หุ่นยนต์สำรวจและเขียนแผนที่

ROBOTIC SERVEY AND MAP BUILDING

นายสาธิต โยคเสนะกุล

นางสาวเณริศศา บรรลือ

นายศิริชัย ปานพรหมมินทร์

โครงงานวิศวกรรมนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

วิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ

พ.ศ. 2550

หุ่นยนต์สำรวจและเขียนแผนที่

ROBOTIC SURVEY AND MAP BUILDING

นายสาธิต โยคเสนะกุล

นางสาวเณริศศา บรรลือ

นายศิริชัย ปานพรหมมินทร์

โครงงานวิศวกรรมสาขาวิชาวิศวกรรมไฟฟ้านี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

วิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ

พ.ศ. 2550

ลิขสิทธิ์ของคณะวิศวกรรมศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ

หัวข้อโครงงานวิศวกรรมไฟฟ้า เรื่อง หุ่นยนต์สำรวจและเขียนแผนที่

โดย นายสาธิต โยคเสนะกุล
นางสาวเณริศศา บรรลือ
นายศิริชัย ปานพรหมมินทร์

ภาควิชา สาขาวิชาวิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา ผู้ช่วยศาตราจารย์ชัยณรงค์ คล้ายมณี

คณะวิศวกรรมศาสตร์มหาวิทยาลัยศรีนครินทรวิโรฒอนุมัติให้นับโครงงานวิศวกรรมสาขาวิชาวิศวกรรมไฟฟ้า เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

……..…….……..… คณบดีคณะวิศวกรรมศาสตร์
(อาจารย์อารีย์ หาญสืบสาย)

คณะกรรมการสอบโครงงานวิศวกรรม

ประธานกรรมการ
……...…
(ผู้ช่วยศาสตราจารย์ชัยณรงค์ คล้ายมณี)

กรรมการ
...……...…
(อาจารย์ธานินทร์ ดวงจันทร์)

กรรมการ
……....
(อาจารย์อาคม ม่วงเขาแดง)

กรรมการ
...……...…
(ผู้ช่วยศาสตราจารย์ศิริพงษ์ ฉายสินธ์)

สารบัญ

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 2 (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

ฎ

บทที่ 1
บทนำ

1.1 แนวความคิดในการจัดทำโครงงาน

จากอดีตจนถึงปัจจุบัน จะพบได้ว่าในการดำรงชีวิตอยู่ของคนเรานั้นอาจเกิดอุบัติเหตุขึ้นได้เสมอ ซึ่งบางครั้งการเกิดอุบัติเหตุในจุดที่มนุษย์สามัญชนทั่วไปไม่อาจเข้าไปถึงจุดที่เกิดเหตุเพื่อช่วยเหลือได้ ทำให้ผู้ประสบเหตุไม่ได้รับความช่วยเหลือจนเกิดความเสียหายทั้งชีวิตและทรัพย์สิน ทำให้คณะผู้จัดทำเกิดแนวคิดที่จะสร้างโปรแกรมสำรวจแผนที่คร่าวๆ โดยได้ติดตั้งในหุ่นยนต์กู้ภัย เพื่อที่จะสำรวจสถานที่เกิดเหตุคร่าวๆ เพราะเหตุที่มนุษย์ไม่อาจเข้าถึงได้

ในปัจจุบันนี้ ทางสถาบันทางการศึกษา โดยเฉพาะมหาลัยศรีนครินทรวิโรฒนั้นได้มีการพัฒนา หุ่นยนต์กู้ภัยอยู่เรื่อยมา แต่ทว่า ในด้านการสร้างภาพจำลองจากสถานที่จริงนั้น ยังไม่มีการพัฒนาไม่มีผู้บุกเบิกเลย ทางคณะผู้จัดทำจึงได้เกิดความคิดที่จะทำโปรแกรมสำรวจแผนที่ ในหุ่นยนต์กู้ภัยขึ้นมา ซึ่งโครงงานนี้ ได้ใช้ความรู้ที่ได้ร่ำเรียนมา คือเราใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุม อุปกรณ์ทั้งหมด ซึ่งมี Ultrasonic โมดูลเข็มทิศ encoder และส่งข้อมูลต่างๆด้วยสาย RS-232 เข้าสู่คอมพิวเตอร์ และใช้ VB (Visual Basic)
ในการแสดงผลข้อมูลบนหน้าจอคอมพิวเตอร์

1.2 วัตถุประสงค์

1.2.1 เพื่อต้องจะคาดเดาสิ่งกีดขวางในแผ่นที่คร่าวๆของจุดที่ต้องการทราบ

1.2.2 เพื่อจะได้แผนที่แบบฉายด้านบนอย่างง่าย

1.2.3 เพื่อนำมาประยุกต์และนำไปใช้ให้มีประโยชน์ต่องานทางด้าน Rescue robot

1.3 ขอบเขตของโครงงาน

1.3.1 สามารถที่จะทำให้อุปกรณ์ SRF-04 (ultrasonic) ZX-21(encoder) CMPS03 (โมดูลเข็มทิศ) ทำงานร่วมกันได้ใน MCS-51

1.3.2 สามารถทำให้บอร์ดทดลอง MCS-51 สื่อสารกับคอมพิวเตอร์ และแสดงข้อมูลที่ได้บนจอคอมพิวเตอร์ด้วยโปรแกรม VB (Visual Basic) ได้

1.3.3 สามารถเขียนแผนที่อย่างคร่าวๆ ของจุดที่ต้องการจะสำรวจได้ในแนวระนาบและเป็นพื้นเรียบ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 เขียนแผนที่ออกมาให้เห็นระยะทางระหว่างรถกับสิ่งกีดขวางได้

