปกรณ์มาสเตอร์จะเป็นอุปกรณ์เพียงตัวเดียวบนระบบบัสที่สามารถทำการอินิเชียลสายสัญญาณได้ โดยอุปกรณ์มาสเตอร์จะกำเนิดจุดเริ่มต้นของไทม์สล็อตด้วยการทำให้สายสัญญาณเป็นลอจิกต่ำในช่วงเวลาหนึ่ง จากนั้นก็จะทำให้กลับมาเป็นลอจิกสูง ถ้าหากอุปกรณ์สเลฟต้องการส่งข้อมูลมายังอุปกรณ์มาสเตอร์ อุปกรณ์สเลฟจะเป็นตัวควบคุมสภาวะของสายสัญญาณต่อไป จนเสร็จสิ้นกะรบวนการ แต่ถ้าหากอุปกรณ์มาสเตอร์ต้องการส่งข้อมูลก็จะสามารถดำเนินการต่อไปได้เลย

2.10.1.1 ฟังก์ชั่นของไทม์สล็อตที่กำหนดโดยอุปกรณ์มาสเตอร์

มีด้วยกัน 4 ฟังก์ชั่นคือ

1. รีเซต (reset) ใช้ในการเริ่มต้นติดต่อกับอุปกรณ์สเลฟ
2. อ่านข้อมูล (read data) ใช้อ่านข้อมูลที่ส่งมาจากอุปกรณ์สเลฟ
3. เขียนข้อมูล "1" (write one) ใช้สำหรับเขียนข้อมูล "1" ไปยังอุปกรณ์สเลฟผ่านสายสัญญาณของระบบ
4. เขียนข้อมูล "0" (write zero) ใช้สำหรับเขียนข้อมูล "0" ไปยังอุปกรณ์สเลฟผ่านสายสัญญาณของระบบ

2.10.1.2 ฟังก์ชั่นของไทม์สล็อตในอุปกรณ์สเลฟ

มี 3 ฟังก์ชั่นคือ

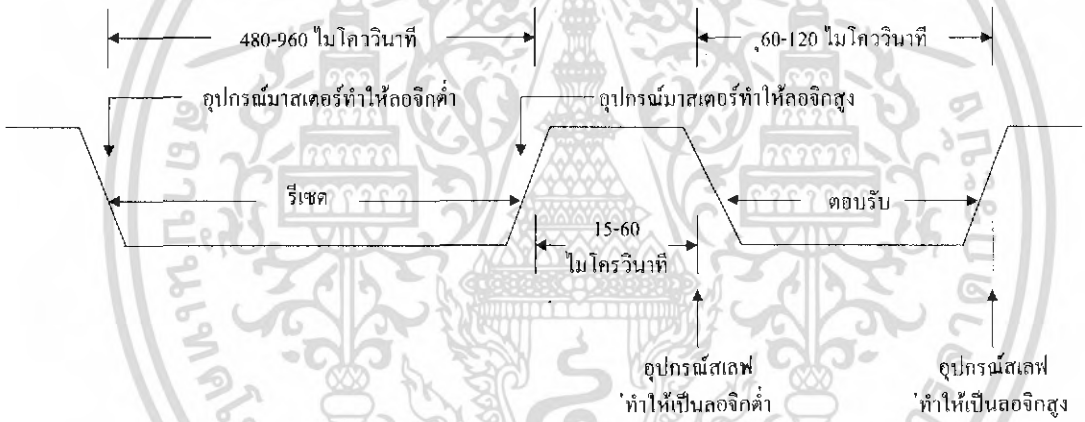
1. ตอบสนอง (presence) ใช้สำหรับตอบสนองการติดต่อจากอุปกรณ์มาสเตอร์โดยอุปกรณ์สเลฟตัวที่ถูกเลือกจากอุปกรณ์มาสเตอร์จะต้องส่งสัญญาณตอบสนองบนสายสัญญาณเพื่อแจ้งให้อุปกรณ์มาสเตอร์ทราบว่า ขณะนี้สามารถติดต่อกันได้แล้ว
2. เขียนข้อมูล "1" (write one) ใช้สำหรับส่งข้อมูล "1" ไปยังอุปกรณ์มาสเตอร์ผ่านสายสัญญาณของระบบ ซึ่งจะสัมพันธ์กับไทม์สล็อตการอ่านข้อมูลของอุปกรณ์มาสเตอร์

3. เขียนข้อมล "0" (write zero) ใช้สำหรับส่งข้อมูล "0" ไปยังอุปกรณ์มาสเตอร์ผ่านสายสัญญาณของระบบ ซึ่งจะสัมพันธ์กับไทม์สล็อตการอ่านข้อมูลของอุปกรณ์มาสเตอร์

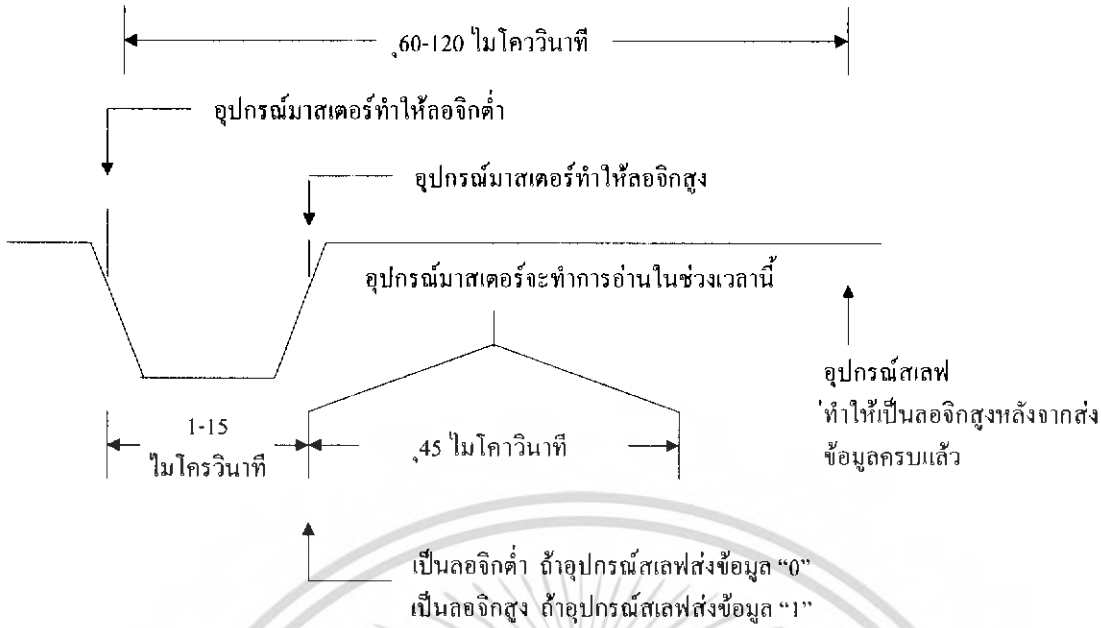
การแยกแยะฟังก์ชั่นของแต่ละไทม์สล็อตจะใช้ความยาวของคาบเวลาและลักษณะของรูปสัญญาณเป็นตัวกำหนด และทุกครั้งที่มีการเปลี่ยนแปลงฟังก์ชั่นต้องทำให้สายสัญญาณอยู่ในสภาวะว่างเสมอ ซึ่งก็คือ การทำให้สายสัญญาณเป็นลอจิกสูงอย่างน้อยเป็นเวลา 1 ไมโครวินาที

2.10.1.3 ไทม์สล็อตการรีเซตและตอบสนอง

อุปกรณ์มาสเตอร์ซึ่งในที่นี้คือ ไมโครคอนโทรลเลอร์จะทำให้เกิดการรีเซตบนสายสัญญาณเพื่อแจ้งแก่อุปกรณ์สเลฟ ด้วยการทำให้สายสัญญาณเป็นลอจิก "0" ต่อเนื่องอย่างน้อย 480 ไมโครวินาที และจะต้องทำให้สายสัญญาณกลับมาเป็นลอจิก "1" ภายใน 480 ไมโครวินาทีหลังจากนั้น ถ้าหากมีอุปกรณ์สเลฟต่ออยู่บนสายสัญญาณ จะมีการตอบสนองสัญญาณรีเซตนั้นด้วยสัญญาณตอบสนอง (presence) ด้วยการทำให้สายสัญญาณเป็นลอจิก "0" ต่อเนื่องนานประมาณ 60-240 ไมโครวินาที หลังจากที่สัญญาณรีเซตปรากฏขึ้นประมาณ 15-60 ไมโครวินาที ในรูปที่ 2.12 แสดงไทม์สล็อตของการรีเซตและการตอบสนอง



รูปที่ 2.13 ไทม์สล็อตการรีเซตและการตอบรับของอุปกรณ์บนระบบบัสหนึ่งสาย



รูปที่ 2.14 ไทม์สล็อตการอ่านข้อมูลของไมโครคอนโทรลเลอร์ซึ่งตรงกับไทม์สล็อตการเขียนข้อมูลของอุปกรณ์สเลฟ

2.10.1.4 ไทม์สล็อตการอ่านข้อมูลของไมโครคอนโทรลเลอร์ และการเขียนข้อมูลของอุปกรณ์สเลฟ

เมื่อต้องการอ่านข้อมูลจากอุปกรณ์สเลฟ ไมโครคอนโทรลเลอร์จะทำให้สายสัญญาณเป็นลอจิก "0" ประมาณ 1-15 ไมโครวินาที่ จากนั้นต้องทำให้สภานะของสายกลับมาเป็นลอจิก "1" อุปกรณ์สเลฟจะส่งข้อมูลกลับมาให้ไมโครคอนโทรลเลอร์ ถ้าข้อมูลเป็น "0" อุปกรณ์สเลฟจะทำให้สายสัญญาณเป็นลอจิก "0" นานประมาณ 45 ไมโครวินาที แล้วทำให้สายสัญญาณกลับมาสู่สภาวะลอจิก "1" อีกครั้ง แต่ ถ้าเป็นข้อมูล "1" อุปกรณ์สเลฟจะทำให้สายสัญญาณเป็นลอจิก "1" ต่อเนื่องไปอีก 45 ไมโครวินาที รวมเวลาทั้งหมดประมาณ 60-120 ไมโครวินาที นั่นคือในไทม์สล็อตนี้ต้องใช้เวลารวมไม่เกิน 120 ไมโครวินาที ในขณะที่ไมโครคอนโทรลเลอร์จะใช้เวลาในการอ่านข้อมูลอยู่ระหว่าง 15 และ 60 ไมโครวินาทีหลังจากเริ่มต้นไทม์สล็อตนี้ ในรูปที่ 2.14 แสดงรูปของไทม์สล็อตการอ่านข้อมูลของไมโครคอนโทรลเลอร์ ซึ่งจะตรงกับจังหวะการเขียนข้อมูลของอุปกรณ์สเลฟ และไทม์สล็อตทั้งสองจะเกิดขึ้นในช่วงเวลาเดียวกัน กล่าวคือ เมื่อไมโครคอนโทรลเลอร์อ่านข้อมูล อุปกรณ์สเลฟก็ต้องทำการเขียนข้อมูล

2.10.1.5 ไทม์สล็อตการเขียนข้อมูลของไมโครคอนโทรลเลอร์

เมื่อไมโครคอนโทรลเลอร์ต้องการเขียนข้อมูล จะเริ่มต้นกระบวนการด้วยการทำให้สายสัญญาณเป็นลอจิก "0" ประมาณ 1-15 ไมโครวินาที จากนั้นทำให้สถานะของสายกลับมาเป็นลอจิก "1" แล้วดำเนินการเขียนข้อมูลได้ในทันที ถ้าต้องการเขียนข้อมูล 0 ไมโครคอนโทรลเลอร์จะทำให้สายสัญญาณเป็นลอจิก "0" นานประมาณ 45 ไมโครวินาที แล้วทำให้สายสัญญาณกลับมาสู่สภาวะลอจิก "1" อีกครั้ง แต่ถ้าต้องการเขียนข้อมูล "1" ไมโครคอนโทรเลอร์จะทำให้สายสัญญาณเป็นลอจิก "1" ต่อเนื่องไปอีก 45 ไมโครวินาที รวมเวลาทั้งหมดในไทม์สล็อตนี้ประมาณ 60-120 ไมโครวินาที ในรูปที่ 2.15 และ 2.16

แสดงรูปสัญญาณของไทม์สล็อตการเขียนข้อมูลไมโครคอนโทรเลอร์ซึ่งจะตรงกับจังหวะอ่านข้อมูลของอุปกรณ์ สเลฟและไทม์สล็อตทั้งสองจะเกิดขึ้นในช่วงเวลาเดียวกัน กล่าวคือ เมื่อไมโครคอนโทรลเลอร์เขียนข้อมูลอุปกรณ์สเลฟก็ต้องทำการอ่านข้อมูล

รูปที่ 2.15 ไทม์สล็อตการเขียนข้อมูล "1" ของไมโครคอนโทรลเลอร์ซึ่งตรงกับไทม์สล็อตการอ่านข้อมูลของอุปกรณ์สเลฟ

รูปที่ 2.16 ไทม์สล็อตการเขียนข้อมูล "0" ของไมโครคอนโทรลเลอร์

2.10.2 รูปแบบของการสื่อสารข้อมูลแบบหนึ่งสาย

การสื่อสารข้อมูลในระบบบัสหนึ่งสายไมโครคอนโทรลเลอร์จะติดต่อกับอุปกรณ์สเลฟได้ครั้งละ 1 ตัวเท่านั้น ดังนั้นอุปกรณ์สเลฟแต่ละตัวต้องมีข้อมูลกำหนดแอดเดรสเฉพาะตัว โดยจะเก็บไว้ในหน่วยความจำขนาด 64 บิต หรือ 8 ไบต์ สำหรับเก็บข้อมูลต่างๆที่สำคัญของอุปกรณ์แต่ละตัวซึ่งประกอบด้วย

1. รหัสของตระกูล จำนวน 8 บิต
2. เลขหมายประจำตัว (serial number) จำนวน 48 บิต
3. รหัสตรวจสอบความผิดพลาด (CRC : Cyclical Redundancy Check) จำนวน 8 บิต

ผู้ใช้งานสามารถอ่านข้อมูลประจำตัวของอุปกรณ์สเลฟได้ด้วยการใช้คำสั่ง read rom หรือคำสั่งอ่านหน่วยความจำรอม ในกรณีที่บนสายสัญญาณมีอุปกรณ์สเลฟเพียงตัวเดียวไม่จำเป็นต้องอ้างแอดเดรสในการติดต่อ

รูปแบบการติดต่อบนระบบบัสหนึ่งสายจะเริ่มต้นขึ้นเมื่ออุปกรณ์มาสเตอร์ทำการรีเซตและกำหนดแอดเดรสของอุปกรณ์ที่ทำการติดต่อ ถ้าหากมีอุปกรณ์สเลฟเพียงตัวเดียวสามารถข้ามขั้นตอนการติดต่อกับหน่วยความจำรอบในอุปกรณ์สเลฟได้ เรียกวิธีการดังกล่าวว่า การไม่ติดต่อหน่วยความจำรอม หรือ สคิปรอม(skip rom) จากนั้นรอการตอบรับจากอุปกรณ์สเลฟ เมื่อการตอบรับสมบูรณ์ก็จะสามารถเริ่มต้นขั้นตอนการอ่านหรือเขียนข้อมูลได้ต่อไป

2.10.3 การเขียนโปรแกรมภาษา C เพื่อกำหนดค่าเวลาสำหรับติดต่อกับอุปกรณ์บนระบบบัส 1 สายของไมโครคอนโทรลเลอร์ MCS-51

ในการทำงานบนบัสหนึ่งสายนี้ของไมโครคอนโทรลเลอร์ MCS-51 จะต้องเขียนโปรแกรมเพื่อสร้างรูปสัญญาณให้มีรูปแบบตรงกับข้อกำหนดการสื่อสารชนิดนี้ที่เรียกว่า ไทม์สล็อต ซึ่งจะต้องทำการหน่วงเวลาเป็นช่วงสั้นๆ เพื่อให้ตรงตามเงื่อนไขในการสื่อสาร เนื่องจากในการเขียนโปรแกรมภาษา C นั้นคำสั่งแต่ละคำสั่งจะใช้เวลามาก้อยแตกต่างกันซึ่งไม่อาจคาดเดาได้ นอกจากจะดูจากไฟลิสติ้ง (listing file) ที่ C คอมไพเลอร์สร้างขึ้นจากการเปลี่ยนแปลงคำสั่งภาษา C เป็นคำสั่งภาษาของแอสเซมบลี ซึ่งแต่ละคำสั่งของภาษาแอสเซมบลีจะใช้เวลาที่แน่นอน ซึ่งนับเป็นจำนวนหน่วยของแมชีนไซเคิล โดยที่

เวลา 1 แมชีนไซเกิล = 12/f xtal สำหรับไมโครคอนโทรลเลอร์ MCS-51 มาตรฐานทั่วไป

เวลา 1 แมชีนไซเคิล = 6/f xtal สำหรับไมโครคอนโทรลเลอร์ MCS-51 เบอร์ P89C51RD2BN เมื่อเลือกโหมด 6 สัญญาณนาฬิกาต่อไซเคิล

สำหรับการทดลองนี้ ใช้ไมโครคอนโทรลเลอร์ MCS-51 เบอร์ P89C51RD2BN และเมื่อเลือกโหมด 6 สัญญาณนาฬิกาต่อไซเคิล ส่วนสัญญาณนาฬิกาหลักใช้คริสตอลความถี่ 11.0529 MHz

ดังนั้นเวลา 1 แมชีนไซเคิล = 6/f xtal = 6/11.0529x10 = 0.54 ไมโครวินาที

ในการเขียนโปรแกรมจะใช้คำสั่ง _nop_( ); ซึ่งเรียกจากไฟล์ไลบรารี intrins.h ร่วมกับคำสั่ง for เพื่อใช้หน่วงเวลาให้แต่ละไทม์สล็อตมีค่าตรงตามข้อกำหนดของการสื่อสารข้อมูลบนบัสหนึ่งสาย ในการทดสอบจะเริ่มจากเขียนคำสั่งในโปรแกรมภาษา C แล้วคอมไพล์เป็นโปรแกรมภาษาแอสเซมบลี จากนั้นคำนวณเวลาจากคำสั่งที่ใช้ในโปรแกรมภาษาแอสเซมบลี

2.11 อุปกรณ์ PCF 8591

PCF8591 ไอซีแปลงสัญญาณอนาลอกเป็นดิจิตอลและแปลงดิจิตอลเป็นอนาลอกซึ่งเป็นอุปกรณ์พิเศษในการติดต่อกับไมโครคอนโทรลเลอร์ P89C51RD2 บนระบบบัส I^2C

2.11.1 ข้อมูลเบื้องต้นของ PCF8591

มีรายละเอียดดังนี้

1. ทำงานโดยใช้แหล่งจ่ายไฟชุดเดียว
2. ทำงานที่แรงดัน2.5 V ถึง 6 V
3. กินกระแสขณะอยู่ในสภาวะสแตนบายด์ต่ำ
4. ติดต่อกับไมโครคอมพิวเตอร์หรือไมโครคอนโทรลเลอร์ผ่านระบบบัส I^2C
5. สัญญาณอนาลอกมีระดับแรงดันตั้งแต่ Vss ไปจนถึง Vdd
6. วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลเป็นแบบซักเซสซีฟ แอปพร๊อกซิเมชั่นขนาด 8 บิต
7. มีวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอกขนาด 8 บิต 1 ช่อง
8. เลือกตำแหน่งแอดเดรสทางฮาร์ดแวร์จากขา A0,A1,A2 ทำให้สามารถต่อพ่วงกันได้สูงสุดถึง 8 ตัว
9. อัตราการสุ่มข้อมูล (sampling) ขึ้นอยู่กับความเร็วของสัญญาณนาฬิกาบนบัส I^2C
10. วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล (ADC) สามารถรับสัญญาณอนาลอกได้ 4 ช่อง ทั้งยังเลือกได้ว่าให้ทำงานแบบแบกช่องหรือทำงานเป็นวงจรดิฟเฟอเรนเชียล
11. การอ่านค่าสามารถกำหนดให้เลือกช่องอินพุตโดยอัตโนมัติได้

ส่วนรายละเอียดตำแหน่งขาต่างๆมีดังตารางที่ 2.11

AN0-AN3 (ขา 1-4)	อินพุตสำหรับป้อนสัญญาณอนาลอกที่ต้องการแปลงค่า
A0-A2 (ขา 5-7)	ขากำหนดแอดเดรสทางฮาร์ดแวร์ ปกติต่อลงกราวด์แต่ถ้ามีการใช้งาน PCF8591 มากกว่า 1 ตัว ต้องกำหนดการต่อขา A0-A2 ของ PCF8591 ให้ไม่ตรงกัน จึงทำให้สามารถต่อใช้งานร่วมกันได้สูงสุด 8 ตัว
Vss (ขา 8)	ขาต่อกราวด์
SDA , SCL (ขา 9,10)	ขาเชื่อมต่อระบบบัส I^2C
OSC (ขา 11)	ขาต่อกับสัญญาณนาฬิกาภายนอกเมื่อขา EXT ต่อกับไฟ +5V และเป็นเอาต์พุตสัญญาณนาฬิกาถ้าขา EXT ต่อลงกราวด์
EXT (ขา 12)	ขาเลือกตำแหน่งกำเนิดสัญญาณนาฬิกา ถ้าต่อ +5V เป็นการเลือกใช้สัญญาณนาฬิกาจากภายนอก โดยต่อสัญญาณนาฬิกาเข้าที่ขา OSC ถ้าต่อขานี้ลง

	กราวด์ จะเป็นการเลือกใช้สัญญาณนาฬิกาจากภายใน
AGND (ขา 13)	ขากราวด์ของแรงดันอ้างอิง ปกติต่อลงกราวด์
Vref (ขา 14)	ขาป้อนแรงดันอ้างอิง ปกติต่อเข้าไฟเลี้ยง +5V
AOUT (ขา 15)	ขาเอาต์พุตของวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอก
Vdo (ขา 16)	ขาต่อไฟเลี้ยง จ่ายได้ตั้งแต่ +2 ถึง +6V ปกติใช้ +5V

ตารางที่ 2.11 แสดงขาต่างๆของไอซี PCF8591

2.11.2 ข้อมูลควบคุม

หลังจากส่งข้อมูลกำหนดแอดเดรสให้แก่ PCF8591 แล้ว ต้องส่งข้อมูลควบคุมตามไปด้วยเพื่อกำหนดคุณสมบัติของวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลและวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอกภายใน PCF8591 โดย บิต 6 ของข้อมูลควบคุมใช้สำหรับการเอ็นเอเบิลขาอนาลอกเอาต์พุต เมื่อต้องการเอ็นเอเบิลต้องกำหนดให้ขานี้เป็น "1" บิต 4 และบิต 5 ของข้อมูลควบคุมใช้สำหรับการกำหนดรูปแบบของสัญญาณอนาลอกอินพุตที่ป้อนให้แก่ PCF8591 บิต 2 ใช้สำหรับเลือกรูปแบบการอ่านข้อมูลจากขาอินพุตอนาลอกว่าจะเป็นการอ่านจากเพียงอินพุตเดียวหรืออ่านแบบเรียงลำดับทุกอินพุต ถ้าต้องการเลือกให้อ่านแบบเรียงลำดับต้องกำหนดให้บิตนี้เป็น "1" บิต 0 และ 1 ใช้สำหรับกำหนดช่องของอินพุตอนาลอกที่ต้องการอ่าน ถ้ากำหนดให้บิต 2 เป็น "1" หลังจากอ่านค่าของบิต "0" และบิต "1" แล้วในการอ่านค่าครั้งต่อไปจะเป็นการอ่านค่าอินพุตจากช่องที่ 1 โดยที่ข้อมูลควบคุมทั้งหมดจะถูกเก็บไว้ในรีจิสเตอร์ควบคุมภายใน PCF8591 เมื่อจ่ายไฟให้แก่ PCF8591 ครั้งแรกบิตต่างๆของข้อมูลภายในรีจีสเตอร์ควบคุมจะเป็น "0" และวงจรออสซิเลเตอร์ภายใน PCF8591 จะสร้างสัญญาณนาฬิกาสำรับการแปลงสัญญาณอนาลอกเป็นดิจิตอล เมื่อต้องการใช้วงจรออสซิลเลเตอร์ภายในขา EXT ต้องต่อลงกราวด์ ถ้าต้องการใช้ออสซิลเลเตอร์จากภายนอกขา EXT ต้องต่อเข้ากับไฟบวก และป้อนสัญญาณนาฬิกาเข้าที่ขา OSC ของ PCF8591 ความถี่ของสัญญาณนาฬิกาสูงสุดที่ป้อนให้กับออสซิลเลเตอร์เท่ากับ 1.25 MHz

2.11.3 โครงสร้างของโปรแกรม

ข้อมูลที่ต้องส่งไปยัง PCF8591 คือ ข้อมูลแอดเดรส และข้อมูลควบคุมโดยที่ข้อมูลแอดเดรสมีลักษณะดังนี้

1	0	0	1	A2	A1	A0	R/W
บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0

รูปที่ 2.17 แสดงบิตแอดเดรส

ข้อมูลถัดมาคือคำสั่งควบคุม (control byte) ใช้ในการเลือกอ่านข้อมูลจากช่องสัญญาณอินพุตอนาลอก และกำหนดค่าของสัญญาณอนาลอกที่ต้องการส่งออกไปทางเอาต์พุตอนาลอกดังรูปที่ 2.18

รูปที่ 2.18 รายละเอียดข้อมูลควบคุมที่เขียนลงในรีจิสเตอร์ควบคุมภายใน IC PCF8591

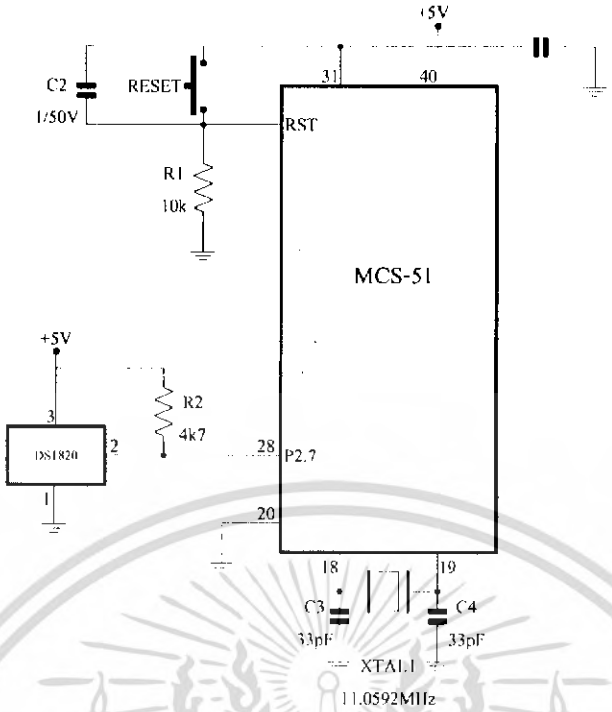
2.12 อุปกรณ์ DS1820

DS1820 เป็นไอซีตรวจจับอุณหภูมิที่ใช้การติดต่อแบบระบบบัสหนึ่งสาย มีขาต่อใช้งานเพียง 3 ขา คือ DQ ซึ่งเป็นขาเชื่อมต่อกับระบบบัส , ขาต่อไฟเลี้ยงภายนอก และขากราวด์ ดังแสดงการจัดขา และโครงสร้างการทำงานภายในแสดงในตารางที่ 2.10 หัวใจสำคัญของ DS 1820 อยู่ที่ตัวตรวจจับอุณหภูมิและหน่วยความจำความเร็วสูงที่เรียกว่า สแครตช์แพด (scrachtchpad) ขนาด 9 ไบต์ มีการจัดสรรหน่วยความจำส่วนนี้

เมื่อวัดอุณหภูมิได้ก็จะนำค่าที่วัดได้นี้มาเก็บไว้ในสแครตช์แพดที่ไบต์ 0 และ 1 ทั้งนี้เนื่องจากไอซี DS1820 สามารถให้ข้อมูลของอุณหภูมิได้ละเอียดถึง 16 บิต เมื่อจำมาแปลงเป็นข้อมูลเลขฐานสิบจึงสามารถแสดงความละเอียดของค่าอุณหภูมิได้ถึง 0.5 องศาเซลเซียสและ 0.9 องศาฟาเรนไฮต์ โดยมีย่านวัดอุณหภูมิ -55 ถึง +125 องศาเซลเซียส หรือ -67 ถึง +257 องศาฟาเรนไฮต์ โดยค่าขององศาฟาเรนไฮต์ต้องใช้การแปลงหน่วยเข้ามาช่วย ใช้เวลาในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลประมาณ 200 มิลลิวินาที สามารถกำหนดขอบเขตของอุณหภูมิที่ทำการวัดได้ และให้แจ้งเตือนเมื่อค่าของอุณหภูมิสูงขึ้นหรือลดต่ำลงถึงค่าที่กำหนด โดยค่าอุณหภูมิที่กำหนดนี้จะเก็บไว้ที่สแครตช์แพดในไบต์ 2 และ 3

คำอธิบาย	ไบต์
ข้อมูลอุณหภูมิไบต์ต่ำ(TL)	0
ข้อมูลอุณหภูมิไบต์สูง	1
ข้อมูลอุณหภูมิค่าสูง	2
ข้อมูลอุณหภูมิค่าต่ำ (TL)	3
สำรองไว้	4
สำรองไว้	5
รีจิสเตอร์เก็บค่าการนับ	6
รีจิสเตอร์เก็บค่าการนับต่อ C	7
CRC	8

ตารางที่ 2.12 แสดงการจัดสรรพื้นที่ของสแครตช์แพดใน DS1820



รูปที่ 2.19 การเชื่อมต่อ DS1820 กับไมโครคอนโทรลเลอร์ MCS-51

2.12.1 คำสั่งเพื่อควบคุมการทำงานของ DS1820

ในการติดต่อกับไอซี DS1820 จะมีคำสั่งที่ต้องส่งให้แก่ DS1820 เพื่อกำหนดรูปแบบการทำงาน คำสั่งที่ใช้มากที่สุดมีด้วยกัน 3 คำสั่งดังนี้

1. คำสั่งไม่ติดต่อกับหน่วยความจำรอม หรือ สคิปรอม (skip rom) เนื่องจากในการใช้งาน DS1820 โดยปกติแล้วจะมี DS1820 อยู่บนสายสัญญาณเพียงตัวเดียว จึงไม่จำเป็นต้องใช้ข้อมูลกำหนดแอดเดรส ดังนั้นจึงไม่ต้องติดต่อกับหน่วยความจำรอม เพื่ออ่านอ่านข้อมูล ข้อมูลของคำสั่งสคิปรอมที่ต้องส่งให้ DS1820 คือ 0xCC
2. คำสั่งแปลงอุณหภูมิ (Convert T) มีค่าเท่ากับ 0x44 เมื่อส่งคำสั่งนี้ให้ DS1820 จะต้องทำการวนลูปรออย่างน้อย 200 มิลลิวินาที เพื่อให้ DS1820 ได้ใช้เวลานี้ในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลมาเก็บไว้ในสแครตช์แพด
3. คำสั่งอ่านข้อมูลจากสแครตช์แพด (Read Scratchpad) มีค่าเท่ากับ 0xBE เมื่อส่งคำสั่งนี้ DS1820 จะทยอยส่งข้อมูลค่าอุณหภูมิออกมาทั้งหมด 9 ไบต์

