จำลองระบบการขนย้ายวัตถุ ควบคุมโดยคอมพิวเตอร์และรีโมตไร้สาย

TWO-MODE CRANE SIMULATOR

โดย

นาย กฤษฎา จิรวงศ์วัฒนะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

จำลองระบบการขนย้ายวัตถุ
ควบคุมโดยคอมพิวเตอร์และรีโมตไร้สาย
TWO-MODE CRANE SIMULATOR

โดย
นาย กฤษฎา จิรวงศ์วัฒนะ

อาจารย์ที่ปรึกษา
ผศ. พลศาสตร์ เลิศประเสริฐ

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549

จำลองระบบการขนย้ายวัตถุควบคุมโดยคอมพิวเตอร์และรีโมตไร้สาย

TWO-MODE CRANE SIMULATOR

นาย กฤษฎา จิรวงศ์วัฒนะ รหัส 46010015

โครงงานนี้ได้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....

(ผศ. พลศาสตร์ เลิศประเสริฐ)

อาจารย์ที่ปรึกษา

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาอิเล็กทรอนิกส์ สาขาวิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง จำลองระบบการขนย้ายวัตถุควบคุมโดยคอมพิวเตอร์และรีโมตไร้สาย

ผู้จัดทำ
นาย กฤษฎา จิรวงศ์วัฒนะ

.....

(ผศ. พลศาสตร์ เลิศประเสริฐ)
อาจารย์ที่ปรึกษา

จำลองระบบการขนย้ายวัตถุควบคุมโดยคอมพิวเตอร์และรีโมตไร้สาย

นาย กฤษฎา จิรวงศ์วัฒนะ รหัส 46010015
ผศ.พลศาสตร์ เลิศประเสริฐ (อาจารย์ที่ปรึกษา)
ภาคเรียนที่ 2 ปีการศึกษา 2549

บทคัดย่อ

เป็นโครงงานจำลองเครนอิเล็คทรอนิคส์สำหรับเคลื่อนย้ายวัตถุ ซึ่งสามารถควบคุมการเคลื่อนที่ได้ทั้งในระบบ บังคับโดยผู้ใช้ (Manual) และ ระบบอัตโนมัติ (Auto) ในระบบ Manual ผู้ใช้จะบังคับให้เครนเคลื่อนที่ไปยังพิกัดต่างๆ รวมทั้งการขึ้นลงของเครนผ่านทางรีโมตคอนโทรลไร้สายได้อย่างอิสระ ส่วนในระบบ Auto ผู้ใช้สามารถระบุพิกัด (X,Y) ใน ซอร์ฟแวร์คอมพิวเตอร์ ซึ่งได้ แก่พิกัด (X1,Y1) คือ พิกัดเริ่มต้น และ (X2,Y2)คือ พิกัดปลายทาง โดยข้อมูลพิกัดจะถูกส่งไปยัง พีไอซีไมโครคอนโทรลเลอร์เพื่อควบคุมเครน ผ่านทางพอร์ทอนุกรม ในการทำงาน เครนจะเริ่มเคลื่อนที่ไปยังพิกัด (X1,Y1) เพื่อรอการยกสัมภาระขึ้นโดยผู้ใช้ จากนั้น จะเคลื่อนที่ไปยังพิกัด (X2,Y2) เพื่อรอการนำสัมภาระลง ผู้ใช้สามารถยกเลิกการทำงานของเครนได้ขณะปฏิบัติงาน ซึ่งทั้งสองระบบ จะมีวงจรป้องกันการชนและหยุดฉุกเฉินได้ตลอดเวลา โดยจะมีตัวเลขพิกัด (X,Y)แสดงทาง ตัวเลข7ส่วน

Two-mode crane

Mr. Krissada Jirawongwattana ID. 46010015

Prof.Assist. Ponlasart Lertprasert (Advisor)

2 nd Semester, Education Year 2006

Abstract

This is a model of electronics crane with two-mode controlable, Manual mode and Auto mode. In Manual mode, user can freely control the crain in X-Y-Z axis by wireless remote control. And in Auto mode, user can specify (X,Y) position in PC software. There are (X1,Y1) for initial position and (X2,Y2) for terminal position. The (X,Y) position data will be sent to PIC microcontroller by serial port. Then the crane will move to (X1,Y1) for loading and move to (X2,Y2) before ending work. However user can stop or cancel working the crane while operating. By the way this model has safty circuit, emergency stop and always show the present position (X,Y) by 7-segment LED.

กิตติกรรมประกาศ

โครงงานและเอกสารประกอบโครงงานนี้จะสำเร็จลุล่วงมาด้วยดีไม่ได้หากขาดอาจารย์ทุกท่านที่ให้คำแนะนำ ดูแลในเรื่องของวงจรอย่างใกล้ชิดมาโดยตลอด ขอบคุณอาจารย์ ผศ. พลศาสตร์ เลิศประเสริฐ ผู้เสียสละเวลาเป็นที่ปรึกษาโครงงาน ท่านคอยช่วยเหลือและสอนให้รู้สิ่งต่างๆและประสบการณ์มากมาย อาจารย์จะแนะนำสิ่งต่างๆที่เป็นเรื่องยากๆให้เป็นเรื่องง่ายๆจนสามารถมองภาพรวมได้ง่าย ท่านดูแลนักศึกษาอย่างใกล้ชิดและให้เครื่องมือเครื่องใช้รวมทั้งห้องสำหรับทดลองโครงงานนี้ ขอบคุณอาจารย์ ประภากร สุวรรณะ เป็นผู้ที่ให้ประสบการณ์และความรู้ในด้านวงจรให้ท่านสามารถบรรยายวงจรให้เข้าใจได้อย่างรวดเร็ว และขอบคุณอาจารย์ทุกท่านที่ช่วยสั่งสอนให้ความรู้ในทุกๆวิชาจนทำให้สามารถนำความรู้ในวิชาต่างๆมาใช้ในโครงงานนี้ ขอบคุณรุ่นพี่ที่ห้องโปรเจค ขอบคุณพ่อและแม่ซึ่งให้ความช่วยเหลือในทุกๆด้าน

นาย กฤษฎา จิรวงศ์วัฒนะ

ผู้จัดทำ

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงงาน

ในปัจจุบัน การเคลื่อนย้ายวัตถุสามารถทำได้หลากหลายวิธี โดยขั้นพื้นฐาน มนุษย์สามารถเรียนรู้เคลื่อนย้ายวัตถุด้วยตนเอง ต่อมา มีการพัฒนาใช้เครื่องผ่อนแรง ทำให้สามารถเคลื่อนย้ายวัตถุสิ่งของโดยเครื่องมือต่างๆ ทำให้สามารถกระทำได้เร็วและแม่นยำ จึงสร้างระบบการเคลื่อนย้ายวัตถุอัตโนมัติเพื่อสามารถนำแนวคิดไปพัฒนาต่อไปได้

1.2 ลักษณะของโครงงาน

เป็นการจำลองเครื่องย้ายวัตถุ หรือ เครน ที่สมารถติดตั้งในระบบใดๆที่จำเป็นต้องเคลื่อนย้ายวัตถุอยู่เสมอ เช่น โกดังเก็บของ การจัดเก็บกล่องจำนวนมากในที่แคบๆ ซึ่งสามารถระบุพิกัดที่ตั้งของวัตถุได้จากคอมพิวเตอร์ ทำให้สามารถเคลื่อนย้ายวัตถุได้โดยง่าย เหมาะสำหรับการเคลื่อนย้ายที่รู้พิกัดโดยแน่นอน หรือ ต้องย้ายไปมาในที่ๆซ้ำกันหลายๆรอบ ก็สามารถกระทำได้ง่าย สามารถใช้พิกัดเดิมได้ และยังเพิ่มความยืดหยุ่นโดยเพิ่มรีโมตคอนโทรลไร้สาย ให้ผู้ใช้สามารถควบคุมการเคลื่อนที่เองได้ ทั้งนี้ใช้ไมโครคอนโทรลเลอร์ PIC ในการควบคุมส่วนของการรับข้อมูลพิกัดจากคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม และสั่งมอเตอร์ให้ไปยังพิกัดที่กำหนด และใช้การรับส่งโดยคลื่นวิทยุในการสั่งในระบบรีโมตคอนโทรล

1.3 ขั้นตอนและวิธีการดำเนินงาน

1) มองภาพรวมและออกแบบระบบเป็นบล็อกการทำงานที่แยกจากกัน
โดยแยกแต่ละส่วนออกจากกัน ซึ่งได้แก่
- ส่วนวงจร เช่น ออกแบบวงจรเซ็นเซอร์ที่ใช้การนับหรือตรวจจับ, วงจรควบคุมมอเตอร์, วงจรจ่ายไฟ, วงจรไมโครคอนโทรลเลอร์ ฯลฯ
- ส่วนโปรแกรม ทั้งในไมโครคอนโทรลเลอร์ PIC และ ในคอมพิวเตอร์

2) ศึกษาการควบคุมโดยใช้ไมโครคอนโทรลเลอร์ PIC และภาษาซี และเขียนโปรแกรมภาษาซีสำหรับไมโครคอนโทรลเลอร์

3) เขียนโปรแกรมในคอมพิวเตอร์โดย Visual Basic

4) ศึกษาการรับส่งข้อมูลทางพอร์ตอนุกรมโดยใช้ Visual Basic และเขียนโปรแกรม

5) ทดสอบในส่วนของโปรแกรมทั้งหมด

6) ออกแบบวงจรและลายพิมพ์ แล้วประกอบอุปกรณ์ และทดสอบวงจร

7) ออกแบบโครงสร้างของแบบจำลอง และประกอบ

8) นำฮาร์ดแวร์และซอฟท์แวร์ประกอบรวมกัน

9) ทดสอบโครงงานทั้งหมด

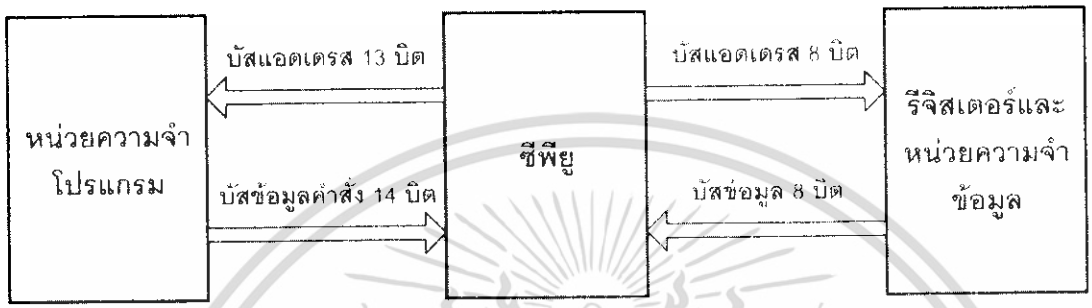
1.4 ประโยชน์ที่คาดว่าจะได้รับจากโครงงานนี้

- ความรู้ความเข้าใจ และสามารถใช้งานไมโครคอนโทรลเลอร์ PIC16อย่างคุ้มค่า
- ความรู้และความเข้าใจในการใช้ ภาษาซี และ การเขียนโปรแกรม
- ความรู้และความเข้าใจในการใช้ Basic และ การพัฒนาโปรแกรมบน Windows
- ความรู้และความเข้าใจในการสื่อสารโดยคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม
- ความรู้และความเข้าใจเรื่องการสื่อสารไร้สายโดยคลื่นวิทยุ
- แนวคิดที่สามารถนำมาประยุกต์และพัฒนาต่อๆไป ให้สามารถใช้งานจริงๆในโครงสร้างที่ใหญ่ขึ้นและสะดวกขึ้นได้ โดยเพิ่มแอ็พพริเคชั่นต่างๆในการใช้งาน

บทที่ 2
ไมโครคอนโทรลเลอร์ PIC16F87x

2.1 สถาปัตยกรรมของไมโครคอนโทรลเลอร์ PIC16F87x

ไมโครคอนโทรลเลอร์ตระกูล PIC มีการแยกหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลออกจากกัน โดยมีบัสสำหรับติดต่อแยกกัน ดังรูปที่ 2.1

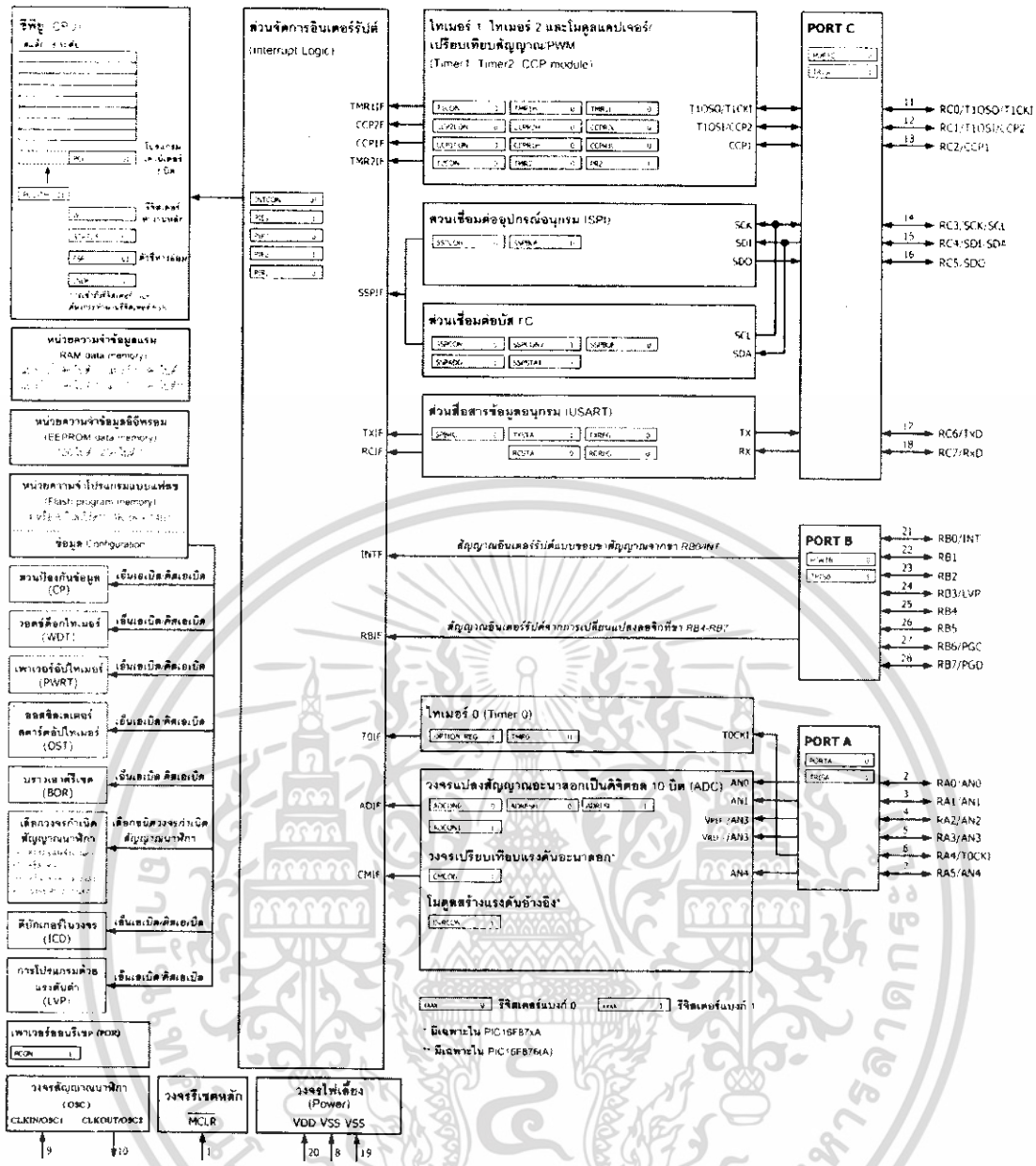


รูป 2.1 ไดอะแกรมแสดงรูปสถาปัตยกรรมของไมโครคอนโทรลเลอร์

ซึ่งการกระคำสั่งจะใช้กระบวนการที่เรียกว่า ไปป์ไลน์ (Pipeline) ทำให้สามารถเฟตช์คำสั่งถัดไปในขณะที่กำลังเอ็กซิคิวต์คำสั่งในปัจจุบัน ส่งผลให้ความเร็วของไมโครคอนโทรลเลอร์สูงขึ้น จึงเป็นที่มาของความสามารถในการกระทำคำสั่ง 1 คำสั่งในสัญญาณนาฬิกา 1 ลูก

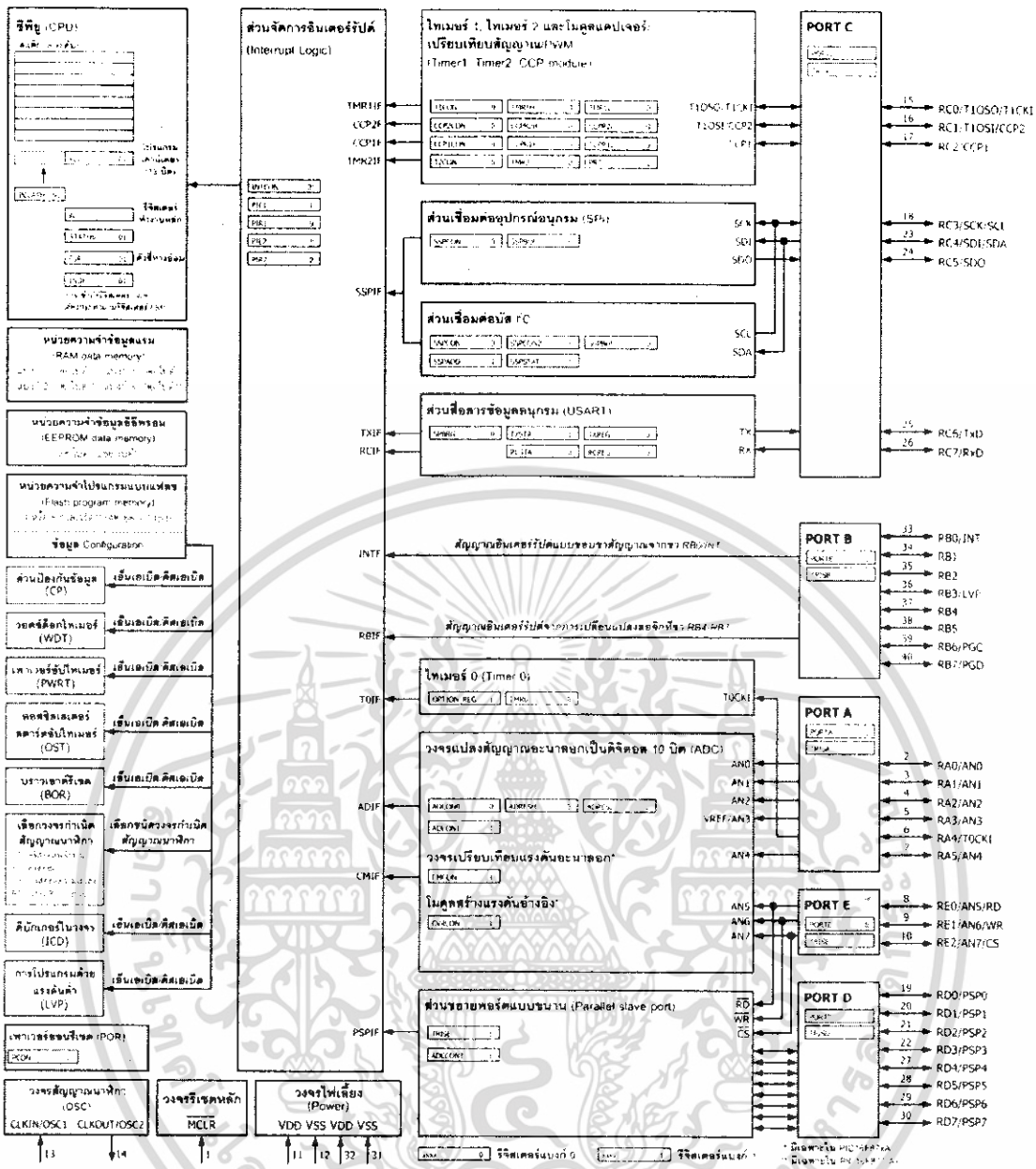
2.2 โครงสร้างของไมโครคอนโทรลเลอร์ PIC16F87x

แสดงดังรูปที่ 2.2 ถึง 2.3



รูป 2.2 โครงสร้างการทำงานของไมโครคอนโทรลเลอร์PIC16F87X

รุ่น28ขา PIC16F873 / PIC16F876(A)



รูป 2.3 โครงสร้างการทำงานของไมโครคอนโทรลเลอร์PIC16F87X

รุ่น40ขา PIC16F874 / PIC16F877(A)

สามารถสรุปได้ดังตาราง 2-1

คุณสมบัติสำคัญ	PIC16F870	PIC16F871	PIC16F872	PIC16F873/873A	PIC16F874/874A	PIC16F876/876A	PIC16F877/877A
ความถี่สัญญาณนาฬิกา	ไฟตรง-20MHz	ไฟตรง-20MHz	ไฟตรง-20MHz	ไฟตรง-20MHz	ไฟตรง-20MHz	ไฟตรง-20MHz	ไฟตรง-20MHz
ควบไรเซต (และหน่วงเวลา)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
หน่วยความจำโปรแกรม	2K x 14 บิต	2K x 14 บิต	2K x 14 บิต	4K x 14 บิต	4K x 14 บิต	8K x 14 บิต	8K x 14 บิต
หน่วยความจำข้อมูลแรม	128 ไบต์	128 ไบต์	128 ไบต์	192 ไบต์	192 ไบต์	368 ไบต์	368 ไบต์
หน่วยความจำข้อมูลอีอีพรอม	64 ไบต์	64 ไบต์	64 ไบต์	128 ไบต์	128 ไบต์	256 ไบต์	256 ไบต์
จำนวนแหล่งกำเนิดอินเตอร์รัปต์	10	11	10	13	14	13	14
จำนวนพอร์ตอินพุตเอาต์พุต	พอร์ต A, B, C 22 บิต	พอร์ต A-E 33 บิต	พอร์ต A, B, C 22 บิต	พอร์ต A, B, C 22บิต	พอร์ต A-E 33 บิต	พอร์ต A, B, C 22 บิต	พอร์ต A-E 33 บิต
จำนวนไทเมอร์/เคาน์เตอร์	3	3	3	3	3	3	3
โมดูลแคปเจอร์/เปรียบเทียบ/PWM	2	2	2	2	2	2	2
ส่วนสื่อสารข้อมูลอนุกรม	USART	USART	SPI, I²C	SPI, I²C, USART	SPI, I²C, USART	SPI, I²C, USART	SPI, I²C, USART
ส่วนสื่อสารข้อมูลขนาน	-	PSP	-	-	PSP	-	PSP
วงจร ADC 10 บิต	5 ช่อง	8 ช่อง	5 ช่อง	5 ช่อง	8 ช่อง	5 ช่อง	8 ช่อง
วงจรเปรียบเทียบแรงดันอะนาลอก	-	-	-	2 ช่อง (PIC16F873A)	2 ช่อง (PIC16F874A)	2 ช่อง (PIC16F876A)	2 ช่อง (PIC16F877A)
โมดูลสร้างแรงดันอ้างอิง	-	-	-	1 ชุด (PIC16F873A)	1 ชุด (PIC16F874A)	1 ชุด (PIC16F876A)	1 ชุด (PIC16F877A)
จำนวนคำสั่ง	35	35	35	35	35	35	35

ตาราง 2.1 ตารางเปรียบเทียบคุณสมบัติของไมโครคอนโทรลเลอร์ PIC16F87x แต่ละเบอร์

2.3 การใช้งานและจัดวางตำแหน่งขาของไมโครคอนโทรลเลอร์ PIC16F87x

แสดงดังตารางที่ 2.2

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
OSC1/CLKIN	9 (13)	อินพุต	ชมิตต์ทริกเกอร์/ซีมอส [3][4]	ขาออสซิลเลอร์ รับสัญญาณนาฬิกาจากภายนอก
OSC2/CLKOUT	10 (14)	เอาต์พุต	-	ขาออสซิลเลอร์ ในโหมด RC เป็นขาเอาต์พุต สัญญาณนาฬิกาความถี่ 1/4 ของสัญญาณที่ขา OSC1
$\overline{MCLR}$/Vpp	1	อินพุต	ชมิตต์ทริกเกอร์	- ขารับสัญญาณรีเซตหลัก (Master Clear Input) ทำงานที่ลอจิก "0" - ขารับแรงดันโปรแกรม (programming voltage)
ขาพอร์ต A เป็นขาพอร์ต 2 ทิศทาง				
RA0/AN0	2	อินพุต/เอาต์พุต	ทีทีแอล/อะนาลอก	- ขาพอร์ต RA0 - อินพุตวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ช่อง 0
RA1/AN1	3	อินพุต/เอาต์พุต	ทีทีแอล/อะนาลอก	- ขาพอร์ต RA1 - อินพุตวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ช่อง 1
RA2/AN2/VREF-/ CVREF*	4	อินพุต/เอาต์พุต	ทีทีแอล/อะนาลอก	- ขาพอร์ต RA2 - อินพุตวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ช่อง 2 - อินพุตแรงดันอ้างอิงลบของวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล - เอาต์พุตแรงดันอ้างอิงของโมดูลแรงดันอ้างอิง (PIC16F87xA)
RA3/AN3/VREF+	5	อินพุต/เอาต์พุต	ทีทีแอล/อะนาลอก	- ขาพอร์ต RA3 - อินพุตวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ช่อง 3 - อินพุตแรงดันอ้างอิงบวกของวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล
RA4/T0CKI/ C1OUT*	6	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RA4 กรณีใช้พอร์ตเอาต์พุตมีโครงสร้างแบบเดรนเปิด - อินพุตสัญญาณนาฬิกาของไทเมอร์ 0 - เอาต์พุตวงจรเปรียบเทียบแรงดันอะนาลอกของ 1 (PIC16F87/xA)
RA5/AN4/$\overline{SS}$/ C2OUT*	7	อินพุต/เอาต์พุต	ทีทีแอล/อะนาลอก	- ขาพอร์ต RA5 - อินพุตวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ช่อง 4 - ขาสัญญาณ Slave Select ใช้ในการสื่อสารข้อมูลอนุกรมแบบซิงโครนัส - เอาต์พุตวงจรเปรียบเทียบแรงดันอะนาลอกของ 2 (PIC16F87/xA)
ขาพอร์ต B เป็นขาพอร์ต 2 ทิศทาง สามารถกำหนดให้ต่อตัวต้านทานพูลอัปภายในเมื่อทำงานเป็นอินพุตได้ทางซอฟต์แวร์				
RB0/INT	21 (33)	อินพุต/เอาต์พุต	ทีทีแอล/ชมิตต์ทริกเกอร์ [1]	- ขาพอร์ต RB0 - อินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอก
RB1	22 (34)	อินพุต/เอาต์พุต	ทีทีแอล	- ขาพอร์ต RB1
RB2	23 (35)	อินพุต/เอาต์พุต	ทีทีแอล	- ขาพอร์ต RB2
RB3/LVP	24 (36)	อินพุต/เอาต์พุต	ทีทีแอล	- ขาพอร์ต RB3 - อินพุตรับแรงดันโปรแกรมต่ำ (+5V) ถ้าเซ็นเอเบิลไว้
RB4	25 (37)	อินพุต/เอาต์พุต	ทีทีแอล	- ขาพอร์ต RB4 และสามารถเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงลอจิกขึ้นที่ขานี้ หากเซ็นเอเบิลไว้
RB5	26 (38)	อินพุต/เอาต์พุต	ทีทีแอล	- ขาพอร์ต RB5 และสามารถเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงลอจิกขึ้นที่ขานี้ หากเซ็นเอเบิลไว้
RB6/PGC	27 (39)	อินพุต/เอาต์พุต	ทีทีแอล/ชมิตต์ทริกเกอร์ [2]	- ขาพอร์ต RB6 - เป็นขาสัญญาณนาฬิกาของการดีบักในวงจร (ICD) - สามารถเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงลอจิกขึ้นที่ขานี้ หากเซ็นเอเบิลไว้
RB7/PGD	28 (40)	อินพุต/เอาต์พุต	ทีทีแอล/ชมิตต์ทริกเกอร์ [2]	- ขาพอร์ต RB7 - เป็นขาสัญญาณข้อมูลของการดีบักในวงจร (ICD) - สามารถเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงลอจิกขึ้นที่ขานี้ หากเซ็นเอเบิลไว้

ตาราง 2.2ก ตารางสรุปการทำงานของขาพอร์ต OSC, MCLR, RA, RB

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
ขาพอร์ต C เป็นขาพอร์ต 2 ทิศทาง				
RC0/T1OSO/ T1CKI	11 (15)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC0 - เอาต์พุตวงจรออสซิลเลเตอร์ของไทม์เมอร์ 1 อินพุตสัญญาณนาฬิกาของไทม์เมอร์ 1
RC1/T1OSI/ CCP2	12 (16)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC1 อินพุตวงจรออสซิลเลเตอร์ของไทม์เมอร์ 1 อินพุตของแคปเจอร์ เอาต์พุตวงจรเปรียบเทียบ เอาต์พุต PWM สำหรับโมดูล CCP2
RC2/CCP1	13 (17)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC2 - อินพุตของแคปเจอร์ เอาต์พุตวงจรเปรียบเทียบ เอาต์พุต PWM สำหรับโมดูล CCP1
RC3/SCK/SCL	14 (18)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC3 - ขาสัญญาณนาฬิกาของวงจร SPI และระบบบัส I²C
RC4/SDI/SDA	15 (23)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC4 - ขาข้อมูลอินพุตวงจร SPI - ขาข้อมูลสัญญาณของระบบบัส I²C
RC5/SDO	16 (24)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC5 - ขาข้อมูลเอาต์พุตของวงจร SPI
RC6/TxD	17 (25)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC6 - ขาเอาต์พุตวงจร USART สำหรับเชื่อมต่อพอร์ตอนุกรม
RC7/RxD	18 (26)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RC7 - ขาอินพุตวงจร USART สำหรับเชื่อมต่อพอร์ตอนุกรม
ขาพอร์ต D เป็นขาพอร์ต 2 ทิศทาง สามารถใช้เป็นส่วนขยายพอร์ตแบบขนานเพื่อติดต่อกับระบบบัสอื่น ไม่มีใน PIC16F873(A)/876(A)				
RD0/PSP0	(19)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD0 - ขาขยายพอร์ตแบบขนานบิต 0
RD1/PSP1	(20)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD1 - ขาขยายพอร์ตแบบขนานบิต 1
RD2/PSP2	(21)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD2 - ขาขยายพอร์ตแบบขนานบิต 2
RD3/PSP3	(22)	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD3 - ขาขยายพอร์ตแบบขนานบิต 3
RD4/PSP4	(27)	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD4 - ขาขยายพอร์ตแบบขนานบิต 4
RD5/PSP5	(28)	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD5 - ขาขยายพอร์ตแบบขนานบิต 5
RD6/PSP6	(29)	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD6 - ขาขยายพอร์ตแบบขนานบิต 6
RD7/PSP7	(30)	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์ / ทีทีแอล[3]	- ขาพอร์ต RD7 - ขาขยายพอร์ตแบบขนานบิต 7

ตาราง 2.2ข ตารางสรุปการทำงานของขาพอร์ต RC, RD

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
ขาพอร์ต E เป็นขาพอร์ต 2 ทิศทาง ไม่มีใน PIC16F873(A)/876(A)				
RE0/AN5/$\overline{RD}$	(8)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์ ทีทีแอล(3)	- ขาพอร์ต RE0 อินพุตของสัญญาณนาฬิกาหรือข้อมูล - ขาสัญญาณ $\overline{RD}$ สำหรับส่วนขยายพอร์ตแบบขนาน
RE1/AN6/$\overline{WR}$	(9)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์ ทีทีแอล(3)	- ขาพอร์ต RE1 อินพุตของสัญญาณนาฬิกาหรือข้อมูล ขาสัญญาณ $\overline{WR}$ สำหรับส่วนขยายพอร์ตแบบขนาน
RE2/AN7/$\overline{CS}$	(10)	อินพุต เอาต์พุต	ชมิตต์ทริกเกอร์ ทีทีแอล(3)	- ขาพอร์ต RE2 อินพุตของสัญญาณนาฬิกาหรือข้อมูล ขาสัญญาณ $\overline{CS}$ สำหรับส่วนขยายพอร์ตแบบขนาน
ขาต่อไฟเลี้ยง				
VDD	20 (11, 32)	อินพุต		ขาต่อไฟเลี้ยง ใช้ได้ตั้งแต่ +2 ถึง +5.5V
Vss	8, 19 (12, 31)	อินพุต		ขาต่อกราวด์

หมายเหตุ

(1) อินพุตของวงจรบัฟเฟอร์จะเป็นแบบชมิตต์ทริกเกอร์ เมื่อใช้งานเป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอก

(2) อินพุตของวงจรบัฟเฟอร์จะเป็นแบบชมิตต์ทริกเกอร์ เมื่อทำงานในโหมดโปรแกรมข้อมูลอนุกรม (Serial programming mode)

(3) อินพุตของวงจรบัฟเฟอร์จะเป็นแบบชมิตต์ทริกเกอร์ เมื่อกำหนดให้ทำงานเป็นขาพอร์ตปกติ และเป็นแบบทีทีแอลเมื่อกำหนดให้ทำงานเป็นส่วนขยายพอร์ตแบบขนาน (PSP) สำหรับเชื่อมต่อกับระบบบัสไมโครโปรเซสเซอร์อื่น

(4) สำหรับ PIC16F874/877 อินพุตของวงจรบัฟเฟอร์จะเป็นแบบชมิตต์ทริกเกอร์ เมื่อกำหนดให้ทำงานในโหมด RC และเป็นซีมอสเมื่อทำงานในโหมดอื่น

ตาราง 2.2ค ตารางสรุปการทำงานของขาพอร์ต RE, VDD, Vss

2.4 โหมดสัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ PIC16F87x

ไมโครคอนโทรลเลอร์ PIC16F877 สามารถเลือกโหมดสัญญาณนาฬิกาได้เพื่อกำหนดจังหวะการทำงานดังนี้

2.4.1 โหมด LP

ใช้กับคริสตอลหรือเซรามิคเรโซเนเตอร์พลังงานต่ำ ความถี่ 32 kHz ~ 200 kHz เข้ากับขา OSC1 และ OSC2

2.4.2 โหมด XT

ใช้กับคริสตอลหรือเซรามิคเรโซเนเตอร์มารตรฐาน ความถี่ 200 kHz ~ 4 MHz เข้ากับขา OSC1 และ OSC2

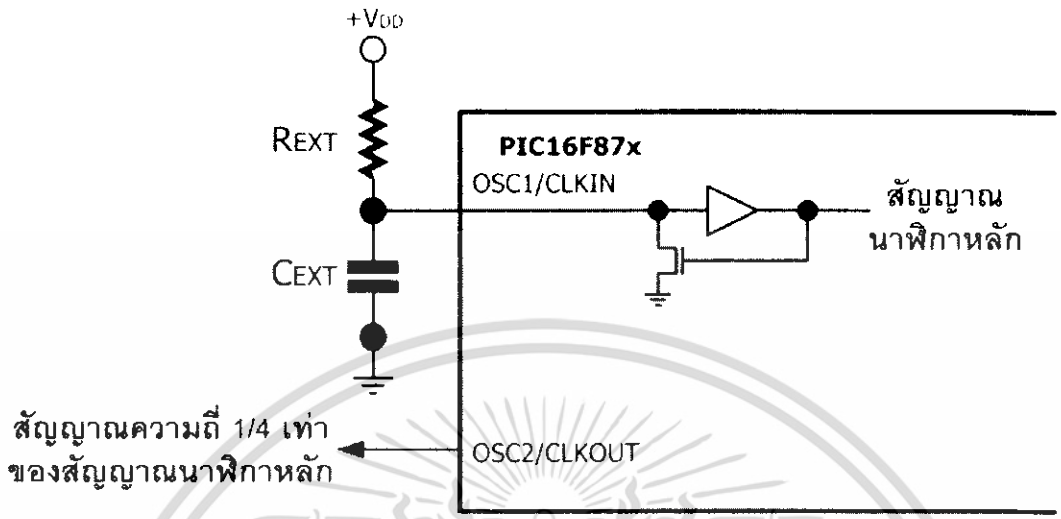
2.4.3 โหมด HS

ใช้กับคริสตอลหรือเซรามิคเรโซเนเตอร์ความถี่สูง 4 MHz ~ 20 MHz เข้ากับขา OSC1 และ OSC2