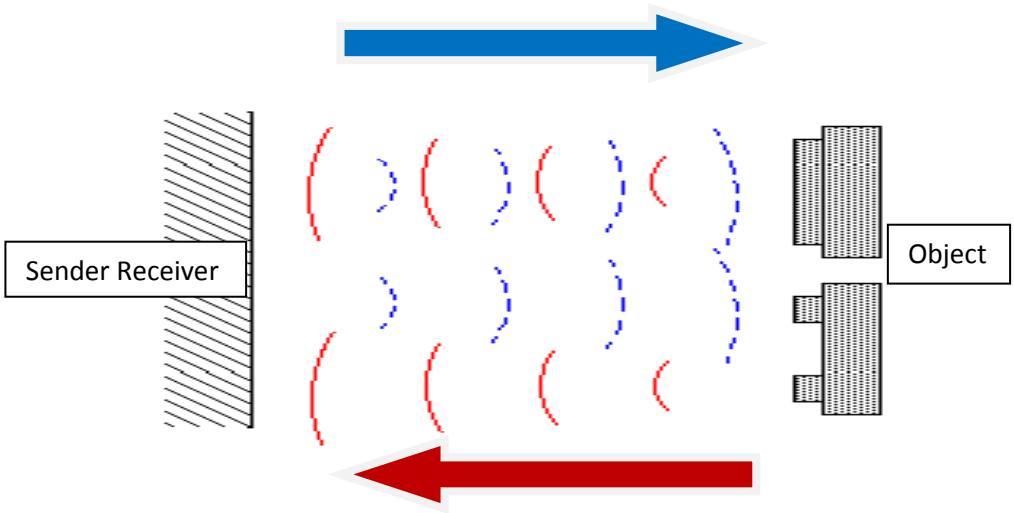
1.4.2 เป็นโปรแกรมสำรวจที่ใช้รวมกับหุ่นยนต์กู้ภัยได้เป็นอย่างดี

1.4.3 การนำร่องการพัฒนา หรือเป็นโปรแกรมตัวอย่างในการนำไปใช้ให้เกิดประโยชน์ได้จริง

บทที่ 2
ทฤษฎีและหลักการ

2.1 อัลตร้าโซนิค (Ultrasonic)

อัลตร้าโซนิค หมายถึง คลื่นเสียงที่มีความถี่สูงเกินกว่าที่หูมนุษย์จะได้ยิน โดยทั่วไปแล้วหูของมนุษย์โดยเฉลี่ยจะได้ยินเสียงทีมีความถี่ไม่เกิน 20 KHz เท่านั้น ดังนั้นโดยปกติแล้วคำว่าอัลตร้าโซนิค จึงมักจะหมายถึงคลื่นเสียงที่มีความถี่สูงกว่า 20 KHz ขึ้นไป จะสูงขึ้นจนถึงเท่าใดไม่ได้ระบุจำกัดเอาไว้ เหตุผลที่มีการนำเอาคลื่นย่านอัลตร้าโซนิคมาใช้ก็เพราะว่าเป็นคลื่นทีมีทิศทางทำให้เราสามารถเล็งคลื่นเสียงไปยังเป้าหมายที่ต้องการได้โดยเจาะจง การมีทิศทางของคลื่นเสียงย่านอัลตร้าโซนิคทำให้เรานำไปใช้งานได้หลายอย่าง เช่น นำไปใช้ในเครื่องควบคุมระยะไกล (Ultrasonic remote control) เครื่องล้างอุปกรณ์ (Ultrasonic cleaner) เครื่องวัดความหนาของวัตถุโดยสังเกตระยะเวลาที่คลื่นสะท้อนกลับมา เครื่องวัดความลึกและทำแผนที่ใต้ท้องทะเล ใช้ในเครื่องหาตำแหน่งอวัยวะบางส่วนในร่างกาย ใช้ทดสอบการรั่วไหลของท่อ เป็นต้น อุปกรณ์ที่สามารถแปลงพลังงานในรูปอื่นให้มาเป็นพลังงานทางกลโดยการสั่นไปมา ซึ่งทำให้เกิดคลื่นเสียงย่านอัลตร้าโซนิค กระจายไปในอากาศได้หรือแปลงพลังงานทางกลให้มาเป็นพลังงานในรูปอื่นได้นั้นชื่อเรียกว่าอัลตร้าโซนิคทรานดิวเซอร์ (Ultrasonic Transducer) ซึ่งหลักการทำงานอัลตร้าโซนิค แบ่งออกเป็น 2 ส่วน คือ ภาคส่ง และภาครับ โดยมีหลักการทำงานดังภาพ



รูปที่ 2.1 หลักการทำงานของอัลตร้าโซนิคทรานดิวเซอร์

ภาคส่ง คือ อัลตร้าโซนิคทรานสดิวเซอร์ที่ถูกออกแบบเจาะจงมาให้แปลงสัญญาณไฟฟ้าที่ให้แก่ตัวมัน ให้ออกมาเป็นคลื่นเสียงย่านอัลตร้าโซนิค

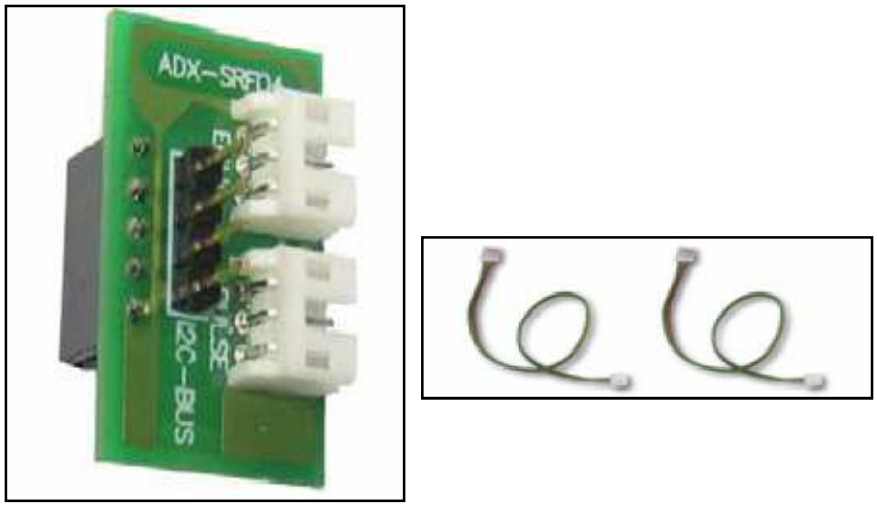
ภาครับ คือ อัลตร้าโซนิคทรานสดิวเซอร์ที่ถูกออกแบบเจาะจงมาให้แปลงคลื่นเสียงย่านอัลตร้าโซนิคที่มาตกกระทบตัวมันให้ออกมาเป็นสัญญาณไฟฟ้าและในโครงงานนี้เลือกใช้ อัลตร้าโซนิคทรานสดิวเซอร์รุ่น SRF04 ซึ่งจะกล่าวโดยละเอียดในหัวข้อต่อไป

