นาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า

CONTROL ELECTRICAL EQUIPMENTS BY ALARM CLOCK

โดย

นางสาวจุฑาเพชร เวชรังษี

นางสาวน้ำฝน ศักดิ์พิมานพร

นายสุทธิพร สุดยอด

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ภาควิชา
วิศวกรรมโทรคมนาคม

ผ่านการตรวจชั้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

นาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า

CONTROL ELECTRICAL EQUIPMENTS BY ALARM CLOCK

โดย

นางสาวจุฑาเพชร	เวชรังษี	46010124
นางสาวน้ำฝน	ศักดิ์พิมานพร	46010361
นายสุทธิพร	สุดยอด	46010852

อาจารย์ที่ปรึกษา

ผศ.ดร. พรชัย ทรัพย์นิธิ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **นาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า**

CONTROL ELECTRICAL EQUIPMENTS BY ALARM CLOCK

ผู้จัดทำ

1. นางสาวอุฑาเพชร เวชรังษี 46010124

2. นางสาวน้ำฝน ศักดิ์พิมานพร 46010361

3. นายสุทธิพร สุดยอด 46010852

P. Supa ..... อาจารย์ที่ปรึกษา

(ผศ.ดร.พรชัย ทรัพย์นิธิ)

นาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า
CONTROL ELECTRICAL EQUIPMENTS BY ALARM CLOCK

โดย นางสาวจุฑาเพชร เวชรังษี 46010124
นางสาวน้ำฝน ศักดิ์พิมานพร 46010361
นายสุทธิพร สุดยอด 46010852

อาจารย์ที่ปรึกษา ผศ.ดร.พรชัย ทรัพย์นิธิ

บทคัดย่อ

โครงงานนี้นำเสนอระบบควบคุมสถานะอุปกรณ์ไฟฟ้าที่ใช้ในชีวิตประจำวันภายหลังการตื่นนอน เป็นการควบคุมผ่านนาฬิกาปลุกที่สร้างขึ้นจากไมโครคอนโทรลเลอร์ ซึ่งเชื่อมต่อกับแผงควบคุมอุปกรณ์ไฟฟ้าแบบไร้สาย โดยที่สามารถเลือกและตั้งเวลาเปิด - ปิดอุปกรณ์ไฟฟ้าได้ตามความต้องการ เพื่อวัตถุประสงค์ในการอำนวยความสะดวกแก่ผู้ใช้งาน

ABTRACT

This project is created to control the status of electrical equipments after wake up. The alarm clock implemented using MCS-51controlled to the main equipment wireless control board. The board is adjustable and can set the time.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงได้ด้วยดี ด้วยความกรุณาเป็นอย่างดียิ่งจาก ผศ.ดร.พรชัย ทรัพย์นิธิ ที่กรุณาให้ทั้งความรู้ ความช่วยเหลือ และคำปรึกษาที่ดีเสมอมา รวมถึงการอนุเคราะห์สถานที่ เครื่องมือ และอุปกรณ์ต่างๆ สำหรับการดำเนินงานในครั้งนี้

สุดท้ายนี้ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และขอขอบคุณเพื่อนๆ ทุกคน ที่คอยช่วยเหลือ ให้การสนับสนุน และให้กำลังใจที่ดีเสมอมา จนการจัดทำปริญญานิพนธ์ครั้งนี้สำเร็จลงได้ด้วยดี

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

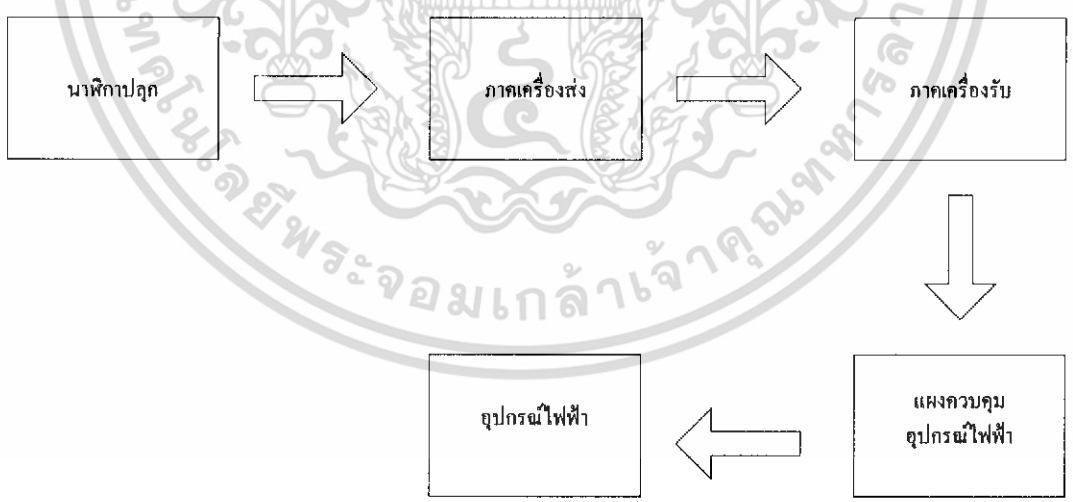
สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาของโครงงาน

ในปัจจุบัน การดำเนินชีวิตของมนุษย์ตั้งแต่ตื่นนอนจนเข้านอนอีกครั้งนั้นเต็มไปด้วยความเร่งรีบ การบริหารเวลาที่มีอยู่อย่างจำกัดให้เกิดประโยชน์สูงสุดจึงเป็นสิ่งจำเป็น โครงงาน "นาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า" จึงเป็นอีกแนวทางหนึ่งที่จะตอบสนองความต้องการในส่วนนี้ได้

โครงงาน "นาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า" เป็นการสร้างนาฬิกาปลุกขึ้นจากไมโครคอน - โทรลเลอร์ซึ่งสามารถแสดงเวลาและปฏิทิน 100 ปีได้ โดยนำมาประยุกต์เข้ากับชุดควบคุมอุปกรณ์ไฟฟ้าที่สร้างขึ้น ทำให้สามารถควบคุมอุปกรณ์ไฟฟ้าได้หลายตัวพร้อมกัน โดยสามารถเลือกได้ว่าจะสั่งงานให้แก่อุปกรณ์ตัวใด และในการสั่งงานนั้นอาจเป็นการเปลี่ยนสถานะจากปิดให้เปิด หรือ จากเปิดให้ปิดได้ โดยผ่านนาฬิกาปลุกที่ตั้งอยู่ภายในห้องนอน ยกตัวอย่างเช่น การสั่งให้เครื่องชงกาแฟทำงานในตอนเช้าโดยที่ไม่ต้องไปกดสวิตซ์ที่ตัวเครื่องซึ่งถือเป็นการช่วยประหยัดเวลา หรือ การสั่งงานหลอดไฟหน้าบ้านที่ก่อนนอนได้เปิดทิ้งไว้เพื่อให้แสงสว่างกับตัวบ้าน เมื่อรุ่งเช้าก็สามารถปิดไฟหน้าบ้านได้พร้อมกับอุปกรณ์ไฟฟ้าอื่นๆ จึงเหมาะสำหรับบุคคลที่มักจะลืมปิดสวิตซ์อุปกรณ์ไฟฟ้าหรือแม้กระทั่งผู้ป่วยและผู้สูงอายุที่เคลื่อนไหวร่างกายได้ยาก ด้วยตัวอย่างเหล่านี้ ชี้ให้เห็นว่านี่เป็นอีกหนึ่งทางเลือกที่ไม่เพียงแต่จะอำนวยความสะดวกเท่านั้น แต่ยังเป็นการช่วยประหยัดค่าไฟฟ้าอีกด้วย ดังมีโครงสร้างของโครงงานแสดงดังรูปที่ 1.1



รูปที่ 1.1 แสดงระบบที่ใช้ในการควบคุมอุปกรณ์ไฟฟ้า

1.2 โครงสร้างของโครงงานสามารถแบ่งเป็น 4 ส่วนหลักๆ ดังนี้

1. นาฬิกาปลุก
2. ภาคเครื่องส่ง
3. ภาคเครื่องรับ
4. แผงควบคุมอุปกรณ์ไฟฟ้า

การควบคุมอุปกรณ์ไฟฟ้าในลักษณะการควบคุมแบบเปิด - ปิด เป็นการส่งข้อมูลขนาดเล็กและไม่ต้องการความเร็วสูงมากนัก ผู้ทำการทดลองจึงเลือกใช้การสื่อสารแบบไร้สาย (TRW) เป็นตัวกลางในการเชื่อมต่อระหว่างภาคเครื่องรับและเครื่องส่งเพราะจะสามารถส่งข้อมูลไปได้ในระยะไกล และง่ายต่อการติดตั้ง เคลื่อนย้ายอุปกรณ์ โดยใช้ภาษาแอสเซมบลีในการเขียนโปรแกรม

1.3 ขั้นตอนการดำเนินงาน

ภาคการศึกษาที่ 1/2549 เป็นการนำเอานาฬิกาปลุกมาประยุกต์เข้ากับชุดควบคุมอุปกรณ์ไฟฟ้าที่สร้างขึ้น โดยที่ภาคเครื่องส่งจะส่งข้อมูลแบบอนุกรมไปยังภาคเครื่องรับ เพื่อที่จะส่งสัญญาณไปควบคุมอุปกรณ์ไฟฟ้าตามต้องการ

ภาคการศึกษาที่ 2/2549 ได้ดำเนินการสร้างนาฬิกาปลุกและชุดควบคุมอุปกรณ์ไฟฟ้าที่สร้างจากไมโครคอนโทรลเลอร์ซึ่งสามารถควบคุมอุปกรณ์ไฟฟ้าได้ 2 อุปกรณ์ โดยที่การเชื่อมต่อระหว่างภาคเครื่องส่งและเครื่องรับเป็นการส่งสัญญาณแบบไร้สายในวงจรมอดูเลเตอร์แบบ TRW 2.4 GHz และสามารถเลือกได้ว่าต้องการให้อุปกรณ์ไฟฟ้าตัวใดมีสถานะเปิดหรือปิด

บทที่ 2
ทฤษฎีและหลักการ

2.1 ทฤษฎีของไมโครคอนโทรลเลอร์ตระกูล MCS-51

โครงสร้างและพื้นฐานทั่วไปของไมโครคอนโทรลเลอร์ที่จำเป็นต้องทำการศึกษาเพื่อนำไปใช้ในโครงงานนาฬิกาปลุกควบคุมอุปกรณ์ไฟฟ้า มีดังต่อไปนี้

2.1.1 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ตระกูล MCS-51

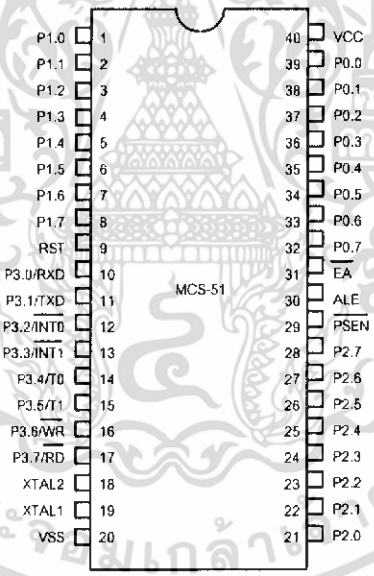
คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีดังต่อไปนี้

- เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต
- มีวงจรออสซิลเลเตอร์และวงจรผลิตสัญญาณนาฬิกาภายในไอซี
- มีขาสัญญาณอินพุทและเอาท์พุทจำนวน 32 บิต
- สามารถเชื่อมต่อกับหน่วยความจำข้อมูลภายนอก (External Data Memory) โดยการอ้างตำแหน่งแอดเดรสได้ถึง 64 K
- สามารถเชื่อมต่อหน่วยความจำโปรแกรมภายนอก (External Program Memory) โดยการอ้างตำแหน่งแอดเดรสได้ถึง 64 K
- มีหน่วยความจำข้อมูลภายในตัว (On-Chip Data Memory) ขนาด 128 ไบต์ โดยเฉพาะหมายเลข 8032 และ 8052 จะมีหน่วยความจำในส่วนนี้ถึง 256 ไบต์
- มีหน่วยความจำโปรแกรมภายในตัว (On-Chip Program Memory) ขนาด 4 K โดยเฉพาะหมายเลข 8052 จะมีหน่วยความจำในส่วนนี้ถึง 8 K สำหรับหมายเลข 8031 และ 8032 จะไม่มีหน่วยความจำในส่วนนี้
- หน่วยความจำข้อมูลภายในบางส่วนสามารถเข้าถึงข้อมูลระดับบิตได้ด้วย ทำให้สามารถควบคุมหรือตรวจสอบสถานะบิตได้ง่าย ส่งผลให้สามารถเขียนโปรแกรมได้ง่ายขึ้น
- มีไทม์เมอร์/เคาน์เตอร์ (Timer/Counter) ขนาด 16 บิตจำนวน 2 ตัว โดยเฉพาะหมายเลข 8032 หรือ 8052 จะมีไทม์เมอร์/เคาน์เตอร์จำนวน 3 ตัว
- สามารถทำการอินเตอร์รัพท์ได้จากแหล่งกำเนิด 5 แหล่ง โดยเฉพาะหมายเลข 8032 และ 8052 จะทำการอินเตอร์รัพท์ได้จากแหล่งกำเนิด 6 แหล่ง พร้อมทั้งยังสามารถจัดลำดับความสำคัญของการอินเตอร์รัพท์ได้ 2 ระดับ
- มีพอร์ตสื่อสารอนุกรมภายในตัวเอง ซึ่งการทำงานจะเป็นแบบฟูลดูเพลิกซ์ (Full Duplex)
- มีคำสั่งในการคำนวณทางคณิตศาสตร์และทางตรรกศาสตร์
- คำสั่งส่วนใหญ่จะใช้เวลาในการประมวลผลเพียง 1 ไมโครวินาที เมื่อใช้คริสตอลออสซิลเลเตอร์ความถี่ 12 MHz
- ต้องการแหล่งจ่ายไฟกระแสตรงแรงดัน 5 V

ตารางที่ 2.1 แสดงคุณสมบัติของไมโครคอนโทรลเลอร์แต่ละหมายเลขในตระกูล MCS-51

ชื่อหมายเลข	หน่วยความจำภายใน		จำนวนไทม์เมอร์/ เคาน์เตอร์	จำนวน อินเตอร์รัพท์
	เก็บโปรแกรม	เก็บข้อมูล		
8052AH	8K x 8 ROM	256 x 8 RAM	3 x 16-Bit	6
8051AH	4K x 8 ROM	128 x 8 RAM	2 x 16-Bit	5
8051	4K x 8 ROM	128 x 8 RAM	2 x 16-Bit	5
8032AH	ไม่มี	256 x 8 RAM	2 x 16-Bit	6
8031AH	ไม่มี	128 x 8 RAM	2 x 16-Bit	5
8031	ไม่มี	128 x 8 RAM	2 x 16-Bit	5
8751H	4K x 8 EPROM	128 x 8 RAM	2 x 16-Bit	5
8751H-12	4K x 8 EPROM	128 x 8 RAM	2 x 16-Bit	5

2.1.2 โครงสร้างภายนอกของไมโครคอนโทรลเลอร์ตระกูลของ MCS-51



รูปที่ 2.1 แสดงการจัดตำแหน่งขาต่างๆ ของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกหมายเลขจะมีตำแหน่งขาพื้นฐานที่เหมือนกันดังแสดงในรูปที่ 2.1 สำหรับหน้าที่การทำงานของแต่ละขามีดังนี้

- ขา V_{CC} เป็นขาสำหรับป้อนแรงดันไฟเลี้ยง +5 V
- ขา V_{SS} เป็นขา ground

- ขาพอร์ต 0 (Port 0) มี 8 ขาได้แก่ขา P0.0-P0.7 เป็นขาพอร์ตอินพุทและเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุทพอร์ตต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้ขาพอร์ตเหล่านั้นอยู่ในสภาวะปล่อยลอย ซึ่งในสถานะนี้เองที่สามารถนำใช้เป็นพอร์ตอินพุทอิมพีแดนซ์สูงได้ นอกจากพอร์ตนี้จะใช้งานเป็นพอร์ตอินพุทและเอาท์พุทแล้ว ยังถูกใช้งานในการติดต่อกับหน่วยความจำภายนอกด้วย โดยทำหน้าที่ในการกำหนดแอดเดรสไบต์ต่ำ (A0–A7) ซึ่งจะใช้งานเป็นแบบมัลติเพล็กซ์กับการรับส่งข้อมูลขนาด 8 บิต (D0–D7)
- ขาพอร์ต 1 (Port 1) มี 8 ขาได้แก่ขา P1.0-P1.7 เป็นขาพอร์ตอินพุทและเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุทพอร์ตต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้เป็นพอร์ตอินพุท นอกจากนี้สำหรับหมายเลข 8032 และ 8052 ขาพอร์ต P1.0 และ P1.1 จะถูกนำมาใช้งานเป็นขา T2 และ T2EX ตามลำดับด้วย
- ขาพอร์ต 2 (Port 2) มี 8 ขาได้แก่ขา P2.0-P2.7 เป็นขาพอร์ตอินพุทและเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นพอร์ตอินพุทต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้เป็นพอร์ตอินพุท นอกจากพอร์ตนี้จะใช้งานเป็นพอร์ตอินพุทและเอาท์พุทแล้ว ยังถูกใช้งานในการติดต่อกับหน่วยความจำภายนอกด้วย โดยทำหน้าที่ในการกำหนดแอดเดรสไบต์สูง (A8-A15)
- ขาพอร์ต 3 (Port 3) มี 8 ขาได้แก่ขา P3.0-P3.7 เป็นขาพอร์ตอินพุทและเอาท์พุทแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นพอร์ตอินพุทต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้เป็นพอร์ตอินพุท นอกจากพอร์ตนี้จะใช้งานเป็นพอร์ตอินพุทและเอาท์พุทแล้ว ยังถูกใช้งานในหน้าที่พิเศษต่างๆ ดังแสดงในตารางที่ 2.2 ด้วย

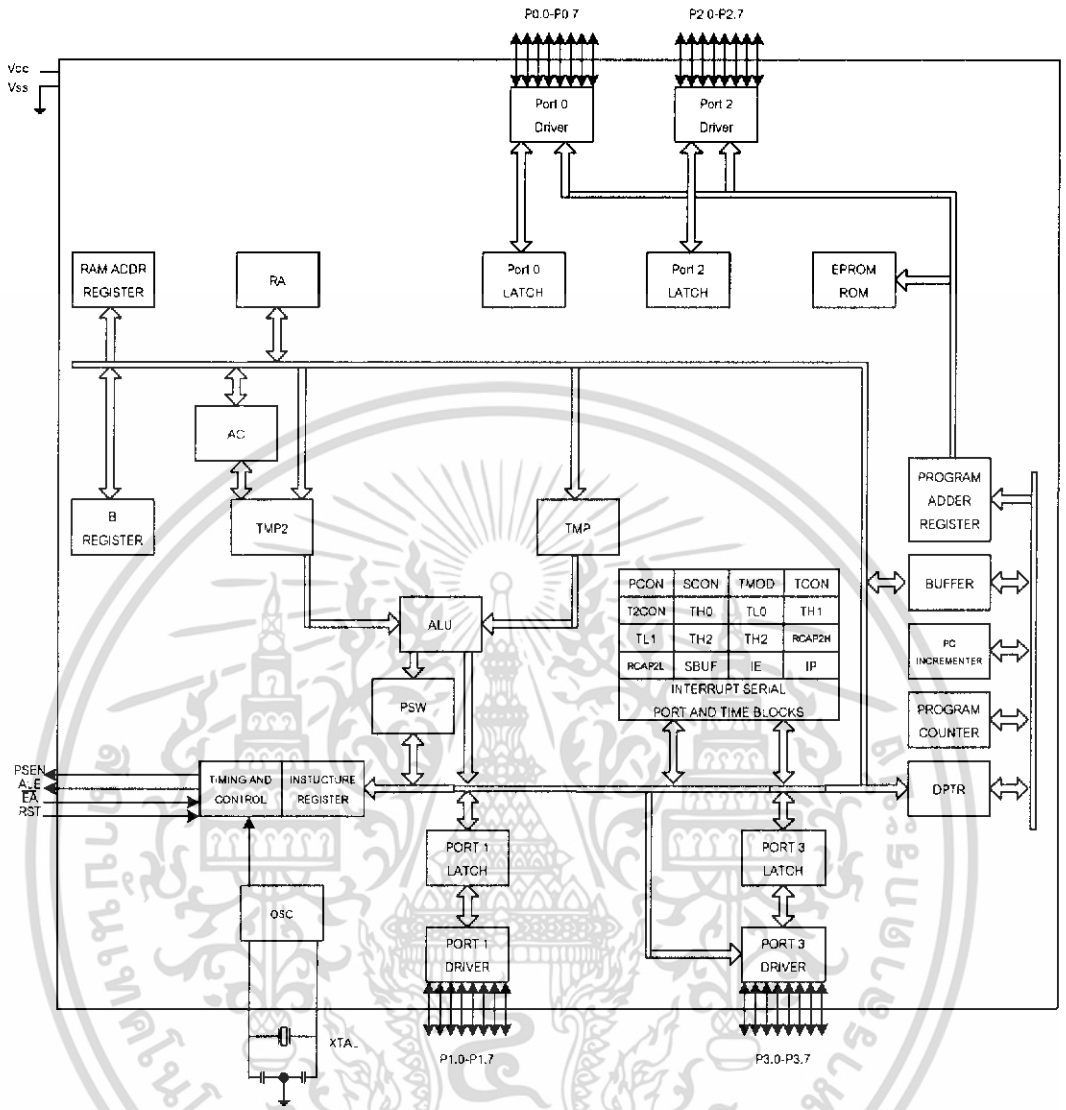
ตารางที่ 2.2 แสดงหน้าที่พิเศษของแต่ละขาของพอร์ต 3

ขาพอร์ต	หน้าที่พิเศษ
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

- ขา Reset (RST) ใช้สำหรับทำการ Reset การทำงานของไมโครคอนโทรลเลอร์ โดยการ Reset ต้องคงสถานะเป็น 1 นานอย่างน้อย 2 แมชชีนไซเคิล ในขณะที่ออสซิลเลเตอร์ยังคงทำงานอยู่
- ขา $\overline{\text{ALE/PROG}}$ เป็นขาสัญญาณเพื่อทำหน้าที่ควบคุมการแลตช์ (Latch) ค่าตำแหน่งแอดเดรสไบต์ต่ำ (Address Latch Enable) เมื่อต้องการติดต่อกับหน่วยความจำภายนอก นอกจากนี้ขานี้ยังทำหน้าที่เป็นอินพุทรับพัลส์ในการโปรแกรม (Program Pulse Input) ในส่วนของหน่วยความจำ EPROM สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-51 ที่มีหน่วยความจำโปรแกรมภายในเป็น EPROM
- ขา $\overline{\text{PSEN}}$ (Program Store Enable) ทำหน้าที่เป็นสัญญาณสโตรบเพื่ออ่านคำสั่งจากหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ทำการประมวลผลคำสั่งจากหน่วยความจำภายนอก ขานี้จะส่งสัญญาณสโตรบจำนวน 2 ครั้งในแต่ละแมชชีนไซเคิล แต่ในขณะที่ติดต่อกับหน่วยความจำภายนอกจะไม่มีการส่งสัญญาณสโตรบแต่อย่างใด
- ขา $\overline{\text{EA}}$/VPP (External Access Enable/VPP) เป็นขาสำหรับการเลือกใช้หน่วยความจำโปรแกรมจากภายในหรือจากภายนอก โดยถ้าขานี้มีสถานะเป็น "0" จะหมายถึง ให้ไมโครคอนโทรลเลอร์รับคำสั่งจากหน่วยความจำภายนอกที่ตำแหน่งแอดเดรส 0-0FFFH (0-1FFFH ถ้าเป็นหมายเลข 8052) อย่างไรก็ตามถ้าบิตป้องกัน (Security Bit) ในหน่วยความจำ EPROM ถูกโปรแกรมไว้ ไมโครคอนโทรลเลอร์จะไม่รับคำสั่งจากหน่วยความจำภายนอกเลย นอกจากนี้ ขานี้ยังทำหน้าที่รับแรงดันไฟสำหรับการโปรแกรม (V_{pp}) ขนาด 21 V เพื่อใช้ระหว่างการโปรแกรม EPROM
- ขา XTAL1 และขา XTAL2 เป็นขาอินพุทและเอาท์พุทของวงจร Inverting Oscillator Amplifier สำหรับใช้ต่อร่วมกับคริสตอลออสซิลเลเตอร์ภายนอก

2.1.3 โครงสร้างภายในของ MCS-51

โครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51 แสดงดังรูปที่ 2.2

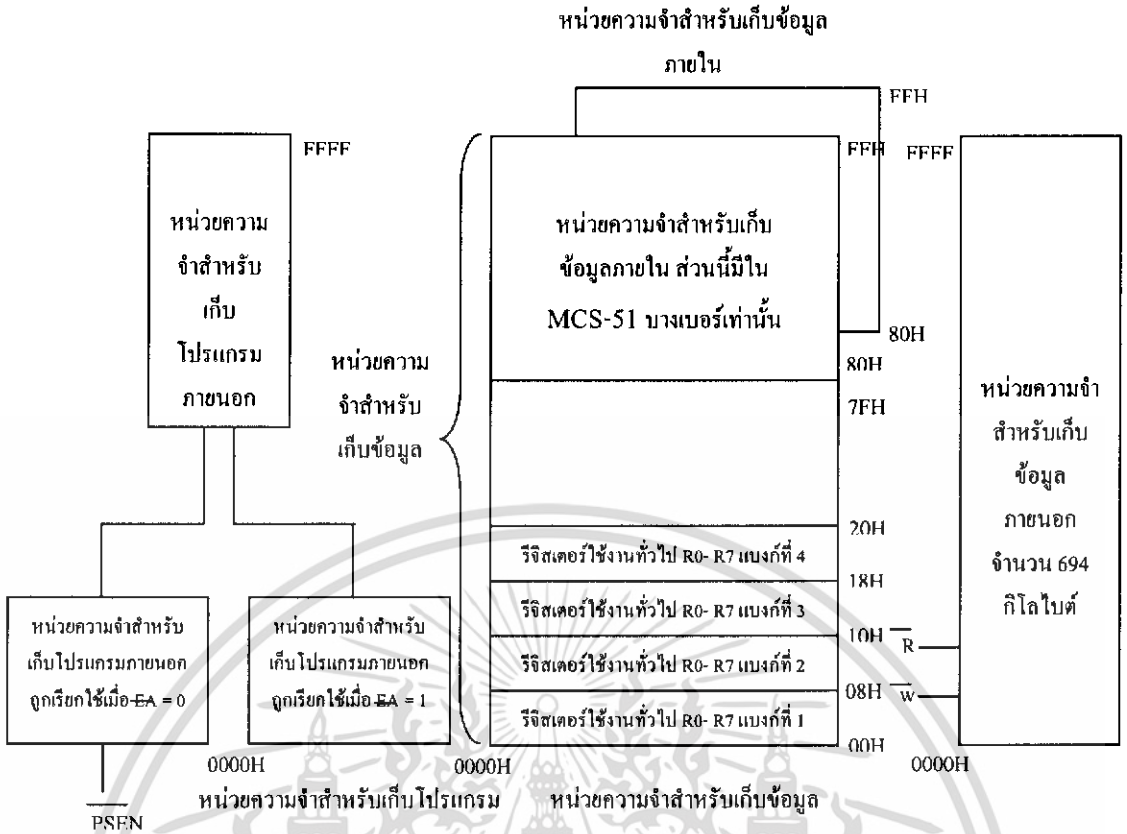


รูปที่ 2.2 แสดงโครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51

2.1.4 การจัดหน่วยความจำ

ในไมโครคอนโทรลเลอร์ตระกูล MCS-51 แบ่งชนิดหรือหน้าที่ของหน่วยความจำออกเป็น 2 ส่วนคือ หน่วยความจำโปรแกรม (Program Memory) และ หน่วยความจำข้อมูล (Data Memory)

หน่วยความจำโปรแกรม ใช้สำหรับเก็บโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ ซึ่งบางหมายเลขจะมีหน่วยความจำนี้อยู่ในตัว โดยอาจจะมีขนาดไม่เท่ากันหรือเป็นหน่วยความจำต่างชนิดกัน เช่น บางหมายเลขเป็น ROM แต่บางหมายเลขอาจเป็น EPROM และบางหมายเลขอาจไม่มีหน่วยความจำในส่วนนี้เลย โปรแกรมการทำงานจะถูกเก็บไว้ที่หน่วยความจำโปรแกรมภายนอกทั้งหมด



รูปที่ 2.3 แสดงการจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูล

สำหรับหน่วยความจำข้อมูลจะใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่างๆ จากการทำงานของโปรแกรมซึ่งใน MCS-51 ทุกหมายเลขจะมีหน่วยความจำส่วนนี้อยู่จำนวนหนึ่ง แต่อาจมีขนาดมากน้อยต่างกันไปในแต่ละหมายเลข สำหรับการจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลแสดงไว้ในรูปที่ 2.3

2.1.4.1 หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมสามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำโปรแกรมภายในและหน่วยความจำโปรแกรมภายนอก หน่วยความจำภายในจะถูกเลือกใช้งานถ้าขาสัญญาณ EA มีค่าเป็น 1 โดยจะถูกใช้งานในช่วงแอดเดรส 0-0FFFH (หรือช่วงแอดเดรส 0-1FFFH ในหมายเลข 8052) ในกรณีตรงข้ามถ้าขาสัญญาณ EA มีค่าเป็น 0 ในช่วงแอดเดรส 0-0FFFH (หรือช่วงแอดเดรส 0-1FFFH ในหมายเลข 8052) จะถูกใช้จากหน่วยความจำภายนอก หรือกล่าวได้ว่าถ้าขาสัญญาณ EA มีค่าเป็น 0 จะเป็นการเลือกใช้หน่วยความจำโปรแกรมภายนอกทั้งหมดตลอดช่วงแอดเดรส

2.1.4.2 หน่วยความจำข้อมูล

หน่วยความจำข้อมูลสามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำข้อมูลภายในและหน่วยความจำข้อมูลภายนอก สำหรับหน่วยความจำข้อมูลภายในยังแบ่งออกได้เป็น 2 ส่วนย่อยคือ ส่วนที่ใช้เก็บข้อมูลทั่วไปและส่วนที่ใช้เป็นรีจิสเตอร์หน้าที่พิเศษหรือ SFR (Special Function Register) โดยส่วนที่ทำหน้าที่เก็บข้อมูลทั่วไปจะถูกใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่างๆ จากการทำงานของโปรแกรม ส่วนรีจิสเตอร์หน้าที่พิเศษจะถูกใช้งานเป็นรีจิสเตอร์ควบคุมการทำงาน และแสดงสถานะการทำงานของไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกหมายเลขจะมีหน่วยความจำข้อมูลภายในขนาด 128 ไบต์เป็นอย่างน้อย และบางหมายเลขอาจมีถึง 256 ไบต์

2.1.4.3 รีจิสเตอร์ใช้งานทั่วไป

รีจิสเตอร์ใช้งานทั่วไป มีไว้สำหรับให้ผู้เขียนโปรแกรมสามารถนำข้อมูลไปพักไว้ชั่วคราวหรือใช้งานทั่วไปได้ตามต้องการ ซึ่งรีจิสเตอร์ใช้งานทั่วไปนี้มีอยู่ด้วยกัน 8 ตัว คือ รีจิสเตอร์ R0-R7 โดยรีจิสเตอร์ทั้ง 8 ตัวถูกจัดให้อยู่รวมกันและมีให้เลือกใช้ถึง 4 แบงก์ (bank) นั่นคือ มีรีจิสเตอร์ใช้งานทั่วไปถึง 32 ตัวให้ใช้งาน เพียงแต่การเลือกใช้รีจิสเตอร์ R0-R7 ในแบงก์ใดแบงก์หนึ่งจะถูกกำหนดจากบิต RS0, RS1 ในรีจิสเตอร์หน้าที่พิเศษ PSW ดังนั้นการเลือกใช้จึงเลือกได้เพียงแบงก์เดียวในขณะใดขณะหนึ่ง อย่างไรก็ตาม ค่าของข้อมูลที่เก็บไว้ในรีจิสเตอร์มีชื่อเดียวกันแต่อยู่คนละแบงก์จะไม่มีผลซึ่งกันและกันเลย ทำให้ผู้เขียนโปรแกรมสามารถใช้งานรีจิสเตอร์ทั่วไปได้ทั้ง 32 ตัวอย่างเต็มที่และไม่ยุ่งยากในการเขียนโปรแกรม