2.4.4 โหมด RC (External Resistor Capacitor)

สามารถกำหนดค่าความถี่ได้จากค่าของตัวต้านทานและตัวเก็บประจุที่ต่อเข้ากับขา OSC1/CLKIN ดังรูปที่ 2.4 ความถี่สูงสุดคือ 4MHz (f=1/RC) อย่างไรก็ตาม ไม่สามารถกำหนดความถี่ในโหมดนี้ได้อย่างชัดเจน เนื่องจากต้องพิจารณาองค์ประกอบที่สามารถเปลี่ยนแปลงได้ในขอบเขตที่กว้าง ไม่ว่าจะเป็นค่าของแรงดันไฟเลี้ยง, ค่าความต้านทานและตัวเก็บประจุ ซึ่งรวมไปถึงค่าความ

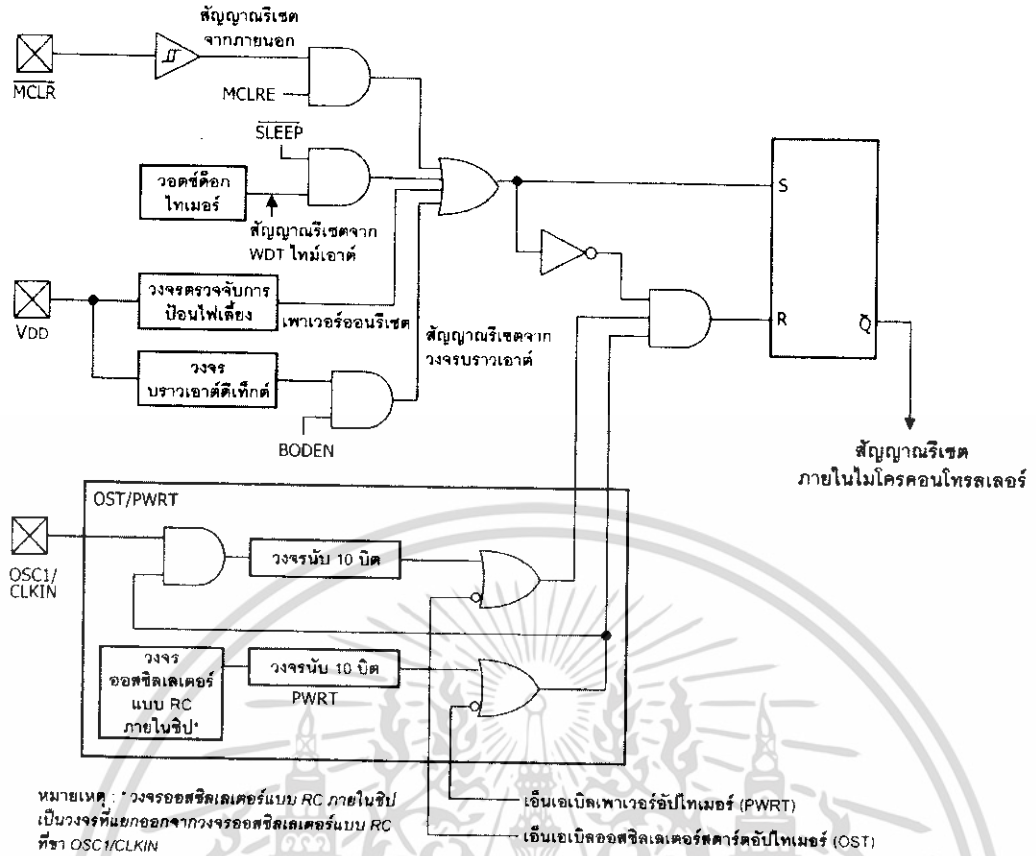
ผิดพลาดในตัวของอุปกรณ์เอง อย่างไรก็ตาม ค่าของคาปาซิเตอร์ที่เหมาะสมอยู่ในช่วง 3k-100k และตัวเก็บประจุควรมีค่ามากกว่า 20pF นอกจากนั้นที่ขา OSC2/CLKOUT จะมีสัญญาณความถี่ 1/4 เท่าของความถี่หลักส่งออกมา



รูป 2.4 การต่อตัวต้านทานและตัวเก็บประจุที่ขา OSC1

2.5 กระบวนการรีเซ็ตของไมโครคอนโทรลเลอร์ PIC16F87x

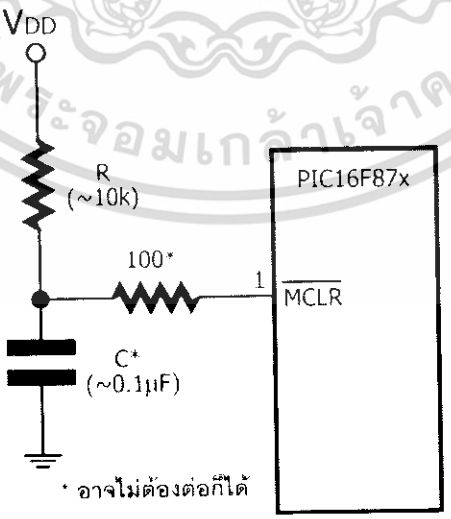
รีเซ็ต (Reset) เป็นกระบวนการกำหนดให้ซีพียูภายในไมโครคอนโทรลเลอร์ เริ่มต้นทำงานใหม่ เพื่อประโยชน์ในการแก้ไขความผิดปกติ หรือการทำงานที่ผิดพลาด ซึ่งทำให้ทำงานค้างอยู่ที่สถานะใดสถานะหนึ่งหรือหยุดทำงาน เมื่อเกิดการรีเซ็ต ไมโครคอนโทรลเลอร์จะกลับมาเริ่มต้นทำงานใหม่ ในไมโครคอนโทรลเลอร์ PIC16F877 ดังรูป 2.5



รูป 2.5 กลไกทางลอจิคของวงจรรีเซ็ตในไมโครคอนโทรลเลอร์ PIC16F87x

2.5.1 พาวเวอร์ออนรีเซ็ต (Power-on Reset)

เป็นการรีเซ็ตหลังการเริ่มต้นจ่ายไฟเลี้ยงใหม่ ซึ่งมีนัยสำคัญสูงสุด เป็นการเตรียมพร้อมให้กับไมโครคอนโทรลเลอร์ แสดงดังรูปที่ 2.6 โดยการรีเซ็ตจะทำให้ไมโครคอนโทรลเลอร์เริ่มต้นทำงานใหม่ที่ แอดเดรส 0X0000 ใหม่ โดยไม่กระทบข้อมูลในหน่วยความจำแรม



รูป 2.6 วงจรพาวเวอร์ออนรีเซ็นสำหรับไมโครคอนโทรลเลอร์ PIC16F87x

2.5.2 การรีเซ็ตที่ขา 1(/MCLR)

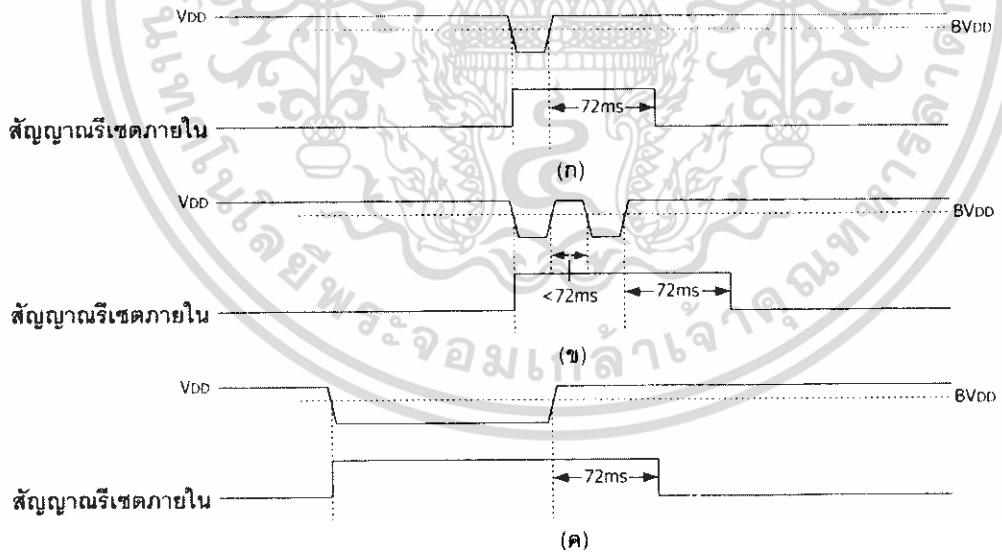
ในการทำงานปกติขา /MCLRจะต่อเข้ากับลอจิค "1" (โดยความต้านทานพูลอัพ) แต่เมื่อขา /MCLR ได้รับลอจิค "0" (เช่นใน Power-On Reset โดยตัวเก็บประจุลัดวงจรชั่วขณะ) จะเกิดการรีเซ็ตที่ ไมโครคอนโทรลเลอร์ เช่นเดียวกับ Power-on Reset ซึ่งในทางปฏิบัติ จะใช้วงจรเดียวกับรูปที่ 2.6 แต่จะต่อสวิตซ์กดติดปล่อยดับที่ขานี้กับกราวด์เพิ่ม เพื่อสามารถกดรีเซ็ตโดยผู้ใช้เองได้

2.5.3 การรีเซ็ตเนื่องจากวอตช์ด็อกไทเมอร์

สามารถกำหนดได้ทางซอฟแวร์ เกิดขึ้นเมื่อเอ็นเนเบิลให้วอตช์ด็อกไทเมอร์ทำงาน ซึ่งจะเคลียรีค่าการนับของไทเมอร์ได้ทันภายในคาบเวลาของวอตช์ด็อกไทเมอร์ เนื่องจากไมโครคอนโทรลเลอร์อาจทำงานผิดพลาด วนอยู่กับบางคำสั่ง ทำให้วอตช์ด็อกไทเมอร์เกิดไทม์เอ็าท์ ตัวซ์ด็อกไทเมอร์จะส่งสัญญาณไปยังซีพียู เพื่อกระทำการรีเซ็ตไมโครคอนโทรลเลอร์ และถ้าไมโครคอนโทรลเลอร์ อยู่ในโหมดสลีป วอตช์ด็อกไทเมอร์ไทม์เอ็าท์จะกระตุ้นให้เวกอัปกลับมาสู่โหมดการทำงานปกติ

2.5.4 บราวเอาต์รีเซ็ต

ไมโครคอนโทรลเลอร์ PIC16F877 มีวงจรตรวจจับระดับแรงดันไฟเลี้ยงที่ต่ำกว่ากำหนด หรือ บราวเอาต์ดีเท็ก (Brown Out Detect : BOD) เมื่อพบระดับแรงดันต่ำกว่ากำหนด(3.7V ~ 4.3V) จะเกิดการรีเซ็ตภายใน และหน่วงเวลา 72 ms ดังรูป 2.7 ซึ่งกระบวนการนี้ สามารถกำหนดได้ทางซอฟแวร์



รูป 2.7 ไดอะแกรมเวลาแสดงการเกิดบาวเอาต์รีเซ็ตในลักษณะต่างๆ

ก. กรณีเกิดไฟเลี้ยงตกชั่วขณะ

ข. กรณีเกิดไฟเลี้ยงกระเพื่อม

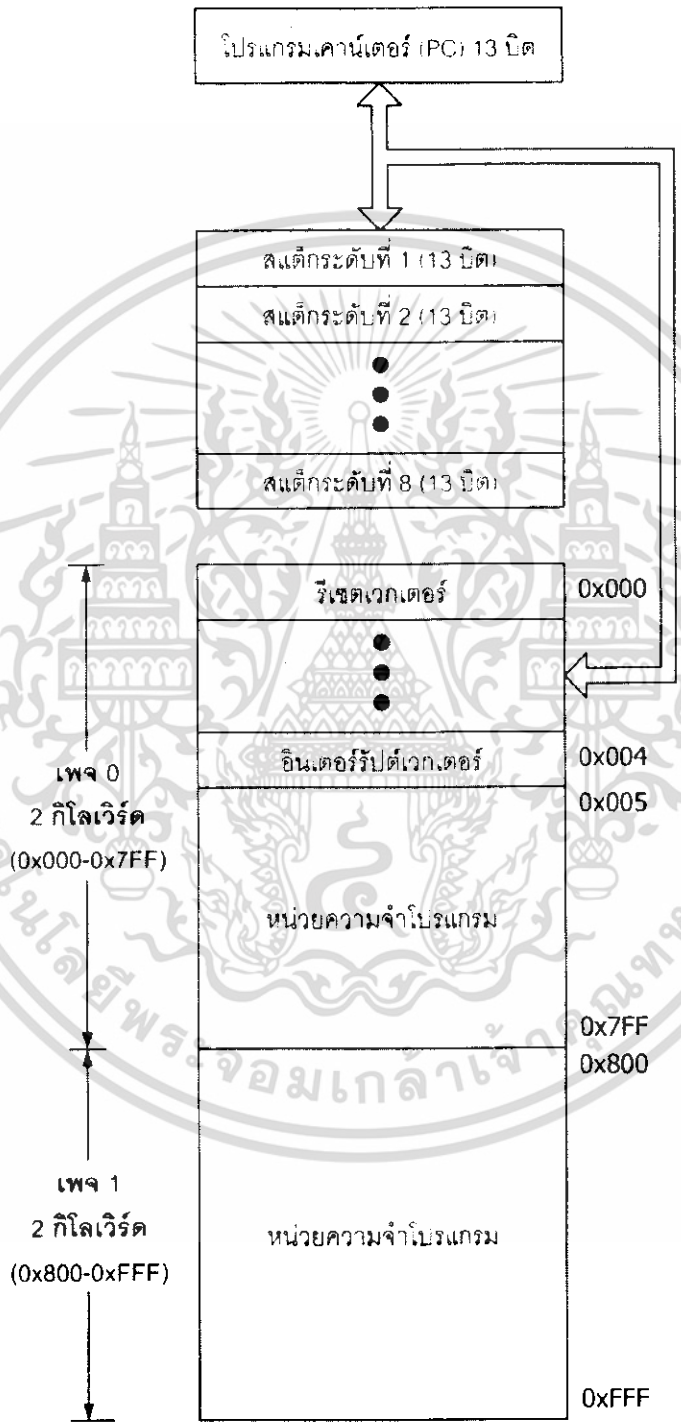
ค. กรณีเกิดไฟเลี้ยงตกยาวนาน

2.6 การจัดสรรหน่วยความจำของไมโครคอนโทรลเลอร์ PIC16F87x

ไมโครคอนโทรลเลอร์ PIC16F87x มีหน่วยความจำให้ใช้งาน 3 ประเภท ได้แก่

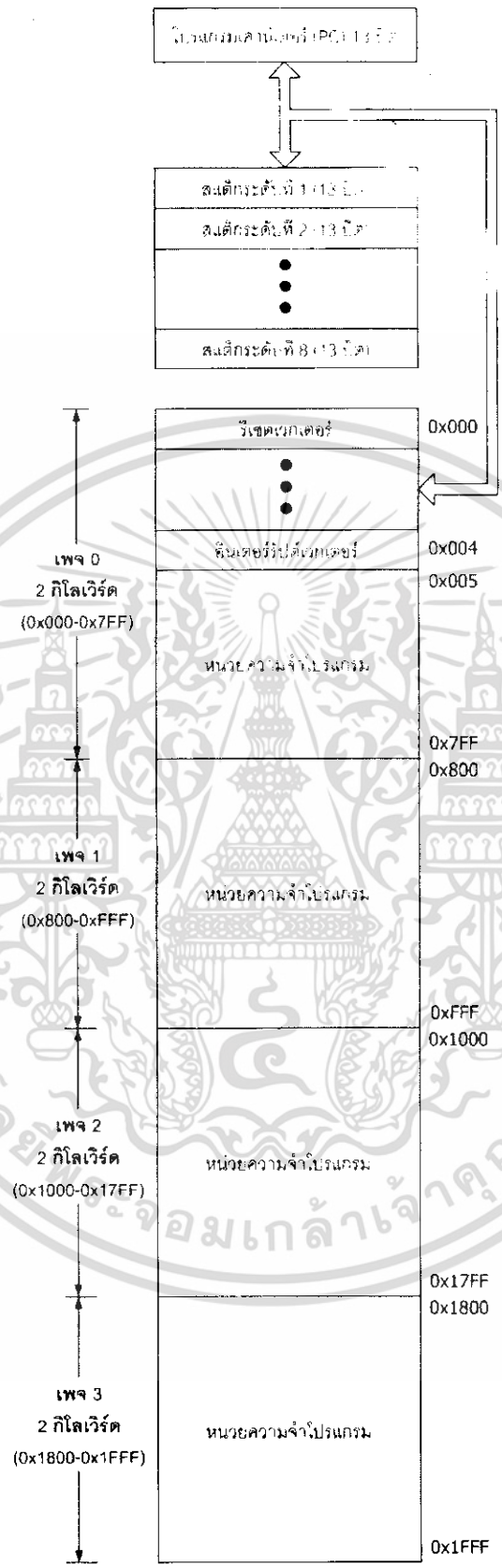
2.6.1 หน่วยความจำโปรแกรมแบบแฟลช (ROM)

สามารถลบและเขียนใหม่ได้กว่า 100,000รอบ มีความจุ 2 ~ 8 กิโลเวิร์ด สำหรับเก็บชุดคำสั่งทั้งหมด แสดงดังรูป 2.8



(ก)

รูป 2.8ก การจัดสรรหน่วยความจำข้อมูลโปรแกรมของไมโครคอนโทรลเลอร์ PIC16F873(A) / 874(A)

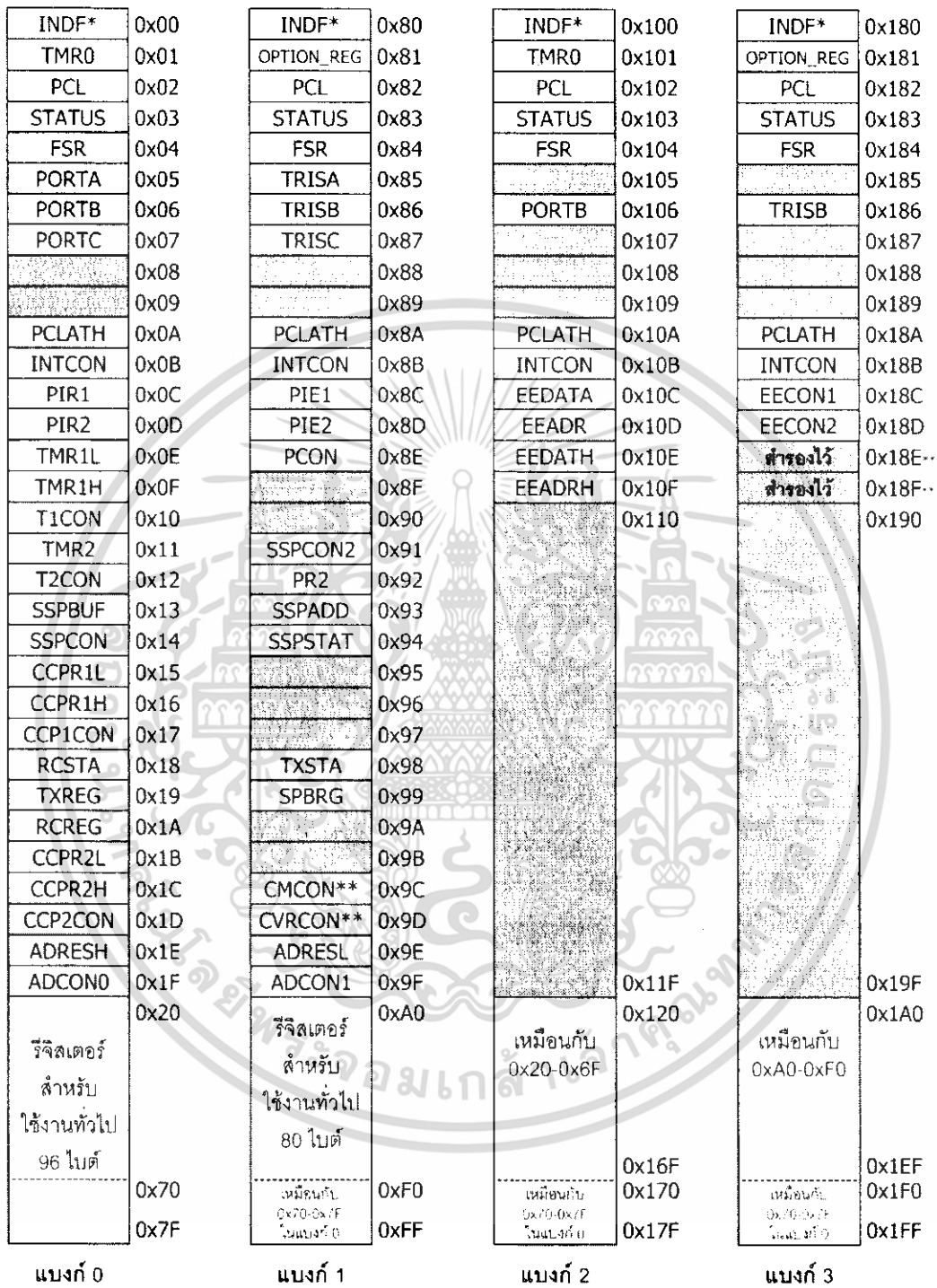


(ข)

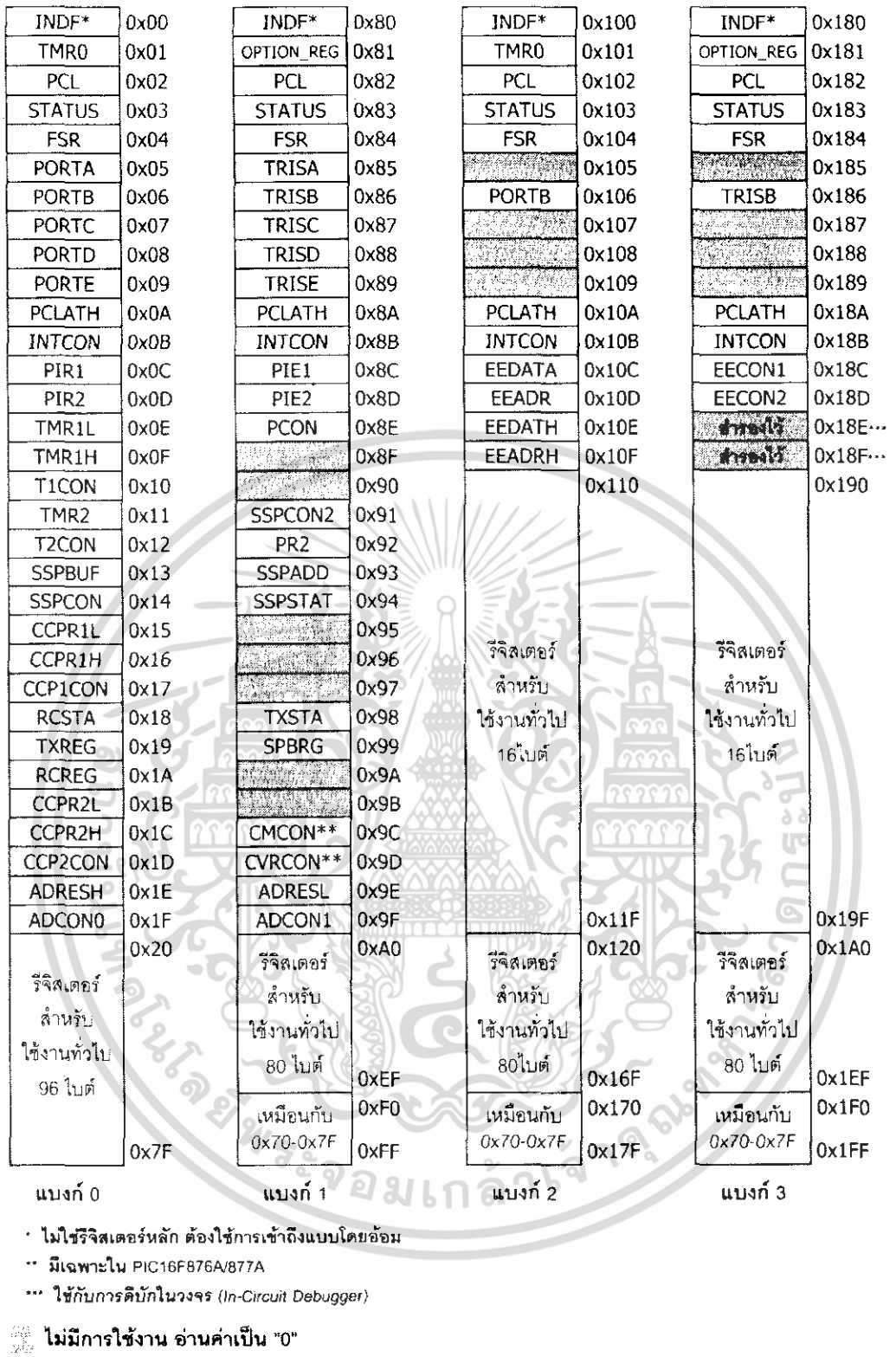
รูป 2.8ข การจัดสรรหน่วยความจำข้อมูลโปรแกรมของไมโครคอนโทรลเลอร์ PIC16F876(A) / 877(A)

2.6.2 หน่วยความจำข้อมูล (RAM)

ขนาด 129 ~ 368 ไบต์ เป็นพื้นที่ของรีจิสเตอร์ สำหรับเก็บข้อมูลควบคุมการทำงาน และข้อมูลประมวลผล แสดงดังรูป 2.9



รูป 2.9ก การจัดสรรหน่วยความจำข้อมูลแรมของไมโครคอนโทรลเลอร์ PIC16F873(A) / 874(A)



รูป 2.9ข การจัดสรรหน่วยความจำข้อมูลแรมของไมโครคอนโทรลเลอร์ PIC16F876(A) / 877(A)

2.6.3 หน่วยความจำอีอีพรอม

เป็นหน่วยความจำที่สามารถเขียนใหม่ได้กว่า 100,000รอบ ใช้เก็บข้อมูลเช่นเดียวกับรอม แต่ผู้ใช้สามารถเข้าถึงได้โดยตรง มีขนาดตั้งแต่ 128 ~ 368 ไบต์ ขึ้นกับเบอร์ของไมโครคอนโทรลเลอร์

บทที่ 3
การใช้ภาษาซีในไมโครคอนโทรลเลอร์ PIC

3.1 พื้นฐานภาษาซี

โครงสร้างของภาษาซีในรูปแบบมาตรฐาน (ANSI Standard C) จะประกอบไปด้วยรายละเอียดดังนี้

3.1.1 พรีโปรเซสเซอร์ไดเร็กทีฟ (Preprocessor Directives)

เป็นชุดคำสั่งที่ใช้ในการจัดเตรียมข้อมูลสำหรับประมวลผล โดยจะกระทำก่อนที่จะคอมไพล์ (Complie)เป็นภาษาเครื่อง โดยพรีโปรเซสเซอร์ไดเร็กทีฟ เช่น การกำหนดคุณสมบัติข้างต้นของไมโครคอนโทรลเลอร์ที่ใช้ ดังนี้

```
#define _PIC16F877_ // Use PIC16F877
#include <16f877.h> // Standard header file for the PIC16F877 device
#fuses HS // Use highspeed oscillator
#fuses NOLVP, NOWDT //No low voltage Program, No watchdog timer
#fuses NOPROTECT // No code protection
#define CLOCK_SP 20000000 // Clock frequency 20MHz
#use delay (clock=CLOCK_SP) // Use built-in function: delay_ms() & delay_us()
```

สำหรับไมโครคอนโทรลเลอร์ PIC มีพรีโปรเซสเซอร์ไดเร็กทีฟ แสดงดังตาราง3.1

พรีโปรเซสเซอร์มาตรฐาน	ความหมายและตัวอย่างการใช้งาน

#DEFINE ID STRING	กำหนดให้ ID เท่ากับ STRING
	ตัวอย่าง
	`#define BITS 8`
	`a=a+BITS; // same as a=a+8;`
	`#define hi(x) (x<<4)`
	`a=hi(a); // same as a=(a<<4);`

#IF expr
#ELSE
#ENDIF
	กำหนดการแปลอย่างมีเงื่อนไขแบบ if(expr)... else... endif
	ตัวอย่าง
	`#if MAX_VALUE > 255`
	`long value; // ถ้า MAX_VALUE มากกว่า 255 บรรทัดนี้จะถูกแปล`
	`#else`
	`int value; // ถ้า MAX_VALUE น้อยกว่า 255 บรรทัดนี้จะถูกแปล`
	`#endif`

#ERROR	สร้างข้อผิดพลาดเตือนในขณะคอมไพล์โปรแกรม
	ตัวอย่าง
	`#if BUFFER_SIZE>16`
	`#error Buffer size is too large`
	`#endif // ถ้า BUFFER_SIZE มากกว่า 16 ข้อความหลังไดเร็กทีฟ #error จะเตือน`

#IFDEF id
#IFNDEF id
	กำหนดการแปลอย่างมีเงื่อนไขแบบ มีการกำหนด ifdef และไม่มีการกำหนด ifndef
	ตัวอย่าง
	`#define DEBUG // Comment line out for no debug`
	`...`
	`#ifdef DEBUG`
	`printf("debug point a");` // ถูกเรียกใช้งานเพราะมีการกำหนด DEBUG
	`#endif`
	`#ifndef DEUBG_V2`
	`printf("debug two point");` // ถูกเรียกใช้งานเพราะไม่มีกำหนด DEBUG_V2
	`#endif`

#INCLUDE "FILENAME"	นำไฟล์จากที่กำหนดในไดเร็กทีฟ #include เข้ามาคอมไพล์ร่วมด้วย
	ตัวอย่าง
	`#include "16C54.H"`
	`#include "C:\INCLUDES\COMLIB\MYRS232.C"`

#INCLUDE <FILENAME>	นำไฟล์ที่กำหนดในไดเร็กทีฟ #include<...> มาคอมไพล์ร่วม แล้วเก็บในไดเร็กตอรีของคอมไพเลอร์ด้วย
	ตัวอย่าง
	`#include <16C54.H>`
	`#include <C:\INCLUDES\COMLIB\MYRS232.C>`

#LIST
#NOLIST
	รวมรายละเอียดลงในลิสต์ไฟล์ (.LST) ขณะคอมไพล์ ส่วน #NOLIST จะไม่รวมรายละเอียด
	ตัวอย่าง
	`#NOLIST // ไม่ต้องแสดงไฟล์ cdriver.h ในลิสต์ไฟล์`
	`#include <cdriver.h>`
	`#LIST // รายละเอียดต่อจากนี้ให้รวมใน ลิสต์ไฟล์`

#PRAGMA cmd	กำหนดให้ใช้ไมโครคอนโทรลเลอร์เบอร์ที่กำหนดขณะที่คอมไพล์ ตัวอย่าง `#pragma device PIC16F877 // ใช้ ไมโครคอนโทรลเลอร์ PIC เบอร์ PIC16F877` ใน CCS C คอมไพเลอร์ สามารถใช้งานไดเร็กทีฟ #device แทนได้ เช่น #device pic16f877

ตาราง 3.1 ตารางแสดงรายละเอียดของพรีโปรเซสเซอร์ไดเร็กทีฟใน CCSC คอมไพเลอร์ (มีต่อ)

พรีโปรเซสเซอร์มาตรฐาน	ความหมายและตัวอย่างการใช้งาน

#UNDEF id	ยกเลิกสิ่งที่ได้กำหนดไว้ในไดเร็กทีฟ #DEFINE
	ตัวอย่าง
	`#define MAXSIZE 1000`
	`#if MAXSIZE<200`
	`#undef MAXSIZE`
	`#define MAXSIZE 200`
	`#endif`

Function Qualifier	ความหมายและตัวอย่างการใช้งาน

#INLINE	กำหนดให้ฟังก์ชั่นที่ต่อจากไดเร็กทีฟ #inline เป็นฟังก์ชั่นแบบ inline function เมื่อมีการเรียกใช้ฟังก์ชั่นนี้ โค้ดของฟังก์ชั่นที่กำหนดเป็น inline จะถูกคัดลอกไปยังที่มีการเรียกใช้งาน
	ตัวอย่าง
	`#inline`
	`swapbyte(int &a, int &b) {`
	`int t;`
	`t=a;`
	`a=b;`
	`b=t;`
	`}`

#INT_DEFAULT	กำหนดให้ฟังก์ชั่นที่ต่อจากไดเร็กทีฟ #int_default เป็นฟังก์ชั่นอินเตอร์รัปต์เริ่มต้น ถ้าอินเตอร์รัปต์ถูกการกระตุ้นและไม่มีการกำหนดฟังก์ชั่นอินเตอร์รัปต์
	ตัวอย่าง
	`#int_default`
	`default_isr() {`
	`printf("Unexplained interrupt\r\n");`
	`}`

#INT_GLOBAL	กำหนดให้ฟังก์ชั่นที่ต่อจากไดเร็กทีฟ #int_global เป็นฟังก์ชั่นอินเตอร์รัปต์แบบโกลบอล โดยปกติแล้วไม่นิยมใช้ แต่ถ้าใช้ต้องระวังเนื่องจากคอมไพเลอร์จะไม่สร้างโค้ดเริ่มต้นและเคลียร์ค่าให้ ทั้งยังไม่มีการเก็บค่าใด ๆ ในรีจิสเตอร์ (ใช้สร้างอินเตอร์รัปต์ด้วยตนเอง)
	ตัวอย่าง
	`#int_global`
	`isr() {  // Will be located at location 4`
	`#asm`
	`bsf isr_flag`
	`retfie`
	`#endasm`
	`}`

#INT_xxx	กำหนดให้ฟังก์ชั่นที่ต่อจากไดเร็กทีฟ #INT_xxx เป็นฟังก์ชั่นอินเตอร์รัปต์ตามที่กำหนดในไดเร็กทีฟ xxx แทนชื่ออินเตอร์รัปต์ต่างๆ เช่น
	`#INT_AD  Analog to digital conversion complete`
	`#INT_ADOF  Analog to digital conversion timeout`
	`#INT_CCP1  Capture or Compare on unit 1` เป็นต้น (ดูเพิ่มเติมใน Help)
	ตัวอย่าง
	`#int_ad`
	`adc_handler() {`
	`adc_active=FALSE;`
	`}`
	
	`#int_rtcc  noclear`
	`isr() {`
	`...`
	`}`

ตาราง 3.1 ตารางแสดงรายละเอียดของพรีโปรเซสเซอร์ไดเร็กทีฟใน CCSC คอมไพเลอร์ (มีต่อ)

Function Qualifier	ความหมายและตัวอย่างการใช้งาน

#SEPARATE	กำหนดให้ฟังก์ชั่นที่ต่อจากไดเร็กติฟ #separate ไม่ใช่ฟังก์ชั่นแบบ INLINE
	เพื่อป้องกันไม่ให้คอมไพเลอร์สร้างโพรซีเยอร์ INLINE อัตโนมัติ
	ตัวอย่าง
	`#separate`
	`swapbyte (int *a, int *b) {`
	`  int t;`
	`  t=*a;`
	`  *a=*b;`
	`  *b=t;`
	`}`

ควบคุมคอมไพเลอร์	ความหมายและตัวอย่างการใช้งาน

#CASE	กำหนดให้คอมไพเลอร์ใช้ case sensitive (พิจารณาตัวพิมพ์เล็กและตัวพิมพ์ใหญ่)
	โดยปกติแล้วค่าตั้งต้นจะเป็น case insensitive หมายถึง ไม่มีการพิจารณาตัวพิมพ์เล็กหรือใหญ่
	ตัวอย่าง
	`#case`
	`int STATUS;`
	`void func() {`
	`  int status;`
	`  ...`
	`  STATUS = status; // Copy local status to global`
	`}`