2.12.2 การเชื่อมต่อไมโครคอนโทรลเลอร์ MCS-51

ไอซีตัวนี้ต้องการใช้ขาพอร์ตเพียง 1 ขาเท่านั้นสำหรับการเชื่อมต่อกับ DS1820 โดยต้องมีตัวต้านทานค่า 1 ถึง 4.7kΩ ต่อพูลอัพกับไฟเลี้ยง 5 โวลต์จากนั้นจึงเขียนโปรแกรมเพื่อติดต่อกัน โดยใช้รูปแบบการติดต่อตามมาตรฐานระบบบัสหนึ่งสายของดัลลัสและนอกจากนั้นมีการเรียกใช้งานฟังก์ชั่นต่างๆ เพื่อสร้างรูปแบบการสื่อสารบนระบบบัสหนึ่งสายซึ่งอธิบายได้ดังนี้

1. ฟังก์ชั่นที่ทำให้เกิดการรีเซตบนบัส เพื่อเริ่มต้นติดต่อกับอุปกรณ์สเลฟ ด้วยการทำให้สายสัญญาณเป็นลอจิก "0" เป็นเวลาประมาณ 800 ไมโครวินาที และกลับมาเป็น ลอจิก "1" เป็นเวลาประมาณ 16 ไมโครวินาทีหรือ 50 เท่านั่นเอง
2. ฟังก์ชั่นรอการตอบรับจากอุปกรณ์สเลฟ
3. ฟังก์ชั่นอ่านข้อมูลที่ส่งมาบนบัสหนึ่งสายทีละ 1 บิต ซึ่งฟังก์ชั่นนี้จะคืนค่าผลลัพธ์ เป็นค่าของข้อมูลที่อ่านได้นั่นเอง เริ่มต้นด้วยการทำให้สายสัญญาณเป็นลอจิก "0" เป็นเวลาประมาณ 2 ไมโครวินาที และกลับมาเป็นลอจิก "1" ประมาณ 4 ไมโครวินาที แล้วอ่านค่าข้อมูลจากบัส หลังจากนั้นหน่วงเวลาอีกประมาณ 64 ไม่โครวินาที
4. ฟังก์ชั่นอ่านค่าอุณหภูมิซึ่งเป็นข้อมูล 1 ไบต์ ด้วยการเรียกใช้ฟังก์ชั่นอ่านข้อมูล 1 บิต 8 ครั้ง
5. ฟังก์ชั่นส่งคำสั่งซึ่งเป็นข้อมูล 1 ไบต์ เพื่อควบคุมการทำงานของ ไอซี DS1820

2.13 การเขียนโปรแกรมเชิงวัตถุและคลาส (Object-Oriented Programming & Class)

ในการทดลองนี้ใช้ในการเขียนโปรแกรมควบคุมบอร์ดอีเธอร์เนตไอโอ

2.13.1 หลักการเขียนโปรแกรมเชิงวัตถุ

ภาษา C++ เป็นภาษาเชิงวัตถุเพราะสามารถสนับสนุนหลักการเขียนโปรแกรมเชิงวัตถุซึ่งจะประกอบด้วย

1. การกำหนดข้อมูลนามธรรม (Abstraction)
2. การซ่อนข้อมูล (Data Hiding)
3. การทำให้ข้อมูลอยู่ในขอบเขตจำกัด (Encapsulation)
4. การสืบทอดและการนำโค้ดกลับมาใช้อีกครั้ง(Inheritance & Reusable Code)
5. การทำหลายรูปแบบ (Polymorphism)

2.13.2 การเขียนโปรแกรมเชิงวัตถุ

ข้อเสียอย่างหนึ่งของการเขียนโปรแกรมโครงสร้าง ก็คือ เมื่อสร้างเป็นโปรเจ็กต์ขนาดใหญ่ๆ จะไม่เหมาะสม โดยเฉพาะเรื่องการจัดการข้อมูล แต่การเขียนโปรแกรมเชิงวัตถุ หรือเชิงออบเจ็กต์ จะได้รับความนิยมมากกว่าเนื่องจาก การเขียนโปรแกรมเชิงวัดถุจะเน้นไปที่ข้อมูลเป็นหลัก โดยการใช้คลาส ทำให้มีความเหมาะสมต่อการพัฒนาโปรแกรมขนาดใหญ่ๆ

การเขียนโปรแกรมเชิงวัตถุจะมองไปที่ปัญหา แล้วตีความหมายออกมาเป็นข้อมูลชนิดใหม่ หรือเรียกว่า Abstract Data Type โดยใช้คำสงวนคลาส (Class) ในคลาสหนึ่งก็จะรวบรวมข้อมูลชนิดต่างๆ และฟังก์ชันทั้งหลายเข้าด้วยกัน ที่สามารถนำมาใช้แก้ปัญหาโจทย์ ลำดับต่อไปก็จะนำคลาสมาสร้างเป็นออบเจ็กต์ แล้วส่งเมสเสจหรือเรียกเมมเบอร์ฟังก์ชั่นต่างๆใช้งาน เราก็จะได้โปรแกรมเชิงวัตถุที่แก้ปัญหาโจทย์ได้

อย่างไรก็ตามในการเขียนโปรแกรมเชิงวัตถุใช้งานจริงๆ จะมีหลักเกณฑ์มากมายและซับซ้อนกว่าการเขียนโปรแกรมเชิงโครงสร้างอย่างมาก ตัวอย่างเช่น วิธีการทำการก๊อปปี้คอนสตรัคเตอร์อย่างลึก

(Deep Copy Constructor) การทำโอเวอร์โหลดโอเปอร์เรเตอร์ต่างๆ การนำโค้ดที่มีอยู่แล้วมาใช้อีกครั้ง วิธีการสร้างคลาสเทมเพลต และอื่นๆรูปแบบต่างๆ พร้อมทั้งวิธีการนำมาใช้งานอย่างละเอียดเป็นขั้นตอน ซึ่งจะครอบคลุมเนื้อหาตามหลักการเขียนโปรแกรมเชิงวัตถุได้ในระดับหนึ่ง

2.13.3 คลาส (Class)

เมื่อแก้ปัญหาหรือวิเคราะห์โจทย์ได้แล้ว จะได้ข้อมูลและการกระทำประกอบกัน ข้อมูลจะบ่งบอกถึงคุณลักษณะของคลาสที่เรียกว่า แอตทริบิวต์ (Attributes) ภาษา C++ จะใช้ศัพท์เทคนิคว่า ดาต้าเมมเบอร์ ส่วนการกระทำต่างๆ ของคลาสจะบ่งบอกถึงพฤติกรรมของคลาส ภาษา C++ จะใช้ศัพท์เทคนิคว่า เมมเบอร์ฟังก์ชัน

Private , protected และ public คือคำสงวนที่ใช้กำหนดการเข้าถึงข้อมูล ดังรายละเอียดต่อไปนี้

2.13.3.1 คำสงวน private

ดาต้าเมมเบอร์และหรือเมมเบอร์ฟังก์ชันที่ถูประกาศการเข้าถึงอยู่ในกลุ่ม private จะถูกจำกัดการใช้งานได้เฉพาะภายในขอบเขตคลาสเท่านั้น หรือภายในเมมเบอร์ฟังก์ชัน ไม่สามารถนำดาต้าเมมเบอร์หรือเมมเบอร์ฟังก์ชันไปใช้ภายนอกขอบเขตคลาส (Class Scope) โดยปกติแล้ว จะประกาศดาต้าเมมเบอร์อยู่ในกลุ่ม private มากกว่า เพราว่าไม่ละเมิดกฎของการเขียนโปรแกรมเชิงวัตถุที่ว่า ข้อมูลจะต้องถูกซ่อนไว้และจะต้องอยู่ในขอบเขตที่จำกัด

2.13.3.2 คำสงวน protected

ดาต้าเมมเบอร์และหรือเมมเบอร์ฟังก์ชัน ที่ถูกประกาศการเข้าถึงแบบ protected จะถูกกำกัดการใช้งานได้เฉพาะภายในขอบเขตคลาสและคลาสสืบทอด หรือภายในเมมเบอร์ฟังก์ชัน โดยส่วนใหญ่จะนิยมประกาศดาต้าเมมเบอร์อยู่ในกลุ่ม protected เมื่อต้องการให้คลาสสืบทอดสามารถใช้หรือเข้าถึงข้อมูล

2.13.3.3 คำสงวน public

ดาต้าเมมเบอร์และเมมเบอร์ฟังก์ชันที่ถูกประกาศการเข้าถึงแบบ public จะเหมือนกับการประกาศออบเจ็กต์โกลบอล คือสามารถเรียกใช้งานได้ทุกที่ภายในโปรแกรม ถ้าเรียกภายนอกขอบเขตคลาส จะต้องเรียกผ่านออบเจ็กต์ โดยปกติแล้วจะประกาศเมมเบอร์ฟังก์ชัน อยู่ในกลุ่ม public เพื่อให้คลาสมีอินเตอร์เฟส และมีน้อยมากที่จะประกาศดาต้าเมมเบอร์ให้อยู่ในกลุ่มนี้ เพราะว่าจะละเมิดกฎของการเขียนโปรแกรมเชิงวัตถุที่ว่า ข้อมูลต้องถูกซ่อนไว้ และจะต้องอยู่ในของเขตที่จำกัด

Data member คือสมาชิกคลาส ดาต้าเมมเบอร์ส่วนใหญ่จะเป็นข้อมูลชนิดพื้นฐาน เช่น double , Int , flot boll หรือ พอยน์เตอร์ที่ชี้ไปยังข้อมูลชนิดต่างๆ เป็นต้น นอกจากนี้ก็จะเป็นข้อมูลชนิดที่สร้างขึ้นใหม่ เช่น string (ต้องรวมไฟล์ไลบรารี string เข้ามาด้วย) คลาสที่สร้างขึ้นใหม่ เป็นต้น บางครั้งจะมีการใช้คำสงวน const , static , volatile , mutable หรือใช้ร่วมทั้ง const และ static วางหน้าดาต้าเมเบอร์

Member function คือ สมาชิกหนึ่งของ คลาส เมมเบอร์ฟังก์ชันส่วนใหญ่จะเป็นคอนสตรัคเตอร์ (constructor) โอเปอร์เรเตอร์กำหนดค่า (Assignment Operator) ฟังก์ชันต่างๆ สแตติกเมมเบอร์ฟังก์ชัน (Static Member Function) เป็นต้น

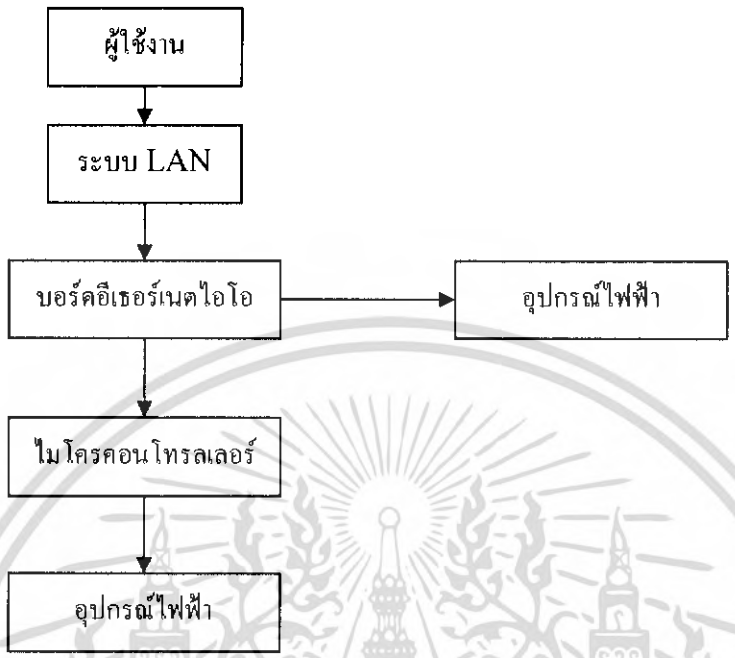
สมาชิกคลาส นอกจากดาต้าเมมเบอร์และเมมเบอร์ฟังก์ชันแล้ว ยังสามารถมีคลาสซ้อนคลาสได้ (Nested Class)

ส่วน friend ฟังก์ชันไม่ถือว่าเป็นสมาชิกคลาส แต่เป็นฟังก์ชันที่อำนวยความสะดวกในการใช้ดาต้าเมมเบอร์ที่ถูกประกาศการเข้าถึงแบบ private หรือ protection

ในการประกาศเฟร็นด์ฟังก์ชัน (Friend Function) จะใช้คำสงวน friend วางหน้าฟังก์ชันไว้ในคลาส และจะถูกประกาศการเข้าถึงให้อยู่ในกลุ่มใดก็ได้ เนื่องจากเพร็นด์ฟังก์ชันเป็นฟังก์ชันโกลบอล จึงไม่ถูกจำกัดการใช้งานภายในขอบเขตคลาส และสามารถเรียกเฟร็นด์ฟังก์ชันได้โดยตรงโดยไม่ต้องเรียกผ่านออบเจ็กต์

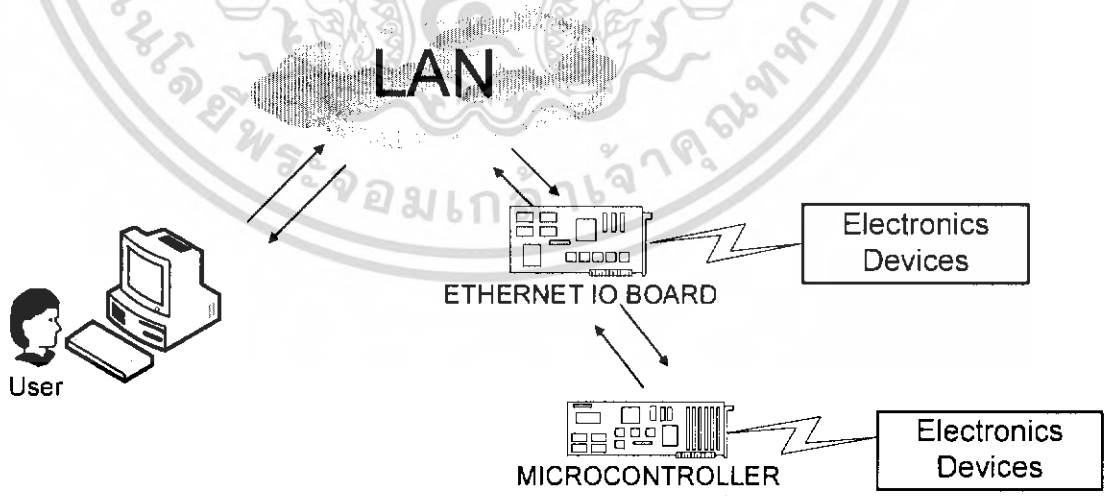
บทที่ 3 การคำนวณและการสร้าง

3.1 หลักการทำงาน



รูปที่ 3.1 แสดงการทำงานของระบบโดยรวม

หลักการของการทำงานโดยรวมของโปรแกรมคือ ผู้ใช้จะทำการสั่งงานไปยังเครื่องใช้ไฟฟ้าต่างๆ หรือสั่งงานไปยังไมโครคอนโทรลเลอร์ ผ่านทางระบบ LAN เพื่อทำการเปิดหรือปิดอุปกรณ์ไฟฟ้า



รูปที่ 3.2 ภาพของระบบโดยรวม

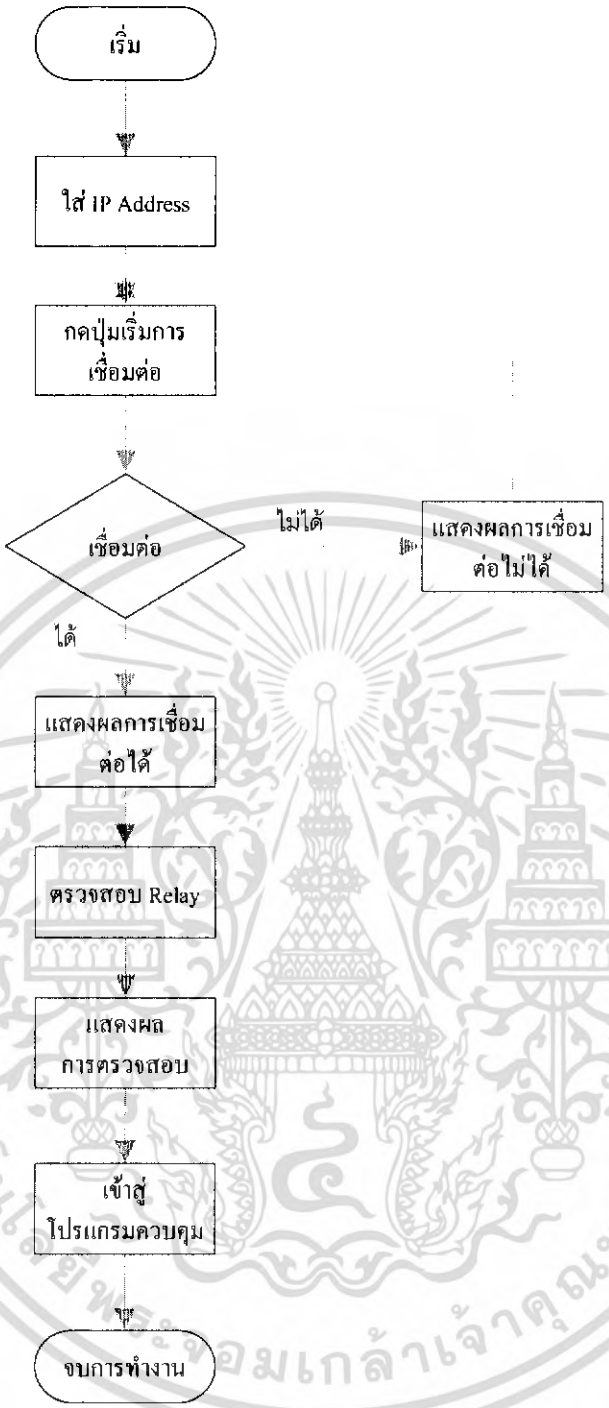
3.2 โปรแกรมควบคุมบอร์ดอีเธอร์เนตไอโอ

ในการทำงานของโปรแกรมที่ใช้ควบคุมบอร์ดอีเธอร์เนตไอโอนั้นสามารถควบคุมได้ด้วยโปรแกรมVisual C++ ซึ่งบอร์ดอีเธอร์เนตไอโอนี้จะสามารถเป็นพอร์ทอินพุทเอาท์พุทที่สามารถจ่ายโลจิกออกมาทางพอร์ทอินพุทเอาพุททั่วไปของบอร์ดได้และยังสามารถเชื่อมต่อกับ Serial Port เพื่อเป็นตัวกลางในการส่งผ่านข้อมูลได้

ในโครงงานนี้ได้ใช้บอร์ดอีเธอร์เนตไอโอนี้ควบคุมวงจรรีเลย์ทางทางพอร์ทอินพุทเอาพุททั่วไปของบอร์ดและควบคุมวงจรปรับอุณหภูมิและอ่านค่าอุณหภูมิผ่านทางไมโครคอนโทรลเลอร์ซึ่งเชื่อมต่อกับบอรดอีเธอร์เนตไอโอทาง Serial Port ซึ่งได้แสดงฟังก์ชั่นต่างๆดังนี้

3.2.1 ฟังก์ชั่นการเชื่อมต่อกับบอร์ด

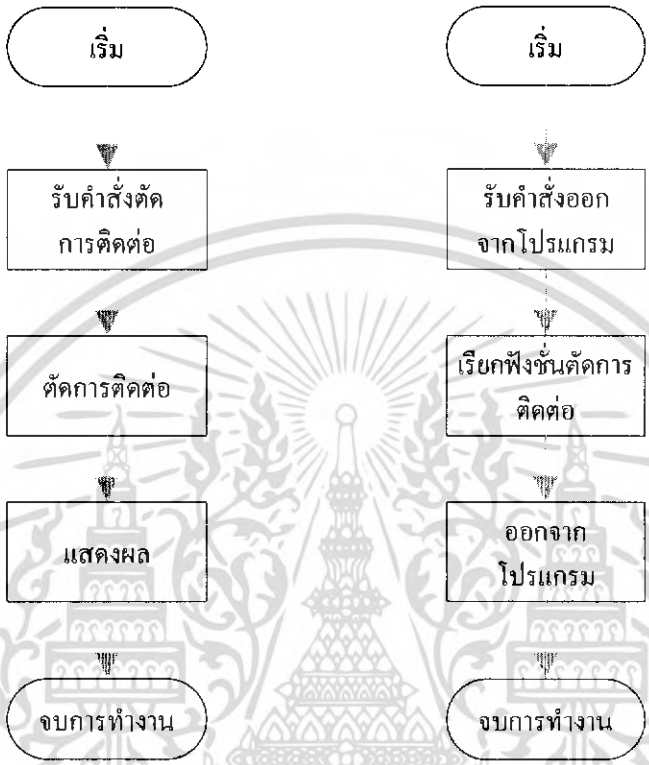
การทำงานของฟังก์ชั่นนี้เป็นการเริ่มต้นการเชื่อมต่อระหว่างโปรแกรมในเครื่องคอมพิวเตอร์กับบอร์ดอีเธอร์เนตไอโอโดยการเชื่อมต่อนี้ผ่านระบบ IP Address ซึ่งระบบนี่ได้มีการใช้ในระบบของ LAN การเชื่อมต่อเริ่มต้นโดยการกำหนดค่า IP Address ลงไปในโปรแกรมจากนั้นจึงสั่งให้โปรแกรมทำการเชื่อมต่อกับบอร์ดผ่านทาง IP Address ที่กำหนดค่าไว้ผ่านระบบ LAN และรอการตอบรับจากบอร์ดอีเธอร์เนตไอโอ เมื่อบอร์ดอีเธอร์เนตไอโอรับรู้ถึงความต้องการในการเชื่อมต่อจะส่งการตอบรับกลับมายังโปรแกรมในคอมพิวเตอร์ และเมื่อโปรแกรมได้รับการตอบรับแล้วโปรแกรมจะทำการตรวจสอบสถานะเบื้องต้นของบอร์ดอีเธอร์เนตไอโอในขณะนั้นจากนั้นผู้ใช้จึงสามารถควบคุมอุปกรณ์ไฟฟ้าผ่านบอร์ดอีเธอร์เนตไอโอได้



รูปที่ 3.3 แสดงการทำงานในส่วนของการเชื่อมต่อโปรแกรม

3.2.2 ฟังก์ชั่นการหยุดการเชื่อมต่อและออกจากโปรแกรม

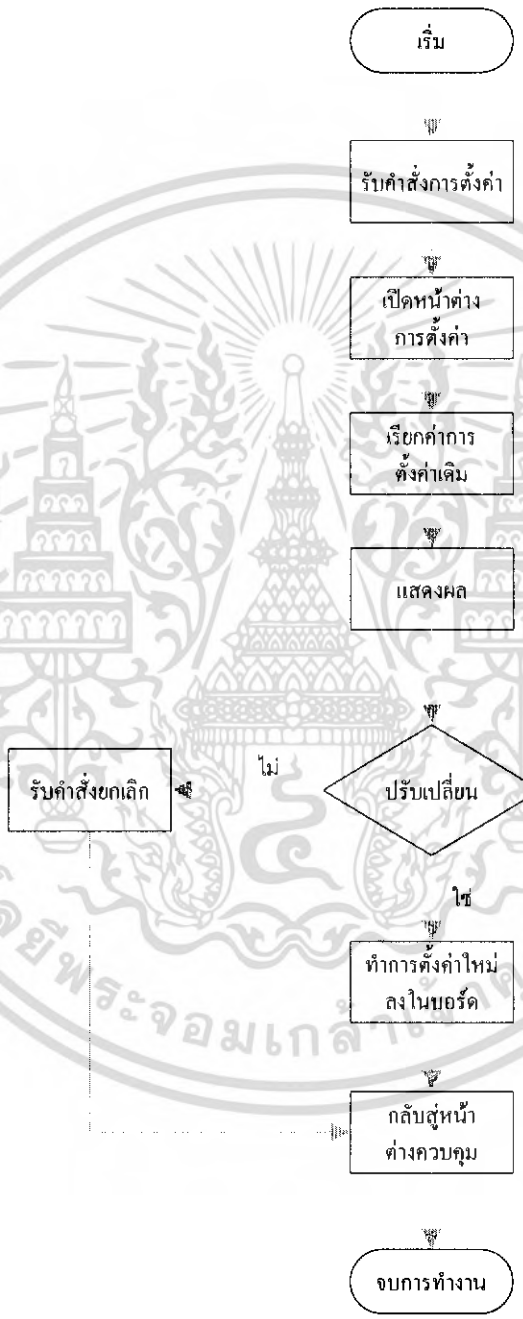
การหยุดการเชื่อมต่อสามารถใช้เมื่อต้องการตัดการติดต่อกับบอร์ดอีเธอร์เนตไอโอและการออกจากโปรแกรมนั้นต้องทำการหยุดการเชื่อมต่อเสียก่อนดังนั้นจึงเรียกฟังก์ชั่นหยุดการเชื่อมต่อก่อนจึงทำการออกจากโปรแกรม



รูปที่ 3.4 - 3.5แสดงการหยุดการเชื่อมต่อและ แสดงการออกจากโปรแกรม

3.2.3 ฟังก์ชั่นการตั้งค่าการเชื่อมต่อต่างๆบอร์ดอีเธอร์เนตไอโอ

ในส่วนนี้เป็นการตั้งค่าการเชื่อมต่อของบอร์ดอีเธอร์เนตไอโอ การตั้งค่าการเชื่อมต่อนั้นมีอยู่ด้วยกัน 7 ค่าคือ IP Address Subnet Gateway Baud rate Bit Parity Stop bit โดยการเปิดหน้าต่างการตั้งค่านี้จำเป็นต้องเรียกค่าเดิมที่ได้ตั้งไว้มาแสดงผลเพื่อที่จะได้รู้ถึงค่าต่างๆที่ได้ตั้งไว้อยู่แล้วแล้วจึงทำการตั้งค่าให้บอร์ดอีเธอร์เนตไอโอใหม่



รูปที่ 3.6 แสดงการตั้งค่าการเชื่อมต่อต่างๆบอร์ดอีเธอร์เนตไอโอ

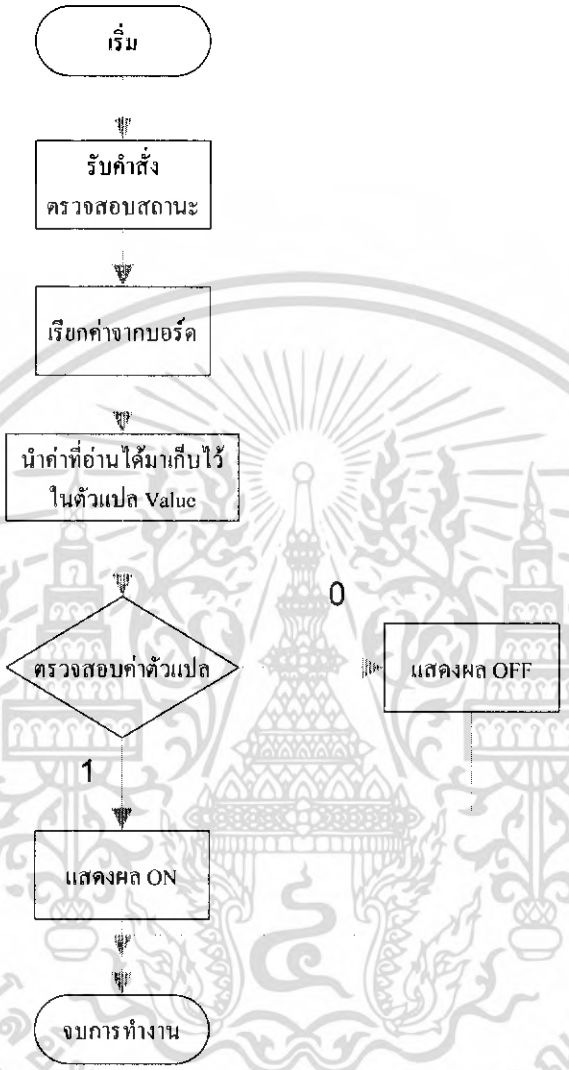
3.2.4 ฟังก์ชั่นการเปิดและปิดไฟ

โดยการเปิดและปิดไฟนี้จะควบคุมโดยการเปิดและปิดรีเลย์จากบอร์ดอีเธอร์เนตไอโอ คำสั่งในการเปิดและปิดนี้ทำโดยการส่งคำสั่งในการเปิดและปิดจากโปรแกรมไปยังบอร์ดอีเธอร์เนตไอโอโดยใช้ค่าลอจิกสูงและลอจิกต่ำในการควบคุม

รูปที่ 3.7 - 3.8 แสดงการส่งคำสั่งจากโปรแกรมไปยังบอร์ดอีเธอร์เนตไอโอเพื่อเปิดและปิดไฟ

3.2.5 ฟังก์ชั่นการตรวจสอบการเปิดหรือปิดไฟ

การตรวจสอบนี้จะทำการตรวจสอบสถานะของไฟในขณะนั้นโดยการตรวจสอบที่บอร์ดอีเธอร์เนตไอโอโดยเป็นการอ่านค่าลอจิกของบอร์ดอีเธอร์เนตไอโอที่กระทอยู่กับไฟ ณ จุดนั้นแล้วจึงนำผลที่ได้มาวิเคราะห์แล้วจึงทำการแสดงผลที่หน้าจอโปรแกรมควบคุม



รูปที่ 3.9 แสดงการตรวจสอบสถานะของไฟ

3.2.6 ฟังก์ชั่นการเปิด ปิด และตรวจสอบไฟทั้งหมด

การเปิด ปิดและการตรวจสอบทั้งหมดนั้นจะเป็นการเรียกฟังก์ชั่นของการเปิด ปิด และการตรวจสอบของทุกตัวมาใช้งานทั้งหมดแล้วจึงทำการแสดงผลที่หน้าจอโปรแกรมควบคุม