2.1.1 อัลตร้าโซนิคทรานสดิวเซอร์ SRF04

SRF04 เป็นแผงวงจรวัดตรวจจับและวัดระยะทางด้วยคลื่นอัลตร้าโซนิคที่มีความเที่ยงตรงสูงโดยสามารถวัดระยะได้ตั้งแต่ 1 เซนติเมตรไปจนถึง 4 เมตร SRF04 ถูกออกแบบมาให้ใช้งานกับไมโครคอนโทรเลอร์ได้ง่ายโดยใช้ขาเชื่อมต่อเพียง 1 หรือ 2 ขา ขึ้นอยู่กับการกำหนดรูปแบบการทำงานทางฮาร์ดแวร์เหมาะอย่างยิ่งกับการประยุกต์ใช้งานทางด้านหุ่นยนต์

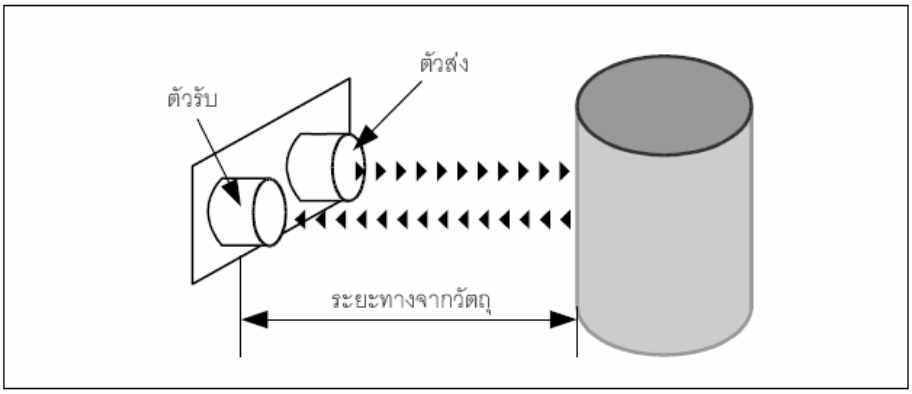
คุณสมบัติของ SRF04 คือ ใช้ไฟเลี้ยง +5 V ต้องการกระแสไฟฟ้า 30 มิลลิแอมป์ ใช้ตัวรับและส่งคลื่นอัลตร้าโซนิคใช้ความถี่ 40 kHz ในการทำงานวัดระยะทางในช่วง 3 เซนติเมตร ถึง 3 เมตรสัญญาณพัลส์สำหรับกระตุ้นการทำงาน ต้องมีความกว้างอย่างน้อย 10ไมโครวินาที ให้ผลลัพธ์จากการวัดระยะเป็นค่าความกว้างพัลส์ซึ่งเป็นสัดส่วนกับระยะทางที่วัดได้ มีขนาดเล็กคือ กว้าง 43 มิลลิเมตร ยาว 20 มิลลิเมตร สูง 17 มิลลิเมตร สามารถติดต่อได้ 2 แบบคือ แบบ 2

สัญญาณ (Echo กับ Trigger)



รูปที่ 2.2 บอร์ด ADX–SRF04 และ สาย PCB3A

SRF04 จะทำการส่งสัญญาณคลื่นอัลตร้าโซนิกออกไป แล้ววัดระยะเวลาที่มีสัญญาณสะท้อนตอบกลับมา เอาต์พุตที่ได้จาก SRF04 จะอยู่ในรูปความกว้างพัลส์ซึ่งสัมพันธ์กับระยะทางของวัตถุที่ตรวจจับได้ความถี่สัญญาณอัลตร้าโซนิคของ SRF04 คือ 40 kHz ถูกส่งออกไปในอากาศด้วยความเร็ว ประมาณ 346 เมตรต่อวินาที ดังนั้นเมื่อทราบความเร็วในการเคลื่อนที่ของคลื่น เวลาเริ่มต้นส่งคลื่น และเวลาที่รับเสียงสะท้อนกลับมาจึงสามารถคำนวณหาค่าของระยะทางได้ ดังแสดงการตรวจจับในรูปที่ 2.3



รูปที่ 2.3 หลักการตรวจจับวัตถุโดยคลื่นเสียงอัลตร้าโซนิค

ระยะทางที่ได้นั้นจะต้องมีการคำนวณค่ากลับทางคณิตศาสตร์เมื่อใช้กับไมโครคอนโทรลเลอร์แล้วถือว่าเป็นเรื่องยุ่งยากพอสมควร ดังนั้น SRF04 จึงประมวลผลค่าทางคณิตศาสตร์ต่าง ๆ เหล่านี้ไว้เรียบร้อยแล้วจากนั้นส่งผลลัพธ์ที่วัดได้ออกมาเป็นพัลส์ที่มีความกว้างสัมพันธ์กับระยะทางที่วัดได้

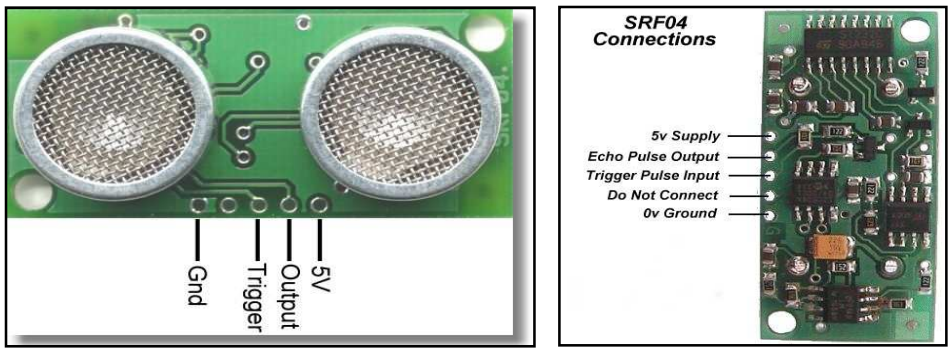
กำหนดให้

R = ระยะทางจากเครื่องส่งไปยังวัตถุ

T = ช่วงเวลาที่สัญญาณสะท้อนกลับมายังตัวรับ

R = (1/2) * T * 346 (m)