#OPT n	กำหนดให้มีการ optimization (จัดการขนาดของไฟล์ หลังการคอมไพล์) มีลำดับตั้งแต่ 0-9 **ตัวอย่าง** `#opt 5`
#PRIORITY	กำหนดระดับความสำคัญของการเกิดอินเตอร์รัปต์ **ตัวอย่าง** `#priority rtcc,rb`

#ORG	การกำหนดจุดเริ่มต้นในการใช้งานพื้นที่หน่วยความจำรอม
	ตัวอย่าง
	`#ORG 0x1E00, 0x1FFF`
	`MyFunc() {`
	`  //This function located at 1E00`
	`}`
	
	`#ORG 0x1E00`
	`Anotherfunc(){`
	`  // This will be somewhere 1E00-1F00`
	`}`
	
	`#ORG 0x800, 0x820 {}`
	`  //Nothing will be at 800-820`
	
	`#ORG 0x1C00, 0x1C0F`
	`CHAR CONST ID[10]= {"123456789"};`
	`  //This ID will be at 1C00`
	`  //Note some extra code will`
	`  //proceed the 123456789`
	
	`#ORG 0x1F00, 0x1FF0`
	`void loader (){`
	`.`
	`.`
	`}`

ตาราง 3.1 ตารางแสดงรายละเอียดของพรีโปรเซสเซอร์ไดเร็กทีฟใน CCSC คอมไพเลอร์ (มีต่อ)

กำหนดอุปกรณ์	ความหมายและตัวอย่างการใช้งาน

#DEVICE CHIP	กำหนดเบอร์ของไมโครคอนโทรลเลอร์ที่ใช้งานหรือรายละเอียดของ DEVICE
	ตัวอย่าง
	`#device PIC16C74`
	`#device PIC16C67 *=16`
	`#device *=16 ICD=TRUE`
	`#device PIC16F877 *=16 ADC=10`

#FUSES options	กำหนดเงื่อนไขการคอนฟิกูเรชั่นของไมโครคอนโทรลเลอร์ ตัวอย่าง `#fuses HS,NOWDT`

ไลบรารีภายในคอมไพเลอร์	ความหมายและตัวอย่างการใช้งาน

#USE DELAY CLOCK	ใช้กำหนดความถี่ของไมโครคอนโทรลเลอร์ (เกี่ยวข้องกับการใช้งานฟังก์ชั่น delay_ms() และ delay_us() และ ไดเร็กตีฟ rs232())
	ตัวอย่าง
	`#use delay (clock=20000000)`
	`#use delay (clock=32000, RESTART_WDT)`

#USE FAST_IO	กำหนดให้การใช้งานพอร์ตอินพุตเอาต์พุตเข้าถึงรีจิสเตอร์ที่เกี่ยวข้องโดยตรง ผู้ใช้ต้องแน่ใจว่าได้กำหนดการใช้งานพอร์ตด้วยฟังก์ชั่น set_tris_X() แล้ว ตัวอย่าง `#use fast_io(A)`
#USE FIXED_IO	กำหนดพอร์ตที่ต้องการใช้งานเป็นอินพุตหรือเอาต์พุตอย่างใดอย่างหนึ่งตลอดการใช้งาน ตัวอย่าง `#use fixed_io(a_outputs=PIN_A2, PIN_A3)`

#USE I2C	กำหนดใช้งานไลบรารี I2C BUS ที่เกี่ยวข้องกับฟังก์ชั่น I2C_START, I2C_STOP, I2C_READ, I2C_WRITE และ I2C_POLL
	ตัวอย่าง
	`#use I2C(master, sda=PIN_B0, scl=PIN_B1)`
	`#use I2C(slave,sda=PIN_C4,scl=PIN_C3,address=0xa0,FORCE_HW)`

#USE RS232	กำหนดการใช้งานพอร์ตอนุกรม อัตราบอด และขาพอร์ตที่ใช้เป็นอินพุตเอาต์พุตอนุกรม ตัวอย่าง `#use rs232(baud=9600, xmit=PIN_A2,rcv=PIN_A3)`
#USE STANDARD_IO	ค่าเริ่มต้นสำหรับกำหนดการทำงานของพอร์ตตลอดเวลาที่ใช้งาน ตัวอย่าง `#use standard_io(A)`

ตาราง 3.1 ตารางแสดงรายละเอียดของพรีโปรเซสเซอร์ไดเร็กทีฟใน CCSC คอมไพเลอร์ (มีต่อ)

ควบคุมหน่วยความจำ	ความหมายและตัวอย่างการใช้งาน

#ASM
#ENDASM
	กำหนดจุดเริ่มต้นและสิ้นสุดของโค้ดภาษาแอสเซมบลี
	ตัวอย่าง
	`int find_parity (int data){`
	`  int count;`
	`#asm`
	`  movlw  0x8`
	`  movwf  count`
	`  movlw  0`
	`loop:`
	`  xorwf  data,w`
	`  rrf  data,f`
	`  decfsz  count,f`
	`  goto  loop`
	`  movwf  _return_`
	`#endasm`
	`}`

#BIT id=const.const
#BIT id = id.const
	กำหนดการใช้งานบิตข้อมูล
	ตัวอย่าง
	`#bit T0IF = 0xb.2`
	`...`
	`T0IF = 0;  // Clear Timer 0 interrupt flag`
	
	`int result;`
	`#bit result_odd = result.0`
	`...`
	`if (result_odd)`
	`...`

#BYTE id=const
#BYTE id=id
	กำหนดการใช้งานไบต์ข้อมูล
	ตัวอย่าง
	`#byte  status = 3`
	`#byte  b_port = 6`
	
	`struct  {`
	`  short int r_w;`
	`  short int c_d;`
	`  int unused : 2;`
	`  int data  : 4; } a_port;`
	`#byte a_port = 5`
	`...`
	`a_port.c_d = 1;`

#LOCATE id=const	จองหน่วยความจำเริ่มต้นที่ต้องการให้กับตัวแปรที่ได้กำหนดขึ้น
	ตัวอย่าง
	`// This will locate the float variable at 50-53`
	`// and C will not use this memory for other`
	`// variables automatically located.`
	`float x;`
	`#locate x=0x50`

#RESERVE	สงวนพื้นที่หน่วยความจำ ต้องกำหนดไดเร็กติฟ #DEVICE ก่อนการใช้งาน
	ตัวอย่าง
	`#DEVICE PIC16C74`
	`#RESERVE  0x60:0X6f`

ตาราง 3.1 ตารางแสดงรายละเอียดของพรีโปรเซสเซอร์ไดเร็กทีฟใน CCSC คอมไพเลอร์ (มีต่อ)

ควบคุมหน่วยความจำ	ความหมายและตัวอย่างการใช้งาน
=ROM	กำหนดให้เขียนข้อมูลไปที่หน่วยความจำ EEPROM ในตำแหน่งที่กำหนด **ตัวอย่าง** `#rom 0x2100={1,2,3,4,5,6,7,8}`

=ZERO_RAM	กำหนดให้รีจิสเตอร์ภายในมีค่าเป็นศูนย์ก่อนที่โปรแกรมจะเริ่มต้นทำงาน
	ตัวอย่าง
	`#zero_ram`
	`void main() {`
	`}`

กำหนดค่าเบื้องต้น	ความหมายและตัวอย่างการใช้งาน

__DATE__	พรีโปรเซสเซอร์ที่กำหนดให้คอมไพเลอร์แทน __DATE__ ด้วยวันที่ ที่คอมไพล์โปรแกรม
	ตัวอย่าง
	`printf("Software was compiled on ");`
	`printf(__DATE__);`

__DEVICE__	พรีโปรเซสเซอร์กำหนดเงื่อนไขการคอมไพล์ถ้ามีการกำหนด Device ที่เกี่ยวข้องกับ พรีโปรเซสเซอร์ __device__
	ตัวอย่าง
	`#if __device__== 71`
	`setup_port_a( ALL_DIGITAL );`
	`#endif`

__PCB__	พรีโปรเซสเซอร์กำหนดเงื่อนไขการคอมไพล์ถ้ามีการกำหนดการคอมไพล์ที่เกี่ยวข้องกับ พรีโปรเซสเซอร์ __pcb__
	ตัวอย่าง
	`#ifdef __pcb__`
	`#device PIC16c54`
	`#endif`

__PCM__	พรีโปรเซสเซอร์กำหนดเงื่อนไขการคอมไพล์ถ้ามีการกำหนดการคอมไพล์ที่เกี่ยวข้องกับ พรีโปรเซสเซอร์ __pcm__
	ตัวอย่าง
	`#ifdef __pcm__`
	`#device PIC16c71`
	`#endif`

__PCH__	พรีโปรเซสเซอร์กำหนดเงื่อนไขการคอมไพล์ถ้ามีการกำหนดการคอมไพล์ที่เกี่ยวข้องกับ พรีโปรเซสเซอร์ __pch__
	ตัวอย่าง
	`#ifdef __pch__`
	`#device PIC18C452`
	`#endif`

ตาราง 3.1 ตารางแสดงรายละเอียดของพรีโปรเซสเซอร์ไดเร็กทีฟใน CCSC คอมไพเลอร์

3.1.2 การประกาศ (Decralations)

เป็นการประกาศก่อนใช้งานตัวแปร และ ฟังก์ชั่น ต้องมีการประกาศขึ้นมาก่อน

3.1.3 การกำหนดค่า (Definitions)

สำหรับประกาศและจองหน่วยความจำให้กับตัวแปรหรือฟังก์ชั่น

3.1.4 นิพจน์ (Expressions)

คือการนำตัวถูกดำเนินการ (Operands) เช่น ตัวแปร, ค่าคงที่ หรือ ตัวเลข มากระทำกันผ่านทางตัวดำเนินการ(Operators) ด้วยเครื่องหมายดำเนินการภายใต้เงื่อนไขใดเงื่อนไขหนึ่ง

3.1.5 สเตทเมนต์ (Statements)

เป็นคำสั่งการทำงานตามความต้องการของผู้เขียนโปรแกรม

3.1.6 ฟังก์ชั่น (Functions)

เป็นส่วนประกอบของโปรแกรมที่กำหนดการทำงานอย่างใดอย่างหนึ่งจนเสร็จสิ้น เสมือนโปรแกรมย่อย โดยภายในฟังก์ชั่นจะประกอบไปด้วย การประกาศใช้งานทั้งตัวแปร นิพจน์ และคำสั่งการทำงาน

3.1.7 ฟังก์ชั่น Main() (Main Function)

จะต้องประการทุกครั้งที่เขียนโปรแกรมภาษาซี เพราะเป็นฟังก์ชั่นหลักที่เรียกฟังก์ชั่นอื่นมาใช้งาน

3.1.8 คีย์เวอร์ด (Keywords)

เป็นคำสงวนไว้สำหรับภาษาซี ห้ามนำมาใช้เป็นชื่อฟังก์ชั่น หรือตัวแปร เพราะไปซ้ำกับพรีโปรเซสเซอร์ไดเร็กทีฟหรือ คำสั่ง อาจทำให้เกิดข้อผิดพลาดในการคอมไพล์ได้ ได้แก่

auto double int struct break elae do

long switch case enum register typedef

char extern return union const float if

default goto sizeof volatile static while

3.1.9 คอมเมนต์ (Comments)

หรือหมายเหตุเพื่อให้ผู้อ่านเข้าใจเท่านั้น โดยจะใช้เครื่องหมายคอมเมนต์ "//" สำหรับคอมเมนต์บรรทัดเดียว และ "/*" สำหรับคอมเมนต์หลายบรรทัด(ใช้ทุกครั้งเมื่อคอมเมนต์ขึ้นบรรทัดใหม่) โดยข้อความใดๆที่อยู่หลังเครื่องหมายคอมเมนต์ จะไม่ถูกนำมาคอมไพล์

3.2 ตัวแปรและชนิด ของข้อมูล

3.2.1 ชนิดข้อมูล (Data Type)

เมื่อจะมีการประกาศและสร้างตัวแปรขึ้นมา จำเป็นต้องกำหนดชนิดของข้อมูลก่อน แต่ถ้าไม่มีการกำหนดชนิด จะถือเป็นข้อมูลแบบอัตโนมัติ (Auto) คือสามารถเปลี่ยนได้ตามคำสั่งภายในฟังก์ชั่นนั้นๆ ชนิดของข้อมูลแบ่งได้ตามตารางที่ 3.2

ชนิดข้อมูล	ขนาด	ค่าของข้อมูล
int1	ตัวเลข 1 บิต	0 หรือ 1
int8	ตัวเลข 8 บิต	0 ถึง 255
int16	ตัวเลข 16 บิต	0 ถึง 65,535
int32	ตัวเลข 32 บิต	0 ถึง 4,294,967,295
char	ตัวอักขระ 8 บิต	ตัวอักขระรหัสแอสกี้
float	ตัวเลขทศนิยม 32 บิต	$3.4x10^{-38}$ ถึง $3.4x10^{38}$
short	ตัวเลข 1 บิต	0 หรือ 1
int	ตัวเลขจำนวนเต็ม 8 บิต	0 ถึง 255
long	ตัวเลข 16 บิต	0 ถึง 65,535
void	ไม่กำหนด	-

ตาราง 3.2 ตารางแสดงรายละเอียดของชนิดของข้อมูล

3.2.2 ตัวแปร (Variables)

คือชื่อที่ผู้พัฒนาโปรแกรมประกาศขึ้นเพื่อเก็บข้อมูลชั่วคราวตามชนิดของข้อมูลดังข้อ 3.2.1 การประกาศตัวแปรก็คือการประกาศจองและใช้งานหน่วยความจำแรม ดังนั้นไม่ควรจะประกาศใช้ตัวแปรมากเกินความจำเป็น โดยตามกฎการตั้งชื่อตัวแปร ดังนี้

- ไม่ใช้ Keywords หรือคำที่ตรงกับคำสั่ง และ ฟังก์ชั่นภายใน (Built-in Function)
- ไม่ขึ้นต้นด้วยตัวเลข
- ไม่มีช่องว่างแทรกในชื่อ
- ไม่มีอักษรพิเศษเช่น ! @ # $ % ^ & * เป็นต้น
- อักษรพิมพ์เล็กกับพิมพ์ใหญ่จะไม่เหมือนกัน ถือเป็นคนละกรณี

3.3 ตัวดำเนินการ (Operators)

คือเครื่องหมายดำเนินการของนิพจน์ ได้แก่

3.3.1 เครื่องหมายดำเนินการทางคณิตศาสตร์ (Arithmetic & Unary Operators)

ได้แก่. +(บวกค่า), -(ลบค่า), *(คูณค่า), /(หารค่า), %(หารค่าแล้วเอาเฉพาะเศษ), -(ค่าติดลบใส่ข้างหน้า), ++(บวกค่า1), --(ลบค่า1)

3.3.2 เครื่องหมายดำเนินการสัมพันธ์ทางลอจิค (Relational & Logical Operators)

ได้แก่. <(มากกว่า), <=(มากกว่าหรือเท่ากับ), >(น้อยกว่า), >=(น้อยกว่าหรือเท่ากับ), ==(เท่ากับ), !=(ไม่เท่ากับ), &&(และ), ||(หรือ), !(ไม่) โดยถ้าข้อมูลใดเป็นจริง(True) จะมีผลลัพธ์ทางลอจิคเป็น "1" และข้อมูลใดเป็นเท็จ(Fault) จะมีผลลัพธ์ทางลอจิคเป็น "0"

เช่น a=9 , b=15

(a<b) // จริง ผลลัพท์ของ 1 หรือ True

(a>b) // เท็จ ผลลัพท์ของ 0 หรือ Fault

3.3.3 เครื่องหมายดำเนินการกำหนดค่าหรือให้ค่า (Assignment Operators)

ใช้กำหนดค่าของข้อมูลโดยใช้เครื่องหมาย "=" โดยจะนำค่าของข้อมูลด้านขวาของเครื่องหมาย กำหนดให้กับตัวแปร หรือค่าคงที่ทางด้านซ้าย

3.3.4 เครื่องหมายดำเนินการเงื่อนไข (Conditional Operators)

ใช้เครื่องหมาย "?:" เพื่อเปรียบเทียบทางลอจิค

เช่น a=9 , b=15

x = (a<b) ? 40:99 // จริง ผลลัพธ์ของ xจะเป็น 99

y = (a>b) ? 40:99 // เท็จ ผลลัพธ์ของ yจะเป็น 40

3.3.5 เครื่องหมายดำเนินการทางบิต (Bitwise Operators)

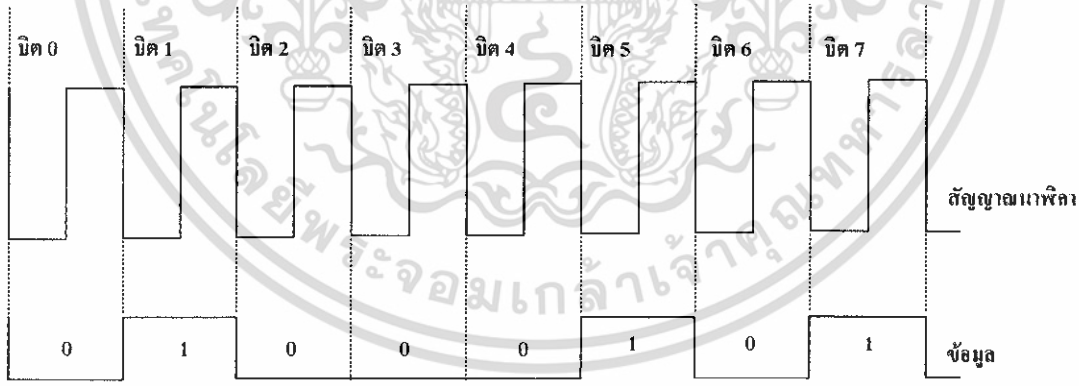
เป็นการนำข้อมูลเรียงเป็นไบนารี่ แล้วดำเนินการทางบิต ได้แก่. & (การแอนด์ค่าทางบิต), |(การออร์ค่าทางบิต), ^(การเอ็กคูลซีฟ-ออร์ค่าทางบิต), <<(เลื่อนบิตทางซ้าย 1 ตำแหน่ง), >>(เลื่อนบิตทางขวา 1 ตำแหน่ง), ~(กลับค่าบิต)

บทที่ 4
การติดต่อกับอุปกรณ์ภายนอกของไมโครคอนโทรลเลอร์ PIC16F87x

4.1 การอินเตอร์เฟส (Interface)กับพอร์ตอนุกรม (Serial Port)

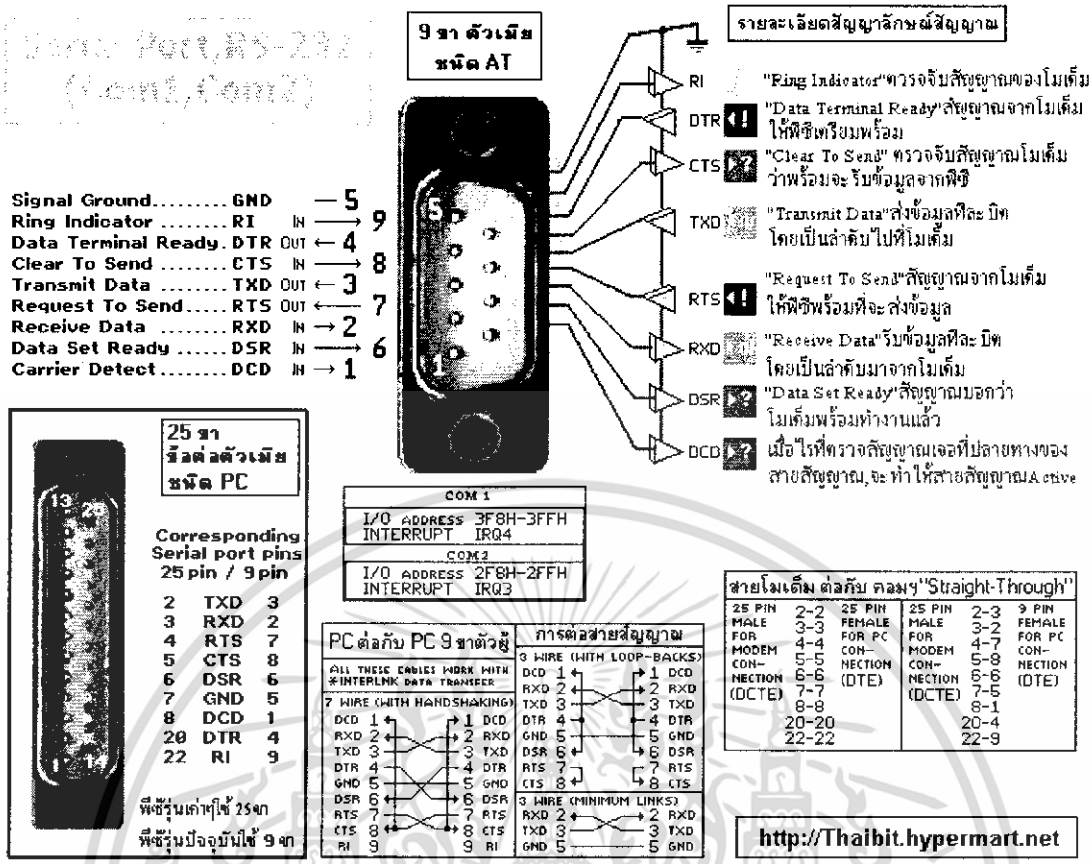
การอินเตอร์เฟส คือ การทำงานติดต่อร่วมกันระหว่างซีพียูกับอุปกรณ์อื่นๆกับการโอนถ่ายข้อมูลระหว่างอุปกรณ์ต่างๆ นอกเหนือจากจะต้องทำงานติดต่อกับ RAM,ROM แล้วยังต้องมีการติดตอกับอุปกรณ์ภายนอกที่มีการส่งข้อมูลอินพุท,เอาท์พุทอีกทางหนึ่ง ซึ่งเป็นการเพิ่มประสิทธิภาพให้ระบบสมบูรณ์ ในระบบต่างของอุปกรณ์อิเล็กทรอนิกส์ จะทำงานต่อเนื่องเป็นลูกโซ่ ดังเช่น การส่งรับข้อมูลจากซีพียูไปยังส่วนอื่นๆ เป็นต้น ไมโครคอนโทรลเลอร์ PICหลายๆเบอร์ ได้จัดเตรียมโมดูลภายในที่ใช้สำหรับติดต่อกับพอร์ตอนุกรมไว้ให้ที่เรียกว่า USART(Universal Synchronous Asynchronous Reciever Transmitter) ซึ่งทำให้สามารถรับส่งสื่อสารข้อมูลแบบซิงโครนัส และ อะซิงโครนัส ได้ ในที่นี้จะใช้การรับส่งแบบอะซิงโครนัสเพียงอย่างเดียว

การสื่อสารแบบซิงโครนัส จะใช้สัญญาณนาฬิกาเป็นตัวควบคุมการรับส่งสัญญาณแสดงดังรูปที่ 4.1 โดยจะใช้สายเส้นหนึ่งเป็นสายสัญญาณนาฬิกา อีกเส้นหนึ่งเป็นสายกราวนด์ ในขณะที่ การสื่อสารแบบอะซิงโครนัส จะใช้สายสัญญาณเพียงเส้นเดียว แต่จะใช้รูปแบบการส่ง (Bit Pattern) เป็นตัวกำหนดว่าส่วนไหนเป็นส่วนเริ่ม,ส่วนปิดท้าย,ส่วนตรวจสอบ และส่วนข้อมูล ซึ่งทั้งภาครับและภาคส่งจะต้องมีอุปกรณ์ที่เรียกว่า USRT (Universal Asynchronous Reciever Transmitter) คอยควบคุมการรับส่งข้อมูล



รูป 4.1 รับส่งข้อมูลแบบซิงโครนัส

มาตรฐานการสื่อสารโดยพอร์ตอนุกรมปัจจุบันจะใช้มาตรฐาน RS-232C ซึ่งถูกออกแบบมาเพื่อให้อุปกรณ์ต่อพ่วงจากผู้ผลิตสามารถทำงานร่วมกันได้ มีรายละเอียดดังรูปที่ 4.2 โดยข้อมูลแบบอะซิงโครนัสจะมีอยู่ 4ส่วนแสดงดังรูปที่ 4.3



รูป 4.2 พอร์ตอนุกรมปัจจุบันตามมาตรฐาน RS-232C

รูป 4.3 การส่งข้อมูลแบบอะซิงโครนัส

1) บิตเริ่มต้น (Start Bit) มีขนาด 1บิต เพื่อเตือนอุปกรณ์ฝ่ายรับว่าข้อมูลกำลังมาถึง

2) บิตข้อมูล (Data Characters) ขนาด 7-8 บิต เป็นส่วนของข้อมูล

3) บิตพาริตี้(Parity Bit) ตรวจสอบความถูกต้องโดยกำหนดให้ใช้หลักการเดียวกันทั้งฝ่ายรับและฝ่ายส่ง ซึ่งมีได้หลายแบบ เช่น

- บิตพาริตี้คู่ (Even Parity) จำนวนบิตที่เป็นเลข 1 ในข้อมูล จะเป็นจำนวนคู่ ดังนั้น บิตพาริตี้นี้มีค่า 0
- บิตพาริตี้คี่ (Odd Parity) จำนวนบิตที่เป็นเลข 1 ในข้อมูล จะเป็นจำนวนคี่ ดังนั้น บิตพาริตี้นี้มีค่า 1
- ไม่มีบิตพาริตี้ (None) ไม่มีการตรวจสอบ

4) บิตปิดท้าย (Stop Bit) ปิดท้ายเพื่อสิ้นสุดการรับส่ง

ซึ่งข้อมูลทั้งหมดจะต้องมีอัตราการรับส่งที่ตรงกันทั้งฝ่ายรับและฝ่ายส่ง คือความเร็วในการรับส่งที่เรียกว่า บอดเรต (Baud Rate) หน่วย บิต/วินาที (Bits Per Second :Bps) เพื่อให้การรับส่งครบถ้วนสมบูรณ์

โดยจะต้องกำหนดโดยส่วนของโปรแกรมพรีโปรเซสเซอร์ไดเร็กทีฟดังนี้

```
#define TXD     PIN_C6          // Define serial transmission data to PC = RC6
#define RXD     PIN_C7          // Define serial recieve data from PC = RC7
#use RS232 (baud=1200, xmit=TXD, rcv=RXD, parity=N) //Baud Rate 1200 Bps,
External Transmission Pin = RC6, Reciever Pin = RC7, None Parity
```

ในการกำหนดค่าบอดเรต มักจะใช้ตามค่าบอดเรตมาตรฐานRS-232C คือ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 Bps

แต่จะมีค่าความผิดพลาดต่างกัน สำหรับไมโครคอนโทรลเลอร์ PIC

ซึ่งสามารถคำนวณได้จาก สมการที่ 4.1

$$BR = fo/64(X+1) \qquad (4.1)$$

BR คือ ค่าบอดเรตที่กำหนด (Bps)

fo คือ ค่าความถี่นาฬิกาที่ไมโครคอนโทรลเลอร์ทำงาน (Hz)

X คือ ค่าของรีจิสเตอร์ ต้องเป็นจำนวนเต็มเท่านั้น

เช่น ใช้ความถี่นาฬิกา 20MHz แต่กำหนดค่าบอดเรต 2400 เมื่อแทนในสมการ 4.1 จะได้ว่า

2400=20000000/64(X+1)

จะได้ค่า X = 129.2083333 แต่ต้องเป็นเลขจำนวนเต็ม จึงให้ X=129 เมื่อนำกลับไปแทนในสมการที่ 4.1 จะหาบอดเรตที่ทำงานจริงได้ คือ BR=20000000/64(129+1) ได้ค่า BR=2403.846 คือค่าจริงที่ทำงาน จะเห็นว่ามีความผิดพลาด (2403.846-2400)*100/2400=0.16%

โดยอัตราบอดเรตที่ใช้ จะสัมพันธ์กับความถี่นาฬิกาดังตาราง 4.1

บอดเรต (kbps)	f_{osc} = 20MHz			f_{osc} = 10MHz			f_{osc} = 4MHz			f_{osc} = 3.579545MHz			f_{osc} = 32.768kHz		
	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG
0.3	N/A	-	-	N/A	-	-	0.3005	-0.17	207	0.301	+0.23	185	0.256	-14.67	1
1.2	1.221	+1.73	255	1.202	+0.16	129	1.202	+1.67	51	1.190	-0.83	46	N/A	-	-
2.4	2.404	+0.16	129	2.404	+0.16	64	2.404	+1.67	25	2.432	+1.32	22	N/A		-
9.6	9.469	-1.36	32	9.766	+1.73	15	10.42	+8.54	5	9.322	-2.90	5	N/A	-	-
19.2	19.53	+1.73	15	19.53	+1.73	7	20.83	+8.50	2	18.64	-2.90	2	N/A	-	-
76.8	78.13	+1.73	3	78.13	+1.73	1	N/A	-	-	N/A	-	-	N/A	-	-
96	104.2	+8.51	2	N/A	-	-	N/A	-	-	N/A	-	-	N/A	-	-
300	312.5	+4.17	1	N/A	-		N/A	-	-	N/A	-	-	N/A	-	-
500	N/A	-	-	N/A	-	-	N/A	-	-	N/A	-	-	N/A	-	-
1250	1250		0	1256.3		0	62.500	-	0	55.93	-	0	0.512	-	0
ต่ำสุด	1.221	-	255	0.6104	-	255	0.244	-	255	0.2185	-	255	0.0020	-	255

ตาราง 4.1ก ตารางแสดงรายละเอียดของรีจิสเตอร์, ค่าบอดเรตจริง และค่าเปอร์เซ็นต์ความผิดพลาด เมื่อกำหนดให้โมดูล USART ทำงานในโหมดอะซิงโครนัสความเร็วต่ำ

บอดเรต (kbps)	f_{osc} = 20MHz			f_{osc} = 10MHz			f_{osc} = 4MHz			f_{osc} = 3.579545MHz		
	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG	บอดเรตที่เกิดจริง	ผิดพลาด (%)	ค่าของ SPBRG
9.6	9.615	+0.16	129	9.615	+0.16	64	9.615	+0.16	25	9.727	+1.32	22
19.2	19.230	+0.16	64	18.939	-1.36	32	19.230	+0.16	12	18.643	-2.90	11
38.4	37.878	-1.36	32	39.062	+1.7	15	41.667	+8.51	5	37.286	-2.90	5
57.6	56.818	-1.36	21	56.818	-1.36	10	62.5	+8.51	3	55.930	-2.90	3
115.2	113.636	-1.36	10	125	+8.51	4	125.0	+8.51	1	111.860	-2.90	1
250	250	0	4	312.50	+28.6	1	250	0	0	223.721	-10.51	0
625	625	0	1	625	0	0	N/A	-	-	N/A	-	-
1250	1250	0	0	N/A	-	-	N/A	-	-	N/A	-	-

ตาราง 4.1ข ตารางแสดงรายละเอียดของรีจิสเตอร์, ค่าบอดเรตจริง และค่าเปอร์เซ็นต์ความผิดพลาด เมื่อกำหนดให้โมดูล USART ทำงานในโหมดอะซิงโครนัสความเร็วสูง

4.2 การใช้อินเตอร์รัพท์ (Interrupt)

อินเตอร์รัพท์ คือ การขัดจังหวะการทำงานของซีพียู หรือโปรแกรมที่กำลังทำงานอยู่ เพื่อมาทำงานในส่วนของโปรแกรมบริการอินเตอร์รัพท์ที่ได้กำหนดล่วงหน้าไว้แล้ว ซึ่งจะช่วยให้ประหยัดเวลา และโปรแกรมง่ายขึ้น เนื่องจากไม่ต้องคอยเช็คเงื่อนไขบางอย่างอยู่ตลอดเวลาที่เรียกว่าโพลลิ่ง(Polling) ทำให้ไมโครคอนโทรลเลอร์สามารถทำงานได้เร็วขึ้น สามารถทำงานอย่างอื่นๆได้ โดยให้บริการอินเตอร์รัพท์เป็นตัวเช็คเงื่อนไขแทน ในไมโครคอนโทรลเลอร์ PIC16F87x จะมีอินเตอร์รัพท์อยู่หลายประเภท ได้แก่

1) อินเตอร์รัพท์จาก ไทเมอร์โอเวอร์โฟลว์

2) อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณอะนาลอก เป็น ดิจิตอล

3) อินเตอร์รัพท์จาก โมดูล CCP1 และ CCP2

4) อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณของพอร์ต B และ C

5) อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณจากการส่งข้อมูลอนุกรม RS-232 (USART)

ซึ่งจะกล่าวถึงแบบที่ 4 และ 5 เท่านั้น คือ อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณของพอร์ต B (RB0) หรือ อินเตอร์รัพท์จากภายนอก (External Interrupt) และ อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณจากการส่งข้อมูลอนุกรม RS-232 (USART)

4.2.1 อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณของพอร์ต B (RB0) หรือ อินเตอร์รัพท์จากภายนอก (External Interrupt)

เป็นอินเตอร์รัพท์ที่เกิดขึ้นเนื่องจากมีการเปลี่ยนแปลงสัญญาณที่ขา RB0 เช่นการเปลี่ยนระดับจากไฟเลี้ยงที่ขาRB0 เป็นกราวนด์ อินเตอร์รัพท์จะทำงาน ทั้งนี้ในโปรแกรมจะต้องเอ็นเนเบิลความสามารถนี้ไว้ โดยกำหนดในโปรแกรมดังนี้

```
enable_interrupts (global);          // Use interrupt service
enable_interrupts (int_ext);         // Use external interrupt
```

```
#int_ext
void AAA (void) {          // Interrupt program
                    ...;
                    ...;
                    }
```

ซึ่งเมื่อเกิดอินเตอร์รัพท์ โปรแกรมจะหยุดจากโปรแกรมหลัก มาทำงาใน โปรแกรม AAA จนหมด จึงกลับไปทำงานเดิมต่อ บางครั้งอาจจะหลุดจากลูปการทำงานเดิมได้ เนื่องจากเป็นคำสั่งภายในโปรแกรมอินเตอร์รัพท์

กรณีถ้าต้องการยกเลิกอินเตอร์รัพท์ ให้เปลี่ยนเป็น

```
disable_interrupts (global);
disble_interrupts (int_ext);
```

4.2.2 อินเตอร์รัพท์จาก การเปลี่ยนแปลงสัญญาณจากการส่งข้อมูลอนุกรม RS-232 (USART)

เพื่อที่ไม่ต้องคอยตรวจสอบการกดคีย์หรือการส่งค่าจากอนุกรม ในการทำงานจะใช้หลักการเดียวกับอินเตอร์รัพท์จากภายนอก ดดยเมื่อใดก็ตามที่มีสัญญาณ หรือข้อมูลเข้ามาในบัฟเฟอร์ (Buffer) ของขารับข้อมูลจากอนุกรม อินเตอร์รัพท์จะทำงาน จนกว่าข้อมูลนั้นจะหมดไป ดังนั้น ในการปฏิบัติจะต้องมีการสร้างตัวแปรขึ้นตัวหนึ่ง ใช้รับข้อมูลที่ส่งเข้ามา เพื่อเคลียร์ค่าในบัฟเฟอร์ไม่ให้อินเตอร์รัพท์ทำงานค้าง โดยจะกำหนดในโปรแกรมดังนี้

```
enable_interrupts (global); // Use interrupt service
enable_interrupts (int_rda); // Use serial interrupt
#int_rda
void AAA (void) { // Interrupt program
                    ...;
                    ...;
                    }
```

ซึ่งเมื่อเกิดอินเตอร์รัพท์ โปรแกรมจะหยุดจากโปรแกรมหลัก มาทำงานใน โปรแกรม AAA จนหมด จึงกลับไปทำงานเดิมต่อ บางครั้งอาจจะหลุดจากลูปการทำงานเดิมได้ เนื่องจากเป็นคำสั่งภายในโปรแกรมอินเตอร์รัพท์