รูปที่ 3.10 – 3.11 แสดงการเปิดและปิดไฟทั้งหมด

รูปที่ 3.12 แสดงการตรวจสอบสถานะทั้งหมด

3.2.7 ฟังก์ชั่นการส่งข้อมูลควบคุมไปยังไมโครคอนโทรลเลอร์

การควบคุมไมโครคอนโทรลเลอร์ผ่านบอร์ดอีเธอร์เนตไอโอนั้นต้องใช้การควบคุมผ่าน Serial Port ของบอร์ดอีเธอร์เนตไอโอโดยที่โปรแกรมควบคุมบอร์ดอีเธอร์เนตไอโอนั้นจะสั่งการให้บอร์ดอีเธอร์เนตไอโอส่งคำสั่งไปยังไมโครคอนโทรลเลอร์ให้ปฏิบัติตามคำสั่งของผู้ใช้

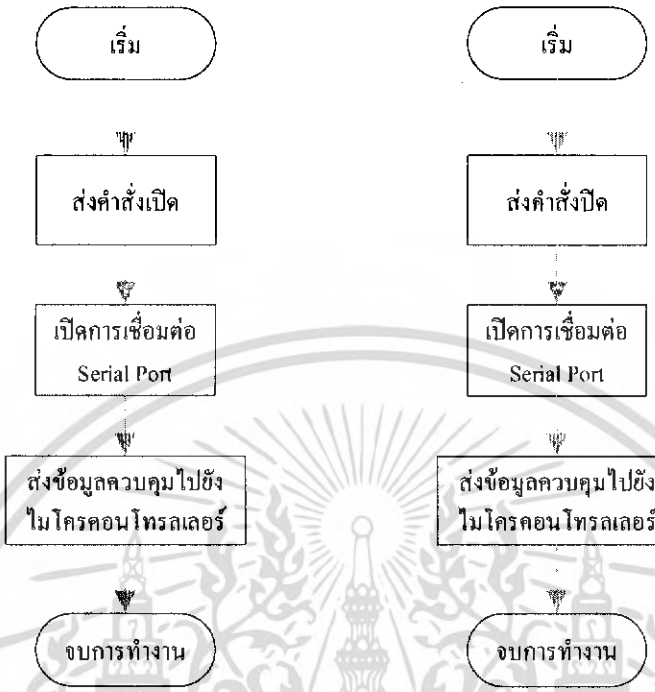
3.2.7.1 คำสั่งตรวจสอบอุณหภูมิ

เป็นคำสั่งที่ไมโครคอนโทรลเลอร์ทำการอ่านค่าอุณหภูมิที่ไอซี DS1820 แล้วส่งผลที่อ่านได้มาแสดงค่าที่โปรแกรมควบคุมบอร์ดอีเธอร์เนตไอโอ

รูปที่ 3.13 แสดงการอ่านค่าอุณหภูมิจาก IC วัดอุณหภูมิ

3.2.7.2 คำสั่งเปิดและปิดอุปกรณ์ควบคุมอุณหภูมิ

คำสั่งนี้จะเป็นการควบคุมการเปิดปิดอุปกรณ์โดยใช้ไมโครคอนโทรลเลอร์พอร์ท 1.2



รูปที่ 3.14-3.15แสดงการส่งคำสั่งเพื่อเปิดและปิดแอร์ไปยังไมโครคอนโทรลเลอร์ ผ่าน P1.2

3.2.7.3 คำสั่งส่งอุณหภูมิไปยังไอซีควบคุมอุณหภูมิ

คำสั่งนี้ทำโดยเลือกอุณหภูมิที่ต้องการแล้วจึงทำการส่งอุณหภูมิที่ต้องการไปยังไอซี

รูปที่ 3.16 แสดงการส่งข้อมูลอุณหภูมิไปยังไมโครคอนโทรลเลอร์

3.3 วงจรที่ใช้ในการทดลอง

3.3.1 วงจรควบคุมอุณหภูมิแอร์

ในการทดลองได้ใช้ IC PCF8591 ในการจำลองควบคุมอุณหภูมิแอร์โดยมีการกำหนดค่าในการควบคุมดังนี้

ข้อมูลแอดเดรส เนื่องจากในวงจรที่ทำการทดลองใช้ PCF8591 เพียงตัวเดียว แล้วต่อขา A0-A2 ลงกราวด์ ดังนั้นโครงสร้างข้อมูลแอดเดรสของ PCF8591 ในการทดลองนี้เป็นดังนี้

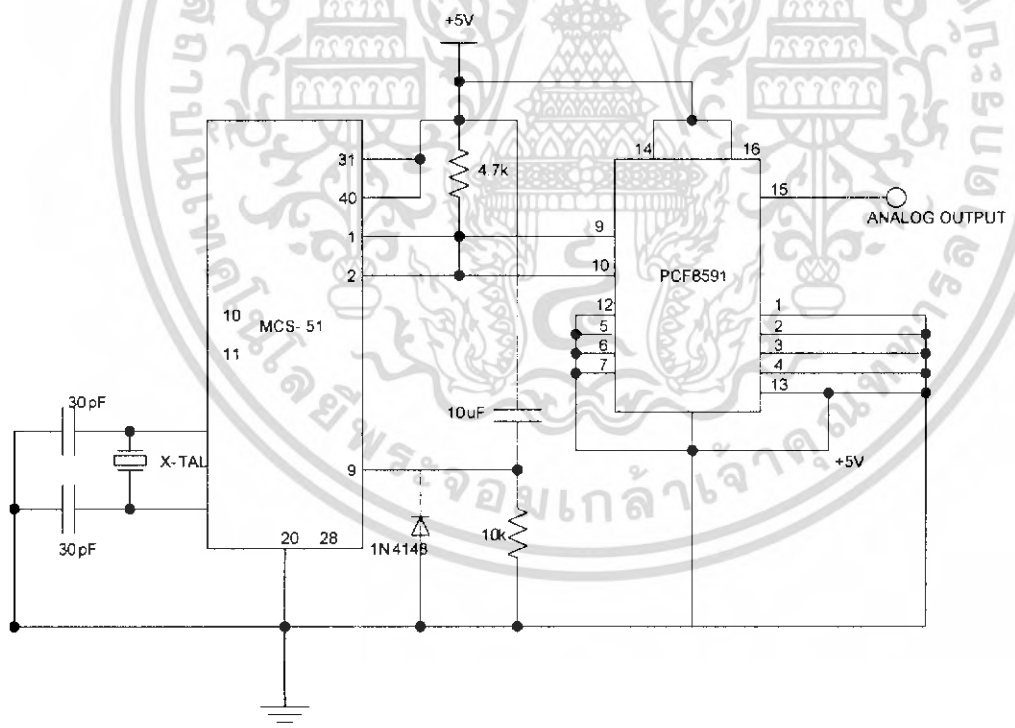
1	0	0	1	0	0	0	R/W
บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0

รูปที่ 3.17แสดงบิตแอดเดรสที่ใช้งาน

สามารถกำหนดข้อมูลแอดเดรสเท่ากับ

0x90 เมื่อต้องการเขียนข้อมูลไปยัง PCF8591 และ

0x91 เมื่อต้องการอ่านข้อมูลจาก PCF8591



รูปที่ 3.18 วงจรจำลองสำหรับการควบคุมอุณหภูมิ

3.3.2 วงจรอ่านค่าอุณหภูมิ

ไอซี DS1820 เป็นไอซีที่ออกแบบเพื่อวัดอุณหภูมิในย่าน -55 ถึง +125 องศาเซลเซียสโดยใช้สายสัญญาณเชื่อมต่อกับไมโครคอนโทลเลอร์เพียงเส้นเดียวโดยในการควบคุมมีขั้นตอนดังนี้

1. เริ่มต้นด้วยการส่งสถานะรีเซตด้วยการเรียกฟังก์ชั่น onewire_reset
2. รอการตอบรับจาก DS1820 ด้วยฟังก์ชั่น onewire_ans
3. เมื่อมีการตอบรับแล้ว จะส่งคำสั่ง Skip ROM ด้วยการเรียกใช้ฟังก์ชั่น onewire_write โดยค่าที่ส่งไปคือ 0xCC เพื่อข้ามการตรวจสอบรหัส CRC แล้วตามด้วยการส่งคำสั่งให้ DS1820 แปลงค่าอุณหภูมิมาเก็บไว้ที่หน่วยความจำสแครตช์แพดภายในตัว DS1820 ด้วยการเรียกใช้ฟังก์ชั่น onewire_write โดยค่าที่ส่งไปคือ 0x44
4. หลังจากนั้น DS1820 ต้องใช้เวลาในการแปลงค่าของอุณหภูมิ ดังนั้นจึงต้องเรียกฟังก์ชั่นหน่วงเวลาเพื่อรอ DS1820 แปลงค่าอุณหภูมิ ซึ่งไม่ควรน้อยกว่า 200 มิลลิวินาที
5. หลังจากหน่วงเวลาแล้ว จะเริ่มติดต่อใหม่อีกครั้ง ด้วยการส่งสถานะรีเซตและรอการตอบรับ
6. หลังจาก DS1820 ตอบรับแล้ว ไมโครคอนโทรลเลอร์ MCS-51 ส่งคำสั่ง Skip ROM อีกครั้ง
7. ส่งคำสั่งอ่านค่าอุณหภูมิจาก DS1820 ด้วยการเรียกใช้ฟังก์ชั่น onewire_write โดยค่าที่ส่งไปคือ 0xBE
8. เรียกใช้ฟังก์ชั่น onewire_read จะคืนค่าข้อมูลเก็บไว้ที่ตัวแปร temp ซึ่งต้องนำมาวิเคราะห์อีกครั้ง

   8.1 รูปแบบข้อมูลอุณหภูมิ 8 บิตของ DS1820 เป็นดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
T6	T5	T4	T3	T2	T1	T0	x

รูปที่ 3.19 แสดงบิตข้อมูลของไอซี DS1820

โดยที่

X เป็นบิตเลือกการแสดงทศนิยม

"0" แสดงค่าทศนิยม 0 องศาเซลเซียส

"1" แสดงค่าทศนิยม 0.5 องศาเซลเซียส

T6-T0 ทำงานร่วมกันเป็นข้อมูลอุณหภูมิจำนวนเต็มขนาด 7 บิต

8.2 ตัวอย่างการวิเคราะห์ข้อมูล

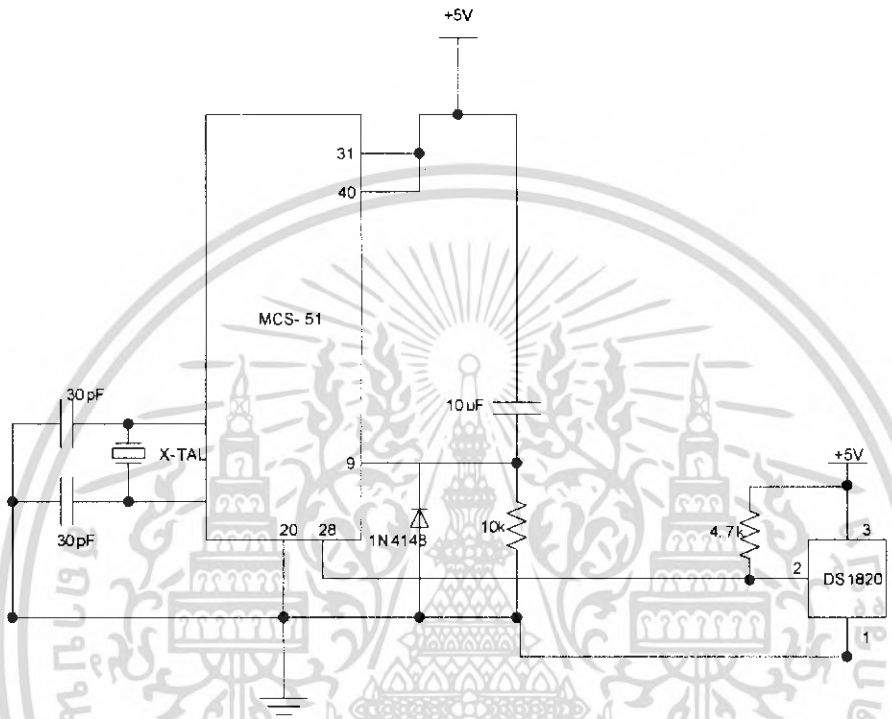
ถ้าค่าของตัวแปร temp ที่ได้จาก DS1820 คือ 0x49 แปลงเป็นเลขฐานสองได้ 01001001 ดังนั้น

X คือ "1" หมายถึง ให้แสดงความละเอียด 0.5 องศาเซลเซียสด้วย

T6-T0 คือ 00100100 หรือ 0x24 หรือ 36 ฐานสิบ

จึงได้ค่าอุณหภูมิออกมาเป็น 36.5 องศาเซลเซียส

9. หลังจรากที่ได้รับข้อมูลอุณหภูมิจาก DS1820 มาเก็บที่ตัวแปร temp แล้ว จะนำบิตล่างสุดของข้อมูลไปตรวจสอบเพื่อเลือกแสดงค่าทศนิยมเป็น 0 หรือ .5 องศาเซลเซียส หลังจากนั้นเลื่อนข้อมูลไปทางขวา 1 ครั้ง เพื่อให้ได้ค่าอุณหภูมิจำนวนเต็ม เมื่อนำข้อมูลมาประกอบกันก็จะเป็นค่าอุณหภูมิที่อ่านได้



รูปที่ 3.20 วงจรอ่านค่าอุณหภูมิ

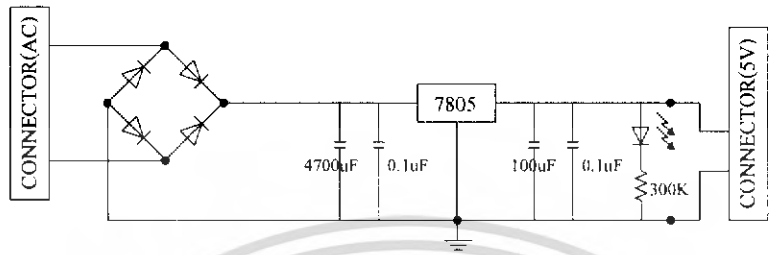
3.3.3 วงจรแปลงไฟฟ้า

ในการแปลงไฟฟ้า 220 V ac เป็น 5 V dc เพื่อนำไฟฟ้า 5 V dc ไปใช้ในวงจรอื่นๆ โดยใช้ไอซี 7805 ในการแปลงไฟฟ้า โดยมีคุณสมบัติดังนี้

ไอซี 7850 เป็นวงจรเรกกูเรเตอร์ที่ให้กระแสไฟฟ้าคงที่ ที่ออกแบบมาสำหรับการใช้งานที่หลากหลาย มี on-card regulation สำหรับการกำจัดสัญญาณรบกวน และ ปัญหาที่เกี่ยวข้องกับ single-point regulation เรกกูเรเตอร์แต่ละตัวสามารถให้กระแสมากถึง 1.5 A ได้ และภายในไอซีจะมี การจำกัดกระแสไฟ และหยุดการทำงานเมื่อความร้อนเกิดกำหนดเพื่อป้องกันการเกิดโอเวอร์โหลดซึ่งมีคุณสมบัติดังนี้

1. มีแรงดันอินพุท Vi 35 V
2. อุณหภูมิขณะที่ทำงาน Tj 150 องศาเซลเซียส
3. อุณหภูมิของสายนำไฟฟ้า 1,6 มิลลิเมตร (1/16 นิ้ว) จากกรณี 10 วินาที 260 องศาเซลเซียส

4. ช่วงของอุณหภูมิที่ใช้เก็บรักษา Tstg -65 ถึง 150 องศาเซลเซียส
5. แรงดันอินพุท Vi ต่ำสุด 7V สูงสุด 25V
6. กระแสเอาท์พุท Io สูงสุด 1.5A
7. อุณหภูมิขณะทำงานจริงที่ขาอุปกรณ์ สูงสุด 125 องศาเซลเซียส



รูปที่ 3.21 วงจรแปลงไฟฟ้าจาก 220 โวลต์เป็น 5 โวลต์

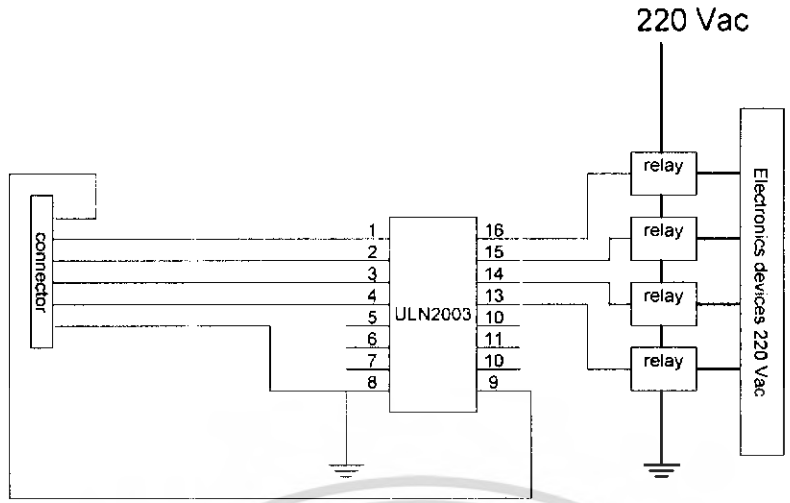
3.3.4 วงจรรีเลย์

วงจรรีเลย์นี้ใช้ไอซี ULN2003 ในการขับรีเลย์ หน้าสัมผัสเดี่ยวขนาด 5V หน้าสัมผัส 250 V ac 10 Amp มีคุณสมบัติดังนี้

1. มีกระแสเอาท์พุท 500 mA ต่อหนึ่ง ไดร์วเวอร์ (สูงสุด 600 mA)
2. มี ซับเพรสชั่น ไดโอด สำหรับโหลดไฟฟ้า
3. เอาท์พุทสามารถเทียบสำหรับ กระแสที่สูงกว่าได้
4. ต่ออินพุทกับ TTL/COMS.PMOS.DTL ได้

ULN2003 เป็นอุปกรณ์ที่ใช้แรงดันสูง และ กระแสสูง มีวงจร ดาลิงตัน อะเรย์(darlington arrays) แต่ละวงจรประกอบด้วย โอเพ่นคอเล็กเตอร์ดาลิงตันแพร์(open collector darlington pair)กับ คอมมอนอิมิเตอร์(common emitter) แต่ละช่อง รองรับกระแสได้ 500mA และสามารถทนได้สูงสุด 600mA

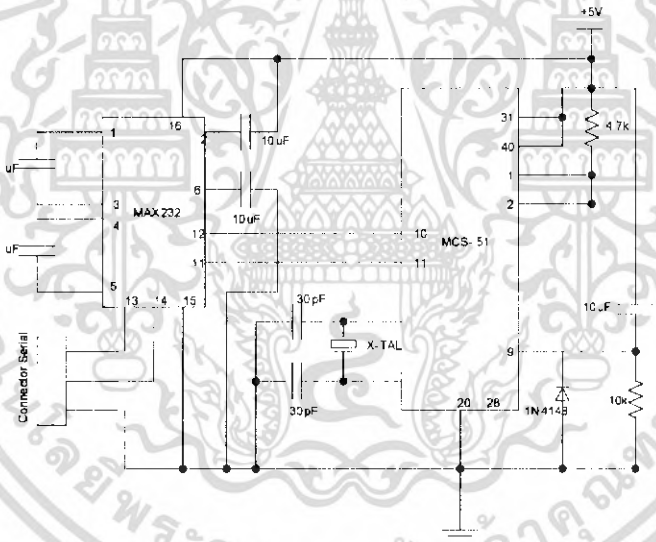
ไอซี ULN2003 นี้เป็นอุปกรณ์ที่สามารถต่อกับอุปกรณ์ไฟฟ้าได้อย่างกว้างขวางรวมถึง ขดลวดโซลลินอย (solenoids), รีเรย์ DC มอเตอร์ , จอแสดงผลLED และอื่นๆ



รูปที่ 3.22 วงจรขับรีเลย์

3.3.5วงจรเชื่อมต่อSerial port

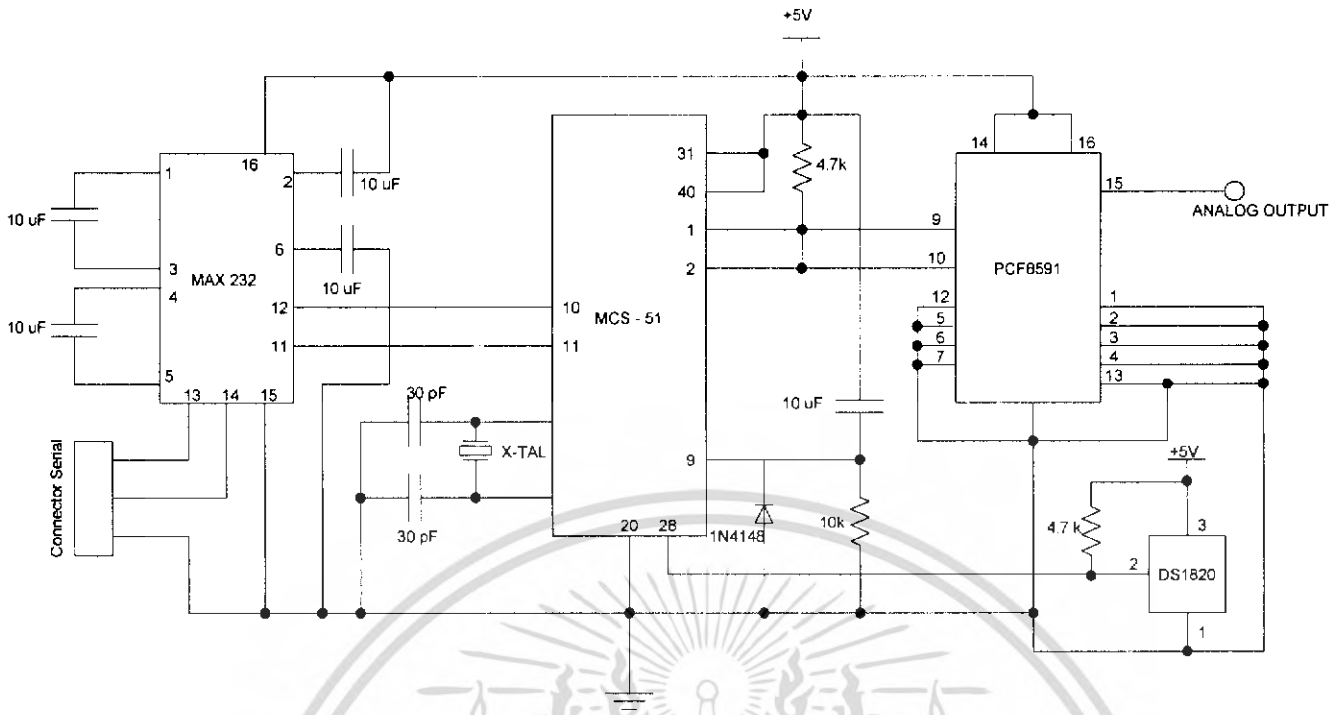
วงจรนี้ทำหน้าที่เชื่อมต่อระหว่างไมโครคอนโทรลเลอร์และบอร์ดอีเธอร์เนตไอโอโดยใช้ไอซี MAX 232



รูปที่ 3.23 วงจรเชื่อมต่อ Serial Port

3.3.6วงจรไมโครคอนโทรลเลอร์

วงจรวงจรไมโครคอนโทรลเลอร์นี้ใช้ไมโครคอนโทรลเลอร์เบอร์ 89C51RD2 ซึ่งเชื่อมต่ออยู่กับวงจรควบคุมอุณหภูมิของแอร์และวงจรอ่านค่าอุณหภูมิที่พอร์ท 1 และพอร์ท 2.8 ตามลำดับส่วนของการควบคุมนั้นใช้ภาษา C ในการควบคุมโดยควบคุมให้ตอบสนองต่ออินเตอร์รัปที่เข้าทาง Serial Port ผ่านวงจร MAX-232 เพื่อทำการปรับเปลี่ยนค่าตามอินเตอร์รับที่ได้รับเข้ามา



รูปที่ 3.24 วงจรรวม

บทที่ 4 การทดลองและผลการทดลอง

4.1. การทดลอง

4.1.1. การทดลองที่ 1 วัดค่าโวลเตจที่ออกการพอร์ทของบอร์ดและการส่งข้อมูลผ่าน Serial port

ขั้นตอนการทดลอง

1. เขียนโปรแกรมที่ใช้ในการเชื่อมต่อและสั่งการบอร์ด
2. เริ่มใช้โปรแกรมเชื่อมต่อกับบอร์ด
3. ทดลองการกำหนดค่าเอาพุตที่พอร์ตต่างๆ ให้เป็นลอจิกสูงหรือโลจิกต่ำ
    a. วัดค่าเอาพุตโวลเตจของพอร์ตและขานั้นๆ
    b. จดบันทึกผลการทดลอง
4. ทดลองการส่งข้อมูลผ่าน Serial port
    a. เปิด hyper terminal กำหนดให้ บอดเรตมีค่าเท่ากับ 9600 บิตข้อมูลมีค่าเท่ากับ 8 บิต หยุดให้มีค่าเท่ากับ 1 ไม่มีพาริตี้บิต และ ไม่มีโฟลวคอนโทรล
    b. ทำการส่งค่าที่กำหนดไว้ผ่าน Serial port สังเกตผลที่เกิดขึ้นใน hyper terminal
5. ทดลองการรับข้อมูลผ่าน Serial port
    a. เปิด hyper terminal กำหนดให้ บอดเรตมีค่าเท่ากับ 9600 บิตข้อมูลมีค่าเท่ากับ 8 บิต หยุดให้มีค่าเท่ากับ 1 ไม่มีพาริตี้บิต และ ไม่มีโฟลวคอนโทรล
    b. กดแป้นคีย์บอร์ดที่ hyper terminal สังเกตผลที่เกิดขึ้นในโปรแกรม

4.1.2. การทดลองที่ 2 ทดลองใช้บอร์ดรวมกับรีเลย์สวิตช์

ขั้นตอนการทดลอง

1. ต่อพอร์ตอินพุตเอาท์พุตของบอร์ดอีเทอร์เน็ตไอโอเข้ากับชุดของรีเลย์
2. ป้อนค่าให้อินพุตและเอาพุตพอร์ตของอีเทอร์เน็ตไอโอเปลี่ยนจากโลจิกต่ำเป็นโลจิกสูงและจากโลจิกสูงเป็นโลจิกต่ำ
3. สังเกตผลที่เกิดขึ้น

4.1.3. การทดลองที่ 3 การทดลองค่าเอาพุตของวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอก ของไอซี PCF8951

ขั้นตอนการทดลอง

1. โปรแกรมควบคุมไมโครคอนโทรลเลอร์ที่ใช้สำหรับเพิ่มค่าข้อมูลดิจิตอลเพื่อใช้ในการทดลอง
2. นำโวลต์มิเตอร์มาวัดที่เอาพุตของไอซี
3. จดบันทึกผลการทดลอง

4.1.4. การทดลองที่ 4 อ่านค่าอุณหภูมิจาก ไอซี DS1820

ขั้นตอนการทดลอง

1. โปรแกรมควบคุมไมโครคอนโทรลเลอร์ที่ใช้สำหรับเพิ่มค่าข้อมูลดิจิตอลเพื่อใช้ในการทดลอง
2. เปิด hyper terminal กำหนดให้ บอดเรตมีค่าเท่ากับ 9600 บิตข้อมูลมีค่าเท่ากับ 8 บิตหยุดให้มีค่าเท่ากับ 1 ไม่มีพาริตี้บิต และ ไม่มีโฟลวคอนโทรล

สังเกตผลที่ได้และเทียบกับอุณหภูมิจากเทอร์โมมิเตอร์จริง

4.1.5. การทดลองที่ 5 ทดลองโปรแกรมใช้งานจริง

ขั้นตอนการทดลอง

1. เปิดโปรแกรมและทำการเชื่อมต่อบอร์ดอีเทอร์เน็ตไอโอ
2. ทดสอบการปรับตั้งค่าต่างๆให้แก่บอร์ดอีเทอร์เน็ตไอโอ
3. ทดสอบการทำงานของสวิตช์รีเลย์
4. ทดสอบการทำงานของวงจรแปลงดิจิตอลเป็นสัญญาณอนาลอก
5. ทดสอบการทำงานของวงจรอ่านค่าอุณหภูมิ

4.2. ผลการทดลอง

4.2.1. ผลการทดลองที่ 1

เมื่อสั่งให้บอร์ดอีเทอร์เน็ตทำงานในโลจิกต่างๆจะได้ผลดังนี้

สภาวะ	โวลเตจจากเอาท์พุต(โวลต์)
โลจิกสูง	3.24
โลจิกต่ำ	0

ตารางที่ 4.1 ผลการทดลองที่ 1

4.2.2. ผลการทดลองที่ 2

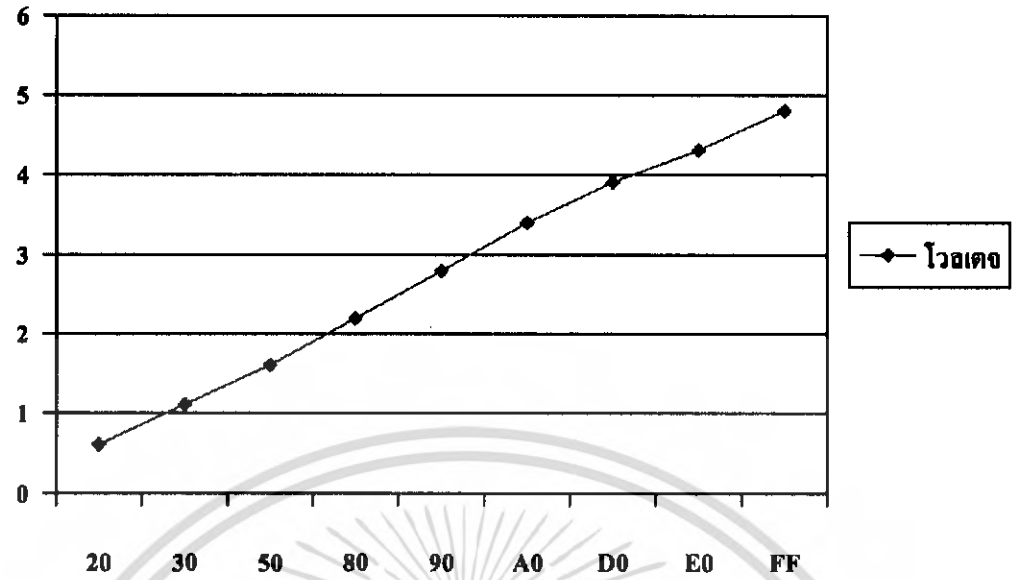
เมื่อสั่งบอร์ดอีเทอร์เน็ตไอโอทำให้สามารถควบคุมการปิดเปิดสวิตช์ได้ตามต้องการ

4.2.3. ผลการทดลองที่ 3

เมื่อใช้โปรแกรมสั่งให้เพิ่มค่าโดยการสั่งให้ไมโครคอนโทลเลอร์เพิ่มค่าดิจิตอลทีละ 16 แล้วจดบันทึกโวลเตจที่ได้พบว่าเมื่อดิจิตอลให้มีค่ามากขึ้นพบว่าโวลเตจที่ออกจากเอาท์พุตมีค่าเพิ่มขึ้น