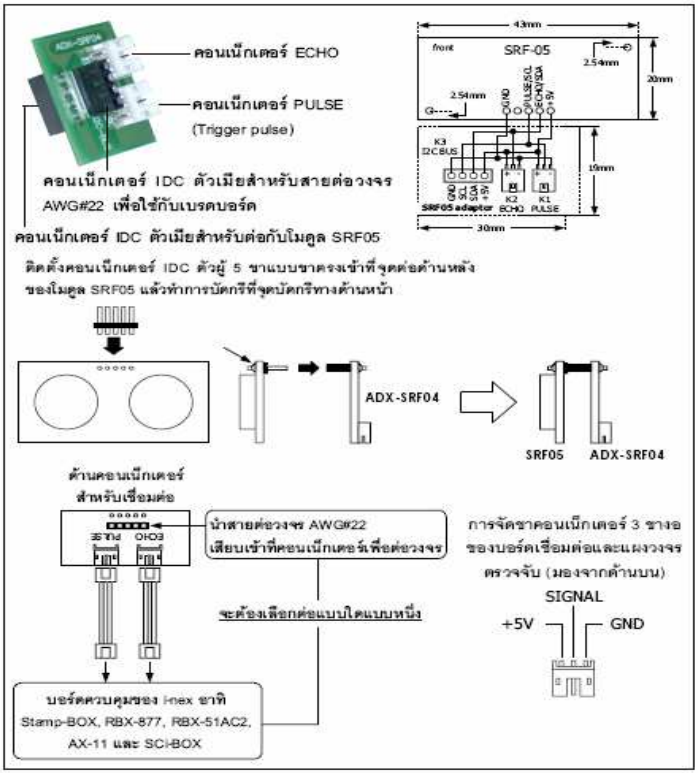
2.1.1.1 จุดต่อการใช้งานของ SRF04



รูปที่ 2.4 แสดงขาสัญญาณของ SRF04 แผงวงจรตรวจจับและวัดระยะทางด้วยคลื่นอัลตร้าโซนิค

มีจุดต่อสำหรับการใช้งานอยู่ทั้งหมด 4 จุด ขาไฟเลี้ยง(+5 v) สำหรับต่อไฟเลี้ยง +5v ขา Echo Pulse Output (ECHO) เป็นขาเอาต์พุตสำหรับส่งสัญญาณพัลส์ออกจาก SRF04 ซึ่งการใช้งานจะนำขานี้ไปต่อเข้ากับพอร์ตอินพุตของไมโครคอนโทรลเลอร์เพื่อตรวจจับความกว้างของสัญญาณพัลส์ที่ส่งออกมาเพื่อแปลความหมายออกมาเป็นระยะทางอีกครั้งหนึ่ง ขา Trigger Pulse Input (Trigger) เป็นขาอินพุตรับสัญญาณที่มีความกว้างอย่างน้อย 10 ไมโครวินาทีเพื่อกระตุ้นการสร้างคลื่นอัลตร้าโซนิคความถี่ 40 kHz ออกสู่อากาศจากตัวส่ง ดังนั้นเมื่อคลื่นความถี่ดังกล่าวเคลื่อนที่ไปกระทบสิ่งกีดขวางที่อยู่เบื้อง

หน้าก็จะเกิดการสะท้อนกลับเข้ามายังตัวรับ และถูกแปลงออกมาเป็นความกว้างของสัญญาณพัลส์ที่จะถูกส่งออกไปทางขา Echo Pulse Output นอกจากนี้ในโหมดหนึ่งสัญญาณ จะใช้จุดนี้เป็นจุดสื่อสารข้อมูลอนุกรมเพื่อส่งรับส่งค่าการวัดกับไมโครคอนโทรลเลอร์ ขาGNDสำหรับต่อกราวด์



รูปที่ 2.5 วงจรบอร์ด ADX – SRF04

บนบอร์ด ADX – SRF04 ได้จัดเตรียมคอนเน็กเตอร์ PCBแบบสามขาตัวผู้สองตัวแยกกันระหว่างสัญญาณ ECHO กับ TRIGGER สำหรับเชื่อมต่อกับบอร์ดควบคุม และคอนเน็กเตอร์ IDC ตัวเมียแถวเดียว 4 ขาสำหรับเสียบสายต่อกับวงจร

2.1.2 การใช้งาน Ultrasonic Distance Detector Module SRF04 กับไมโครคอนโทรเลอร์

สร้างไลบรารีภาษา C เพื่อใช้งานโมดูล SRF04 กับไมโครคอนโทรเลอร์ MCS-51 การเขียนโปรแกรมภาษา C เพื่อกำหนดให้ไมโครคอนโทรเลอร์ MCS-51 ติดต่อกับโมดูล SRF04

สามารถทำได้ไม่ยาก และควรเริ่มต้นด้วยการสร้างไฟล์ไลบรารี จากนั้นนำไฟล์ ไลบราลีไปใช้เขียนโปรแกรมควบคุมต่อไปนี้ สำหรับไลบราลีที่สร้างขึ้นนี้ตั้งชื่อว่า ultrasonic.h เป็นไฟล์ไลบราลีสำหรับใช้งาน SRF04 เพื่อวัดระยะทางในหน่วยเซนติเมตร ภายในไลบราลี ultrasonic.h จะกำหนดให้ใช้งานขอ P0.0 ต่อเข้ากับขา Echo Pulse Output และ P0.1 ต่อเข้ากับขา Trigger Pulse I