กรณีถ้าต้องการยกเลิกอินเตอร์รัพท์ ให้เปลี่ยนเป็น

```
disable_interrupts (global);
disble_interrupts (int_rda);
```

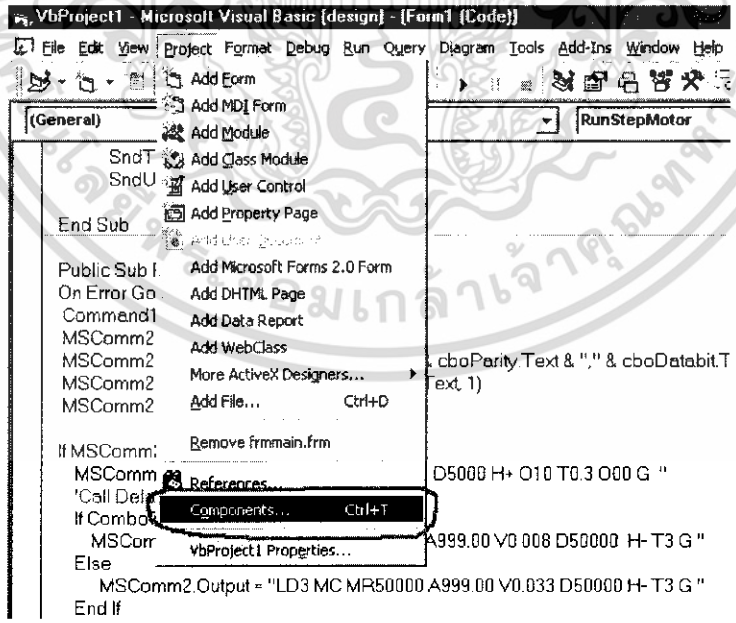
บทที่ 5

Visual Basic to Serial Port

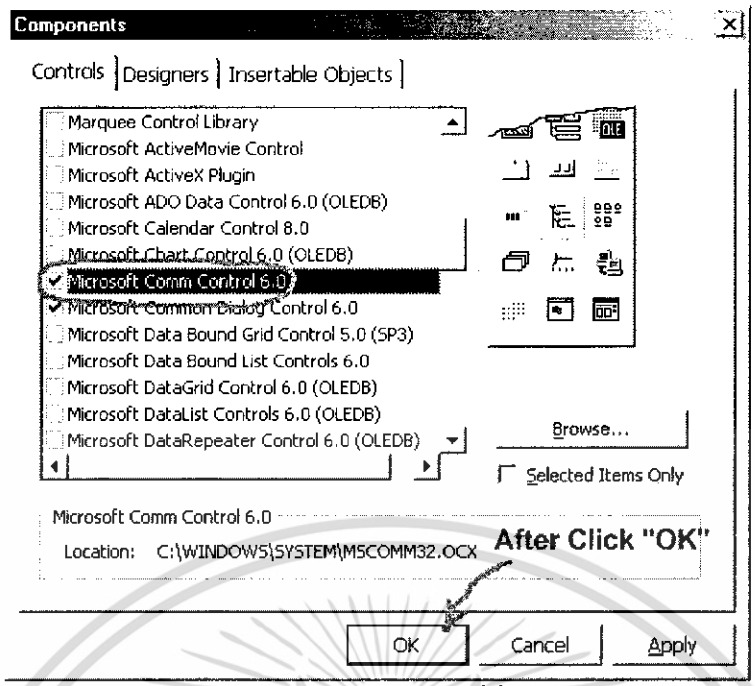
Visual Basic เป็นโปรแกรมสำเร็จรูปที่ใช้สำหรับพัฒนาโปรแกรมบน windowsที่ใช้งานง่าย และกว้างขวาง เนื่องจากใช้เทคโนโลยีด้าน Visualize เข้ามาประกอบในจอภาพ ทำให้ผู้ใช้สามารถจัดวางตัวรูปแบบโปรแกรมได้โดยง่าย ต่างจากภาษาอื่นๆ ตรงที่ สามารถเห็นลักษณะภายนอกของตัวโปรแกรมที่สมบูรณ์ ก่อนโดยการจัดวางวัตถุ(Object) เช่น ปุ่มกดต่างๆ, ข้อความ เป็นต้น ตามความพอใจ จากนั้นค่อยเขียนภาษาควบคุม Object อีกที ทำให้งายต่อผู้ใช้ในด้านการควบคุมและความสวยงาม ในขณะที่ภาษาอื่นๆจะต้องเข้าใจและเขียนตัวภาษาออกมาก่อน ซึ่งการพัฒนาจะทำได้ค่อนข้างยากโดยต้องเข้าใจในตัวภาษามากพอสมควร

บทนี้จะกล่าวถึงเฉพาะการใช้งานสำหรับการต่ออุปกรณ์ภายนอกนั่นคือ พอร์ตอนุกรมที่ใช้ในโปรเจ็คนี้ เท่านั้น ในส่วนของรายละเอียดการเขียนโปรแกรม สีสัน รวมถึงเทคนิคต่างๆจะขอไม่กล่าวถึงในณ.ที่นี้

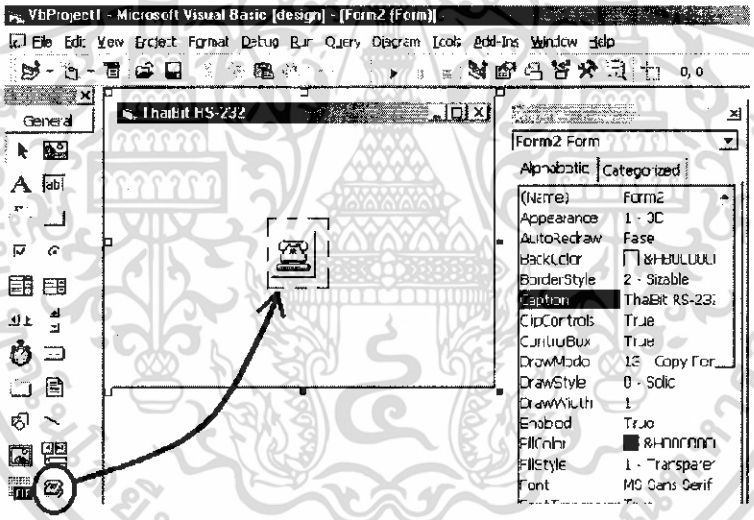
การติดต่อกับพอร์ตอนุกรมสามารถทำได้โดยใช้ VB Control ที่ชื่อว่า MSComm โดยที่กำหนดใน Custom Control เข้าไปที่ เมนู Project --> Companents แล้วเลือกที่ช่อง MSComm ก็จะปรากฏ เป็นรูปไอคอนโทรศัพท์สีเหลือง ให้คลิกที่ไอคอนลากนำมาไว้บน Form ใน Project ของโปรแกรมเพื่อที่สามารถเขียนคำสั่งติดต่อกับพอร์ตอนุกรม RS-232 ได้ แสดงดังรูป5.1



รูป 5.1 แสดงการเรียกใช้งาน VB Control ที่ชื่อว่า MSComm (มีต่อ)



รูป 5.1 แสดงการเรียกใช้งาน VB Control ที่ชื่อว่า MSComm (มีต่อ)



รูป 5.1 แสดงการเรียกใช้งาน VB Control ที่ชื่อว่า MSComm

5.1 องค์ประกอบในการใช้ MSComm

5.1.1 การตั้งค่าติดต่อกับพอร์ต

- ComPort คือ เราต้องกำหนดหมายเลข Port ที่ใช้ต่อRS-232 (Com1,Com2)รายละเอียดดูในเมนูด้านซ้าย Serial Port Detail

- Setting คือ เราต้องกำหนด อัตราBaud, Parity, Data(จำนวนบิต), Stop เช่น 1200,n,8,1 เป็นต้น

- HandShaking คือ เราจะกำหนดได้ 4 แบบ

1.comNone 2.comXonXoff 3. comRTS 4.comTRSXonXoff

5.1.2 การใช้ Buffer ในการรับส่งข้อมูล

-InBuffersize คือ การกำหนด Buffer ในการรับข้อมูลเข้ามา

-OutBuffersize คือ การกำหนด Buffer ในการส่งข้อมูลออกไป

- Rthreshold คือ การที่เรากำหนดการเกิด Event-driven ในการรับข้อมูลเข้ามา

- Sthreshold คือ การที่เรากำหนดการเกิด Event-driven ในการรับข้อมูลออกไป

- Inputlen คือ จำนวนของข้อมูลที่ไปอ่านใน Buffer รับข้อมูล

- EOFEnable คือ การที่บอกว่าสิ้นสุดของไฟล์(EOF) End of File

5.1.3 ด้านฮาร์ดแวร์

- ParityReplace คือ ค่าของคาเลกเตอร์ที่จะแทนในเมื่อเกิด Parity Error

- NullDiscard คือ การกำหนดให้รับหรือไม่รับ NULL CHARACTER

- RTSEnable คือ ทำให้มีสัญญาณ RTS (Request To Send)

- DTSEnable คือ ทำให้มีสัญญาณ DTR(Data Terminal Ready)

5.2 การกำหนดคุณสมบัติของ MSComm Control ให้สามารถติดต่อกับพอร์ต

5.2.1 Property ชื่อ CommPort

คือ เลือกคอมพอร์ตที่เราจะต่อใช้งาน

เช่น. MSComm1.CommPort=1 ในที่นี้เลือกจะใช้ Com1อยู่ที่ด้านหลังเครื่องคอม

5.2.2 Property ชื่อ Settings

คือ การตั้งค่าของการรับส่งข้อมูล ซึ่งจะต้องรู้ด้วยว่าอัตราบอด ของอุปกรณ์ที่จะติดต่อด้วยเป็นเท่าไร โดยมีรายละเอียดการใส่ต่างๆค่าดังนี้

MSComm1.Settings = "Baud Rate(อัตราการรับส่งข้อมูล), Parity(ถ้าไม่ใช้ใส่ N), จำนวนบิตข้อมูล, บิตปิดท้าย"

เช่น. MSComm1.Settings="2400,N,8,1"

5.2.3 Property ชื่อ InputLen

คือ กำหนดขนาดขณะที่มีข้อมูลเข้ามาให้ไปอ่านข้อมูลทั้งหมดที่อยู่ในบัฟเฟอร์

เช่น. MSComm1.InputLen=1

5.2.4 Property ชื่อ PortOpen

คือ จะเปิดให้พอร์ตใช้งานหรือไม่ ถ้าเปิด =True ถ้าปิด =False

เช่น. MSComm1.PortOpen=True

5.2.5 Property ชื่อ Rthreshold

คือ ทำให้เกิดการกระตุ้นด้วย Event-driven เมื่อมีข้อมูลในบัฟเฟอร์รับข้อมูลมันทำให้เกิด CommEvent ใน OnComm Event คล้ายๆกับการเกิดอินเตอร์รัพท์ในไมโครคอนโทรลเลอร์

เช่น. MSComm1.Rthreshold =1

นำรายละเอียดที่กล่าวมา เขียนใน โพรซีเดอร์ หรือจะสร้าง Sub ขึ้นใหม่ในกรณีที่จะเรียกใช้ภายหลัง ดังนี้

```
Private Sub ...()
    MSComm1.Settings="2400,N,8,1"
    MSComm1.CommPort=1
    MSComm1.InputLen=1
    MSComm1.PortOpen=True
    MSComm1.Rthreshold =1
End Sub
```

5.3 วิธีของการรับส่งข้อมูลจาก Serial Port

เมื่อกำหนดค่าเริ่มต้นให้กับคอมพอร์ตและเปิดใช้การรับและส่งของพอร์ต RS-232 ก็จะสามารถจะรับและส่งข้อมูลทางพอร์ตได้ โดยใช้ Property ดังนี้

Output = จะเป็นการส่งข้อมูลไปที่พอร์ต

Input = เป็นส่วนของการรับข้อมูลจากพอร์ต แต่ในส่วนนี้จะต้องนำคำสั่งไปเขียนที่ Event Property OnComm จะอยู่ใน Sub MSComm_OnComm ซึ่ง จะอ่านข้อมูลเข้ามาจากทางพอร์ต RS232

5.4 ปัญหา หรือ ข้อผิดพลาดที่อาจเกิดขึ้นใน MSComm

แสดงดังตารางที่ 5.1

ชื่อ Property	**คำอธิบาย**
ComEventBreak	การได้รับสัญญาณเบรก
ComEventCDTO	เมื่อเกิดไทม์เอาต์ ขณะที่กำลังคอยสัญญาณ CD(Carrier Detect)
ComEventCTSTO	เมื่อเกิดไทม์เอาต์ ขณะที่กำลังคอยสัญญาณ CTS(Carrier To Send)
ComEventDSRTO	เมื่อเกิดไทม์เอาต์ ขณะที่กำลังคอยสัญญาณ DSR (Data Set Ready)
ComEventFrame	การที่เกิดความผิดพลาดทางเฟรม เป็นลักษณะที่ไม่พบบิตจบตามที่ควรจะเป็น

ตาราง 5.1ก ตาราง CommEvent เกี่ยวกับการเกิดสถานะเมื่อเกิดการผิดพลาดในการสื่อสาร

ComEventOverrun	การที่เกิดความผิดพลาดโอเวอร์รัน เป็นลักษณะที่รับข้อมูลไม่ทันในการประมวลผล
ComEventRxOver	บัฟเฟอร์ที่รับข้อมูลเกิดโอเวอร์โฟวล์ ก็คือรับตัวอักษรหลังจากการรับ EOF Char
ComEventRxParIty	การที่เกิดความผิดพลาดทางพาริตี้ เป็นลักษณะที่ตัวอักษรที่รับได้มีพาริตี้ไม่ถูกต้อง
ComEventTxFull	ตัวบัฟเฟอร์ที่ส่งข้อมูลเต็ม
ComEventDCB	การที่เกิดความผิดผลาดขึ้นโดยไม่ได้คาดถึง

ตาราง 5.1ก ตาราง CommEvent เกี่ยวกับการเกิดสถานะเมื่อเกิดการผิดพลาดในการสื่อสาร (ต่อ)

ชื่อ Property	**คำอธิบาย**
ComEvCD	CD(CarrIrt Detect) เมื่อเปลี่ยนซึ่งคือสายของสัญญาณ ReceIve LIne SIgnal Detect(RLSD)
ComEvCTS	RCTS(CarrIer To Send)เมื่อมีการเปลี่ยนสถานะเกิดขึ้น
ComEvDSR	DSR(Data Set Ready) เมื่อมีการเปลี่ยนสถานะเกิดขึ้น
ComEvRIng	เมื่อตรวจจับสัญญาณ Ring IndIcator ได้
ComEvReceIve	เมื่อได้รับข้อมูลเก็บลงใน InputBuffer
ComEvSend	เมื่อส่งข้อมูลออกจาก OutputBuffer
ComEvEof	เมื่อพบอักขระ EOF(End Of FIle)

ตาราง 5.1ข ตาราง CommEvent เกี่ยวกับการเกิดสถานะเมื่อเกิดการผิดพลาดในการสื่อสาร

ชื่อ Property	**คำอธิบาย**
ComNone	ไม่ใช้ให้ตรวจสอบแฮนเช็ก
ComXonZXoff	ให้มีการตรวจสอบแฮนเช็ก ในแบบ Xon/Xoff

ตาราง 5.1ค ตาราง Handshake Property

ComRTS	ให้มีการตรวจสอบแฮนเช็ก ผ่านขา RTS และ CTS
ComRTSXOnXoff	กำหนดให้มีการตรวจทั้ง2แบบคือ RTS-CTS และXOn/Xo

ตาราง 5.1ค ตาราง Handshake Property (ต่อ)

ชื่อ Properties	**คำอธิบาย**
ComInputModeText	คุณสมบัติในการรับข้อมูลมาเป็นแบบข้อความ ปกติจะเป็นค่านี้อยู่แล้ว
ComInputModeBinary	คุณสมบัติในการรับข้อมูลมาเป็นแบบไบนารี่หรือเลขฐานสองนั้นเอง

ตาราง 5.1ง ตาราง Input Mode Property

ชื่อ Property	**คำอธิบาย**
Break	ในการที่เรากำหนดหรือเคลียร์สัญญาณเบรก
CDHoldIng	ตรวจสอบสัญญาณ Carrier Detect(CD)ดูว่ายังคงมีสถานะอยู่หรือเปล่า
CDTImeout	การกำหนดค่าหรือว่าให้ค่าของเวลา(หน่วย mmSec) ที่รอสัญญาณ Carrier Detect
CommEvent	จะให้ผลของการเกิด Event ของ Communucation
CommID	จะให้ผลของการเฮดเดิลของ Communucation ที่เปิดใช้อยู่
CommPort	การกำหนดหรือว่าอ้างอิงของหมายเลขคอมพอร์ต ที่เปิดใช้อยู่ เช่น Com1=1,Com2=2
CTSHoldIng	เป็นการตรวจสอบสัญญาณของ Clear To Send ว่ายังคงมีสถานะอยู่หรือเปล่า
CTSTImeout	การกำหนดค่าหรือว่าให้ค่าของเวลา(หน่วย mmSec) ที่รอสัญญาณ Data Set Ready
DSRHoldIng	เป็นการตรวจสอบสัญญาณของ Data Set Ready ว่ายังคงมีสถานะอยู่หรือเปล่า
DSRTimeout	การกำหนดค่าหรือว่าให้ค่าของเวลา(หน่วย mmSec) ที่รอสัญญาณ Clear To Send
DTREnable	ให้อีนาเบิล สายของสัญญาณ Data TermInal Ready(DTR)

ตาราง 5.1จ ตาราง MSComm Control Property

HandshakIng	กำหนดการแฮนเช็คทางฮาร์ดแวร์ เพื่อที่คอยตรวจสอบการรับส่งข้อมูล
InBufferCount	ให้ค่าของจำนวนข้อมูลที่อยู่ภายในบัฟเฟอร์รับข้อมูล
InBufferSIze	กำหนดหรือว่าให้ค่าของขนาดในบัฟเฟอร์รับข้อมูล
Input	เป็นการให้ค่าหรือว่าเคลื่อนย้ายข้อมูลจากบัฟเฟอร์รับข้อมูล
InoputLen	การกำหนดหรือว่าให้ของจำนวนข้อมูลที่นำมาจากบัฟเฟอร์รับข้อมูล
Interval	เป็นการกำหนดอัตราความเร็วของการใช้งานในโหมดโพลลิ่ง
NullDIscard	เป็นการกำหนดให้มีการรับ Null Character เก็บลงในบัฟเฟอร์รับข้อมูล
OutBufferCount	เป็นจำนวนข้อมูลที่คอยอยู่ในบัฟเฟอร์ส่งข้อมูล
OutBufferSIze	การกำหนดหรือว่าให้ค่าขนาดของบัฟเฟอร์ส่งข้อมูล
Output	เป็นการส่งข้อมูลให้กับบัฟเฟอร์ส่งข้อมูลเพื่อทำการส่งข้อมูลออก
ParItyReplace	เป็นการกำหนดให้ส่งอักขระที่กำหนดนี้แทนหากเกิดการผิดพลาดในข้อมูล
PortOpen	เป็นการกำหนดหรือว่าให้ค่าของสถานะพอร์ตว่าเปิดหรือปิดอยู่
Rthrehold	การกำหนดหรือว่าให้ค่าของจำนวนข้อมูลที่เก็บลงในบัฟเฟอร์รับข้อมูลก่อนการเกิด CommEvent ในการรับข้อมูล
RTSEnable	ให้อีนาเบิล สัญญาณ Request To Send(RTS)
SettIngs	เป็นการกำหนดอัตราบอด พาริตี้ ข้อมูล บิตหยุด
Sthreshold	การกำหนดหรือว่าให้ผลของจำนวนข้อมูลที่เก็บลงในของบัฟเฟอร์ส่งข้อมูลก่อนการเกิด CommEvent ในการที่ส่งข้อมูล

ตาราง 5.1จ ตาราง MSComm Control Property (ต่อ)

ชื่อ Property	ค่าตัวเลข	คำอธิบาย
ComInvalidPrppertyValue	380	ตั้งค่าไม่ถูกต้อง
ComSetNotSupported	383	กำหนดค่าที่ตั้งไว้สามารถอ่านได้อย่างเดียว เขียนหรือเปลี่ยนข้อมูลไม่ได้
ComGetNotSupported	394	กำหนดค่าที่รับไว้สามารถอ่านได้อย่างเดียว เขียนหรือเปลี่ยนข้อมูลไม่ได้
ComPortOpen	8000	จะอ่านค่าไม่ได้ในขณะที่ Port นั้นยังถูกเปิดใช้อยู่
ComPortOpen	8001	ค่าของเวลาที่หาออกมาได้ต้องมีค่ามากกว่าศูนย์
ComPortOpen	8002	กำหนดหมายเลข Port ไม่ถูกต้อง
ComPortOpen	8003	ผลลัพธ์ของข้อมูลจะเกิดในขณะที่มีการทำงาน
ComPortOpen	8004	Port นั้นจะสามารถอ่านค่าได้ในขณะที่มีการทำงานเท่านั้น
ComPortAlreadyOpen	8005	Port ได้ถูกเปิดไว้เรียบร้อยแล้ว
ComPortAlreadyOpen	8006	อุปกรณ์เกิดความผิดพลาดหรือไม่สามารถรองรับค่าได้
ComPortAlreadyOpen	8007	อุปกรณ์ไม่ยอมรับค่าที่ Baud Rate ถูกตั้งเอาไว้
ComPortAlreadyOpen	8008	ขนานของข้อมูลผิดพลาด
ComPortAlreadyOpen	8009	ค่าของตัวแปรที่แสดงอยู่ผิดพลาด
ComPortAlreadyOpen	8010	อุปกรณ์ภายนอก(Hardware)ยังไม่พร้อมที่จะทำงาน
ComPortAlreadyOpen	8011	ฟังก์ชั่นไม่สามารถกำหนดแถวข้อมูลได้
ComNoOpen	8012	Com Port ยังไม่พร้อมที่จะถูกเปิดใช้งาน
ComNoOpen	8013	Com Port พร้อมที่จะถูกเปิดใช้งาน

ตาราง 5.1ฉ ตาราง รายละเอียดที่บ่งชี้ถึงความผิดพลาดในการใช้ MS Comm Control

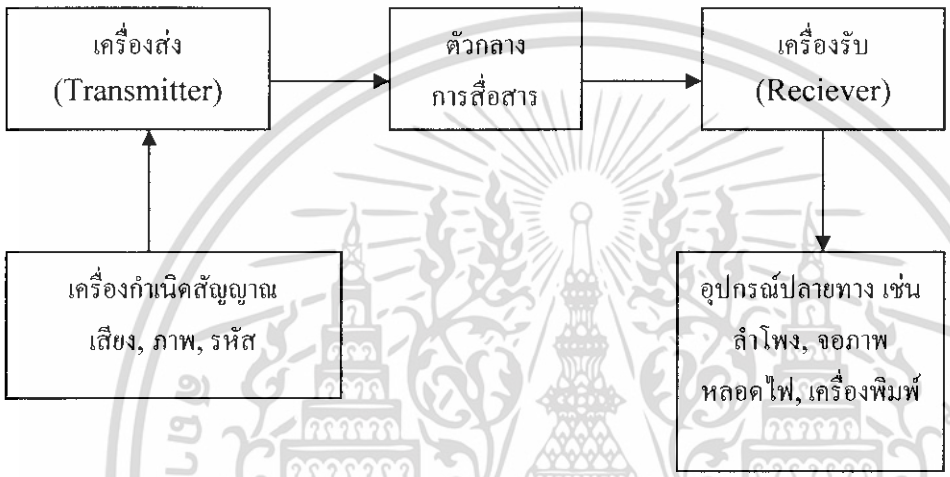
ComNoOpen	8014	Com Port ไม่สามารถทำงานได้
ComSetCommStateFailed	8015	ไม่สามารถตั้งค่าสถานะของ Port ได้
ComSetCommStateFailed	8016	ไม่สามารถSet Port ตามเหตุการณ์ที่กำหนดให้ได้
ComPortNotOpen	8018	จะสามารถหาผลลัพธ์ของข้อมูลได้ก็ต่อเมื่อ Port มีการทำงานแล้วเท่านั้น
ComPortNotOpen	8019	Port ไม่มีที่ว่างมีข้อมูลเต็มใน Port
ComReadError	8020	เกิดความผิดลาดขึ้นขณะที่อ่าน
ComDCBError	8021	เกิดความผิดพลาดภายในต้องไปแก้ไขที่ตัวควบคุม Port

ตาราง 5.1ฉ ตาราง รายละเอียดที่บ่งชี้ถึงความผิดพลาดในการใช้ MS Comm Control (ต่อ)

บทที่ 6
การสื่อสารอิเล็กทรอนิกส์

6.1 ระบบสื่อสารอิเล็กทรอนิกส์ (Electronics Communications)

การสื่อสารอิเล็กทรอนิกส์ หมายถึง การส่ง(Transmission), การรับ(Reception) และการประมวลผลของข้อมูล หรือข่าวสารระหว่างจุด 2 จุด หรือมากกว่าด้วยการใช้อิเล็กทรอนิกส์ แสดงบล็อกไดอะแกรมของระบบสื่อสารอิเล็กทรอนิกส์ดังรูป 6.1



รูป 6.1 บล็อกไดอะแกรมของระบบสื่อสารอิเล็กทรอนิกส์

ตัวกลางการสื่อสาร (Communication Mediam) เป็นช่องทางหรือตัวกลางสัญญาณของระบบสื่อสารใช้เป็นทางผ่านระหว่างจุดส่งไปยังจุดรับ เราสามารถแบ่งชนิดของการสื่อสารอิเล็กทรอนิกส์ ตามชนิดของตัวกลางสื่อสารได้ 2 แบบคือ

1) แบบมีสาย (Wire) สายในที่นี้อาจเป็นสายตัวนำไฟฟ้า 1 คู่หรือเส้นใยนำแสง (Optic Fiber)

2) แบบไร้สาย (Wireless) หรือวิทยุ(Radio) สัญญาณของระบบสื่อสารแบบไร้สายจะอยู่ในรูปแบบของคลื่นแม่เหล็กไฟฟ้า ซึ่งรวมถึงแสง

เครื่องส่ง (Transmitter) เป็นอุปกรณ์หรือวงจรอิเล็กทรอนิกส์ที่ถูกออกแบบสำหรับ แปลงสัญญาณจากแหล่งกำเนิดสัญญาณที่จะสื่อสาร ให้กลายเป็นสัญญาณที่มีรูปแบบ และระดับพลังงานที่เหมาะสมกับตัวกลางสื่อสารของแต่ละระบบ

เครื่องส่งอาจเป็นเพียงคีย์สวิทซ์ของระบบโทรเลขแบบใช้สายหรืออาจเป็นวงจรอิเล็กทรอนิกส์สลับซับซ้อนของระบบสื่อสารดาวเทียม สัญญาณที่จะสื่อสาร อาจอยู่ในรูปของสัญญาณเสียง,ภาพหรือข้อมูลในรูปของสัญญาณดิจิตอล ซึ่งสัญญาณแต่ละชนิดจะมีค่าความกว้างของ

แถบความถี่(Bandwidth) แตกต่างกัน ซึ่งความกว้างของแถบความถี่ของสัญญาณนี้เป็นพารามิเตอร์ที่สำคัญที่สุดในการพิจารณา เลือกใช้หรือออกแบบระบบสื่อสาร

เครื่องรับ (Reciever) จะเป็นอุปกรณ์และวงจรอิเล็กทรอนิกส์อีกชุดหนึ่งที่จะทำหน้าที่แปลงสัญญาณที่รับมาได้จากตัวกลาง ให้กลายเป็นสัญญาณที่มีรูปแบบและระดับพลังงานที่เหมาะสมกับอุปกรณ์ปลายทางด้านรับ เช่นเครื่องรับของระบบโทรเลขใช้สายจะเป็นเพียงขดลวด โซลีนอยด์ (Solenoid) หรือวงจรอิเล็กทรอนิกส์ที่ซับซ้อนของการรับสัญญาณโทรทัศน์ผ่านดาวเทียม

สัญญาณทางอิเล็กทรอนิกส์เกือบทุกสัญญาณ จะเกิดจากผลรวมของคลื่นซายน์หลายๆความถี่ การเดินทางของสัญญาณในวงจร จากจุดหนึ่งไปยังอีกจุดหนึ่ง หรือการเดินทางของวงจรหนึ่งไปยังอีกวงจรหนึ่งหรือจากระบบหนึ่งไปสู่อีกระบบหนึ่ง สามารถพิจารณาเป็นการเดินทางของคลื่ซายน์ทุกความถี่ที่มีอยู่ในสัญญาณนั้น จากที่หนึ่งไปอีกที่หนึ่งในรูปแบบของคลื่นแม่เหล็กไฟฟ้า (Electromagnetic waves) ซึ่งอาจจะเป็นการเคลื่อนที่ผ่านตัวนำไฟฟ้าหรือตัวอุปกรณ์ต่างๆและการเคลื่อนที่ผ่านอากาศหรืออวกาศในรูปแบบของการแผ่รังสี (Radiation)

ความกว้างแถบความถี่ของสัญญาณคือ ค่าแถบความถี่ของคลื่นซายน์ (Sine) หลายความถี่ที่รวมกันเป็นสัญญาณ ซึ่งจะค่าเท่ากับผลต่างของความถี่สูงสุดกับความถี่ต่ำสุดที่มีอยู่ในสัญญาณถ้ามีสัญญาณมากกว่าหนึ่งสัญญาณในอาณาบริเวณเดียวกัน และสัญญาณเหล่านั้นมีค่าแถบความถี่ที่ทับซ้อนกัน จะทำให้เกิดการรบกวนซึ่งกันและกัน ถ้าระดับกำลังของสัญญาณต่างๆที่มีแถบความถี่ซ้อนทับกัน มีค่าใกล้เคียงกัน ก็จะเกิดการรบกวนซึ่งกันและกัน แต่ถ้าระดับกำลังของสัญญาณแตกต่างกันมาก สัญญาณที่มีกำลังมากแทบไม่ถูกรบกวนจากสัญญาณที่มีกำลังต่ำกว่ามาก

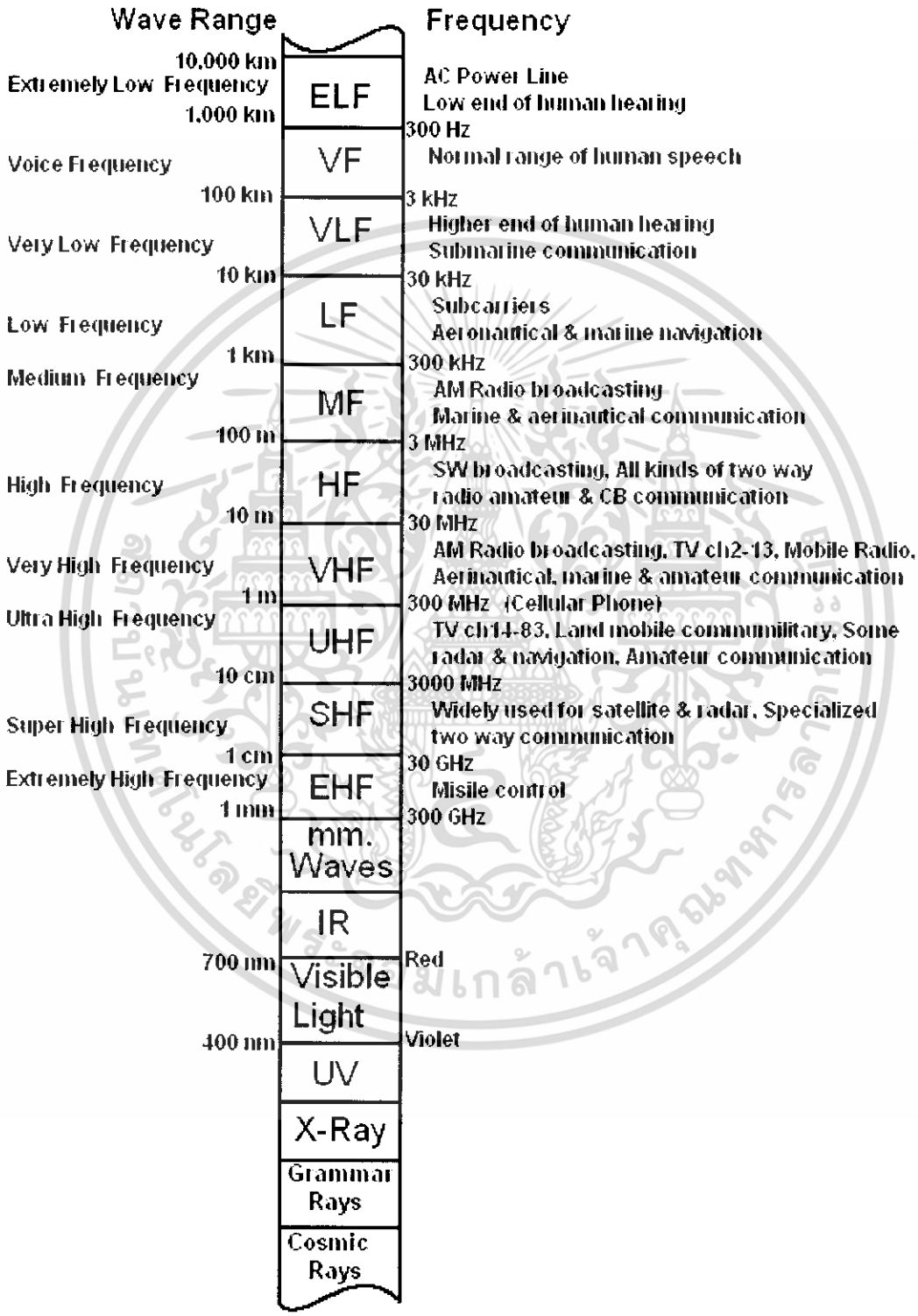
ในทางตรงกันข้าม สัญญาณที่มีกำลังต่ำจะถูกสัญญาณที่มีกำลังสูงกว่ากลบจนหมดในระบบสื่อสาร เราไม่สามารถส่งสัญญาณที่มีแถบความถี่ซ้อนทับกัน ผ่านตัวกลางของการสื่อสารเดียวกัน ภายในเวลาเดียวกันได้

สัญญาณใดก็ตามที่มีแถบความถี่ซ้อนทับกับสัญญาณที่เราต้องการสื่อสาร จะถูกเรียกว่าสัญญาณรบกวน (NOISE) การเดินทางของสัญญาณจากเครื่องส่งไปยังเครื่องรับจะถูกลดทอนให้มีกำลังต่ำลง ในขณะที่ผ่านตัวกลาง เมื่อสัญญาณมีกำลังไฟฟ้าลดลง จะมีโอกาสถูกรบกวนจากสัญญาณรบกวนที่อยู่ระหว่างเส้นทาง ยิ่งไปกว่านั้นภายในอุปกรณ์อิเล็กทรอนิกส์ก็เป็นแหล่งกำเนิดสัญญาณรบกวนอีกด้วย ถ้าความกว้างแถบความถี่ของสัญญาณยิ่งกว้างก็จะยิ่งเพิ่มโอกาสที่จะถูกรบกวนมากยิ่งขึ้น

จุดประสงค์ของ Electronic Communication คือการติดต่อสื่อสารระหว่างจุด 2 จุด ข้อมูลข่าวสารของการสื่อสารจะต้องอยู่ในรูปของคลื่นแม่เหล็กไฟฟ้า ซึ่งคลื่นแม่เหล็กไฟฟ้านี้จะสามารถแผ่กระทบผ่านตัวนำไฟฟ้า หรืออากาศ/อวกาศได้

แถบความถี่ของสัญญาณ(Signal Bandwidth) คือแถบความถี่ของคลื่นแม่เหล็กไฟฟ้าที่สัญญาณครอบครองอยู่ ซึ่งจะมีค่าเท่ากับผลต่างของความถี่สูงสุด กับความถี่ต่ำสุดของสัญญาณ