สัญญาณดิจิตอล(ฐานสิบหก)	สัญญาณเอาท์พุต(โวลต์)
20	0.6
30	1.1
50	1.6
80	2.2
90	2.8
A0	3.4
D0	3.9
E0	4.3
FF	4.8

ตารางที่ 4.2 ผลการทดลองที่ 3

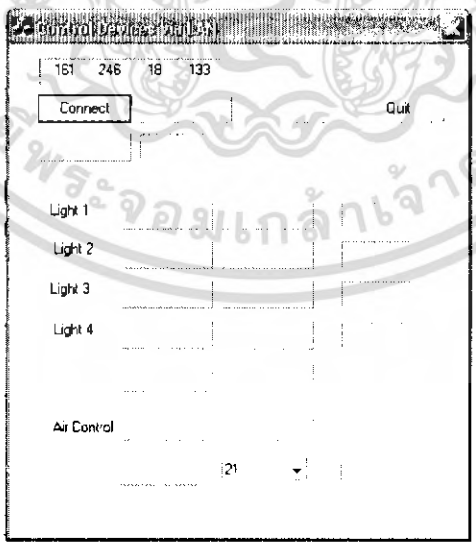


รูปที่ 4.1 แสดงกราฟการความสัมพันธ์ระหว่างค่าดิจิตอลและโวลเตจ

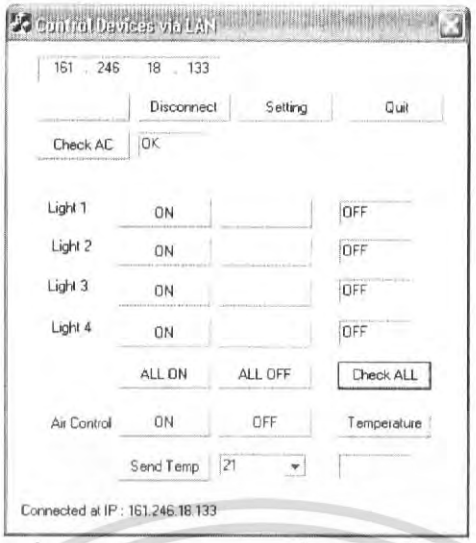
4.2.4. ผลการทดลองที่ 4

เมื่อใช้ hyper terminal อ่านค่าจะพบว่า ค่าของอุณหภูมิที่อ่านได้นั้นมีค่าแตกต่างจากอุณหภูมิจริง อยู่ในช่วง 0.5 องศาเซลเซียส ถึง 1 องศาเซลเซียส

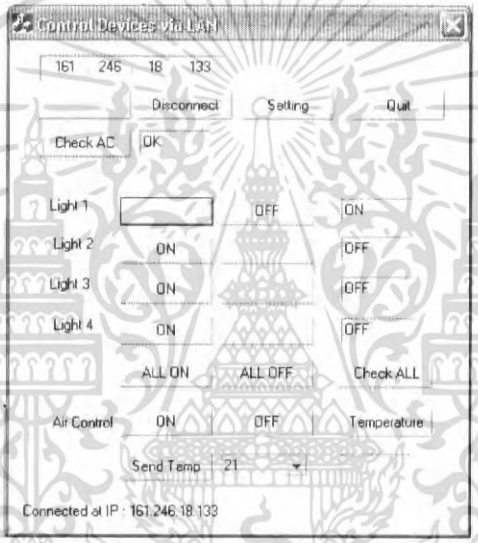
4.2.5. ผลการทดลองที่ 5



รูปที่ 4.2 หน้าจอโปรแกรมเริ่มใช้งาน

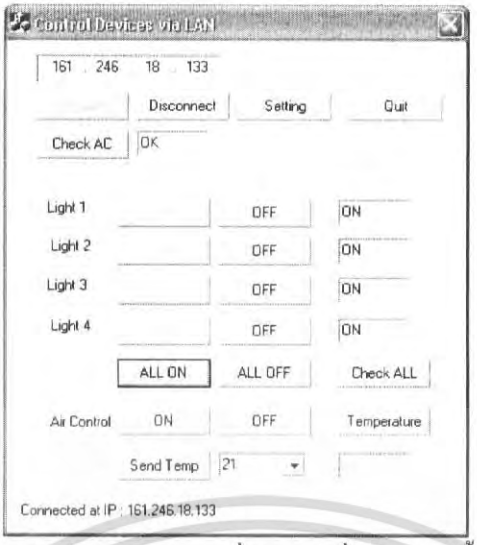


รูปที่ 4.3 หน้าจอโปรแกรมเมื่อทำการเชื่อมต่อ



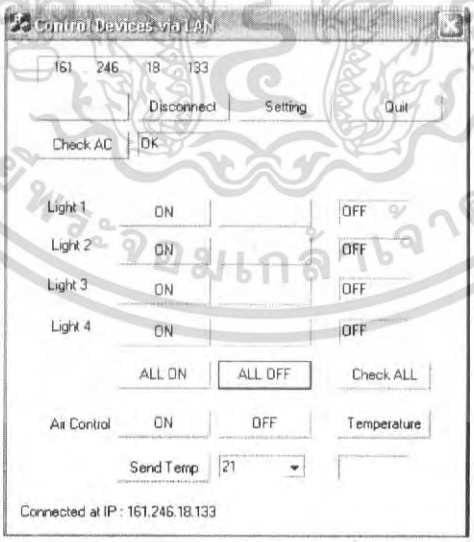
รูปที่ 4.4 หน้าจอโปรแกรมเมื่อทำการเชื่อมต่อ

รูปที่ 4.5 ผลที่เกิดขึ้นเมื่อทำการเปลี่ยนการทำงานของรีเลย์

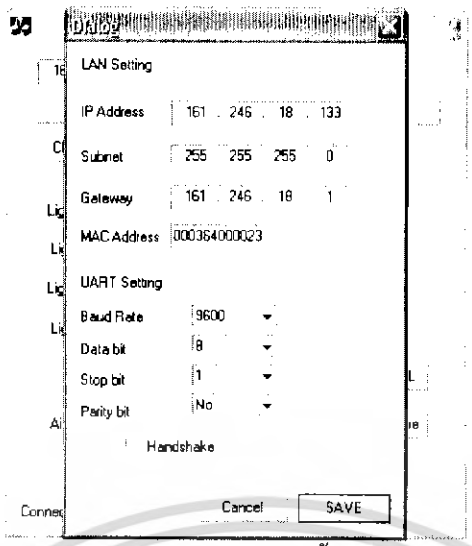


รูปที่ 4.6 หน้าจอโปรแกรมเมื่อทำการสั่งเปิดรีเลย์ทั้งหมด

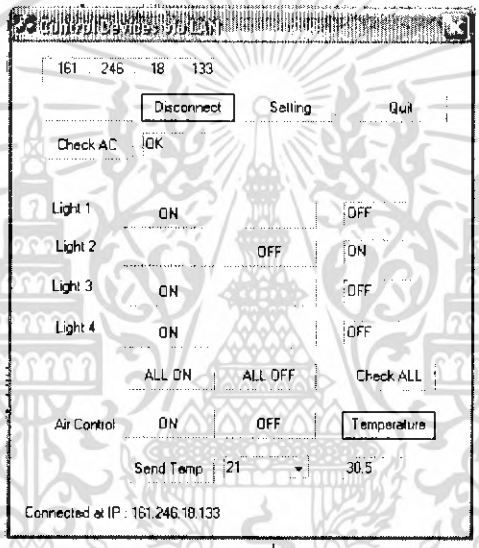
รูปที่ 4.7 เมื่อทำการสั่งเปิดรีเลย์ทั้งหมด



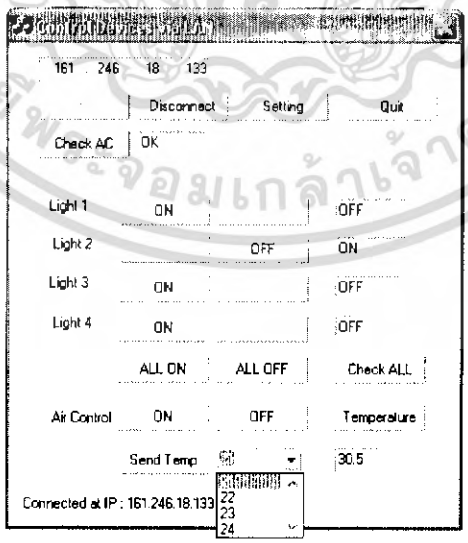
รูปที่ 4.8 หน้าจอโปรแกรมเมื่อทำการสั่งปิดรีเลย์ทั้งหมด



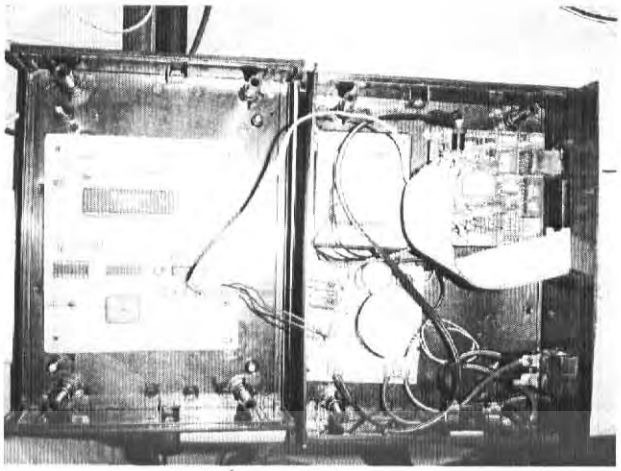
รูปที่ 4.9 หน้าจอโปรแกรมใช้สำหรับการตั้งค่าบอร์ดอีเทอร์เน็ตไอโอ



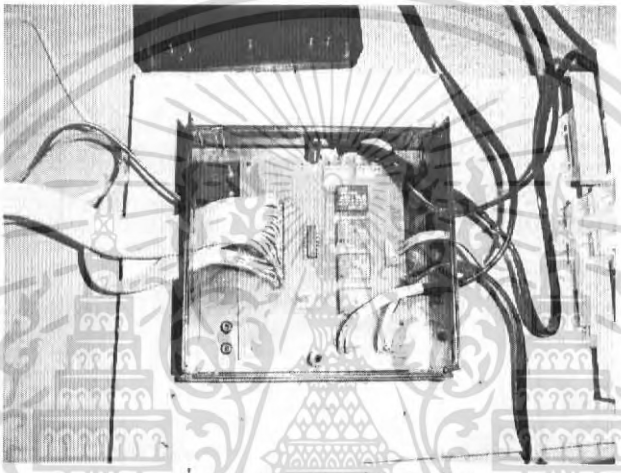
รูปที่ 4.10 หน้าจอโปรแกรมเมื่อทำการอ่านค่าอุณหภูมิ



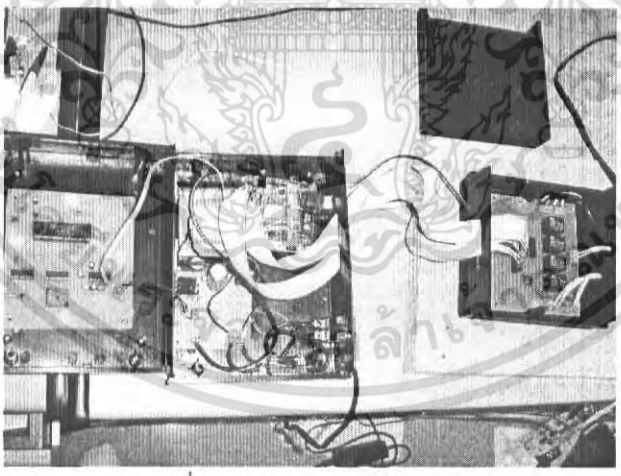
รูปที่ 4.11 หน้าจอโปรแกรมแสดงการปรับค่าเอาท์พุตของ
วงจรแปลงสัญญาณดิจิตอลเป็นอนาลอก



รูปที่ 4.12 วงจรควบคุม



รูปที่ 4.13 วงจรรีเลย์เปิดปิดไฟ



รูปที่ 4.14 วงจรรวมใช้งานจริง

บทที่ 5 สรุปผลการทดลอง

5.1. สรุปผลการทดลอง

การทำงานของบอร์ดอีเธอร์เนตไอโอและไมโครคอนโทรลเลอร์เป็นไปตามที่ผู้ใช้สั่งงาน โดยสามารถควบคุมอุปกรณ์ไฟฟ้า ควบคุมแบบจำลองการควบคุมอุณหภูมิแอร์ และส่งข้อมูลอุณหภูมิของสภาพแวดล้อมกลับมายังคอมพิวเตอร์ได้

5.2. วิเคราะห์ผลการทดลอง

5.2.1. ปัญหาที่เกิดขึ้น

1. โปรแกรมที่เขียนขึ้นมาควบคุมนั้นมีความผิดพลาดเกิดขึ้นเนื่องจากทำการเขียนและทดลองไปด้วย
2. วงจรของบอร์ดอีเธอร์เนตไอโอมีโวลเตจเอาท์พุท 3.3 โวลต์แต่ต้องการใช้มากกว่านั้น
3. การทำงานของไมโครคอนโทรลเลอร์ไม่ราบลื่นเกิดการติดขัดในบางครั้ง

5.2.2. แนวทางแก้ไข

1. การเขียนโปรแกรมควบคุมควรเขียนให้เสร็จสมบูรณ์ก่อนนำไปใช้งานจึงทำให้ความผิดพลาดนั้นไม่เกิดขึ้น
2. ใช้วงจรเพิ่มเติมในการนำเอาท์พุทของบอร์ดอีเธอร์เนตไอโอมาใช้งาน
3. ทำการแก้โปรแกรมในส่วนของไมโครคอนโทรลเลอร์ให้มีความสมบูรณ์และสอดคล้องกับโปรแกรมที่ใช้งานควบคุมบอร์ดอีเธอร์เนตไอโอให้มากขึ้น

5.3. ข้อเสนอแนะ

1. ปรับปรุงวงจรให้มีขนาดเล็กลง
2. ปรับปรุงโปรแกรมการใช้งานให้ใช้งานง่ายและมีรูปแบบที่สวยงาม
3. เขียนโปรแกรม Web Application ในการควบคุมผ่านอินเทอร์เนต
4. ทำการพัฒนาบอร์ดอีเธอร์เนตไอโอขึ้นมาใช้งานเอง

ภาคผนวก

โปรแกรมควบคุมไมโครคอนโทรลเลอร์

header.h

```
#define PCF8591_ID 0x90
sbit onewire=P2^7;
sbit PCF=P1^2;
unsigned char key=0;
unsigned char temp =0,temp_l=0,temp_h=0,half=0;
unsigned int i;
bit ans;
void ds1820_reset()
{
      onewire=0;
      for (i=0;i<100;i++)
      _nop_();

      onewire=0;
      for(i=0;i<2;i++)
      _nop_();
}
void ds81820 ans()
{
      ans=0;
      while(onewire);
      while(~onewire);
      for(i=0;i<2;i++);
      _nop_();
}
bit readbit(void)
{
      bit dat;
      onewire=0;
      _nop_();
      _nop_();
      onewire=0;
      _nop_();
      _nop_();
      _nop_();
      _nop_();
      dat=onewire;
      for(i=0;i<8;i++)
      _nop_();
      return dat;
}
unsigned char ds1820_read()
{
      unsigned char i,j,dat;
      dat=0;
      for(i=0;i<8;i++)
      {
            j=readbit();
            dat=(j<<7)|(dat>>1);
      }
      return dat;
}
void ds1820_write(unsigned char com)
{
      unsigned char j;
```

```
        bit send;
        for(j=0;j<8;j++)
        {
                send=com&0x01;
                com=com>>1;
                if(send)
                {
                        onewire=0;
                        _nop_();
                        _nop_();
                        _nop_();
                        _nop_();
                        onewire=1;
                        for(i=0;i<8;i++)
                        _nop_();
                }
                else
                {
                        onewire=0;
                        for(i=0;i<8;i++)
                        _nop_();
                        onewire=1;
                        _nop_();
                        _nop_();
                        _nop_();
                        _nop_();
                }
        }
}
void delay_convert(unsigned int count)
{
        do
        { count --;}
        while (count>0);
}
void tempread()
{
                ds1820_reset();
                ds1820_ans();
                ds1820_write(0xCC);
                ds1820_write(0x44);
                delay_convert(200);
                ds1820_reset();
                ds1820_ans();
                ds1820_write(0xCC);
                ds1820_write(0xBE);
                temp=ds1820_read();
                half=temp & 0x01;
                if(half)
                half=0x35;
                else half=0x30;
                temp=temp>>1;
                temp_h=(temp/10)|0x30;
                temp_l=(temp%10)|0x30;
        SBUF=temp_h;
        while(~TI);
```

```
        TI=0;
        SBUF=temp_l;
        while(~TI);
        TI=0;
        SBUF='.';
        while(~TI);
        TI=0;
        SBUF=half;
        while(~TI);
        TI=0;
        SBUF=' ';
        while(~TI);
        TI=0;
    key=0;
}
void airtemp(unsigned char value)
{
            i2c_start();
            i2c_write(PCF8591_ID);
            i2c_write(0x40);
            i2c_write(value);
            i2c_stop();
}
i2c.h
sbit SDA  = P1^0;
sbit SCL  = P1^1;
void  i2c_wait(void)
{
      unsigned  char  i;
      for(i=0;i<30;i++) { }
}
void  i2c_clk(void)
{
      SCL   =     1;
      i2c_wait();
      SCL   =     0;
      i2c_wait();
}
void  i2c_Ack(void)
{
      SDA   = 0;
      i2c_clk();
}
void  i2c_Nack(void)
{
      SDA   =  1;
      i2c_clk();
}
void  i2c_start(void)
{
      SCL   =     0;
      SDA   =     1;
      SCL   =     1;
      i2c_wait();
      SDA   =     0;
      i2c_wait();
}
```

```
void  i2c_stop(void)
{
      SCL   =     0;
      SDA   =     0;
      SCL   =     1;
      i2c_wait();
      SDA   =     1;
      i2c_wait();
}
void  i2c_write(unsigned char Data )
{
      unsigned char  i;
      bit   out;
      SCL   =     0;
   for (i=0;i<8;i++)
      {
         out      =     Data & 0x80;
         Data     =     Data << 1;
         SDA      =     out;
         i2c_clk();
      }
   SDA      =     1;
   i2c_clk();
}
unsigned  char  i2c_read(void)
{
      unsigned char     Data,i;
   bit     out;
      SCL   =     0;
      SDA   =     1;
      i2c_wait();
      for (i=0;i<8;i++)
   {
            SCL   =     1;
            i2c_wait();
            out   =     SDA;
            Data  =     Data << 1;
            Data  =     Data |  out;
            SCL   =     0;
            i2c_wait();
       }
    return(Data);
}
main.c
#include <reg51rx.h>
#include <intrins.h>
#include <i2c.h>
#include <header.h>
void service_serial() interrupt 4
{
      if (RI)
      {
            RI=0;
            key=SBUF;
      }
      if (TI)
      {
```

```
            TI=0;
        }
}
void main()
{
        TMOD=0x21;
        SCON=0x50;
        TH1=0xFA;
        TL1=0xFA;
        EA=1;
        ES=1;
        REN=1;
        TI=0;
        RI=0;
        TR1=1;
        while(1)
        {
            switch(key)
            {
                case 'A' : {PCF=1;key=0;}break;
                case 'B' : {PCF=0;key=0;}break;
                case 'C' : {airtemp(0x36);key=0;}break;
                case 'D' : {airtemp(0x50);key=0;}break;
                case 'E' : {airtemp(0x6B);key=0;}break;
                case 'F' : {airtemp(0x80);key=0;}break;
                case 'G' : {airtemp(0x9D);key=0;}break;
                case 'I' : {airtemp(0xB3);key=0;}break;
                case 'J' : {airtemp(0xD3);key=0;}break;
                case 'K' : {airtemp(0xE6);key=0;}break;
                case 'L' : {airtemp(0xFF);key=0;}break;
                case 'H' : {tempread();key=0;}break;
                default:break;
            }
    }
}
```