หลักการสร้างฟังก์ชั่นสำหรับไลบรารี Ultrasonic.h เริ่มต้นด้วยการกำหนดให้ P0.1 อยู่ในสภาวะลอจิก “0” ก่อนการสร้างพัลส์บวกที่มีความกว้างอย่างน้อย 10 ไมโครวินาทีเพื่อส่งให้ SRF04 เริ่มต้นกระบวนการส่งคลื่นอัลตร้าโซนิค กำหนดให้ P0.1 อยู่ในสภาวะลอจิก “0” ก่อนเข้ากระบวนการตรวจจับพัลส์บวกที่ออกมาทางขา Echo Pulse Output ของ SRF04 ตั้งโหมดการนับของไทเมอร์ ในที่นี้ใช้ไทเมอร์ 1ในโหมด16 บิต พร้อมกับเคลียร์ค่าการนับให้เป็นศูนย์ (TH1=0 และ TL1=0) เพื่อนับช่วงเวลาของพัลส์บวกที่ได้จากขา Echo ของ SRF04 ซึ่งต่อกับขา P0.0 ส่งพัลส์บวกที่มีความกว้างอย่างน้อย 10 ไมโครวินาทีออกไปทางยังขา TRIGGER ซึ่งต่ออยู่กับขา P0.1 เพื่อเริ่มต้นกระบวนการสร้างคลื่นอัลตร้าโซนิค (กระบวนการนี้ถูกดำเนินการภายใต้ฟังก์ชั่น trigger pulse) หลังจากขั้นตอนที่ 4 คลื่นอัลตร้าโซนิคจะถูกปล่อยออกสู่อากาศ เมื่อเดินทางไปพบวัตถุหรือสิ่งกีดขวางใดๆ จะเกิดปรากฏการสะท้อนกลับของคลื่นมายังตัวรับของแผงวงจร SRF04 จะทำให้ SRF04 ส่งพัลส์บวกที่มีความกว้างเป็นสัดส่วนกับระยะทางออกมาทางขา ECHOในช่วงนี้ของโปรแกรมจะวนตรวจสอบขา P0.0 ว่า มีการเปลี่ยนแปลงจากสภาวะลอจิก “0” เป็น “1” หรือไม่ ซึ่งถ้า “ใช่” จะเริ่มนับเวลาด้วยการเปิดการนับไทเมอร์ 1 แล้วรอจนกระทั่งขา P0.0 มีการเปลี่ยนลอจิก “1” เป็น “0” จึงปิดการนับของไทเมอร์1 แต่ถ้า “ไม่” ก็จะวนตรวจสอบตลอดเวลา เมื่อสิ้นกระบวนการตรวจจับความกว้างพัลส์บวกในแต่ละรอบควรหน่วงเวลา 10มิลลิวินาที กระทำขั้นตอนที่ 4 ถึง 6 ต่อเนื่องกัน 5 รอบ เพื่อให้ได้ค่าเฉลี่ยที่น่าเชื่อถือของข้อมูลมากขึ้น หลังจากที่ได้ข้อมูลความกว้างพัลส์แล้ว ต้องนำไปหารกับค่าคงที่หนึ่งที่เหมาะสมกับความเร็วในการทำงานของไมโครคอนโทรเลอร์แต่ละเบอร์ ซึ่งจากการทดลอง พบว่า ค่านั้นคือ114 (ที่ความเร็ว 2 เท่าของไมโครคอนโทรเลอร์ MCS-51มาตรฐาน และความถี่คริสตอล 11.0592 MHz) รายละเอียดของไฟล์ไลบราลีแสดงในโปรแกรมที่ 1 ฟังก์ชั่นเรียกใช้งานในไลบราลี ultrasonic.h คือฟังก์ชั่น distance: เป็นฟังก์ชั่นอ่านค่าระยะทางจาก SRF04 กับวัตถุที่ตรวจพบ

รูปแบบฟังก์ชั่น unsigned int distance (void)

การคืนค่า คืนค่าข้อมูลเป็นระยะทางในหน่วยเซนติเมตร

โปรแกรมที่ 2.1 ไฟล์ไลบรารีที่ใช้ทำงานร่วมกับโมดูล SRF04

```
/*_____________________________________________________________*/
// Promgram : Module function read distance from SFR04 sensor
// Description : Call function control LCD display group type 4 bit
// Filename : ultrasonic.h
// C compiler: RIDE 51 V6.1
/*_____________________________________________________________*/
#include <intrins.h>          // Include library for lop function
sbit echo = P0^0;             // Define receive pulse pin
sbit trigger = P0^1;          // Define trigger pulse pin
/*************************************************************/
/****************Function Trigger pulse for start process*************/
/*************************************************************/
void trigger pulse (void)
{
    unsigned char i;          // Variable for counter
    trigger = 1;              // Start positive pulse
    for (i=0;i<10;i++)        // For loop 10 time
    _nop_();                  // Delay 1 us function
    trigger = 0;              // End of positive  pulse
}
/*************************************************************/
/*******************Function Read distance**********************/
```

โปรแกรมที่ 2.1 ไฟล์ไลบราลีที่ใช้ทำงานร่วมกับโมดูล SRF04 (ต่อ)

```
/*********************************************************/
Unsigned int distance()
{
    unsigned int mc, dat,i ;    //  Variable for internal this function
    trigger = 0;                //   Initial logic low
    echo = 1;                   //   Initial logic high

 TMOD &=0x0F;           //   Configuration timer1 mode2 (16 bit counter)
    TMOD  1 =0x10;
     H1   =   0x00;           // Initial timer1 counter value at zero
     TL1   =    0x00;
     TF1   =    0;             // Clear overflow flag
     TR1 =     0;              // Start Timer1
     For (i=o; i<5; i++)       // For loop 5 time for average data
{
     trigger pulse ();         // Send trigger pulse signal
     while (! echo);           //  Detect rising pulse
      tR1 = 1;                // Start timer count
      while (echo);           // Detect falling pulse
      TR1 = 0;                // Stop timer
      TF1 = 0;                // Clear bit overflow flag
      mc = TH1;               //  Keep high byte
      mc <<= 8;               //  Shift to high byte
      mc +=  \TL1;            //  Keep low byte
```

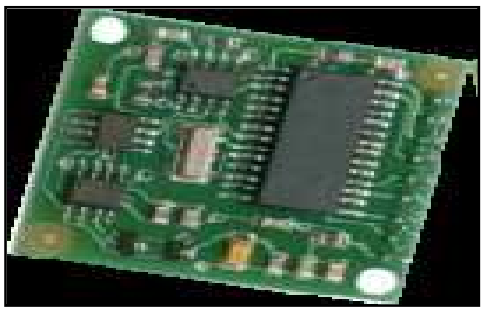
โปรแกรมที่ 2.1 ไฟล์ไลบราลีที่ใช้ทำงานร่วมกับโมดูล SRF04 (ต่อ)

```
    TH1 = 0x00;          //  Initial timer1 counter value at zero
    TL1 = 0x00;
    dat  =  dat  +  (mc/5);  //  Keep positive pulse length
    delay_ms (10);        // Delay 10 ms
}
return (dat/114);          //  Return distance cm scale
```