แถบความถี่ทั้งหมดของคลื่นแม่เหล็กไฟฟ้าที่สามารถนำมาใช้ประโยชน์ในการสื่อสาร ได้ถูกแบ่งเป็นแถบความถี่ต่างๆและได้ถูกตั้งชื่อ โดย CCIR (The International Radio Consulative Committee) ซึ่งในรูปที่ 6.2 ได้แสดงถึง ค่าความถี่และชื่อของแถบความถี่ต่างๆ รวมทั้งการนำไปใช้งาน



รูป 6.2 ค่าความถี่และประเภทของคลื่นแม่เหล็กไฟฟ้า

ความกว้างของแถบความถี่ของช่องสื่อสาร(Communication Channel Bandwidth) จะต้องมีความกว้างพอที่จะให้ความถี่ที่มีนัยสำคัญทั้งหมดของสัญญาณข้อมูลข่าวสารผ่านไปได้ทั้งหมด ความกว้างของแถบความถี่ของช่องสื่อสาร (BWch) จะต้องมีค่าเท่ากับหรือมากกว่าแถบความถี่ของสัญญาณ (Bwsignal)

BWch >= Bwsignal (6.1)

เช่นแถบความถี่ของเสียงมนุษย์คือ 300H-3KHz ความกว้างของแถบความถี่ของช่องสื่อสารสำหรับเสียงมนุษย์จะต้องมีความกว้างอย่างน้อยที่สุดเป็น 2.7 KHz (3 KHz – 300 Hz = 2700 Hz)

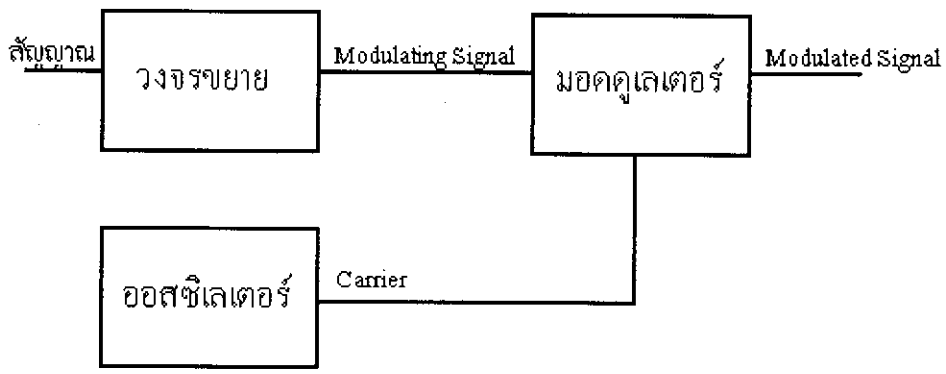
ถึงแม้ว่าระบบสื่อสารที่มีความกว้างของแถบความถี่ของช่องสื่อสารกว้าง จะมีความจุของข้อมูลข่าวสารมากอย่างไรก็ตาม กฎพื้นฐานในการออกแบบระบบสื่อสารที่วิศวกรผู้ออกแบบพึงระลึกอยู่ตลอดเวลา คือ จะต้องพยายามทำให้ความกว้างของแถบความถี่ของช่องสื่อสารมีความกว้างให้น้อยที่สุดที่จะเป็นไปได้ โดยไม่ทำให้สูญเสียข้อมูลข่าวสารที่มีนัยสำคัญ เพื่อให้มีจำนวนของช่องสื่อสารให้มากที่สุดเท่าที่จะเป็นไปได้ซึ่งจะเป็นการเปิดโอกาสให้คนจำนวนมากได้ใช้ประโยชน์จากแถบความถี่ของคลื่นแม่เหล็กไฟฟ้าอันเป็นทรัพยากรที่มีจำกัดร่วมกัน

ความถี่บางความถี่ เช่น ความถี่ที่สูงกว่าย่าน HF (VHF,UHF,...) จะสามารถแพร่กระจายได้ในขอบเขตจำกัด เราจะสามารถใช้ความถี่ของระบบสื่อสารซ้อนทับกันในบริเวณที่ห่างไกลกัน สัญญาณของระบบสื่อสารก็จะไม่รบกวนกัน การจำกัดบริเวณของการแพร่กระจายคลื่นอาจทำได้โดยไม่ใช้เครื่องส่งที่มีกำลังสูงเกินกว่าความจำเป็น แต่ความถี่บางความถี่เช่นในย่าน HF จะมีความสามารถสะท้อนกับบรรยากาศของโลก(ชั้น Ionozsphere)ได้ดี จะสามารถแพร่กระจายคลื่นไปได้ไกลมาก

ITU (International Telecommunication Union) ซึ่งเป็นองค์การย่อยของสหประชาชาติ มีหน้าที่ในการแบ่งสรรย่านความถี่ของช่องสื่อสารให้กับประเทศต่างๆ ในการตั้งสถานีวิทยุหรือการส่งดาวเทียมสื่อสาร ประเทศผู้เป็นเจ้าของจะต้องขอใช้ช่องสื่อสารจาก ITU ทั้งนี้เพื่อป้องกันมิให้ช่องสื่อสารต่างๆรบกวนซึ่งกันและกัน

6.2 มอดดูเลชั่นและดีมอดดูเลชั่น (Modulation & Demodulation)

ความถี่ของข้อมูลหรือสัญญาณโดยทั่วไปมักจะมีความถี่ต่ำ การแพร่กระจายของคลื่นแม่เหล็กไฟฟ้าความถี่ต่ำจะกระทำได้ไม่ดี เพราะสัญญาณความถี่ต่ำจะมีความยาวคลื่นยาวมากเราสามารถที่จะเลื่อนความถี่ของสัญญาณให้มีค่าสูงขึ้นได้โดยการมอดดูเลท สัญญาณที่ต้องการจะส่งกับคลื่นพาห์(carrier)ความถี่สูงหรือกล่าวอีกนัยหนึ่งได้ว่าการมอดดูเลทคือกระบวนการที่สัญญาณที่จะส่ง (Modulating Signal)ทำให้คุณสมบัติของคลื่นพาห์(ขนาด ความถี่ และเฟส)เปลี่ยนแปลงไปตามสภาวะของสัญญาณ สัญญาณที่ได้จากการมอดดูเลทเรียกว่า Modulated Signal(Wave) ดังรูปที่ 6.3



รูป 6.3 บล็อกไดอะแกรมของการมอดดูเลทในเครื่องส่งวิทยุ

การมอดดูเลทสามารถแบ่งได้เป็น 3 ประเภทใหญ่ๆ คือ

- Amplitude Modulation (AM) ขนาดของคลื่นพาห์จะเปลี่ยนไปตามขนาดของสัญญาณ

- Frequency Modulation (FM) และ Phase Modulation (PM) ความถี่ของคลื่นพาห์หรือเฟสของคลื่นพาห์จะเปลี่ยนแปลงไปตามขนาดของสัญญาณ ลักษณะของ Modulated wave ของ FM กับ PM จะคล้ายกัน

- Pulse Mudulation คลื่นพาห์จะเป็นพัลส์ สัญญาณจะทำให้คุณสมบัติต่างๆของพัลส์เปลี่ยนแปลงไป เช่น ขนาดของพัลส์ ความกว้างของพัลส์ เป็นต้น

6.2.1 Amplitude Modulation

จากที่กล่าวมาแล้วว่าการมอดดูเลทแบบ AM นั้น ขนาดของสัญญาณจะทำให้ขนาดของคลื่นพาห์เปลี่ยนแปลงไปสามารถเขียนสมการของ AM Modulated wave ได้เป็น

$$V = (Vcp+Vs(t)) \sin(\omega ct) \quad (6.2)$$

V : AM Modolated wave

Vs(t) : สัญญาณที่จะส่งออกไปหรือ Modulating Signal

ถ้า Vs มีความถี่เป็น ωs ค่าของ Vs(t) คือ

$$Vs(t) = Vsp \sin(\omega st) \quad (6.3)$$

Vcp Sin(ωc t) : คลื่นพาหะซึ่งมีความถี่ ωc

โดยการแทนค่า Vs(t) ลงใน 2.2 จะได้สมการของคลื่น AM เป็น

$$V(AM) = (Vcp + Vsp \sin(\omega s\ t)) \sin(\omega c\ t) \quad (6.4)$$

ให้ $m = Vsp/Vcp$ (6.5)

จัดรูปสมการ 6.4 ใหม่เป็น

$$V(AM) = Vcp(1+m \sin(\omega s\ t)) \sin(\omega c\ t) \quad (6.6)$$

จากความสัมพันธ์ทางตรีโกณมิติ

$\text{Cos}(x + y) - \text{Cos}(x - y) = 2 \text{Sin}\, x \text{Sin}\, y$

สมการ 6.6 ก็จะเป็น

$$V(AM) = Vcp(1 + m \text{Sin}(\omega s\, t)) \text{Sin}(\omega c\, t) = Vcp \text{Sin}(\omega c\, t)$$

$$+ m/2\ Vcp(\text{Cos}((\omega c - \omega s)t) - \text{Cos}((\omega c + \omega s)t)) \quad (6.7)$$

ซึ่งค่าต่างๆทางขวามือจะแสดงให้เห็นสเปคตรัมความถี่ของคลื่น AM 2 ส่วน

เทอมของ ωc เรียกว่าส่วนของคลื่นพาห์

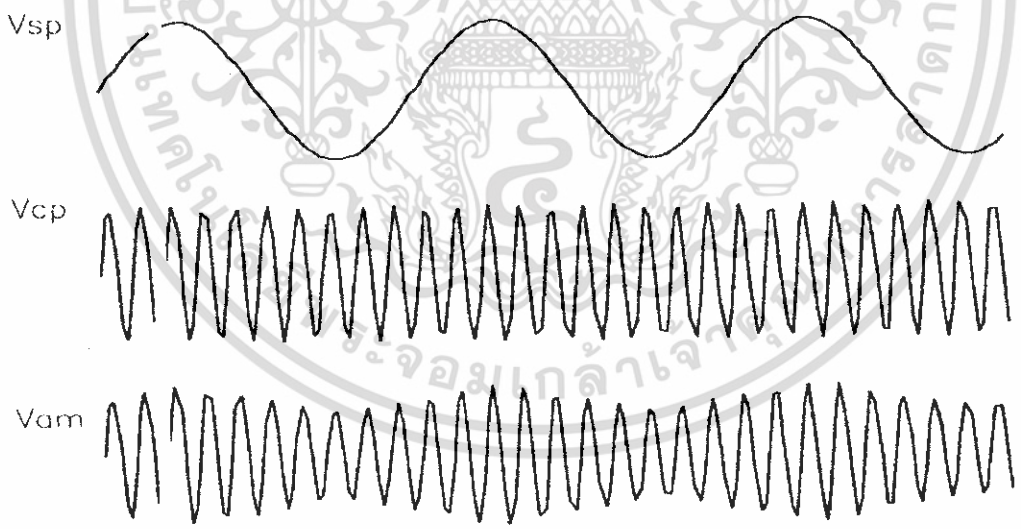
เทอมของ $\omega c + \omega m$ และ $\omega c - \omega s$ เรียกว่าส่วนของ Side Band ซึ่ง $m/2\ Vcp \text{Cos}((\omega c - \omega s)t)$ หรือ $Vsp/2 \text{Cos}((\omega c - \omega s)t)$ เรียกว่า Lower Side Band

และ $m/2\ Vcp\text{Cos}((\omega c + \omega s)t)$ หรือ $\text{Cos}((\omega c + \omega s)t)$ เรียกว่า Upper Side Band

ซึ่งจะเห็นได้ว่า Modulating Signal ซึ่งเดิมมีความถี่ ωs จะถูกเลื่อนความถี่เป็น $\omega c - \omega s$ และ $\omega c + \omega s$ เมื่อคลื่นดังกล่าวเดินทางผ่านอวกาศไปยังเครื่องรับ วงจรเตรื่องรับก็จะทำการเลื่อนความถี่ $\omega c + \omega s$ ให้กลับมาเป็น ωs ตามเดิม วงจรที่ทำหน้าที่นี้เรียกว่าดีมอดูเลเตอร์

ค่าของ m มีชื่อเรียกว่า modulation index ถ้าทำเป็นร้อยละ ก็จะเรียกว่า Percentage of Modulation หรือ $\%Mod = (Vsp/Vcp)*100$ (6.8)

ซึ่ง m จะมีค่าได้ระหว่าง 0-1 % Mod จะมีค่าได้ไม่เกิน 100% ถ้า m = 1 จะทำให้ Modulated Wave ขาดหายไปเป็นช่วงๆ



รูป 6.4 AM Modulated Wave

จากรูป 6-4 จะได้

$$\tfrac{1}{2}\ Ven(max) = \tfrac{1}{2}\ Ven(min) + 2Vsp \quad (6.9)$$

$$Vcp = Ven(max) - Vsp \quad (6.10)$$

จากสมการ 6.9,6.10 และ 6.8 จะได้

$$\%Mod = ((Ven(max) - Ven(min)) / (Ven(max) + Ven(min)))*100 \quad (6.11)$$

6.2.2 Frequency Modulation และ Phase Modulation

การมอดูเลทแบบ FM นั้นสัญญาณที่ต้องการส่งจะไปทำให้ความถี่ของคลื่นพาห์เปลี่ยนแปลง ถ้าสัญญาณมีค่าเป็นบวก ก็จะทำให้ความถี่ของคลื่นพาห์สูงขึ้นและ เมื่อสัญญาณเป็นลบก็จะทำให้ความถี่ของคลื่นพาห์ลดลงหรือในทางตรงกันข้าม นั่นคือขนาดของสัญญาณจะเป็นตัวทำให้ความถี่ของคลื่นพาห์เปลี่ยนแปลง ให้ Kf เป็นค่าคงที่เรียกว่า Frequency Deviation constant ซึ่ง

$$Kf = \Delta fc/ \Delta Vs \quad (6.12)$$

ความถี่ของ FM Modulated wave จะเป็น

$$f(FM) = fc + Kf \qquad Vs = fc + \Delta fc \quad (6.13)$$

fc เรียกว่า deviation Frequency

สมการของ FM Modolated wave คือ

$$V(FM) = Vcp\ Sin(\omega c - (\omega c/\omega s)Cos(\omega s\ t)) \quad (6.14)$$

ซึ่งค่า $\Delta fc/f$ คือ Modulation Index ของFM ดังนั้นสมการ()อาจเขียนในรูปของ

$$V(FM) = Vcp\ Sin(\omega c\ t - mfCos(\omega s\ t)$$

Mf คือค่าของ Modulating index สมการ (2.14)

สามารถกระจายอยู่ในรูปของ Bessel function ได้เป็น

$$V(FM) = Jo(mf)Sin(\omega c\ t) + J1(mf)VcpSin(\omega c + \omega s) + Sin((\omega c - \omega s)t)$$

$$+ J2(mf)VcpSin((\omega c + 2\omega s)t) + Sin((\omega c + 2\omega s)t) + J3(mf)Vcp \quad (6.15)$$

ซึ่งจะเกิด Side Band ค่าต่างๆมากมายขึ้นอยู่กับค่าของ modulation index (mf) ซึ่งสเปคตรัมของความถี่ต่างๆจะเป็นไปตามค่าของ Jo(mf), J1(mf).... โดยที่ Jn(mf) คือ Bassel Function of the First Kind

สำหรับกรณีของ Phase Modulation ก็จะให้ผลทำนองเดียวกับ FM เพราะ $\omega = de/dt$ เมื่อเราเปลี่ยนค่าเฟสของคลื่นพาห์ก็จะมีผลทำให้ความถี่ของ คลื่นพาห์เปลี่ยนแปลงได้ PM อาจเรียกอีกแบบหนึ่งว่า indirect FM

จากรูปที่ 6.4 จะเห็นได้ว่า frequency deviation ของ PM จะมีเฟสนำ Frequency deviation ของ FM อยู่ 90 องศา(จาก W = de/dt) และ Frequency deviation ของ pm จะเปลี่ยนแปลงตามความถี่ของ Modulating Signal ด้วย ซึ่ง

$$F(PM) = O\ fsCos\omega s\ t \quad (6.16)$$

โดยที่

$$O = K0\ V \quad (6.17)$$

ซึ่ง K0 เป็นค่าคงที่ Phase deviation constant ทำนองเดียวกับ Kf

การที่จะทำให้ Modulating wave ของ PM เหมือนกับ FM ทำได้โดยการผ่านสัญญาณ Modulating Signal ผ่านวงจร Signal correction Network โดย Network ดังกล่าวจะทำหน้าที่ 2 อย่าง คือ Lag phase ของ Modulating signal ไป 90 องศา และลดขนาดของ Modulating signal ที่ความถี่สูง

6.3 สัญญาณรบกวน (Noise)

สัญญาณรบกวน (Noise) คือ สัญญาณที่เราไม่ต้องการในระบบ มักไม่มีรูปคลื่นที่แน่นอนอยู่ในรูปแบบของการสุ่ม(Random) อย่างไรก็ตามสัญญาณที่เราไม่ต้องการ ในบางครั้งอาจมีรูปคลื่นที่แน่นอน เช่น สัญญาณที่เกิดจากสวิทซ์ของสัญญาณนาฬิกา(clock) เป็นต้น สัญญาณรบกวนอาจแบ่งได้เป็น 2 ประเภท คือ

1) Correlated Niose เป็น Noise ที่เกิดจากสัญญาณเอง ถ้าไม่มีสัญญาณก็จะไม่มี Noise ชนิดนี้ Correlated Noise จะมีรูปคลื่นที่แน่นอน(สามารถเขียน V(t) ได้) เกิดจากคุณสมบัติที่ไม่เป็นเชิงเส้นของวงจรขยาย Noise ชนิดนี้คือ Harmonics Distortions และ Intermodulation Distortions) หน่วยที่เราใช้วัด Distortion คือ % Distortion ซึ่งนิยามโดย

$$\% \text{ Distortion} = Vn(rms)*100/V \text{ fundamental(rms)} \quad (6.18)$$

Vn(rms) คือ ค่า rms ของ noise อาจหมายถึงผลรวมของ ฮาร์โมนิกส์

หรือ ผลรวมของ Cross Product

2) Uncorrelated Noise เป็น Noise ที่เกิดขึ้นตลอดเวลาไม่ว่าจะมีสัญญาณหรือไม่ก็ตาม ซึ่ง oise ชนิดนี้มักอยู่ในรูปแบบของสัญญาณแบบสุ่ม (Random Signal) มักไม่สามารถเขียนค่า V(t)ได้การศึกษาที่เกี่ยวข้องกับ Noise นี้มักใช้ขบวนการทางสถิติเข้ามาเกี่ยวข้องกับ Uncorrelated Noise ยังสามารถแบ่งได้เป็น 2 แบบคือ

- External Noise
- Internal noise

6.3.1 External Noise

สามารถแบ่งได้ตามสาเหตุหรือแหล่งกำเนิดได้ 3 ชนิดคือ

1) Man-Made Noise หรือ Industrial Noise

เกิดจากการกระทำของมนุษย์ สาเหตุที่ทำให้เกิด noise ชนิดนี้คือ การสวิทซ์ ไม่ว่าจะเป็นกลไก หรืออิเล็กทรอนิกส์ เช่นการสปาร์คของหัวเทียน, หน้าสัมผัสของสวิทซ์หรือ รีเลย์ แปรงถ่านของมอเตอร์, วงจรดิจิตอล หรือ ภาคจ่ายไฟแบบสวิทซ์ชิ่ง Man-Made Noise เดินทางจากแหล่งกำเนิดเข้ามาสู่วงจร ได้ทั้งโดยการนำไฟฟ้า (Conduction) ตามสายตัวนำ และ แผ่กระจก(Radiate) เข้ามาสู่วงจร ในรูปคลื่นแม่เหล็กไฟฟ้า การลด Noise ชนิดนี้อาจทำได้โดย การต่อกราวนด์ที่ถูกต้อง การบายพาส หรือดีคัปปลิ้ง (Bypass or Decoupling) แหล่งจ่ายไฟตรงและการชีลด์ (Shield) Man-Made Noise จะมีความเข้มสูงในบริเวณที่มีประชากร หรือ อุตสาหกรรมหนาแน่น จะอยู่ในรูปแบบของสัญญาณแบบสุ่ม มีช่วงความถี่ตั้งแต่ไฟตรง ถึงประมาณ 600 MHz

2) Atmospheric Noise

เกิดโดยธรรมชาติ จากความปั่นป่วนของประจุไฟฟ้า ในบรรยากาศของโลก เช่น ฟ้าแลบ ฟ้าผ่า แถบความถี่ของ Atmospheric Noise แผ่กระจายตลอดย่านความถี่วิทยุ แต่อย่างไรก็ตาม องค์ประกอบความถี่ที่ความถี่สูง จะมีกำลังไฟฟ้าลดลงมาก จนถือว่า Atmospheric Noise จะมีผลต่อระบบสื่อสารน้อยมาก ที่ความถี่ มากกว่า 30MHz

3) Space Noise or Extra Terrestrial Noise

เกิดจากสัญญาณไฟฟ้า นอกโลก อาจเรียกอีกชื่อหนึ่งว่า Deep Space Noise ซึ่งอาจเกิดจากดวงอาทิตย์,ทางช้างเผือก หรือกาแล็กซี่อื่น Space Noise สามารถแบ่งย่อยได้เป็นอีก 2 ชนิดย่อยคือ

- Solar noise เกิดจากดวงอาทิตย์ ซึ่งอาจแบ่งได้เป็น 2 สภาวะคือ สภาวะสงบ(Quiet Condition) เกิดขึ้นขณะที่ดวงอาทิตย์แผ่รังสีคงที่และสภาวะที่มีความเข้มสูง จากปรากฏการณ์ Sun Spot และ Solar Flare-ups ซึ่ง Solar Noise นี่จะมีความเข้มสูงเป็นคาบทุก 11 ปี

- Cosmic Noise เกิดจากดวงดาวทั้งหลายในกาแล็กซี่ต่างๆถึงแม้ว่าระยะทางห่างจากโลกมาก แต่จำนวนของดวงดาวก็มีจำนวนมากมาย ผลรวมของ Noise เหล่านี้จะมีพลังงาน รบกวนระบบสื่อสารได้

แถบความถี่ของ Space Noise ที่มีนัยสำคัญ คือ 8 MHz จนถึง 1.5 GHz

Space Noise ที่มีความถี่ต่ำกว่า 8 MHz จะถูกลดทอนลงอย่างมาก จากโอโซนจนไม่มีผลต่อระบบสื่อสารบนโลก

6.3.2 Internal Noise

เป็น Noise ที่เกิดขึ้นจากตัวอุปกรณ์อิเล็กทรอนิกส์ในวงจร ณ. จุดต่อสัญญาณเข้าก็จะมี Noise เกิดขึ้น ณ. จุดนั้นแล้ว การออกแบบวงจรขยาย ภาคแรกของเครื่องรับ ต้องมีความพิถีพิถันเป็นพิเศษ เพราะสัญญาณที่รับมาได้ครั้งแรก(ก่อนถูกขยาย) มักมีขนาดเล็กมาก ถ้า Noise ที่เกิดขึ้น ณ. จุดต่อสัญญาณเข้า ที่เกิดจากตัวอุปกรณ์ในวงจรมีค่ามากก็จะทำให้การสื่อสารล้มเหลว เนื่องจาก Noise จะรวมเข้ากับสัญญาณจนไม่สามารถแยกแยะได้ สามารถแบ่งเป็น 5 ชนิดได้แก่

1) Thermal Noise หรือ White noise

Thermal Noise หรือ White noise ถูกค้นพบครั้งแรกโดย JB. Johnson 1928

Johnson สังเกตพบว่ามีแรงดันไฟฟ้า ซึ่งไม่มีคาบแน่นอน เกิดขึ้นกับตัวนำไฟฟ้าทุกชนิด และมีค่าเพิ่มขึ้นเมื่ออุณหภูมิเพิ่มขึ้น Thermal Noise เกิดจากการเคลื่อนที่อย่างสุ่มของอิเล็กตรอน หรือ การสั่นของอะตอม เนื่องจากพลังงานความร้อน ซึ่งจะเกิดขึ้นกับตัวอุปกรณ์ทุกชนิดที่มีความต้านทานไฟฟ้า บางครั้ง เราอาจเรียก Thermal Noise นี้ว่า Johnson Noise หรือ Resistance Noise

Thermal Noise นี้มีค่าพลังงานเท่ากับทุกความถี่ แผ่กระจายตลอดย่านความถี่ของคลื่นแม่เหล็กไฟฟ้า ด้วยเหตุนี้ Thermal Noise จึงมีอีกชื่อหนึ่งว่า White noise

2) Shot Noise

เป็น Noise ที่เกิดขึ้นในตัวอุปกรณ์ที่ใช้ในการขยาย เกิดจากการกระเพื่อมของกระแส (Fluctuation of Current) ไปจากค่ากระแสเฉลี่ย ในกรณีของหลอดสุญญากาศ เกิดจากการที่อิเล็กตรอนหลุดออกมาจากคาโถด เป็นไปแบบสุ่ม ทำให้อิเล็กตรอนเดินทางไปถึงเพลา(อาโนด)ไม่พร้อมกัน ในกรณีของไดโอดหรือทรานซิสเตอร์ จะเกิดจากการแพร่อย่างสุ่มของประจุพาหนะ และการกำเนิด (generation) หรือการรวมตัว (Recombination)ของคู่อิเลคตรอนโฮล อย่าสุ่มในผลึกสารกึ่งตัวนำ

3) Transittione Noise

เป็น Noise ที่เกิดขึ้นที่ความถี่สูง ของตัวอุปกรณ์ขยายสัญญาณ การเคลื่อนที่ของประจุพาหนะ จากขั้วไฟฟ้า ที่เป็นต้นกำเนิดของประจุ (อีมิตเตอร์ใน BJT, คาโถด ในหลอด,ซอส ใน FET) ไปยังขั้วไฟฟ้าที่ทางออก(คอลเลคเตอร์ หรือ เพลท หรือ เดรน)จะต้องใช้เวลาค่าหนึ่งเรียกว่า Transit Time (หรือ Delay Time) ประจุพาหนะ เหล่านี้บางส่วน เมื่อเดินทางไปถึงขั้วไฟฟ้าที่ทางออก จะเคลื่อนที่สะท้อนกลับมายังจุดกำเนิด ถ้าคาบเวลาของสัญญาณ มีค่าใกล้เคียงหรือน้อยกว่า Transit Delay Time ของประจุพาหนะจะทำให้ เกิด Noise

4) เกิดจากการกระเพื่อมของสภาพความนำไฟฟ้า (Fluctuation of Conductivity)

เนื่องจากความไม่สมบูรณ์ของจุดต่อระหว่างสาร 2 ชนิด เช่น หน้าสัมผัสของสวิตซ์ จุดต่อของสารกึ่งตัวนำกับลวด (Ohmic Contact) เป็นต้น

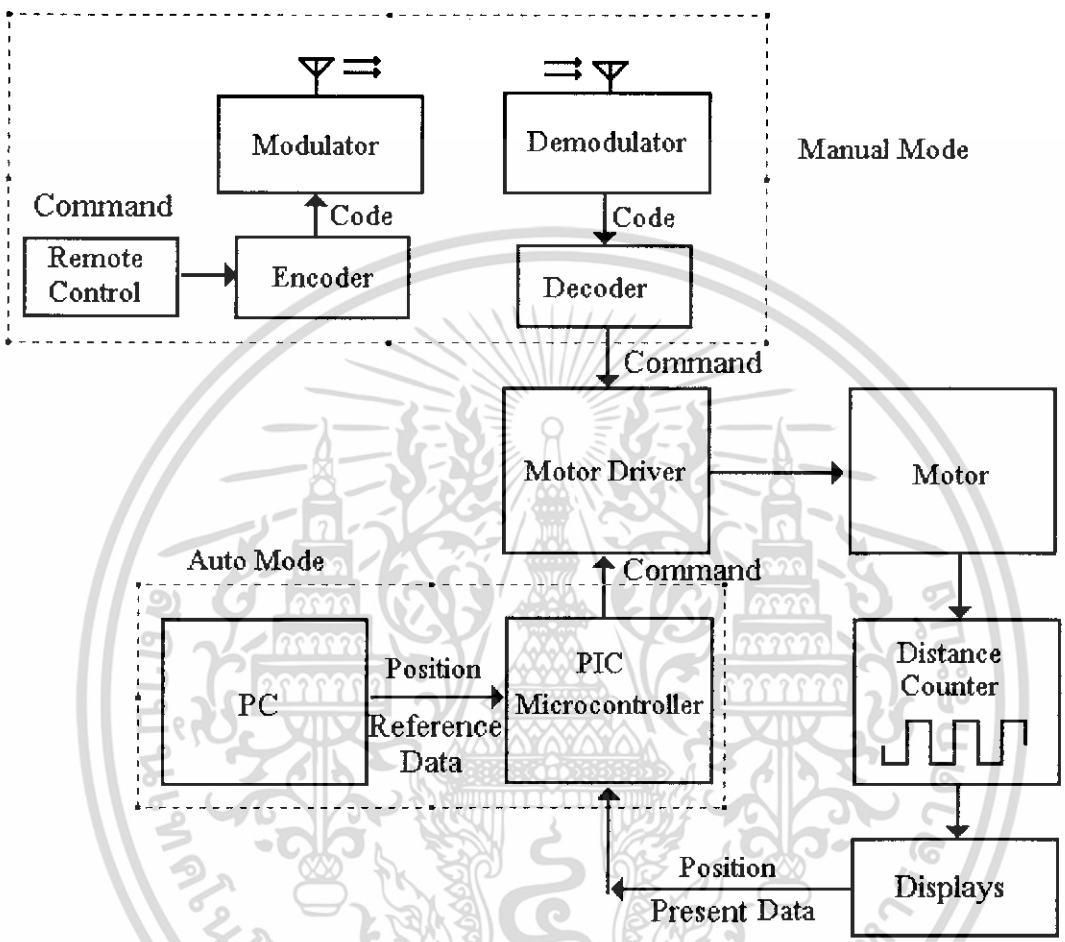
5) Pop Corn Noise หรือ Burst Noise

เกิดจากข้อบกพร่องในการผลิตอุปกรณ์ของโรงงาน

บทที่ 7.

การออกแบบ แนวคิดและการหลักการทำงานของวงจร

แสดงลักษณะโครงงานโดยรูปที่7.1



รูป 7.1 บล็อกไดอะแกรมสำหรับโครงงาน

ซึ่งแบ่งการทำงานเป็นส่วนๆดังนี้

ส่วน Manual Mode

ส่วนภาคส่ง จะรับคำสั่งที่จะสั่งให้มอเตอร์แกนต่างๆหมุน จากรีโมตคอนโทรล ซึ่งสร้างรหัสคำสั่งโดย Encoder ก่อนที่จะส่งรหัสคำสั่งดังกล่าวโดย AM Modulator เพื่อส่งข้อมูลไปยังเสาอากาศฝั่งส่ง จากนั้นตัว Demodulator ที่ภาครับ เมื่อได้รับข้อมูลมาจากทางเสาอากาศ จะกระทำการแยกเอาพาหะออก เหลือเฉพาะคำรหัส เพื่อเข้าสู่ Decoder เพื่อทำการถอดรหัส ภายหลังจากการถอดรหัส จะได้คำสั่งที่สำหรับสั่งให้วงจร Motor Driver ขับมอเตอร์ในแกนต่างๆต่อไป

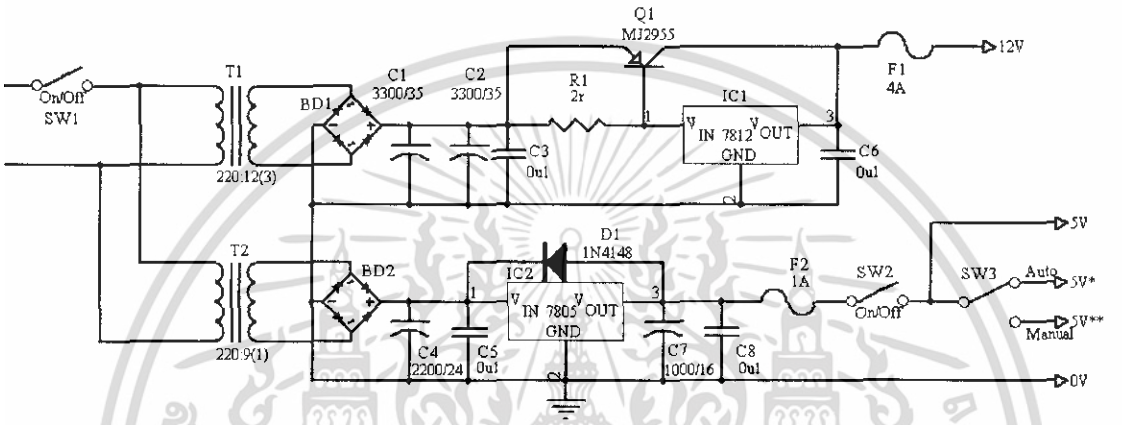
ส่วน Auto Mode

ในส่วนนี้จะทำงานร่วมกับวงจรต่างๆ รวมทั้งโปรแกรมในคอมพิวเตอร์ด้วย กล่าวคือ ผู้ใช้จะกำหนดพิกัดแกนต่างๆที่ต้องการ ผ่านทางคอมพิวเตอร์ PCจะกระทำการส่งข้อมูลพิกัดที่ได้จากผู้ใช้

ไปยัง PIC Microcontroller ซึ่งจะเป็นพิกัดอ้างอิง จากนั้น ตัวMicrocontroller จะกระทำการอ่านพิกัดปัจจุบัน เพื่อนำมาเทียบกับพิกัดอ้างอิง เมื่อประมวลผลแล้ว หากพิกัดยังไม่ตรงกัน จะกระทำการสั่งมอเตอร์แกนต่างๆให้หมุนต่อไป และกระทำการอ่านพิกัดอีก จะทำเช่นนี้จนกระทั่งพิกัดปัจจุบันเท่ากับพิกัดอ้างอิง นั่นหมายถึง เครนได้ไปยังพิกัดที่ผู้ใช้ต้องการแล้ว ซึ่งพิกัดปัจจุบันจะอ่านได้จากการนับ พัลส์ ของ Distance Counter เพื่อแปลงระยะทางเป็นรหัสข้อมูล (ADC) โดยในขณะเดียวกันจะนำรหัสข้อมูลนั้นแสดงเป็นตัวเลขเป็น Display เพื่อความสะดวกต่อผู้ใช้

วงจรต่างตามที่กล่าวมานั้นได้แก่

7.1 ภาคจ่ายไฟ



รูป 7.2 วงจรภาคจ่ายไฟ

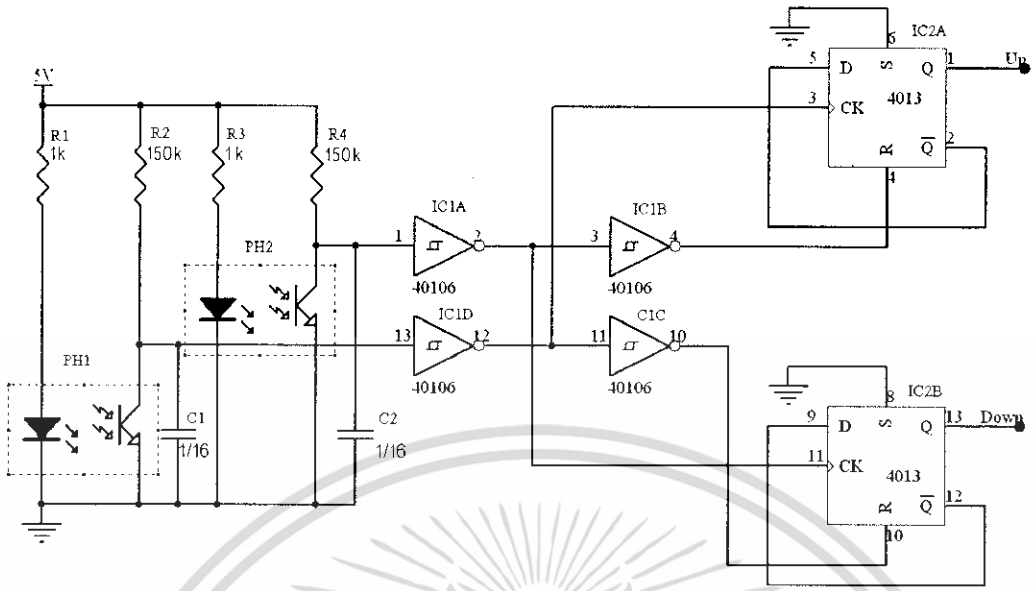
แสดงดังรูป 7.2 จะใช้หม้อแปลงสองตัว ตัวแรกใช้สำหรับจ่ายไฟสำหรับมอเตอร์ขับเคลื่อนเครนทั้งหมด(แกน X Y Z) สร้างเป็นแรงดัน 12V. โดยไอซีเรกูเลเตอร์ 7812 ขยายกระแสโดยทรานซิสเตอร์ 2N2955 ทั้งนี้ต้องติดฮีทซิงค์ระบายความร้อนไว้ด้วย และติดฟิวส์ตัดเมื่อกระแสมอเตอร์ใช้เกิน 4 A.