2.1.4.4 รีจิสเตอร์หน้าที่พิเศษ (SFR)

รีจิสเตอร์หน้าที่พิเศษ (SFR) มีบทบาทหน้าที่อย่างมากในการควบคุมการทำงานของไมโครคอนโทรลเลอร์และทำให้การเขียนโปรแกรมสามารถทำได้สะดวกขึ้น รีจิสเตอร์หน้าที่พิเศษนี้ทำหน้าที่สำคัญคือควบคุมการทำงานในส่วนต่างๆ ภายในไมโครคอนโทรลเลอร์และยังทำหน้าที่แสดงสถานะการทำงาน ซึ่งในรีจิสเตอร์หน้าที่พิเศษบางตัวยังสามารถเข้าถึงได้ในระดับบิต (Bit addressable) ด้วย ดังแสดงการจัดหน่วยความจำและตำแหน่งรีจิสเตอร์หน้าที่พิเศษต่างๆ

DIRECT Byte Address	Bit Address								Special Function Register Symbol
	(MSB)							(LSB)	
	WDT	T32	SERR	IZC	P3HZ	P2HZ	P1HZ	ALF	
0F8H	FF	FE	FD	FC	FB	FA	F9	F8	IOCON
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	0V	F1	P	
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0CDH	Not Bit Addressable								TH2
0CCH	Not Bit Addressable								TL2
0CBH	Not Bit Addressable								RCAP2H
0CAH	Not Bit Addressable								RCAP2L
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T	CP/RL	
0C8H	CF	CE	CD	CC	CB	CA	C9	C8	T2CON
	PCT		PT2	PS	PT1	PX1	PT0	PX0	
0B8H	BF	—	BD	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA		ET2	ES	ET1	EX1	ET0	EX0	
0A8H	AF	—	AD	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
99H	Not Bit Addressable								SBUF
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
8DH	Not Bit Addressable								TH1
8CH	Not Bit Addressable								TH0
8BH	Not Bit Addressable								TL1
8AH	Not Bit Addressable								TL0
89H	Not Bit Addressable								TMOD
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
87H	Not Bit Addressable								PCON
83H	Not Bit Addressable								DPH
82H	Not Bit Addressable								DPL
81H	Not Bit Addressable								SP
80H	87	86	85	84	83	82	81	80	P0

รูปที่ 2.4 แสดงการจัดหน่วยความจำและตำแหน่งรีจิสเตอร์หน้าที่พิเศษต่างๆ

การทำงานของรีจิสเตอร์ต่างๆ เป็นดังนี้

ACC: แอคคิวมูเลเตอร์มีขนาด 8 บิตทำหน้าที่เช่นเดียวกับแอคคิวมูเลเตอร์ของซีพียูอื่น ๆ คือใช้เป็นตัวกระทำร่วมทางคณิตศาสตร์ของเลข 2 จำนวน เช่น การบวก ลบ คูณ และหาร เป็นต้น และทำหน้าที่เป็นตัวเก็บผลลัพธ์ที่ได้จากการคำนวณทางคณิตศาสตร์

Register B: เป็นรีจิสเตอร์ที่ใช้งานเฉพาะในคำสั่งการคูณ (MUL) และการหาร (DIV) เท่านั้น โดยจะทำงานร่วมกับ แอคคิวมูเลเตอร์ ในการเก็บผลลัพธ์ของการคูณและเศษที่ได้จากการหาร

SP (Stack Pointer): มีขนาด 8 บิต ทำหน้าที่ชี้ตำแหน่งสแตก ซึ่งไมโครคอนโทรลเลอร์ MCS-51 ใช้พื้นที่ของหน่วยความจำภายในเป็นสแตก โดยตำแหน่งเริ่มต้นหลังจากการรีเซตจะเป็น 07 การทำงานของ SP จะเพิ่มขึ้นหรือลดลงอย่างอัตโนมัติ เมื่อมีการเก็บข้อมูลลงในสแตกหรือนำข้อมูลออกจากสแตก การเก็บข้อมูลลงในสแตกจะเก็บได้ครั้งละ 8 บิต โดยใช้คำสั่ง PUSH โดยค่าของ SP จะเพิ่มขึ้น 1 แล้วเก็บข้อมูลลงในสแตกที่ตำแหน่งที่ SP ชี้อยู่ ซึ่งจะทำให้ตำแหน่งแรกของการเก็บข้อมูลของสแตกคือ 08 การนำข้อมูลออกจากสแตก ทำได้จากการทำคำสั่ง POP โดยจะดึงข้อมูลออกจากสแตกในตำแหน่งที่ SP ชี้อยู่ออกไปกำหนดให้กับหน่วยความจำที่รับข้อมูล แล้วค่าของสแตกจะลดลง 1 หลังจากมีการนำข้อมูลออกจากสแตกไปแล้ว สามารถกำหนดพื้นที่ของสแตกไปอยู่ในตำแหน่งใดๆ ในหน่วยความจำข้อมูลภายในตัวไมโครคอนโทรลเลอร์ได้โดยการกำหนดค่าให้กับ SP ตามที่ต้องการแต่ไม่ควรใช้ในตำแหน่งเริ่มต้น 00 เนื่องจากเป็นตำแหน่งเดียวกับรีจิสเตอร์ R0-R7 ในแบงค์ 0 และไม่ควรกำหนดตำแหน่งสูงเกินไป เพราะจะทำให้มีพื้นที่ของสแตกน้อยซึ่งอาจไม่เพียงพอกับใช้งาน

DPTR (Data Pointer): เป็นรีจิสเตอร์ขนาด 16 บิตใช้สำหรับเป็นตัวชี้ตำแหน่งของหน่วยความจำภายนอก หรือตำแหน่งของอุปกรณ์อินพุทเอาท์พุทที่ไมโครคอนโทรลเลอร์ต้องการติดต่อด้วย และใช้เป็นตัวกำหนดตำแหน่งเริ่มต้น (Base) ของตารางในการทำงานเกี่ยวกับ Look up table รีจิสเตอร์ DPTR ประกอบด้วยรีจิสเตอร์ขนาด 8 บิต 2 ตัวคือรีจิสเตอร์ DPH และ DPL ซึ่งผู้ทำการทดลองสามารถเลือกการใช้งานในลักษณะ 8 บิต 2 ตัวหรือ 16 บิต 1 ตัวก็ได้

P0 - P3: พอร์ต P0 - พอร์ต P3 เป็นรีจิสเตอร์ขนาด 8 บิต ซึ่งค่าที่อยู่ในรีจิสเตอร์เหล่านี้จะเป็นค่าเดียวกับค่าของสัญญาณที่ขาต่างๆของพอร์ต ดังนั้นการส่งข้อมูลออกไปที่พอร์ตจะทำได้โดยการกำหนดค่าให้กับรีจิสเตอร์ P0 - P3 ในกรณีที่ใช้พอร์ตเป็นอินพุทผู้ทำการทดลองสามารถอ่านสัญญาณที่ต่อกับขาของพอร์ตได้จากการอ่านค่าของรีจิสเตอร์ P0 - P3 เช่นกัน

SBUF: เป็นรีจิสเตอร์ขนาด 8 บิต ทำหน้าที่เป็นบัฟเฟอร์ของการรับส่งข้อมูลแบบอนุกรม โดยมีทั้งบัฟเฟอร์ด้านรับข้อมูลและบัฟเฟอร์ด้านส่งข้อมูล บัฟเฟอร์ด้านรับข้อมูลจะทำหน้าที่เก็บข้อมูลที่ส่งเข้ามาให้กับไมโครคอนโทรลเลอร์ทางพอร์ตอนุกรม เมื่อข้อมูลเข้ามาครบจำนวนบิตแล้วค่าที่อยู่ในบัฟเฟอร์ตัวนี้จะถูกไมโครคอนโทรลเลอร์อ่านออกไปอีกทีหนึ่ง สำหรับบัฟเฟอร์ด้านส่งข้อมูลจะรับข้อมูลขนาด 8 บิตที่ไมโครคอนโทรลเลอร์ต้องการส่งออกไปทางพอร์ตอนุกรม โดยข้อมูลในบัฟเฟอร์จะถูกส่งออกไปยังพอร์ตอนุกรมทีละบิตต่อไป

RCAP2H, RCAP2L: เป็นรีจิสเตอร์ขนาด 8 บิต ที่มีอยู่เฉพาะในเบอร์ 8032 และ 8052 ซึ่งใช้ร่วมกับ Timer/Counter 2 ในโหมด Capture รีจิสเตอร์ RCAP2H, RCAP2L จะทำหน้าที่เก็บค่าปัจจุบันของ

รีจิสเตอร์ TH2 และ TL2 ตามลำดับเมื่อมีการเปลี่ยนแปลงของสัญญาณ T2EX สำหรับการทำงานในโหมด Auto reload รีจิสเตอร์ RCAP2H, RCAP2L จะทำหน้าที่เก็บค่าเริ่มต้นของตัวตั้งเวลา และค่านี้จะถูกส่งไปยัง TH2 และ TL2 เมื่อมีการเปลี่ยนแปลงของสัญญาณ T2EX

IP (Interrupt Priority Control): เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่ควบคุมการจัดลำดับความสำคัญของ การอินเตอร์รัพต์

IE (Interrupt Enable): เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่ควบคุมการส่งสัญญาณและการตอบรับอินเตอร์รัพต์ของไมโครคอนโทรลเลอร์

TMOD (Timer Mode Control): เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการเลือกโหมดการทำงานของ Timer/Counter0 และ Timer/Counter1

TCON (Timer Control): เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการทำงานของ Timer/Counter0 และ Timer/Counter1

T2CON: เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการทำงานของ Timer/Counter2

SCON (Serial Control): เป็นรีจิสเตอร์ขนาด 8 บิตใช้สำหรับควบคุมการทำงานของพอร์ตอนุกรม

PCON: เป็นรีจิสเตอร์ทำหน้าที่ควบคุมการใช้กำลังไฟของตัวไมโครคอนโทรลเลอร์

PSW (Program Status Word): เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่แสดงสถานะการทำงานของโปรแกรม หรือเรียกว่าแฟลก ซึ่งจะมีการเปลี่ยนแปลงหลังจากมีการทำงานในคำสั่งต่างๆ ที่มีผลต่อแฟลก และยังใช้เป็นตัวเลือกตำแหน่งแบงค์ของรีจิสเตอร์ (Register Bank) R0-R7 อีกด้วย ผลของบิตต่างๆ สามารถนำไปเป็นเงื่อนไขในการกระโดด (Jump) ได้ค่าของบิตต่างๆ ใน PSW สามารถเซต/เคลียร์ด้วยคำสั่งทางซอฟต์แวร์ได้ แฟลกต่าง ๆ ของรีจิสเตอร์ PSW จะอยู่ในตำแหน่งของบิตต่างๆ ดังนี้

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV	-	P

รูปที่ 2.5 แสดงรีจิสเตอร์ PSW

CY (Carry Flag): เป็นแฟลกตัวทดทำหน้าที่หลายอย่าง เช่น เป็นตัวทดในกรณีของการบวก ทำหน้าที่เป็นตัวยืมในกรณีของการลบ ใช้เป็นตัวร่วมกับแอคคิวมูเลเตอร์ในการหมุนบิต และสามารถใช้ค่าของ CY เป็นเงื่อนไขของการกระโดด (Jump) ได้

AC (Auxiliary Carry Flag): เป็นแฟลกตัวทดช่วยระหว่าง บิตที่ 3 และบิตที่ 4 ซึ่งแฟลกนี้จะถูกเซตเป็น 1 หากทำการบวกเลขแล้วมีการทดจากบิตที่ 3 ไปยังบิตที่ 4 หากไม่มีการทดแฟลกนี้จะเป็น 0 แฟลก AC จะใช้สำหรับการทำงานในลักษณะของเลข BCD ซึ่งจะถูกใช้โดยซีพียู

F0 (Flag 0): เป็นแฟลกที่ใช้งานทั่วไปซึ่งผู้ทำการทดลองสามารถใช้เป็นแฟลกสภาวะ (Status flag) ของโปรแกรมได้โดยการเซตหรือรีเซตด้วยคำสั่งทางซอฟต์แวร์

RS1-RS0 (Register Bank Select): เป็นตัวกำหนดการเลือกพื้นที่ใช้งานของกลุ่มรีจิสเตอร์ R0 - R7 ซึ่งเป็นรีจิสเตอร์ที่ใช้งานทั่วไป พื้นที่ที่เลือกใช้ได้แสดงในตารางต่อไปนี้

ตารางที่ 2.3 แสดงการเลือก Register

RS1	RS0	แบงค์	ตำแหน่ง
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

OV (Overflow Flag): เป็นแฟลกที่แสดงค่า overflow ซึ่งจะถูกเซตหรือเคลียร์จากการทำงานของคำสั่งทางคณิตศาสตร์ การเปลี่ยนแปลงของแฟลกจะพิจารณาในลักษณะของ ตัวเลข 8 บิตเครื่องหมาย (8 bits sign number) โดยบิตซ้ายมือสุดเป็นเครื่องหมาย (0 = บวก, 1 = ลบ) เลขลบจะจัดเก็บในลักษณะของเลข 2 's complement ค่าของจำนวนเลขที่เก็บด้วย 8 บิต เครื่องหมายจะมีค่าบวกสูงสุด 127 (01111111) และมีค่าค่าลบต่ำสุด คือ -128 (10000000) ในการนำเลข 2 จำนวนมารวมกันแล้วได้ผลลัพธ์มากกว่า +127 หรือต่ำกว่า -128 (เกินความสามารถของ 8 บิต) จะทำให้แฟลก OV ถูกเซตเป็น 1 แฟลก overflow จะเกิดเมื่อนำเลขลบ 2 จำนวนมารวมกันแล้วได้ค่าลบต่ำกว่า -128 หรือนำเลขบวก 2 จำนวนมารวมกันแล้วได้ค่าเป็นบวกมากกว่า 127 การนำเลข 2 จำนวนที่จำนวนหนึ่งเป็นค่าลบ และอีกจำนวนหนึ่งเป็นค่าบวก มารวมกันจะไม่ทำให้เกิด overflow

P (Parity Flag): เป็นแฟลกที่แสดงจำนวนบิตที่เป็น 1 ในแอคคิวมูเลเตอร์ โดยแฟลก P จะถูกเซตเป็น 1 เมื่อแอคคิวมูเลเตอร์มีจำนวนบิตที่เป็น 1 เป็นคี่ (odd) และแฟลก P จะถูกเคลียร์เป็น 0 เมื่อแอคคิวมูเลเตอร์มีจำนวนบิตที่เป็น 1 เป็นคู่ (even) ดังนั้นจำนวนบิตในแอคคิวมูเลเตอร์รวมกับแฟลก P จะมีจำนวนบิตที่เป็น 1 ทั้งหมดเป็นจำนวนคู่

2.1.5 การใช้งานรีจิสเตอร์

โดยปกติไมโครคอนโทรลเลอร์ตระกูล MCS-51 จะทำการประมวลผลข้อมูลครั้งละ 1 ไบต์ ซึ่งกระทำกับรีจิสเตอร์ภายในโดยที่รีจิสเตอร์แต่ละตัวเก็บข้อมูลได้ขนาด 1 ไบต์เช่นกัน เช่น รีจิสเตอร์ A ซึ่งเป็นแอคคิวมูเลเตอร์ (accumulator) ทำหน้าที่เป็นรีจิสเตอร์กลางสำหรับการคำนวณทางคณิตศาสตร์หรือทางลอจิกของตัวกระทำ 2 ตัว ตัวอย่างเช่น ถ้าต้องการบวกค่า 10 กับข้อมูลตัวหนึ่ง ให้ทำการโหลดข้อมูลไปเก็บไว้ในรีจิสเตอร์ A ก่อน จากนั้นให้ใช้คำสั่งนำค่า 10 ไปบวกกับ A ผลที่ได้จกการบวกข้อมูลและ

ค่า 10 จะถูกเก็บไว้ในรีจิสเตอร์ A นอกจากรีจิสเตอร์ A ทำการบวกด้วยการกำหนดค่าโดยตรงแล้วยังสามารถทำการคำนวณร่วมกับรีจิสเตอร์ขนาด 8 บิตตัวอื่นๆ ได้อีกด้วย

ทั้งไมโครโปรเซสเซอร์ และไมโครคอนโทรลเลอร์จะมีรีจิสเตอร์สำหรับใช้งานในคำสั่งพิเศษโดยผู้เขียนโปรแกรมอาจกำหนดขึ้นเองได้ โดยที่กำหนดให้อยู่ในตำแหน่งแอดเดรสพิเศษซึ่งในที่นี้มีค่ามากกว่า 07FH ขึ้นไป ตัวอย่างเช่น แอคคิวมูเลเตอร์ถูกกำหนดให้ใช้หน่วยความจำภายในที่ 0E0H รีจิสเตอร์เหล่านี้เรียกว่า รีจิสเตอร์หน้าที่พิเศษ (Special Function Registers: SFRs) จำนวนของรีจิสเตอร์หน้าที่พิเศษอาจจะมีไม่เท่ากันในไมโครคอนโทรลเลอร์แต่ละหมายเลขในตระกูล MCS-51 ขึ้นอยู่กับคำสั่งที่ได้ทำการตั้งค่าไว้ เพราะรีจิสเตอร์หน้าที่พิเศษเหล่านี้ถูกรวม หรือใช้พื้นที่ในหน่วยความจำภายในของไมโครคอนโทรลเลอร์ ซึ่งหน่วยความจำภายในหรือแรมภายในจะมีขนาดไม่เท่ากันในแต่ละหมายเลข

นอกจากรีจิสเตอร์หน้าที่พิเศษหรือ SFR แล้วยังมีรีจิสเตอร์สำหรับใช้งานทั่วไปอีก 8 ตัวคือ รีจิสเตอร์ R0-R7 รีจิสเตอร์ทั้ง 8 ตัวนี้ถูกบรรจุอยู่ภายในแรมภายในไมโครคอนโทรลเลอร์ในรูปของแบงก์ และใช้สำหรับเก็บข้อมูลชั่วคราวระหว่างการประมวลผล

2.1.6 พอร์ตอินพุทและพอร์ตเอาท์พุท

พอร์ต คือ แอดเดรสหนึ่งที่ได้รับการกำหนดไว้เพื่อการโอนย้ายข้อมูลระหว่างไมโครคอนโทรล-เลอร์กับอุปกรณ์ภายนอก การกำหนดประเภทของการติดต่อขึ้นอยู่กับทิศทางการไหลของข้อมูล เมื่อพิจารณาจากไมโครคอนโทรลเลอร์เป็นหลัก

รูปที่ 2.6 (ก), (ข) แสดงการใช้ไมโครคอนโทรลเลอร์เป็นพอร์ตอินพุทและเอาท์พุท

จากรูปที่ 2.6 (ก) เป็นการใช้ไมโครคอนโทรลเลอร์เป็นพอร์ตเอาท์พุท และจากรูป 2.6 (ข) เป็นการใช้ไมโครคอนโทรลเลอร์เป็นพอร์ตอินพุท

2.1.6.1 การใช้งานพอร์ตเป็นอินพุท

การใช้งานพอร์ตเป็นการอินพุทข้อมูล จะต้องเริ่มต้นด้วยการส่งข้อมูลที่มีค่าเป็น 1 ออกมาทางบิตของพอร์ตสั้นก่อนเป็นอันดับแรก เพื่อหยุดการทำงานของทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณ

เอาท์พุทของบิตนั้น ทำให้ขาสัญญาณของบิตถูกต่อเข้ากับตัวต้านทานซึ่งทำหน้าที่พูลอัพ ภายในซึ่งมีผลทำให้บิตนั้นของพอร์ต 1, 2 และ 3 เป็นสภาวะลอจิกสูง ตัวต้านทานนี้มีค่าประมาณ 50 kΩ ซึ่งเป็นค่าที่สูงมาก และทำให้อุปกรณ์ภายนอกสามารถขับสัญญาณของพอร์ตเหล่านี้เป็นลอจิกต่ำได้ง่าย สำหรับบิตของพอร์ต 0 นั้นแม้ว่าจะมีหลักการทำงานที่คล้ายคลึงกันกับบิตของพอร์ตอื่นๆ แต่เนื่องจากไม่มีตัวต้านทานซึ่งทำหน้าที่พูลอัพภายในไว้ ทำให้เมื่อทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นหยุดการทำงาน ก็จะเป็นผลให้สัญญาณนี้อยู่ในสภาวะอิมพีแดนซ์สูงแทน

2.1.6.2 การใช้งานพอร์ตเป็นเอาท์พุท

เมื่อมีการส่งข้อมูลที่มีค่าเป็น 0 ให้กับแต่ละบิตของพอร์ตทุกพอร์ต ข้อมูลนี้จะถูกส่งให้กับฟลิป-ฟลอป ซึ่งจะค้างค่านี้ไว้ และมีผลทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นทำงาน ดังนั้นขาสัญญาณก็จะมีสภาวะลอจิกเป็นลอจิกต่ำด้วย

ส่วนการส่งข้อมูลที่มีค่าเป็น 1 ออกมานั้น ในกรณีที่เป็นการทำงานในแต่ละบิตของพอร์ต 1, 2 และ 3 จะทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นหยุดทำงาน มีผลทำให้ขาของสัญญาณเป็นลอจิกสูงด้วยตัวต้านทานที่พูลอัพอยู่ภายในนั้น แต่สำหรับการใช้งานในแต่ละบิตทางพอร์ต 0 นั้นจะมีผลแตกต่างออกไป โดยขาสัญญาณจะมีสภาวะอิมพีแดนซ์สูงแทน เนื่องจากไม่มีตัวต้านทานภายในเชื่อมต่ออยู่นั่นเอง ดังนั้นการใช้งานพอร์ต 0 เป็นการนำข้อมูลออกทางเอาท์พุท จึงจำเป็นจะต้องใช้ตัวต้านทานภายนอกพูลอัพสัญญาณไว้กับลอจิกสูงแทน

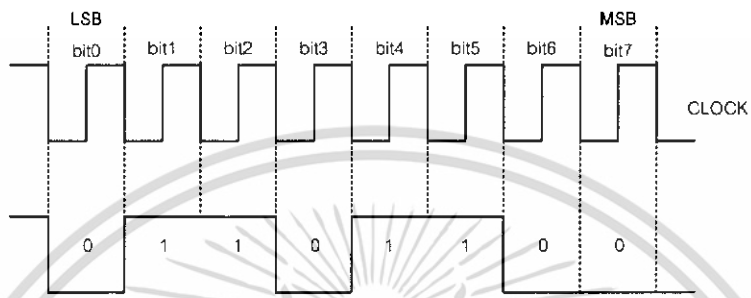
2.2 การสื่อสารข้อมูล

การเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพ่วงอื่นๆหรือคอมพิวเตอร์ด้วยกันนั้น มี 2 วิธี คือ การรับส่งข้อมูลแบบขนานและการรับส่งข้อมูลแบบอนุกรม การรับส่งข้อมูลแบบขนานจะเป็นการรับส่งข้อมูลคราวละ 4 หรือ 8 บิต ในเวลาเดียวกันซึ่งจะทำให้การรับและส่งข้อมูลทำได้ที่ความเร็วสูง ซึ่งหมายความว่า จำนวนสายที่ใช้ในการส่งจะต้องมีมากเท่ากับจำนวนบิตของข้อมูลที่จะส่งด้วย นอกจากนี้ยังจะต้องรวมถึงสายที่ใช้สำหรับควบคุมและการตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจจะต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตข้อมูลที่จะส่งก็ได้ ซึ่งก็เป็นปัญหาในเรื่องของราคาของสายที่ใช้ในการเชื่อมต่อแบบขนานมักจะมีราคาแพง

ในขณะที่การรับส่งข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลายๆ บิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลมาให้ครบทุกบิตเสียก่อนจึงจะทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลแบบอนุกรมอาจมีความเร็วต่ำกว่าแบบขนาน อย่างไรก็ตาม การรับส่งข้อมูลแบบอนุกรมนั้นสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

2.2.1 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นจะแบ่งออกได้เป็น 2 แบบ คือ การสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสคือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นหนึ่งจะเป็นสายของข้อมูล ดังนั้นการติดต่อแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้น คือ สัญญาณนาฬิกา, ข้อมูล และ ground



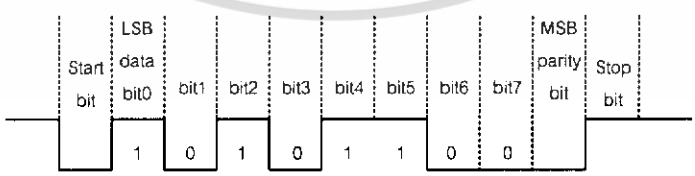
รูปที่ 2.7 แสดงรูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

2.2.2 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัส คือการรับส่งข้อมูลไปในสายสัญญาณ โดยไม่จำเป็นต้องส่งสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายเทข้อมูล หรือ อัตราบอด (Baud rate) ที่มีหน่วยเป็น บิตต่อวินาที (bit per second: bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกัน คือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลอนุกรมจะมีขนาด 5 - 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิตหรือไม่มีก็ได้
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1, 1.5 หรือ 2 บิต



รูปที่ 2.8 แสดงรูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

ในรูปที่ 2.8 เป็นการแสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส ซึ่งเมื่อไม่มีข้อมูลที่จะส่ง ขาดาต้าจะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะว่าง (idle stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขาดาต้ามีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต ซึ่งจะเรียกบิตนี้ว่าบิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไปโดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต 5 - 8 บิตก็ได้ จากนั้นจะตามด้วยบิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งคือบิตปิดท้าย ซึ่งจะให้ขาดาต้ามีสถานะลอจิก 1 อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและการส่งข้อมูลแบบอะซิงโครนัส เรียกว่า Universal Asynchronous Receiver / Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัส คือ ค่าอัตราบอด ซึ่งก็คือค่าอัตราการเข้ารหัสที่ใช้ในการรับและส่งข้อมูล อัตราบอดมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าอัตราบอดได้สูงถึง 115200 บิตต่อวินาที เนื่องจากอัตราบอดคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิตและบิตปิดท้าย 1 บิตความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้อัตราบอดในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (odd), แบบคู่ (even) หรืออาจไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก "1" ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบิตพาริตี้เข้าไปด้วย ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิตและมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสอง จะเห็นว่าข้อมูลในไบต์นี้มีลอจิก "1" จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้น ถ้ากำหนดค่าพาริตี้เป็นคู่ ค่าในบิตพาริตี้จะต้องมีลอจิก "0" แต่ถ้าพาริตี้เป็นคี่ ค่าที่บิตพาริตี้จะต้องเป็น "1" เพื่อให้ข้อมูล 1 ไบต์ รวมทั้งบิตพาริตี้มีจำนวนบิตที่เป็นลอจิก "1" มีจำนวนรวมกันเป็นเลขคี่ ตารางที่ 2.4 แสดงตัวอย่างของบิตพาริตี้ในการรับส่งข้อมูลอนุกรม

ตารางที่ 2.4 แสดงบิตพาริตี้ของข้อมูล

ข้อมูล	บิตพาริตี้คู่	บิตพาริตี้คี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	1	0
11111111	0	1

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART ซึ่งทางภาครับจะต้องทำการกำหนดคุณสมบัติการตรวจสอบพาริตี้ให้ตรงกันว่าจะตรวจสอบพาริตี้คี่หรือพาริตี้คู่ จากนั้นภาครับของ UART จะทำการตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแสดงข้อผิดพลาดออกมาให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดที่เกิดขึ้นในการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดแค่เพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่งจะไม่มีการตรวจสอบพาริตี้

2.3 การสื่อสารข้อมูลอนุกรม

การสื่อสารข้อมูลอนุกรมเป็นการรับหรือส่งข้อมูล ในลักษณะกลุ่มของบิตคราวละหนึ่งบิตเรียงลำดับเรื่อยไปจนสิ้นสุด การสื่อสารแบบนี้จะมีข้อแตกต่างจากการสื่อสารแบบขนานเป็นอย่างมาก เนื่องจากข้อมูลมีการโอนย้ายมาพร้อมกัน จึงมีความจำเป็นต้องใช้จำนวนเส้นสัญญาณมากขึ้นตามจำนวนบิตของข้อมูลด้วย ในขณะที่การสื่อสารแบบอนุกรมนั้นต้องการเส้นสัญญาณเพียงสองหรือสามเส้นเท่านั้น ดังนั้นในการสื่อสารแบบขนานจึงไม่เหมาะสมในการสื่อสารกับอุปกรณ์ภายนอกเป็นระยะไกลๆ เพราะจะทำให้สิ้นเปลืองค่าใช้จ่ายมาก

2.3.1 ความเร็วของการสื่อสารแบบอนุกรม

เนื่องจากการสื่อสารข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลในลักษณะกลุ่มของบิตข้อมูล (Bit Stream) ดังนั้น จึงต้องให้ความสนใจในการพิจารณาถึงเรื่องอัตราความเร็วในการรับส่งบิตเหล่านี้เป็นลำดับแรก โดยทั่วไปมักจะระบุกันในหน่วยของจำนวนบิตข้อมูลภายในเวลาหนึ่งวินาที เรียกว่า อัตราบอด ตามค่ามาตรฐานเหล่านี้ ได้แก่ 110, 150, 300, 1200, 2400, 4800, 9600, 19200 บอด ข้อมูลทั้ง 8 บิตนี้

หากถูกส่งออกมาด้วยอัตรา 2400 บอด จะใช้เวลาในการส่งข้อมูลหนึ่งบิตมีค่าเท่ากับ 1/ 2400 หรือ 416 us และ เวลาในการส่งข้อมูลทั้ง 8 บิตมีค่าเท่ากับ (8*416) หรือ 3328 us

2.3.2 รูปแบบของการส่งข้อมูลอนุกรม

การสื่อสารอนุกรมแบบอะซิงโครนัสจะใช้การแปลงข้อมูลขนานให้เป็นอนุกรม แล้วเพิ่มบิตบางอย่างรวมไปกับการส่งข้อมูลจริง ซึ่งได้แก่

- บิตเริ่มต้น (Start Bit)

บิตเริ่มต้นมีหน้าที่สำหรับการบ่งบอกให้ทราบถึงตำแหน่งจุดเริ่มต้นก่อนบิตข้อมูล ตามปกติแล้วค่าของบิตเริ่มต้นจะเป็นระดับลอจิกต่ำ

- บิตแสดงภาวะความเป็นเลขคู่หรือเลขคี่ (Parity Bit)

บิตนี้มีหน้าที่เพื่อการตรวจสอบความถูกต้องของข้อมูลโดยทั่วไปมักเรียกว่า " บิตพาริตี้ " และจะนำไปต่อท้ายบิตข้อมูล ค่าของบิตนี้จะขึ้นอยู่กับจำนวนค่าของบิตที่เป็น 1 ซึ่งจะเป็นได้สองลักษณะคือ พาริตี้คู่ (Even Parity) หรือพาริตี้คี่ (Odd Parity) ตัวอย่างเช่น ระบบที่ติดต่อกัน โดยระบุว่าจะใช้พาริตี้คี่ ทางด้านส่งจะนำข้อมูลที่จะส่งมาพิจารณา จำนวนของบิตที่มีค่า 1 เป็นเลขจำนวนคู่อยู่แล้ว ค่าของบิตพาริตี้จะมีค่าเป็น 0 แต่หากว่าจำนวนของบิตที่มีค่าเป็น 1 เป็นจำนวนเลขคี่ ค่าของบิตพาริตี้จะมีค่าเป็น 1 การพิจารณาทางด้านรับเป็นการตรวจสอบจำนวนบิตที่มีค่า 1 ของข้อมูลที่ได้รับมาทั้งหมดรวมทั้งบิตพาริตี้ ถ้ามีค่าเป็นเลขจำนวนคู่ แสดงว่าข้อมูลที่รับเข้ามานี้ถูกต้อง หากไม่เป็นเลขจำนวนคู่แสดงว่าเกิดการผิดพลาดของข้อมูลขึ้น เป็นต้น