2.2 โมดูลเข็มทิศดิจิตอล CMPS 03 Digital Compass Module

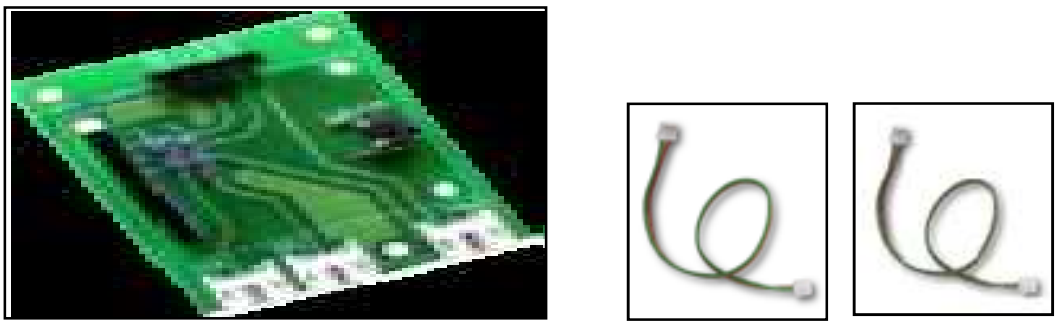
2.2.1 คุณสมบัติ

ใช้ไฟเลี้ยง +5 ต้องการกระแสไฟฟ้า 20 มิลลิเมตร ใช้ตัวตรวจจับสนามแม่เหล็กเบอร์ KMZ51 ของ Philips จำนวน 2 ตัว เพื่อให้สามารถตรวจจับสนามแม่เหล็กได้อย่างสมบูรณ์และมีความละเอียดมากเพียงพอ ความละเอียดของมุม 0.1 องศา ค่าความผิดพลาด 3-4 องศา โดยประมาณ หลังจากการปรับแต่ง เอาต์พุตแบบสัญญาณพัลส์ ความกว้าง 1 ถึง 37 มิลลิวินาที โดยอัตราเพิ่มครั้งละ 0.1 มิลลิวินาที เอาต์พุตข้อมูลดิจิตอลผ่านการติดต่อระบบบัส I^2C รองรับสัญญาณนาฬิกาความถี่สูงถึง 1 MHz โดยให้ข้อมูล 2 รูปแบบคือ 0-255 และ 0-3599 ขนาดเล็กเพียง 32 คูณ 35 มิลลิเมตร สื่อสารกับไมโครคอนโทรเลอร์ยอดนิยมได้ทุกตระกูล อาทิ เบสิกแสตมป์ 2SX/2P, PIC, MCS-51, PSoC, 68HC11 ทั้งผ่านระบบบัส I^2C และด้วยการวัดสัญญาพัลส์



รูปที่ 2.6 แสดงโมดูลเข็มทิศดิจิตอล CMPS 03 Digital Compass Module

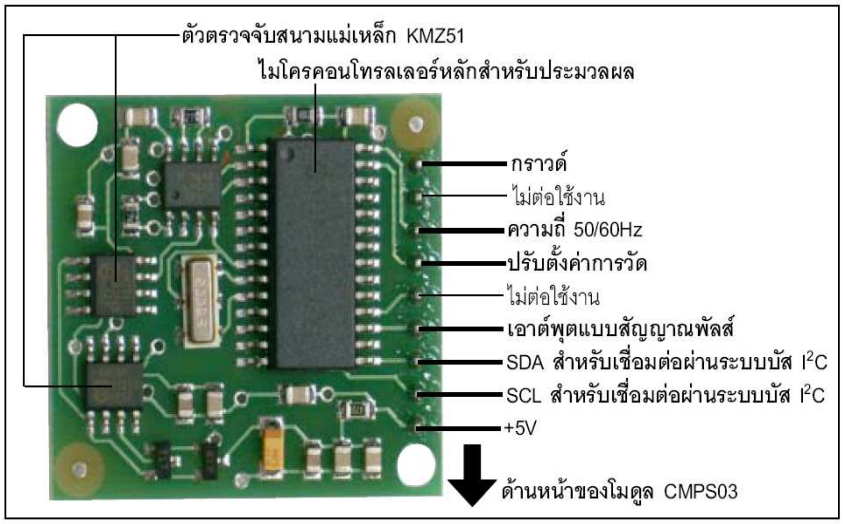
โมดูลเข็มทิศดิจิตอล CMPS03 เป็นผลงานของ Devantech (www.radio-electronics.co.uk) ออกแบบมาเพื่อช่วยในการกำหนดทิศทางการเคลื่อนที่ของหุ่นยนต์อัตโนมัติ และนำมาใช้ในการสร้างเครื่องมือวัดและตรวจสอบทิศระบบอิเล็กทรอนิกส์ โดยหัวใจสำคัญของโมดูล CMPS03 คือตัวตรวจจับสนามแม่เหล็กเบอร์ KM51 ของ Philip จำนวน 2 ตัว เพื่อให้มีความไวเพียงพอในการตรวจสนามแม่เหล็กโลก (Earth magnetic) และไมโครคอนโทรเลอร์เพื่อรับสัญญาณจากตัวตรวจจับมาประมวลผลเป็นข้อมูลดิจิตอลและสัญญาณพัลส์สำหรับแจ้งผลการวัดทิศทาง