หม้อแปลงตัวที่สอง จะใช้สำหรับสร้างไฟเลี้ยงวงจรทั้งหมด ซึ่งเป็นไฟเลี้ยง 5V. โดยผ่านไอซีเรกูเลเตอร์ 7815 และติดฟิวส์ตัดเมื่อกระแสเอ้าท์พุทเกิน 1A. โดยวงจรนี้จะใช้สวิตซ์ SW3 เลือกโหมดเพื่อแยกจ่ายไฟให้ 2วงจรออกจากกัน ทำให้มีไฟเลี้ยง 5V. ออกมา 3 แห่ง คือ 5V. ,5V* ,5V** ซึ่งแรงดันทั้งสามค่ามีเท่ากัน แต่ใช้คนละที่ คือ 5V.ใช้สำหรับวงจรที่ต้องการใช้ไฟเลี้ยงตลอด เช่น วงจร ตัวนับรอบ,วงจรเซ็นเซอร์ตรวจจับ เป็นต้น ,5V* ใช้สำหรับวงจรโหมดออโต้ (Auto) คือสำหรับไมโครคอลโทรลเลอร์และการติดต่อกับคอมพิวเตอร์เท่านั้นนั้น ซึ่งจะไม่ต้องการใช้ไฟเลี้ยงเมื่อผู้ใช้เลือกใช้โหมดแมนนวล (Manual) หรือ การใช้รีโมตไร้สาย โดยในโหมด Manualจะใช้ชุดไฟเลี้ยง 5V**

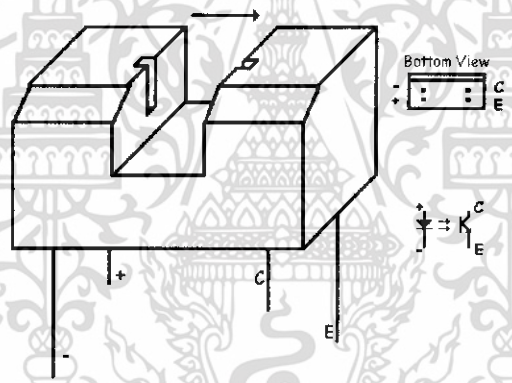
7.2 วงจรตรวจนับ

จะทำงานตลอดทั้งโหมด Auto และ Manual ใช้ไฟเลี้ยง 5V. แบ่งเป็น2ส่วน

7.2.1 วงจรตรวจนับระยะการกระจัด



รูป 7.3 รูปวงจรตรวจนับระยะการกระจัด

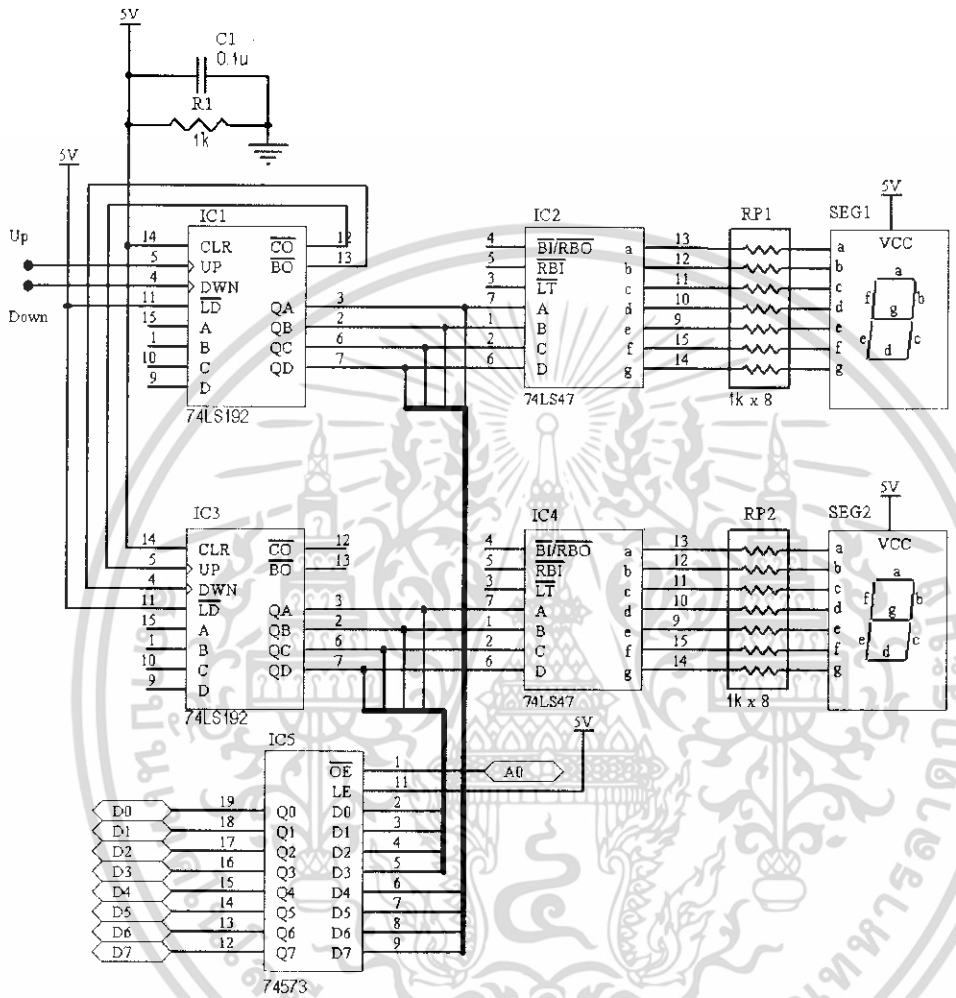


รูป 7.4 รูปชุดดีเท็กเตอร์ที่ใช้

แสดงดังรูป 7.3 เป็นวงจรแปลงอนาล็อกเป็นดิจิตอล กล่าวคือทำหน้าที่สร้างพัลส์ (Pulse) ออกมาเมื่อมีการกระจัดของเครน โดยคำนึงถึงทิศทาง ถ้ามอเตอร์มีการหมุนทำให้เครนวิ่ง ขณะเดียวกันตัวตัดผ่านแสง ที่ติดกับแกนมอเตอร์จะหมุนไปด้วย เราเรียกตัวตัดผ่านแสงดังกล่าวว่า Encoder (ลักษณะเป็นจานกลมสีดำ ที่รอบขอบมีช่องให้แสงผ่านได้บางช่วง ซึ่งช่องไฟเท่าๆกัน ไม่เกี่ยวข้องกับไอซีสร้างรหัสในวงจร Manual) เมื่อตัว Encoder ตัดผ่านชุดดีเท็กเตอร์ลักษณะดังรูปที่ 7.4 ทำให้วงจรสามารถสร้างพัลส์ออกมาได้ จะใช้หลักการคือ ชุดดีเท็กเตอร์ 2 ชุดติดแยกจากกัน ทำให้เวลาในการตัดผ่านต่างกัน โดยชุดหนึ่งจะเกิดการเปลี่ยนระดับลอจิคก่อนอีกชุด จึงทำให้เกิดระดับแรงดันต่างเวลากัน ผลคือการทำงานของไอซีดิจิตอลสองตัว (ในที่นี้คือไอซี อินเวอร์เตอร์แบบชมิททริกเกอร์ IC1 กับ ดีฟริปฟร็อฟ IC2) ทำงานไม่ตรงกัน สรุปคือ เมื่อมอเตอร์หมุนวนซ้ายจะมีพัลส์ออกจากขา Q ของดีฟริปฟร็ฟตัวแรก (IC1) และมอเตอร์หมุนวนขวาจะมีพัลส์ออกจากขา Q ของดีฟริปฟร็ฟตัวที่สอง(IC2) นั่นคือสัญญาณที่ใช้นับขึ้นนับลงของระยะการกระจัดนั่นเอง จะใช้พัลส์นี้เป็นข้อมูลของ

การตรวจนับแต่พัลส์ที่ออกมา จะหมายถึงมอเตอร์หมุนในทิศทางได้การกระจัดจริง 1 หน่วยแล้ว และมีพัลส์ออก 2 ทางดังนั้นได้เป็นข้อมูล 2 ชุด ชุดแรกนับขึ้น เช่น มอเตอร์หมุนไปทาง +X ระยะทาง N ส่วนอีกชุดเป็นข้อมูลนับลง โดยจะนำข้อมูลทั้งสองชุดนี้ให้กับวงจรนับ 99 หมายถึงระยะการกระจัดในระบบทั้งหมดมี 99 หน่วย

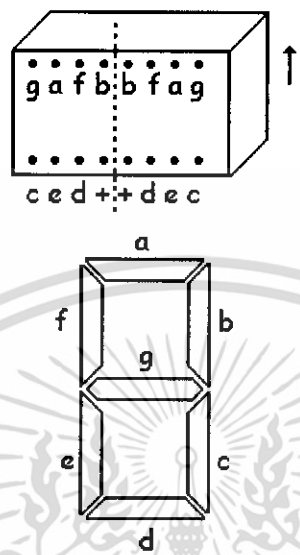
7.2.2 วงจรนับ 99



รูป 7.5 รูปวงจรนับ 99

วงจรดังรูป 7.5 จะใช้ไอซีดิจิตอลนับ 10ที่สามารถนับขึ้นและลงได้ สองตัวต่อกัน ตัวแรกเป็นหลักหน่วย(0-9) ตัวหลังเป็นหลักสิบ(10-90) คือไอซีเบอร์ 74192 สามารถนับขึ้นและลง และมีการทดและยืมเมื่อมีการนับขึ้น-ลง เกินหลัก ที่พิเศษกว่าคือ Output เป็นข้อมูลที่นับได้ เป็นไบนารี่ 4 บิตตั้งแต่ 0000 - 1001 (0-9) ซึ่งมีประโยชน์คือ สามารถนำข้อมูลสองหน่วย (หลักหน่วยและหลักสิบ) เป็นไบนารี่ 8 บิต (1 ไบต์)เข้าสู่ไมโครคอนโทรลเลอร์พอร์ต D แบบขนานได้ จะได้ข้อมูลระยะการกระจัดเป็นค่าตัวเลข 2 หลัก(8บิต) 0-99 แต่เนื่องจากมอเตอร์เคลื่อนที่สองแกนในแนวระนาบ คือแกน X และY จึงมีข้อมูล 2 ไบต์ เพื่อการประหยัดพอร์ตไม่ต้องใช้การต่อขนานเข้าอีกพอร์ต จึงต่อวงจรขยายพอร์ตโดยใช้ไอซี 74573 แล้วใช้การเขียนซอฟแวร์เพื่อเลือกว่าจะรับข้อมูลชุดแกน X หรือY ให้ใช้ขาเอ็นเนเบิลเป็นขาไมโครคอนโทรลเลอร์ A0 และ A1 เลือกแกนXและYตามลำดับ ทำให้ไมโครคอนโทรลเลอร์สามารถรู้ตำแหน่งพิกัดทั้งสองแกนและสามารถนำไปประมวลผลต่อได้

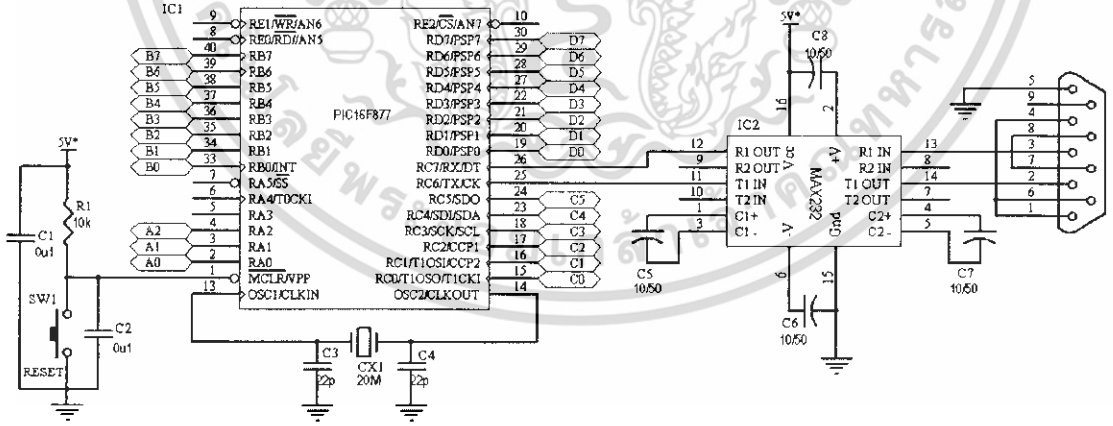
ประโยชน์อีกประการคือ สามารถต่อเข้ากับไอซี 7447 เป็นไอซีที่แปลงตัวเลข ไบนารี่ขับ4บิต เพื่อขับแอลอีดีตัวเลข 7ส่วน (7-Segment) ได้เพื่อแสดงผลเป็นตัวเลขพิกัดในฐานสิบได้ทันที เพื่อสะดวกต่อผู้ใช้ ทราบว่าขณะใดๆ เดรนอยู่ที่พิกัดใด จะใช้7-Segment ดังรูป 7.6 ซึ่งมีตัวเลข 2 หลัก เพื่อความสวยงามและง่ายต่อการจัดวาง



รูป 7.6 รูป 7-Segment ที่ใช้

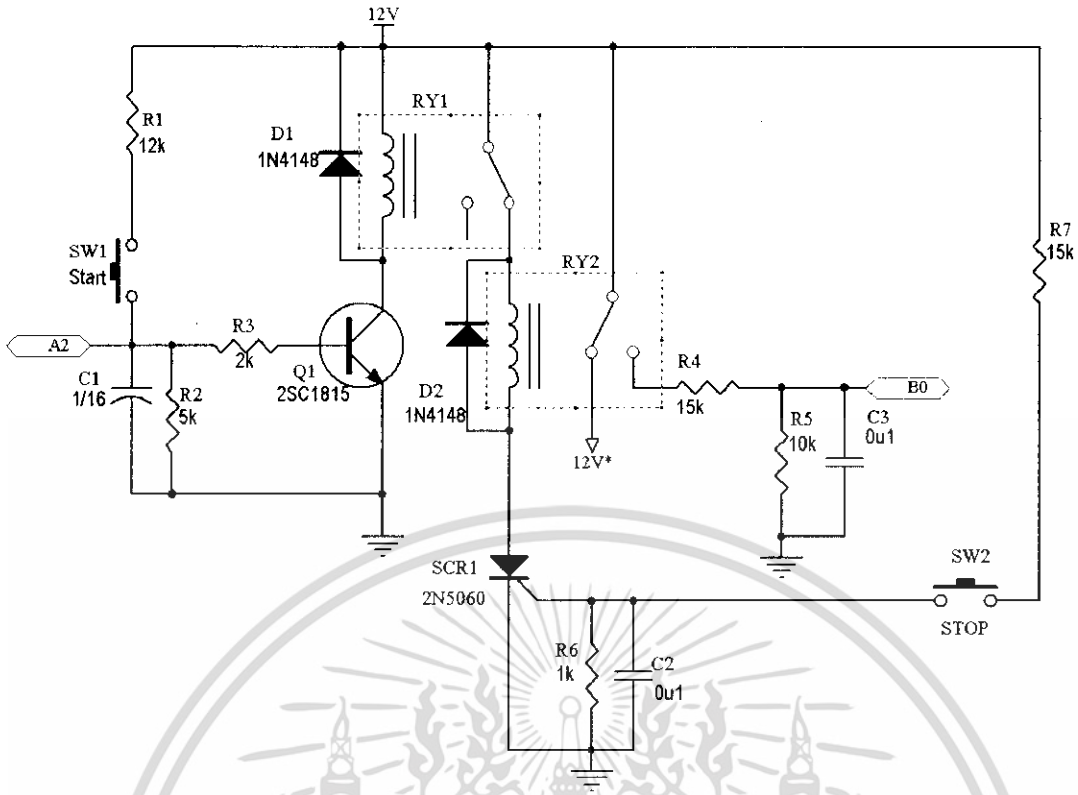
7.3 วงจรไมโครคอนโทรลเลอร์

ใช้ไฟ5V* เป็นวงจรหลักของโหมด ออโต้ดังรูปที่ 7.7 มีการต่อพอร์ตออกมาเพื่องานต่อการเชื่อต่อกับวงจรอื่นๆ และใช้ไอซี MAX232 เป็นตัวเชื่อมต่อกับพอร์ตอนุกรมของคอมพิวเตอร์



รูป 7.7 รูปวงจรไมโครคอนโทรลเลอร์

7.4 วงจรหยุดฉุกเฉินและสตาร์ตหลังการหยุดฉุกเฉิน (Emergency Stop & Start after stopping)



รูป 7.8 รูปวงจรหยุดฉุกเฉินและสตาร์ตหลังจากการหยุดฉุกเฉิน

ดังรูป 7.8 จะใช้ไฟ 12V. โดยเมื่อสวิตซ์STOP ถูกกด จะทำให้ SCR1 Turn ON ไปขับรีเลย์ RY2 ทำงาน RY2จะตัดไปที่ตำแหน่ง NO ผลคือ

1) ไม่มีไฟเลี้ยงสำหรับมอเตอร์ เพราะขา C ของ RY2 ตัดไปที่ตำแหน่ง NO ผลคือ มอเตอร์จึงหยุด โดยกำหนดให้ 12V* เป็นไฟเลี้ยงของมอเตอร์ที่ผ่านขา NC ของ RY2

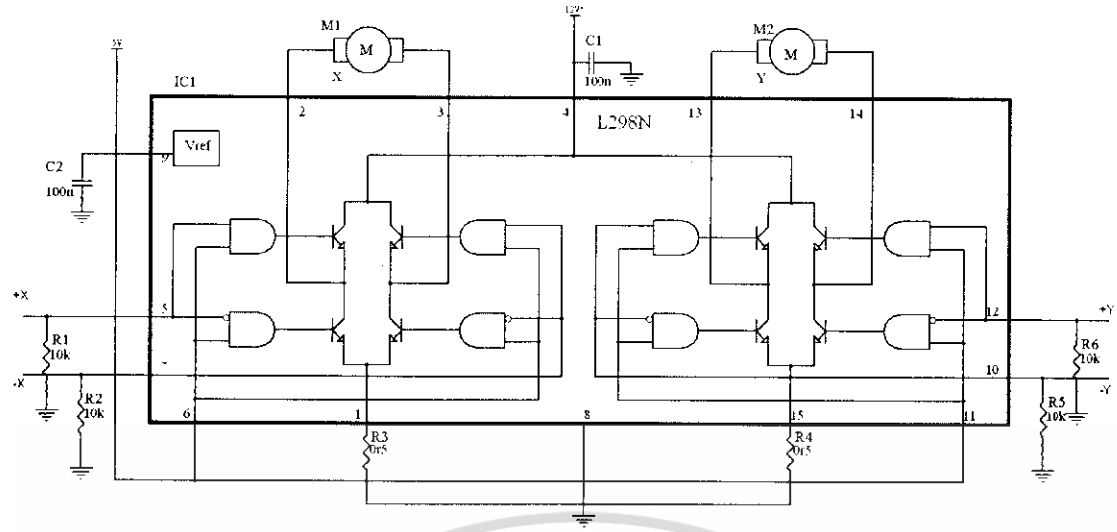
2) ไฟจากขา C ของRY2 ต่อไปที่ NO จึงมีการเปลี่ยนแปลงระดับแรงดันจาก 0 เป็น 5V (Voltage Devider จาก R4 และ R5)เกิดอินเตอร์รัพท์ RB0 ทำให้ไมโครคอนโทรลเลอร์จะทราบว่ามีการหยุดฉุกเฉินขึ้น และหยุดการสั่งงาน เพื่อรอการรีเซ็ต

และถ้าต้องการให้มอเตอร์มีไฟเลี้ยงต่อ ก็ต้องมีการกดสวิตซ์ Start เพื่อให้ไฟกระตุ้นขา เบสของทรานซิสเตอร์ และสามารถสั่งให้ Start โดยขา A2ของไมโครคอนโทรลเลอร์

7.5 วงจรขับมอเตอร์และตรวจจับการชน

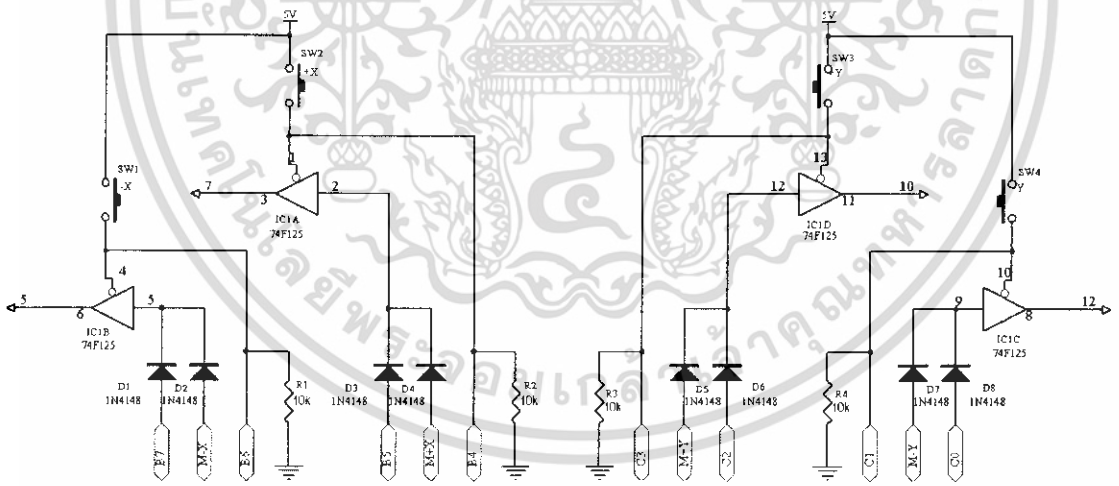
ใช้ไฟ 5V เพราะต้องการใช้ได้ทั้งสองโหมด และ12V*สำหรับขับมอเตอร์ จะใช้ไอซี L298 จำนวนสองตัวสำหรับการขับเคลื่อนมอเตอร์เลื่อนแกนทั้งสามแกน (X Y Z)

7.5.1) แกน XY



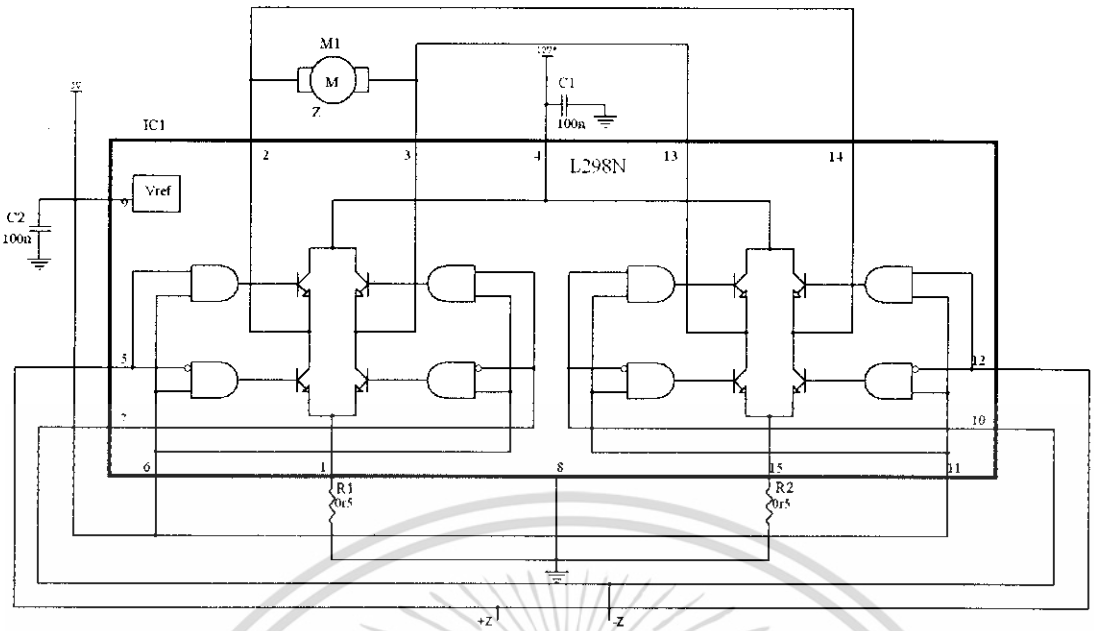
รูป 7.9 L298 ที่ใช้ขับมอเตอร์แกน XY

จะใช้มอเตอร์เกียร์ 2 ชุด โดย L298เป็นตัวขับ 1ตัว แสดงดังรูป 7.9 ซึ่งสัญญาณคำสั่งคำสั่งที่จะเข้าไปสั่งการขับเคลื่อน จะผ่านวงจรตรวจจับก่อนทอดหนึ่ง นั่นคือ มีการติดลิมิตสวิตซ์ตามแกนต่างๆ 4 ตัว เมื่อเครนชนกับสวิตซ์ จะทำให้สัญญาณจากไมโครคอนโทรลเลอร์ที่ขา B5, B7, C0 หรือ C2 ไม่สามารถไปยัง L298 เพื่อสั่งขับมอเตอร์ได้ เนื่องจากไฟต่อเข้าที่ดิสเอเบิลของไอซีบัฟเฟอร์ 74125 ขณะเดียวกันจะมีอินพุตเข้าสู่ไมโครคอนโทรลเลอร์ที่ขา B4, B6, C1 หรือ C3 เพื่อให้ไมโครคอนโทรลเลอร์สั่งหยุดทำงาน วงจรแสดงดังรูป 7.10



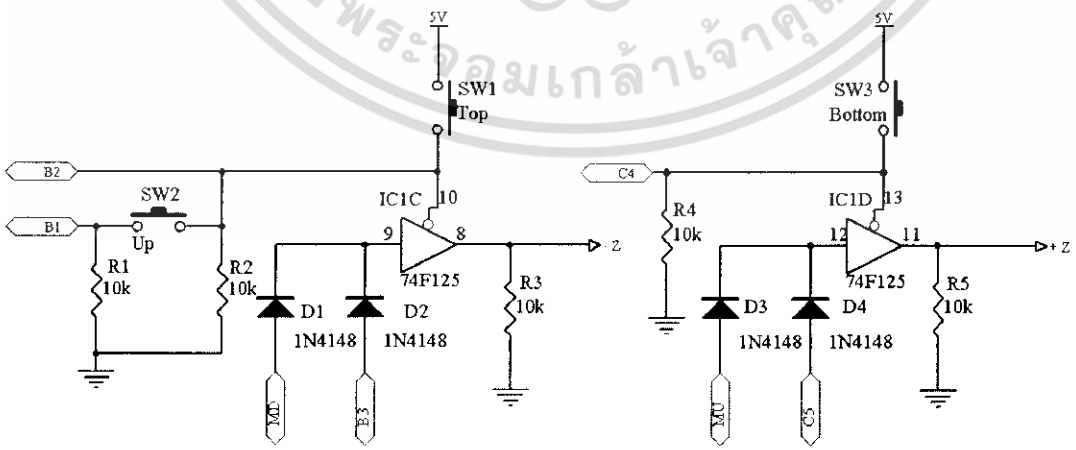
รูป 7.10 วงจรควบคุมการขับมอเตอร์แกน XY

7.5.2) แกน Z



รูป 7.11 L298 ที่ใช้ขับมอเตอร์แกน Z

จะใช้มอเตอร์เกียร์ โดย L298เป็นตัวขับ 1ตัว เพื่อการรับโหลดที่มากในแกนนี้ จึงขนาน2ชุดให้สามารถจ่ายกระแสได้มากขึ้น แสดงดังรูป 7.11 ซึ่งสัญญาณคำสั่งคำสั่งที่จะเข้าไปสั่งการขับเคลื่อนจะผ่านวงจรตรวจจับก่อนทอดหนึ่งเช่นกัน นั่นคือ มีการตรวจสอบการขึ้นถึงจุดสูงสุด และต่ำสุดของเครนโดยลิมิตสวิตซ์เช่นกัน แสดงดังรูป 7.12 เมื่อเครนขึ้นถึงจุดสูงสุด หรือลงต่ำสุด เนื่องจาก ลิมิตสวิตซ์จะต่อเข้าที่ดิสเอเบิลของไอซีบัฟเฟอร์ 74125 ทำให้สัญญาณจากไมโครคอนโทรลเลอร์ที่ขา B3 หรือ C5 ไม่สามารถไปยัง L298 เพื่อสั่งขับมอเตอร์ได้ ขณะเดียวกันจะมีอินพุตเข้าสู่ไมโครคอนโทรลเลอร์ที่ขา B2 และ C4 เพื่อให้ไมโครคอนโทรลเลอร์ทราบว่าขึ้นถึงจุดสูงสุดแล้ว หรือ ลงยังจุดต่ำสุดแล้ว เพื่อกระทำคำสั่งต่อไป และผู้ใช้สามารถกดสวิตซ์ Up เพื่อบอกกับไมโครคอนโทรลเลอร์ว่าโหลดของเสร็จสิ้นต้องการยกเครนขึ้น โดยสัญญาณนี้จะเข้าที่ขา RB1



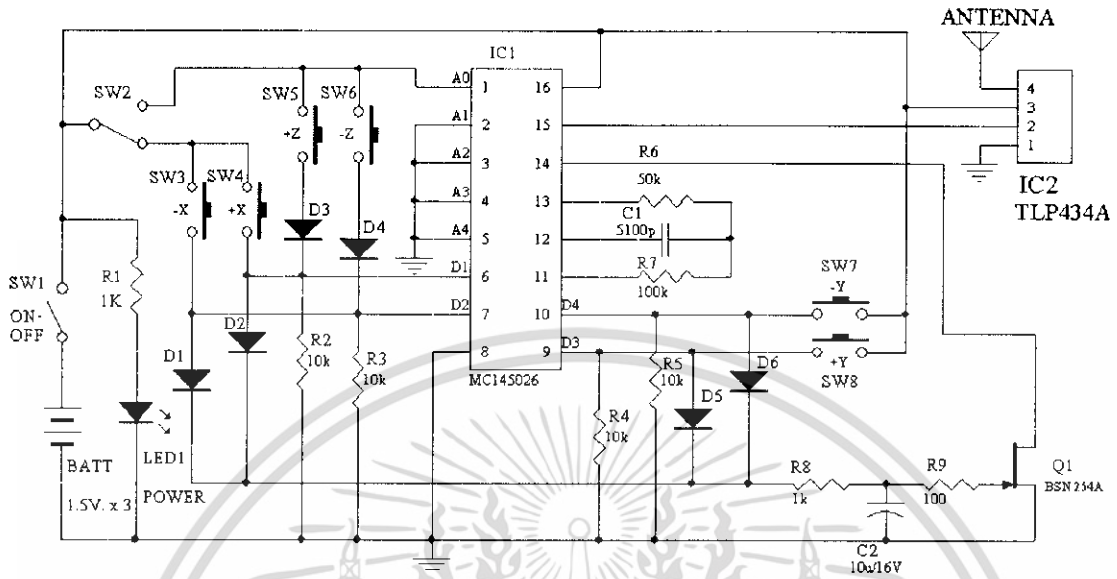
รูป 7.12 วงจรควบคุมการขับมอเตอร์แกน Z

7.6 Manual Mode

ในโหมดการควบคุมโดยผู้ใช้เองนี้จะแบ่งวงจรเป็นสองภาค คือ ภาคส่ง และ ภาครับ

7.6.1) ภาคส่ง

แสดงดังรูป 7.13



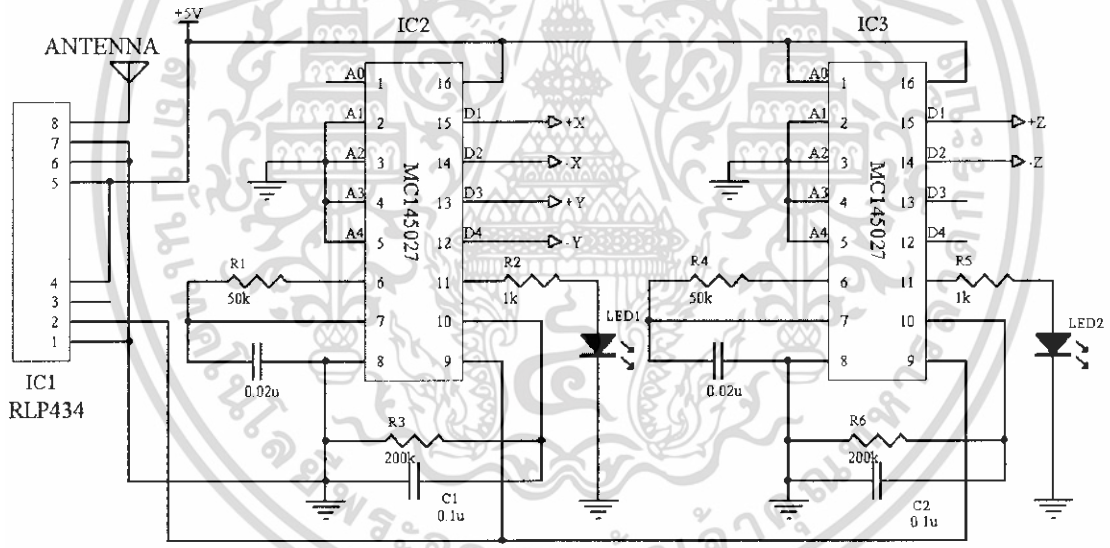
รูป 7.13 ภาคส่งของ Manual Mode

จะใช้ไอซีสร้างรหัส MC145026 โดยมีหลักการคือ การกดปุ่ม SW3~SW8 เพื่อบังคับให้ภาครับไปสั่งมอเตอร์หมุนไปทางที่ต้องการ จากวงจร จะทำให้ขา Data (D1~D4) ของไอซีสร้างรหัสเกิดเป็นลอจิก 1 ตามการกด เช่น กดปุ่ม SW4 และ SW6 ที่เหลือปล่อยไว้(จากวงจร ถ้าไม่กด จะเป็นลอจิก 0 ที่ขานั้น) รหัสที่สร้างได้จะเป็นค่า 1001 (D4,D3,D2,D1) ถ้าส่งได้ข้อมูล Binary ได้ถูกต้อง ไอซีถอดรหัสฝั่งรับ จะทำงานเพื่อให้ ขา D4 และ D1 ของไอซีถอดรหัสออกลอจิก 1 ซึ่งจะนำค่าลอจิกนี้ไปสั่งให้มอเตอร์ทำงานได้ ตัวไอซีถอดรหัสจะกล่าวในหัวข้อต่อไป แต่เนื่องจาเราต้องการควบคุม 6 ทิศทาง (+X,-X,+Y,-Y,+Z และ -Z) ขาอินพุทของมอเตอร์จะต้องมี6เส้น แต่ไอซีมีเพียง 4 ข้อมูล จำเป็นต้องทำการเพิ่มรหัส โดยใช้ความสามารถของการแยกที่อยู่ (Address) ของคู่ไอซีสร้างรหัสและถอดรหัส ซึ่งจะมี 5 รหัส Address (A0~A4) หลักการคือ ไอซีถอดรหัส จะถอดรหัสเฉพาะข้อมูลที่มี Address ทั้ง 5 บิตตรงกันเท่านั้น ดังนั้น ถ้าเราใช้ไอซีถอดรหัส สองตัวที่ตั้งค่า Address ต่างกัน เช่น ตัวแรกตั้ง Address เป็น *0000 กับตัวที่สองตั้ง Addressเป็น 10000 (A0~A4) (ค่า * หมายถึง High Impedanze เนื่องจากไอซีดังกล่าวสามารถแยกแยะ Trinary Address ได้ แต่ข้อมูลต้องเป็น Binary เท่านั้น) จะทำให้ค่าทางเอ๊าท์พุทของตัวถอดรหัสแบ่งเป็น 8 ข้อมูล (สั่งได้ 8 คำสั่ง) โดยที่ทางตัวสร้างรหัสจะต้องตั้งรหัส Address เป็น *0000 และ 10000 (A0~A4) เพื่อทำให้ตัวถอดรหัสตัวแรกและตัวที่สองทำงานตามลำดับ เราจะใช้หลักการนี้ แต่ใช้ตัวสร้างรหัสตัวเดียว อาศัยการปรับ SW2 เพื่อให้ Address (A0) เปลี่ยนไป ทำให้ข้อมูลที่สร้างจากการกด SW5 กับ SW6 มีAddress ที่แตกต่างจากการกด SW2 ,SW3, SW7, SW8 โดย SW5 กับ SW6 จะใช้สำหรับ ควบคุมแกน Z ดังนั้น เมื่อผู้ใช้ต้องการควบคุมแกน Z จะต้องปรับ SW2 ไปด้านบนตามวงจรรูป7.13 ทำให้ Address ขณะนั้นเป็น 10000 สามารถใช้ปุ่ม