- บิตสุดท้าย (Stop Bit)

บิตสุดท้ายเป็นบิตที่เพิ่มขึ้นเพื่อระบุถึงขอบเขตการสิ้นสุดของกลุ่ม บิตสุดท้ายนี้สามารถโปรแกรมได้คือ 1 บิต, 1 ½ บิต และ 2 บิต ดังนั้นกรณีของการส่งข้อมูล 8 บิต หากข้อมูลถูกส่งออกไปด้วยอัตราเร็ว 2400 บอด เวลาโดยรวมในการส่งข้อมูลหนึ่งไบต์จะมีค่าเป็น (12*416) us หรือ 4.99 ms

2.3.3 การส่งข้อมูลอนุกรมของ 8051

พอร์ตอนุกรมของ 8051 มีโครงสร้างการทำงานในแบบที่เรียกว่า ฟูลดูเพล็กซ์ (Full Duplex) ในการรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน โดยทางด้านวงจรของตัวส่ง (Transmitter) ประกอบด้วยข้อมูลออกไปยังพอร์ตอนุกรมทางขาสัญญาณ TxD (P3.1) ส่วนวงจรด้านตัวรับ (Receiver) ประกอบด้วย SUBF เช่นเดียวกับสัญญาณข้อมูลอนุกรมที่รับเข้ามาทางขาสัญญาณ RxD (P3.0)

พอร์ตอนุกรมของ 8051

สามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM 0 และ SM 1 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมดของพอร์ตอนุกรม มีดังนี้

โหมด 0: ใช้รับส่งข้อมูล 8 บิต โดยการส่งจะเลื่อนออกทีละบิต โดยส่งบิต 0 ออกไปก่อนทางขา RxD และไม่มีการส่ง start bit แต่จะส่ง shift clock ทางขา TxD ความเร็ว 1/12 เท่าของ CPU CLOCK

โหมด 1: ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART (Universal Asynchronous Receiver/Transmitter) โดยส่งแบบ 10 บิต ข้อมูล 8 บิต 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลงอัตราเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิต SMOD ใน PCON และอัตรา overflow ของ Timer 1

โหมด 2: ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และกำหนดอัตราความเร็วในการส่งข้อมูลเท่ากับ 1/32 และ 1/64 ของ CPU CLOCK โดยโปรแกรมที่บิต SMOD ใน PCON

โหมด 3: ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้ โดยควบคุมที่บิต SMOD และอัตรา overflow ของ Timer 1 นอกจากนี้ โหมด 2 และ 3 ยังมีการดำเนินการอีกแบบหนึ่ง โดยสามารถนำมาใช้ประโยชน์ในการสื่อสารข้อมูลแบบที่มีไมโครโปรเซสเซอร์หลายตัวทำงานร่วมกันได้ซึ่งมีชื่อเรียกว่า "Multiprocessor Mode"

ตารางที่ 2.5 แสดงการเลือกโหมดการทำงาน

SM 1	SM 0	โหมด	การทำงาน
0	0	0	ทำงานเป็น shift register อัตราเร็วในการรับหรือส่งข้อมูลเท่ากับ 1/12 ของความถี่ออสซิลเลเตอร์
0	1	1	8 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดเองได้
1	0	2	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูล 1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์
1	1	3	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดเองได้

SM 0, SM 1 บิตเลือกการทำงานแบบ
1: เลือก Multiprocessor Mode ใช้ได้กับโหมด 2, 3
2: เลือก Single Processor Mode ใช้ได้กับทุกโหมด

REN บิตควบคุมให้รับหรือไม่รับข้อมูล
1: ให้รับข้อมูลได้
2: ห้ามรับข้อมูล

TB 8 (Transmit Bit D8) ข้อมูลบิตที่ 9 ที่ส่งออกไปในโหมด 2, 3 ให้ใส่ในบิต TB 8

RB 8 (Receiver Bit D8) ข้อมูลบิตที่ 9 ที่รับเข้ามาจะเก็บในบิตนี้ ข้อมูลบิตที่ 9 ก็คือค่าใน TB 8 ทางด้านส่งนั้นเอง

TI	บิต TI จะเป็น 1 เมื่อสิ้นสุดการส่งข้อมูล 1 ไบต์
RI	บิต RI จะเป็น 1 เมื่อรับข้อมูลเสร็จ 1 ไบต์

2.3.4 กระบวนการรับและส่งข้อมูลอนุกรมของ 8051

การส่งข้อมูลออกทางพอร์ตอนุกรมของ 8051 จะเริ่มต้นขึ้นภายหลังเมื่อมีการเขียนข้อมูลลงใน SBUF ข้อมูลนี้จะถูกเลื่อนทีละบิตและส่งสัญญาณออกไปภายนอกโดยอัตโนมัติ เมื่อข้อมูลเหล่านี้ได้ส่งออกครบถ้วนแล้วจะทำให้ค่าของแฟลก TI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าขณะนี้ SBUF ว่างและพร้อมที่จะส่งข้อมูลไบต์ต่อไปแล้ว ในกรณีที่ใช้เขียนข้อมูลใหม่ลงในรีจิสเตอร์ SBUF โดยไม่รอให้แฟลก TI มีค่าเป็น 1 ก่อนจะมีผลทำให้ข้อมูลที่ส่งไปผิดพลาดได้ สำหรับการรับข้อมูลจากพอร์ตอนุกรมจะต้องเริ่มต้นโดยการกำหนดเซตค่าดังนี้ REN (Receiver Enable) ให้มีค่าเป็น 1 ก่อน หลังจากนั้นเมื่อข้อมูลภายนอกถูกส่งเข้ามายัง 8051 ทีละบิตจนครบ และเมื่อบิตสุดท้ายถูกเลื่อนเข้ามาเรียบร้อยแล้วข้อมูลนั้นจะถูกย้ายมาเก็บไว้ยังรีจิสเตอร์ SBUF และแฟลก RI ก็จะมีค่าเป็น 1 หลังจากนั้นก็จะเกิดการอินเตอร์รัพต์ขึ้น

ลักษณะของการส่งข้อมูลแบบอนุกรมนั้น ข้อมูลจะส่งออกมาทีละบิตจากตัวส่งไปตัวรับข้อมูล ช่องสัญญาณในการส่งข้อมูลอาจใช้เพียง 1 หรือ 2 ช่องสัญญาณเท่านั้น ทำให้ค่าใช้จ่ายในการสื่อสารจะถูกกว่าแบบขนาน แต่อัตราการรับ-ส่งข้อมูลจะช้ากว่าแบบขนาน ในการส่งข้อมูลแบบอนุกรมข้อมูลที่ต้องการส่งจะอยู่ในลักษณะเป็นไบต์จะทยอยส่งทีละบิต และทางตัวรับจะต้องรับข้อมูลเข้ามาทีละบิตแล้วมารวมกันเป็นไบต์ ซึ่งทางตัวรับต้องคอยตรวจสอบว่าบิตใดเป็นบิตเริ่มต้นหรือบิตสุดท้ายของข้อมูล การตรวจสอบนั้นจะขึ้นอยู่กับรูปแบบของรหัสของบิตข้อมูลที่ใช้

2.3.5 การเขียนโปรแกรมควบคุมการรับและส่งข้อมูลทำได้ 2 วิธี

- การตรวจสอบบิต TI หรือ RI โดยใช้คำสั่งตรวจสอบบิต เช่น ใช้คำสั่ง WAIT: JNB TI,WAIT

คำสั่งนี้หมายความว่า ถ้า TI = 0 ให้วนไปยังแอดเดรสชื่อ WAIT

ถ้า TI = 1 ถือว่าส่งข้อมูลเสร็จแล้วให้ทำคำสั่งถัดไป

- การใช้อินเตอร์รัพต์ควบคุม

โหมด 1 : พอร์ตสื่อสารอนุกรม 10 บิต ข้อมูล 8 บิต 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลงความเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิต SMOD ใน PCON และ อัตรา overflow ของ Timer 1, 2

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

ถ้าต้องการ 1200 บอด

ถ้ากำหนด CPU OSC = 11.059 MHz

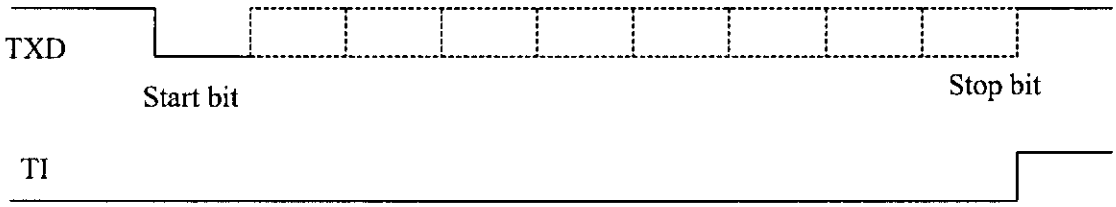
ถ้า SMOD = 0

สามารถหา TH1 ได้โดย

$$1200=\frac{2^{0}\times 11.059\times 10^{-6}}{32\times 12\times[256-(TH1)]}$$

โดยใช้ Timer 1

$TH1=232_{10}=E8H$

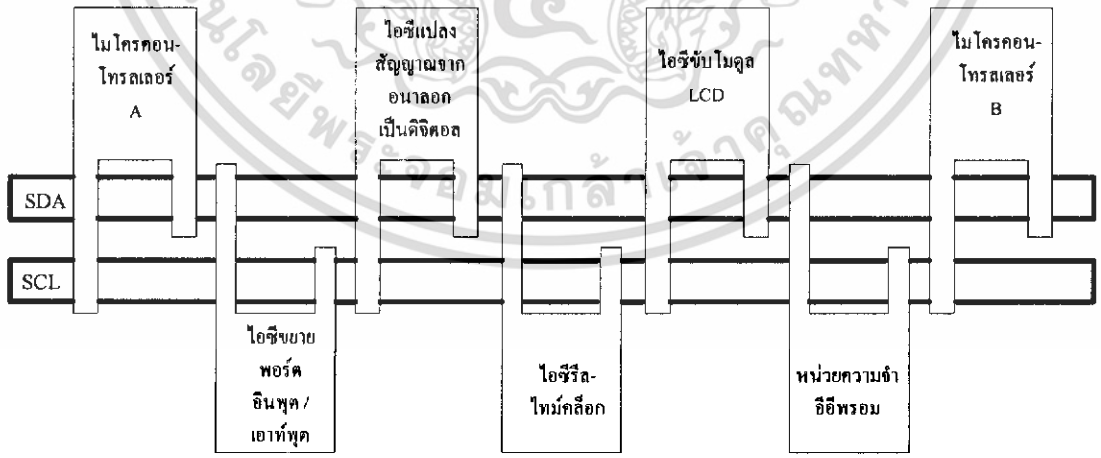


รูปที่ 2.9 แสดงรูปแบบเฟรมของโหมด 1

2.4 ระบบบัส I^2C

I^2C ย่อมาจาก Inter – IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซี โดยบัส I^2C ได้รับการพัฒนาขึ้นด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงาน และควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นหนึ่งคือ สายข้อมูล อีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I^2C ทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่อกับอุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่าสายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock line) ในการอธิบายต่อไปนี้จะเรียกสายสัญญาณทั้งสองว่า สาย SDA และ SCL

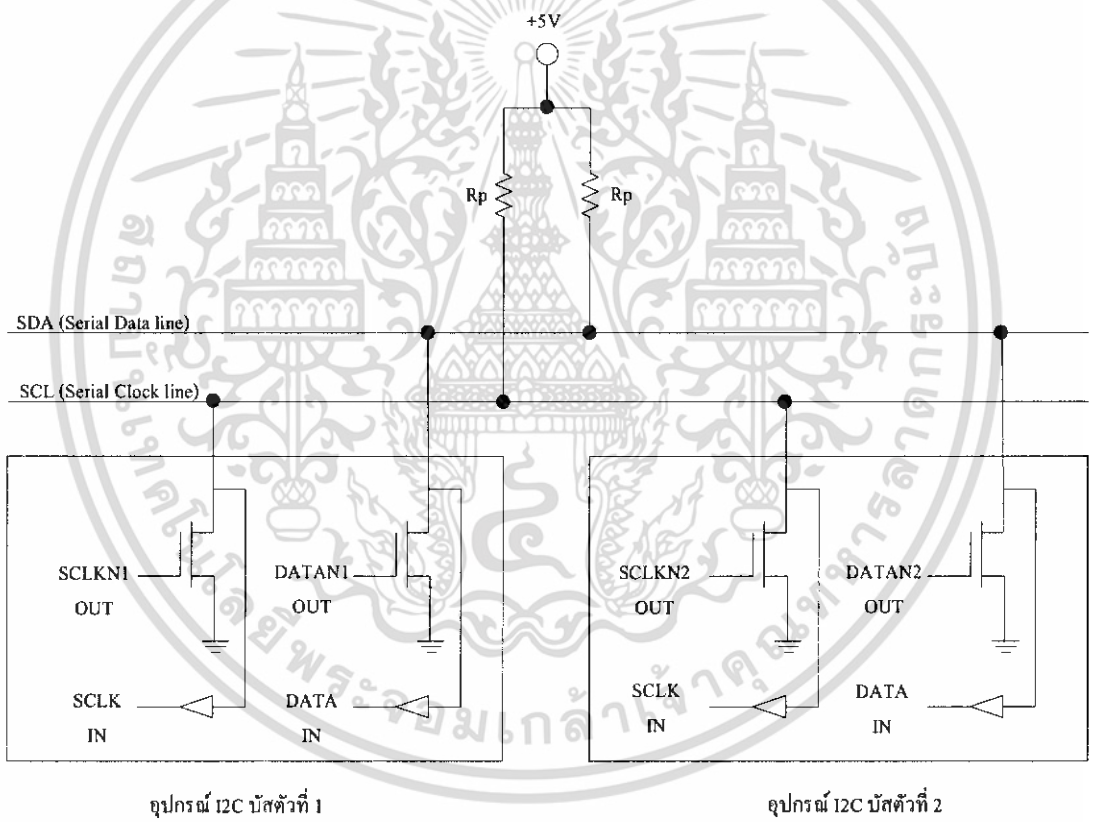


รูปที่ 2.10 แสดงผังการเชื่อมต่อของอุปกรณ์ต่างๆ บนระบบบัส I^2C

ในรูปที่ 2.10 แสดงผังของการเชื่อมต่ออุปกรณ์ต่างๆ บนบัส I^2C จะเห็นได้ว่า อุปกรณ์ที่ทำการเชื่อมต่อบนบัส I^2C มีหลากหลาย ไม่ว่าจะเป็นไอซีขยายพอร์ตอินพุทเอาท์พุท (I/O Expander), ไอซีแปลงสัญญาณอนาลอกเป็นดิจิตอล (ADC) และแปลงสัญญาณดิจิตอลเป็นอนาลอก (DAC), ไอซีรีล-ไทม์คล็อก (RTC), ไอซีขับโมดูล LCD, หน่วยความจำอีอีพรอม และไมโครคอนโทรลเลอร์

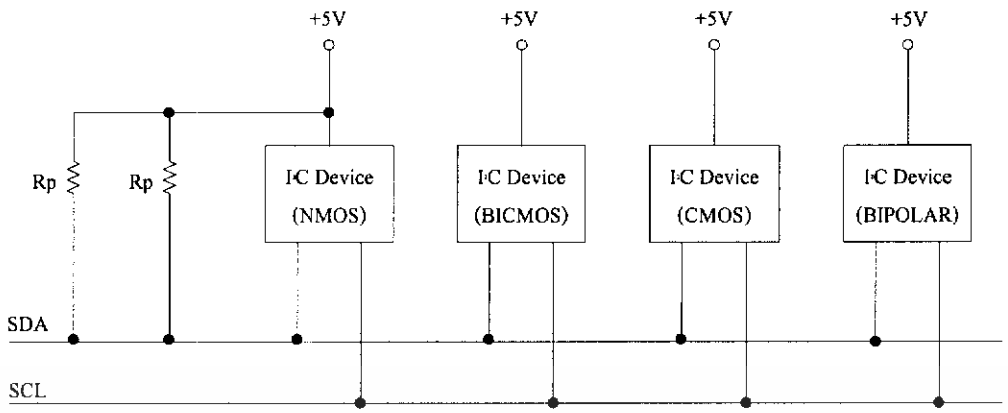
2.4.1 คุณสมบัติทั่วไปของบัส I^2C

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพูลอัพ (Rp) กับแรงดัน +5V ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาท์พุท ของอุปกรณ์ที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (open-drain) หรือคอลเล็กเตอร์เปิด (open-collector) ดังรายละเอียดในรูปที่ 2.11



รูปที่ 2.11 แสดงวงจรเอาท์พุทของอุปกรณ์ในระบบบัส I^2C

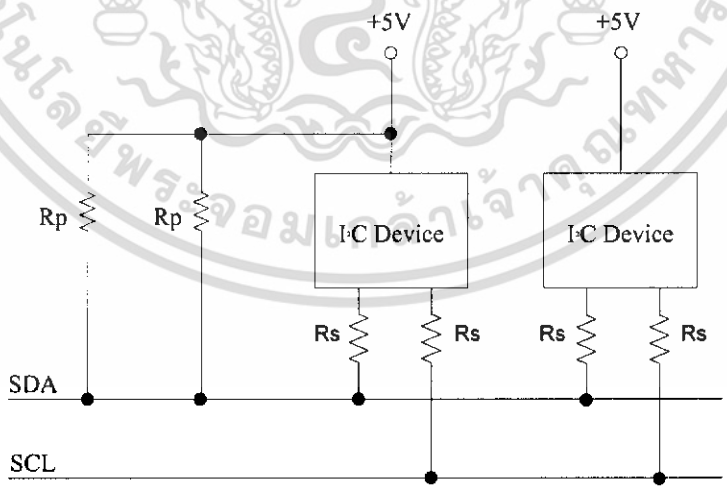
อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติ (standard mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (fast mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง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



รูปที่ 2.12 แสดงการต่อตัวต้านทานพูลอัพ (Rp) บนสายสัญญาณบนระบบบัส I^2C

ข้อเด่นอีกประการหนึ่งของบัส I^2C คือ สามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้สามารถติดต่อสื่อสารกันได้ โดยอุปกรณ์บนบัส I^2C ตัวหนึ่งอาจใช้ไฟเลี้ยง +5V ในขณะที่อีกตัวหนึ่งใช้ไฟเลี้ยง +12 V การต่อร่วมกันบนบัส I^2C สามารถกระทำได้ในลักษณะเดียวกับกรณีที่อุปกรณ์ทั้งสองใช้ไฟเลี้ยงเท่ากัน กล่าวคือ ให้ต่อสาย SDA และ SCL ของอุปกรณ์แต่ละตัวเข้าด้วยกัน และต้องต่อตัวต้านทานพูลอัพ (Rp) เข้ากับแรงดัน +5 V ไว้ด้วยเสมอ ดังแสดงในรูปที่ 2.12

ในกรณีที่อาจมีแรงดันไฟกระชากขนาดใหญ่ปะปนเข้ามาในบัส I^2C ที่ขา SDA และ SCL ของอุปกรณ์แต่ละตัว จะต้องต่อตัวต้านทานอนุกรมกับขา SDA และ SCL เรียกว่า Rs ก่อนต่อเข้าสู่บัส I^2C ดังแสดงในรูปที่ 2.13



รูปที่ 2.13 แสดงการต่อตัวต้านทาน Rs เพื่อลดสัญญาณรบกวนขนาดใหญ่ที่อาจเข้ามาในบัส I^2C

2.4.2 หลักการของบัส I^2C

บัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้น ดังที่ได้กล่าวมาแล้ว คือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส หรือเรียกว่า โปรโตคอล (Protocol) เพื่อให้ผู้ใช้งานทราบว่า ขณะนี้อุปกรณ์ใดติดต่อกันอยู่และอุปกรณ์ตัวใดเป็นตัวรับหรือตัวส่ง ต่อไปนี้จะขออธิบายลักษณะ หน้าที่และนิยามของอุปกรณ์ที่ต่ออยู่บนบัส I^2C เพื่อเป็นข้อตกลงพื้นฐานก่อนที่จะอธิบายการทำงานของบัส I^2C ต่อไป

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง (Transmitter)

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

อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I^2C เรียกว่า มาสเตอร์ (master)

อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ

(1) การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น

(2) ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะเป็นลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

2.4.3 สภาวะที่เกิดขึ้นบนบัส I^2C

มีด้วยกัน 5 สภาวะ ดังนี้

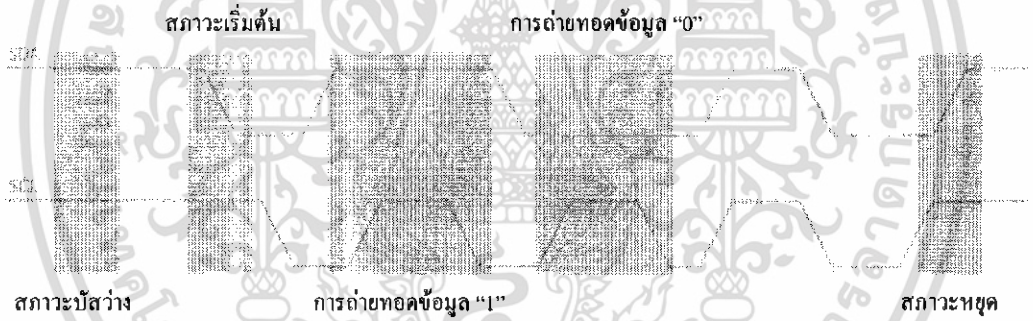
(1) บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้งคู่ นั่นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้

(2) เริ่มต้นการถ่ายทอดข้อมูล (start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะเริ่มต้น (START)

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

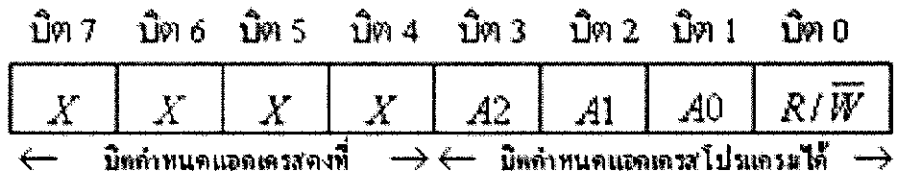
(4) ข้อมูลดำรงอยู่บนบัส (data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่า เป็น "0" หรือ "1" ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น

(5) รับรู้ข้อมูล (acknowledge) เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่า บิตรับรู้ (acknowledge bit) มีสถานะเป็นลอจิกสูง หลังจากส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกา เพื่อตอบสนองบิตรับรู้ที่ส่งมาจากตัวส่ง ทางด้านตัวรับจะส่งบิตรับรู้ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างอิงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้ เพื่อตอบสนองให้ทราบว่าได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว



รูปที่ 2.14 แสดงไดอะแกรมเวลาแสดงสถานะต่าง ๆ ในบัส I^2C

ในรูปที่ 2.14 เป็นไดอะแกรมเวลาที่แสดงถึงการเกิดสภาวะต่าง ๆ บนบัส I^2C ไม่ว่าจะเป็นสภาวะบัสว่าง, เริ่มต้น, ถ่ายทอดข้อมูล, รับรู้และหยุดการถ่ายทอดข้อมูล



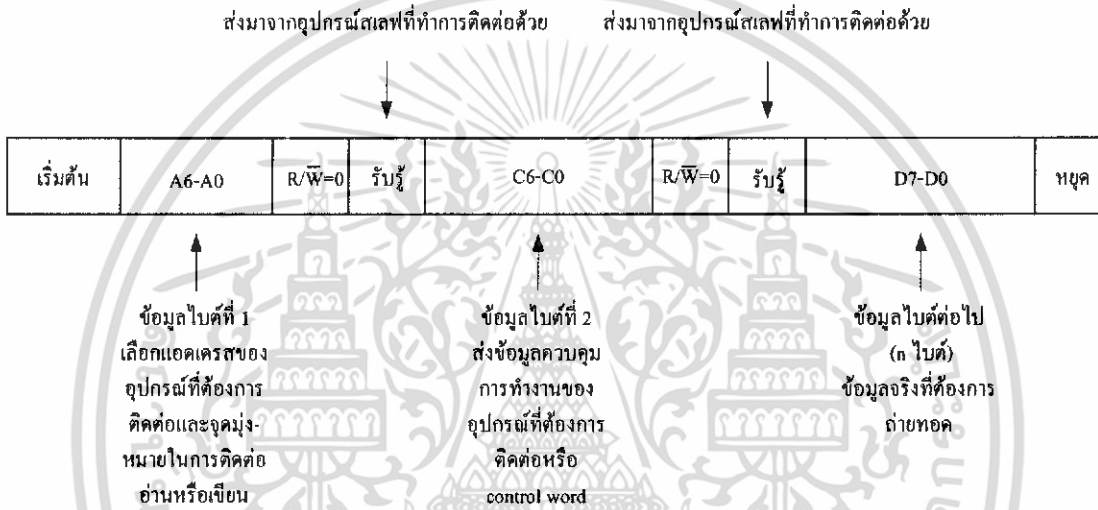
รูปที่ 2.15 แสดงรูปแบบของข้อมูลกำหนดแอดเดรสที่ใช้ในการอ้างถึงแบบ 7 บิต

2.4.4 การทำงานบนบัส I^2C

ก่อนที่จะเริ่มต้นการถ่ายทอดข้อมูลระหว่างอุปกรณ์ต่าง ๆ ที่ต่ออยู่บนบัส ต้องมีการอ้างถึงเสียก่อน โดยการอ้างถึงอุปกรณ์บนบัส I^2C นั้นจะใช้การอ้างถึงแบบ 7 บิตหรือ 10 บิต ในกรณีที่มีอุปกรณ์ต่ออยู่บนบัสไม่มาก ใช้การอ้างถึงแบบ 7 บิตก็พอ แต่ถ้ามีอุปกรณ์ต่ออยู่บนบัสมากกว่า 127 แอดเดรส จำเป็นต้องใช้การอ้างถึงแบบ 10 บิต หลังจากที่ติดต่ออุปกรณ์แต่ละตัวได้เรียบร้อยแล้ว ก็จะเริ่มต้นการถ่ายทอดข้อมูลกันต่อไป

ดังนั้นหัวใจสำคัญในอันดับแรกของการทำงานบนบัส I^2C คือการอ้างถึงอุปกรณ์แต่ละตัว ซึ่งในที่นี้จะอธิบายรายละเอียดของการอ้างอิงถึงทั้ง 2 รูปแบบ คือ

(1) การอ้างถึงแบบ 7 บิต (7-bit addressing)



รูปที่ 2.16 แสดงรูปแบบของข้อมูลอนุกรมที่ใช้ติดต่อกับอุปกรณ์บัส I^2C เมื่อใช้การอ้างถึงแบบ 7 บิต

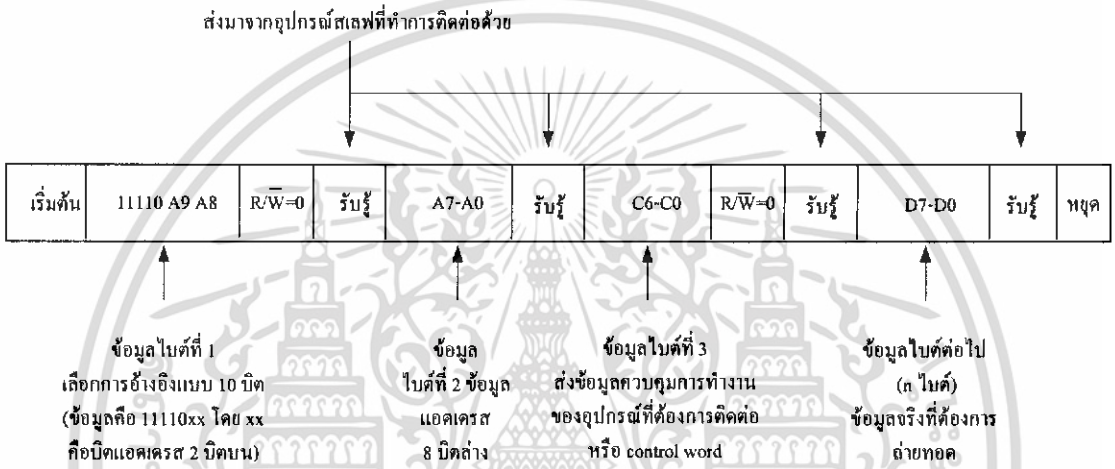
ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้น คือ ข้อมูลที่ใช้ในการอ้างถึงอุปกรณ์ที่ต้องการติดต่อหรือข้อมูลกำหนดแอดเดรส โดยมีรูปแบบแสดงในรูปที่ 2.16 ใน 7 บิตบนรวมทั้งบิต MSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็นบิตกำหนดแอดเดรสคงที่ (fixed - address bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิต ไม่สามารถเปลี่ยนแปลงแก้ไขได้ ถัดมาอีก 3 บิต เป็นบิตกำหนดแอดเดรสที่สามารถกำหนดได้ (programmable address bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0 - A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I^2C ส่วนในบิต LSB เป็นบิตที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวนั้น ๆ หากบิต LSB เป็น "0" หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ

ข้อมูลในไบต์ต่อมา คือ ข้อมูลควบคุม (Control byte) ในแต่ละตัวมีการกำหนดข้อมูลควบคุมที่แตกต่างกันไป ยกตัวอย่าง ไอซีขยายพอร์ตมีข้อมูลควบคุมที่ใช้กำหนดว่า บิตใดเป็นอินพุท บิตใดเป็นเอาท์พุท ในขณะที่ไอซี ADC/DAC ต้องการข้อมูลควบคุมเพื่อกำหนดให้ทำงานเป็นวงจร ADC หรือ DAC เป็นต้น

ข้อมูลในไบต์ต่อมา คือ ข้อมูลที่ทำการถ่ายทอดจริง (Data)

หลังจากที่มีการถ่ายทอดข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง เพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้ ในรูปที่ 2.16 แสดงรูปแบบข้อมูลอนุกรมที่เกิดขึ้นในการติดต่อบนบัส I^2C ของการอ้างถึงแบบ 7 บิต

(2) การอ้างถึงแบบ 10 บิต

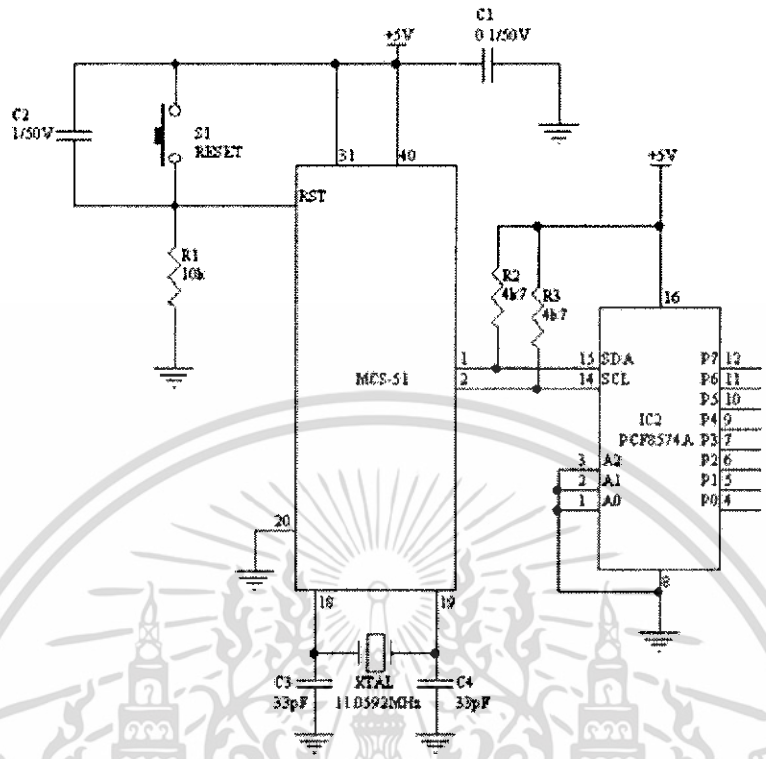


รูปที่ 2.17 แสดงรูปแบบของข้อมูลอนุกรมที่ใช้ติดต่อกับอุปกรณ์บัส I^2C เมื่อใช้การอ้างถึงแบบ 10 บิต