รูปที่ 2.7 แสดงบอร์ด ADX-CMPS03 และ สาย PCB3A

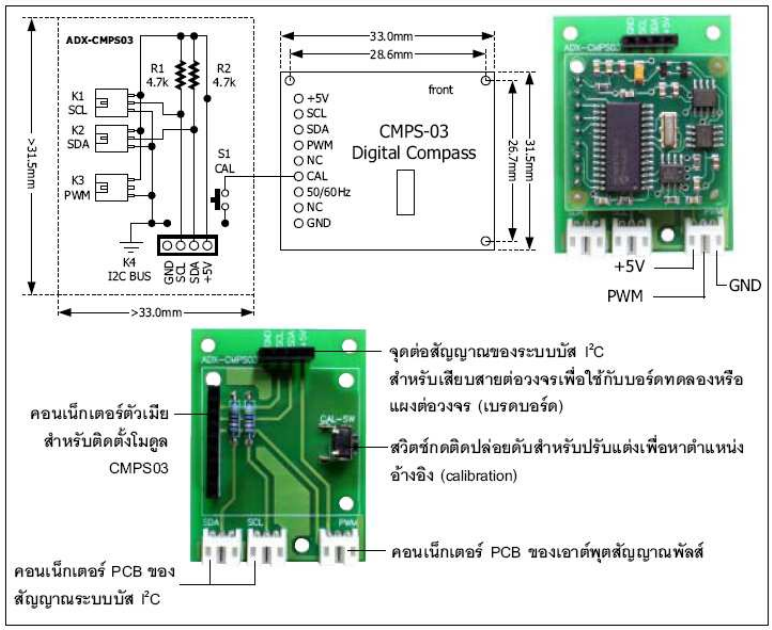
2.2.2 ตำแหน่งขาและการใช้งาน

จากรูปที่ 2.10 แสดงรูปร่างหน้าตาและการจัดขาของ CMPS03 โมดูลเข็มทิศดิจิตอล จะเห็นว่าเป็นแผงวงจรที่คอนเน็กเตอร์ต่อออกมาเพื่อให้เชื่อมต่อไปใช้งาน อย่างไรก็ตามเพื่ออำนวยความสะดวกแก่ผู้ใช้งานกับบอร์ดควบคุมหุ่นยนต์ของบริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด (i-nex: เป็นตัวแทนจำหน่ายสินค้าของ Devantech ในประเทศไทยอย่างเป็นทางการ) จึงได้พัฒนาบอร์ดอะแดปเตอร์รุ่น ADX-CMPS03 เพื่อให้นำโมดูล CMPS03 มาติดตั้ง (โดยบอร์ด ADX-CMPS03 ต้องจัดซื้อแยก)



รูปที่ 2.8 แสดงรูปร่างและตำแหน่งขาสำหรับการต่อใช้งาน

บอร์ด ADX-CMPS03 ได้จัดเตรียมคอนเน็กเตอร์ PCB 3 ขาตัวผู้สำหรับเชื่อมต่อกับบอร์ดควบคุมหุ่นยนต์ และคอนเน็กเตอร์ IDC ตัวเมียแถวเดี่ยว ขาสำหรับเสียบสายต่อวงจรเบอร์AWG#22 เพื่อต่อกับแผงวงจรหรือเบรดบอร์ด นอกจากนั้นยังมีสวิตซ์กดสำหรับปรับตั้งค่า (calibration) เพื่อกำหนดตำแหน่งอ้างอิง โดยวงจรของบอร์ด ADX-CMPS03 แสดงดังรูป 2.11



รูปที่ 2.9 แสดงวงจรของบอร์ด ADX-CMPS03 และ การเชื่อมต่อโมดูล CMP03

2.2.3 การปรับแต่งค่าทิศทางอ้างอิงโมดูล CMPS03

เพื่อให้การวัดทิศทางของโมดูล CMPS03 มีความแม่นยำมากที่สุด จึงมีอินพุตสำหรับการปรับแต่งค่าทิศทางอ้างอิง ทั้งนี้เพื่อประโยชน์ในการกำหนดอ้างอิงเฉพาะสำหรับผู้ใช้งาน โดยต้องป้อนสัญญาณลอจิก “0” เข้าที่ขาอินพุตสำหรับปรับแต่งโมดูล CMPS03 ซึ่งก็คือขา 6 หากใช้บอร์ด ADX-CMPS03 กับโมดูล CMPS03 จะมีสวิตช์กดติดปล่อยดับติดตั้งไว้ให้แล้ว การปรับแต่งมีขั้นตอนดังนี้

2.2.3.1 วางโมดูล CMPS03 ขนานกับพื้น หันด้านของโมดูลไปทางทิศเหนือกดสวิตช์ 1 ครั้ง

2.2.3.2 วางโมดูล CMPS03 ขนานกับพื้น หันด้านของโมดูลไปทางทิศตะวันออก กดสวิตช์

2.2.3.3 วางโมดูล CMPS03 ขนานกับพื้น หันด้านของโมดูลไปทางทิศใต้ กดสวิตช์ 1 ครั้ง

2.2.3.4 วางโมดูล CMPS03 ขนานกับพื้น หันด้านของโมดูลไปทางทิศทางตะวันตก กดสวิตช์

เป็นอันสิ้นสุดการปรับตั้งค่าทิศทางอ้างอิงของโมดูล CMPS03 โดยโมดูลจะเก็บค่าอ้างอิงนี้ไว้ในหน่วยความจำอีอีพรอมและไม่ต้องปรับตั้งค่าใหม่เมื่อจ่ายไฟเลี้ยงครั้งใหม่

2.2.4 การอ่านค่าสัญญาณเอาต์พุตของโมดูล CMPS03

2.2.4.1 การอ่านค่าทิศทางจากเอาต์พุตสัญญาณพัลส์

การอ่านค่าสัญญาณในโหมดนี้ เป็นการนำค่าความกว้างพัลส์ที่ได้จากเอาต์พุตสัญญาณพัลส์ของโมดูล CMPS03 มาระบุตำแหน่งองศา จาก 0 ถึง 359.9 องศา โดยมีย่านของค่าความกว้างสัญญาณพัลส์จาก 1 มิลลิวินาทีไปจนถึง 36.99 มิลลิวินาที มีความละเอียด 0.1มิลลิวินาทีต่อองศา ในสัญญาณพัลส์แต่ละไซเกิล มีช่วงลอจิก “0” กว้าง 65 มิลลิวินาที

ดังนั้นในการนำสัญญาณพัลส์มาประมวลผลเป็นค่ามุม จึงต้องใช้การนับความกว้างของสัญญาณพัลส์เป็นหลักในการคำนวณหาค่ามุมที่โมดูล CMPS03 วัดได้

2.2.4.2 ตัวอย่างโปรแกรมที่ใช้ควบคุมสำหรับเบสิกแสตมป์ 2SX และ i-Stamp

การใช้งานร่วมกับเบสิกแสตมป์ 2SX และ i-Stamp นั้น จะใช้คำสั่ง PULSIN ในการนับสัญญาณพัลส์ โดยจะเพิ่มค่าการนับขึ้นทุกๆ 0.8 ไมโครวินาที ดังนั้นที่ความกว้างของพัลส์ที่ 1 มิลลิวินาทีสำหรับตำแหน่ง 0 องศา เบสิกแสตมป์ 2SX และ i-Stamp จะนับค่าได้เท่ากับ 1,250 จึง