โปรแกรมควบคุมอีเธอร์เนตไอโอ

```
// boardlink.h:
////////////////////////////////////////////////////////////////////
#if
!defined(AFX_BOARDLINK_H__8A39DB12_9569_488F_8836_85D80E062168__INCL
UDED_)
#define
AFX_BOARDLINK_H__8A39DB12_9569_488F_8836_85D80E062168__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "EtherIOComm.h"
#define WM_UPDATE_CONNECTION  WM_USER+0x1234
#define WM_ON_DATA_RECEIVE          WM_USER+0x1235
#define FILE_SEND       0
#define TEXT_SEND       1
class Cboardlink : public CEtherIOComm
{
public:
      Cboardlink();
      virtual ~Cboardlink();
      void SetMessageWindow(CEdit* pMsgCtrl);
      void SetStatusMsg(CString *pStatusMsg);
      void AppendMessage(LPCTSTR strText );
      virtual void OnDataReceived(const LPBYTE lpBuffer, DWORD
dwCount);
      virtual void OnEvent(UINT uEvent);
protected:
      void DisplayData(const LPBYTE lpData, DWORD dwCount, const
SockAddrIn& sfrom);

      CEdit* m_pMsgCtrl;
      CString *m_StatusMsg;
      CString     m_ReceiveFileName;
      BOOL  m_SaveReceiveToFile;
};
#endif //
!defined(AFX_BOARDLINK_H__FCF4832C_AB59_43F5_8099_6569E7623D56__INCL
UDED_)

// boardlink.cpp
////////////////////////////////////////////////////////////////////
/
#include "stdafx.h"
#include "control.h"
#include "boardlink.h"
#include <atlconv.h>
#include "resource.h"
#ifdef _DEBUG

#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
// Construction/Destruction
Cboardlink::Cboardlink()
{
```

```
	m_pMsgCtrl = NULL;
	m_StatusMsg = NULL;
}
Cboardlink::~Cboardlink()
{

}
void Cboardlink::DisplayData(const LPBYTE lpData, DWORD dwCount,
const SockAddrIn& sfrom)
{
	CString strData;
	memcpy(strData.GetBuffer(dwCount), A2CT ((LPSTR)lpData),
dwCount);
	strData.ReleaseBuffer();
	if (!sfrom.IsNull())
	{
		LONG  uAddr = sfrom.GetIPAddr();
		BYTE* sAddr = (BYTE*) &uAddr;
		// show port in host format
		short nPort = ntohs( sfrom.GetPort() );
		CString strAddr;
		// Address  store in network format
		strAddr.Format(_T("%u.%u.%u.%u (%d)>"),
				(UINT)(sAddr[0]), (UINT)(sAddr[1]),
				(UINT)(sAddr[2]), (UINT)(sAddr[3]),
nPort);
		strData = strAddr + strData;
	}
	AppendMessage( strData );
}
void Cboardlink::AppendMessage(LPCTSTR strText )
{
	if (NULL == m_pMsgCtrl)
		return;
	if (::IsWindow( m_pMsgCtrl->GetSafeHwnd() ))
	{
		int nLen = m_pMsgCtrl->GetWindowTextLength();
		if(nLen >= (m_pMsgCtrl->GetLimitText()-10)){
			m_pMsgCtrl->SetSel(0, -1);
			m_pMsgCtrl->ReplaceSel( " " );
			TRACE("Clear receive edit\n");
			return;
		}
		m_pMsgCtrl->SetSel(nLen, nLen);
		m_pMsgCtrl->ReplaceSel( strText );
	}
}
void Cboardlink::SetMessageWindow(CEdit* pMsgCtrl)
{
	m_pMsgCtrl = pMsgCtrl;
}
void Cboardlink::SetStatusMsg(CString *pStatusMsg)
{
	m_StatusMsg = pStatusMsg;
}
void Cboardlink::OnDataReceived(const LPBYTE lpBuffer, DWORD
dwCount)
```

```
{
	static UINT byreceive= 0;
	byreceive += dwCount;
	TRACE("Data receive= %d\n!",byreceive);

	SockAddrIn saddr_in;
	LPBYTE lpData = lpBuffer;
	// Write it to file
	if(m_SaveReceiveToFile==TRUE)
	{
		CFile Rfile(m_ReceiveFileName ,
		CFile::modeCreate | CFile::modeWrite |
CFile::modeNoTruncate);
		Rfile.SeekToEnd();
		Rfile.Write( lpBuffer,dwCount);
		Rfile.Close();
	}
	else
	{
		// Display data to message list
		DisplayData( lpData, dwCount, saddr_in );
	}
	return;
}
void Cboardlink::OnEvent(UINT uEvent)
{
	TRACE("ONEVENT\n");

	if (NULL == m_pMsgCtrl)
		return;
	CWnd* pParent = m_pMsgCtrl->GetParent();
	if (!::IsWindow( pParent->GetSafeHwnd()))
		return;
	if(NULL == m_StatusMsg)
		return;
	switch( uEvent )
	{
		case EVT_CONSUCCESS:
			m_StatusMsg->Format("CONNECTION_ESTABLISHED");
			break;
		case EVT_CONFAILURE:
			m_StatusMsg->Format("CONNECTION_FAILED");
			break;
		case EVT_CONDROP:
			m_StatusMsg->Format("CONN_ABADON");
			break;
		case EVT_ZEROLENGTH:
			m_StatusMsg->Format("ZERO_LENGTH_MSG");
			break;
		default:
			m_StatusMsg->Format("UNKNOW_ERROR");
			break;
	}
	AfxGetMainWnd()->PostMessage( WM_UPDATE_CONNECTION, uEvent,
(LPARAM) this);
}
```

```
//boardsetting.h : header file
////////////////////////////////////////////////////////////////////////
/
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class Cboardsetting : public CDialog
{
//Construction
public:
      Cboardsetting(CWnd* pParent = NULL);   // standard constructor
      BOOL              m_EnableHWHandshak;
      CComboBox         m_BuadRateCombo;
      CComboBox         m_ParityBitCombo;
      CComboBox         m_StopBitCombo;
      CComboBox         m_DataBitCombo;
      BYTE              m_DataBits;
      BYTE              m_StopBits;
      BYTE              m_ParityBits;
      DWORD             m_BuadRate;
      BYTE              m_IP[4];
      BYTE              m_Subnet[4];
      BYTE              m_Gateway[4];
      BYTE              m_Mac[6];
//Dialog Data
      //{{AFX_DATA(Cboardsetting)
      enum { IDD = IDD_BOARDSETTING_DIALOG };
      CIPAddressCtrl    m_IPSubnet;
      CIPAddressCtrl    m_IPGateway;
      CIPAddressCtrl    m_IPconfig;
      CString     m_Macaddr;
      //}}AFX_DATA
//Overrides
      //ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(Cboardsetting)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL
//Implementation
protected:

      //Generated message map functions
      //{{AFX_MSG(Cboardsetting)
      virtual void OnOK();
      virtual void OnCancel();
      virtual BOOL OnInitDialog();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
//Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_BOARDSETTING_H__A242B966_49E2_4A51_870D_E5E9F78E7457__INCLUDED_)
```

```
//boardsetting.cpp
////////////////////////////////////////////////////////////////////////
/

#include "stdafx.h"
#include "control.h"
#include "boardsetting.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//Cboardsetting dialog
Cboardsetting::Cboardsetting(CWnd* pParent /*=NULL*/)
      :CDialog(Cboardsetting::IDD, pParent)
{
      //{{AFX_DATA_INIT(Cboardsetting)
            //NOTE: the ClassWizard will add member initialization
here
      //}}AFX_DATA_INIT
}
void Cboardsetting::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(Cboardsetting)
      DDX_Control(pDX, IDC_IPSUBNET, m_IPSubnet);
      DDX_Control(pDX, IDC_IPGATEWAY, m_IPGateway);
      DDX_Control(pDX, IDC_IPCONFIG, m_IPconfig);
      DDX_Text(pDX, IDC_MACADDRESS, m_Macaddr);
      DDX_Check(pDX, IDC_CHECK, m_EnableHWHandshak);
            //NOTE: the ClassWizard will add DDX and DDV calls here
      //}}AFX_DATA_MAP
      DDX_Control(pDX, IDC_BAUDRATE_COMBO, m_BuadRateCombo);
      DDX_Control(pDX, IDC_DATABIT_COMBO, m_DataBitCombo);
      DDX_Control(pDX, IDC_STOPBIT_COMBO, m_StopBitCombo);
      DDX_Control(pDX, IDC_PARITY_COMBO, m_ParityBitCombo);
}
BEGIN_MESSAGE_MAP(Cboardsetting, CDialog)
      //{{AFX_MSG_MAP(Cboardsetting)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////
//boardsetting message handlers

BOOL Cboardsetting::OnInitDialog()
{
      CDialog::OnInitDialog();
      UpdateData(FALSE);
      int i=0;
      int index=0;
      //Set ComboBox
      CString ComboBoxMsg;
      for(i = IDS_BUADRATE_OPT1; i<= IDS_BUADRATE_OPT15; i++)
      {
```

```
        ComboBoxMsg.LoadString(i);
        m_BuadRateCombo.AddString( ComboBoxMsg);
}
for( i= IDS_DATABIT_OPT1; i<= IDS_DATABIT_OPT2; i++)
{
        ComboBoxMsg.LoadString(i);
        m_DataBitCombo.AddString(ComboBoxMsg);
}
for( i = IDS_STOPBIT_OPT1; i<= IDS_STOPBIT_OPT2; i++)
{
        ComboBoxMsg.LoadString(i);
        m_StopBitCombo.AddString( ComboBoxMsg);
}
for( i = IDS_PARITY_OPT1; i<= IDS_PARITY_OPT3; i++)
{
        ComboBoxMsg.LoadString(i);
        m_ParityBitCombo.AddString( ComboBoxMsg);
}
{//Load old config from Default setting
        m_DataBitCombo.SetCurSel(0);
        CString ComboBoxMsg;
        //Data Bit
        index = 0;
        for( i= IDS_DATABIT_OPT1; i<= IDS_DATABIT_OPT2; i++)
        {
                ComboBoxMsg.LoadString(i);
                if ( m_DataBits ==atoi(ComboBoxMsg ))
                {
                        m_DataBitCombo.SetCurSel(index);
                }
                index++;
        }
        //Stop bit
        index = 0;
        for( i= IDS_STOPBIT_OPT1; i<= IDS_STOPBIT_OPT2; i++)
        {
                ComboBoxMsg.LoadString(i);
                if ( m_StopBits == atoi(ComboBoxMsg ))
                {
                        m_StopBitCombo.SetCurSel(index);
                }
                index++;
        }
        //Parity bit no->0, odd->1, even->2
        m_ParityBitCombo.SetCurSel(m_ParityBits );
        //Baud rate
        index = 0;
        for( i= IDS_BUADRATE_OPT1; i<= IDS_BUADRATE_OPT15; i++)
        {
                ComboBoxMsg.LoadString(i);
                if ( m_BuadRate == (DWORD)atoi(ComboBoxMsg ) )
                {
                        m_BuadRateCombo.SetCurSel(index);
                }
                index++;
        }
        m_EnableHWHandshak ;
```

```
            UpdateData(FALSE);
      //Load Network config

            UpdateData(FALSE);
            m_IPconfig.SetAddress(m_IP[0],m_IP[1],m_IP[2],m_IP[3]);

      m_IPSubnet.SetAddress(m_Subnet[0],m_Subnet[1],m_Subnet[2],m_Su
bnet[3]);
            m_IPGateway.SetAddress(m_Gateway[0],
m_Gateway[1],m_Gateway[2],m_Gateway[3] );
            m_Macaddr.Format("%02X%02X%02X%02X%02X%02X",
                                                  m_Mac[0],
                                                  m_Mac[1],
                                                  m_Mac[2],
                                                  m_Mac[3],
                                                  m_Mac[4],
                                                  m_Mac[5]);
            UpdateData(FALSE);
      }

            return TRUE;  // return TRUE  unless you set the focus
to a control
}
int get_hex_value(char* buf,char* str,int len)
{
   int i;
   char c,val;
   for(i=0;i<len;i++){
        if( !(c = tolower(*str++))) return -1;
        if(isdigit(c)) val = c-'0';
        else if(c >= 'a' && c <= 'f') val = c-'a'+10;
        else return -1;
  *buf = val << 4;
        if( !(c = tolower(*str++))) return -1;
        if(isdigit(c)) val = c-'0';
        else if(c >= 'a' && c <= 'f') val = c-'a'+10;
        else return -1;
  *buf++ |= val;
        if( (*str == ':')||(*str == ',')||(*str == '.'))
            str++;
        }
   return 0;
}
void Cboardsetting::OnOK()
{
      UpdateData(TRUE);
      {
            int i=0;
            int index=0;
            UpdateData(FALSE);
            CString DataBits, StopBit, Parity, BaudRate;
            m_DataBitCombo.GetLBText( m_DataBitCombo.GetCurSel(),
DataBits);
            m_StopBitCombo.GetLBText( m_StopBitCombo.GetCurSel(),
StopBit  );
```

```
            m_BuadRateCombo.GetLBText( m_BuadRateCombo.GetCurSel(),
BaudRate  );
            //Data Bit
            m_DataBits = atoi(DataBits );
            //Stop bit
            m_StopBits = atoi(StopBit);
            //Parity bit no->0, odd->1, even->2
            m_ParityBits = m_ParityBitCombo.GetCurSel();
            //Baud rate
            m_BuadRate =    (DWORD)atoi(BaudRate) ;
            m_EnableHWHandshak ;
      //save Network config
            //IP address
            BYTE IP1,IP2,IP3,IP4;
            m_IPconfig.GetAddress(IP1,IP2,IP3,IP4);
            m_IP[0]=IP1;
            m_IP[1]=IP2;
            m_IP[2]=IP3;
            m_IP[3]=IP4;
            //Sub address
            m_IPSubnet.GetAddress(IP1,IP2,IP3,IP4);
            m_Subnet[0]=IP1;
            m_Subnet[1]=IP2;
            m_Subnet[2]=IP3;
            m_Subnet[3]=IP4;
            //Gateway
            m_IPGateway.GetAddress(IP1,IP2,IP3,IP4);
            m_Gateway[0]=IP1;
            m_Gateway[1]=IP2;
            m_Gateway[2]=IP3;
            m_Gateway[3]=IP4;
            BYTE MacAdd[6];
            LPTSTR p = m_Macaddr.GetBuffer(256);
            if(get_hex_value( (char*)&MacAdd[0], p , 6)==(-1)){
                  AfxMessageBox("Error Mac address value!");
                  CDialog::OnCancel();
            }
            m_Mac[0]=MacAdd[0];
            m_Mac[1]=MacAdd[1];
            m_Mac[2]=MacAdd[2];
            m_Mac[3]=MacAdd[3];
            m_Mac[4]=MacAdd[4];
            m_Mac[5]=MacAdd[5];
            UpdateData(TRUE);
      }
      CDialog::OnOK();
}
void Cboardsetting::OnCancel()
{
      CDialog::OnCancel();
}

//FILE :            CEtherIOComm.h
```

```
//Author            Ernest Laurentin
//////////////////////////////////////////////////
#ifndef _CEtherIOComm_H_
#define _CEtherIOComm_H_
#include <list>

#include <stdlib.h>
#if(_WIN32_WINNT >= 0x0400)
#include <winsock2.h>
#include <mswsock.h>
#else
#include <winsock.h>
#endif /* _WIN32_WINNT >=  0x0400*/
#pragma comment(lib, "ws2_32")
//Event value
#define EVT_CONSUCCESS          0x0000      //Connection established
#define EVT_CONFAILURE          0x0001      //General failure - Wait
Connection failed
#define EVT_CONDROP                     0x0002      //Connection dropped
#define EVT_ZEROLENGTH          0x0003      //Zero length message
#define BUFFER_SIZE     MAX_PATH
#define HOSTNAME_SIZE   MAX_PATH
#define STRING_LENGTH   40
#define MAX_HOSTNAME    256
#define MAX_HOSTADDR    40
//EtherIO HW define
#define DEFAULT_DATA_BIT        8
#define DEFAULT_STOP_BIT        1
#define DEFAULT_PARITY          0
#define DEFAULT_HW_HANDSHA      0
#define DEFAULT_BAUD_RATE       9600
#define MAX_PACKET_SZ           1024
#define PORT_A                  0x0020
#define PORT_B                  0x0024
#define PORT_C                  0x0028
#define PORT_D                  0x002C
#define PORT_E                  0x0030
#define PORT_F                  0x0034
#define PORT_G                  0x0038
#define     UART_SET_CMD        0x11
#define     SETBIT_CMD          0x12
#define     CLEARBIT_CMD        0x13
#define     OUT_PORT_CMD        0x14
#define     READ_PORT_CMD       0x15
#define     READ_ADC_CMD        0x16
#define READ_BIT_CMD            0x17
struct AFX_EXT_CLASS SockAddrIn {
public:
      SockAddrIn() { Clear(); }
      SockAddrIn(const SockAddrIn& sin) { Copy( sin ); }
      ~SockAddrIn() { }
      SockAddrIn& Copy(const SockAddrIn& sin);
      void  Clear() { memset(&sockAddrIn, 0, sizeof(sockAddrIn)); }
      bool  IsEqual(const SockAddrIn& sin);
      bool  IsGreater(const SockAddrIn& sin);
      bool  IsLower(const SockAddrIn& pm);
```

```
      bool  IsNull() const { return
((sockAddrIn.sin_addr.s_addr==0L)&&(sockAddrIn.sin_port==0)); }
      ULONG GetIPAddr() const { return sockAddrIn.sin_addr.s_addr; }
      short GetPort() const { return sockAddrIn.sin_port; }
      bool  CreateFrom(LPCTSTR sAddr, LPCTSTR sService);
      SockAddrIn& operator=(const SockAddrIn& sin) { return Copy(
sin ); }
      bool  operator==(const SockAddrIn& sin) { return IsEqual( sin
); }
      bool  operator!=(const SockAddrIn& sin) { return !IsEqual( sin
); }
      bool  operator<(const SockAddrIn& sin)  { return IsLower( sin
); }
      bool  operator>(const SockAddrIn& sin)  { return IsGreater(
sin ); }
      bool  operator<=(const SockAddrIn& sin) { return !IsGreater(
sin ); }
      bool  operator>=(const SockAddrIn& sin) { return !IsLower( sin
); }
      operator LPSOCKADDR() { return (LPSOCKADDR)(&sockAddrIn); }
      size_t      Size() const { return sizeof(sockAddrIn); }
      void  SetAddr(SOCKADDR_IN* psin) { memcpy(&sockAddrIn, psin,
Size()); }
      SOCKADDR_IN sockAddrIn;
};
typedef std::list<SockAddrIn> CSockAddrList;
class AFX_EXT_CLASS CEtherIOComm
{
public:
      CEtherIOComm();
      virtual ~CEtherIOComm();
      bool IsOpen() const;    // Is Socket valid?
      bool IsStart() const;   // Is Thread started?
      bool IsServer() const;  // Is running in server mode
      bool IsBroadcast() const; // Is UDP Broadcast active
      bool IsSmartAddressing() const;     // Is Smart Addressing mode
support
      SOCKET GetSocket() const;     // return socket handle
      void SetServerState(bool bServer);  // Run as server mode if
true
      void SetSmartAddressing(bool bSmartAddressing); // Set Smart
addressing mode
      bool GetSockName(SockAddrIn& saddr_in);   // Get Socket name -
address
      bool GetPeerName(SockAddrIn& saddr_in);   // Get Peer Socket
name - address
      void AddToList(const SockAddrIn& saddr_in);     // Add an
address to the list
      void RemoveFromList(const SockAddrIn& saddr_in);      // Remove
an address from the list
      void CloseComm();       // Close Socket
      bool WatchComm();       // Start Socket thread
      void StopComm();        // Stop Socket thread
//Create a Socket - Server side
      bool CreateSocket(LPCTSTR strServiceName, int nProtocol, int
nType, UINT uOptions = 0);
//Create a socket, connect to (Client side)
```

```
      bool ConnectTo(LPCTSTR strDestination);
//Event function - overwrite to get data
      virtual void OnDataReceived(const LPBYTE lpBuffer, DWORD
dwCount);
      virtual void OnEvent(UINT uEvent);
//Run function - overwrite to implement a new behaviour
      virtual void Run();
//Data function
      DWORD ReadComm(LPBYTE lpBuffer, DWORD dwSize, DWORD
dwTimeout);
      DWORD WriteComm(const LPBYTE lpBuffer, DWORD dwCount, DWORD
dwTimeout);
//Utility functions
      static SOCKET WaitForConnection(SOCKET sock); // Wait For a
new connection (Server side)
      static bool ShutdownConnection(SOCKET sock);  // Shutdown a
connection
      static USHORT GetPortNumber( LPCTSTR strServiceName );      //
Get service port number
      static ULONG GetIPAddress( LPCTSTR strHostName );     // Get IP
address of a host
      static bool GetLocalName(LPTSTR strName, UINT nSize); //
GetLocalName
      static bool GetLocalAddress(LPTSTR strAddress, UINT nSize); //
GetLocalAddress
//EtherIO IO port addition function
      BOOL SetUartOpt (unsigned char DataBit= DEFAULT_DATA_BIT,
                       unsigned char StopBit = DEFAULT_STOP_BIT,
                       unsigned char ParityBit = DEFAULT_PARITY,
                       BOOL HwHandshak = DEFAULT_HW_HANDSHA,
                       DWORD BaudRate      = DEFAULT_BAUD_RATE

                  );
      BOOL SetBit  ( unsigned char port,unsigned char pin);
      BOOL ClearBit(unsigned char port, unsigned char pin);
      BOOL OutPort (unsigned char port, BYTE SendByte);
      BOOL ReadBit (unsigned char port, unsigned char pin,unsigned
char *bit);
      BOOL ReadPort(unsigned char port, BYTE *ReadByte);
      BOOL ReadADC (unsigned unsigned char channel, unsigned short
*ADCValue);
      BOOL FifoWrite(unsigned char addr, const LPBYTE lpBuffer,
unsigned short dwCount );
      BOOL FifoRead( unsigned char addr, const LPBYTE lpBuffer,
unsigned short dwCount );
      BOOL ChangeIP( unsigned char *NewIP);
      BOOL ChangeMAC( unsigned char *NewIP);
      BOOL WRUserFlash(unsigned short addr , const LPBYTE lpBuffer,
unsigned short dwCount);
      BOOL RDUserFlash(unsigned short addr , const LPBYTE lpBuffer,
unsigned short dwCount);
      BOOL SystemSoftReset(void);
      BOOL LoadConfig(              unsigned char *DataBit,
                                    unsigned char *StopBit ,
                                    unsigned char *ParityBit ,
                                    BOOL *HwHandshak ,
                                    DWORD *BaudRate
```

```
                                        unsigned char *OldIP,
                                        unsigned char *SupnetIP,
                                        unsigned char *GatewayIP,
                                        unsigned char *MacAdd
                    );
      BOOL SaveConfig(  unsigned char DataBit= DEFAULT_DATA_BIT,
                        unsigned char StopBit = DEFAULT_STOP_BIT,
                         unsigned char ParityBit = DEFAULT_PARITY,
                         BOOL HwHandshak = DEFAULT_HW_HANDSHA,
                         DWORD BaudRate    = DEFAULT_BAUD_RATE,
                         unsigned char *OldIP =NULL,
                         unsigned char *SupnetIP = NULL,
                         unsigned char *GatewayIP = NULL,
                         unsigned char *MacAdd=NULL
                                                    );
//CEtherIOComm - data
protected:
      HANDLE            m_hComm;          // Serial Comm handle
      HANDLE            m_hThread;        // Thread Comm handle
      bool        m_bServer;          // Server mode (true)
      bool        m_bSmartAddressing;     // Smart Addressing mode
(true) - many listeners
      bool        m_bBroadcast;     // Broadcast mode
      CSockAddrList m_AddrList;     // Connection address list for
broadcast
      HANDLE            m_hMutex;         // Mutex object
//CEtherIOComm - function
protected:
      //Synchronization function
      void LockList();                    // Lock the object
      void UnlockList();                  // Unlock the object
      static UINT WINAPI SocketThreadProc(LPVOID pParam);
      //control socket
      SOCKET m_ControlSock;
      bool ConnectToControl(LPCTSTR strDestination );
      BOOL IsValidCmdPacket( unsigned char *CommanOPT);
      void MakeSignature( unsigned char *CommanOPT);
      BOOL FlashRead( unsigned short addr , const LPBYTE lpBuffer,
unsigned short dwCount );
      BOOL FlashWrite(unsigned short addr , const LPBYTE lpBuffer,
unsigned short dwCount );
private:
};
#endif // _CEtherIOComm_H_

//controlDlg.h : header file
////////////////////////////////////////////////////////////////

#if
!defined(AFX_CONTROLDLG_H__E973774F_7538_4A4A_9FB6_CE5E4399AF80__INCLUDED_)
#define AFX_CONTROLDLG_H__E973774F_7538_4A4A_9FB6_CE5E4399AF80__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "boardlink.h"
#include "boardsetting.h"
```

```
#define       DEFAULT_IP "161.246.18.133"
//CControlDlg dialog
class CControlDlg : public CDialog
{
//Construction
public:
      CControlDlg(CWnd* pParent = NULL);  // standard constructor
      Cboardlink        m_EthernetIO;
      CString           m_StatusStr;
      CEdit             m_ReceiveEditCtrl;
      CString           m_ReceiveStr;
//Dialog Data
      //{{AFX_DATA(CControlDlg)
      enum { IDD = IDD_CONTROL_DIALOG };
      CIPAddressCtrl    m_IPaddress;
      unsigned char     value;
      CString                 m_Value;
      CString                 m_Value1;
      CString                 m_Value2;
      CString                 m_Value3;
      CString                 m_Value4;
      CComboBox               _Temp;
      //}}AFX_DATA
      //ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CControlDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);      //
DDX/DDV support
      //}}AFX_VIRTUAL
//Implementation
protected:
      HICON m_hIcon;
      Cboardsetting     m_SettingDlg;
      //Generated message map functions
      //{{AFX_MSG(CControlDlg)
      virtual BOOL OnInitDialog();
      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
      afx_msg void OnPaint();
      afx_msg HCURSOR OnQueryDragIcon();
      afx_msg void OnConnect();
      afx_msg void OnDiscon();
      afx_msg void OnQuit();
      afx_msg void OnSetting();
      afx_msg void OnCkac();
      afx_msg void OnOn1();
      afx_msg void OnOff1();
      afx_msg void OnOn2();
      afx_msg void OnOff2();
      afx_msg void OnOn3();
      afx_msg void OnOff3();
      afx_msg void OnOn4();
      afx_msg void OnOff4();
      afx_msg void OnOnall();
      afx_msg void OnOffall();
```

```
        afx_msg void OnCkall();
        afx_msg void OnOnair();
        afx_msg void OnOffair();
        afx_msg void OnReadtemp();
        afx_msg void OnSendtemp();
        afx_msg void Ck1();
        afx_msg void Ck2();
        afx_msg void Ck2();
        afx_msg void Ck4();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_CONTROLDLG_H__56F3F4C8_E5F1_4F74_AABD_F079E19B83CF__INCLUDED_)

//controlDlg.cpp :
////////////////////////////////////////////////////////////////////////
///////
#include "stdafx.h"
#include "control.h"
#include "controlDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
        CAboutDlg();
//Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        //ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV
support
        //}}AFX_VIRTUAL
//Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		//No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
//CControlDlg dialog
CControlDlg::CControlDlg(CWnd* pParent /*=NULL*/)
	:CDialog(CControlDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CControlDlg)
		//NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
	//Note that LoadIcon does not require a subsequent DestroyIcon
in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CControlDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CControlDlg)
	//}}AFX_DATA_MAP
	DDX_Text(pDX, IDC_VALUE, m_Value);
	DDX_Text(pDX, IDC_VALUE1, m_Value1);
	DDX_Text(pDX, IDC_VALUE2, m_Value2);
	DDX_Text(pDX, IDC_VALUE3, m_Value3);
	DDX_Text(pDX, IDC_VALUE4, m_Value4);
	DDX_Text(pDX, IDC_STATIC1, m_StatusStr);
	DDX_Text(pDX, IDC_AIRTEMP, m_ReceiveStr);
	DDX_Control(pDX, IDC_IPADDRESS1, m_IPaddress);
	DDX_Control(pDX, IDC_AIRTEMP, m_ReceiveEditCtrl);
	DDX_Control(pDX, IDC_TEMP_COMBO, m_Temp);
}
BEGIN_MESSAGE_MAP(CControlDlg, CDialog)
	//{{AFX_MSG_MAP(CControlDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_CONNECT, OnConnect)
	ON_BN_CLICKED(IDC_DISCON, OnDiscon)
	ON_BN_CLICKED(IDC_QUIT, OnQuit)
	ON_BN_CLICKED(IDC_SETTING, OnSetting)
	ON_BN_CLICKED(IDC_CKAC, OnCkac)
	ON_BN_CLICKED(IDC_ON1, OnOn1)
	ON_BN_CLICKED(IDC_OFF1, OnOff1)
	ON_BN_CLICKED(IDC_ON2, OnOn2)
	ON_BN_CLICKED(IDC_OFF2, OnOff2)
	ON_BN_CLICKED(IDC_ON3, OnOn3)
	ON_BN_CLICKED(IDC_OFF3, OnOff3)
```

```
	ON_BN_CLICKED(IDC_ON4, OnOn4)
	ON_BN_CLICKED(IDC_OFF4, OnOff4)
	ON_BN_CLICKED(IDC_ONALL, OnOnall)
	ON_BN_CLICKED(IDC_OFFALL, OnOffall)
	ON_BN_CLICKED(IDC_CKALL, OnCkall)
	ON_BN_CLICKED(IDC_ONAIR, OnOnair)
	ON_BN_CLICKED(IDC_OFFAIR, OnOffair)
	ON_BN_CLICKED(IDC_READTEMP, OnReadtemp)
	ON_BN_CLICKED(IDC_SENDTEMP, OnSendtemp)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
//CControlDlg message handlers
BOOL CControlDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	//Add "About..." menu item to system menu.
	//IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);
	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
	//Set the icon for this dialog.  The framework does this
automatically
	// when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
//Program_initial
	m_Temp.AddString("21");
	m_Temp.AddString("22");
	m_Temp.AddString("23");
	m_Temp.AddString("24");
	m_Temp.AddString("25");
	m_Temp.AddString("26");
	m_Temp.AddString("27");
	m_Temp.AddString("28");
	m_Temp.AddString("29");
	m_Temp.SetCurSel(0);
	GetDlgItem(IDC_CONNECT)->EnableWindow( TRUE );
	GetDlgItem(IDC_DISCON)->EnableWindow( FALSE );
	GetDlgItem(IDC_SETTING)->EnableWindow( FALSE );
	GetDlgItem(IDC_CKAC)->EnableWindow( FALSE );
	GetDlgItem(IDC_ON1)->EnableWindow( FALSE );
	GetDlgItem(IDC_OFF1)->EnableWindow( FALSE );
	GetDlgItem(IDC_ON2)->EnableWindow( FALSE );
	GetDlgItem(IDC_OFF2)->EnableWindow( FALSE );
```

```
	GetDlgItem(IDC_ON3)->EnableWindow( FALSE );
	GetDlgItem(IDC_OFF3)->EnableWindow( FALSE );
	GetDlgItem(IDC_ON4)->EnableWindow( FALSE );
	GetDlgItem(IDC_OFF4)->EnableWindow( FALSE );
	GetDlgItem(IDC_ONALL)->EnableWindow( FALSE );
	GetDlgItem(IDC_OFFALL)->EnableWindow( FALSE );
	GetDlgItem(IDC_CKALL)->EnableWindow( FALSE );
	GetDlgItem(IDC_ONAIR)->EnableWindow( FALSE );
	GetDlgItem(IDC_OFFAIR)->EnableWindow( FALSE );
	GetDlgItem(IDC_READTEMP)->EnableWindow( FALSE );
	GetDlgItem(IDC_SENDTEMP)->EnableWindow( FALSE );
//Ether_initial
	m_IPaddress.SetWindowText(_T(DEFAULT_IP));
	m_EthernetIO.SetMessageWindow( &m_ReceiveEditCtrl);
	m_EthernetIO.SetStatusMsg( &m_Value);
	m_EthernetIO.SetStatusMsg( &m_Value1);
	m_EthernetIO.SetStatusMsg( &m_Value2);
	m_EthernetIO.SetStatusMsg( &m_Value3);
	m_EthernetIO.SetStatusMsg( &m_Value4);
	m_EthernetIO.SetStatusMsg( &m_StatusStr);
	m_EthernetIO.SetServerState( false );	// run as client
	m_EthernetIO.SetSmartAddressing( false ); // always send to
server
	return TRUE;  // return TRUE  unless you set the focus to a
control
}
void CControlDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0)==IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}
// If you add a minimize button to your dialog, you will need the code
below
// to draw the icon.  For MFC applications using the document/view
model,
// this is automatically done for you by the framework.
void CControlDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(),
0);
		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
```

```
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) /2;
            int y = (rect.Height() - cyIcon + 1) /2;
            //Draw the icon
            dc.DrawIcon(x, y, m_hIcon);
      }
      else
      {
            CDialog::OnPaint();
      }
}
//The system calls this to obtain the cursor to display while the user
drags
// the minimized window.
HCURSOR CControlDlg::OnQueryDragIcon()
{
      return (HCURSOR) m_hIcon;
}
void CControlDlg::OnConnect()
{
      UpdateData(TRUE);
      CString IPServerStr;
      m_IPaddress.GetWindowText( IPServerStr );
      bool bSuccess;
      m_StatusStr.LoadString(IDS_CONNECTING_MSG);
      UpdateData(FALSE);
      bSuccess = m_EthernetIO.ConnectTo( IPServerStr ); // TCP
      if (bSuccess && m_EthernetIO.WatchComm())
      {
            m_StatusStr = _T("Connected at IP : ") + IPServerStr;
            GetDlgItem(IDC_CONNECT)->EnableWindow( FALSE );
            GetDlgItem(IDC_DISCON)->EnableWindow( TRUE );
            GetDlgItem(IDC_SETTING)->EnableWindow( TRUE );
            GetDlgItem(IDC_CKAC)->EnableWindow( TRUE );
            GetDlgItem(IDC_ONALL)->EnableWindow( TRUE );
            GetDlgItem(IDC_OFFALL)->EnableWindow( TRUE );
            GetDlgItem(IDC_CKALL)->EnableWindow( TRUE );
            GetDlgItem(IDC_ONAIR)->EnableWindow( TRUE );
            GetDlgItem(IDC_OFFAIR)->EnableWindow( TRUE );
            GetDlgItem(IDC_READTEMP)->EnableWindow( TRUE );
            GetDlgItem(IDC_SENDTEMP)->EnableWindow( TRUE );
            CControlDlg::OnCkall();
            UpdateData(FALSE);
      }
      else
      {
            m_StatusStr.LoadString(IDS_CONN_FAILED);
            UpdateData(FALSE);
      }
}
void CControlDlg::OnDiscon()
{
      m_EthernetIO.StopComm();
      m_StatusStr=_T("Disconnected");
      GetDlgItem(IDC_CONNECT)->EnableWindow( TRUE );
      GetDlgItem(IDC_DISCON)->EnableWindow( FALSE );
      GetDlgItem(IDC_SETTING)->EnableWindow( FALSE );
```

```
        GetDlgItem(IDC_CKAC)->EnableWindow( FALSE );
        GetDlgItem(IDC_ON1)->EnableWindow( FALSE );
        GetDlgItem(IDC_OFF1)->EnableWindow( FALSE );
        GetDlgItem(IDC_ON2)->EnableWindow( FALSE );
        GetDlgItem(IDC_OFF2)->EnableWindow( FALSE );
        GetDlgItem(IDC_ON3)->EnableWindow( FALSE );
        GetDlgItem(IDC_OFF3)->EnableWindow( FALSE );
        GetDlgItem(IDC_ON4)->EnableWindow( FALSE );
        GetDlgItem(IDC_OFF4)->EnableWindow( FALSE );
        GetDlgItem(IDC_ONALL)->EnableWindow( FALSE );
        GetDlgItem(IDC_OFFALL)->EnableWindow( FALSE );
        GetDlgItem(IDC_CKALL)->EnableWindow( FALSE );
        GetDlgItem(IDC_ONAIR)->EnableWindow( FALSE );
        GetDlgItem(IDC_OFFAIR)->EnableWindow( FALSE );
        GetDlgItem(IDC_READTEMP)->EnableWindow( FALSE );
        GetDlgItem(IDC_SENDTEMP)->EnableWindow( FALSE );
        UpdateData(FALSE);
}
void CControlDlg::OnQuit()
{
        CControlDlg::OnDiscon();
        CDialog::OnCancel();
}
void CControlDlg::OnSetting()
{
        if (m_EthernetIO.IsOpen() )
        {
            //Load old config
            if( m_EthernetIO.LoadConfig(  &m_SettingDlg.m_DataBits,
                                          &m_SettingDlg.m_StopBits ,
                                          &m_SettingDlg.m_ParityBits ,
                                          &m_SettingDlg.m_EnableHWHandshak,
                                          &m_SettingDlg.m_BuadRate      ,
                                          m_SettingDlg.m_IP,
                                          m_SettingDlg.m_Subnet,
                                          m_SettingDlg.m_Gateway,
                                          m_SettingDlg.m_Mac      ) !=
TRUE)
                {
                        AfxMessageBox("Load Old config fail!\n");
                        return;
                }
            int nRet = -1;
            nRet = m_SettingDlg.DoModal();
            switch ( nRet )
            {
            case -1:
                AfxMessageBox("Dialog box could not be created!");
                break;
            case IDABORT:
                break;
            case IDOK:
                if(m_EthernetIO.SaveConfig(
                    m_SettingDlg.m_DataBits,
                    m_SettingDlg.m_StopBits ,
                    m_SettingDlg.m_ParityBits ,
```

```
                    m_SettingDlg.m_EnableHWHandshak ,
                    m_SettingDlg.m_BuadRate  ,
                    m_SettingDlg.m_IP,
                    m_SettingDlg.m_Subnet,
                    m_SettingDlg.m_Gateway,
                    m_SettingDlg.m_Mac      ) != TRUE)
                {
                    AfxMessageBox("Save config fail!");
                    return;
                }
                m_EthernetIO.SystemSoftReset();
                CControlDlg::OnDiscon();
                break;
            case IDCANCEL:
                break;
            default:
                break;
            };
        }
}
void CControlDlg::OnCkac()
{
        char *tmpHex;
        value = (BYTE) strtoul((LPCTSTR) m_Value, &tmpHex, 16);
        m_EthernetIO.ReadBit(PORT_F, 7 , &value);
        if (value==1)
        {
            m_EthernetIO.SetBit(PORT_F, 7);
            m_Value.Format( _T("OK"));
            UpdateData(FALSE);
        }
        else
        {
            m_EthernetIO.ClearBit(PORT_F, 7);
            m_Value.Format( _T("NOT HAVE"));
            UpdateData(FALSE);
        }
}
void CControlDlg::OnOn()
{
        m_EthernetIO.SetBit(PORT_E, 0);
        CControlDlg::Ck1();
}
void CControlDlg::OnOff1()
{
        m_EthernetIO.ClearBit(PORT_E, 0);
        CControlDlg::Ck1();
}
void CControlDlg::Ck1()
{
        char *tmpHex;
        value = (BYTE) strtoul((LPCTSTR) m_Value, &tmpHex, 16);
        m_EthernetIO.ReadBit(PORT_E, 0 , &value);
        if (value==1)
        {
            m_EthernetIO.SetBit(PORT_E, 0);
            m_Value.Format( _T("ON"));
```

```
            GetDlgItem(IDC_ON1)->EnableWindow( FALSE );
            GetDlgItem(IDC_OFF1)->EnableWindow( TRUE );
      }
      else
      {
            m_EthernetIO.ClearBit(PORT_E, 0);
            m_Value1.Format( _T("OFF"));
            GetDlgItem(IDC_ON1)->EnableWindow( TRUE );
            GetDlgItem(IDC_OFF1)->EnableWindow( FALSE );
      }
      UpdateData(FALSE);
}
void CControlDlg::Ck2()
{
      char *tmpHex;
      value = (BYTE) strtoul((LPCTSTR) m_Value, &tmpHex, 16);
      m_EthernetIO.ReadBit(PORT_E, 1 , &value);
      if (value==1)
      {
            m_EthernetIO.SetBit(PORT_E, 1);
            m_Value2.Format( _T("ON"));
            GetDlgItem(IDC_ON2)->EnableWindow( FALSE );
            GetDlgItem(IDC_OFF2)->EnableWindow( TRUE );
      }
      else
      {
            m_EthernetIO.ClearBit(PORT_E, 1);
            m_Value2.Format( _T("OFF"));
            GetDlgItem(IDC_ON2)->EnableWindow( TRUE );
            GetDlgItem(IDC_OFF2)->EnableWindow( FALSE );
      }
      UpdateData(FALSE);
}
void CControlDlg::Ck3()
{
      char *tmpHex;
      value = (BYTE) strtoul((LPCTSTR) m_Value, &tmpHex, 16);
      m_EthernetIO.ReadBit(PORT_E, 2 , &value);
      if (value==1)
      {
            m_EthernetIO.SetBit(PORT_E, 2);
            m_Value3.Format( _T("ON"));
            GetDlgItem(IDC_ON3)->EnableWindow( FALSE );
            GetDlgItem(IDC_OFF3)->EnableWindow( TRUE );
      }
      else
      {
            m_EthernetIO.ClearBit(PORT_E, 2);
            m_Value3.Format( _T("OFF"));
            GetDlgItem(IDC_ON3)->EnableWindow( TRUE );
            GetDlgItem(IDC_OFF3)->EnableWindow( FALSE );
      }
      UpdateData(FALSE);
}
void CControlDlg::Ck4()
```

```
{
	char *tmpHex;
	value = (BYTE) strtoul((LPCTSTR) m_Value, &tmpHex, 16);
	m_EthernetIO.ReadBit(PORT_E, 4 , &value);
	if (value==1)
	{
		m_EthernetIO.SetBit(PORT_E, 4);
		m_Value4.Format( _T("ON"));
		GetDlgItem(IDC_ON4)->EnableWindow( FALSE );
		GetDlgItem(IDC_OFF4)->EnableWindow( TRUE );
	}
	else
	{
		m_EthernetIO.ClearBit(PORT_E, 4);
		m_Value4.Format( _T("OFF"));
		GetDlgItem(IDC_ON4)->EnableWindow( TRUE );
		GetDlgItem(IDC_OFF4)->EnableWindow( FALSE );
	}
	UpdateData(FALSE);
}
void CControlDlg::OnOn2()
{
	m_EthernetIO.SetBit(PORT_E, 1);
	CControlDlg::Ck2();
}

void CControlDlg::OnOff2()
{
	m_EthernetIO.ClearBit(PORT_E, 1);
	CControlDlg::Ck2();
}
void CControlDlg::OnOn3()
{
	m_EthernetIO.SetBit(PORT_E, 3);
	CControlDlg::Ck3();
}
void CControlDlg::OnOff3()
{
	m_EthernetIO.ClearBit(PORT_E, 2);
	CControlDlg::Ck3();
}
void CControlDlg::OnOn4()
{
	m_EthernetIO.SetBit(PORT_E, 4);
	CControlDlg::Ck4();
}
void CControlDlg::OnOff4()
{
	m_EthernetIO.ClearBit(PORT_E, 4 );
	CControlDlg::Ck4();
}
void CControlDlg::OnOnall()
{
		CControlDlg::OnOn1();
		CControlDlg::OnOn2();
		CControlDlg::OnOn3();
		CControlDlg::OnOn4();
```