ในการอ้างถึงแบบนี้ ยังคงใช้รูปแบบข้อมูลอนุกรมที่เหมือนกับแบบ 7 บิต หากแต่จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อย โดยในข้อมูลไบต์แรกหลังจากเกิดสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของอุปกรณ์ที่ต้องการติดต่อ ในบิต LSB ของข้อมูลไบต์แรกยังคงเป็นการกำหนดว่าต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟที่ต้องการติดต่อด้วย ข้อมูลไบต์ถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนั้นก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อ

เช่นเดียวกับการอ้างถึงแบบ 7 บิต หลังจากถ่ายทอดข้อมูลครบทุกไบต์ ต้องมีสภาวะรับรู้เกิดขึ้นเพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้ ในรูปที่ 2.17 แสดงรูปแบบขอมูลของการอ้างถึงแบบ 10 บิต

2.4.5 การต่ออุปกรณ์ระบบบัส I^2C กับไมโครคอนโทรลเลอร์ MCS-51



รูปที่ 2.18 แสดงวงจรตัวอย่างการต่อไมโครคอนโทรลเลอร์ MCS-51 กับอุปกรณ์ระบบบัส I^2C

สามารถทำได้ง่ายมากเพียงใช้ขาพอร์ต 2 ขา โดยกำหนดให้ขาหนึ่งเป็น SDA อีกขาหนึ่งเป็น SCL และต่อตัวต้านทานค่าประมาณ 4.7k พูลอัพ (Rp) ที่ขาพอร์ตทั้งสองขา เพียงเท่านี้ก็สามารถติดต่อกับอุปกรณ์ระบบบัส I^2C ได้แล้ว

ในรูปที่ 2.18 เป็นวงจรตัวอย่างการต่อไมโครคอนโทรลเลอร์ MCS-51 เข้ากับระบบบัส I^2C จากวงจรจะใช้ขาพอร์ต P1.0 เป็นขา SDA และ P1.1 เป็นขา SCL อุปกรณ์ที่ทำการติดต่อด้วย คือ ไอซีขยายพอร์ตอินพุทเอาท์พุทเบอร์ PCF8574

2.4.6 การเขียนโปรแกรมติดต่อบัส I^2C

เริ่มต้นด้วยการสร้างสภาวะมาตรฐานของบัส I^2C อันประกอบด้วยสภาวะเริ่มต้น, สภาวะสิ้นสุด การส่งข้อมูล, สภาวะหยุด, สัญญาณนาฬิกาบนขา SCL, การเขียนข้อมูลและอ่านข้อมูลกับอุปกรณ์บนระบบบัส I^2C

2.4.7 การสร้างสภาวะเริ่มต้น

1. เมื่อต้องการติดต่อกับบัส I^2C สิ่งแรกที่ต้องทำสำหรับไมโครคอนโทรลเลอร์ซึ่งถือว่าเป็นอุปกรณ์มาสเตอร์ คือ การทำให้บัสว่างด้วยการกำหนดให้ขา SCL และขา SDA มีลอจิกเป็น "1" ทั้งคู่
2. จากนั้นทำให้ขา SDA มีลอจิก "0" โดยที่ขา SCL ยังคงเป็นลอจิก "1" อยู่
3. กำหนดให้ขา SCL มีลอจิกเป็น "0" ถึงตอนนี้ทั้ง SCL และ SDA จะมีลอจิกเป็น "0" ทั้งคู่ พร้อมที่จะติดต่อได้แล้ว

2.4.8 การสร้างสภาวะหยุด

1. เมื่อต้องการหยุดส่งข้อมูลจะต้องส่งสภาวะหยุดออกไป โดยในตอนแรกต้องกำหนดให้ขา SCL และ SDA เป็นลอจิก "0" ทั้งคู่ก่อน
2. กำหนดให้ขา SCL มีลอจิกเป็น "1" โดย SDA ยังคงมีลอจิกเป็น "0"
3. จากนั้นทำให้ขา SDA มีลอจิกเป็น "1" ซึ่งจะทำให้ระบบบัสกลับเข้าสู่บัสว่างอีกครั้ง พร้อมที่จะรับหรือส่งข้อมูลต่อไป

2.4.9 การส่งข้อมูลลอจิก "0" และลอจิก "1"

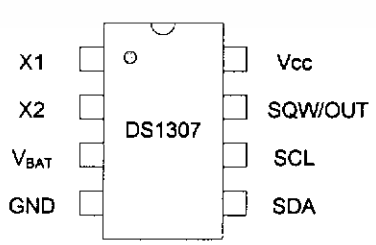
หลังจากที่ทำการส่งบิตเริ่มต้นแล้ว ลำดับต่อไป คือ จะต้องสงข้อมูลควบคุมซึ่งจะเป็นขบวนของลอจิก "0" และลอจิก "1" สำหรับการส่งข้อมูลลอจิก "0" ต้องดำเนินการตามขั้นตอนดังนี้

1. ทำให้ขา SDA เป็น "0" สำหรับการส่งข้อมูลลอจิก "0"
2. ทำให้ขา SCL เป็น "1" สำหรับการป้อนสัญญาณนาฬิกา ในขณะที่ขา SDA ยังคงเป็น "0" อยู่
3. จากนั้นทำให้ขา SCL กลับมามีสถานะเป็นลอจิก "0" เหมือนเดิม

ในขณะที่การส่งข้อมูลลอจิก "1" มีขั้นตอนดังนี้

1. ทำให้ขา SDA มีลอจิกเป็น "1" สำหรับการส่งข้อมูลลอจิก "1"
2. ทำให้ขา SCL เป็น "1" สำหรับการส่งสัญญาณนาฬิกา ในขณะที่ขา SDA ยังคงเป็น "0" อยู่
3. จากนั้นทำให้ขา SCL กลับมามีสถานะเป็นลอจิก "0" เหมือนเดิม

2.5 DS1307



รูปที่ 2.19 การจัดขาของไอซี DS1307 ไอซีสร้างฐานเวลาจริง (RTC)

รายละเอียดขาต่อใช้งานของ DS1307

ในรูปที่ 2.19 แสดงการจัดขาของ DS1307 แต่ละขามีหน้าที่และการใช้งานดังนี้

V_{cc}, GND (ขา 8, 4) ต่อกับไฟเลี้ยง +5 V และ ground

V_{BAT} (ขา 3) ใช้ต่อกับแบตเตอรี่ 3V เพื่อรักษาการทำงานของวงจรสร้างฐานเวลาของ DS1307 ให้คงอยู่ต่อไป แม้ว่าไม่มีไฟเลี้ยงจ่ายให้แก่ DS1307 ชนิดของแบตเตอรี่ที่เหมาะสมคือ แบตเตอรี่แบบลิเธียม ซึ่งมีอุณหภูมิ 25 องศาเซลเซียส

SDA, SCL (ขา 5 และ 6) เป็นขาสำหรับเชื่อมต่อกับกับไมโครคอนโทรลเลอร์บนระบบบัส I^2C

SQW OUT (ขา 7) ที่ขานี้จะมีสัญญาณรูปสี่เหลี่ยมส่งออกมา โดยสามารถเลือกความถี่ได้ 1 Hz, 4.096 kHz, 8.192 kHz และ 32 kHz ในการใช้งานต้องต่อตัวต้านทาน 1k พูลอัพที่ขานี้ด้วย

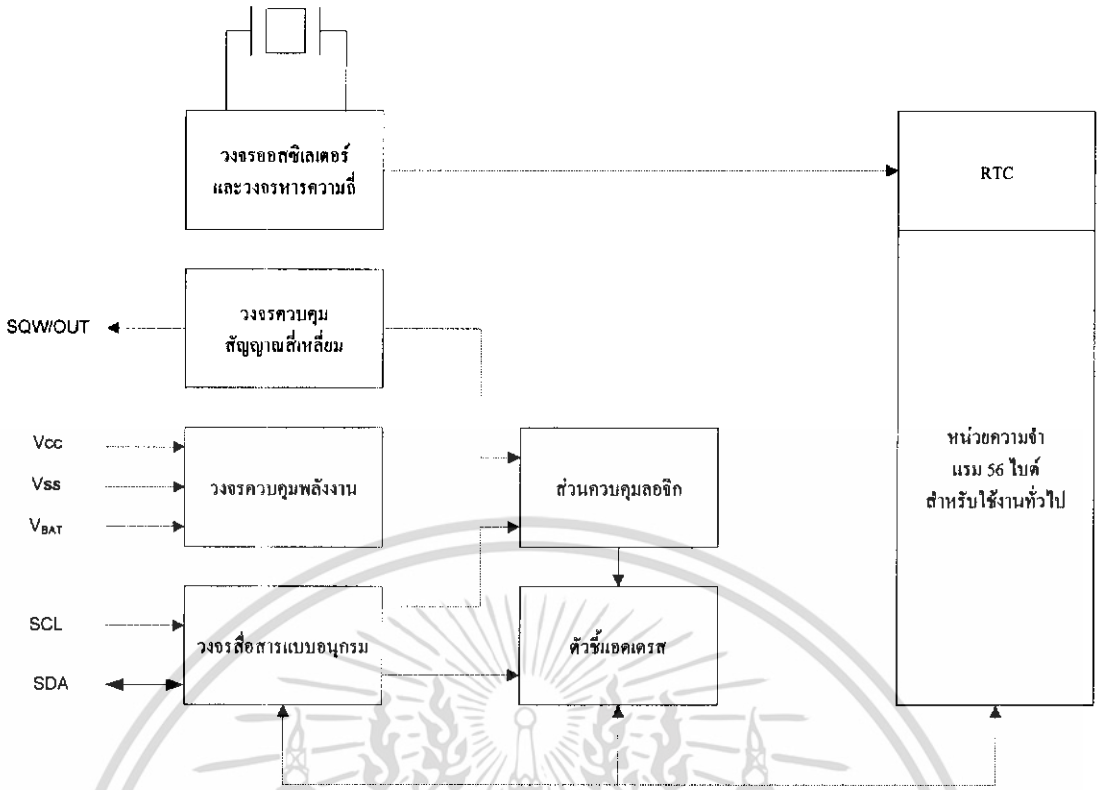
X_1, X_2 (ขา 1 และ 2) ใช้ต่อกับคริสตอลความถี่มาตรฐาน 32.768 kHz เพื่อใช้เป็นฐานเวลาในการสร้างค่าเวลาจริง ในการใช้งานต้องต่อคริสตอลเข้ากับขาทั้งสองนี้และที่แต่ละขาต้องต่อตัวเก็บประจุค่าต่ำๆ ประมาณ 15 pF ครอ่มกับขากราวด์ด้วย

2.5.1 การทำงานของ DS1307

ไอซี DS1307 จัดการเชื่อมต่อในแบบบัส I^2C โดยจะทำงานเป็นอุปกรณ์สเลฟเสมอ ดังนั้นการติดต่อเพื่อใช้งานจึงต้องกำหนดรูปแบบตามที่กำหนดไว้ในการติดต่อแบบ I^2C ในรูปที่ 2.19 แสดงส่วนประกอบหลักที่สำคัญและไดอะแกรมการทำงานของ DS1307 วงจรออสซิลเลเตอร์ถือเป็นหัวใจหลักของไอซี เนื่องจากเป็นจุดเริ่มต้นของการสร้างข้อมูลเวลาจริง ในขณะที่ DS1307 ทำงานที่ขา SQW/OUT จะมีสัญญาณพัลส์สี่เหลี่ยมส่งออกมาตลอดเวลาในกรณีที่มีการอีนาเบิลวงจรกำเนิดสัญญาณพัลส์ที่รีจิสเตอร์ควบคุม ค่าความถี่ของสัญญาณนี้สามารถเลือกได้ 4 ค่า คือ 1 Hz, 4.096 KHz, 8.192 kHz และ 32 kHz พร้อมกันนั้นก็จะมีการเก็บค่าของเวลาไว้ในหน่วยความจำนอนโวลาไทล์แรม ซึ่งมีขนาดรวมทั้งหมด 64 ไบต์ แต่จัดสรรให้ใช้เก็บข้อมูลเวลา 8 ไบต์ และเป็นหน่วยความจำสำหรับเก็บข้อมูลทั่วไปสำหรับผู้ใช้งานอีก 56 ไบต์

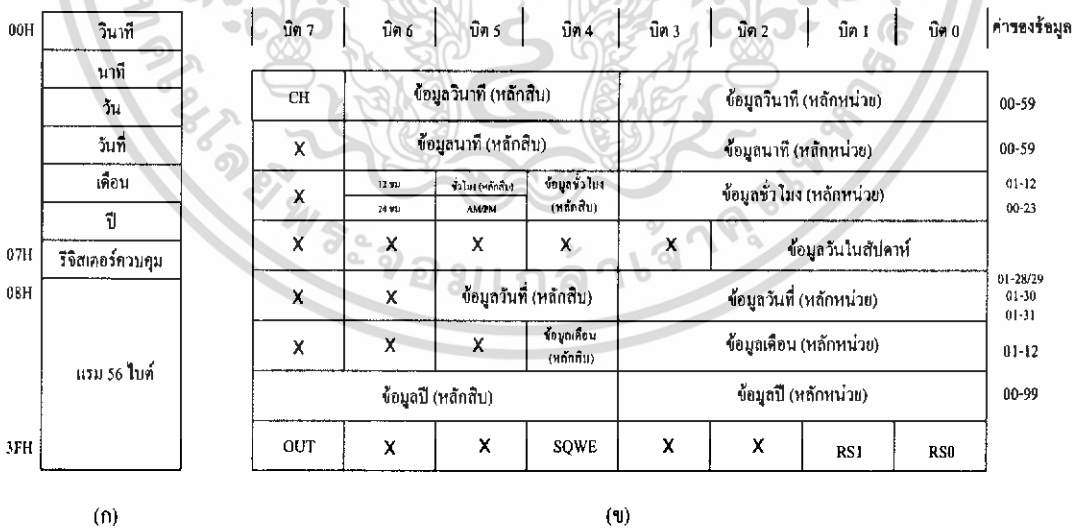
วงจรควบคุมพลังงานไฟฟ้าจะคอยทำหน้าที่ตรวจสอบสภาวะของไฟเลี้ยงไอซี หากไฟเลี้ยงต่ำกว่า $1.25 \times V_{BAT}$ ก็จะควบคุมให้ DS1307 หยุดการทำงานรีเซตค่าตัวนับแอดเดรสภายใน ทำให้ไม่สามารถติดต่อกับ DS1307 ได้ ดังนั้นในการใช้งานรีเซตค่าตัวนับแอดเดรสภายใน ทำให้ไม่สามารถติดต่อกับ DS1307 ต้องระมัดระวังอย่าให้ไฟเลี้ยงมีค่าต่ำกว่า $1.25 \times V_{BAT}$ หรือประมาณ 3.75 ในกรณีที่ใช้ V_{BAT} เท่ากับ 3V ถ้าหากไฟเลี้ยงมีค่าต่ำกว่า V_{BAT} ไอซี DS1307 จะเข้าสู่โหมดสำรองข้อมูลกระแสต่ำทันทีจะไม่มีการส่งสัญญาณพัลส์ออกมาที่ SQW/OUT แต่วงจรสร้างฐานเวลายังคงทำงานเพื่อให้ค่าของเวลาเดินไปอย่างไม่ผิดพลาด เมื่อมีไฟเลี้ยงปรากฏขึ้นอีกครั้ง DS1307 ก็จะสามารถให้ค่าของเวลาที่เป็นจริงแก่ผู้ใช้งานได้ต่อไป

วงจรสื่อสารอนุกรมภายใน DS1307 ได้รับการกำหนดให้ทำงานตามรูปแบบของบัส I^2C เป็นช่องทางการสื่อสารระหว่าง DS1307 กับอุปกรณ์มาสเตอร์ ผู้ใช้งานสามารถเข้าถึงหน่วยความจำที่ใช้เก็บค่าเวลาและหน่วยความจำใช้งานทั่วไปได้โดยการเขียนข้อมูลตามรูปแบบที่กำหนดในระบบบัส I^2C



รูปที่ 2. 20 โครงสร้างภายในของไอซีรีลไทม์คล็อกเบอร์ DS1307

2.5.2 การจัดสรรหน่วยความจำใน DS1307



รูปที่ 2.21 (ก) แสดงการจัดสรรหน่วยความจำแรมภายใน DS1307
(ข) แสดงรายละเอียดของรีจิสเตอร์เก็บค่าเวลาและรีจิสเตอร์ควบคุมของ DS1307

รูปที่ 2.21 (ก) แสดงการจัดสรรพื้นที่ของหน่วยความจำภายใน DS1307 พื้นที่ 7 ไบต์แรกตั้งแต่แอดเดรส 00H-06H เป็นพื้นที่ของรีจิสเตอร์ค่าเวลาใช้ในเก็บข้อมูลเกี่ยวกับเวลา ไบต์ต่อมาที่แอดเดรส 07H เป็นพื้นที่ของรีจิสเตอร์ควบคุมการทำงานของ DS1307 ในรูปที่ 2.21 (ข) แสดงรายละเอียดของรีจิสเตอร์ค่าเวลาและรีจิสเตอร์ควบคุมของ DS1307

ด้วยการจัดสรรพื้นที่แบบนี้ ทำให้ผู้ใช้งานสามารถเรียกข้อมูลเวลาออกมาได้ตามที่ต้องการ โดยไม่จำเป็นต้องอ่านออกมาทั้งหมดก็ได้ ค่าของเวลาทั้งหมดจะอยู่ในรูปของเลขฐานสิบ สำหรับการแสดงเวลาในรูปของชั่วโมงสามารถเลือกได้ว่าต้องการแบบ 12 หรือ 24 ชั่วโมง โดยกำหนดที่บิต 6 ของแอดเดรส 02H และเมื่อเลือกแบบ 12 ชั่วโมง ที่บิต 5 ในแอดเดรสเดียวกันจะใช้ในการแสดงค่า AM/PM โดยถ้าบิตนี้เป็น "1" หมายถึง ค่าชั่วโมงในขณะนี้เป็นช่วงเวลาหลังเที่ยงวัน ในกรณีที่เป็นแบบ 24 ชั่วโมง บิตนี้จะใช้การแสดง ค่า "2" ของหลักสิบในหน่วยชั่วโมง

2.5.3 รีจิสเตอร์ควบคุม

มีแอดเดรสอยู่ที่ 07H มีรายละเอียดของแต่ละบิตดังนี้

OUT (Output control): ใช้ในการควบคุมระดับลอจิกที่ SQW OUT ในกรณีที่ดิสเอเบิลการกำเนิดสัญญาณสี่เหลี่ยม โดยถ้าบิตนี้เป็น "1" ที่ขา SQW OUT ก็จะเป็น "1" ถ้าบิตนี้เป็น "0" ที่ขา SQW OUT ก็จะเป็น "0"

SQWE (Square Wave Enable): ใช้ในการอีนาเบิลวงจรกำเนิดสัญญาณสี่เหลี่ยมที่ขา SQW OUT ถ้าต้องการให้มีสัญญาณสี่เหลี่ยมออกให้กำหนดบิตนี้เป็น "1"

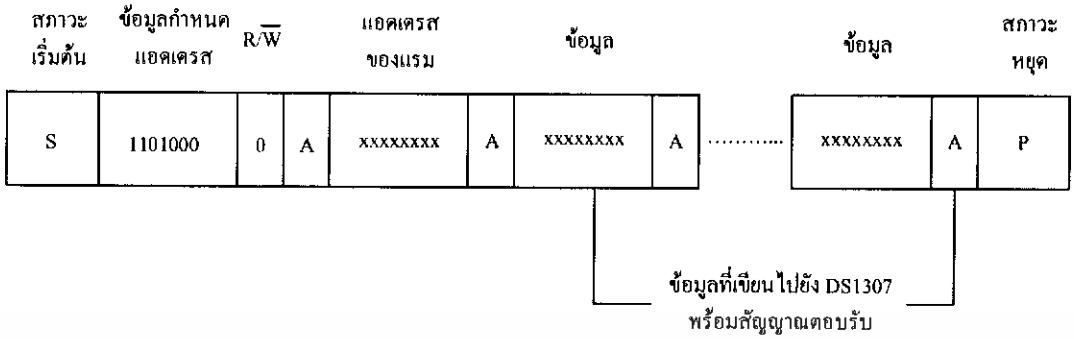
RS1, RS0 (Rate Select): ใช้ในการเลือกความถี่ของสัญญาณสี่เหลี่ยมที่ออกจากขา SQW OUT ดังมีรายละเอียดต่อไปนี้

RS1	RS0	ค่าความถี่ของสัญญาณสี่เหลี่ยม
0	0	1 Hz
0	1	4.096 kHz
1	0	8.192 kHz
1	1	32.768 kHz

2.5.4 โหมดการทำงานของ DS1307

มีด้วยกัน 2 โหมด คือ โหมดเขียนข้อมูลและโหมดอ่านข้อมูล ในการใช้งาน DS1307 ตามปกติจะใช้งานเฉพาะโหมดอ่านข้อมูลเท่านั้น เนื่องจากไมโครคอนโทรลเลอร์จะติดต่อกับ DS1307 จำเป็นอย่างยิ่งที่จะต้องเข้าสู่โหมดการเขียนข้อมูลก่อนเพื่อกำหนดแอดเดรสที่ต้องการอ่านข้อมูล จากนั้นจึงเปลี่ยนโหมดการทำงานมาเป็นโหมดการอ่านข้อมูล

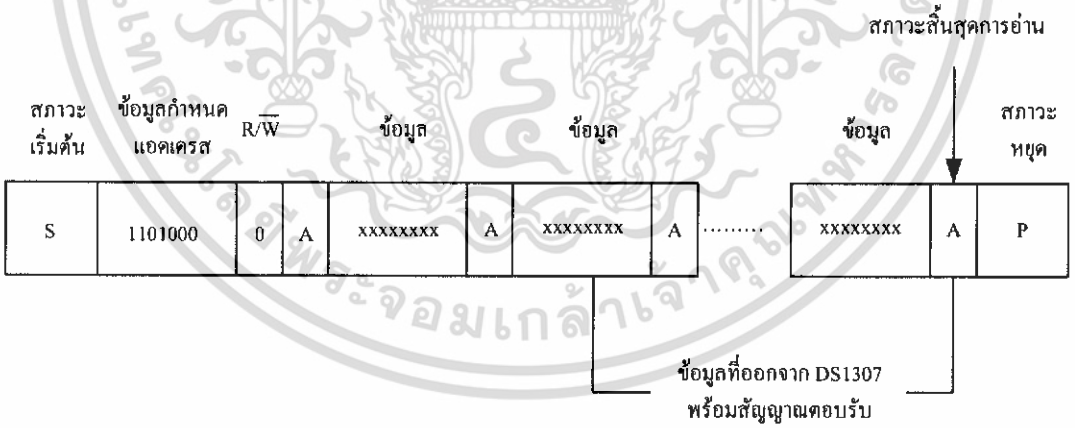
2.5.4.1 โหมดการเขียนข้อมูล



รูปที่ 2.22 รูปแบบของข้อมูลสำหรับติดต่อกับ DS1307 ในโหมดการเขียนข้อมูล

มีรูปแบบดังในรูปที่ 2.22 เริ่มต้นเมื่อไมโครคอนโทรลเลอร์ทำการกำหนดสภาวะเริ่มต้น (START:S) จากนั้นส่งข้อมูลกำหนดแอดเดรส 1101000 ตามด้วยข้อมูลเลือกการเขียน นั่นคือค่า 0 จากนั้นจะรอการตอบรับจาก DS1307 ขั้นตอนต่อมาคือ ส่งข้อมูลเพื่อเลือกแอดเดรสที่ต้องการเขียน จากนั้นรอการตอบรับจาก DS1307 เมื่อมีการตอบรับมาเรียบร้อย ก็เริ่มทยอยเขียนข้อมูลลงไปครั้งละแอดเดรส หลังจากเขียนข้อมูลในแต่ละแอดเดรส จะต้องหยุดรอการตอบรับจาก DS1307 ทุกครั้ง จึงจะสามารถเขียนข้อมูลต่อไปได้ เมื่อเขียนเรียบร้อยแล้วให้ส่งสภาวะหยุด (STOP:P) เป็นอันสิ้นสุดกระบวนการเขียนข้อมูล

2.5.4.2 โหมดการอ่านข้อมูล



รูปที่ 2.23 รูปแบบของข้อมูลสำหรับติดต่อกับ DS1307 ในโหมดการอ่านข้อมูล

มีรูปแบบแสดงในรูปที่ 2.23 เริ่มต้นการทำงานเหมือนกับโหมดการเขียนข้อมูลคือ ไมโครคอนโทรเลอร์กำหนดสภาวะเริ่มต้นแล้วส่งข้อมูลกำหนดแอดเดรสตามด้วยข้อมูลเลือกการอ่าน ซึ่งเท่ากับ 1 จากนั้นรอการตอบรับจาก DS1307 เมื่อตอบรับเรียบร้อย DS1307 จะทยอยส่งข้อมูลออกมาให้ไมโครคอนโทรเลอร์คราวละ 1 แอดเดรสหรือ 1 ไบต์ โดยแอดเดรสที่เลือกอ่านข้อมูลจะต้องมีการกำหนด

มาก่อนล่วงหน้าด้วยโหมดการเขียนข้อมูลวิธีง่ายๆ คือ เข้าสู่โหมดการเขียนข้อมูลก่อน เมื่อถึงจังหวะที่ต้องเขียนข้อมูล ให้ทำการสร้างสภาวะเริ่มต้นและส่งข้อมูลกำหนดแอดเดรสใหม่อีกครั้ง ตามด้วยเลือกโหมดการอ่านข้อมูล ข้อมูลที่ออกมาจาก DS1307 ก็จะเป็นข้อมูลจากแอดเดรสที่กำหนดไว้ก่อนหน้านี้

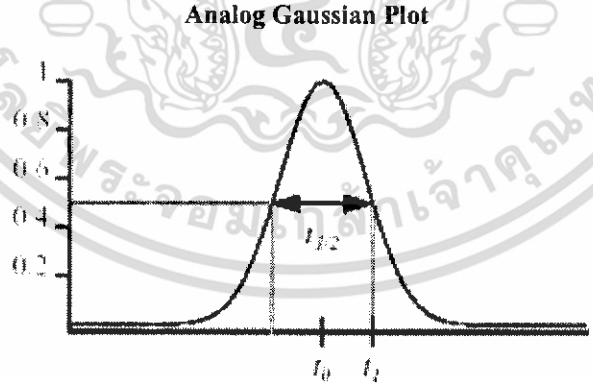
2.6 GFSK (Gaussian Frequency Shift Keying)

เป็นวิธีการมอดูเลต สัญญาณ base band ที่เป็นดิจิตอล โดยจะผ่าน Gaussian filter ก่อนจะผ่านเข้าไปสู่การมอดูเลตแบบ FSK เพื่อทำการลดแบนด์วิทท์ลง แล้วส่งออกไป โดยที่ภาครับก็จะทำการ shaping ข้อมูลรูปแบบเดิมกลับมา

2.6.1 ข้อแตกต่างของ GFSK เมื่อเทียบกับการมอดูเลตแบบ FSK

โดยปกติแล้วการมอดูเลตทั้งสองแบบมีลักษณะที่เกือบจะเหมือนกัน แต่วิธีของ GFSK จะใช้วงจรกรอง Gaussian ซึ่งจะทำให้มีประสิทธิภาพที่ได้ดีกว่า ซึ่งก่อนที่ base band พัลส์ (-1,1) จะเข้าไปในวงจรมอดูเลตของ FSK จะผ่านวงจรกรอง Gaussian ก่อนเพื่อทำให้พัลส์นั้นมีความละเอียดมากขึ้น และเป็นการลดแบนด์วิทด์ลงอีกด้วยทำให้มีประสิทธิภาพที่ได้ดีกว่า นั่นคือข้อเสีย ของการมอดูเลตแบบ FSK ก็คือ ในขณะที่ข้อมูลเปลี่ยน จาก "0" เป็น "1" หรือ "1" เป็น "0" จะเกิดการเปลี่ยนเฟสของแคเรียร์อย่างรวดเร็ว อาจจะสูงขึ้นหรือต่ำลง ซึ่งก็มีผลให้ความถี่แคเรียร์จริงสูงกว่า หรือ ต่ำกว่า f_0 หรือ f_1 ที่กำหนดไว้ นั่นก็คือแบนด์วิทด์จะกว้างขึ้น

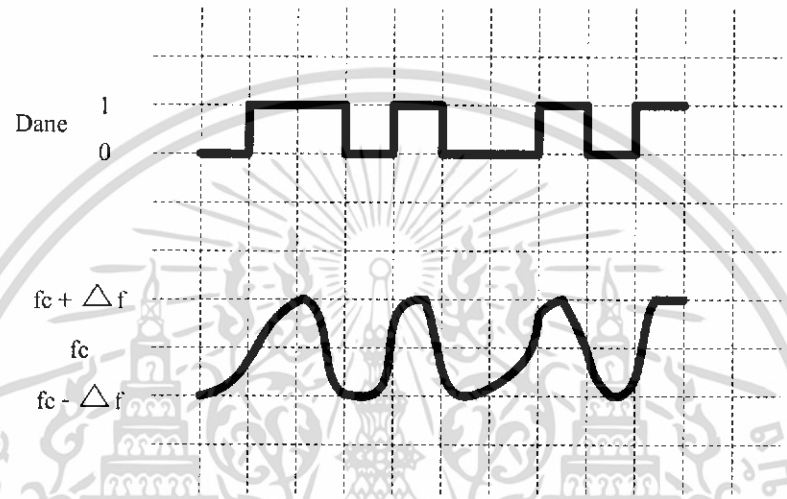
ดังนั้นจึงหาวิธีลดปัญหาดังกล่าว โดยให้การเปลี่ยนแปลงของสัญญาณข้อมูล เป็นแบบค่อยๆ ขึ้นหรือ ค่อยๆ ลง โดยมีความโค้งเป็นแบบ Gaussian pulse ซึ่งเป็นแบบที่ได้คัดเลือกแล้วว่าทำให้มีแบนด์วิทด์แคบใกล้เคียงที่ต้องการ และได้ symbol rate สูง



รูปที่ 2.24 แสดง การพล็อตของ Gaussian pulse

2.6.2 Gaussian filtering

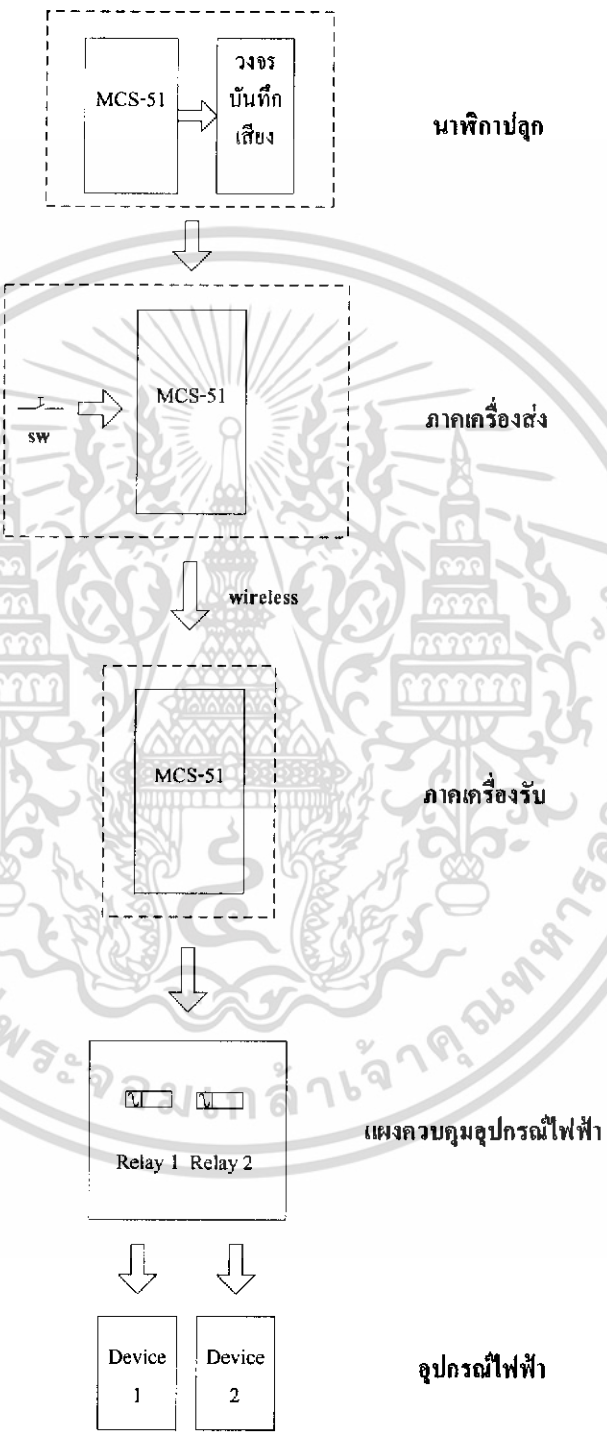
วงจรกรองฟิลเตอร์ Gaussian คือหนึ่งในมาตรฐานของการลดความยาวสเปกตรัมหรือแบนด์วิทด์ โดยปกติจะเรียกว่า Pulse shaping สมมติว่าถ้าเราใช้ค่า -1 แทนจุด f(c) - f(d) และ 1 แทนจุด f(c) + f(d) แล้วเมื่อทำการเปลี่ยนค่าจาก "-1" ไป "1" และ "1" ไป "-1" จะทำให้รูปแบบคลื่นของการมอดูเลต เปลี่ยนแปลงอย่างรวดเร็ว ซึ่งทำให้เกิดการสูญเสียแบนด์วิทด์เป็นจำนวนมาก แต่ถ้าค่อยๆ เคลื่อนย้ายพัลส์ ไปทีละนิดเช่น จาก "-1" ถึง "1" เป็น "-1", "-98", "-93",... "96", "99", "1" และก็ใช้ smoother pulse ไป มอดูเลตแคเรียร์ ก็จะสามารถช่วยให้แบนด์วิทด์มีขนาดลดลงได้