SW5 กับ SW6 ได้แต่ไม่สามารถควบคุมแกน X หรือ Y ได้ เนื่องจากไอซีถอดรหัสที่ตั้ง Address เป็น 10000 ใช้ควบคุมมอเตอร์แกน Z เท่านั้น (ไอซีถอดรหัสที่ตั้ง Address เป็น *0000 ใช้ควบคุมมอเตอร์แกน X และ Y) สรุปคือ ในการส่งข้อมูลแต่ละชุดประกอบด้วย รหัสข้อมูล 9 บิตอนุกรม (Trinary Address 5 บิต+Binary Data 4บิต) สร้างออกจากขา 15 ของไอซีสร้างรหัส เพื่อส่งให้ ไอซี RFมอดดูเลต(TLP434A) เพื่อรวมเอาข้อมู,เข้ากับคลื่นพาหะ และส่งออกไปยังเสาอากาศ ในการสร้างรหัสจำเป็นที่จะต้องให้ขา 14 ของไอซีสร้างรหัสต่อลอจิค 0 ดังนั้น จึงใช้เฟต Q1 ต่อไว้เพื่อประหยัดพลังงานแบตเตอร์รี่ โดยเมื่อใดก็ตามที่มีการกดปุ่ม SW3~SW8 จะกระตุ้นขาเกตของ Q1ผ่านทาง D1~D6 ทำให้เฟต Turn on ขา14ของไอซีสร้างรหัสจึงเสมือนต่อกราวด์ ทำให้เกิดรหัสส่ง(ส่งอย่างต่อเนื่องไปเรื่อยๆทีละชุดจนกว่าจะปล่อยการกด)ไปยัง TLP434A ได้ ส่วน R8, C2, R9 จะทำหน้าที่หน่วงเวลาเล็กน้อย เพื่อที่ภายหลังกดปุ่มแล้ว ยังคงส่งค่าข้อมูล 0000 ต่ออีกระยะหนึ่ง ไม่หยุดทันทีที่ปล่อย เนื่องจากถ้าข้อมูลยังมาไม่ครบชุดที่ไอซีถอดรหัสจะเกิดการค้างค่าของเอ๊าท์พุท หรือเกิดการรบกวนขึ้น

7.6.2) ภาครับ

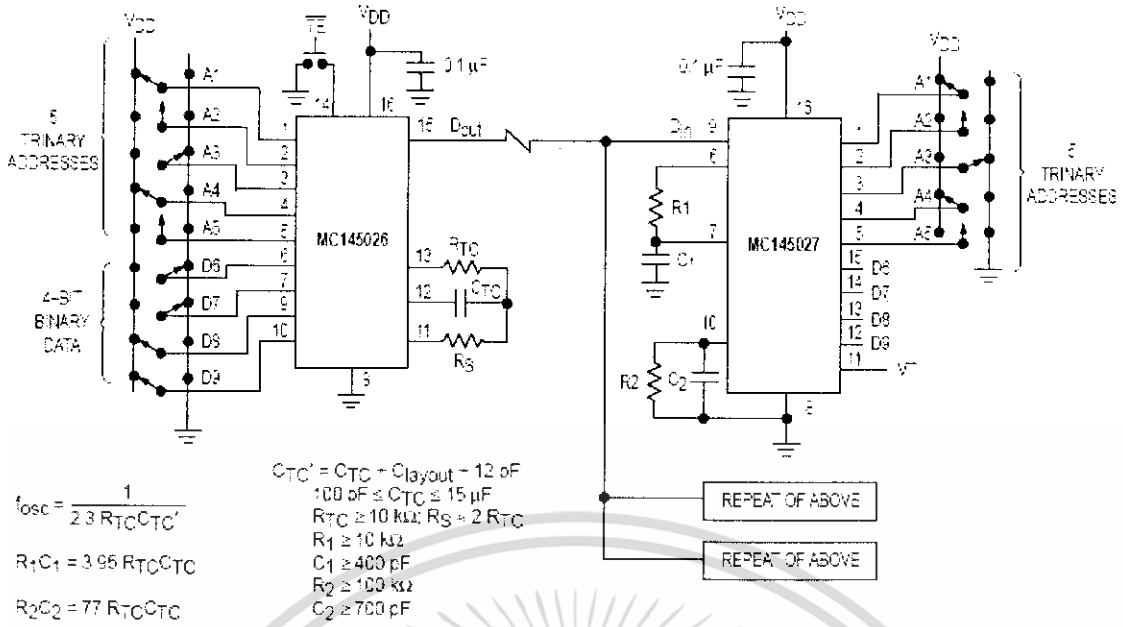
แสดงดังรูป 7.14



รูป 7.14 ภาครับของ Manual Mode

เมื่อข้อมูลเข้าสู่เสาอากาศและมายังไอซี Demodulator RLP434 ที่ภาครับ จะกระทำการแยกเอาคลื่นพาหะออก เหลือเฉพาะคำรหัสจากการกด เพื่อเข้าสู่ ขา9ของไอซีถอดรหัส MC145027 เพื่อทำการถอดรหัส ตามAddressที่ภาคส่งกำหนดมาพร้อมกับData ภายหลังจากการถอดรหัส จะได้คำสั่งที่สำหรับสั่งให้วงจร Motor Driver ขับมอเตอร์ในแกนต่างๆต่อไป LED1 และ LED2 แสดงสถานะว่าข้อมูลที่รับมานั้นตรงกับไอซีถอดรหัสตัวใด

โดยทั่วไปการใช้งานคู่ไอซี Enceder – Decoder แสดงดังรูป 7.15



รูป 7.15 การใช้งานคู่ไอซี Enceder – Decoder

ค่า R1, R2, C1, C2, $C_{TC'}$,R_{TC} , R_S ใช้สำหรับคำนวณค่าความถี่ของข้อมูลที่สร้างตามสมการเบื้องต้น เมื่อขา 14 ต่อ ลอจิค 0 แต่ในการใช้งานมักจะเลือกใช้ค่าต่างๆ แสดงดังตารางที่ 7.1

Example R/C Values (All Resistors and Capacitors are ± 5%)

($C_{TC'} = C_{TC} + 20$ pF)

f_{osc} (kHz)	R_{TC}	$C_{TC'}$	R_S	R_1	C_1	R_2	C_2
362	10 k	120 pF	20 k	10 k	470 pF	100 k	910 pF
181	10 k	240 pF	20 k	10 k	910 pF	100 k	1800 pF
88.7	10 k	490 pF	20 k	10 k	2000 pF	100 k	3900 pF
42.6	10 k	1020 pF	20 k	10 k	3900 pF	100 k	7500 pF
21.5	10 k	2020 pF	20 k	10 k	8200 pF	100 k	0.015 µF
8.53	10 k	5100 pF	20 k	10 k	0.02 µF	200 k	0.02 µF
1.71	50 k	5100 pF	100 k	50 k	0.02 µF	200 k	0.1 µF

ตาราง 7.1 ตารางแสดงค่าความถี่ของข้อมูลที่สร้าง
เมื่อเลือกค่า R1, R2, C1, C2, $C_{TC'}$,R_{TC} , R_S ต่างๆกัน

ในโครงงานนี้ จะใช้ความถี่ข้อมูลต่ำสุดที่ 1.71kHz เพื่อให้ได้ข้อมูลที่ครบถ้วนสมบูรณ์และป้องกันสัญญาณรบกวน

บทที่ 8.

การออกแบบ แนวคิดและการหลักการทำงานของโปรแกรม

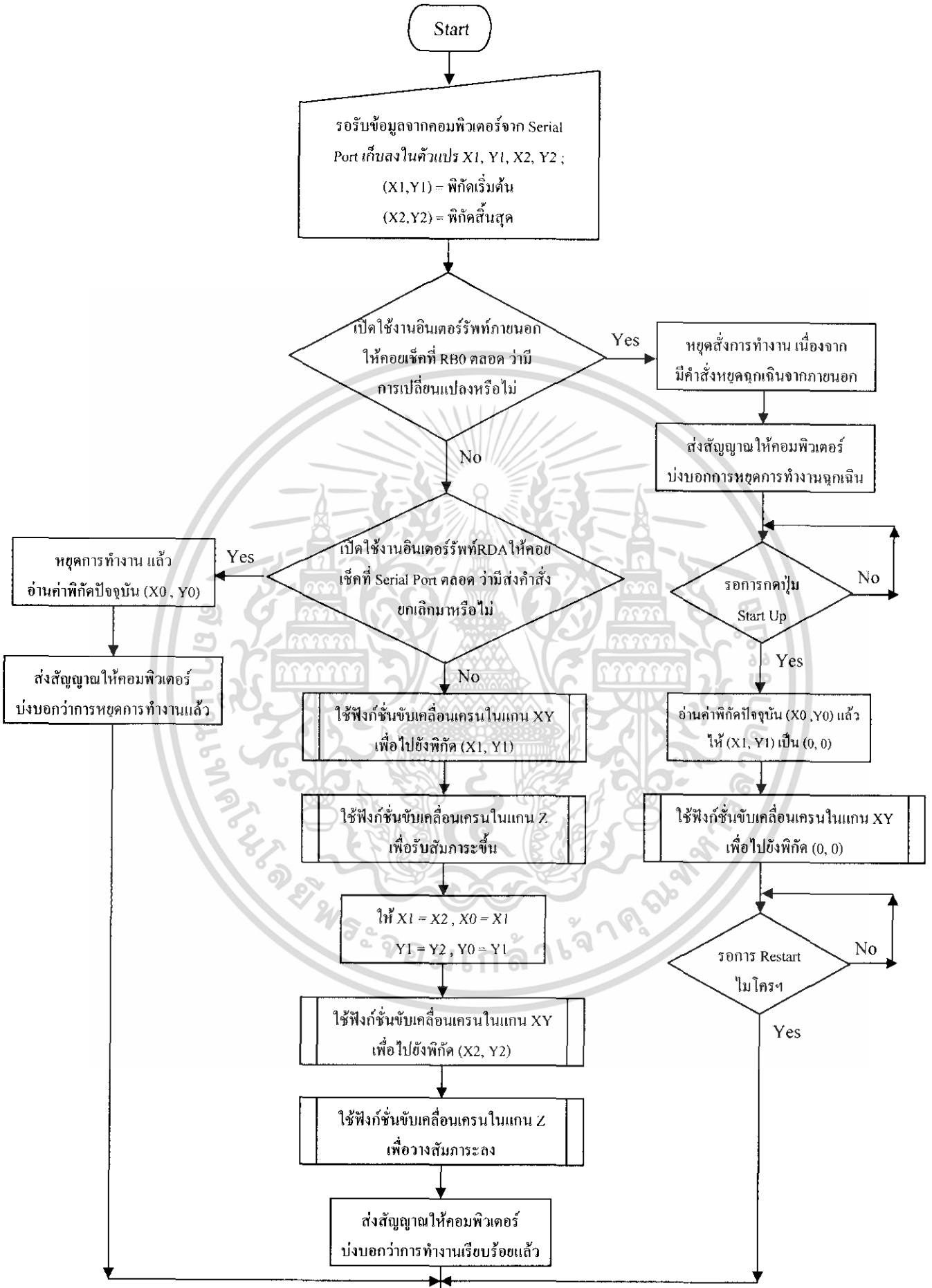
โปรแกรมที่ใช้จะมีอยู่สองส่วนคือ ส่วนของไมโครคอนโทรลเลอร์ และ ส่วนในคอมพิวเตอร์ โดยโค๊ดทั้งหมดจะรวบรวมอยู่ในส่วนของภาคผนวก

8.1 ไมโครคอนโทรลเลอร์

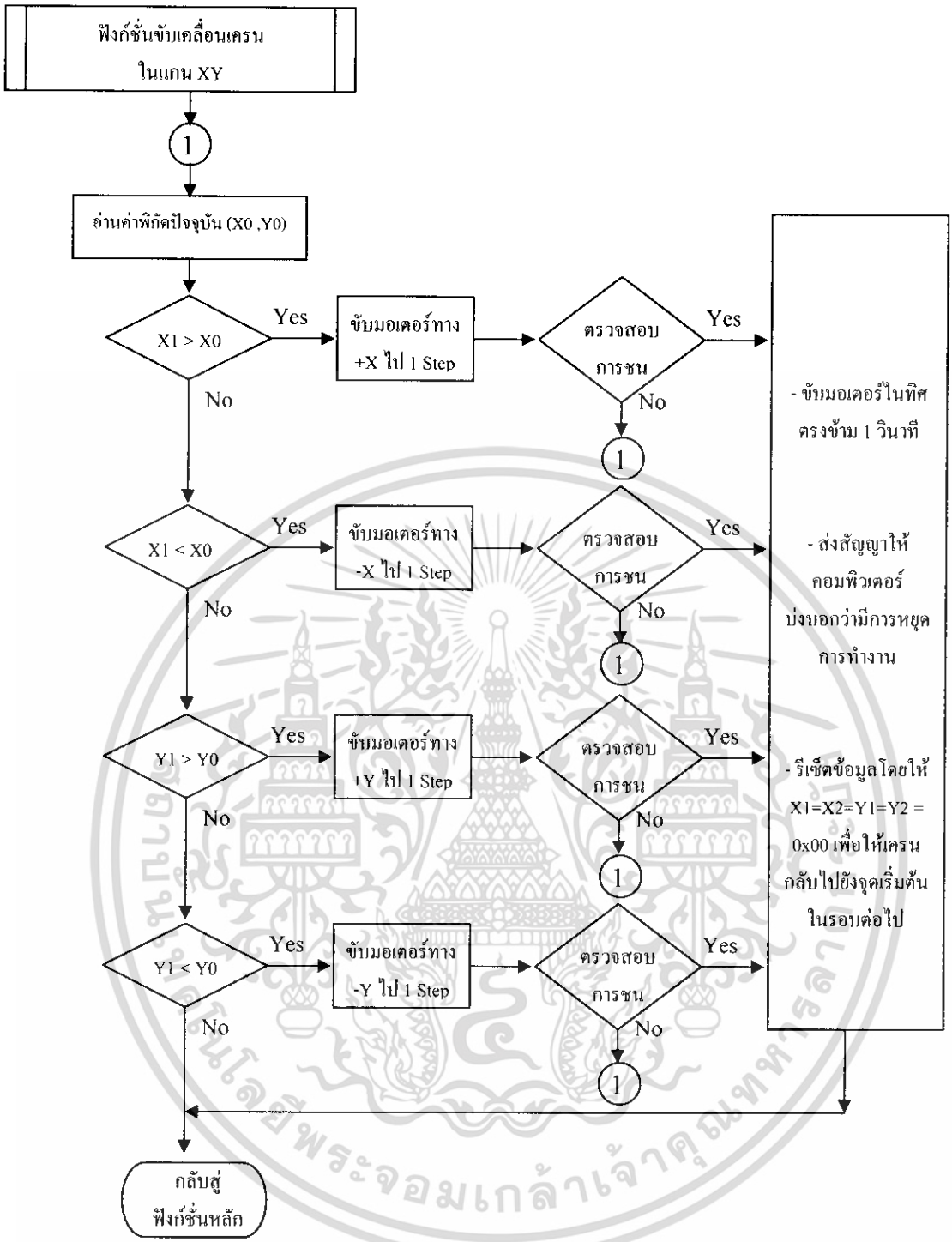
จะใช้ภาษาซี ซึ่งมีโฟร์ลชาร์ดดังรูป 8.1

8.2 คอมพิวเตอร์

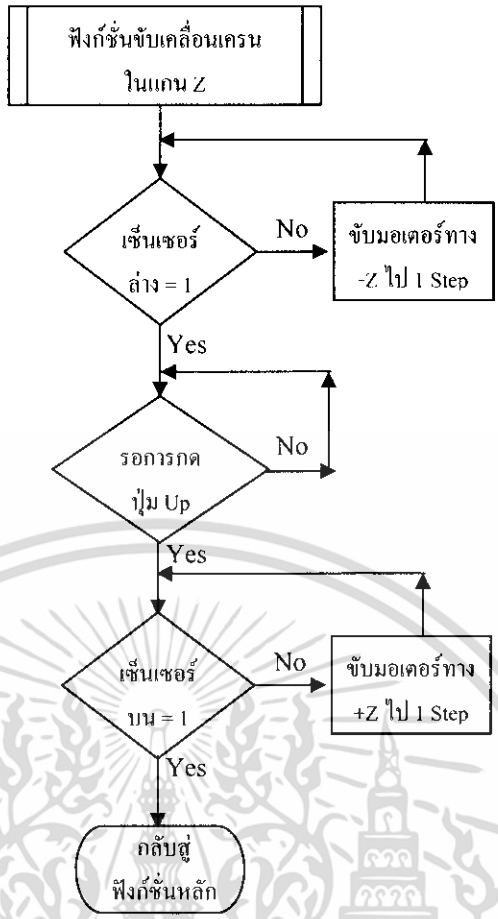
จะใช้ Visual Basic เขียน ได้รูปแบบโปรแกรมที่สำเร็จดังรูปที่ 8.2
หน้าที่หลักของโปรแกรมคือ รับค่าข้อมูลพิกัด 4 ค่าจากผู้ใช้ คือ X1, Y1, X2, Y2 เพื่อไปพล็อตพิกัดในกรอบสี่เหลี่ยมด้านล่าง และส่งข้อมูลไปยังพอร์ตอนุกรมให้กับไมโครคอนโทรลเลอร์
เมื่อมีการ Click ปุ่ม OK โปรแกรมจะส่งค่า AA ออกไปยังพอร์ตอนุกรมเพื่อบอกกับไมโครคอนโทรลเลอร์ว่าพร้อมที่จะส่งข้อมูลแล้ว จากนั้นจะคอยเช็คที่พอร์ตอนุกรมทุกๆ 200 มิลลิวินาทีว่ามีค่าใดส่งมาจากไมโครคอนโทรลเลอร์บ้าง โดยตั้งค่าไว้ดังนี้
เมื่อได้รับค่า H1 จะส่งค่า X1 ออกไป
เมื่อได้รับค่า H2 จะส่งค่า Y1 ออกไป
เมื่อได้รับค่า H3 จะส่งค่า X2 ออกไป
เมื่อได้รับค่า H4 จะส่งค่า Y2 ออกไป
เมื่อได้รับค่า H5 จะส่งหมายถึงได้รับข้อมูลครบแล้ว พร้อมปฏิบัติงาน
เมื่อได้รับค่า H6 จะส่งหมายถึงปฏิบัติงานเรียบร้อยแล้ว พร้อมรับข้อมูลพิกัดต่อไป
เมื่อได้รับค่า H7 จะส่งหมายถึงยกเลิกปฏิบัติงานและรีเซ็ตข้อมูลพิกัด



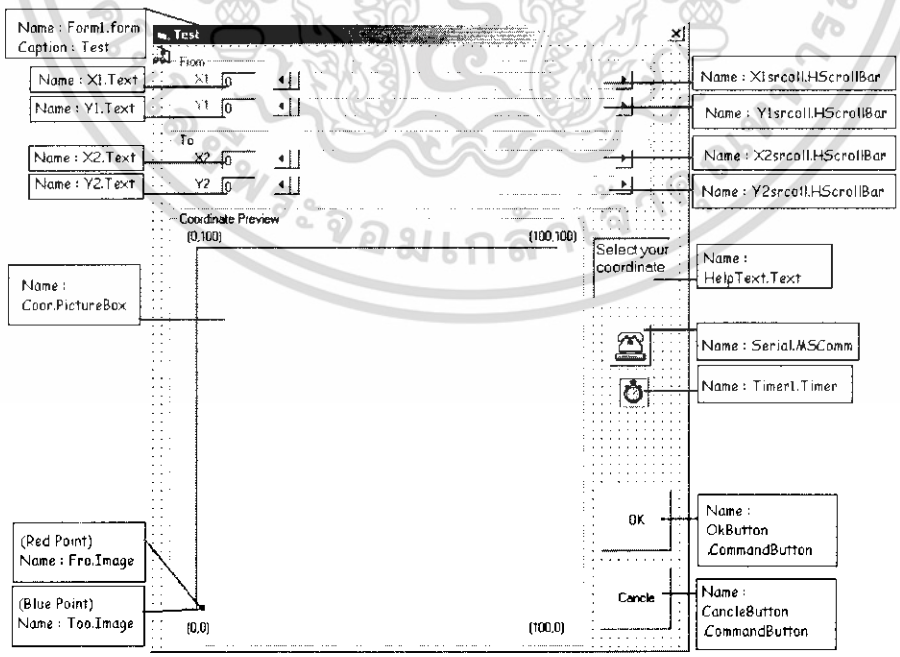
รูป 8.1ก โฟร์ลชาร์ตแสดงการทำงานของโปรแกรมหลัก



รูป 8.1ข โฟร์ลชาร์ดแสดงการทำงานของฟังก์ชั่น
ขับเคลื่อนเครนในแกน XY



รูป 8.1ค โฟร์ลชาร์ดแสดงการทำงานของฟังก์ชั่น
ขับเคลื่อนเครนในแกน Z



รูป 8.2 ลักษณะโปรแกรมที่ใช้ในคอมพิวเตอร์ และการจัดวางคอนโทรลต่างๆ

บทที่ 9

ผลการทดลอง สรุป และวิเคราะห์

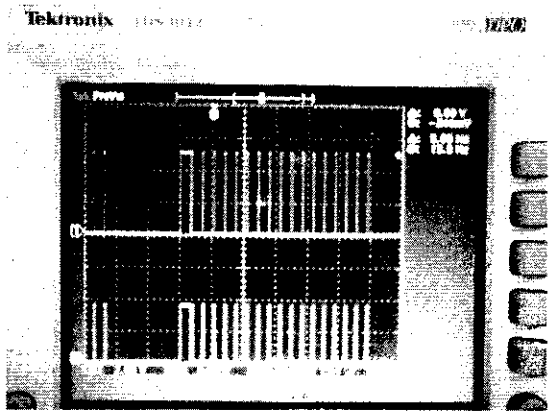
9.1 ผลการทดลอง

9.1.1 การรับส่งข้อมูลระบบ Manual

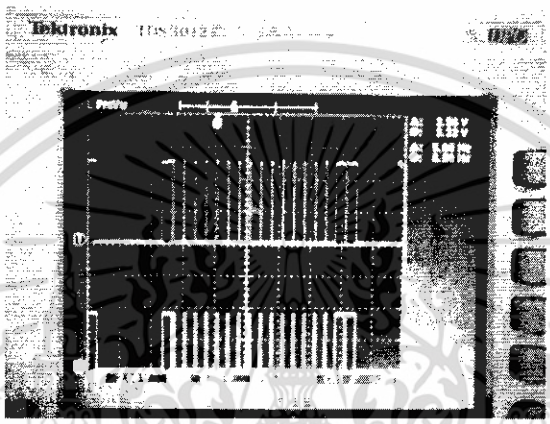
รูปสัญญาณของข้อมูลที่ฝั่งส่งและฝั่งรับของระบบ Manual แสดงดังตารางที่ 9.1

Address = *0000	D1	D2	D3	D4
รูป 9.1	0	0	0	0
รูป 9.2	0	0	0	1
รูป 9.3	0	0	1	0
รูป 9.4	0	0	1	1
รูป 9.5	0	1	0	0
รูป 9.6	0	1	0	1
รูป 9.7	0	1	1	0
รูป 9.8	0	1	1	1
รูป 9.9	1	0	0	0
รูป 9.10	1	0	0	1
รูป 9.11	1	0	1	0
รูป 9.12	1	0	1	1
รูป 9.13	1	1	0	0
รูป 9.14	1	1	0	1
รูป 9.15	1	1	1	0
รูป 9.16	1	1	1	1
Address = 10000	D1	D2	D3	D4
รูป 9.17	0	0	0	0
รูป 9.18	0	1	0	0
รูป 9.19	1	0	0	0
รูป 9.20	1	1	0	0
รูป 9.21	ไม่มีการส่งข้อมูล			