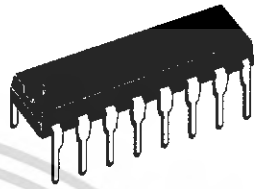
```
}
void CControlDlg::OnOffall()
{
            CControlDlg::OnOff1();
            CControlDlg::OnOff2();
            CControlDlg::OnOff3();
            CControlDlg::OnOff4();
}
void CControlDlg::OnCkall()
{
            CControlDlg::Ck1();
            CControlDlg::Ck2();
            CControlDlg::Ck3();
            CControlDlg::Ck4();
}
void CControlDlg::OnOnair()
{
      char message[]="A";
      if(m_EthernetIO.IsOpen())
      {
            if(m_EthernetIO.WriteComm((unsigned char*) &message[0],
sizeof(message), INFINITE) <= 0L)
            {
                  m_StatusStr.Format("Send fail");
                  UpdateData(FALSE);

            }
      }
}
void CControlDlg::OnOffair()
{
      char message[]="B";
      if(m_EthernetIO.IsOpen())
      {
            f(m_EthernetIO.WriteComm((unsigned char*) &message[0],
sizeof(message), INFINITE) <= 0L)
            {
                  m_StatusStr.Format("Send fail");
                  UpdateData(FALSE);
            }
      }
}
void CControlDlg::OnReadtemp()
{
      char message[]="H";
      UpdateData(TRUE);
      m_ReceiveEditCtrl.Clear();
      m_ReceiveStr = _T("");
      UpdateData(FALSE);
      if(m_EthernetIO.IsOpen())
      {
            m_EthernetIO.WriteComm((unsigned char*) &message[0], 7,
INFINITE);
      }
}
void CControlDlg::OnSendtemp()
{
```

```
        char temp;
        switch(m_Temp.GetCurSel())
        {
        case 0:     temp = 'C'; break;
        case 1:     temp = 'D'; break;
        case 2:     temp = 'E'; break;
        case 3:     temp = 'F'; break;
        case 4:     temp = 'G'; break;
        case 5:     temp = 'I'; break;
        case 6:     temp = 'J'; break;
        case 7:     temp = 'K'; break;
        default:TRACE("Unknow Temprature\n");
                    ASSERT(FALSE);
                    break;
        }
        if(m_EthernetIO.IsOpen())
        {
        if(m_EthernetIO.WriteComm((unsigned char*) &temp,
sizeof(temp), INFINITE) <= 0L)
                {
                    m_StatusStr.Format("Send fail");
                    UpdateData(FALSE);
                }
        }
}
```

ULN2001A-ULN2002A
ULN2003A-ULN2004A

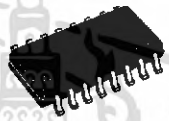
SEVEN DARLINGTON ARRAYS

- SEVEN DARLINGTONS PER PACKAGE
- OUTPUT CURRENT 500mA PER DRIVER (600mA PEAK)
- OUTPUT VOLTAGE 50V
- INTEGRATED SUPPRESSION DIODES FOR INDUCTIVE LOADS
- OUTPUTS CAN BE PARALLELED FOR HIGHER CURRENT
- TTL/CMOS/PMOS/DTL COMPATIBLE INPUTS
- INPUTS PINNED OPPOSITE OUTPUTS TO SIMPLIFY LAYOUT



DIP16

ORDERING NUMBERS: ULN2001A/2A/3A/4A



SO16

ORDERING NUMBERS: ULN2001D/2D/3D/4D

DESCRIPTION

The ULN2001A, ULN2002A, ULN2003 and ULN2004A are high voltage, high current darlington arrays each containing seven open collector darlington pairs with common emitters. Each channel rated at 500mA and can withstand peak currents of 600mA. Suppression diodes are included for inductive load driving and the inputs are pinned opposite the outputs to simplify board layout.

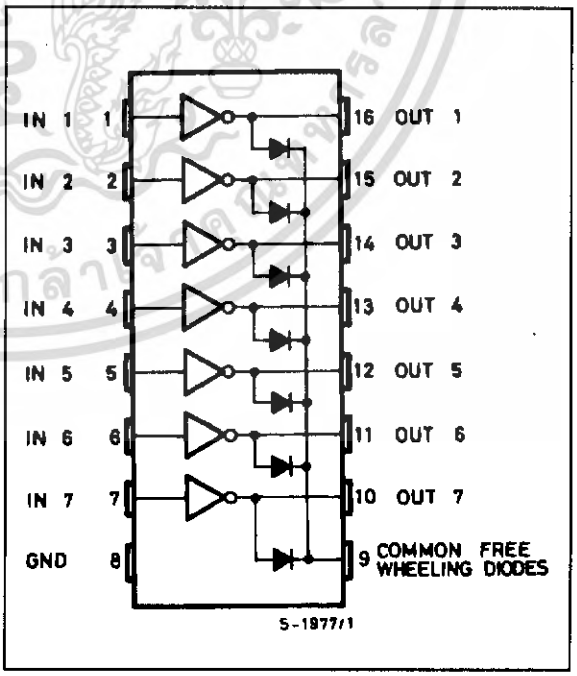
The four versions interface to all common logic families :

ULN2001A	General Purpose, DTL, TTL, PMOS, CMOS
ULN2002A	14-25V PMOS
ULN2003A	5V TTL, CMOS
ULN2004A	6–15V CMOS, PMOS

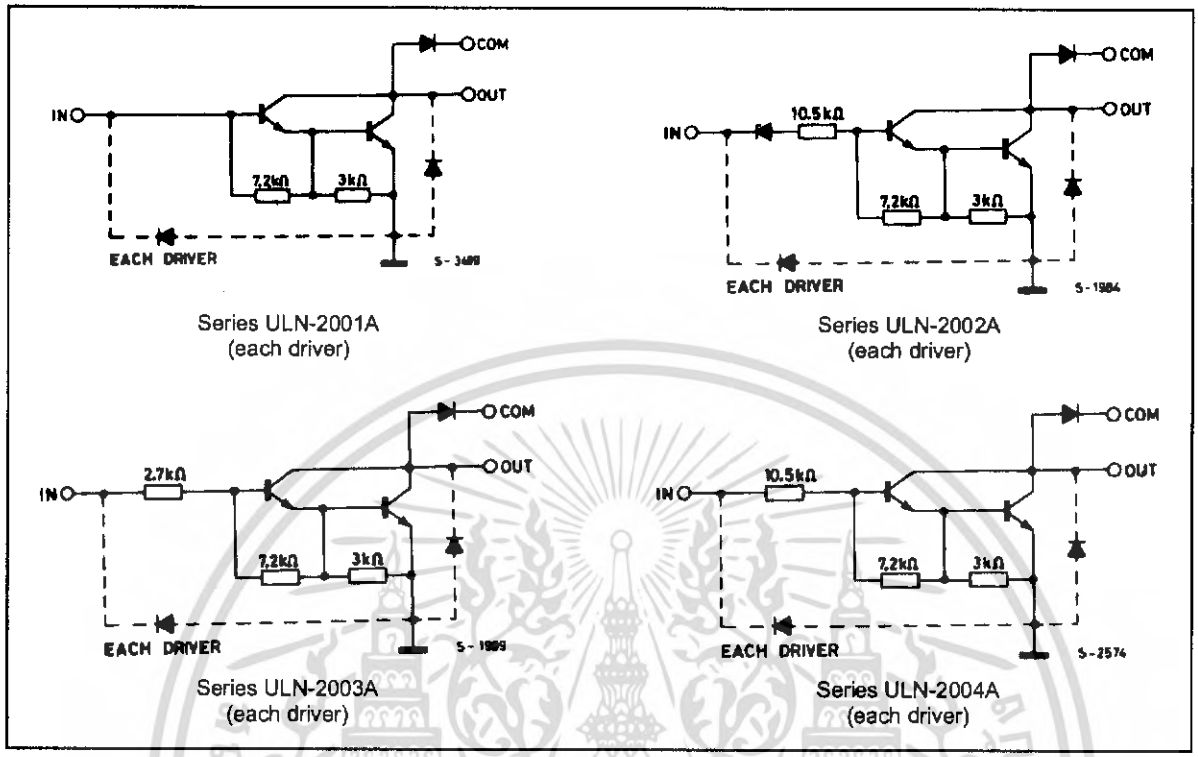
These versatile devices are useful for driving a wide range of loads including solenoids, relays DC motors, LED displays filament lamps, thermal printheads and high power buffers.

The ULN2001A/2002A/2003A and 2004A are supplied in 16 pin plastic DIP packages with a copper leadframe to reduce thermal resistance. They are available also in small outline package (SO-16) as ULN2001D/2002D/2003D/2004D.

PIN CONNECTION



SCHEMATIC DIAGRAM



Series ULN-2001A (each driver)

Series ULN-2002A (each driver)

Series ULN-2003A (each driver)

Series ULN-2004A (each driver)

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_o	Output Voltage	50	V
V_{in}	Input Voltage (for ULN2002A/D - 2003A/D - 2004A/D)	30	V
I_c	Continuous Collector Current	500	mA
I_b	Continuous Base Current	25	mA
T_{amb}	Operating Ambient Temperature Range	– 20 to 85	°C
T_{stg}	Storage Temperature Range	– 55 to 150	°C
T_j	Junction Temperature	150	°C

THERMAL DATA

Symbol	Parameter		DIP16	SO16	Unit
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	70	120	°C/W

ELECTRICAL CHARACTERISTICS (T_{amb} = 25°C unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit	Fig.
I_{CEX}	Output Leakage Current	V_{CE} = 50V			50	µA	1a
		T_{amb} = 70°C, V_{CE} = 50V			100	µA	1a
		T_{amb} = 70°C for ULN2002A V_{CE} = 50V, V_i = 6V			500	µA	1b
		for ULN2004A V_{CE} = 50V, V_i = 1V			500	µA	1b
$V_{CE(sat)}$	Collector-emitter Saturation Voltage	I_C = 100mA, I_B = 250µA		0.9	1.1	V	2
		I_C = 200 mA, I_B = 350µA		1.1	1.3	V	2
		I_C = 350mA, I_B = 500µA		1.3	1.6	V	2
$I_{i(on)}$	Input Current	for ULN2002A, V_i = 17V		0.82	1.25	mA	3
		for ULN2003A, V_i = 3.85V		0.93	1.35	mA	3
		for ULN2004A, V_i = 5V		0.35	0.5	mA	3
		V_i = 12V		1	1.45	mA	3
$I_{i(off)}$	Input Current	T_{amb} = 70°C, I_C = 500µA	50	65		µA	4
$V_{i(on)}$	Input Voltage	V_{CE} = 2V				V	5
		for ULN2002A I_C = 300mA			13		
		for ULN2003A I_C = 200mA			2.4		
		I_C = 250mA			2.7		
		I_C = 300mA			3		
		for ULN2004A I_C = 125mA			5		
		I_C = 200mA			6		
		I_C = 275mA			7		
		I_C = 350mA			8		
h_{FE}	DC Forward Current Gain	for ULN2001A V_{CE} = 2V, I_C = 350mA	1000				2
C_i	Input Capacitance			15	25	pF	
t_{PLH}	Turn-on Delay Time	0.5 V_i to 0.5 V_o		0.25	1	µs	
t_{PHL}	Turn-off Delay Time	0.5 V_i to 0.5 V_o		0.25	1	µs	
I_R	Clamp Diode Leakage Current	V_R = 50V			50	µA	6
		T_{amb} = 70°C, V_R = 50V			100	µA	6
V_F	Clamp Diode Forward Voltage	I_F = 350mA		1.7	2	V	7

TEST CIRCUITS

Figure 1a.

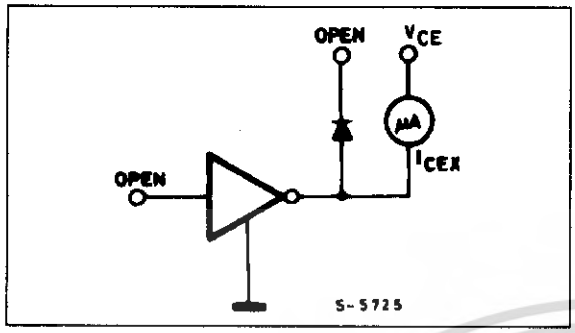


Figure 1b.

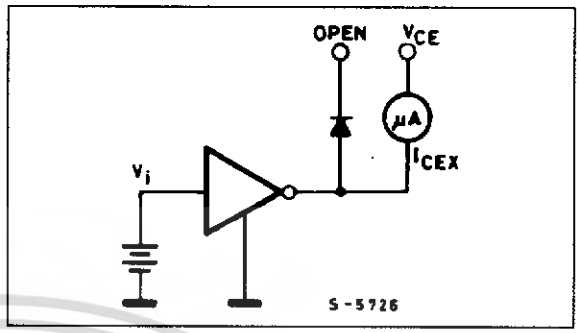


Figure 2.

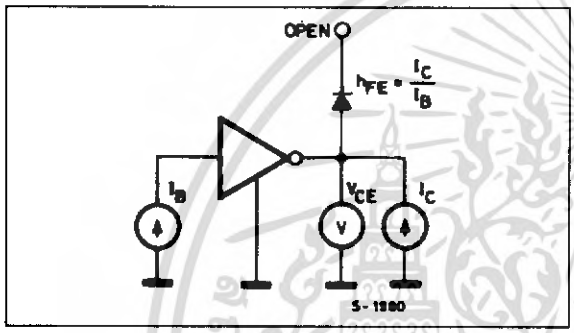


Figure 3.

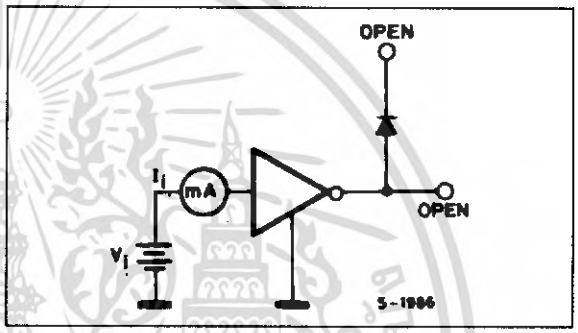


Figure 4.

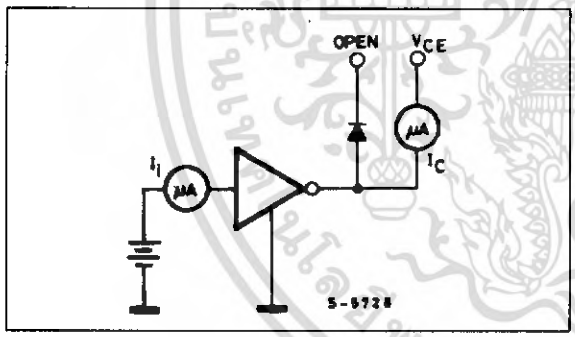


Figure 5.

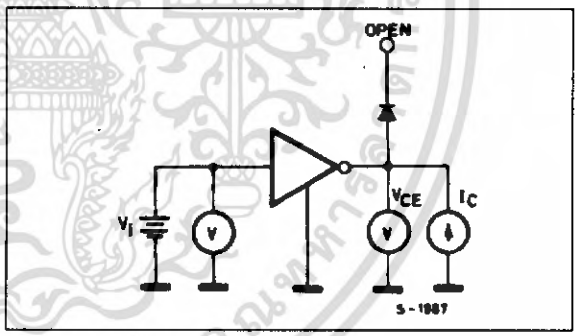


Figure 6.

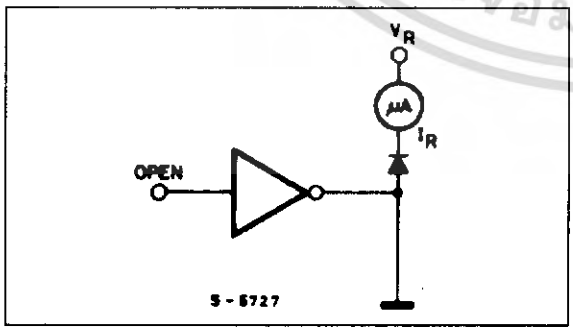


Figure 7.

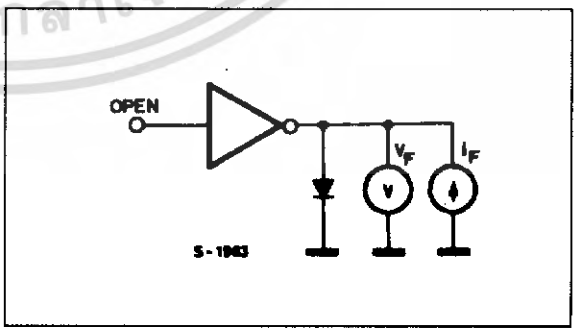


Figure 8: Collector Current versus Input Current

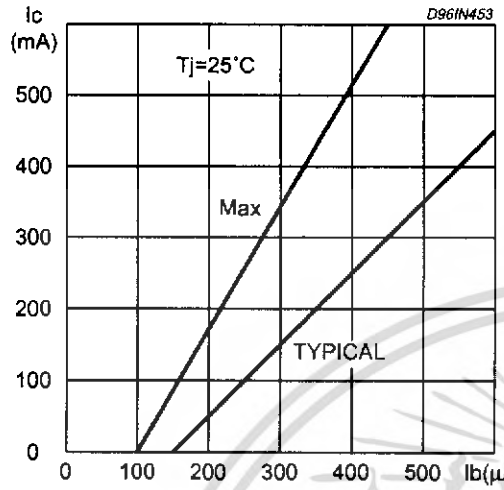


Figure 9: Collector Current versus Saturation Voltage

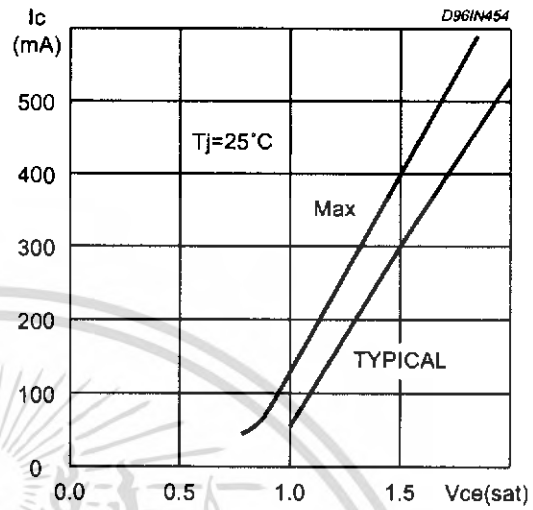


Figure 10: Peak Collector Current versus Duty Cycle

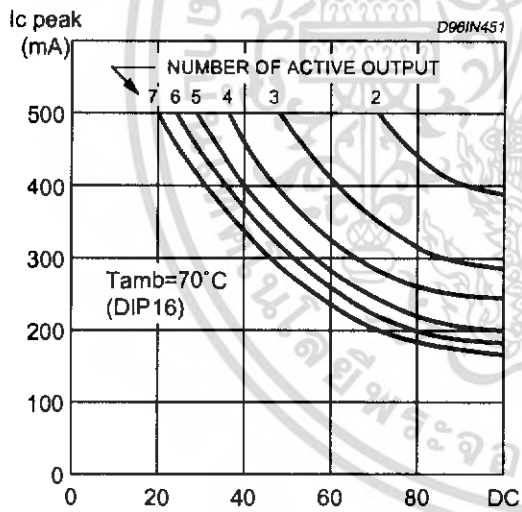
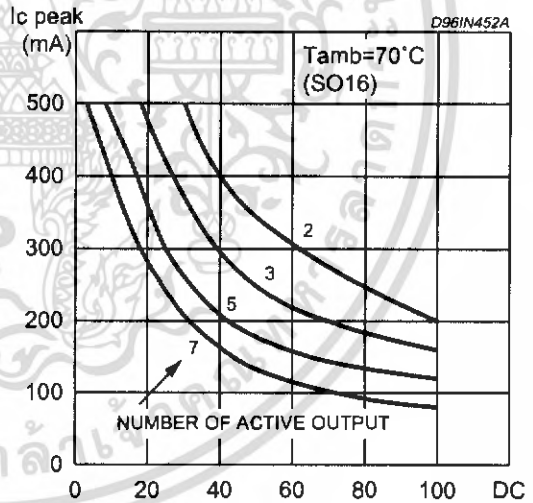
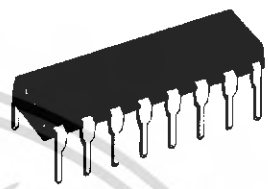


Figure 11: Peak Collector Current versus Duty Cycle

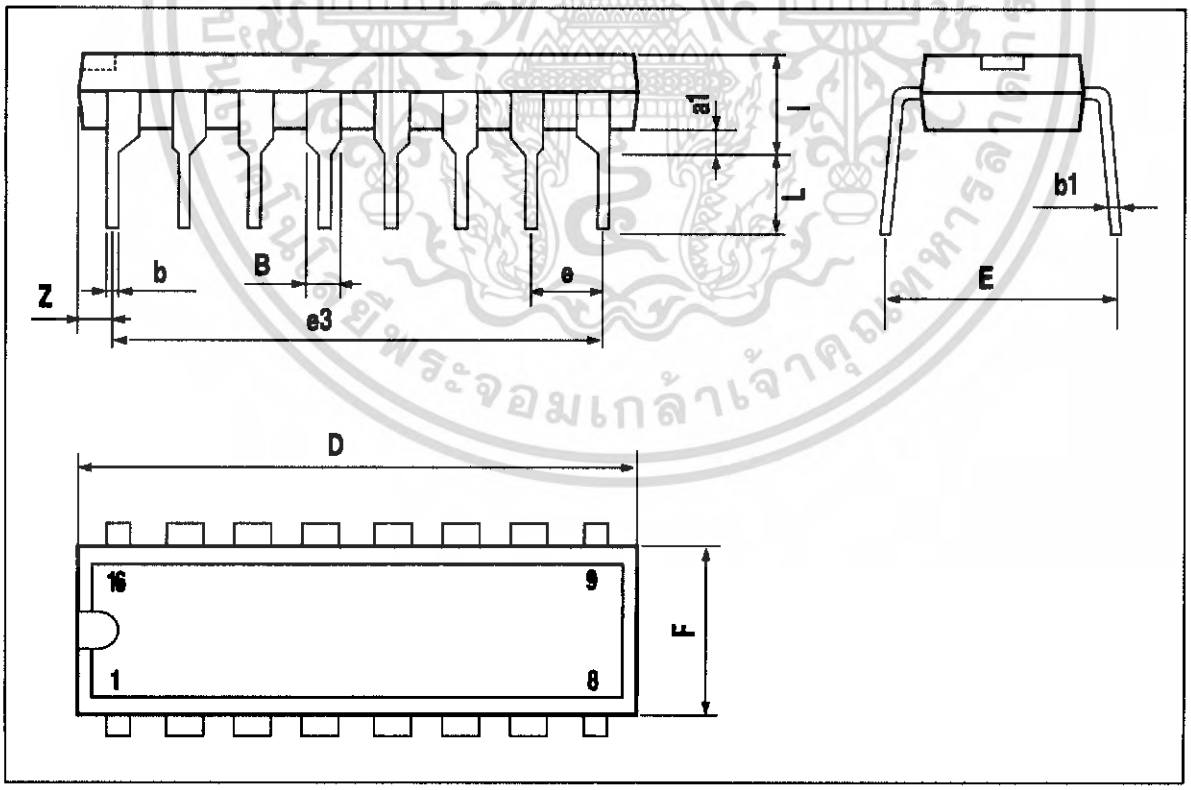


DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
a1	0.51			0.020		
B	0.77		1.65	0.030		0.065
b		0.5			0.020	
b1		0.25			0.010	
D			20			0.787
E		8.5			0.335	
e		2.54			0.100	
e3		17.78			0.700	
F			7.1			0.280
I			5.1			0.201
L		3.3			0.130	
Z			1.27			0.050

OUTLINE AND MECHANICAL DATA



DIP16



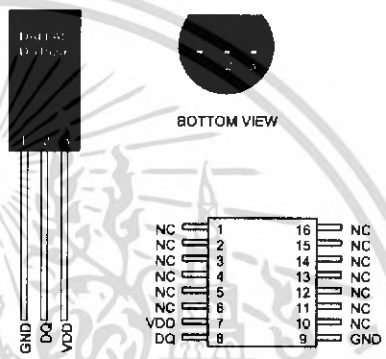
DALLAS SEMICONDUCTOR

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.

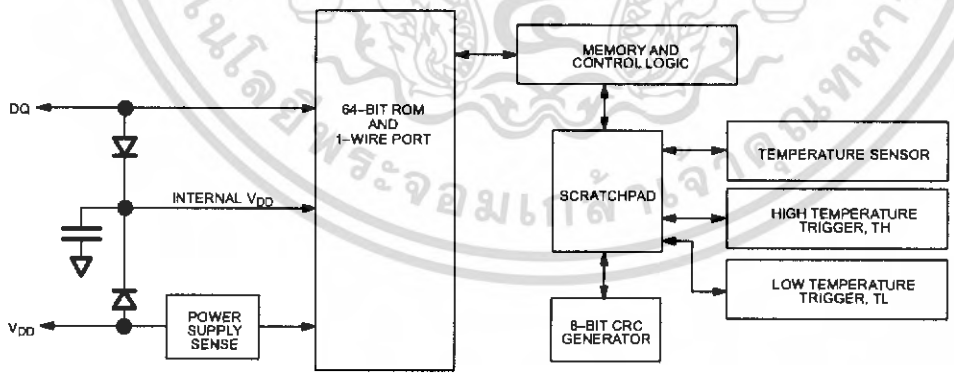
OVERVIEW

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.

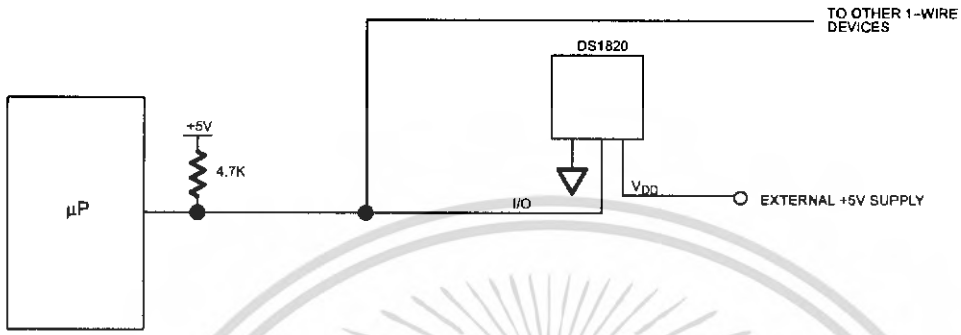
Communication to the DS1820 is via a 1–Wire port. With the 1–Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, 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



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 registar, 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 representad 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

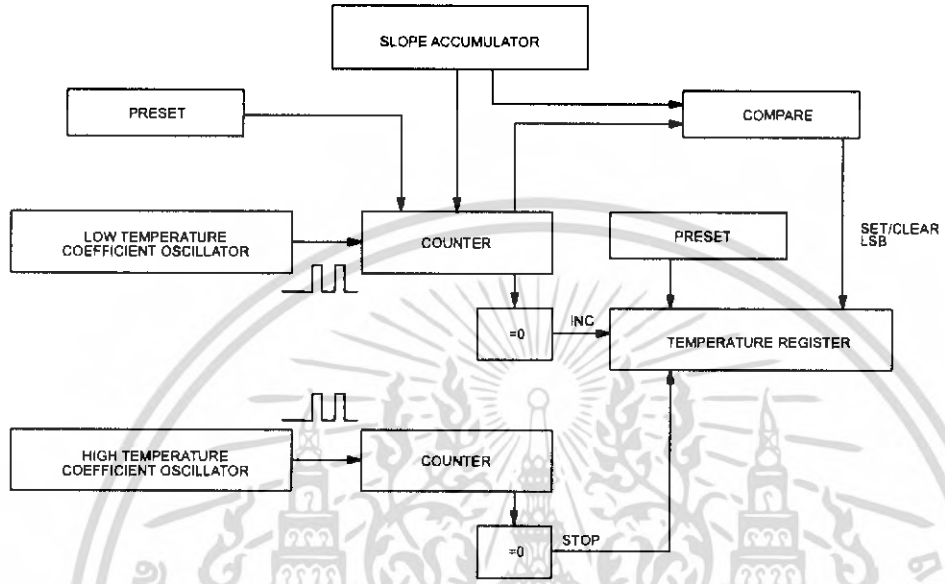
= –25°C

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:

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

TEMPERATURE MEASURING CIRCUITRY Figure 4



TEMPERATURE/DATA RELATIONSHIPS Table 1

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

OPERATION – ALARM SIGNALING

After the DS1820 has performed a temperature conversion, the temperature value is compared to the trigger values stored in TH and TL. Since these registers are 8–bit only, the 0.5°C bit is ignored for comparison. The most significant bit of TH or TL directly corresponds to the sign bit of the 16–bit temperature register. If the result of a temperature measurement is higher than TH or lower than TL, an alarm flag inside the device is set. This flag is updated with every temperature measurement. As long as the alarm flag is set, the DS1820 will respond to the alarm search command. This allows many DS1820s to be connected in parallel doing simultaneous temperature measurements. If somewhere the temperature exceeds the limits, the alarming device(s) can be identified and read immediately without having to read non–alarming devices.