รูปที่ 2.25 แสดงการเปลี่ยนแปลงค่าจาก "-1" ไปยัง "1" เปรียบเทียบระหว่าง การเปลี่ยนแปลงอย่างรวดเร็วและค่อยๆ เปลี่ยนแปลง

บทที่ 3
การคำนวณและการสร้าง

3.1 หลักการทำงาน



รูปที่ 3.1 แสดงบล็อกไดอะแกรมรวมของโครงงาน

ในการควบคุมอุปกรณ์ไฟฟ้าเริ่มต้นที่การตั้งเวลาตามต้องการ และทำการเลือกอุปกรณ์ไฟฟ้าที่ต้องการควบคุมที่แสดงผลบน 7- SEGMENT จากนั้นไมโครคอนโทรลเลอร์ในภาคเครื่องส่งจะรอรับค่าอินพุทจากนาฬิกาปลุกที่สร้างขึ้นจากไมโครคอนโทรลเลอร์ หรือรับการสับสวิตซ์โดยตรงที่เครื่องส่ง หลังจากที่ไมโครคอนโทรลเลอร์ AT89S52 ของภาคเครื่องส่งได้รับสัญญาณอินพุทเข้ามา จะประมวลผล (โดยจะทำงานแบบ Active Low) และทำการส่งสัญญาณแบบไร้สายในวงจรมอดูเลเตอร์แบบ TRW ไปยังภาคเครื่องรับ

ทางด้านภาคเครื่องรับ เมื่อได้รับสัญญาณที่เป็นข้อมูลและทำการประมวลผลว่าจะให้อุปกรณ์ตัวใดทำงานบ้าง จะทำการสั่งงานไปยังแผงควบคุมอุปกรณ์ไฟฟ้าเพื่อควบคุมอุปกรณ์ไฟฟ้าแต่ละตัว

3.1.1 หลักการทำงานของนาฬิกาปลุกและปฏิทิน 100 ปี

ในการสร้างนาฬิกาปลุกจากไมโครคอนโทรลเลอร์ตระกูล MCS-51 (AT89S52) ซึ่งเป็น IC 40 ขา อินพุท เอาท์พุท 4 พอร์ต ในที่นี้ใช้พอร์ตศูนย์ (P0) เป็นพอร์ตเอาท์พุทซึ่งต่ออยู่กับ LCD โดยมี P3.6 และ P3.7 เป็นบิตที่ใช้สำหรับควบคุม LCD

P2.3 และ P2.4 เป็นบิตที่ใช้ควบคุมการส่ง I^2C

P1.0 เป็นบิตที่ใช้ควบคุมค่าที่ส่งไปให้แก่วงจร voice record ทำงาน

สวิตซ์ที่ใช้สำหรับตั้งค่า วัน เดือน ปี ชั่วโมง นาที และ นาฬิกาปลุกต่ออยู่ที่ P2.0

สวิตซ์ที่ใช้สำหรับเพิ่มค่าต่ออยู่ที่ P2.1

สวิตซ์ที่ใช้สำหรับออกจากการตั้งค่าต่ออยู่ที่ P2.2

สวิตซ์ที่ใช้สำหรับกดเปิด - ปิด เสียงของการปลุกต่ออยู่ที่ P2.5

หลักการของนาฬิกาและปฏิทิน 100 ปี คือ ใช้ไอซี DS1307 ซึ่งเป็นไอซีสร้างฐานเวลาจริง (Real Time Clock) โดยตัว RTC สามารถทำงานได้โดยอาศัยพลังงานจากแบตเตอรี่ การอ่านค่าจะใช้การเชื่อมต่อเป็นระบบ 2-WIRE SERIAL DATA BUS หรือที่เรียกว่าระบบบัส I^2C ซึ่งจะใช้สายสัญญาณ 2 เส้นคือ SCL และ SDA โดยจะทำการอ่านค่าของ RTC ตลอดเวลาเพื่อทำการแสดงผลที่หน้าจอ LCD

มีการเปรียบเทียบเวลาที่อ่านได้จากนาฬิกา และเวลาที่อ่านได้จากค่านาฬิกาปลุกที่ตั้งไว้ ซึ่งค่านี้จะถูกเก็บไว้ในไอซีสร้างฐานเวลาจริง ในส่วนที่เป็นแรม โดยเมื่อเปิดเครื่องขึ้นมาจะทำการโหลดค่านี้มาก่อน ถ้าตรงกันก็จะทำให้วงจรอัดเสียงที่ต่อไว้ส่งเสียงดังขึ้น และจะหยุดเมื่อได้ทำการกดปุ่ม Stop Alarm แต่หากไม่มีการกดปุ่มเสียงก็จะดังไปตลอดเป็นเวลา 10 นาทีและจะดับเองโดยอัตโนมัติ ซึ่งสวิตซ์ที่ใช้นี้เป็นสวิตซ์แบบกดจะได้ลอจิก “1” แต่ถ้าปล่อยจะได้ลอจิก “0”

ส่วนของการตั้งค่าจะใช้สวิตซ์ทั้งหมด 3 ตัว โดย

ตัวที่ 1 มีไว้สำหรับกดเวลาเพื่อตั้งค่า และใช้ในการเปลี่ยนโหมดการตั้งค่า

ตัวที่ 2 มีไว้สำหรับเพิ่มค่าของเวลาหรือปฏิทิน

ตัวที่ 3 มีไว้สำหรับกดเพื่อออกจากโหมดการตั้งค่า จากนั้นจะแสดงวันและเวลาตามปกติ

สวิตซ์ทั้ง 3 ตัวนี้จะเป็นสวิตซ์แบบกดติดปล่อยดับ ซึ่งต่อพูลอัพไว้ หรือก็คือถ้ากดจะได้ลอจิก “0” แต่ถ้าปล่อยจะได้ลอจิก “1”

3.1.2 หลักการทำงานของภาคเครื่องส่ง

ในขั้นตอนการส่งสัญญาณจากภาคเครื่องส่งไปสู่ภาคเครื่องรับจะใช้ไมโครคอนโทรลเลอร์ AT89C51 เป็นตัวควบคุมการส่ง โดยที่มีการกำหนดพอร์ทของสวิตซ์แต่ละตัวดังนี้

สัญญาณจากนาฬิกาดิจิตอลที่ขา P1.0

สัญญาณจากสวิตซ์ Manual ที่ P1.1

สัญญาณจากสวิตซ์ Stop อุปกรณ์ที่ P1.2

สัญญาณสวิตซ์เลือกอุปกรณ์ที่ต้องการที่ P1.3

สัญญาณควบคุมอุปกรณ์ตัวที่หนึ่งที่ P2.0

สัญญาณควบคุมอุปกรณ์ตัวที่สองที่ P2.1

จากนั้นจะเป็นการดำเนินการเลือกอุปกรณ์และสั่งการให้อุปกรณ์ทำงานเมื่อมีสัญญาณเข้ามา โดยเริ่มแรกพอร์ตของสวิตซ์แต่ละตัวจะมีค่าเป็น "0" และจะเปลี่ยนเป็น "1" ก็ต่อเมื่อมีสัญญาณจากนาฬิกาหรือมีการกดสวิตซ์ให้ทำงาน โดยจะทำการตรวจสอบส่วนของการเลือกอุปกรณ์ก่อนว่าต้องการให้อุปกรณ์ใดทำงาน แล้วจึงมาตรวจสอบว่ามีสัญญาณแสดงความประสงค์ที่จะให้อุปกรณ์ทำงานหรือไม่

ขั้นตอนสุดท้ายจะเป็นการส่งสัญญาณข้อมูลที่เรากำหนดไว้ไปยังโมดูลไร้สาย TRW (ตัวส่ง) โดยทำการส่งข้อมูลเป็นสองไบต์คือ BUFFER 1 เลือกอุปกรณ์ทำงานและ BUFFER 2 สั่งให้อุปกรณ์ทำงาน

3.1.3 หลักการทำงานของโมดูลไร้สาย TRW

การส่งสัญญาณด้วยโมดูลไร้สาย TRW โดยใช้ไมโครคอนโทรลเลอร์ AT89C2051 เป็นตัวควบคุม เริ่มต้นด้วยการเซตพอร์ตสัญญาณเป็นดังนี้

ขา CE	BIT P1.2
ขา CS	BIT P1.3
ขา DAT	BIT P1.5
ขา CLK	BIT P1.4
ขา DR1	BIT P1.6

หลังจากนั้นจะเป็นการเซตโหมดตัว TRW เพื่อให้อยู่ในสภาวะที่พร้อมจะทำงานรับและส่ง โดยจะพิจารณาข้อมูลทั้งสิบแปดไบต์ และทำการเขียนข้อมูลลงไปใน TRW เรียงจากซ้ายไปขวาตามค่าที่ได้กำหนดไว้ในตัว TRW

ในกรณีของภาคส่งโมดูลไร้สาย TRW จะทำการรับข้อมูลจากไมโครคอนโทรลเลอร์ และเก็บค่าไว้ แล้วจึงส่งข้อมูลไปยังฝั่งภาครับสัญญาณ

เมื่อโมดูลไร้สาย TRW ในส่วนภาครับได้รับสัญญาณมาแล้ว จะทำการส่งข้อมูลที่ได้ไปยังไมโครคอนโทรลเลอร์ AT89C2051 อีกตัวที่คอยรับข้อมูลเพื่อนำไปประมวลผลต่อไป

3.1.4 หลักการทำงานของภาคเครื่องรับ

ในส่วนแรกจะเป็นการรับข้อมูลจากโมดูลไร้สาย TRW (ตัวรับ) ซึ่งจะเก็บข้อมูลที่ได้มาไว้ในบัฟเฟอร์หนึ่งและสองตามลำดับ แล้วจึงนำค่าบัฟเฟอร์ไปเก็บไว้ในรีจิสเตอร์ศูนย์และหนึ่ง ซึ่งจะนำค่าที่ได้นี้ไปเปรียบเทียบ จากนั้นโปรแกรมจะทำการเซตบิตให้กับ P2.0 และ P2.1 ซึ่งเป็นการส่งสัญญาณสั่งการให้อุปกรณ์แต่ละตัวทำงาน

3.2 การคำนวณ

3.2.1 หาค่า delay ของนาฬิกาปลุก

```
                    ACALL     I2C_DELAY          2 MC
I2C_DELAY:          MOV       R6, #0CH           1 MC
I2C_DELAY_1:        NOP                          1 MC
                    NOP                          1 MC
                    DJNZ      R6, I2C_DELAY_1    2 MC
                    RET                          2 MC
```

เพราะฉะนั้น

Delay ทั้งหมด = 2+1+ (4x12) +2 = 53 MC

ใช้ Clock 11.0592 MHz มี 12 MC ดังนั้น 1 MC = 12 / 11.0592 M = 1.085 us

ดังนั้น 53 MC ใช้เวลา = (1.085u)(53) = 57.51 us

```
                    ACALL     LCD_DELAY          2 MC
LCD_DELAY:          MOV       R7, #02D           1 MC
LCD_DELAY_1:        MOV       R6, #0E6D          1 MC
LCD_DELAY_2:        NOP                          1 MC
                    NOP                          1 MC
                    DJNZ      R6, LCD_DELAY_2    2 MC
                    DJNZ      R7, LCD_DELAY_1    2 MC
                    RET                          2 MC
```

เพราะฉะนั้น

Delay ทั้งหมด = 2+1+ {((4 x 230) +3) (2)} +2 = 1851 MC

ใช้ Clock 11.0592 MHz มี 12 MC ดังนั้น 1 MC = 12 / 11.0592 M = 1.085 us

ดังนั้น 53 MC ใช้เวลา = (1.085u) (1851) = 2 ms

```
                    ACALL     DELAY_10ms         2 MC
```

DELAY_10ms:	MOV	R3, #05D	1 MC
DELAY_10ms_1:	ACALL	LCD_DELAY	2 MC
	DJNZ	R7, DELAY_10ms_1	2 MC
	RET		2 MC

เพราะฉะนั้น

Delay ทั้งหมด = 2+1+ {(1851+2) (5)}+2 = 9270 MC

ใช้ Clock 11.0592 MHz มี 12 MC ดังนั้น 1 MC = 12 / 11.0592 M = 1.085 us

ดังนั้น 53 MC ใช้เวลา = (1.085u)(9270) = 10.05 ms

	ACALL	DELAY_100ms	2 MC
DELAY_100ms:	MOV	R2, #10D	1 MC
DELAY_100ms_1:	ACALL	DELAY_10ms	2 MC
	DJNZ	R2, DELAY_100ms_1	2 MC
	RET		2 MC

เพราะฉะนั้น

Delay ทั้งหมด = 2+1+ {(9270+2) (10)} +2 = 92725 MC

ใช้ Clock 11.0592 MHz มี 12 MC ดังนั้น 1 MC = 12 / 11.0592 M = 1.085 us

ดังนั้น 53 MC ใช้เวลา = (1.085u) (92725) = 100.6 ms

3.2.2 หาค่า Link Margin

ตารางที่ 3.1 แสดงการกำหนดค่าการดำเนินการของ RF

RX2 _EN	CM	RFDR_SB	XO_F			RF_PWR	
15	14	13	12	11	10	9	8

บิตที่13 กำหนดความเร็วในการส่ง

ถ้ากำหนดให้เป็น 0 ความเร็วเท่ากับ 250 kbps

ถ้ากำหนดให้เป็น 1 ความเร็วเท่ากับ 1 mbps

หมายเหตุ ถ้าความเร็วมีค่าเท่ากับ 250 kbps ค่า Sensitivity ของตัวรับจะมีค่าเท่ากับ -90 dBm และระยะทางมากสุดที่สามารถส่งได้คือ 280 เมตร

ถ้าความเร็วมีค่าเท่ากับ 1 mbps ค่า Sensitivity ของตัวรับจะมีค่าเท่ากับ -80 dBm และระยะทางมากสุดที่สามารถส่งได้คือ 150 เมตร

*ในที่นี้กำหนดให้ความเร็วเท่ากับ 1 mbps ฉะนั้นค่า Sensitivity คือ -80 dBm และระยะทางมากสุดที่สามารถส่งได้คือ 150 m.

ตารางที่ 3.2 แสดงการกำหนดค่ากำลังที่ใช้ในการส่งข้อมูล

RF OUTPUT POWER		
D9	**D8**	**P [dBm]**
0	0	-20
0	1	-10
1	0	-5
1	1	0

บิตที่ 8-9 เป็นการกำหนดค่ากำลังที่ใช้ส่งข้อมูล

*ในที่นี้กำหนดให้บิต D9 และ D8 เป็น 1

ดังนั้นพลังงานที่ใช้ส่งคือ 0 dBm นั่นคือเท่ากับ 1 mW

ตารางที่ 3.3 แสดงการกำหนดค่าความถี่ที่ใช้ในการส่งข้อมูล

RF CH#							RXEN
7	6	5	4	3	2	1	0

จากตารางที่ 3.3 บิต 1-7 กำหนดความถี่ของ NRF2401 ที่ใช้ในการส่งข้อมูล โดยค่าความถี่สามารถกำหนดจากสูตร

$$\text{Channel (RF)} = 2400\ \text{MHz} + (\text{RF\_CH\#}) \times 1.0\ \text{MHz}$$

กำหนดให้บิต0 (RXEN) = 0 เป็นการกำหนดเป็น Tx operation

*เลือกส่งที่ Channel 2 ดังนั้นกำหนดให้ D7D6D5D4D3D2D1D0=00000100

แปลงเป็นเลขฐาน 16 = 04H

แปลงเป็นเลขฐาน 10 = 4

RF_CH# = 4

ดังนั้นความถี่ที่ใช้ในการส่งคือ

$$\text{Channel (RF)} = 2400\ \text{MHz} + (4) \times 1.0\ \text{MHz}$$

$$= 2404\ \text{MHz}$$

หมายเหตุ ความถี่ที่ใช้ในการส่งสามารถตั้งค่าได้ในช่วง 2400 MHz ถึง 2527 MHz เท่านั้น

3.2.3 หาค่า Link Margin ที่ Line of Sight

กำหนดให้ TxP (Output Power) = 0 dBm

d = 150 m (1Mbps)

f = 2.404 GHZ

$C = f\lambda$: $\lambda = C/f = 3 \times 108/2.404 \times 109 = 0.1248$ m.

สมการการคำนวณ :RxP(dBm) = TxP + TxG - TxL - FSL - ML + RxG – RxL

Link Margin (dB) = RxP (dBm)-Rx Sensitivity (dBm)

โดยที่ :

RxP = กำลังที่รับได้ (dBm)

TxP = กำลังที่ส่ง (dBm)

TxG = Gain ของเสาอากาศทางด้านส่ง (dBi)

TxL = การลดทอนของเสาอากาศทางด้านส่ง (dBi)

FSL = Free space loss หรือ Path loss (dB)

ML = การลดทอนที่เกิดจากปัจจัยอื่นๆ (fading, body loss, polarization mismatch) (dB)

RxG = Gain ของเสาอากาศทางด้านรับ (dBi)

RxL = การลดทอนของเสาอากาศทางด้านรับ (dB)

กำหนดให้ TxP = 0 dBm

TxG = RxG = 0 dBi

เนื่องจากเป็นการคำนวณทางอุดมคติจึงกำหนดให้ไม่มีการลดทอนหรือมีน้อยมากเพราะฉะนั้น

TxL = RxL = ML = 0dB

สมการ Free Space Loss (FSL); $L(s) = 10\log(\frac{4\pi d}{\lambda})^2$ dB.

$= 10\log (4\times3.14\times150/0.1248)^2$

$= 83.577$ dB

ทฤษฎี : RxP = TxP + TxG - TxL - FSL - ML + RxG - RxL

RxP = 0 dBm + 0 dBi - 0 dB - 83.577 dB – dB + dBi

= -83.577 dBm

Sensitivity = -80 dBm

ดังนั้น : Link Margin (dB) = RxP (dBm) - Rx Sensitivity (dBm)

= -83.577 - (-80) dB

= -3.577 dB

3.2.4 หาค่าพลังงานที่สูญเสีย

จากการทดลอง ระยะทางที่มากที่สุดที่สามารถรับสัญญาณได้คือ 115.3 m จากอาคารเรียนรวม A ถึงอาคารเรียนรวม B

$$FSL = 10\log (4\times3.14\times115.3/0.1248)^2$$

$$= 81.292 \text{ dB}$$

เนื่องจากสามารถส่งสัญญาณได้เพียง 115.3 m เท่านั้นแสดงว่า ในทางปฏิบัติ ต้องเกิดการสูญเสียพลังงานการลดทอนขึ้นซึ่งอาจเป็นค่าการสูญเสียของ ML หรือ การลดทอนจากปัจจัยอื่นได้ ดังนั้น

$$RxP = TxP + TxG\text{-}TxL - FSL + RxG\text{-} RxL – (ML+Others\ Loss)$$

โดยให้ : Attenuation = (ML+Others Loss)

= FSL (ทฤษฎี) - FSL (การทดลอง)

= 83.577 - 81.292

= 2.285 dB

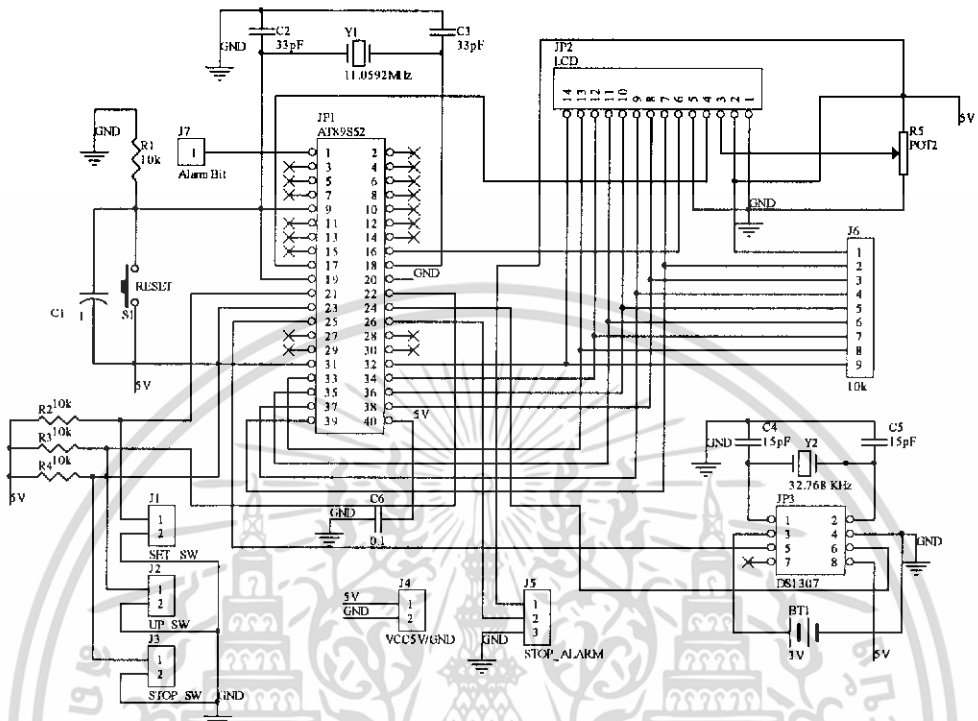
สรุป : ในทางปฏิบัติจะเกิดการลดทอนขึ้น 2.285 dB ซึ่งจะทำให้เราสามารถส่งสัญญาณได้ระยะทางสูงสุดเพียง 115.3 m เท่านั้น

สาเหตุที่คาดว่าก่อให้เกิดการสูญเสียพลังงาน

- อาจเกิดจากสิ่งกีดขวางรอบสถานที่ทำการทดลอง เช่นต้นไม้ รถยนต์ ทำให้ไม่ใช่สถานที่โล่งจริงตามทฤษฎี
- เกิดจากตัวแปรทางสภาพภูมิอากาศ ซึ่งทำให้เกิดการสูญเสียของพลังงาน และเราไม่ได้นำตัวแปรเหล่านี้มาคำนวณด้วย
- เนื่องจากเป็นการส่งแบบไร้สาย ฉะนั้นขณะทำการส่งอาจเกิดคลื่นๆ อื่นรบกวนโดยเฉพาะคลื่นที่อยู่ในย่านความถี่เดียวกัน ทำให้เกิดการสูญเสียพลังงานได้

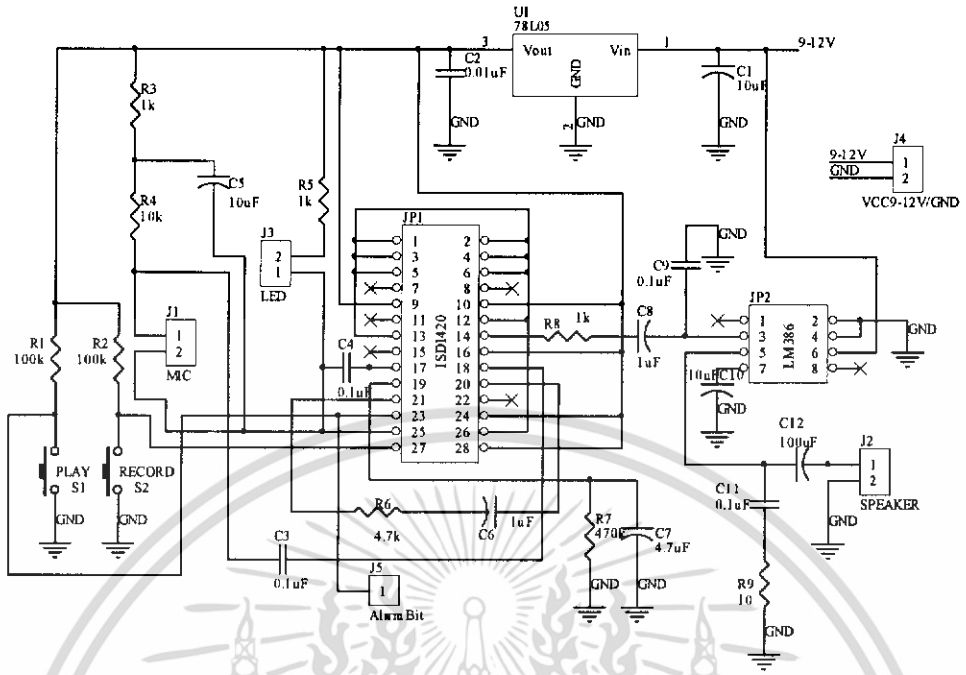
3.3 วงจรการทำงาน

3.3.1 วงจรนาฬิกา



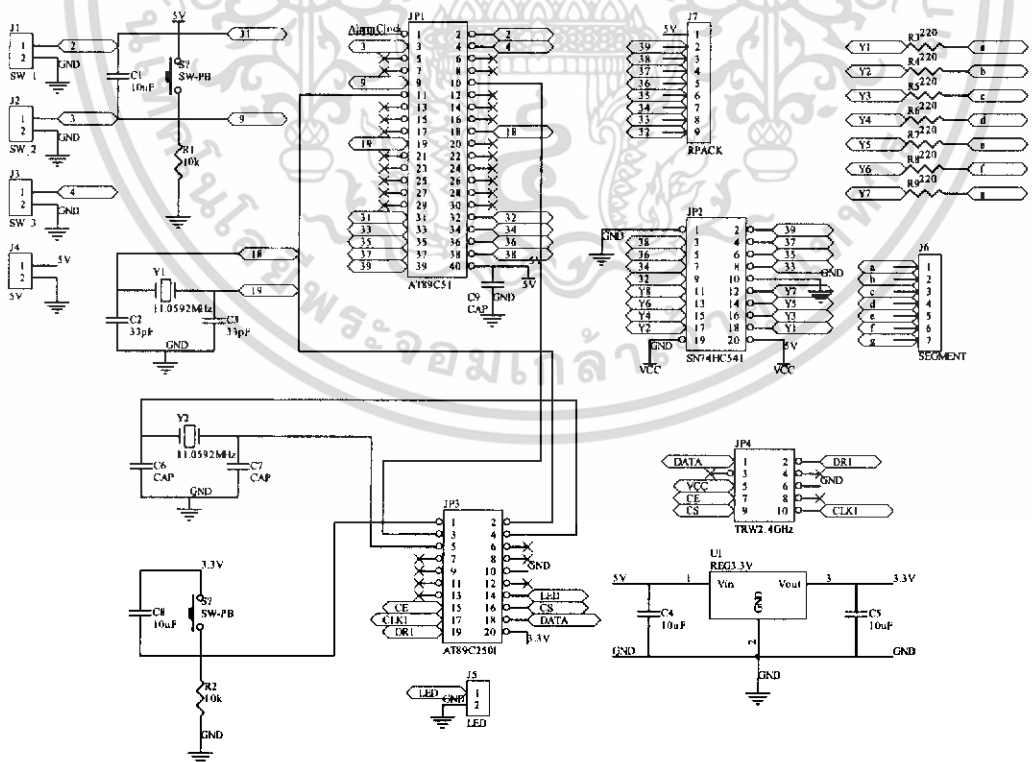
รูปที่ 3.2 แสดงวงจรนาฬิกา

3.3.2 วงจรบันทึกเสียง



รูปที่ 3.3 แสดงวงจรที่ใช้ในการบันทึกเสียงสำหรับตั้งปลุก

3.3.3 วงจรภาคเครื่องส่ง



รูปที่ 3.4 แสดงวงจรในส่วนของภาคเครื่องส่ง

จากรูปที่ 3.4 สัญญาณอินพุทของวงจรทั้งหมดประกอบด้วย

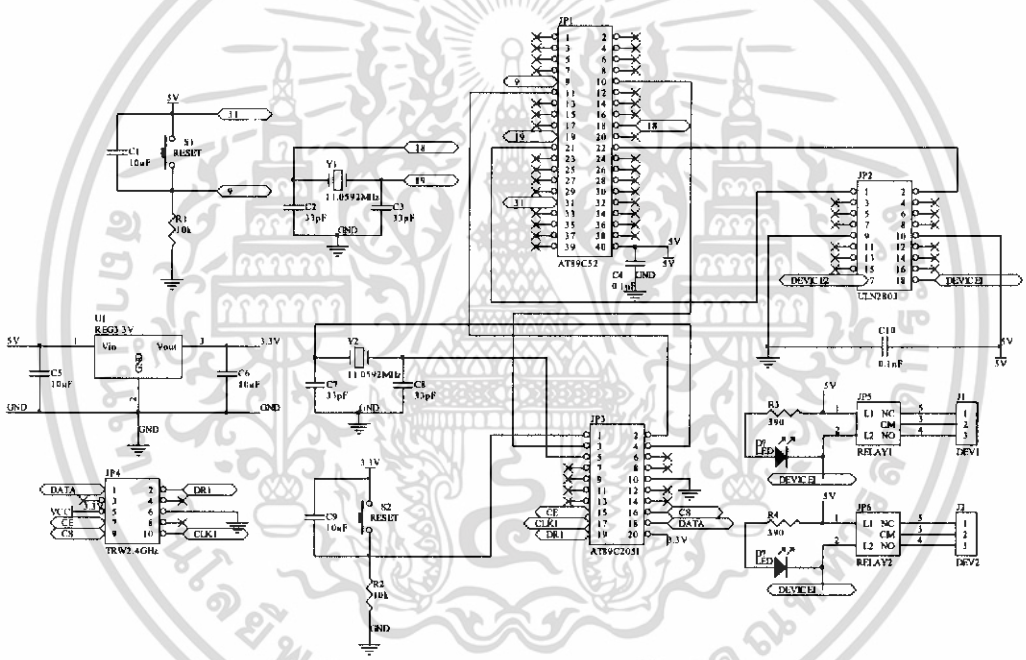
Alarm Clock: เป็นสัญญาณควบคุมการเปิดอุปกรณ์ไฟฟ้าให้แก่วงจร โดยปกติแล้วนาฬิกาจะให้สัญญาณเป็น “1” แต่เมื่อมีการกดปุ่มยกเลิกการตั้งปลุก สัญญาณที่ได้จะมีการเปลี่ยนแปลงจาก “1” เป็น “0” และกลับเป็น “1” อีกครั้ง

SW1: เป็นสัญญาณควบคุมการทำงานของอุปกรณ์ไฟฟ้าที่สามารถใช้ทดแทนสัญญาณ Alarm Clock ได้ ในกรณีที่ต้องการสั่งงานทันทีโดยไม่ต้องรอสัญญาณจากนาฬิกาปลุก