ตาราง 9.1 ตารางแสดงรูปสัญญาณของข้อมูลที่ฝั่งส่งและฝั่งรับของระบบ Manual

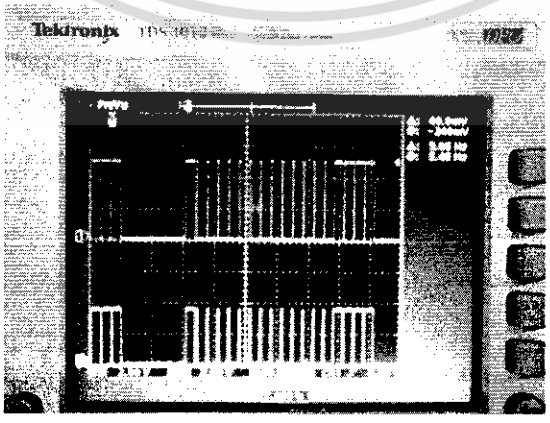


รูป 9.1 ข้อมูล = *0000 0000

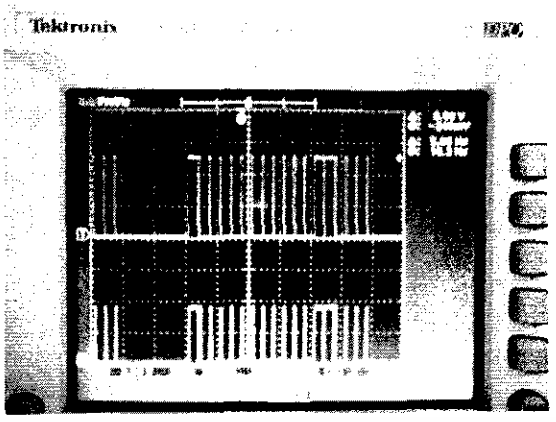


รูป 9.2 ข้อมูล = *0000 0001

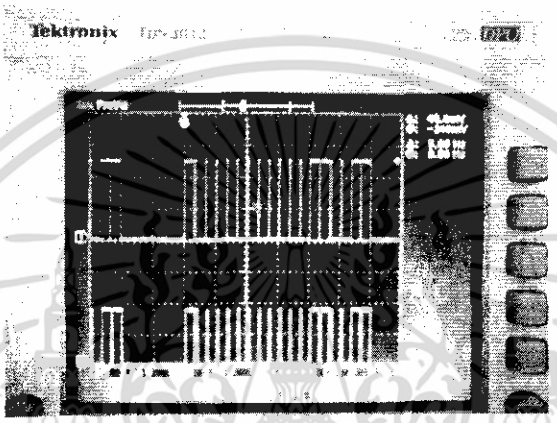
รูป 9.3 ข้อมูล = *00000 0010



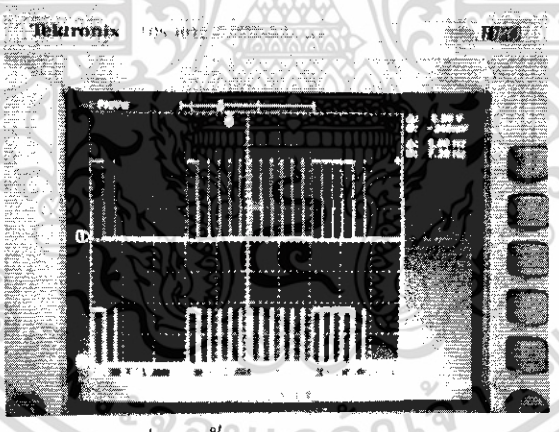
รูป 9.4 ข้อมูล = *0000 0011



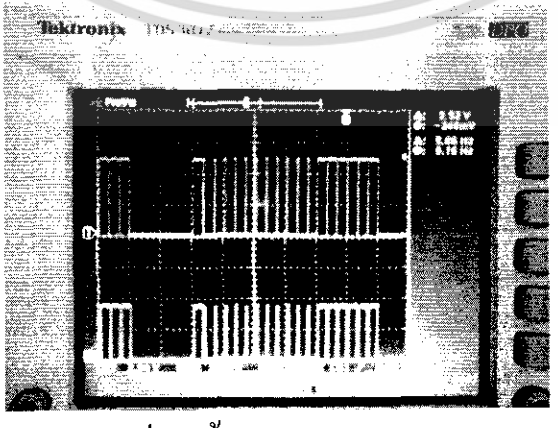
รูป 9.5 ข้อมูล = *0000 0100



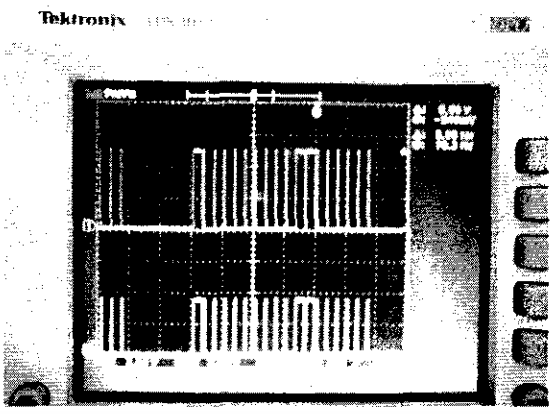
รูป 9.6 ข้อมูล = *0000 0101



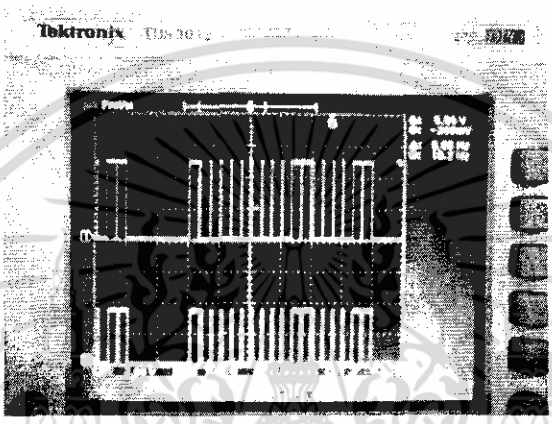
รูป 9.7 ข้อมูล = *0000 0110



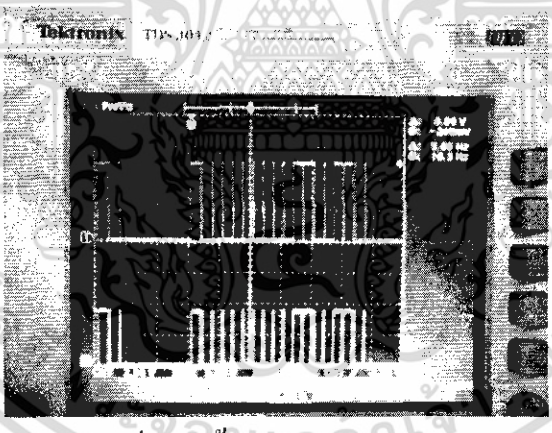
รูป 9.8 ข้อมูล = *0000 0111



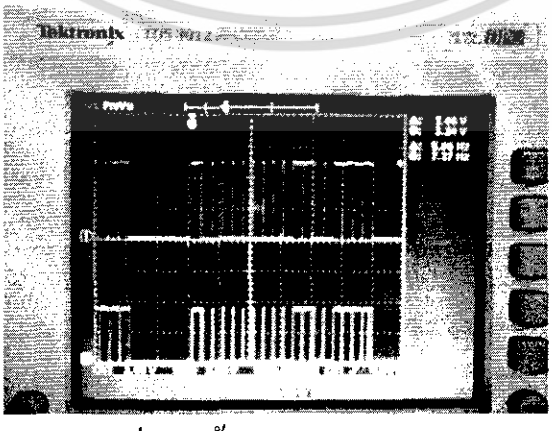
รูป 9.9 ข้อมูล = *0000 1000



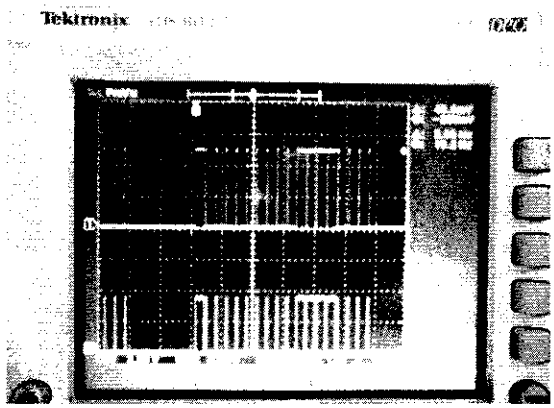
รูป 9.10 ข้อมูล = *0000 1001



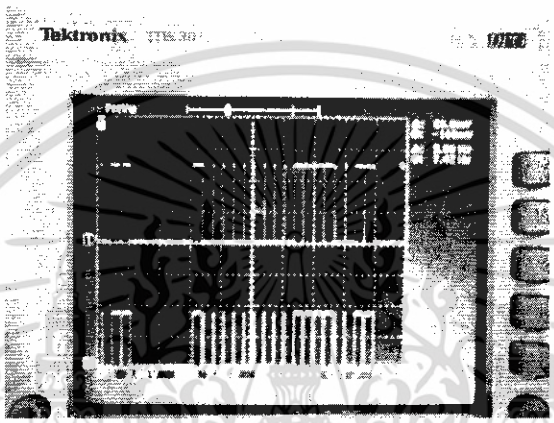
รูป 9.11 ข้อมูล = *0000 1010



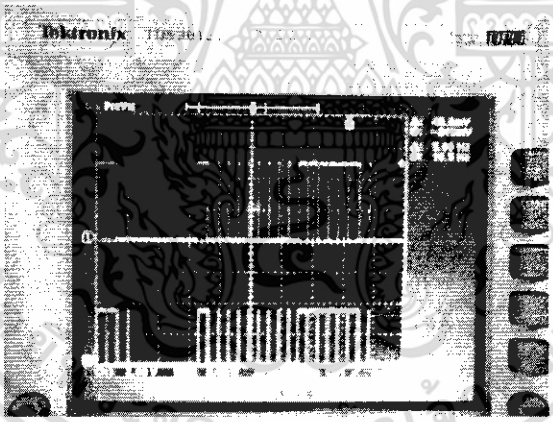
รูป 9.12 ข้อมูล = *0000 1011



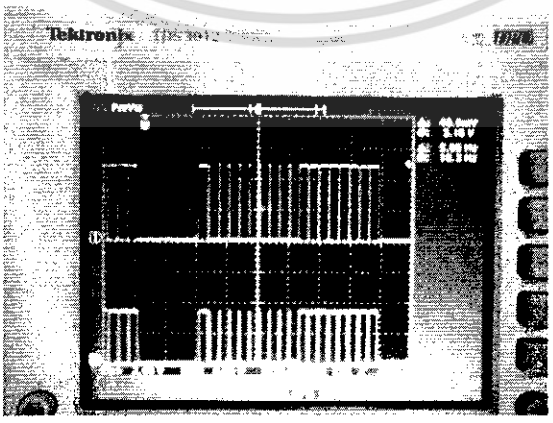
รูป 9.13 ข้อมูล = *0000 1100



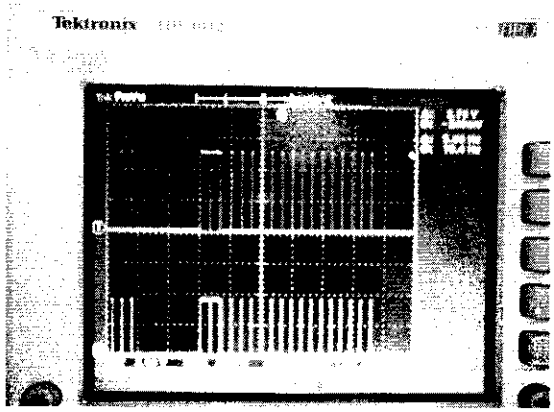
รูป 9.14 ข้อมูล = *0000 1101



รูป 9.15 ข้อมูล = *0000 1110

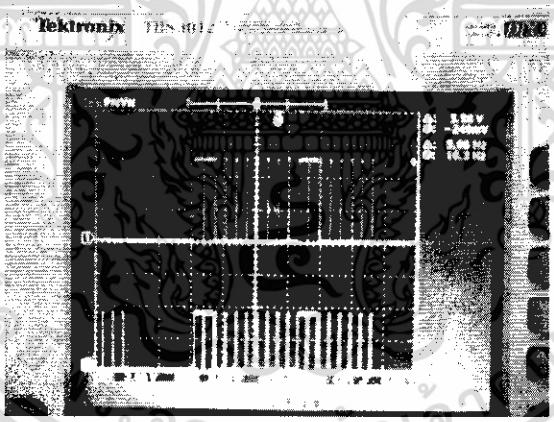


รูป 9.16 ข้อมูล = *0000 1111

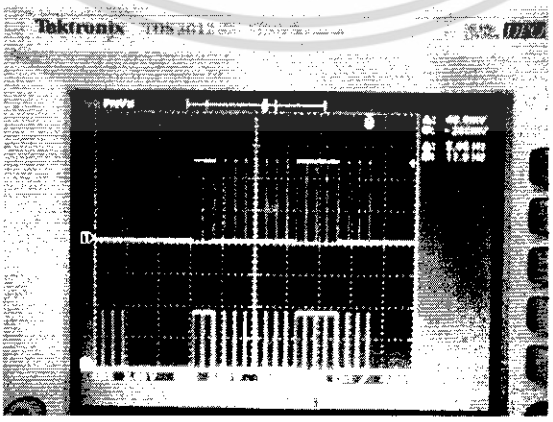


รูป 9.17 ข้อมูล = 10000 0000

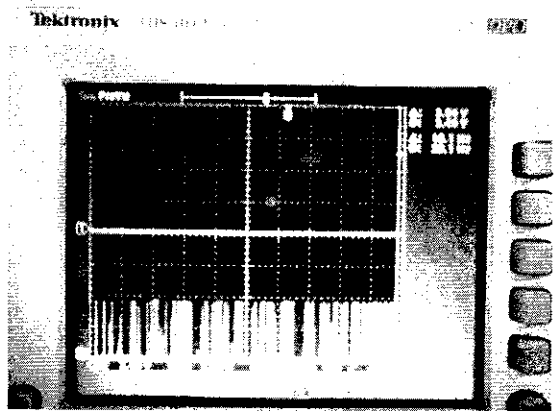
รูป 9.18 ข้อมูล = 10000 0100



รูป 9.19 ข้อมูล = 10000 1000



รูป 9.20 ข้อมูล = 10000 1100



รูป 9.21 ไม่มีการส่งข้อมูล

โดยทุกรูปให้ CH1(บน) = ข้อมูลฝั่งส่ง และ CH2(ล่าง) = ข้อมูลฝั่งรับ

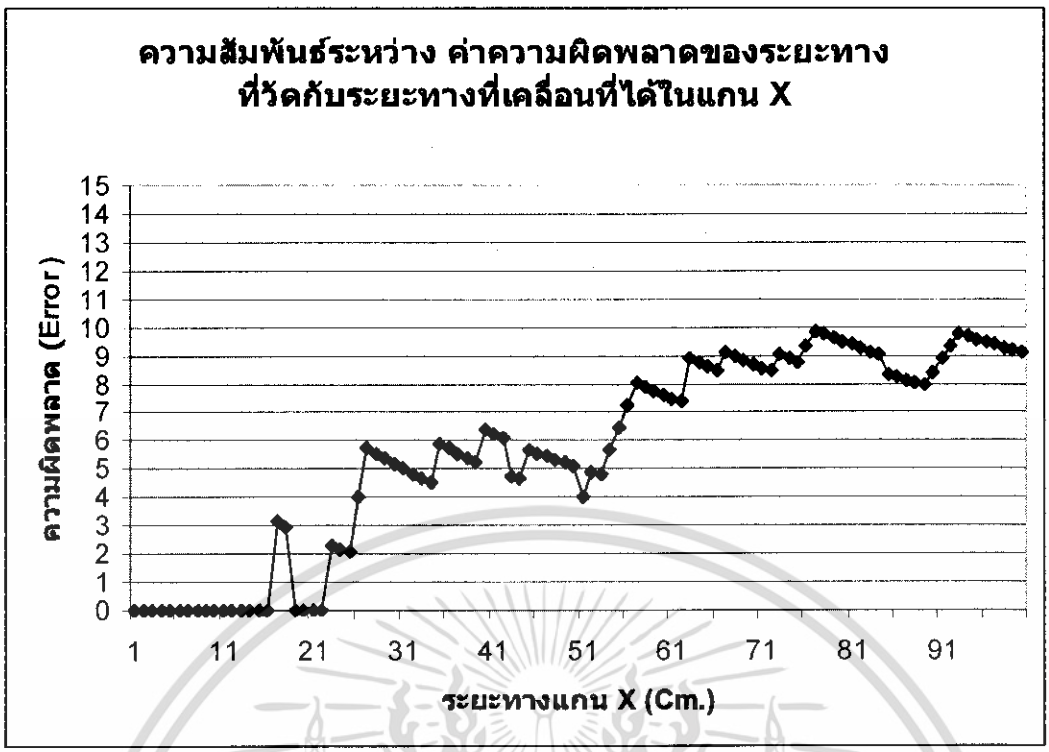
X = 10 ms/Div, Y = 2.00 V/Div ระยะทดลอง 1 เมตร

9.1.2 การวัดระยะที่เคลื่อนที่ได้

แสดงดังตารางที่ 9.2 และ 9.3 และผลดังรูปที่ 9.22 และ 9.23

X (Cm.)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
วัด (Cm.)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Error %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
X (Cm.)	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
วัด (Cm.)	15	17	18	18	19	20	21	23	24	25	26	28	29	30	31
Error %	0	3.1	2.9	0	0	0	0	2.3	2.2	2.1	4	5.8	5.6	5.4	5.2
X (Cm.)	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
วัด (Cm.)	32	33	34	35	36	37	38	39	40	42	43	44	44	45	47
Error %	5	4.8	4.7	4.5	5.9	5.7	5.6	5.4	5.3	6.4	6.3	6.1	4.8	4.7	5.7
X (Cm.)	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
วัด (Cm.)	48	49	50	51	52	52	54	55	56	58	59	61	62	63	64
Error %	5.6	5.4	5.3	5.2	5.1	4	4.9	4.8	5.7	6.5	7.3	8	7.9	7.8	7.6
X (Cm.)	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
วัด (Cm.)	65	66	68	69	70	71	72	73	74	75	76	77	79	80	81
Error %	7.5	7.4	8.9	8.7	8.6	8.5	9.1	9	8.8	8.7	8.6	8.5	9	8.9	8.8
X (Cm.)	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
วัด (Cm.)	82	84	85	86	87	88	89	90	91	91	92	93	94	95	97
Error %	9.3	9.9	9.7	9.6	9.5	9.4	9.3	9.1	9	8.3	8.2	8.1	8	8	8.4
X (Cm.)	90	91	92	93	94	95	96	97	98	99					
วัด (Cm.)	98	100	101	102	103	104	105	106	107	108					
Error %	8.9	9.3	9.8	9.7	9.6	9.5	9.4	9.3	9.2	9.1					

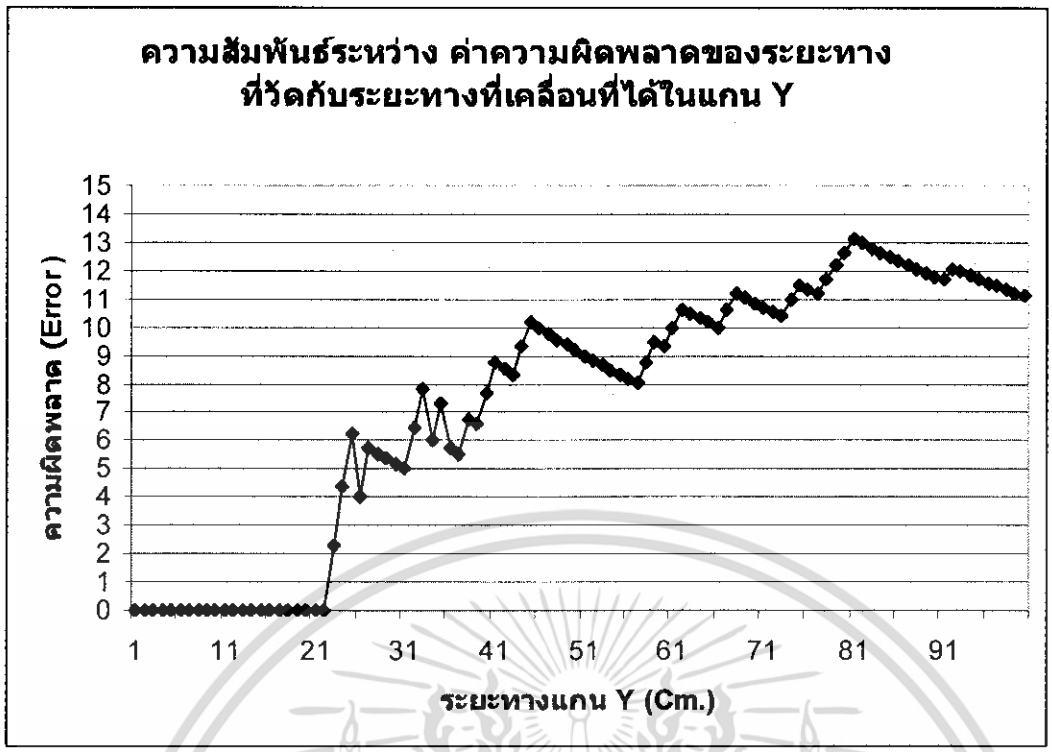
ตาราง 9.2 ตารางแสดงความผลจากการวัดเมื่อเทียบกับค่าที่อ้างอิง และค่าความคลาดเคลื่อนเมื่อเครนเคลื่อนที่ในแกน X เป็นระยะทางต่างๆ



รูป 9.22 กราฟแสดงความสัมพันธ์ระหว่าง ค่าความผิดพลาดของระยะทางที่วัดกับระยะทางที่เคลื่อนที่ได้ในแกน X

Y (Cm.)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
วัด (Cm.)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Error %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y (Cm.)	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
วัด (Cm.)	15	16	17	18	19	20	21	23	24	26	26	28	29	30	31
Error %	0	0	0	0	0	0	0	2.3	4.3	6.3	4	5.8	5.6	5.4	5.2
Y (Cm.)	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
วัด (Cm.)	32	33	35	35	37	37	38	40	41	42	44	45	46	47	49
Error %	5	6.5	7.8	6.1	7.4	5.7	5.6	6.8	6.6	7.7	8.8	8.5	8.3	9.3	10
Y (Cm.)	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
วัด (Cm.)	50	51	52	53	54	55	56	57	58	59	60	61	62	64	65
Error %	10	9.8	9.6	9.4	9.2	9	8.8	8.7	8.5	8.3	8.2	8	8.8	9.5	9.3
Y (Cm.)	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
วัด (Cm.)	66	68	69	70	71	72	73	75	76	77	78	79	80	81	83
Error %	10	11	10	10	10	10	11	11	11	11	11	11	10	11	11
Y (Cm.)	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
วัด (Cm.)	84	85	86	88	89	91	92	93	94	95	96	97	98	99	100
Error %	11	11	12	12	13	13	13	13	13	13	12	12	12	12	12
Y (Cm.)	90	91	92	93	94	95	96	97	98	99					
วัด (Cm.)	101	102	103	104	105	106	107	108	109	110					
Error %	12	12	12	12	12	12	11	11	11	11					

ตาราง 9.3 ตารางแสดงความผลจากการวัดเมื่อเทียบกับค่าที่อ้างอิง และค่าความคลาดเคลื่อนเมื่อเครนเคลื่อนที่ในแกน Y เป็นระยะทางต่างๆ



รูป 9.23 กราฟแสดงความสัมพันธ์ระหว่าง ค่าความผิดพลาดของระยะทางที่วัดกับระยะทางที่เคลื่อนที่ได้ในแกน Y

9.2 สรุป

ลักษณะโครงสร้างภายนอกแสดงโดยรูปที่ 9.24

รูป 9.24 ลักษณะโครงสร้างภายนอกของโครงงาน

- การทำงานของโครงงานนี้คือ ในระบบ Auto ผู้ใช้สามารถจะระบุพิกัด (X1,Y1) คือ พิกัดเริ่มต้น และ (X2,Y2)คือ พิกัดปลายทางจากซอฟแวร์ในคอมพิวเตอร์ โดยข้อมูลพิกัดจะถูกส่งไปยัง พีไอซี

ไมโครคอนโทรลเลอร์เพื่อควบคุมเครน ผ่านทางพอร์ทอนุกรม ในการทำงาน เครนจะเริ่มเคลื่อนที่ไปยังพิกัด (X1,Y1) เพื่อรอการยกสัมภาระขึ้น จากนั้น จะเคลื่อนที่ไปยังพิกัด (X2,Y2) เพื่อรอการนำสัมภาระลง ผู้ใช้สามารถยกเลิกการทำงานของเครนได้ขณะปฏิบัติงาน ส่วนระบบ Manual ผู้ใช้จะบังคับให้เครนเคลื่อนที่ไปยังพิกัดต่างๆ รวมทั้งการขึ้นลงของเครนผ่านทางรีโมตคอนโทรลไร้สายได้อย่างอิสระ ซึ่งทั้งสองระบบ จะมีวงจรป้องกันการชนและหยุดฉุกเฉินได้ตลอดเวลา โดยจะมีตัวเลขพิกัด (X,Y)แสดงทาง ตัวเลข7ส่วน

- จากรูปที่ 9-21 แสดงถึงความสามารถในการกำจัดสัญญาณรบกวนของ Demodulator เมื่อเทียบกับรูปอื่นๆจะเห็นได้ว่า เมื่อไม่มีการส่งข้อมูลเอ๊าท์พุทของ Demodulator จะเป็นสัญญาณรบกวนแบบสุ่มที่ไม่สามารถอ่านเป็นข้อมูลได้ แต่เมื่อใดก็ตามที่ข้อมูลถูกส่ง สัญญาณรบกวนเหล่านั้นจะถูกกำจัดไปเหลือเพียงข้อมูลที่สมบูรณ์

- การสร้างรหัสของตัวไอซีสร้างรหัส ในข้อมูล 1 ชุดจะประกอบไปด้วย 18 บิต(9คู่) แต่ละบิตจะถูกคั่นด้วย สัญญาณ Sync แคบๆ โดยที่ 10 บิตแรก เป็นข้อมูล Address แบ่งเป็น 5 คู่ แต่ละคู่เป็น A0 ~ A4 เรียงตามลำดับ ถ้า Address ใดเป็น 1 จะให้ข้อมูล 2 บิตนั้นเป็น 1 ทั้งคู่ ถ้า Address ใดเป็น 0 จะให้ข้อมูล 2 บิตนั้นเป็น 0 ทั้งคู่ แต่ถ้า Address ใดเป็น High Impedanze จะให้ข้อมูลบิตหน้าเป็น 1 บิตท้ายเป็น 0 ส่วน 8 บิตหลังเป็นข้อมูล Data แบ่งเป็น 4 คู่ แต่ละคู่เป็น D1 ~ A4 เรียงตามลำดับ โดยข้อมูลถ้า Data ใดเป็น 1 จะให้ข้อมูล 2 บิตนั้นเป็น 1 ทั้งคู่ ถ้า Data ใดเป็น 0 จะให้ข้อมูล 2 บิตนั้นเป็น 0 ทั้งคู่ สาเหตุที่ต้องเป็น Data คู่เนื่องจากป้องกันความผิดพลาดของข้อมูล ความถี่ของข้อมูลวัดได้ประมาณ 1.68kHz ซึ่งใกล้เคียงกับค่าตามตารางที่ 7.1 สาเหตุที่คลาดเคลื่อนเล็กน้อยเนื่องจากการหาค่า C ตรงตามกำหนดไม่ได้ รวมทั้งความคลาดเคลื่อนของตัวอุปกรณ์ และสังเกตได้ว่า ข้อมูลฝั่งตัวรับจะน้อยจากแรงดันฝั่งส่งอยู่ 1 V. ซึ่งเป็นแรงดันส่วนที่สูญเสียไปจากการส่ง รวมถึงประสิทธิภาพของตัว Demodulator

- การนับ พัลส์ของ Encoder ในวงจรตรวจระยะ จะให้ 9 พัลส์ต่อการหมุนของมอเตอร์ 1 รอบ สามารถเลื่อนระยะการกระจัดได้ 9.1 เซนติเมตร ตัวนับสามารถนับได้ถึง 99 แสดงว่า มอเตอร์หมุนได้ 11 รอบ เป็นระยะการกระจัด 11*9.1= 99.9 เซนติเมตร ในแต่ละแกน หมายถึง ระบบแกนระนาบการเคลื่อนที่ มีขนาดประมาณ 1 ตารางเมตร

9.3 วิเคราะห์ผลการทดลองและปัญหาการทำงาน

- เนื่องจากผู้ทดลองยังขาดความรู้ในเรื่องการเขียนโปรแกรมทั้งภาษาซี และ Visual Basic รวมทั้งการสื่อสารโดยคอมพิวเตอร์ผ่านพอร์ตอนุกรม จึงมักพบปัญหาการทำงานเกี่ยวกับการใช้ภาษาเป็นอย่างมาก ต้องอาศัยเวลาในการพัฒนาแต่ละส่วนไปอย่างช้าๆ

- แผ่นวงจรพิมพ์ที่ออกแบบยังไม่สวยงามเท่าที่ควร เนื่องจากข้อผิดพลาดในการฉายแสงและ กัดแผ่นทำให้ต้องใช้คัตเตอร์ตัดแต่งภายหลัง บางลายของวงจรจึงไม่ตรง รวมทั้งผิดพลาดหรือหลงลืมในบางอุปกรณ์ จึงต้องตามแก้ในภายหลัง

- วงจรดีเท็กเตอร์นับรอบ มีสัญญาณรบกวนสูงมาก ดังนั้นควรติดตั้งในบริเวณที่มีแสงจากภายนอกน้อย และแผ่นกั้นแสงไม่ควรหมุนเร็วเกินไป (ไม่ควรให้เครนเลื่อนเร็วเกินไป) เพราะจะทำให้การนับผิดพลาดได้ง่าย ผลที่ได้มามีความผิดพลาดข้อนข้างมาก โดยจะเริ่มพบความผิดพลาดเมื่อเครนเคลื่อนที่ไปยังระยะกลางๆของตัวโครงสร้าง เนื่องจากความไม่สมบูรณ์ทางโครงสร้าง ทำให้มีการไถลไปโดยที่ล้อที่ติดตัว Encoder ไว้ไม่เกิดการหมุน ทำให้การเคลื่อนที่จริงคลาดเคลื่อนออกไปจากที่ควรเป็น อีกทั้งสัญญาณรบกวนเนื่องจากสายไฟที่ยาว ทำให้ข้อมูลที่เข้าสู่วงจรนับ 99 ผิดพลาด
- ขาดความรู้และประสบการณ์เรื่องฮาร์ดแวร์ และ โครงสร้างทางแม็คคานิค ทำให้โครงสร้างไม่สวยงามเท่าที่ควร

9.4 แผนการพัฒนาต่อไป

- แก้ข้อผิดพลาด หรือ บัค ในโปรแกรมบางจุด รวมทั้งใช้ภาษาให้สละสลวยขึ้น เพื่อให้โปรแกรมสั้นลง และเอื้อประโยชน์ให้ครอบคลุมการทำงานได้อย่างไร้ข้อผิดพลาดทุกๆกรณี
- ปรับแต่งระบบให้สมบูรณ์มากขึ้น เช่น การลดสัญญาณรบกวน, ประหยัดพลังงานมากขึ้น
- เพิ่มฟังก์ชั่น เช่น การแสดงจุดที่เครนเข้าไปชน หรือ การแจ้งเตือนต่างๆ
- ทำโครงสร้างให้สวยงามยิ่งขึ้น และลดขนาดวงจรให้เล็กลง
- หาแอพริเคชั่นเพิ่มเติมเพื่อความสะดวกหรือแม้แต่ประยุกต์ใช้กับงานอื่นๆ เช่น การติดกล้องไร้สายตลอดจนการติดต่อกับคอมพิวเตอร์โดยไร้สาย

หนังสือและเอกสารอ้างอิง

1. ณัฐพล วงศ์สุนทรชัย และ ชัยวัฒน์ ลิ้มพรจิตรวิไล,
"เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ PIC16F877", หน้า 7-121, หน้า 185-214

2. ประจิน พลังสันติกุล, "เรียนรู้และการใช้งาน CCSC คอมไพเลอร์
(PIC Microcontroller Programming with C Compiler)", หน้า 55-87, หน้า 219-232

3. ผศ. ประภากร สุวรรณะ, "High Freq Communication", หน้า 1- 10

4. ภาควิชาอิเล็กทรอนิกส์ สาขาวิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, เอกสารประกอบการทดลองอิเล็กทรอนิกส์,
" EL 323 การใช้งานของวงจรอิเล็กทรอนิกส์ในด้านการสื่อสาร ", หน้า1 – 5

5. เว็บไซต์ www.thaiio.com

6. อภิชาติ ภู่พลับ, "เริ่มต้นเขียนโปรแกรมติดต่อและควบคุมฮาร์ดแวร์ด้วย Visual Basic ",
หน้า 157-172

ภาคผนวก

โค้ดโปรแกรมที่ใช้ในไมโครคอนโทรลเลอร์ PIC16F877 ดังนี้

```
#define _PIC16F877A_
#include <16f877A.h>
#fuses HS
#fuses NOLVP, NOWDT
#fuses NOPROTECT
#byte porta      = 0x05      // Register port A          at 0x05
#byte porta_tris = 0x85      // Register port A tris at 0x85
#byte portb      = 0x06      // Register port B          at 0x06
#byte portb_tris = 0x86      // Register port B tris at 0x86
#byte portc      = 0x07      // Register port C          at 0x07
#byte portc_tris = 0x87      // Register port C tris at 0x87
#byte portd      = 0x08      // Register port D          at 0x08
#byte portd_tris = 0x88      // Register port D tris     at 0x88
#define ERX     PIN_A0       // O/P : Enable read digit 7 seg X axis     0
#define ERY     PIN_A1       // O/P : Enable read digit 7 seg Y axis     0
#define STD     PIN_A2       // I/O : Start driver                       1
#define EMT     PIN_B0       // I/P : Interrupt Emergency stop           1
#define UPS     PIN_B1       // I/P : Up switch touched                  1
#define CZN     PIN_B2       // I/P : -Z Axis touched                    1
#define DZN     PIN_B3       // O/P : -Z Axis drive                      0
#define CXP     PIN_B4       // I/P : +X Axis touched                    1
#define DXP     PIN_B5       // O/P : +X Axis drive                      0
#define CXN     PIN_B6       // I/P : -X Axis touched                    1
#define DXN     PIN_B7       // O/P : -X Axis drive                      0
#define DYN     PIN_C0       // O/P : -Y Axis drive                      0
#define CYN     PIN_C1       // I/P : -Y Axis touched                    1
#define DYP     PIN_C2       // O/P : +Y Axis drive                      0
#define CYP     PIN_C3       // I/P : +Y Axis touched                    1
#define CZP     PIN_C4       // I/P : +Z Axis touched                    1
#define DZP     PIN_C5       // O/P : +Z Axis drive                      0
```

```
#define TXD     PIN_C6          // O/P : Serial transmission data to PC      0
#define RXD     PIN_C7          // I/P : Serial recieve data from PC         1
#define CLOCK_SP 20000000
#use delay (clock=CLOCK_SP)
#use RS232 (baud=1200, xmit=TXD, rcv=RXD, parity=N)
/*-------------------------------------------------------------------------*/
boolean emergency = true;
boolean checker   = true;
int8      k, x0, x1, x2, y0, y1, y2;
set_tris_b (0x57);          // Set B0-B2,B4,B6 to input,B3,B5,B7 to output
set_tris_c (0x9a);          // Set C1,C3,C4,C7 to input,C0,C2,C5,C6 to output
set_tris_d (0xff);// Set port D to input read 8 bits position
/*-------------------------------------------------------------------------*/
void Chktoc (void)        {             // Check touching limit switch
        if (input(CXP))  {
                output_high (DXN);      // Drive motor back
                delay_ms (1000);
                output_Low (DXN);
                putc(0x37);             // Send Cancle signal to serial port PC
                while (!input(STD));    // Wait User start up
                x0 = 0x00 , y0 = 0x00;
                x1 = 0x00 , y1 = 0x00;
                x2 = 0x00 , y2 = 0x00;  // Clear position
                }
        else if (input(CXN))      {
                output_high (DXP);      // Drive motor back
                delay_ms (1000);
                output_Low (DXP);
                putc(0x37);             // Send Cancle signal to serial port PC
                while (!input(STD));    // Wait User start up
                x0 = 0x00 , y0 = 0x00;
                x1 = 0x00 , y1 = 0x00;
```

```
                x2 = 0x00 , y2 = 0x00;    // Clear position
                }
        else if (input(CYP))        {
                output_high (DYN);        // Drive motor back
                delay_ms (1000);
                output_Low (DYN);
                putc(0x37);               // Send Cancle signal to serial port PC
                while (!input(STD));      // Wait User start up
                x0 = 0x00 , y0 = 0x00;
                x1 = 0x00 , y1 = 0x00;
                x2 = 0x00 , y2 = 0x00;    // Clear position
                }
        else if (input(CYN))        {
                output_high (DYP);        // Drive motor back
                delay_ms (1000);
                output_Low (DYP);
                putc(0x37);               // Send Cancle signal to serial port PC
                while (!input(STD));      // Wait User start up
                x0 = 0x00 , y0 = 0x00;
                x1 = 0x00 , y1 = 0x00;
                x2 = 0x00 , y2 = 0x00;    // Clear position
                }
        }
/*-------------------------------------------------------------------------*/
void Recxy (void)           {             // Check now position
        output_high (ERY);
        delay_ms (100);
        output_low (ERX);
        x0 = input_d();
        delay_ms (100);
        output_high (ERX);                // Read now location X = x0
        delay_ms (100);
```

```
        output_low (ERY);
        y0 = input_d();
        delay_ms (100);
        output_high (ERY);              // Read now location Y = y0
        delay_ms (100);
        }
/*-------------------------------------------------------------------------*/
void Loading (void)     {               // Loading Z axis
        while (!input(CZN))     {
                output_high (DZN);      // Drive motor down
                delay_ms (200);
                output_Low (DZN);
                delay_ms (200);
                }
        while (!input(UPS));            // Wait up
        while (!input(CZP))     {
                output_high (DZP);      // Drive motor up
                delay_ms (200);
                output_Low (DZP);
                delay_ms (200);
                }
        }
/*-------------------------------------------------------------------------*/
void Movxy (void)       {               // Moving XY axis
        if (x1>x0)      {
                output_high (DXP);      // Drive motor +X
                delay_ms (200);
                output_Low (DXP);
                delay_ms (200);
                }
        else if (x1<x0)  {
                output_high (DXN);      // Drive motor -X
```

```
                delay_ms (200);
                output_Low (DXN);
                delay_ms (200);
                }
        else if (y1>y0)   {
                output_high (DYP);      // Drive motor +Y
                delay_ms (200);
                output_Low (DYP);
                delay_ms (200);
                }
        else if (y1<y0)   {
                output_high (DYN);      // Drive motor -Y
                delay_ms (200);
                output_Low (DYN);
                delay_ms (200);
                }
        }
/*-------------------------------------------------------------------------*/
void useserial (void)       {               // Get XY Position from PC
        while(!kbhit());            // Wait "ready to send" data from PC
        k=getc();
        delay_ms (200);
        putc(0x31);                 // Send "request X1" to PC
        while(!kbhit());
        delay_ms (200);
        x1 = getc();                // Recieve X1
        delay_ms (400);
        putc(0x32);                 // Send "request Y1" to PC
        while(!kbhit());
        delay_ms (200);
        y1 = getc();                // Recieve Y1
        delay_ms (400);
```

```
        putc(0x33);                     // Send "request X2" to PC
        while(!kbhit());
        delay_ms (200);
        x2 = getc();                    // Recieve X2
        delay_ms (400);
        putc(0x34);                     // Send "request Y2" to PC
        while(!kbhit());
        delay_ms (200);
        Y2 = getc();                    // Recieve Y2
        delay_ms (400);
        putc(0x35);                     // Send "all data recept" to PC
        }
/*-------------------------------------------------------------------*/
#int_ext
void emer_int (void)    {
        checker = false;
        emergency = false;
  }
/*-------------------------------------------------------------------*/
#int_rda
void esc_int (void)     {
        k=getc();
        checker = false;
        }
/*-------------------------------------------------------------------*/
void main ()    {
        set_tris_a (0x00);              // Set port A to output
        output_high (STD);
        delay_ms (500);
        output_low (STD);
        output_float (STD);             // Send pulse to start all driver
        enable_interrupts (global);     // Use interrupt
```

```
enable_interrupts (int_ext);                // Use external interrupt RB0
while (emergency){                          // Always check emergency stop
checker = true;
useserial ();                               // Get X Y Position
        enable_interrupts (int_rda);        // Use RS232 interrupt
        while (checker){                    // Always check esc signal from PC
        Recxy ();                           // Read location from port D
        while (x0 != x1, y0 != y1) {
        Movxy ();
        Recxy ();
        Chktoc ();              }           // Move to x1, y1
        Loading ();                         // Loading
        x0 = x1 , x1 = x2;
        y0 = y1 , y1 = y2;
        while (x0 != x1, y0 != y1) {
        Movxy ();
        Recxy ();
        Chktoc ();              }           // Move to x2, y2
        Loading ();                         // Loading
        putc(0x36);                         // Send OK signal to PC
        disable_interrupts (int_rda);       // Unuse RS232 interrupt
        checker = false;
                }
        }
set_tris_a (0x04);                          // Set A2 to input else to output
while (!input(STD));                        // Wait User start up
Recxy ();
x1 = 0x00 , y1 = 0x00;
while (x0 != 0x00, y0 != 0x00) {
Recxy ();
Movxy ();       }                           // Back to Origin
while (!input(CZP)) {
```

```
        output_high (DZP);
        delay_ms (300);
        output_low (DZP);
        delay_ms (300);}                    // Lift to highest point
        putc(0x37);                         // Send Cancle signal to PC
        disable_interrupts (int_rda);       // Unuse RS232 interrupt
        while (true);                       // Wait reset
        }
/*-------------------------------------------------------------------*/
```

โค้ดโปรแกรมที่เขียนใน Visual Basic ดังนี้

```
Private Sub CancleButton_Click()
If MsgBox("Do you want to cancle now?", 68, "Stop Working") = 6 Then
  Serial.Output = Chr(&HAA)
  Serial.PortOpen = False
  CancleButton.Enabled = False
  OKButton.Enabled = True
  X1Srcoll.Enabled = True
  Y1Srcoll.Enabled = True
  X2Srcoll.Enabled = True
  Y2Srcoll.Enabled = True
  HelpText.Text = "Select your coordinate"
  End If
End Sub

Private Sub Form_Activate()
For X = 0 To 100 Step 10
Coor.Line (X, -100)-(X, 0), vbBlack
Next X
For Y = -100 To 0 Step 10
Coor.Line (0, Y)-(100, Y), vbBlack
Next Y
End Sub

Private Sub Form_Load()
Dim X1, X2, Y1, Y2 As Byte
Dim a, a1, a2 As Byte
Dim b, b1, b2 As Byte
Dim c, c1, c2 As Byte
Dim d, d1, d2 As Byte
Coor.Scale (-1, -101)-(101, 1)
```

```
Coor.BackColor = vbWhite
CancleButton.Enabled = False
End Sub

Private Sub OKButton_Click()
Serial.Settings = "1200,n,8,1"
Serial.CommPort = 1
Serial.PortOpen = True
OKButton.Enabled = False
X1Srcoll.Enabled = False
Y1Srcoll.Enabled = False
X2Srcoll.Enabled = False
Y2Srcoll.Enabled = False
CancleButton.Enabled = True
HelpText.Text = "Please wait while operating"
Serial.Output = Chr(&HAA)
Timer1.Interval = 400
Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
On Error Resume Next
a1 = X1 Mod 10
a2 = ((X1 - a1) / 10) * 16
a = a1 + a2
b1 = Y1 Mod 10
b2 = ((Y1 - b1) / 10) * 16
b = b1 + b2
c1 = X2 Mod 10
c2 = ((X2 - c1) / 10) * 16
c = c1 + c2
d1 = Y2 Mod 10
```

```
d2 = ((Y2 - d1) / 10) * 16
d = d1 + d2
Serial.InputLen = 1
Data = Serial.Input
Select Case Data

Case &H1
Serial.Output = Chr(a)
HelpText.Text = "Sending X1"

Case &H2
Serial.Output = Chr(b)
HelpText.Text = "Sending Y1"

Case &H3
Serial.Output = Chr(c)
HelpText.Text = "Sending X2"

Case &H4
Serial.Output = Chr(d)
HelpText.Text = "Sending Y2"

Case &H5
 If MsgBox("All data recieved", vbOKOnly, "Operting") = vbOK Then
 HelpText.Text = "Please Wait"
 End If

Case &H6
 If MsgBox("Operated complete", vbOKOnly, "Stopworking") = vbOK Then
 CancleButton.Enabled = False
 OKButton.Enabled = True
 X1Srcoll.Enabled = True
```

```
Y1Srcoll.Enabled = True
X2Srcoll.Enabled = True
Y2Srcoll.Enabled = True
HelpText.Text = "Select your coordinate"
Serial.PortOpen = False
End If

Case &H7
If MsgBox("Emergency Stop", vbOKOnly, "Stopworking") = vbOK Then
CancleButton.Enabled = False
OKButton.Enabled = True
X1Srcoll.Enabled = True
Y1Srcoll.Enabled = True
X2Srcoll.Enabled = True
Y2Srcoll.Enabled = True
X1Srcoll.Value = 0
Y1Srcoll.Value = 0
X2Srcoll.Value = 0
Y2Srcoll.Value = 0
HelpText.Text = "Select your coordinate"
Serial.PortOpen = False
End If

End Select
End Sub

Public Sub X1Srcoll_Change()
X1.Text = X1Srcoll.Value
fro.Move X1 - 0.5
Call Form_Activate
End Sub
```

```
Public Sub X2Srcoll_Change()
X2.Text = X2Srcoll.Value
too.Move X2 - 0.5
Call Form_Activate
End Sub

Public Sub Y1Srcoll_Change()
Y1.Text = Y1Srcoll.Value
fro.Move X1 - 0.5, -Y1 - 0.5
Call Form_Activate
End Sub

Public Sub Y2Srcoll_Change()
Y2.Text = Y2Srcoll.Value
too.Move X2 - 0.5, -Y2 - 0.5
Call Form_Activate
End Sub
```