64–BIT LASERED ROM

Each DS1820 contains a unique ROM code that is 64–bits long. The first eight bits are a 1–Wire family code (DS1820 code is 10h). The next 48 bits are a unique serial number. The last eight bits are a CRC of the first 56 bits. (See Figure 5.) The 64–bit ROM and ROM Function Control section allow the DS1820 to operate as a 1–Wire device and follow the 1–Wire protocol detailed in the section "1–Wire Bus System". The functions required to control sections of the DS1820 are not accessible until the ROM function protocol has been satisfied. This protocol is described in the ROM function protocol flowchart (Figure 6). The 1–Wire bus 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. After a ROM functions sequence has been successfully executed, the functions specific to the DS1820 are accessible and the bus master may then provide and one of the six memory and control function commands.

CRC GENERATION

The DS1820 has an 8–bit CRC stored in the most significant byte of the 64–bit ROM. The bus master can compute a CRC value from the first 56–bits of the 64–bit ROM and compare it to the value stored within the DS1820 to determine if the ROM data has been received error–free by the bus master. The equivalent polynomial function of this CRC is:

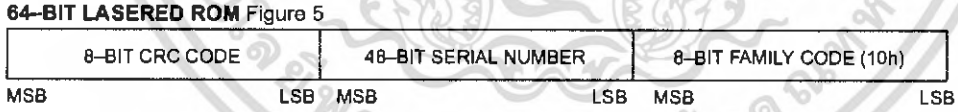
$$CRC = X^8 + X^5 + X^4 + 1$$

The DS1820 also generates an 8–bit CRC value using the same polynomial function shown above and provides this value to the bus master to validate the transfer of data bytes. In each cese where a CRC is used for data transfar validation, the bus master must calculate a CRC value using the polynomial function given above and compare the calculated value to either the 8–bit CRC value stored in the 64–bit ROM portion of the DS1820 (for ROM reads) or the 8–bit CRC value computed within the DS1820 (which is read as a ninth byte when the scretchpad is read). The comparison of CRC values and decision to continue with an operation are determined entirely by the bus mester. There is no circuitry inside the DS1820 that prevents a command sequence from proceeding if the CRC stored in or calculated by the DS1820 does not match the value generated by the bus master.

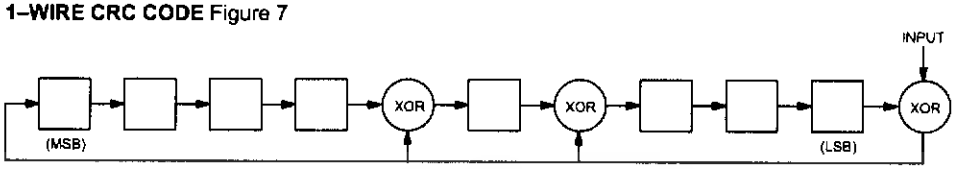
The 1–Wire CRC can be generated using a polynomial generator consisting of a shift register and XOR gates as shown in Figure 7. Additional information about the Dallas 1–Wire Cyclic Redundancy Check is available in Applicetion Note 27 entitled "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Products".

The shift register bits are initialized to zero. Then starting with the least significant bit of the family code, one bit at a time is shifted in. After the 8th bit of the family code has been entered, then the serial number is entered. After the 48th bit of the serial number has been entered, the shift register contains the CRC value. Shifting in the eight bits of CRC should return the shift register to all zeros.

64–BIT LASERED ROM Figure 5



1–WIRE CRC CODE Figure 7



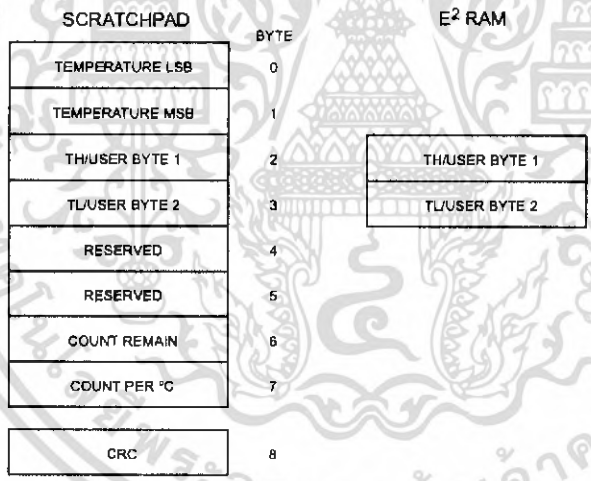
MEMORY

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



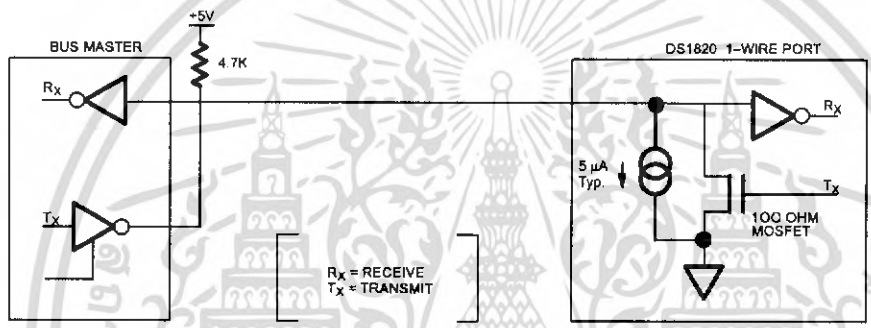
1–WIRE BUS SYSTEM

The 1–Wire bus is a system which has a single bus master and one or more slaves. The DS1820 behaves as a slave. The discussion of this bus system is broken down into three topics: hardware configuration, transaction sequence, and 1–Wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1–Wire bus has only a single line by definition; it is important that each device on the bus be able to drive it at the appropriate time. To facilitate this, each device attached to the 1–Wire bus must have open drain or 3–state outputs. The 1–Wire port of the DS1820 (I/O pin) is open drain with an internal circuit equivalent to that shown in Figure 9. A multidrop bus consists of a 1–Wire bus with multiple slaves attached. The 1–Wire bus requires a pullup resistor of approximately 5KΩ.

HARDWARE CONFIGURATION Figure 9



The idle state for the 1–Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1–Wire bus is in the inactive (high) state during the recovery period. If this does not occur and the bus is left low for more than 480 µs, all components on the bus will be reset.

TRANSACTION SEQUENCE

The protocol for accessing the DS1820 via the 1–Wire port is as follows:

- Initialization
- ROM Function Command
- Memory Function Command
- Transaction/Data

INITIALIZATION

All transactions on the 1–Wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s).

The presence pulse lets the bus master know that the DS1820 is on the bus and is ready to operate. For more details, see the "1–Wire Signaling" section.

ROM FUNCTION COMMANDS

Once the bus master has detected a presence, it can issue one of the five ROM function commands. All ROM function commands are 8–bits long. A list of these commands follows (refer to flowchart in Figure 6):

Read ROM [33h]

This command allows the bus master to read the DS1820's 8–bit family code, unique 48–bit serial number, and 8–bit CRC. This command can only be used if there is a single DS1820 on the bus. If more than one slave is present on the bus, a data collision will occur when all slaves try to transmit at the same time (open drain will produce a wired AND result).

Match ROM [55h]

The match ROM command, followed by a 64–bit ROM sequence, allows the bus master to address a specific DS1820 on a multidrop bus. Only the DS1820 that exactly matches the 64–bit ROM sequence will respond to the following memory function command. All slaves that do not match the 64–bit ROM sequence will wait for a reset pulse. This command can be used with a single or multiple devices on the bus.

Skip ROM [CCh]

This command can save time in a single drop bus system by allowing the bus master to access the memory functions without providing the 64–bit ROM code. If more than one slave is present on the bus and a read command is issued following the Skip ROM command, data collision will occur on the bus as multiple slaves transmit simultaneously (open drain pulldowns will produce a wired AND result).

Search ROM [F0h]

When a system is initially brought up, the bus master might not know the number of devices on the 1–Wire bus or their 64–bit ROM codes. The search ROM command allows the bus master to use a process of elimination to identify the 64–bit ROM codes of all slave devices on the bus.

Alarm Search [ECh]

The flowchart of this command is identical to the Search ROM command. However, the DS1820 will respond to this command only if an alarm condition has been encountered at the last temperature measurement. An alarm condition is defined as a temperature higher than TH or lower than TL. The alarm condition remains set as long as the DS1820 is powered up, or until another temperature measurement reveals a non–alarming value. For alarming, the trigger values stored in EEPROM are taken into account. If an alarm condition exists and the TH or TL settings are changed, another temperature conversion should be done to validate any alarm conditions.

Example of a ROM Search

The ROM search process is the repetition of a simple 3–step routine: read a bit, read the complement of the bit, then write the desired value of that bit. The bus master performs this simple, 3–step routine on each bit of the ROM. After one complete pass, the bus master knows the contents of the ROM in one device. The remaining number of devices and their ROM codes may be identified by additional passes.

The following example of the ROM search process assumes four different devices are connected to the same 1–Wire bus. The ROM data of the four devices is as shown:

ROM1	00110101...
ROM2	10101010...
ROM3	11110101...
ROM4	00010001...

The search process is as follows:

1. The bus master begins the initialization sequence by issuing a reset pulse. The slave devices respond by issuing simultaneous presence pulses.
2. The bus master will then issue the Search ROM command on the 1–Wire bus.
3. The bus master reads a bit from the 1–Wire bus. Each device will respond by placing the value of the first bit of their respective ROM data onto the 1–Wire bus. ROM1 and ROM4 will place a 0 onto the 1–Wire bus, i.e., pull it low. ROM2 and ROM3 will place a 1 onto the 1–Wire bus by allowing the line to stay high. The result is the logical AND of all devices on the line, therefore the bus master sees a 0. The bus master reads another bit. Since the Search ROM data command is being executed, all of the devices on the 1–Wire bus respond to this second read by placing the complement of the first bit of their respective ROM data onto the 1–Wire bus. ROM1 and ROM4 will place a 1 onto the 1–Wire, allowing the line to stay high. ROM2 and ROM3 will place a 0 onto the 1–Wire, thus it will be pulled low. The bus master again observes a 0 for the complement of the first ROM data bit. The bus master has determined that there are some devices on the 1–Wire bus that have a 0 in the first position and others that have a 1.

The data obtained from the two reads of the 3–step routine have the following interpretations:

00 There are still devices attached which have conflicting bits in this position.

01 All devices still coupled have a 0–bit in this bit position.

10 All devices still coupled have a 1–bit in this bit position.

11 There are no devices attached to the 1–Wire bus.

4. The bus master writes a 0. This deselects ROM2 and ROM3 for the remainder of this search pass, leaving only ROM1 and ROM4 connected to the 1–Wire bus.
5. The bus master performs two more reads and receives a 0–bit followed by a 1–bit. This indicates that all devices still coupled to the bus have 0's as their second ROM data bit.
6. The bus master then writes a 0 to keep both ROM1 and ROM4 coupled.
7. The bus master executes two reads and receives two 0–bits. This indicates that both 1–bits and 0–bits exist as the third bit of the ROM data of the attached devices.
8. The bus master writes a 0–bit. This deselects ROM1 leaving ROM4 as the only device still connected.
9. The bus master reads the remainder of the ROM bits for ROM4 and continues to access the part if desired. This completes the first pass and uniquely identifies ona part on the 1–Wire bus.
10. The bus master starts a new ROM search sequence by repeating steps 1 through 7.
11. The bus master writes a 1–bit. This dacouples ROM4, leaving only ROM1 still coupled.
12. The bus master reads the remainder of the ROM bits for ROM1 and communicates to the underlying logic if desired. This completes the second ROM search pass, in which another of the ROMs was found.
13. The bus master starts a new ROM search by repeating steps 1 through 3.
14. The bus master writes a 1–bit. This deselects ROM1 and ROM4 for the remainder of this search pass, leaving only ROM2 and ROM3 coupled to the system.
15. The bus master executes two read time slots and receives two zeros.
16. The bus master writes a 0–bit. This decouples ROM3, and leaving only ROM2.
17. The bus master reads the remainder of the ROM bits for ROM2 and communicates to the underlying logic if desired. This completes the third ROM search pass, in which another of the ROMs was found.
18. The bus master starts a new ROM search by repeating steps 13 through 15.
19. The bus master writes a 1–bit. This decouples ROM2, leaving only ROM3.
20. The bus master reads the remainder of the ROM bits for ROM3 and communicates to the underlying logic if desired. This completes the fourth ROM search pass, in which another of the ROMs was found.

Note the following:
The bus master learns the unique ID number (ROM data pattern) of one 1–Wire device on each ROM Search operation. The time required to derive the part's unique ROM code is:

$$960\ \mu s + (8 + 3 \times 64)\ 61\ \mu s = 13.16\ ms$$

The bus master is therefore capable of identifying 75 different 1–Wire devices per second.

I/O SIGNALING

The DS1820 requires strict protocols to insure data integrity. The protocol consists of several types of signaling on one line: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. All of these signals, with the exception of the presence pulse, are initiated by the bus master.

The initialization sequence required to begin any communicetion with the DS1820 is shown in Figure 11. A reset pulse followed by a presence pulse indicates the DS1820 is ready to send or receive data given the correct ROM command and memory function command.

The bus master transmits (TX) a reset pulse (a low signal for a minimum of 480 µs). The bus master then releases the line and goes into a receive mode (RX). The 1–Wire bus is pulled to a high state via the 5K pull–up resistor . After detecting the rising edge on the

I/O pin, the DS1820 waits 15–60 µs and then transmits the presence pulse (a low signal for 60–240 µs).

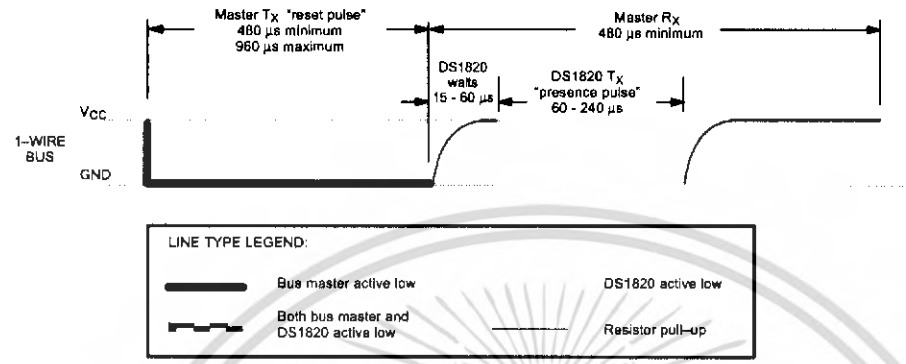
MEMORY COMMAND FUNCTIONS

The following command protocols are summarized in Table 2, and by the flowchart of Figure 10.

Write Scratchpad [4Eh]

This command writes to the scratchpad of the DS1820, starting at address 2. The next two bytes written will be saved in scratchpad memory, at address locations 2 and 3. Writing may be terminated at any point by issuing a reset.

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 et 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 reeding 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 date 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 commend 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).

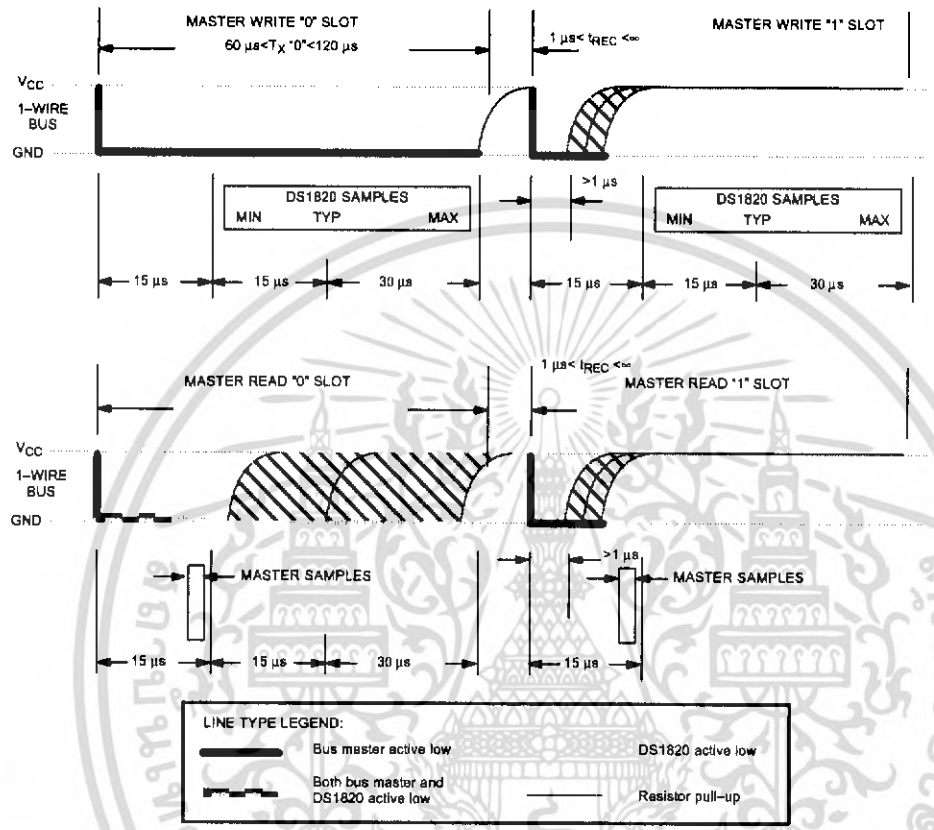
For the host to generate a Write One time slot, the data line must be pulled to a logic low level and then released, allowing the data line to pull up to a high level within 15 µs after the start of the write time slot.

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.

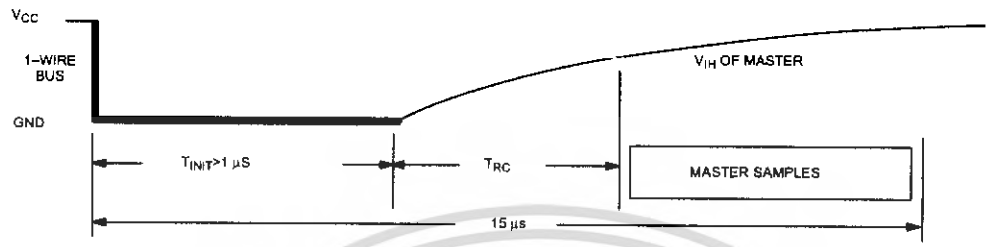
Read Time Slots
The host generates read time slots when data is to be read from the DS1820. A read time slot is initiated when the host pulls the date line from a logic high level to logic low level. The data line must remain at a low logic level for a minimum of one µs; output data from the DS1820 is 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 individuel 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} end T_{RC} as small es possible and by locating the master semple time towards the end of the 15 µs period.

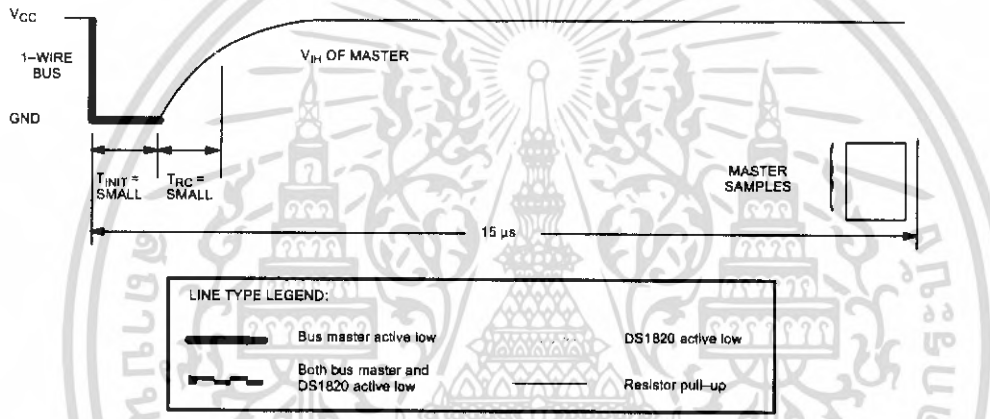
READ/WRITE TIMING DIAGRAM Figure 12



DETAILED MASTER READ "1" TIMING Figure 13



RECOMMENDED MASTER READ "1" TIMING Figure 14



ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to Ground	–0.5V to +7.0V
Operating Temperature	–55°C to +125°C
Storage Temperature	–55°C to +125°C
Soldering Temperature	260°C for 10 seconds

* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	I/O Functions	2.8	5.0	5.5	V	1, 2
		±½°C Accurate Temperature Conversions	4.3		5.5		
Data Pin	I/O		–0.5		+5.5	V	2
Logic 1	V_{IH}		2.0		V_{CC}+0.3	V	2, 3
Logic 0	V_{IL}		–0.3		+0.8	V	2, 4

DC ELECTRICAL CHARACTERISTICS

(–55°C to +125°C; V_{DD}=3.6V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Thermometer Error	t_{ERR}	–0°C to +70°C			±½	°C	
		–55°C to 0°C and +70°C to +125°C		See Typical Curve			1, 9, 10
Input Logic High	V_{IH}		2.2		5.5	V	2, 3
Input Logic Low	V_{IL}		–0.3		+0.8	V	2, 4
Sink Current	I_L	$V_{I/O}$=0.4V	–4.0			mA	2
Standby Current	I_Q			200	350	nA	8
Active Current	I_{DD}			1	1.5	mA	5, 6
Input Load Current	I_L			5		µA	7

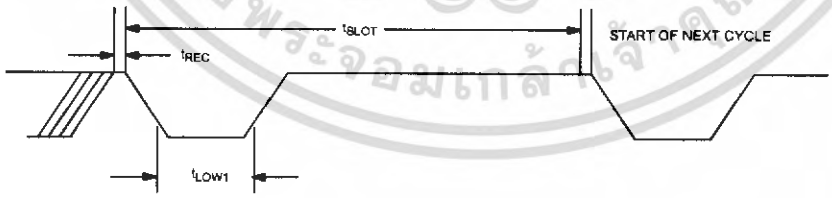
AC ELECTRICAL CHARACTERISTICS: (−55°C to +125°C; V_{DD}=3.6V to 5.5V)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}		200	500	ms	
Time Slot	t_{SLOT}	60		120	µs	
Recovery Time	t_{REC}	1			µs	
Write 0 Low Time	t_{LOW0}	60		120	µs	
Write 1 Low Time	t_{LOW1}	1		15	µs	
Read Data Valid	t_{RDV}			15	µs	
Reset Time High	t_{RSTH}	480			µs	
Reset Time Low	t_{RSTL}	480		4800	µs	
Presence Detect High	t_{PDHIGH}	15		60	µs	
Presence Detect Low	t_{PDLOW}	60		240	µs	
Capacitance	$C_{IN/OUT}$			25	pF	

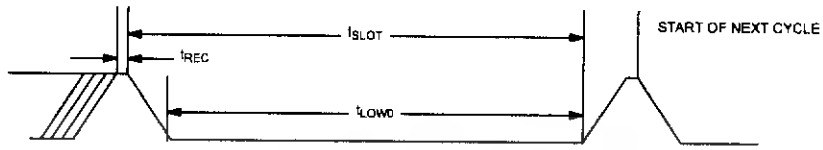
NOTES:

1. Temperature conversion will work with ±2°C accuracy down to V_{DD} = 3.4 volts.
2. All voltages are referenced to ground.
3. Logic one voltages are specified at a source current of 1 mA.
4. Logic zero voltages are specified at a sink current of 4 mA.
5. I_{DD} specified with V_{CC} at 5.0 volts.
6. Active current refers to either temperature conversion or writing to the E^2 memory. Writing to E^2 memory consumes approximately 200 µA for up to 10 ms.
7. Input load is to ground.
8. Standby current specified up to 70°C. Standby current typically is 5 µA at 125°C.
9. See Typical Curve for specification limits outside the 0°C to 70°C range. Thermometer error reflects sensor accuracy as tested during calibration.
10. Typical accuracy curve valid for 4.3V ≤ V_{DD} ≤ 5.5V.

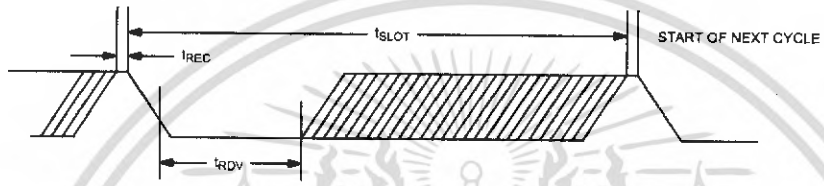
1-WIRE WRITE ONE TIME SLOT



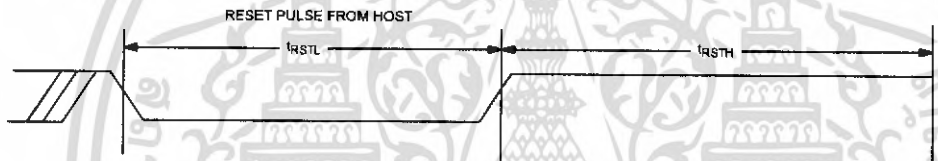
1–WIRE WRITE ZERO TIME SLOT



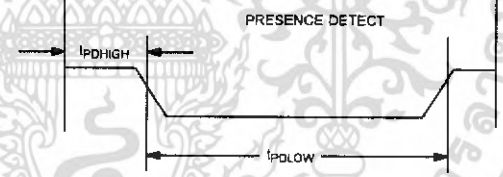
1–WIRE READ ZERO TIME SLOT



1–WIRE RESET PULSE

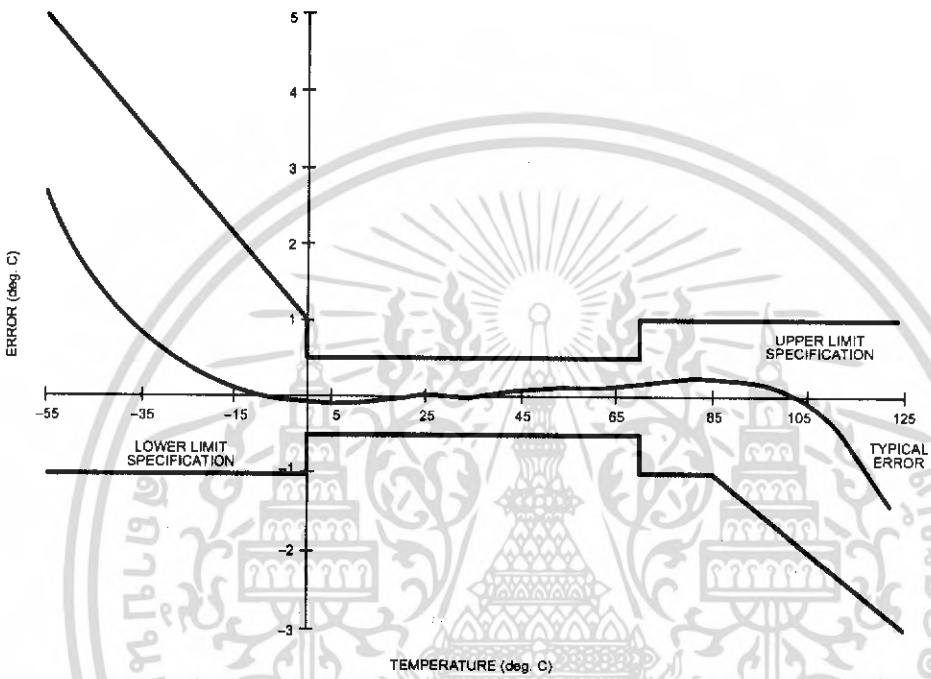


1–WIRE PRESENCE DETECT



TYPICAL PERFORMANCE CURVE

DS1820 DIGITAL TERMOMETER AND THERMOSTAT
TEMPERATURE READING ERROR



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

1 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 analog inputs programmable as single-ended or differential inputs
- Auto-incremented channel selection
- Analog 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 analog output.

2 APPLICATIONS

- Closed loop control systems
- Low power converter for remote data acquisition
- Battery operated equipment
- Acquisition of analog values in automotive, audio and TV applications.

3 GENERAL DESCRIPTION

The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog 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 I^2C-bus.

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

4 ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
PCF8591P	DIP16	plastic dual in-line package; 16 leads (300 mil); long body	SOT38-1
PCF8591T	SO16	plastic small outline package; 16 leads; body width 7.5 mm	SOT162-1

5 BLOCK DIAGRAM

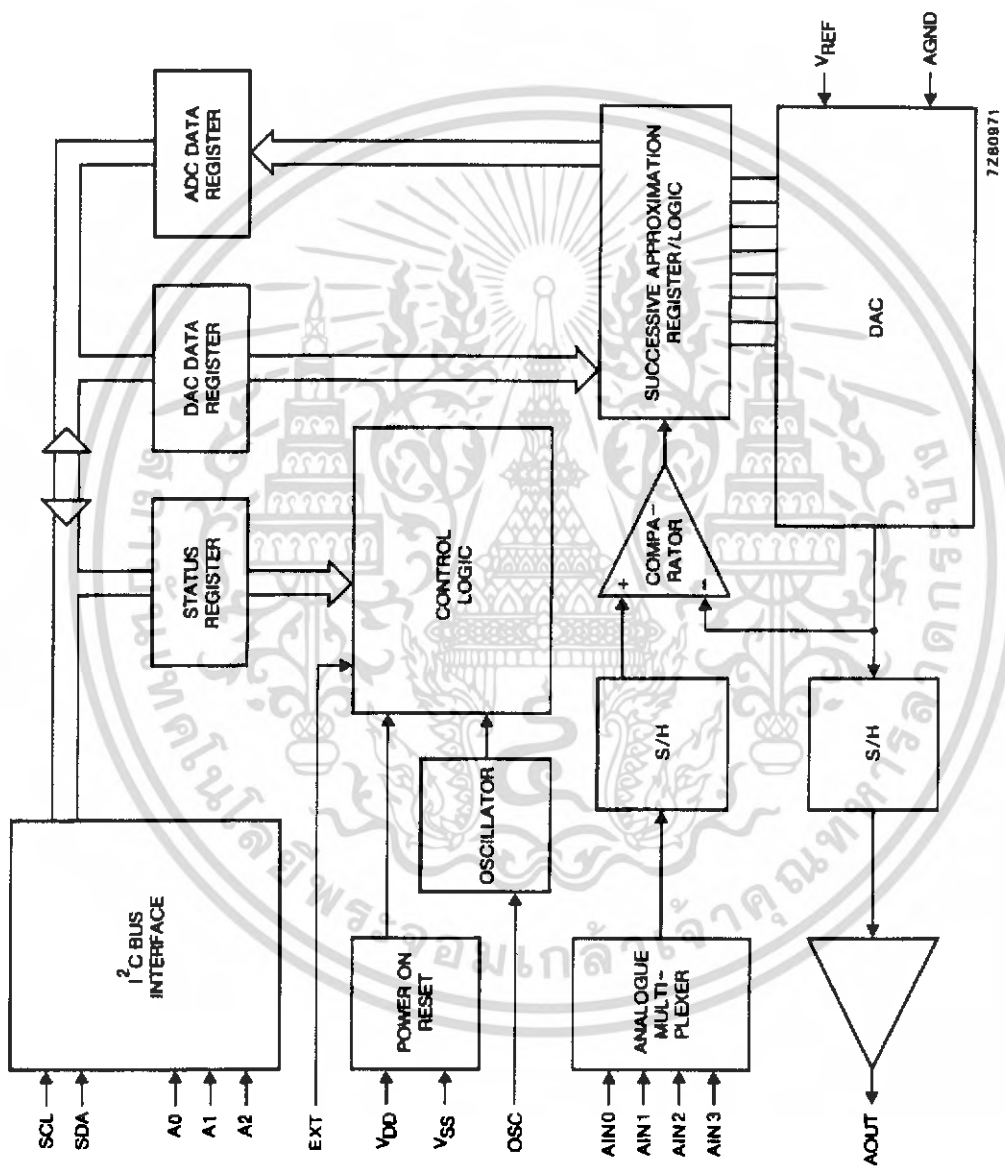


Fig.1 Block diagram.