SW2: เป็นสัญญาณสั่งหยุดการทำงานของอุปกรณ์ไฟฟ้า

SW3: เป็นสวิตช์ที่ใช้ในการเลือกอุปกรณ์ไฟฟ้าที่ต้องการควบคุม ซึ่งจะมีการแสดงผลบน 7 – SEGMENT

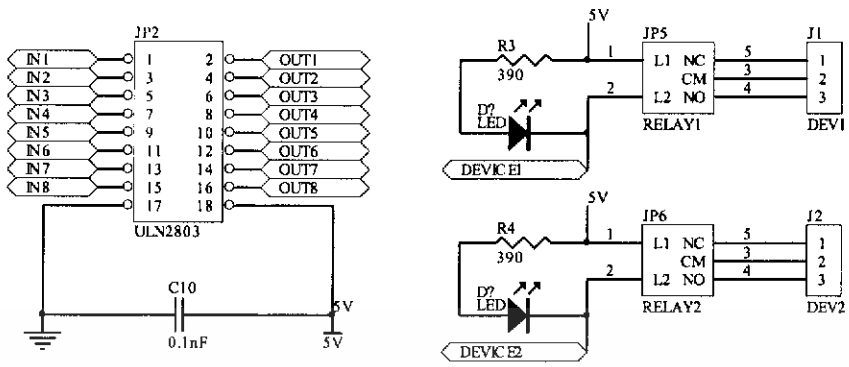
3.3.4 วงจรภาคเครื่องรับ



รูปที่ 3.5 แสดงวงจรในส่วนของภาคเครื่องรับ

จากรูปที่ 3.5 เป็นการแสดงการทำงานของวงจรภาคเครื่องรับ โดยที่ไมโครคอนโทรลเลอร์ MCS-51(AT89C51) จะทำหน้าที่รอรับสัญญาณที่ส่งมาจากภาคส่ง ซึ่งเป็นสัญญาณแบบอนุกรมหลังจากนั้นก็นำสัญญาณแต่ละขาไปควบคุมการ ปิด-เปิดอุปกรณ์ไฟฟ้าได้

3.3.5 วงจรควบคุมรีเลย์

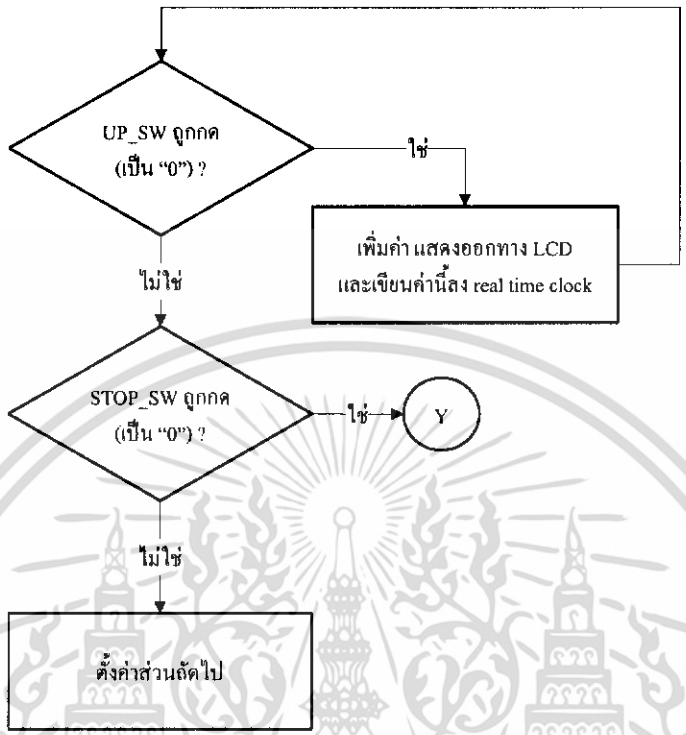


รูปที่ 3.6 แสดงวงจรในส่วนของการควบคุมรีเลย์

จากรูปที่ 3.6 เป็นส่วนการควบคุมรีเลย์ซึ่งเชื่อมต่อมาจากภาคเครื่องรับ โดยจะประกอบด้วยไอซี ULN 2803 ซึ่งทำหน้าที่เป็นตัวขับรีเลย์ เมื่อหน้าสัมผัสของรีเลย์เปลี่ยนตำแหน่ง ก็จะเป็นการควบคุมการจ่ายกำลังให้กับเครื่องใช้ไฟฟ้าที่ผู้ทำการทดลองต้องการจะควบคุมสถานะ

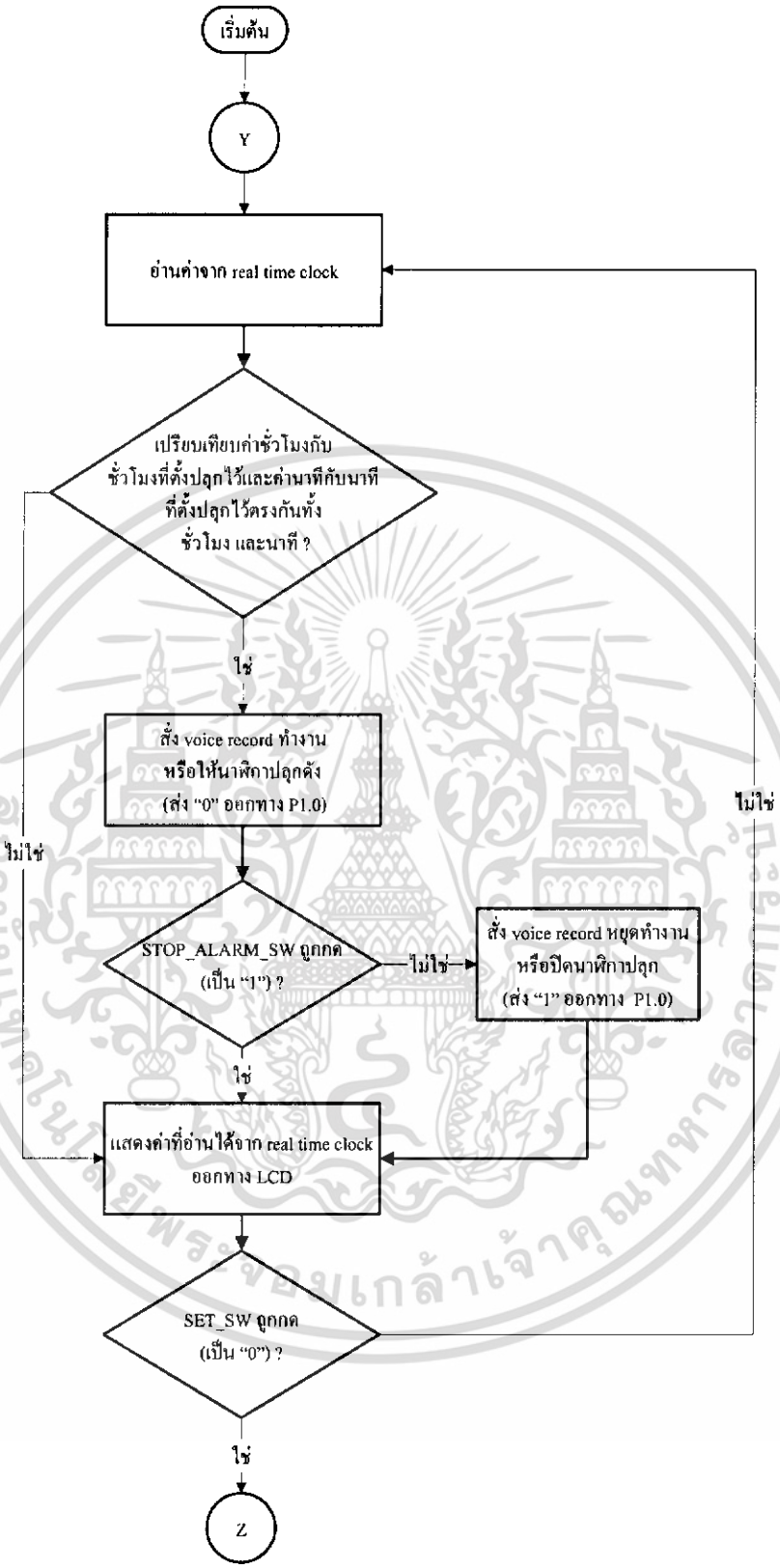
3.4 แผนภูมิแสดงการทำงานของแต่ละระบบ

3.4.1 การตั้งเวลาแก่นาฬิกาปลุก

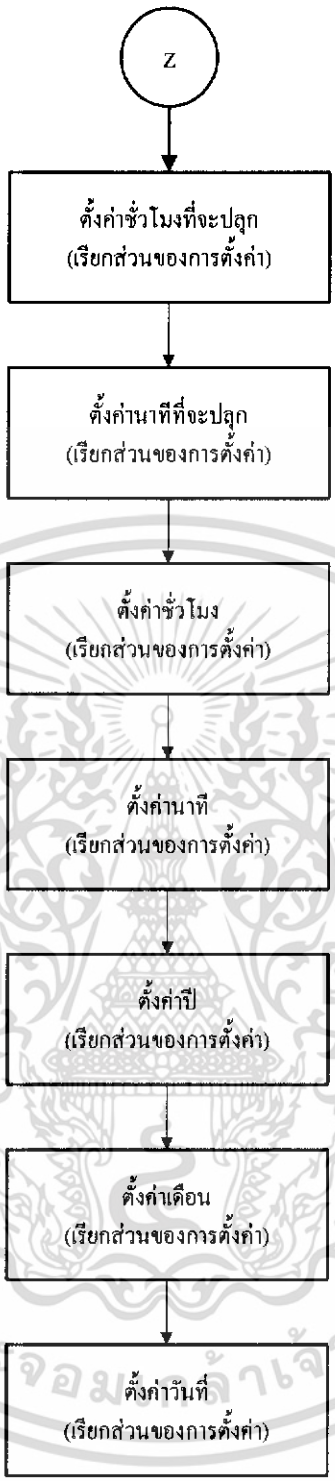


รูปที่ 3.7 แสดงแผนภูมิการทำงานของระบบในการตั้งเวลาแก่นาฬิกา

3.4.2 การทำงานของนาฬิกาปลุกและปฏิทิน 100 ปี

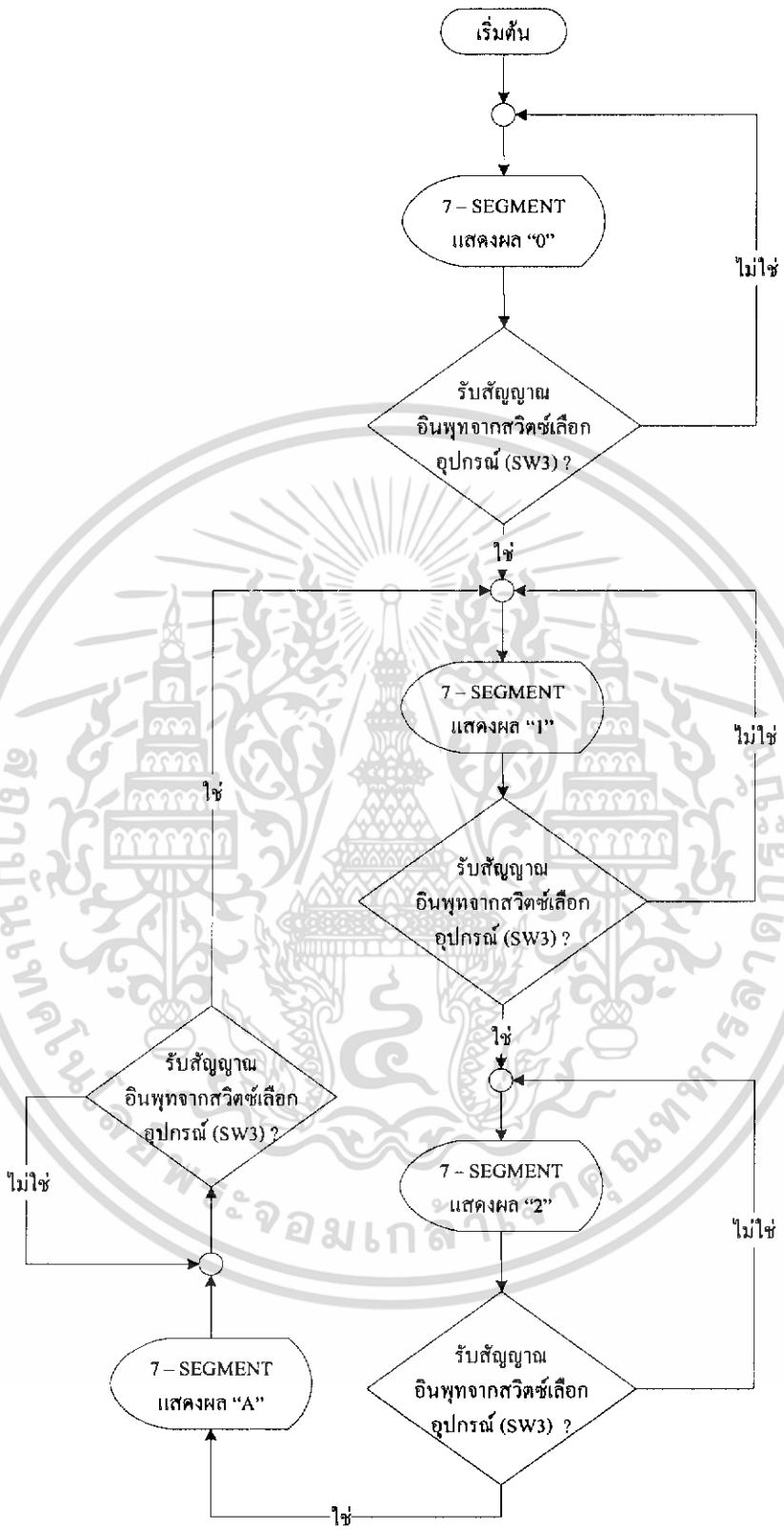


รูปที่ 3.8 แสดงแผนภูมิการทำงานรวมของนาฬิกาปลุกและปฏิทิน 100 ปี



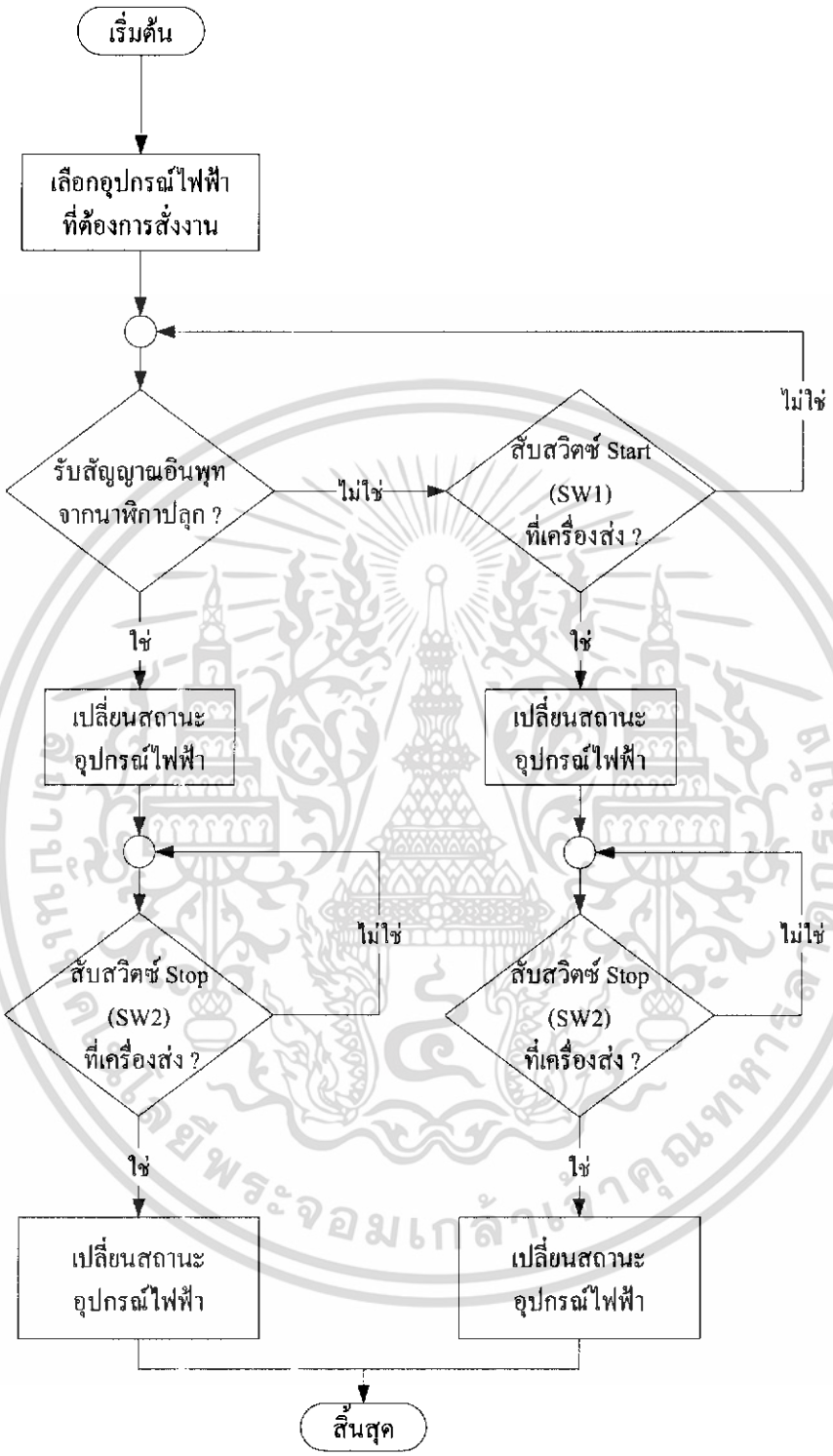
รูปที่ 3.8 แสดงแผนภูมิการทำงานรวมของนาฬิกาปลุกและปฏิทิน 100 ปี (ต่อ)

3.4.3 การเลือกอุปกรณ์ไฟฟ้า



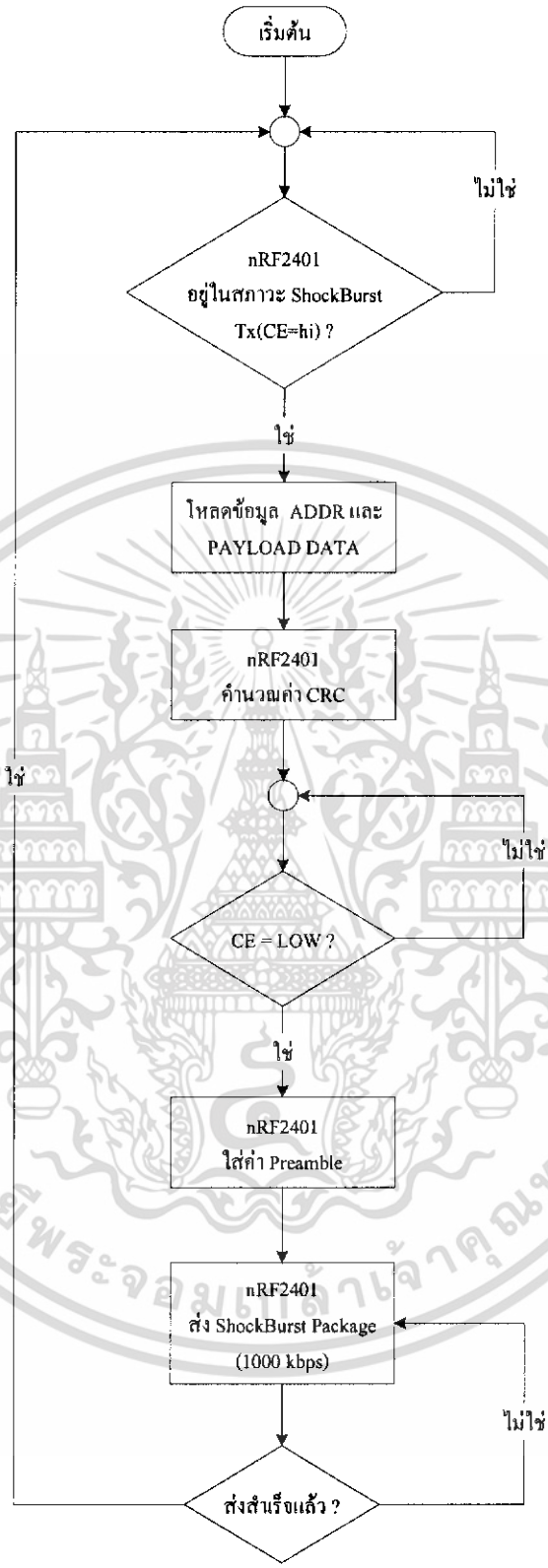
รูปที่ 3.9 แสดงแผนภูมิการทำงานของระบบในการเลือกอุปกรณ์ที่ต้องการควบคุม

3.4.4 การควบคุมอุปกรณ์ไฟฟ้า



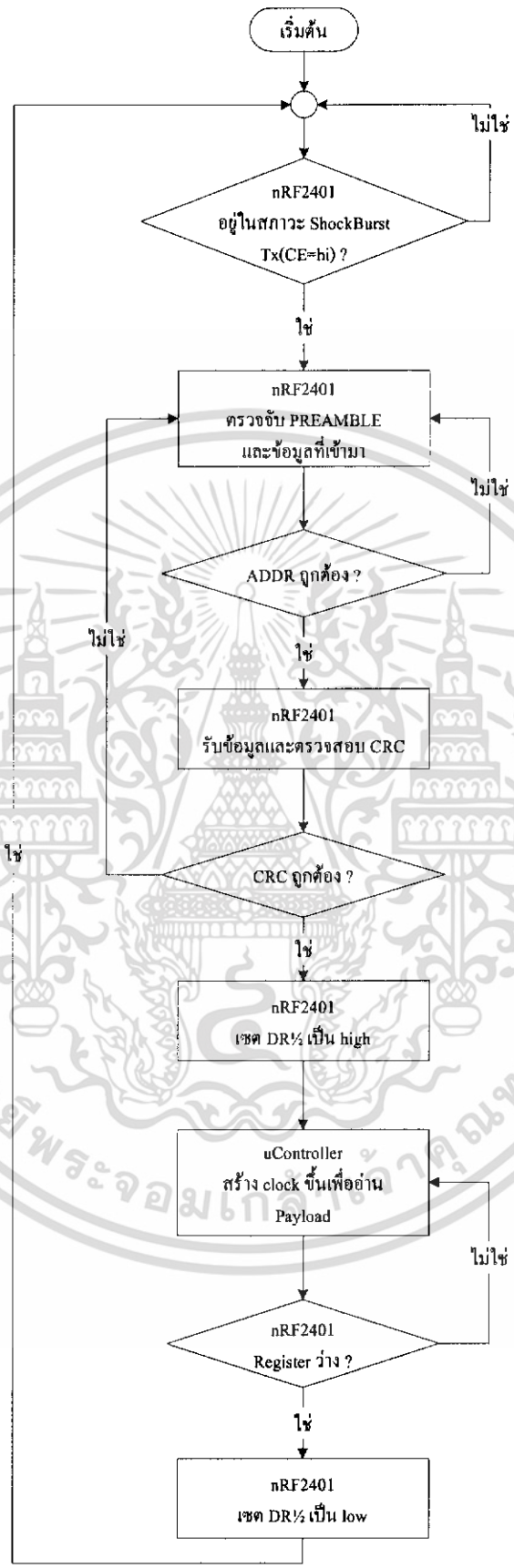
รูปที่ 3.10 แสดงแผนภูมิการทำงานของระบบในการควบคุมอุปกรณ์ไฟฟ้า

3.4.5 การส่งข้อมูลของ TRW 2.4 GHz



รูปที่ 3.11 แสดงแผนภูมิการทำงานของระบบในการส่งข้อมูลของ TRW 2.4 GHz

3.4.6 การรับข้อมูลของ TRW 2.4 GHz

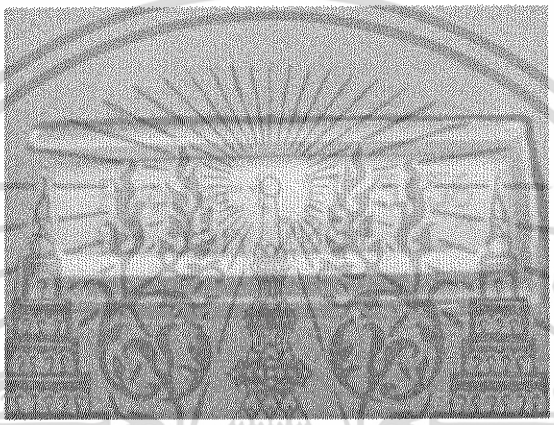


รูปที่ 3.12 แสดงแผนภูมิการทำงานของระบบในการรับข้อมูลของ TRW 2.4 GHz

บทที่ 4
การทดลองและผลการทดลอง

4.1 การทดลองที่ 1

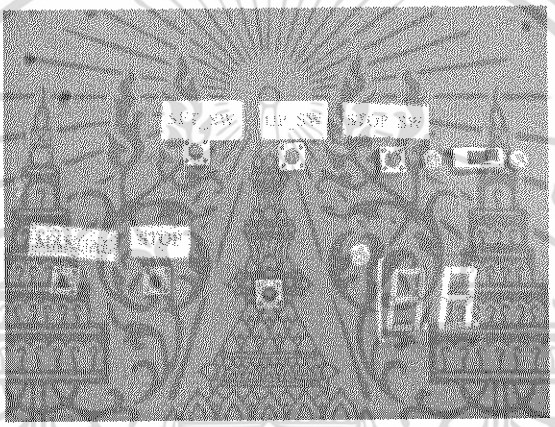
ขั้นตอนแรกเป็นการกำหนดเวลาที่นาฬิกาปลุก และเลือกอุปกรณ์ไฟฟ้าที่ต้องการควบคุมซึ่งแสดงไว้บน 7-SEGMENT โดยสามารถเลือกได้ 3 แบบ คือ “1”, “2” และ “A” จากนั้นเมื่อถึงเวลาที่กำหนดไว้จะมีสัญญาณอินพุทจากนาฬิกาปลุกเข้าที่ขา 1 ของไมโครคอนโทรลเลอร์ MCS-51(AT89C52) แสดงดังรูปที่ 4.1, รูปที่ 4.2, รูปที่ 4.3 และ รูปที่ 4.4 ดังนี้



รูปที่ 4.1 แสดงหน้าจอนาฬิกา

รูปที่ 4.2 แสดงการตั้งเวลาปลุกจากการใส่ค่าชั่วโมง

รูปที่ 4.3 แสดงการตั้งเวลาปลุกจากการใส่ค่านาที



รูปที่ 4.4 แสดงการเลือกอุปกรณ์จาก 7-SEGMENT

ทางด้านเครื่องรับจะเห็นว่าอุปกรณ์ไฟฟ้ามีการเปลี่ยนสถานะถูกต้องตามที่ได้เลือกไว้ในตอนต้น (ในที่นี้กำหนดให้สถานะเริ่มต้นของอุปกรณ์ไฟฟ้าทั้งสองตัวเป็น OFF) แต่ถ้าไม่มีสัญญาณอินพุทจากนาฬิกาปลุก อุปกรณ์ไฟฟ้าทุกตัวก็จะไม่มีการทำงานใดๆ ทั้งสิ้น แสดงผลการทดลองได้ดังตารางที่ 4.1

ตารางที่ 4.1 แสดงผลการทดลองเมื่อรับสัญญาณอินพุทจากนาฬิกาปลุก

สวิตซ์เลือกอุปกรณ์ไฟฟ้า (SW3)		สัญญาณอินพุท นาฬิกาปลุก	อุปกรณ์ไฟฟ้า	
จำนวนครั้งที่กด	7 - SEGMENT แสดงผล		ตัวที่ 1	ตัวที่ 2
1	"1"	OFF	OFF	OFF
		ON	ON	OFF
2	"2"	OFF	OFF	OFF
		ON	OFF	ON
3	"A"	OFF	OFF	OFF
		ON	ON	ON

4.2 การทดลองที่ 2

เมื่อทำการกดสวิตซ์เลือกอุปกรณ์ไฟฟ้าและไม่มีสัญญาณอินพุทจากนาฬิกาปลุก แต่ให้สัญญาณอินพุทเป็นการกดสวิตซ์ควบคุมการทำงานเข้าขาที่ 2 ของไมโครคอนโทรลเลอร์ MCS-51(AT89C52) อุปกรณ์ไฟฟ้าที่ทำการเลือกไว้ก็จะมีการเปลี่ยนสถานะเป็นตรงกันข้าม แสดงผลการทดลองได้ดังตารางที่ 4.2 และแสดงการทำงานของอุปกรณ์ได้ดังรูปที่ 4.5 และ 4.6

ตารางที่ 4.2 แสดงผลการทดลองเมื่อได้รับสัญญาณอินพุทจากสวิตซ์ควบคุมการทำงานของอุปกรณ์ไฟฟ้า

สวิตซ์เลือกอุปกรณ์ไฟฟ้า (SW3)		สวิตซ์ควบคุมการ ทำงานของอุปกรณ์ ไฟฟ้า (SW1)	อุปกรณ์ไฟฟ้า	
จำนวนครั้งที่กด	7 – SEGMENT แสดงผล		ตัวที่ 1	ตัวที่ 2
1	"1"	OFF	OFF	OFF
		ON	ON	OFF
2	"2"	OFF	OFF	OFF
		ON	OFF	ON
3	"A"	OFF	OFF	OFF
		ON	ON	ON

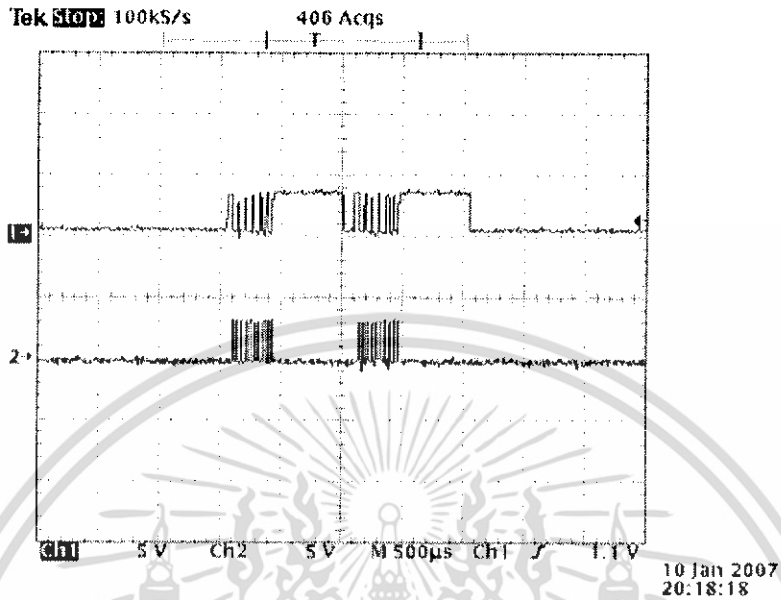
รูปที่ 4.5 แสดงสถานะของอุปกรณ์ไฟฟ้าก่อนกดสวิตช์

รูปที่ 4.6 แสดงสถานะของอุปกรณ์ไฟฟ้าหลังกดสวิตช์

4.3 รูปสัญญาณ

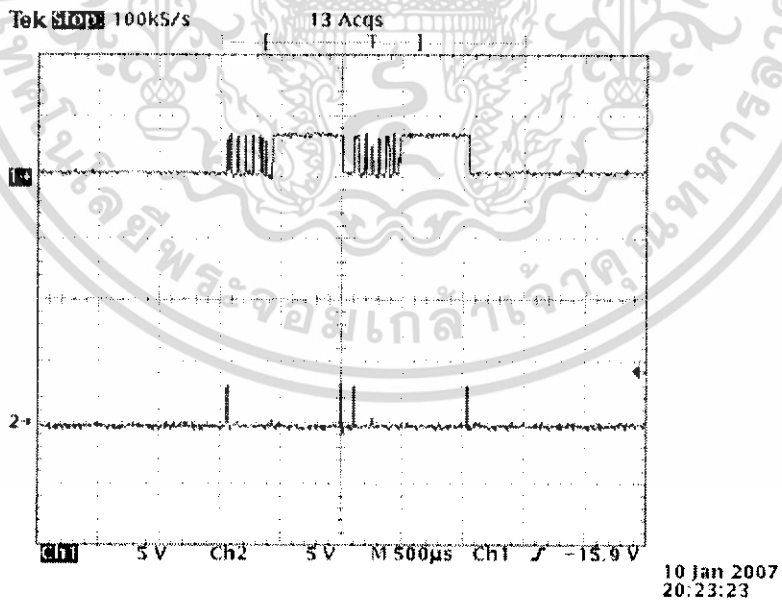
เมื่อทำการจับสัญญาณทั้งด้านรับและด้านส่ง จะได้รูปสัญญาณดังนี้

4.3.1 จับสัญญาณระหว่าง DATA เทียบกับสัญญาณ CLOCK ทางด้านส่ง



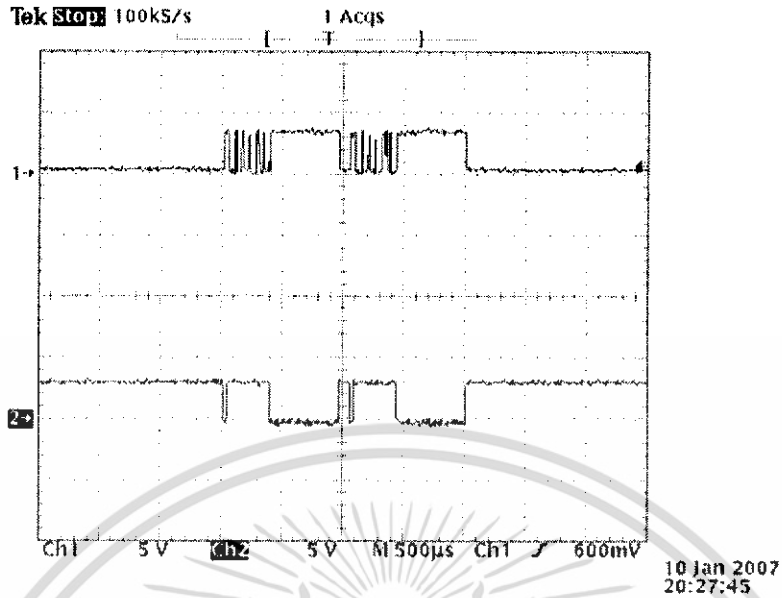
รูปที่ 4.7 แสดงสัญญาณ DATA และ CLOCK ทางด้านส่ง

4.3.2 จับสัญญาณระหว่าง DATA เทียบกับสัญญาณ CS ทางด้านส่ง



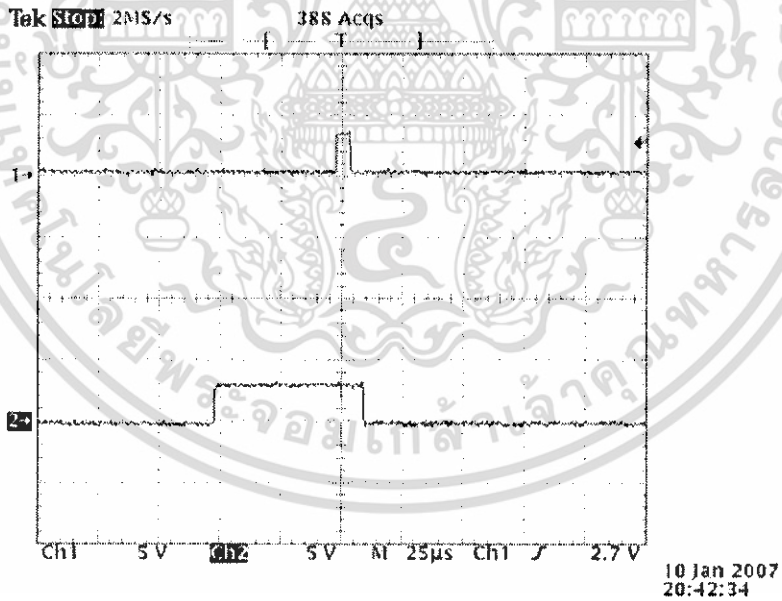
รูปที่ 4.8 แสดงสัญญาณ DATA และ CS ทางด้านส่ง

4.3.3 จับสัญญาณระหว่าง DATA เทียบกับสัญญาณ CE ทางด้านส่ง



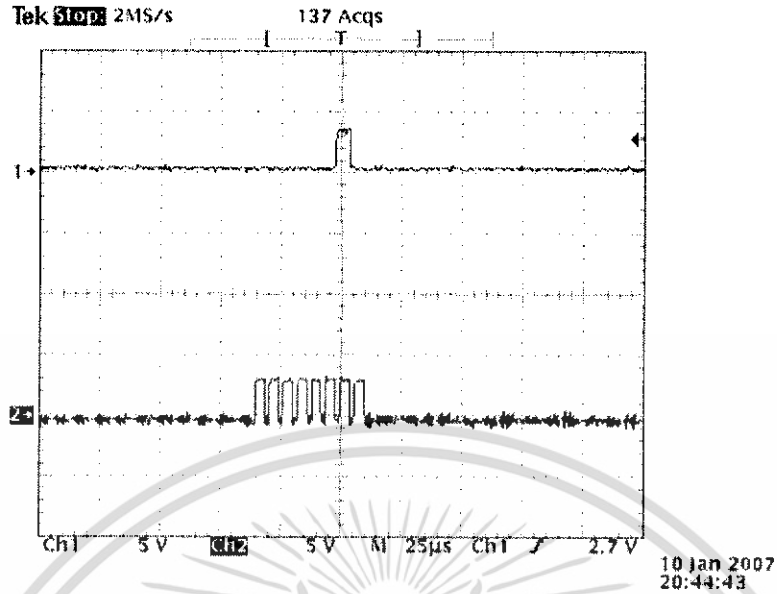
รูปที่ 4.9 แสดงสัญญาณ DATA และ CE ทางด้านส่ง

4.3.4 จับสัญญาณระหว่าง DATA เทียบกับสัญญาณ DR1 ทางด้านรับ



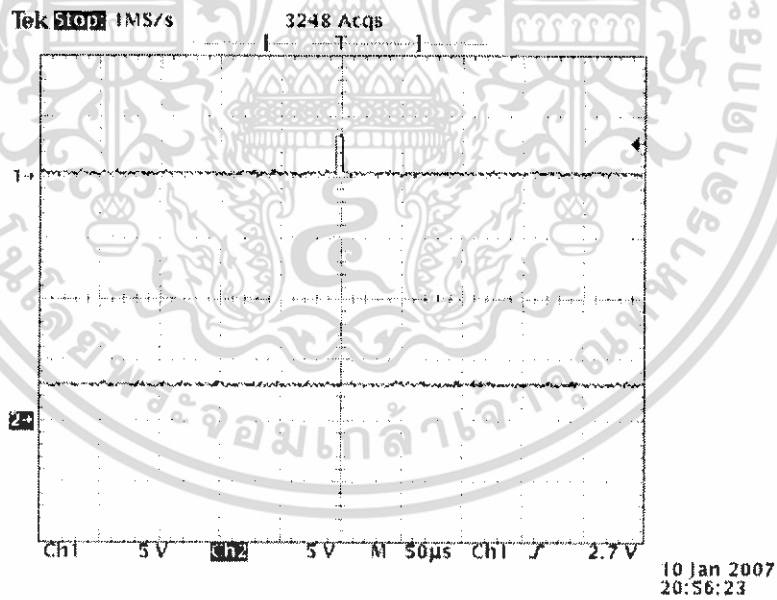
รูปที่ 4.10 แสดงสัญญาณ DATA และ DR1 ทางด้านรับ

4.3.5 จับสัญญาณระหว่าง DATA เทียบกับสัญญาณ CLOCK ทางด้านรับ



รูปที่ 4.11 แสดงสัญญาณ DATA และ CLOCK ทางด้านรับ

4.3.6 จับสัญญาณระหว่าง DATA เทียบกับสัญญาณ CE ทางด้านรับ



รูปที่ 4.12 แสดงสัญญาณ DATA และ CE ทางด้านรับ

บทที่ 5
บทวิจารณ์และบทสรุป

5.1 สรุปผลการทดลอง

จากการทดลองการควบคุมสถานะของอุปกรณ์ไฟฟ้าโดยการรับค่าอินพุทจากนาฬิกาปลุกที่สร้างขึ้นจากไมโครคอนโทรลเลอร์ (ขาที่ 1 ของไมโครคอนโทรลเลอร์ AT89C51) เข้าสู่ภาคเครื่องส่งเพื่อทำการประมวลผล หลังจากนั้นก็จะทำการส่งต่อไปยังเครื่องส่งสัญญาณแบบไร้สาย (โมดูล TRW) เพื่อที่จะส่งไปยังภาคเครื่องรับเพื่อสั่งงานแผงควบคุมอุปกรณ์ไฟฟ้า ซึ่งผลที่ได้คือสามารถสั่งงานอุปกรณ์ไฟฟ้าให้ทำงานได้จริงและถูกต้องตามความต้องการของผู้ใช้งาน โดยที่ผลการทำงานที่ได้นี้ไม่แตกต่างจากการที่ผู้ใช้งานทำการสับสวิตช์ที่อุปกรณ์ไฟฟ้าด้วยตัวเอง จึงเป็นการช่วยอำนวยความสะดวกให้แก่ผู้ใช้งาน

5.2 ปัญหาที่พบจากการทดลอง

เนื่องจากมีการใช้ไฟแรงดัน 220 โวลต์ในการทดลอง แต่ผู้ทำการทดลองขาดความระมัดระวังในการทดสอบวงจร จึงสร้างความเสียหายให้แก่วงจรอย่างมาก

5.3 วิธีแก้ไขปัญหา

ผู้ทำการทดลองต้องมีความระมัดระวังให้มากขึ้น และหมั่นทำการตรวจเช็คอุปกรณ์ที่ใช้ในการทดลองให้พร้อมกว่านี้

5.4 ข้อจำกัดของโครงงาน

1. ในการควบคุมอุปกรณ์ไฟฟ้า เมื่อผู้ใช้งานได้สั่งงานแก่วงจรภาคเครื่องส่งไปแล้ว ผู้ใช้งานมิอาจทราบได้ว่าอุปกรณ์ไฟฟ้าได้มีการเปลี่ยนแปลงสถานะหรือไม่ เนื่องจากไม่มีระบบตรวจสอบสถานะของอุปกรณ์ที่จะช่วยแจ้งสถานะที่แท้จริงให้ทราบ ผู้ทำการทดลองจำเป็นที่จะต้องตรวจสอบที่ตัวอุปกรณ์เท่านั้น
2. ระยะทางในการส่งสัญญาณแบบไร้สายต้องไม่เกิน 150 เมตร (ในที่โล่งแจ้ง) เพราะระยะทางในการส่งสัญญาณของเครื่องส่งสัญญาณแบบไร้สาย (โมดูล TRW) ถูกจำกัดด้วยระยะทางประมาณ 150 เมตรเท่านั้น

5.5 แนวทางการพัฒนาต่อไปในอนาคต

1. เพิ่มส่วนการตรวจสอบสถานะการทำงานของอุปกรณ์ไฟฟ้าที่ภาคเครื่องส่ง
2. เพิ่มจำนวนอุปกรณ์ไฟฟ้าที่จะควบคุม โดยเปลี่ยนการเชื่อมต่อไมโครคอนโทรลเลอร์ให้เป็นชนิด Multiprocessor
3. นำไปประยุกต์ใช้กับอุปกรณ์อื่นๆ เพื่อสร้างเป็นระบบเครือข่ายบ้านอัจฉริยะ

บรรณานุกรม

1. ธีรวัฒน์ ประกอบผล, รศ. **ภาษาแอสเซมบลี สำหรับ MCS - 51**. กรุงเทพฯ: สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2546.
2. บัณฑิต จามรภูติ. **คู่มือการใช้งาน Protel 99.** เชียงใหม่ : บัณฑิตเพรส, 2544.
3. สมยศ จุณณปิยะ, รศ. **การประยุกต์ใช้งาน ไมโครคอนโทรลเลอร์ตระกูล MCS-51.** คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2546.
4. อุทัย สุขสิงห์, ผศ. **ไมโครโพรเซสเซอร์ และ ไมโครคอนโทรลเลอร์ MCS - 51.** กรุงเทพฯ: สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2547.

ภาคผนวก

โปรแกรมส่วนนาฬิกาปลุก

```
;*********************************************************************
; Program          : Real Time Clock
; Description      : Real Time Clock , S1 for set time,alarm time
;*********************************************************************

;---------------------------------------------------------------------
; Define Port&Pin Name
;---------------------------------------------------------------------
ALARM               BIT   P1.0              ; Alarm bit
SCL                 BIT   P2.3              ; SCL I2C Bus
SDA                 BIT   P2.4              ; SDA I2C Bus
SET_SW              BIT   P2.0        ; For set alarm time, time
UP_SW               BIT   P2.1        ; For increase data
STOP_SW             BIT   P2.2        ; For exit set up
STOP_ALARM_SW       BIT   P2.5        ; For stop alarm
LCD_EN              BIT   P3.6        ; LCD Module Enable
LCD_RS              BIT   P3.7        ; LCD Module Register Select

;---------------------------------------------------------------------
; Define User Register
;---------------------------------------------------------------------
FLAG                EQU   02FH              ; User FLAG
I2C_ACK             BIT   FLAG.0            ; Define I2C Acknowledge
                                            ; as bit
SEC20               BIT   FLAG.1            ; Define LOOP2 as bit for
alarm twice

;---------------------------------------------------------------------
; Define User Register
;---------------------------------------------------------------------
LCD_ADDR    EQU   30H                       ; For keep LCD Address
LCD_DATA    EQU   31H                       ; For keep LCD Data
LCD_PTR     EQU   32H                       ; For keep LCD 3 Cha.r
Pointer
I2C_ADDR    EQU   33H                       ; For keep I2C Address
I2C_DATA    EQU   34H                       ; For keep I2C Data
DATA_1      EQU   35H                       ; For keep Data form SW.
LIMIT       EQU   36H                       ; For keep limit number
LCD         EQU   37H                       ; For keep Data to show LCD
ADDRESS     EQU   38H                       ; For keep address in RTC
CLR2ZERO    EQU   39H                       ; For keep number that want
to zero
PAST_SECOND EQU   40H                       ; For keep second before
SECONDS     EQU   41H                       ; For keep Seconds
MINUTES     EQU   42H                       ; For keep Minutes
HOURS       EQU   43H                       ; For keep Hours
DAY         EQU   44H                       ; For keep Day
DATE        EQU   45H                       ; For keep Date
MONTH EQU   46H                             ; For keep Month
YEAR        EQU   47H                       ; For keep Year
CONTROL     EQU   48H                       ; For keep Control Byte
MINUTES_AR  EQU   49H                       ; For keep Minutes Alarm
HOURS_AR    EQU   50H                       ; For keep Hours Alarm
```

```
;-------------------------------------------------------------------
; Define I2C Slave Address
;-------------------------------------------------------------------
RTC_ID          EQU   11010000B          ; RTC Slave Address

;-------------------------------------------------------------------
; Main Program.
;-------------------------------------------------------------------
            ORG   0000H                  ; Reset Vector
            CLR   SEC20
            MOV   P0,#00000000B          ; Clear Data bus
            SETB  SDA                    ; Clear I2C bus
            SETB  SCL                    ;
            MOV   P2,#11111111B          ; Clear status keypad
            MOV   P3,#00011111B          ; Clear status LCD, P3.2-3


            MAIN: ACALL INIT_LCD         ; Call LCD Initial sub
            MOV   LCD_ADDR,#00H          ; Set Address 00H
            ACALL SET_ADDR_LCD
            MOV   DPTR,#TITLE_1      ; Index Pointer ROM to Show LCD
            ACALL WRLINE_LCD         ; 00H-0FH(Increase automatic)

LOOP:       ACALL RTC_RD                 ; Read RTC
      JB    STOP_ALARM_SW,CHECK_TIME; If STOP_ALARM is not pressed
            CLR   SEC20                  ; will check alarm time
            JMP   CLEAR_ALARM

;-------------------------------------------------------------------
; Check for alarm time and send to voice record
;-------------------------------------------------------------------
CHECK_TIME: MOV   A,HOUR; Move HOURS to ACC for compare with HOURS_AR
            CJNE  A,HOURS_AR,ALARM_CONT
            MOV   A,MINUTES              ; Move MINUTES to ACC for
                                         ; compare with MINUTES_AR
            CJNE  A,MINUTES_AR,ALARM_CONT
            JB    SEC20,ALARM_CONT
            SETB  SEC20
            MOV   PAST_SECOND,SECONDS
            MOV   R4,#00D                ; R4 is a counter 20 sec.
            CLR   ALARM
ALARM_CONT: JNB   SEC20,CLEAR_ALARM
            MOV   A,PAST_SECOND
            CJNE  A,SECONDS,INC_R4
            JMP   CLOCK
INC_R4:           MOV   PAST_SECOND,SECONDS
            INC   R4
            MOV   A,R4
            CJNE  A,#20D,INC_R4_1
            SETB  ALARM
INC_R4_1:         CJNE  A,#21D,CLOCK
            CLR   ALARM
            MOV   R4,#00D
            JMP   CLOCK
CLEAR_ALARM:      SETB  ALARM
```

```
;---------------------------------------------------------------------
; Show time and calendar on LCD
;---------------------------------------------------------------------
CLOCK:          MOV   LCD_ADDR,#41H             ; Set Address 41H
                ACALL SET_ADDR_LCD
                MOV   LCD_DATA,DATE             ; Write Date to LCD
                ACALL BCD2LCD                   ; Write from RTC BCD Data
                MOV   LCD_DATA,#' '             ; Write space to LCD
                ACALL WRCHAR_LCD
                MOV   A,MONTH             ; Convert Month from BCD to HEX
                CJNE  A,#10H,WR_CHK_MONTH_1
                MOV   A,#0AH                    ; October => 0AH
                AJMP  WRITE_MONTH_NX
WR_CHK_MONTH_1:   CJNE  A,#011H,WR_CHK_MONTH_2
                MOV   A,#0BH                    ; November => 0BH
                AJMP  WRITE_MONTH_NX
WR_CHK_MONTH_2:   CJNE  A,#012H,WRITE_MONTH_NX ;
                MOV   A,#0CH                    ; December => 0CH
WRITE_MONTH_NX:   MOV   LCD_PTR,A               ; Set 3 Char. Pointer
                MOV   DPTR,#MONTH_JAN           ; Set Start Pointer
                ACALL WR3CHAR_LCD         ; Write 3 Char. (Month) to LCD
                MOV   LCD_DATA,#' '             ; Write space to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_DATA,#'2'             ; Write '2' to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_DATA,#'0'             ; Write '0' to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_DATA,YEAR             ; Write Year to LCD
                ACALL BCD2LCD                   ; Write from RTC BCD Data
                MOV   LCD_DATA,#' '             ; Write space to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_DATA,#'^'             ; Write '^' to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_DATA,#'o'             ; Write 'o' to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_DATA,#'^'             ; Write '^' to LCD
                ACALL WRCHAR_LCD
                MOV   LCD_ADDR,#0BH             ; Set Address 0BH
                ACALL SET_ADDR_LCD
                MOV   A,HOURS                   ; Get Hour
                ANL   A,#00110000B              ; Get x10 Digit
                JZ    WRITE_TIME_HN             ; Check x10 Digit = 0 ?
                SWAP  A                         ; If not => Write to LCD
                ADD   A,#30H                    ; Convert to ASCII
                AJMP  WRITE_TIME_HH
WRITE_TIME_HN:    MOV   A,#' '                  ; x10 = 0 then Write space
WRITE_TIME_HH:    MOV   LCD_DATA,A              ; Write to LCD
                ACALL WRCHAR_LCD
                MOV   A,HOURS                   ; Write x1 Digit to LCD
                ANL   A,#00001111B
                ADD   A,#30H                    ; Convert to ASCII
                MOV   LCD_DATA,A
                ACALL WRCHAR_LCD
                MOV   A,SECONDS           ; Check Second = Odd Number ?
                ANL   A,#01H
                JNZ   WRITE_SPACE               ; Even => Write space
                MOV   LCD_DATA,#':'             ; Odd => Write ':'
                ACALL WRCHAR_LCD
                AJMP  WRITE_MINUTES
WRITE_SPACE:      MOV   LCD_DATA,#' '
                ACALL WRCHAR_LCD
```

```
WRITE_MINUTES:    MOV   LCD_DATA,MINUTES  ; Write Minutes to LCD
            ACALL BCD2LCD                 ; Write from RTC BCD Data
            JNB   SET_SW,SET_HOURS        ; Check SET_SW Pressed?
            AJMP  LOOP                    ; Jump to loop

;---------------------------------------------------------------------
; Set alarm hours
;---------------------------------------------------------------------
SET_HOURS:  JNB   SET_SW,$                ; SET_SW is pressed ?
            MOV   DATA_1,HOURS_AR
            MOV   DPTR,#CLR_SET_HR_AR     ;Set index Pointer ROM to
Show LCD
            MOV   LIMIT,#24H              ; Limit is 24 in BCD
            MOV   CLR2ZERO,#24D           ; CLR2ZERO is a byte use
when write HOURS is 24
            MOV   ADDRESS,#08H            ; Set Slave Address 08H
            CALL  SET_TIME

;---------------------------------------------------------------------
; Set alarm minutes
;---------------------------------------------------------------------
            JNB   SET_SW,$                ; SET_SW is pressed ?
            MOV   DATA_1,MINUTES_AR
            MOV   DPTR,#CLR_SET_MIN_AR    ;Set index Pointer ROM to
Show LCD
            MOV   LIMIT,#60H              ; Limit is 60 in BCD
            MOV   CLR2ZERO,#60D           ; CLR2ZERO is a byte
            MOV   ADDRESS,#09H            ; Set Slave Address 09H
            CALL  SET_TIME

;---------------------------------------------------------------------
; Set hours
;---------------------------------------------------------------------
            JNB   SET_SW,$                ; SET_SW is pressed ?
            MOV   DATA_1,HOURS
            MOV   DPTR,#CLR_SET_HR        ;Set index Pointer ROM to
Show LCD
            MOV   LIMIT,#24H              ; Limit is 24 in BCD
            MOV   CLR2ZERO,#24D           ; CLR2ZERO is a byte use
                                          ; when write   HOURS is 24
            MOV   ADDRESS,#02H            ; Set Slave Address 02H
            CALL  SET_TIME

;---------------------------------------------------------------------
; Set minutes
;---------------------------------------------------------------------
            JNB   SET_SW,$                ; SET_SW is pressed ?
            MOV   DATA_1,MINUTES
            MOV   DPTR,#CLR_SET_MIN       ;Set index Pointer ROM to
Show LCD
            MOV   LIMIT,#60H              ; Limit is 60 in BCD
            MOV   CLR2ZERO,#60D           ; CLR2ZERO is a byte use
                                          ; when write HOURS is 60
            MOV   ADDRESS,#01H            ; Set Slave Address 01H
            CALL  SET_TIME
```

```
;-------------------------------------------------------------------
; Set seconds
;-------------------------------------------------------------------
            JNB   SET_SW,$                  ; SET_SW is pressed ?
            MOV   DATA_1,SECONDS
      MOV   DPTR,#CLR_SET_SEC ;Set index Pointer ROM to Show LCD
            MOV   LIMIT,#60H                ; Limit is 60 in BCD
            MOV   CLR2ZERO,#60D             ; CLR2ZERO is a byte use
                                            ; when write HOURS is 60
            MOV   ADDRESS,#00H              ; Set Slave Address 00H
            CALL  SET_TIME

;-------------------------------------------------------------------
; Set year
;-------------------------------------------------------------------
            JNB   SET_SW,$                  ; SET_SW is pressed ?
            MOV   DATA_1,YEAR
            MOV   DPTR,#CLR_SET_YEAR        ;Set index Pointer
            MOV   LIMIT,#99H                ; Limit is 99 in BCD
            MOV   CLR2ZERO,#100D            ; CLR2ZERO is a byte use
                                            ; when write HOURS is 100
            MOV   ADDRESS,#06H              ; Set Slave Address 06H
            CALL  SET_TIME


;-------------------------------------------------------------------
; Set month
;-------------------------------------------------------------------
            JNB   SET_SW,$                  ; SET_SW is pressed ?
            MOV   DATA_1,MONTH
            MOV   DPTR,#CLR_SET_MON         ;Set index Pointer ROM to
Show LCD
            MOV   LIMIT,#12H                ; Limit is 12 in BCD
            MOV   CLR2ZERO,#13D             ; CLR2ZERO is a byte
      MOV   ADDRESS,#05H                    ; Set Slave Address 05H
            CALL  SET_TIME
            MOV   MONTH,DATA_1

;-------------------------------------------------------------------
; Set date
;-------------------------------------------------------------------

SET_DATE:   JNB   SET_SW,$          ; SET_SW is pressed ?
            MOV   A,DATE
            CJNE  A,#28H,CLR_DATE   ; Compare date with 28 in BCD
CLR_DATE:   JC    SET_DATE_1  ; If date less than 28 carry is set
                              ; this condition is normal
            MOV   DATE,#00H   ; If date more than 28 carry is clear
                              ; will clear date
SET_DATE_1: MOV   DATA_1,DATE
            ACALL INIT_LCD    ; Call LCD Initial subroutine
            MOV   LCD_ADDR,#00H           ; Set Address 00H
            ACALL SET_ADDR_LCD
            MOV   DPTR,#CLR_SET_DATE; Index Pointer ROM to Show LCD
            ACALL WRLINE_LCD        ; 00H-0FH(Increase automatic)
UP_PRESSED_DATE:  JB    UP_SW,TEMP1       ; UP_SW is pressed ?
            JNB   UP_SW,$
            JMP   TEMP2
TEMP1:      JMP   STOP_PRESSED_DATE
TEMP2:      MOV   A,MONTH
            CJNE  A,#04H,DATE28_1
```

```
            JMP   DATE30
DATE28_1:   CJNE  A,#06H,DATE28_2
            JMP   DATE30
DATE28_2:   CJNE  A,#09H,DATE28_3
            JMP   DATE30
DATE28_3:   CJNE  A,#11H,DATE28_4

DATE30:     MOV   LIMIT,#30D
            JMP   DATE_1
DATE28_4:   PUSH  ACC
            PUSH  B
            CJNE  A,#02H,DATE31
            MOV   B,#04H;
            MOV   A,YEAR
            MOV   A,B
            CJNE  A,#00H,DATE28
            MOV   LIMIT,#29D        ; This Year has 29 days
            POP   B
            POP   ACC
            JMP   DATE_1
DATE28:           MOV   LIMIT,#28D  ; This Year has 28 days
            POP   B
            POP   ACC
            JMP   DATE_1
DATE31:           POP   B
            POP   ACC
            MOV   LIMIT,#31D        ; This Month has 31 days
DATE_1:           CALL  DECIMAL     ; Converse DATA_1 to decimal
            MOV   A,DATA_1
            CJNE  A,LIMIT,UP_PRESSED_DATE_1
            MOV   DATA_1,#00H
UP_PRESSED_DATE_1:      JC    UP_PRESSED_DATE_2
            MOV   DATA_1,#00H ; If data_1 more than LIMIT carry is
                              ; clear will clear date
UP_PRESSED_DATE_2:INC   DATA_1; Increase DATA_1 if UP_SW is pressed
            CALL  BCD         ; Converse DATA_1 to BCD
            MOV   LCD,DATA_1  ; Move DATA_1 to LCD_DATA for show LCD
            CALL  SHOW_LCD
SAVE_RTC_DATE:    MOV   I2C_ADDR,#RTC_ID  ; Set RTC as I2C Write
            LCALL I2C_SLAVE               ; Connect Slave
            MOV   I2C_DATA,#04H           ; Set Slave Address 04H
            LCALL I2C_DATA_WR             ; Write Address to Slave
            MOV   I2C_DATA,DATA_1         ; Write DATA_1 to RTC
            LCALL I2C_DATA_WR             ; Write Date to Slave
            CALL  I2C_STOP                ; Send Stop Condition
            JMP   UP_PRESSED_DATE
STOP_PRESSED_DATE:JB    STOP_SW,STOP_TEMP ; STOP_SW is pressed ?
            JMP   STOP_TEMP_1
STOP_TEMP:  JMP   UP_PRESSED_DATE
STOP_TEMP_1:MOV   LCD_ADDR,#00H; Set Address 00H
            ACALL SET_ADDR_LCD
            MOV   DPTR,#TITLE_1           ; Index Pointer ROM to Show
LCD
            ACALL WRLINE_LCD        ; 00H-0FH(Increase automatic)
            MOV   DATE,DATA_1
LOOP_TEMP:  LJMP  LOOP              ; Jump to loop


;---------------------------------------------------------------------
; Check UP_SW pressed
; I/P : DATA_1, FN_SET_ADDR, LIMIT, ADDRESS
```

```
; O/P : DATA_1 (HOURS_AR, MINUTES_AR, HOURS, MINUTES)
; Reserve : ACC,DPTR
;-------------------------------------------------------------------
SET_TIME:    CALL  INIT_LCD          ; Call LCD Initial subroutin
             MOV   LCD_ADDR,#00H     ; Set Address 00H
             CALL  SET_ADDR_LCD
             CALL  WRLINE_LCD        ; 00H-0FH(Increase automatic)
UP_PRESSED_TIME:  JB    UP_SW,STOP_PRESSED_TIME ; UP_SW is pressed ?
             JNB   UP_SW,$
             CALL  DECIMAL           ; Converse DATA_1 to decimal
             MOV   A,DATA_1
             CJNE  A,CLR2ZERO,UP_PRESSED_TIME_1 ; If data_1 less than
CLR2ZERO
             MOV   DATA_1,#00H

UP_PRESSED_TIME_1:       JC    UP_PRESSED_TIME_2
             MOV   DATA_1,#00H
UP_PRESSED_TIME_2:INC    DATA_1
             CALL  BCD         ; Converse DATA_1 to BCD
             MOV   A,DATA_1
             CJNE  A,LIMIT,CLEAR_DATA_1
             JMP   SAVE_RTC_TIME
             MOV   DATA_1,#01H  ; will clear data
SAVE_RTC_TIME:    MOV   LCD,DATA_1
                  CALL  SHOW_LCD
                  MOV   I2C_ADDR,#RTC_ID  ; Set RTC as I2C Write
                                          ; Slave
                  CALL  I2C_SLAVE         ; Connect Slave
                  MOV   I2C_DATA,ADDRESS  ; Set Slave Address
                  CALL  I2C_DATA_WR ; Write Data to Slave
                  MOV   I2C_DATA,DATA_1        ; Write DATA_1 to RTC
                  CALL  I2C_DATA_WR
                  CALL  I2C_STOP          ; Send Stop Condition

                  JMP   UP_PRESSED_TIME

STOP_PRESSED_TIME:JB    STOP_SW,SET_NEXT  ; STOP_SW is pressed
                  MOV   LCD_ADDR,#00H
                  CALL  SET_ADDR_LCD
                  MOV   DPTR,#TITLE_1; Index Pointer ROM to Show LCD
                  CALL  WRLINE_LCD  ; 00H-0FH(Increase automatic)
                  JMP   LOOP
SET_NEXT:         JB    SET_SW,UP_PRESSED_TIME  ; SET_SW is pressed ?
                  RET

;-------------------------------------------------------------------
show LCD
; I/P : LCD_DATA in BCD
; Reserve : ACC,B
;-------------------------------------------------------------------
BCD2LCD:     PUSH  ACC                     ; Push ACC. To Stack
             PUSH  B                       ; Push B to Stack
             MOV   A,LCD_DATA              ; Get input data value
             MOV   B,A                     ; Copy to B
             ANL   A,#11110000B            ; Get higher 4 bit
             SWAP  A                       ; Swap nibble
             ADD   A,#30H                  ; Convert to ASCII
             MOV   LCD_DATA,A              ; Write LCD
             ACALL WRCHAR_LCD
             MOV   A,B                     ; Restore value
             ANL   A,#00001111B            ; Get lower 4 bit
```

```
                ADD     A,#30H                  ; Convert to ASCII
                MOV     LCD_DATA,A              ; Write LCD
                ACALL   WRCHAR_LCD
                POP     B                       ; Pop B from Stack
                POP     ACC                     ; Pop ACC. from Stack
                RET                             ; Return

;-----------------------------------------------------------------------
; Show LCD
; I/P : LCD in BCD
; Reserve : ACC
;-----------------------------------------------------------------------
SHOW_LCD:       MOV     LCD_ADDR,#0EH           ; Set Address 0EH
                ACALL   SET_ADDR_LCD
                MOV     A,LCD                   ; Get LCD
                ANL     A,#11110000B            ; Get x10 Digit
                JZ      WRITE_TIME_1            ; Check x10 Digit = 0 ?
                SWAP    A                       ; If not => Write to LCD
                ADD     A,#30H                  ; Convert to ASCII
                AJMP    WRITE_TIME_2
WRITE_TIME_1:   MOV     A,#'0'                  ; x10 = 0 then Write '0'
WRITE_TIME_2:   MOV     LCD_DATA,A              ; Write to LCD
                ACALL   WRCHAR_LCD
                MOV     A,LCD                   ; Write x1 Digit to LCD
                ANL     A,#00001111B
                ADD     A,#30H                  ; Convert to ASCII
                MOV     LCD_DATA,A
                ACALL   WRCHAR_LCD
                RET

;-----------------------------------------------------------------------
; Converse to BCD
; I/P : DATA_1 in Decimal
; O/P : DATA_1 in BCD
; Reserve : ACC,B
;-----------------------------------------------------------------------
BCD:            PUSH    ACC
                PUSH    B
                MOV     A,DATA_1
                MOV     B,#10D
                DIV     AB
                SWAP    A
                ADD     A,B
                MOV     DATA_1,A
                POP     B
                POP     ACC
                RET

;-----------------------------------------------------------------------
; Converse to decimal
; I/P : DATA_1 in BCD
; O/P : DATA_1 in decimal
; Reserve : Acc,B
;-----------------------------------------------------------------------
DECIMAL:                PUSH    ACC
                PUSH    B
                MOV     A,DATA_1
                ANL     A,#11110000B
                SWAP    A
                MOV     B,#10D
                MUL     AB
```

```
            MOV   B,A
            MOV   A,DATA_1
            ANL   A,#00001111B
            ADD   A,B
            MOV   DATA_1,A
            POP   ACC
            POP   B
            RET

;-------------------------------------------------------------------
; Define LCD's command constant
;-------------------------------------------------------------------
LCD_CMD_CLR       EQU   00000001B              ; Clear
LCD_CMD_HOME      EQU   00000010B              ; Cursor home
LCD_CMD_AUTOINC   EQU   00000110B              ; Auto increment
LCD_CMD_OFF       EQU   00001000B              ; Off
LCD_CMD_ON        EQU   00001100B              ; On


;-------------------------------------------------------------------
; LCD Initialize
;-------------------------------------------------------------------
INIT_LCD:   ACALL   DELAY_100ms                ; Delay
            CLR   LCD_RS                       ; Clear LCD_RS pin
            MOV   P0,#00111000B      ; Force 8bit mode on first times
            ACALL LCD_CLK                      ; Pulse LCD clock
            ACALL DELAY_10ms                   ; Delay
            ACALL LCD_OFF                      ; Display off
            ACALL LCD_AUTOINC ; Set auto increment cursor after write
            ACALL LCD_HOME                     ; Return cursor to home
            ACALL LCD_CLR                      ; Display cleared
            ACALL LCD_ON                       ; Display on
            RET

;-------------------------------------------------------------------
; LCD Clear Display
;-------------------------------------------------------------------
LCD_CLR:    MOV   LCD_DATA,#LCD_CMD_CLR    ; Display cleared
            AJMP  WRCMD_LCD                ; Send command

;-------------------------------------------------------------------
; LCD Return Home
;-------------------------------------------------------------------
LCD_HOME:   MOV   LCD_DATA,#LCD_CMD_HOME
            AJMP  WRCMD_LCD                ; Send command

;-------------------------------------------------------------------
; LCD Return Home
;-------------------------------------------------------------------
LCD_AUTOINC:  MOV   LCD_DATA,#LCD_CMD_AUTOINC
              AJMP  WRCMD_LCD                    ; Send command

;-------------------------------------------------------------------
; LCD Display Off
;-------------------------------------------------------------------
LCD_OFF:    MOV   LCD_DATA,#LCD_CMD_OFF    ; Display off
            AJMP  WRCMD_LCD                ; Send command

;-------------------------------------------------------------------
; LCD Display On
;-------------------------------------------------------------------
```

```
LCD_ON:     MOV   LCD_DATA,#LCD_CMD_ON    ; Display on
            AJMP  WRCMD_LCD               ; Send command

;---------------------------------------------------------------------
; Set LCD Address
; I/P : LCD_ADDR
;---------------------------------------------------------------------
SET_ADDR_LCD:     CLR     LCD_RS          ; Clear LCD_RS pin
            MOV     P0,LCD_ADDR           ; Move LCD_ADDR to P0
            ORL   P0,#10000000B           ; Set MSB of P0
            ACALL   LCD_CLK               ; Pulse LCD clock
            RET                           ; Return

;---------------------------------------------------------------------
; LCD Clear Display
; I/P : LCD_DATA
;---------------------------------------------------------------------
WRCMD_LCD:  CLR   LCD_RS                        ; Clear LCD_RS Pin
            MOV   P0,LCD_DATA             ; Write command to LCD
            ACALL LCD_CLK                       ; Pulse LCD Clock
            RET                           ; Return

;---------------------------------------------------------------------
; Write Character to show LCD
; I/P : LCD_DATA
;---------------------------------------------------------------------
WRCHAR_LCD: SETB  LCD_RS                  ; Set LCD_RS Pin
            MOV   P0,LCD_DATA             ; Move LCD_DATA to DATABUS
            ACALL LCD_CLK                 ; Pulse LCD Clock
            RET                           ; Return

;---------------------------------------------------------------------
; Write Line of 16 Character from ROM
; I/P : DPTR : Locate ROM Address
; Reserve : R0,ACC
;---------------------------------------------------------------------
WRLINE_LCD: MOV   R0,#0                   ; Clear loop counter
WRLINE_LCD_1:     SETB  LCD_RS            ; Set LCD_RS Pin
            CLR   A                       ; Clear ACC.
            MOVC  A,@A+DPTR         ; Move data from @DPTR to ACC.
            MOV   P0,A                    ; Move ACC. to DATABUS
            ACALL LCD_CLK                 ; Pulse LCD Clock
            INC   DPTR                    ; Increase Pointer
            INC   R0                      ; Increase loop counter
            CJNE  R0,#16D,WRLINE_LCD_1    ; Do until 16 times
            RET

;---------------------------------------------------------------------
; LCD Clk
;---------------------------------------------------------------------
LCD_CLK:          SETB  LCD_EN            ; Pulse Clock to LCD_EN
            ACALL LCD_DELAY
            CLR   LCD_EN
            ACALL LCD_DELAY
            RET

;---------------------------------------------------------------------
; Write 3 Character from ROM
; I/P : LCD_PTR
; Reserve : R0,ACC,DPTR,B
;---------------------------------------------------------------------
```

```
WR3CHAR_LCD:      MOV   R0,#0              ; Clear Loop Counter
            MOV   A,LCD_PTR              ; Get LCD Pointer
            DEC   A                      ; Decrease Value
            MOV   B,#3
            MUL   AB                     ; Multiply by 3
WR3CHAR_LCD_1:    SETB  LCD_RS           ; Set LCD_RS Pin
            MOVC  A,@A+DPTR          ; Get Data from ROM with Pointer
            MOV   P0,A                   ; Move ACC. to DATABUS
            ACALL LCD_CLK                ; Pulse LCD Clock
            INC   DPTR                   ; Increase Pointer
            INC   R0                     ; Increase Loop Counter
            MOV   A,LCD_PTR              ; Restore LCD Pointer
            DEC   A                      ; Decrease Value
            MOV   B,#03D
            MUL   AB                     ; Multiply by 3
            CJNE  R0,#3,WR3CHAR_LCD_1    ; Do until 3 times
            LCALL LCD_ON                 ; Show LCD
            RET                          ; Return

;-------------------------------------------------------------------
; I2C RTC Read
; O/P :
SECONDS,MINUTES,HOURS,DAY,DATE,MONTH,YEAR,CONTROL,HOURS_AR,MINUTES_AR
;-------------------------------------------------------------------
RTC_RD:     MOV   I2C_ADDR,#RTC_ID  ; Set RTC as I2C Write Slave
            LCALL I2C_SLAVE              ; Connect Slave
            MOV   I2C_DATA,#00H          ; Set Slave Address 00H
            LCALL I2C_DATA_WR            ; Write Data to Slave
            MOV   I2C_ADDR,#RTC_ID+1     ; Set RTC as I2C Read Slave
            LCALL I2C_SLAVE              ; Connect Slave
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   SECONDS,I2C_DATA       ; Read Data to SECONDS
            LCALL I2C_ACK_BIT            ; Send Acknowledge
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   MINUTES,I2C_DATA       ; Read Data to MINUTES
            LCALL I2C_ACK_BIT            ; Send Acknowledge
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   HOURS,I2C_DATA         ; Read Data to HOURS
            LCALL I2C_ACK_BIT            ; Send Acknowledge
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   DAY,I2C_DATA           ; Read Data to DAY
            LCALL I2C_ACK_BIT            ; Send Acknowledge
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   DATE,I2C_DATA          ; Read Data to DATE
            LCALL I2C_ACK_BIT            ; Send Acknowledge
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   MONTH,I2C_DATA         ; Read Data to MONTH
            LCALL I2C_ACK_BIT            ; Send Acknowledge
            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   YEAR,I2C_DATA          ; Read Data to YEAR
            LCALL I2C_ACK_BIT            ; Read Data from Slave
            MOV   CONTROL,I2C_DATA       ; Read Data to CONTROL
            LCALL I2C_ACK_BIT            ; Send Not Acknowledge

            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   HOURS_AR,I2C_DATA      ; Read Data to MINUTES_AR
            LCALL I2C_ACK_BIT            ; Send Acknowledge

            LCALL I2C_DATA_RD            ; Read Data from Slave
            MOV   MINUTES_AR,I2C_DATA    ; Read Data to MINUTES_AR
            LCALL I2C_NACK_BIT           ; Send Acknowledge
```

```
            LCALL I2C_STOP                  ; Send Stop Condition
            RET                             ; Return

;-------------------------------------------------------------------
; I2C Data Write
; I/P : I2C_DATA
; Reserve : ACC,R5
;-------------------------------------------------------------------
I2C_DATA_WR:      PUSH  ACC                 ; Push ACC.
            SETB  I2C_ACK                   ; Set ACK. bit
            MOV   A,I2C_DATA                ; Get Data
            MOV   R5,#08D                   ; Set loop 8 times
I2C_DATA_WR_1:    RLC   A             ; Rotate ACC. to Left with Carry
            MOV   SDA,C                     ; Move Carry Flag to SDA
            ACALL I2C_CLK                   ; Pulse I2C Clock
            DJNZ  R5,I2C_DATA_WR_1          ; Do until 8 times
            SETB  SDA                       ; Set SDA
            ACALL I2C_DELAY                 ; Delay
            SETB  SCL                       ; Set SCL
            ACALL I2C_DELAY                 ; Delay
            JB    SDA,I2C_DATA_WR_2 ; Check Acknowledge from Slave
            CLR   I2C_ACK                   ; Clear ACK. bit
I2C_DATA_WR_2:    CLR   SCL                 ; Clear SCL
            POP   ACC                       ; Pop ACC.
            RET                             ; Return

;-------------------------------------------------------------------
; I2C Data Read
; O/P : I2C_DATA
; Reserve : ACC,R5
;-------------------------------------------------------------------
I2C_DATA_RD:      PUSH  ACC                 ; Push ACC.
            CLR   A                         ; Clear ACC.
            MOV   R5,#08D                   ; Set loop 8 times
I2C_DATA_RD_1:    ACALL I2C_DELAY           ; Delay
            SETB  SCL                       ; Set SCL
            ACALL I2C_DELAY                 ; Delay
            MOV   C,SDA                     ; Get SDA to Carry Flag
            RLC   A                     ; Rotate ACC. to Left with Carry
            CLR   SCL                       ; Clear SCL
            DJNZ  R5,I2C_DATA_RD_1          ; Do until 8 times
            MOV   I2C_DATA,A                ; Move Data to I2C_DATA
            POP   ACC                       ; Pop ACC.
            RET                             ; Return

;-------------------------------------------------------------------
; I2C Slave Connect
; I/P : I2C_ADDR
; O/P : I2C_ACK
; Reserve : ACC,R5
;-------------------------------------------------------------------
I2C_SLAVE:        PUSH  ACC                 ; Push ACC.
            SETB  I2C_ACK                   ; Set ACK. bit
            MOV   A,I2C_ADDR                ; Get Slave Address
            ACALL I2C_START                 ; Send Start Condition
            MOV   R5,#08D                   ; Set loop 8 times
I2C_SLAVE_1:      RLC   A             ; Rotate ACC. to Left with Carry
            MOV   SDA,C                     ; Move Carry Flag to SDA
            ACALL I2C_CLK                   ; Pulse I2C Clock
            DJNZ  R5,I2C_SLAVE_1            ; Do until 8 times
            SETB  SDA                       ; Set SDA
```

```
            ACALL I2C_DELAY                 ; Delay
            SETB  SCL                       ; Set SCL
            ACALL I2C_DELAY                 ; Delay
            JB    SDA,I2C_SLAVE_2   ; Check Acknowledge from Slave
            CLR   I2C_ACK                   ; Clear ACK.
I2C_SLAVE_2:      CLR   SCL                 ; Clear SCL
            POP   ACC                       ; Pop ACC.
            RET                             ; Return
```

```
;-----------------------------------------------------------------
; I2C Start Condition
;-----------------------------------------------------------------
I2C_START:  JNB   SCL,I2C_START_1           ; Check current SCL set?
            CLR   SCL                       ; Clear SCL
I2C_START_1:SETB  SDA                       ; Set SDA
            SETB  SCL                       ; Set SCL
            ACALL I2C_DELAY                 ; Delay
            CLR   SDA                       ; Clear SDA during SCL set
            ACALL I2C_DELAY                 ; Delay
            CLR   SCL                       ; Clear SCL
            RET                             ; Return

;-----------------------------------------------------------------
; I2C Stop Condition
;-----------------------------------------------------------------
I2C_STOP:   JNB   SCL,I2C_STOP_1            ; Check current SCL set?
            CLR   SCL                       ; Clear SCL
I2C_STOP_1: CLR   SDA                       ; Clear SDA
            ACALL I2C_DELAY                     ; Delay
            SETB  SCL                       ; Set SCL
            ACALL I2C_DELAY                     ; Delay
            SETB  SDA                       ; Set SDA during SCL set
            RET                             ; Return

;-----------------------------------------------------------------
; I2C Clock
;-----------------------------------------------------------------
I2C_CLK:    ACALL I2C_DELAY                 ; Pulse SCL
            SETB  SCL
            ACALL I2C_DELAY
            CLR   SCL
            RET                             ; Return

;-----------------------------------------------------------------
; I2C Acknowledge
;-----------------------------------------------------------------
I2C_ACK_BIT:      CLR   SDA                 ; Clear SDA
            ACALL I2C_DELAY                 ; Delay
            ACALL I2C_CLK                   ; Pulse I2C Clock
            SETB  SDA
            RET                             ; Return

;-----------------------------------------------------------------
```

```
; I2C Not Acknowledge
;---------------------------------------------------------------------
I2C_NACK_BIT:     SETB  SDA                ; Set SDA
            ACALL I2C_DELAY                ; Delay
            ACALL I2C_CLK                  ; Pulse I2C Clock
            SETB  SC
            RET                            ; Return

;---------------------------------------------------------------------
; Delay time I2C_DELAY, LCD_DELAY, 10m, 100m, 1s
; Reserve : R2,R3,R6,R7
;---------------------------------------------------------------------
I2C_DELAY:  MOV   R6,#0CH                  ; Each loop = 50 us
I2C_DELAY_1:NOP
            NOP
            DJNZ  R6,I2C_DELAY_1
            RET
LCD_DELAY:  MOV   R7,#02D                  ; Do 2 times
LCD_DELAY_1:MOV   R6,#0E6H                 ; Each loop = 1 ms
LCD_DELAY_2:NOP
            NOP
            DJNZ  R6,LCD_DELAY_2
            DJNZ  R7,LCD_DELAY_1
            RET
DELAY_10ms: MOV   R3,#05D                  ; Do 5 times
DELAY_10ms_1:     ACALL LCD_DELAY
            DJNZ  R3,DELAY_10ms_1
            RET
DELAY_100ms:      MOV   R2,#10D            ; Do 10 times
DELAY_100ms_1:    ACALL DELAY_10ms
            DJNZ  R2,DELAY_100ms_1
            RET

;---------------------------------------------------------------------
;Define Constant < Store in Flash EEPROM Program Memory >
;---------------------------------------------------------------------
TITLE_1:          DB    ' Project II       '
CLR_SET_DATE:     DB    'Set Date :        '
CLR_SET_MON:      DB    'Set Month :       '
CLR_SET_YEAR:     DB    'Set Year :        '
CLR_SET_HR:       DB    'Set Hour :        '
CLR_SET_MIN:      DB    'Set Minute :      '
CLR_SET_SEC:      DB    'Set Second :      '
CLR_SET_HR_AR:    DB    'Alarm Hour :      '
CLR_SET_MIN_AR:   DB    'Alarm Minute :    '
MONTH_JAN:        DB    'Jan'
MONTH_FEB:        DB    'Feb'
MONTH_MAR:        DB    'Mar'
MONTH_APR:  DB    'Apr'
MONTH_MAY:  DB    'May'
MONTH_JUN:  DB    'Jun'
MONTH_JUL:  DB    'Jul'
MONTH_AUG:  DB    'Aug'
MONTH_SEP:  DB    'Sep'
MONTH_OCT:  DB    'Oct'
MONTH_NOV:  DB    'Nov'
MONTH_DEC:  DB    'Dec'
DAY_MON:          DB    'MON'
DAY_TUE:          DB    'TUE'
DAY_WED:          DB    'WED'
DAY_THU:          DB    'THU'
```

```
DAY_FRI:            DB     'FRI'
DAY_SAT:            DB     'SAT'
DAY_SUN:            DB     'SUN'

END
```