6 PINNING

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

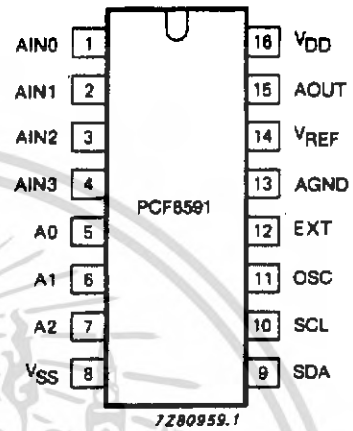


Fig.2 Pinning diagram.

7 FUNCTIONAL DESCRIPTION

7.1 Addressing

Each PCF8591 device in an I²C-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 I²C-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, 15 and 16).

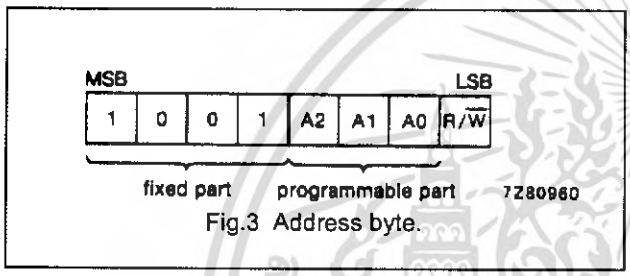


Fig.3 Address byte.

7.2 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 analog output, and for programming the analog inputs as single-ended or differential inputs. The lower nibble selects one of the analog 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 analog 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 analog 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 analog output is switched to a high-impedance state.

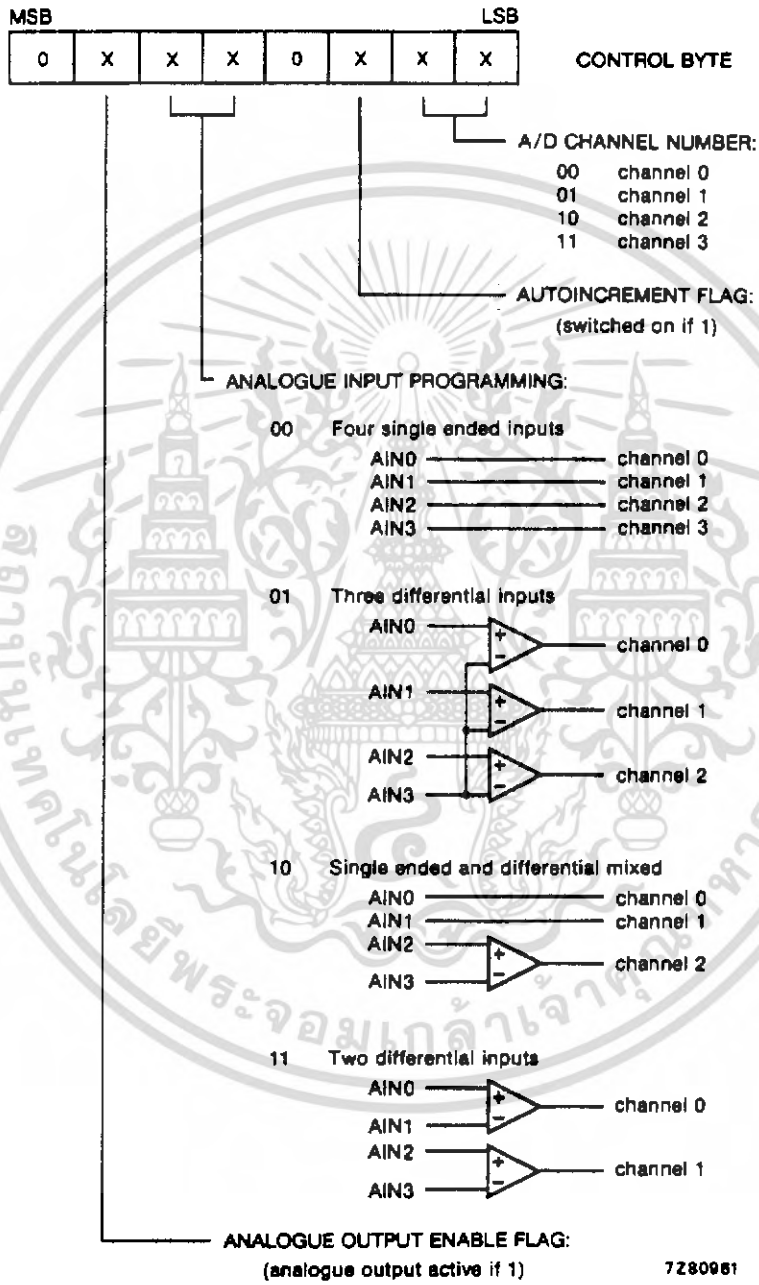


Fig.4 Control byte.

7.3 D/A conversion

The third byte sent to a PCF8591 device is stored in the DAC data register and is converted to the corresponding analog voltage using the on-chip D/A converter. This D/A converter consists of a resistor divider chain connected to the external reference voltage with 256 taps and selection switches. The tap-decoder switches one of these taps to the DAC output line (see Fig.5).

The analog output voltage is buffered by an auto-zeroed unity gain amplifier. This buffer amplifier may be switched on or off by setting the analog output enable flag of the control register. In the active state the output voltage is held until a further data byte is sent.

The on-chip D/A converter is also used for successive approximation A/D conversion. In order to release the DAC for an A/D conversion cycle the unity gain amplifier is equipped with a track and hold circuit. This circuit holds the output voltage while executing the A/D conversion.

The output voltage supplied to the analog output AOUT is given by the formula shown in Fig.6. The waveforms of a D/A conversion sequence are shown in Fig.7.

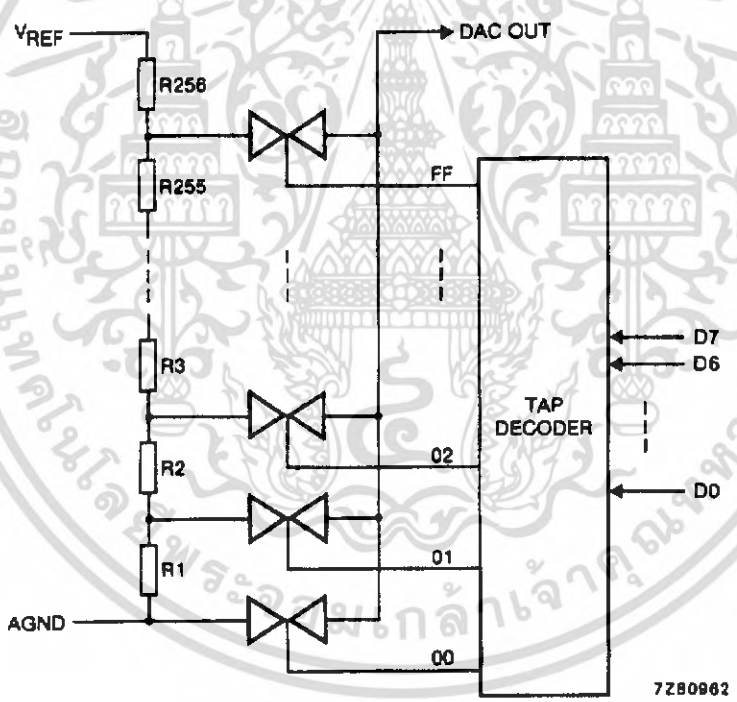


Fig.5 DAC resistor divider chain.

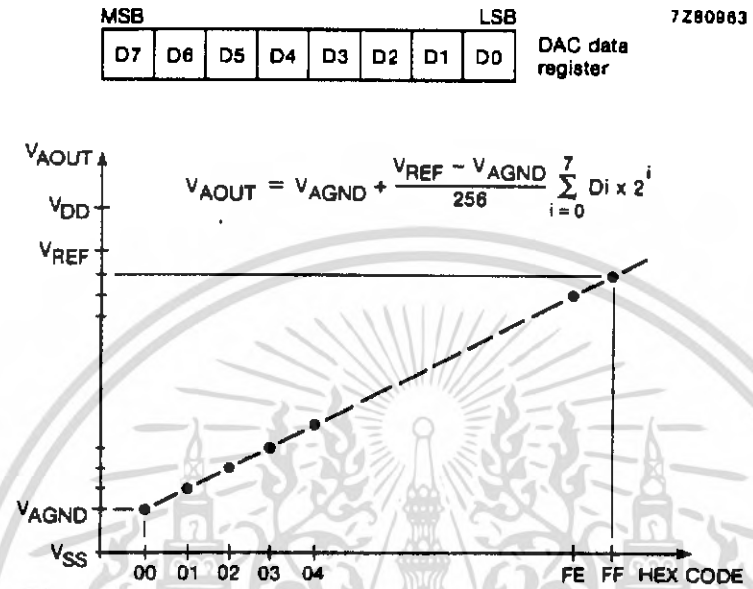


Fig.6 DAC data and DC conversion characteristics.

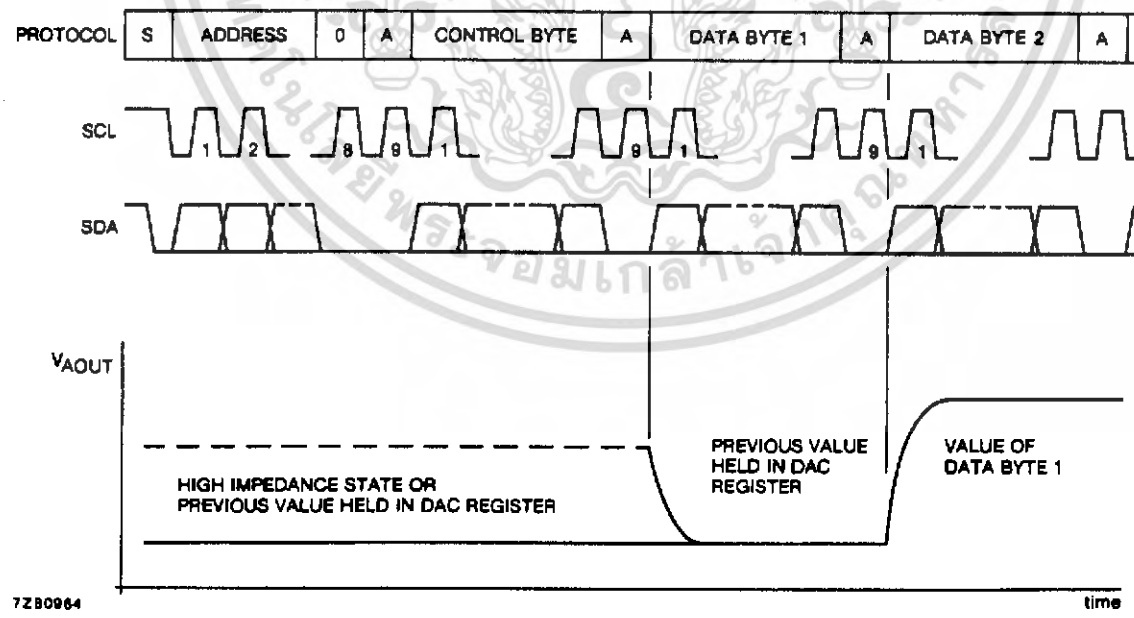


Fig.7 D/A conversion sequence.

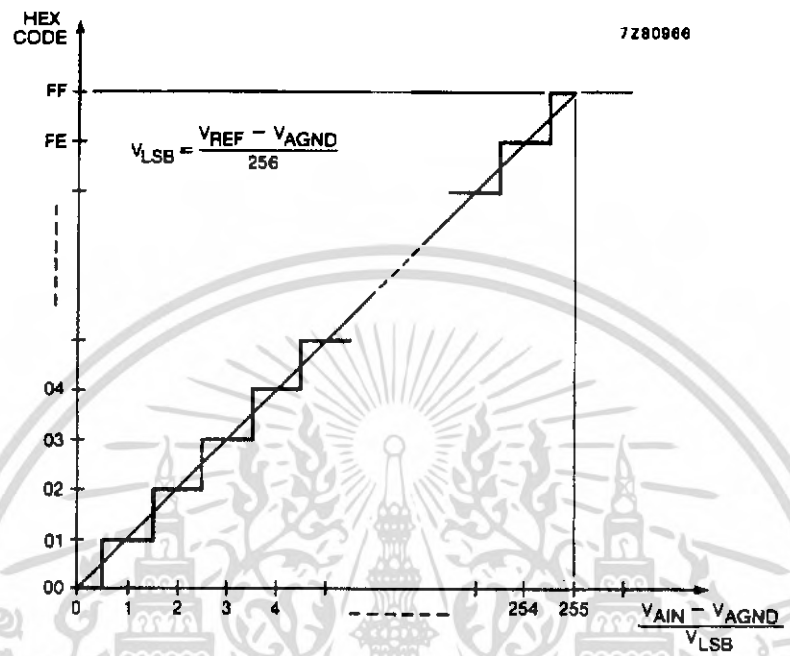


Fig.9 A/D conversion characteristics of single-ended inputs.

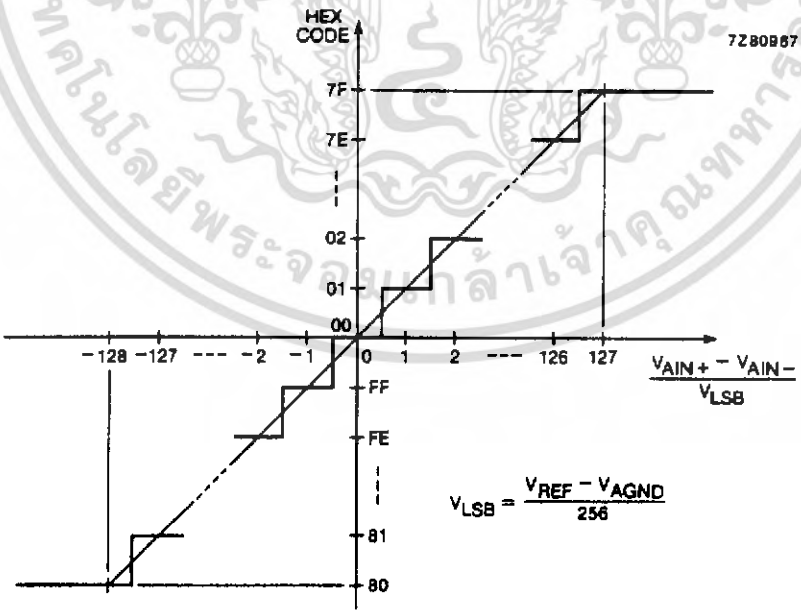


Fig.10 A/D conversion characteristics of differential inputs.

7.5 Reference voltage

For the D/A and A/D conversion either a stable external voltage reference or the supply voltage has to be applied to the resistor divider chain (pins V_{REF} and AGND). The AGND pin has to be connected to the system analog ground and may have a DC off-set with reference to V_{SS}.

A low frequency may be applied to the V_{REF} and AGND pins. This allows the use of the D/A converter as a one-quadrant multiplier; see Chapter 15 and Fig.6.

The A/D converter may also be used as a one or two quadrant analog divider. The analog input voltage is divided by the reference voltage. The result is converted to a binary code. In this application the user has to keep the reference voltage stable during the conversion cycle.

7.6 Oscillator

An on-chip oscillator generates the clock signal required for the A/D conversion cycle and for refreshing the auto-zeroed buffer amplifier. When using this oscillator the EXT pin has to be connected to V_{SS}. At the OSC pin the oscillator frequency is available.

If the EXT pin is connected to V_{DD} the oscillator output OSC is switched to a high-impedance state allowing the user to feed an external clock signal to OSC.

8 CHARACTERISTICS OF THE I²C-BUS

The I²C-bus is for bidirectional, two-line communication between different ICs or modules. The two lines are a serial data line (SDA) and a serial clock line (SCL). Both lines must be connected to a positive supply via a pull-up resistor. Data transfer may be initiated only when the bus is not busy.

8.1 Bit transfer

One data bit is transferred during each clock pulse. The data on the SDA line must remain stable during the HIGH period of the clock pulse as changes in the data line at this time will be interpreted as a control signal.

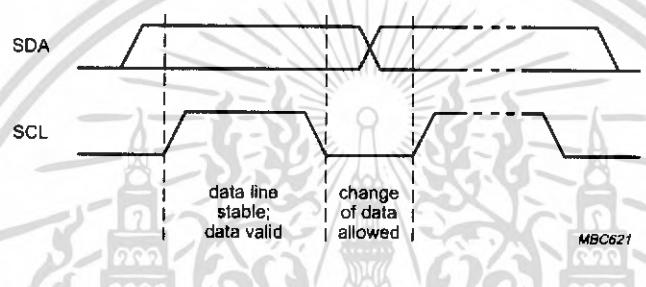


Fig.11 Bit transfer.

8.2 Start and stop conditions

Both data and clock lines remain HIGH when the bus is not busy. A HIGH-to-LOW transition of the data line, while the clock is HIGH, is defined as the start condition (S). A LOW-to-HIGH transition of the data line while the clock is HIGH, is defined as the stop condition (P).

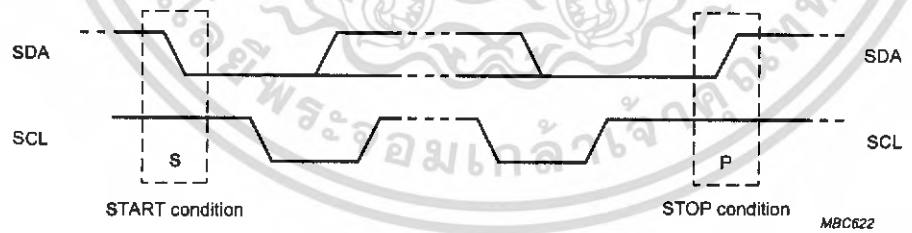


Fig.12 Definition of START and STOP condition.

8.3 System configuration

A device generating a message is a 'transmitter', a device receiving a message is the 'receiver'. The device that controls the message is the 'master' and the devices which are controlled by the master are the 'slaves'.

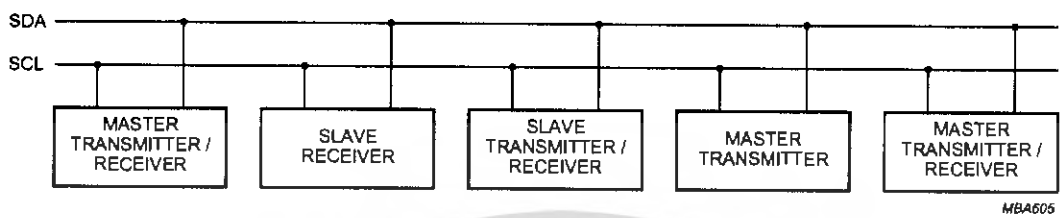


Fig.13 System configuration.

8.4 Acknowledge

The number of data bytes transferred between the start and stop conditions from transmitter to receiver is not limited. Each data byte of eight bits is followed by one acknowledge bit. The acknowledge bit is a HIGH level put on the bus by the transmitter whereas the master also generates an extra acknowledge related clock pulse. A slave receiver which is addressed must generate an acknowledge after the reception of each byte. Also a master must generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse. A master receiver must signal an end of data to the transmitter by **not** generating an acknowledge on the last byte that has been clocked out of the slave. In this event the transmitter must leave the data line HIGH to enable the master to generate a stop condition.

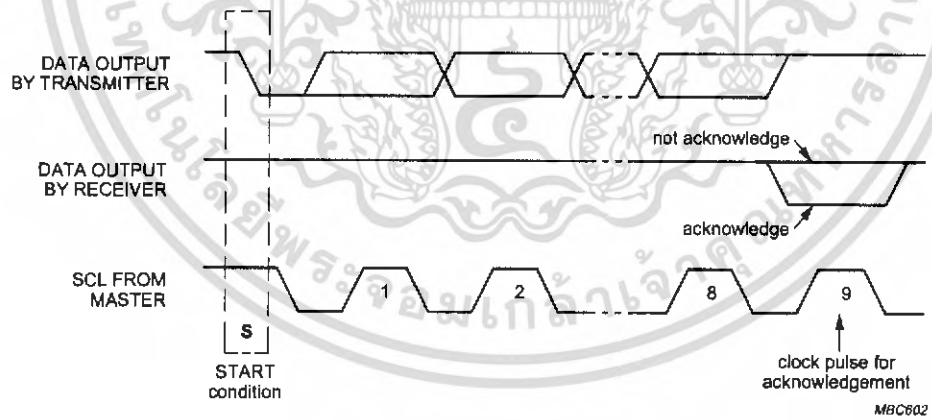


Fig.14 Acknowledgement on the I^2C-bus.

8.5 I²C-bus protocol

After a start condition a valid hardware address has to be sent to a PCF8591 device. The read/write bit defines the direction of the following single or multiple byte data transfer. For the format and the timing of the start condition (S), the stop condition (P) and the acknowledge bit (A) refer to the I²C-bus characteristics. In the write mode a data transfer is terminated by sending either a stop condition or the start condition of the next data transfer.

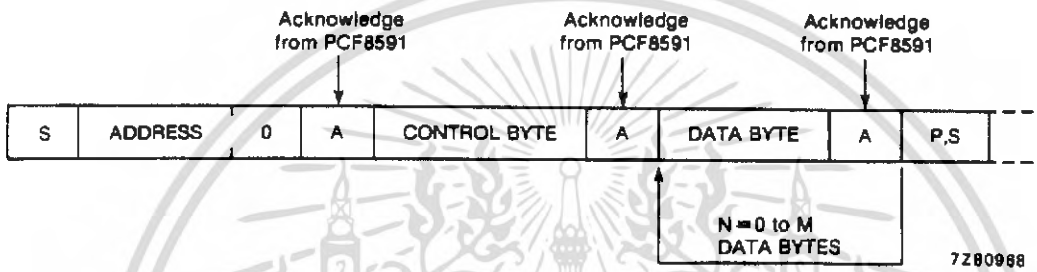


Fig.15 Bus protocol for write mode, D/A conversion.

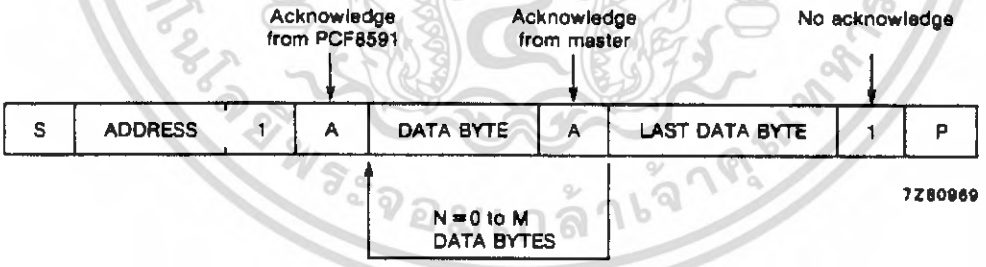


Fig.16 Bus protocol for read mode, A/D conversion.

9 LIMITING VALUES

In accordance with the Absolute Maximum Rating System (IEC 134).

SYMBOL	PARAMETER	MIN.	MAX.	UNIT
V_{DD}	supply voltage (pin 16)	−0.5	+8.0	V
V_I	input voltage (any input)	−0.5	V_{DD} + 0.5	V
I_I	DC input current	−	±10	mA
I_O	DC output current	−	±20	mA
I_{DD}, I_{SS}	V_{DD} or V_{SS} current	−	±50	mA
P_{tot}	total power dissipation per package	−	300	mW
P_O	power dissipation per output	−	100	mW
T_{amb}	operating ambient temperature	−40	+85	°C
T_{stg}	storage temperature	−65	+150	°C

10 HANDLING

Inputs and outputs are protected against electrostatic discharge in normal handling. However, to be totally safe, it is desirable to take precautions appropriate to handling MOS devices. Advice can be found in Data Handbook IC12 under *"Handling MOS Devices"*.

11 DC CHARACTERISTICS

V_{DD} = 2.5 V to 6 V; V_{SS} = 0 V; T_{amb} = –40 °C to +85 °C unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Supply						
V_{DD}	supply voltage (operating)		2.5	–	6.0	V
I_{DD}	supply current					
	standby	$V_I = V_{SS}$ or V_{DD}; no load	–	1	15	µA
	operating, AOUT off	f_{SCL} = 100 kHz	–	125	250	µA
	operating, AOUT active	f_{SCL} = 100 kHz	–	0.45	1.0	mA
V_{POR}	Power-on reset level	note 1	0.8	–	2.0	V
Digital inputs/output: SCL, SDA, A0, A1, A2						
V_{IL}	LOW level input voltage		0	–	$0.3 \times V_{DD}$	V
V_{IH}	HIGH level input voltage		$0.7 \times V_{DD}$	–	V_{DD}	V
I_L	leakage current					
	A0, A1, A2	$V_I = V_{SS}$ to V_{DD}	–250	–	+250	nA
	SCL, SDA	$V_I = V_{SS}$ to V_{DD}	–1	–	+1	µA
C_i	input capacitance		–	–	5	pF
I_{OL}	LOW level SDA output current	V_{OL} = 0.4 V	3.0	–	–	mA
Reference voltage inputs						
V_{REF}	reference voltage	$V_{REF} > V_{AGND}$; note 2	V_{SS} + 1.6	–	V_{DD}	V
V_{AGND}	analog ground voltage	$V_{REF} > V_{AGND}$; note 2	V_{SS}	–	V_{DD} – 0.8	V
I_{LI}	input leakage current		–250	–	+250	nA
R_{REF}	input resistance	pins V_{REF} and AGND	–	100	–	kΩ
Oscillator: OSC, EXT						
I_{LI}	input leakage current		–	–	250	nA
f_{OSC}	oscillator frequency		0.75	–	1.25	MHz

Notes

1. The power on reset circuit resets the I²C-bus logic when V_{DD} is less than V_{POR}.
2. A further extension of the range is possible, if the following conditions are fulfilled:

$$\frac{V_{REF} + V_{AGND}}{2} \geq 0.8V, V_{DD} - \frac{V_{REF} + V_{AGND}}{2} \geq 0.4V$$

12 D/A CHARACTERISTICS

V_{DD} = 5.0 V; V_{SS} = 0 V; V_{REF} = 5.0 V; V_{AGND} = 0 V; R_L = 10 kΩ; C_L = 100 pF; T_{amb} = –40 °C to +85 °C unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Analog output						
V_{OA}	output voltage	no resistive load	V_{SS}	–	V_{DD}	V
		R_L = 10 kΩ	V_{SS}	–	0.9 × V_{DD}	V
I_{LO}	output leakage current	AOUT disabled	–	–	250	nA
Accuracy						
OS_e	offset error	T_{amb} = 25 °C	–	–	50	mV
L_e	linearity error		–	–	±1.5	LSB
G_e	gain error	no resistive load	–	–	1	%
t_{DAC}	settling time	to ½ LSB full scale step	–	–	90	µs
f_{DAC}	conversion rate		–	–	11.1	kHz
SNRR	supply noise rejection ratio	f = 100 Hz; V_{DDN} = 0.1 × V_{PP}	–	40	–	dB

13 A/D CHARACTERISTICS

V_{DD} = 5.0 V; V_{SS} = 0 V; V_{REF} = 5.0 V; V_{AGND} = 0 V; R_S = 10 kΩ; T_{amb} = –40 °C to +85 °C unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Analog inputs						
V_{IA}	analog input voltage		V_{SS}	–	V_{DD}	V
I_{LIA}	analog input leakage current		–	–	100	nA
C_{IA}	analog input capacitance		–	10	–	pF
C_{ID}	differential input capacitance		–	10	–	pF
V_{IS}	single-ended voltage	measuring range	V_{AGND}	–	V_{REF}	V
V_{ID}	differential voltage	measuring range; $V_{FS} = V_{REF} - V_{AGND}$	$\frac{-V_{FS}}{2}$	–	$\frac{+V_{FS}}{2}$	V
Accuracy						
OS_e	offset error	T_{amb} = 25 °C	–	–	20	mV
L_e	linearity error		–	–	±1.5	LSB
G_e	gain error		–	–	1	%
GS_e	small-signal gain error	ΔV_i = 16 LSB	–	–	5	%
CMRR	common-mode rejection ratio		–	60	–	dB
SNRR	supply noise rejection ratio	f = 100 Hz; V_{DDN} = 0.1 × V_{PP}	–	40	–	dB
t_{ADC}	conversion time		–	–	90	µs
f_{ADC}	sampling/conversion rate		–	–	11.1	kHz

หนังสือและเอกสารอ้างอิง

1. ยุทธนา ลีลาศวัฒนกุล, "เริ่มต้นการเขียนโปรแกรมด้วยภาษา C++", หจก. ไทยเจริญการพิมพ์ 2547
2. ยุทธนา ลีลาศวัฒนกุล , "คู่มือการเขียนโปรแกรมและการใช้งาน Visual C++ 6.0 ฉบับโปรแกรมเมอร์" , สำนักพิมพ์ อินโฟเพรส 25444
3. ธีรบูลย์ หล่อวิเชียรรุ่ง , นคร ภักดีชาติ , ชัยวัฒน์ ลิ้มพรจิตรวิไล , "ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษาซี" , บริษัท อินโนเวทีฟ เอกเพอริเมนท์ จำกัด 2521
4. www.alldatasheet.com
5. "Ethernet IO User's Manual", Design Gateway Co., Ltd.
6. "Ethernet IO Programming Manual", Design Gateway Co., Ltd.