โปรแกรมภาคเครื่องส่ง

```
SW_CTR      BIT P1.3
SW_CLK      BIT P1.0
SW_DEVICE   BIT P1.1
SW_CLR      BIT P1.2
DEV_1       BIT P2.0
DEV_2       BIT P2.1
BUFFER      EQU 030H
BUFFER_1    EQU 031H
BUFFER_2    EQU 032H
OUT_SEG     EQU P0

      ORG   0000H
      MOV   BUFFER ,#00H
      SETB  SW_CTR
      SETB  SW_CLK
      SETB  SW_DEVICE
      SETB  SW_CLR

;-------------------------------------------------------------------
INIT: MOV   SCON,#50H
      MOV   TMOD,#20H
      MOV   PCON,#00H
      MOV   TH1,#0FDH
      SETB  TR1

;-------------------------------------------------------------------
      MOV   OUT_SEG,#10111111B
      CLR   DEV_1
      CLR   DEV_2

;-------------------------------------------------------------------
MAIN: CJNE  R0,#04H,NAT
      MOV   BUFFER,#00H
      JMP   BOAT

NAT:  JB    SW_CTR,CHK_INPUT
      CALL  DELAY_100ms
      JB    SW_CTR,CHK_INPUT

BOAT: MOV   R0,BUFFER
      INC   R0
      JMP   SEG

CHK_INPUT:  JNB   SW_CLK,LED_ON_1
            JNB   SW_DEVICE,LED_ON_1
            JNB   SW_CLR,LED_OFF
            JMP    MAIN
```

```
;-------------------------------------------------------------------------
SEG:    CJNE   R0,#01H,SEG1
        MOV    BUFFER,R0
        MOV    OUT_SEG,#10000110B
        MOV    BUFFER_1,#01H
        JNB    SW_CTR,$
        JMP    MAIN

SEG1:   CJNE   R0,#02H,SEG2
        MOV    BUFFER,R0
        MOV    OUT_SEG,#11011011B
        MOV    BUFFER_1,#02H
        JNB    SW_CTR,$
        JMP     MAIN

SEG2:   CJNE   R0,#03H,MAIN
        MOV    BUFFER,R0
        MOV    OUT_SEG,#11110111B
        MOV    BUFFER_1,#03H
        MOV    BUFFER,#03H
        JNB    SW_CTR,$
        JMP    MAIN

;-------------------------------------------------------------------------
LED_ON_1:     MOV    R1,BUFFER
              CJNE   R1,#01H,LED_ON_2
              SETB   DEV_1
              MOV    BUFFER_2,#01H
              CALL   TX_D
              JMP    MAIN

LED_ON_2:     CJNE   R1,#02H,LED_ON_ALL
              SETB   DEV_2
              MOV    BUFFER_2,#02H
              CALL   TX_D
              JMP    MAIN

LED_ON_ALL:   CJNE   R1,#03H,MAIN
              SETB   DEV_1
              SETB   DEV_2
              MOV    BUFFER_2,#03H
              CALL   TX_D
              JMP    MAIN

LED_OFF:      CLR    DEV_1
              CLR    DEV_2
              MOV    BUFFER_2,#0AAH
              CALL   TX_D

              JMP  MAIN
```

```
;-------------------------------------------------------------------------
TX_D:       MOV   A,BUFFER_1
            MOV   SBUF,A
            JNB   TI,$
            CLR   TI
            NOP
            NOP
            NOP
            MOV   A,BUFFER_2
            MOV   SBUF,A
            JNB   TI,$
            CLR   TI
            RET

;-------------------------------------------------------------------------
DELAY_1ms:  MOV   R6,#0E6H                          ; Each loop = 1 ms
DELAY_1ms_1: NOP
            NOP
            DJNZ  R6,DELAY_1ms_1
            RET

DELAY_100ms:        MOV   R7,#10                    ; Do 100 times
DELAY_100ms_1:      MOV   R6,#0E6H                  ; Each loop = 1 ms
DELAY_100ms_2:      NOP
                    NOP
                    DJNZ  R6,DELAY_100ms_2
                    DJNZ  R7,DELAY_100ms_1
                    RET

DELAY_1s:           MOV   R5,#10
DELAY_1s_1:         MOV   R6,#0E6H
                    DJNZ  R6,DELAY_1s_1
                    DJNZ  R7,DELAY_1s
                    RET

END
```

โปรแกรมภาคเครื่องรับ

```
BUFFER_1      EQU    030H
BUFFER_2      EQU    031H
DEV_1         BIT    P2.0
DEV_2         BIT    P2.1

        ORG    0000H
        CLR    DEV_1
        CLR    DEV_2

INIT:   MOV    SCON,#50H
        MOV    TMOD,#20H
        MOV    PCON,#00H
        MOV    TH1,#0FDH
        SETB   TR1

;-------------------------------------------------------------------------
MAIN:   JNB    RI,$
        CLR    RI
        MOV    A,SBUF
        MOV    BUFFER_1,A
        JNB    RI,$
        CLR    RI
        MOV    A,SBUF
        MOV    BUFFER_2,A
        MOV    R0,BUFFER_1
        MOV    R1,BUFFER_2

CHK_DEV_CLK:        CJNE   R0,#01H,CHK_DEV_2
                    CJNE   R1,#01H,CHK_DEV_OFF
                    SETB   DEV_1
                    JMP    MAIN

CHK_DEV_2:          CJNE   R0,#02H,CHK_DEV_ALL
                    CJNE   R1,#02H,CHK_DEV_OFF
                    SETB   DEV_2
                    JMP    MAIN

CHK_DEV_ALL:        CJNE   R0,#03H,CHK_DEV_OFF
                    CJNE   R1,#03H,CHK_DEV_OFF
                    SETB   DEV_1
                    SETB   DEV_2
                    JMP    MAIN

CHK_DEV_OFF:        CJNE   R1,#0AAH,MAIN
                    CLR    DEV_1
                    CLR    DEV_2
                    JMP    MAIN

END
```

โปรแกรมส่วนโมดูลไร้สาย TRW (ภาคส่ง)

```
CE          BIT P1.2
CS          BIT P1.3
DAT         BIT P1.5
CLK         BIT P1.4
DR1         BIT P1.6


            ORG 0000H
;-------------------------------------------------------------------------
            MOV   TMOD,#021H                    ;resolve
            MOV   TH1,#0FDH
            MOV   TL1,#0FDH
            MOV   SCON,#050H
            SETB  TR1

;-------------------------------------------------------------------------
INIT:
            CLR   CE
            CLR   CS
            CLR   DAT
            CLR   CLK


;-------------------------------------------------------------------------
SETMODE_TRW24:
            CLR   CE
            SETB  CS
            CLR   A
            MOV   R1,#18
SETMODE_0:  MOV   DPTR,#CONFIG_TEST
            PUSH  ACC
            MOVC  A,@A+DPTR
            CALL  WRITE_TRW24
            POP   ACC
            INC   A
            DJNZ  R1,SETMODE_0
            SETB  DAT
            SETB  DR1
            SETB  CE
            CLR   CS

;-------------------------------------------------------------------------
MAIN_RX:    JNB   RI,$
            CLR   RI
            MOV   A,SBUF
            CALL  SEND_TRW
            JMP   MAIN_RX
```

```
;-------------------------------------------------------------------
SEND_TRW:    CALL  DELAY_1ms                          ;resolve
             CLR   CS
             SETB  CE
             PUSH  ACC
             CLR   A
             MOV   R1,#5
SEND_TRW_0:  MOV   DPTR,#CONFIG_ADDR1
             PUSH  ACC
             MOVC  A,@A+DPTR
             CALL  WRITE_TRW24
             POP   ACC
             INC   A
             DJNZ  R1,SEND_TRW_0
             POP   ACC
             CALL  WRITE_TRW24
             CLR   CLK
             CLR   CE
             CLR   DAT
             RET

;-------------------------------------------------------------------
             CLK_TRW:    CLR   CLK
             CALL  DELAY_1ms
             SETB  CLK
             CALL  DELAY_1ms
             RET

;-------------------------------------------------------------------
WRITE_TRW24:         MOV   R0,#8
WRITE_TRW24_0:       JB    ACC.7,WRITE1
                     CLR   DAT
                     JMP   WRITE_TRW2
WRITE1:              SETB  DAT
WRITE_TRW2:          CALL  CLK_TRW
                     RL    A
                     DJNZ  R0,WRITE_TRW24_0
                     RET

;-------------------------------------------------------------------
READ_TRW24:          CLR   A
                     MOV   R0,#8
READ_TRW24_0:        RL    A
                     SETB  CLK
                     CALL  DELAY_1ms
                     JB    DAT,READ_1
                     CLR   ACC.0
                     JMP   READ_TRW24_1
READ_1:              SETB  ACC.0
READ_TRW24_1:        CLR   CLK
                     CALL  DELAY_1ms
                     DJNZ  R0,READ_TRW24_0
                     RET
```

```
;*****************************************************************************
; Dummy Delay time 1ms
;*****************************************************************************

DELAY_1ms:    MOV    R6,#0E6H                          ; Each loop = 1 ms
DELAY_1ms_1:  NOP
              NOP
              DJNZ   R6,DELAY_1ms_1
              RET

DELAY_100ms:          MOV    R7,#100                          ; Do 100 times
DELAY_100ms_1:        MOV    R6,#0E6H                  ; Each loop = 1 ms
DELAY_100ms_2:        NOP
                      NOP
                      DJNZ   R6,DELAY_100ms_2
                      DJNZ   R7,DELAY_100ms_1
                      RET

;-----------------------------------------------------------------------------
CONFIG_TEST:          DB 8EH,08H,1CH
CONFIG_LEN2:          DB 08H
CONFIG_LEN1:          DB 08H
CONFIG_ADDR2:         DB 0C0H,0AAH,55H,0AAH,55H
CONFIG_ADDR1:         DB 0AAH,55H,0AAH,55H,0AAH
CONFIG_NUMADDR:       DB 0A3H
CONFIG_RF:            DB 6FH
CONFIG_CH:            DB 0AH ;TX
;CONFIG_CH:           DB 0BH ;RX

END
```

โปรแกรมส่วนโมดูลไร้สาย TRW (ภาครับ)

```
CE          BIT P1.2
CS          BIT P1.3
DAT         BIT P1.5
CLK         BIT P1.4
DR1         BIT P1.6

            ORG 0000H

;-------------------------------------------------------------------------
            MOV   TMOD,#020H                              ;resolve
            MOV   TH1,#0FDH
            MOV   TL1,#0FDH
            MOV   SCON,#050H
            SETB  TR1

;-------------------------------------------------------------------------
INIT:
            CLR   CE
            CLR   CS
            CLR   DAT
            CLR   CLK

;-------------------------------------------------------------------------
SETMODE_TRW24:
            CLR   CE
            SETB  CS
            CLR   A
            MOV   R1,#18
SETMODE_0:  MOV   DPTR,#CONFIG_TEST
            PUSH  ACC
            MOVC  A,@A+DPTR
            CALL  WRITE_TRW24
            POP   ACC
            INC   A
            DJNZ  R1,SETMODE_0
            SETB  DAT
            SETB  DR1
            SETB  CE
            CLR   CS

;-------------------------------------------------------------------------
MAIN_RX:    JNB   RI,$
            CLR   RI
            MOV   A,SBUF
            CALL  SEND_TRW
            JMP   MAIN_RX

;-------------------------------------------------------------------------
SEND_TRW:   CALL  DELAY_1ms  ;resolve
            CLR   CS
            SETB  CE
            PUSH  ACC
            CLR   A
            MOV   R1,#5
```

```
SEND_TRW_0:  MOV   DPTR,#CONFIG_ADDR1
             PUSH  ACC
             MOVC  A,@A+DPTR
             CALL  WRITE_TRW24
             POP   ACC
             INC   A
             DJNZ  R1,SEND_TRW_0
             POP   ACC
             CALL  WRITE_TRW24
             CLR   CLK
             CLR   CE
             CLR   DAT
             RET

;-------------------------------------------------------------------------
CLK_TRW:     CLR   CLK
             CALL  DELAY_1ms
             SETB  CLK
             CALL  DELAY_1ms
             RET

;-------------------------------------------------------------------------
WRITE_TRW24:        MOV   R0,#8
WRITE_TRW24_0:      JB    ACC.7,WRITE1
                    CLR   DAT
                    JMP   WRITE_TRW2
WRITE1:             SETB  DAT
WRITE_TRW2:         CALL  CLK_TRW
                    RL    A
                    DJNZ  R0,WRITE_TRW24_0
                    RET

;-------------------------------------------------------------------------
READ_TRW24:         CLR   A
                    MOV   R0,#8
READ_TRW24_0:       RL    A
                    SETB  CLK
                    CALL  DELAY_1ms
                    JB    DAT,READ_1
                    CLR   ACC.0
                    JMP   READ_TRW24_1
READ_1:             SETB  ACC.0
READ_TRW24_1:       CLR   CLK
                    CALL  DELAY_1ms
                    DJNZ  R0,READ_TRW24_0
                    RET
```

```
;***********************************************************************
; Dummy Delay time 1ms
;***********************************************************************

DELAY_1ms:          MOV   R6,#0E6H                    ; Each loop = 1 ms
DELAY_1ms_1:        NOP
                    NOP
                    DJNZ  R6,DELAY_1ms_1
                    RET

DELAY_100ms:        MOV   R7,#100                     ; Do 100 times
DELAY_100ms_1:      MOV   R6,#0E6H                    ; Each loop = 1 ms
DELAY_100ms_2:      NOP
                    NOP
                    DJNZ  R6,DELAY_100ms_2
                    DJNZ  R7,DELAY_100ms_1
                    RET

;-----------------------------------------------------------------------
CONFIG_TEST:        DB 8EH,08H,1CH
CONFIG_LEN2:        DB 08H
CONFIG_LEN1:        DB 08H
CONFIG_ADDR2:       DB 0C0H,0AAH,55H,0AAH,55H
CONFIG_ADDR1:       DB 0AAH,55H,0AAH,55H,0AAH
CONFIG_NUMADDR:     DB 0A3H
CONFIG_RF:          DB 6FH
CONFIG_CH:          DB 0AH ;TX
;CONFIG_CH:         DB 0BH ;RX

END
```