สามารถใช้ค่านี้เป็นจุดอ้างอิงที่ 0 องศา เมื่อต้องการทราบค่ามุมที่แท้จริง ให้นำค่ามุมที่นับได้ลบด้วย 1,250 แล้วหารด้วย125 ก็จะได้มุมในหน่วยองศาที่ต้องการ และรายละเอียดของโปรแกรมแสดงในโปรแกรมที่ 2

ที่ความกว้างพัลส์สูงสุดคือ 36.99 มิลลิวินาที ค่าที่นับได้จากคำสั่ง PULSIN เท่ากับ 46,237 เมื่อลบด้วย 1,250 แล้วหารด้วย 125 เพื่อแปลงเป็นองศา ค่าสูงสุดที่แสดงเป็นผลลัพธ์ได้คือ 359 เป็นค่าหน่วยองศาสูงสุดนั่นเอง

โปรแกรมที่ 2.2 แสดงการอ่านค่าสัญญาณพัลส์จากโมดูล CMPS03

```
'{$STAMP BS2sx}
'{$PBASIC 2.5}
bearing VAR WORD
main:
      PULSIN 4, 1, bearing        ' Get reading
=     (bearing-1250)/125          ' BS2sx - Calculate Bearing
                                  ' in degrees
DEBUG "Compass Bearing ", DEC3 bearing, CR
                                  ' Display Compass Bearing
       GOTO main
```

ตารางที่ 2.1 แสดงตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03

ตำแหน่งรีจิสเตอร์	รายละเอียด
0	ตัวเลขแสดงรุ่นของบอร์ด CMP03
1	ส่งค่าตำแหน่งแบบหยาบ (0-255)
2,3	ส่งค่าตำแหน่งแบบละเอียดด้วยตัวเลข 16 บิต (0-3599) สามารถแปลงค่าเพื่อแสดงองศา 0-359.9 องศาได้โดยตรง
4,5	สำหรับตรวจสอบค่าภายใน โดยจะแสดงค่าความต่างของ Sensor 1 เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย

ตารางที่ 2.1 แสดงตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03 (ต่อ)

ตำแหน่งรีจิสเตอร์	รายละเอียด
6,7	สำหรับตรวจสอบค่าภายใน โดยจะแสดงค่าความต่างของ Sensor 1 เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
8,9	แสดงค่าตัวเลขการปรับแต่งภายใน (calibration value 1) เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
10,11	แสดงค่าตัวเลขการปรับแต่งภายใน (calibration value 2) เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
12,13	ไม่ใช่งานอ่านค่าได้เป็น 0
14	ไม่ใช่งาน ไม่ได้กำหนดค่าไว้
15	คำสั่งสำหรับการปรับแต่งค่า โดยเมื่อต้องการปรับแต่งค่า ต้องเขียนข้อมูล 255 เข้าที่รีจิสเตอร์ตำแหน่งนี้

2.2.5 การอ่านค่าทิศทางเป็นข้อมูลดิจิตอลผ่านระบบบัส I^2C

การอ่านค่าจากโมดูล CMPS03 ให้ได้ค่าที่มีความแม่นยำสูงควรเลือกเอาต์พุตข้อมูลดิจิตอลผ่านระบบบัส I2C โดยโมดูล CMPS03 สามารถส่งข้อมูลของตำแหน่งออกมาที่ความละเอียดสูงสุด 0.1 องศา โดยไม่จำเป็นต้องมีการคำนวณหรือแปลงค่าใด ๆ อีก

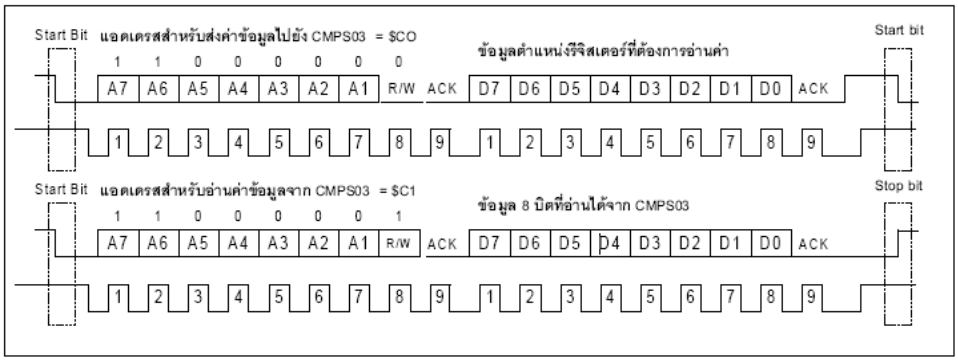
2.2.6 รูปแบบการสื่อสารข้อมูลบัส I^2C

บัส I^2C เชื่อมต่อกับไมโครคอนโทรลเลอร์โดยใช้สายสัญญาณ 2 เส้น ได้แก่ขา SDA (รับและส่งข้อมูล) และ SCL (ขาสัญญาณนาฬิกา) โดยขาสัญญาณทั้งสองจะต้องต่อตัวต้านทานพูลอัปต่อไว้เพื่อกำหนดสถานะลอจิก “1” ให้กับระบบบัส

2.2.7 ลำดับขั้นการติดต่อ

ค่าแอดเดรสของโมดูล CMPS03 คือ $C0 สำหรับการส่งข้อมูล และ $C1 สำหรับการอ่านค่าข้อมูล โดยขั้นตอนการติดต่อกับโมดูล CMPS03 เพื่ออ่านข้อมูลดังนี้

ส่งบิตเริ่มต้นหรือ Start bit เพื่อแจ้งให้ระบบบัส I^2C เตรียมพร้อมรับข้อมูล ส่งค่าแอดเดรส $C0 เพื่อระบุว่าต้องการติดต่อเพื่อเขียนข้อมูลไปยังกับโมดูล CMPS03 ส่งค่าตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03 ที่ต้องการอ่านค่า ซึ่งมีรายละเอียดแสดงในตารางที่ 1 ส่งค่าแอดเดรส $C1 เพื่อระบุว่าต้องการอ่านค่าข้อมูลจากโมดูล CMP03อ่านค่าข้อมูลจากโมดูล CMPS03 มาเก็บไว้ในหน่วยความจำ ส่งบิตหยุดหรือ stop เพื่อหยุดการสื่อสารข้อมูลและกำหนดให้บัสอยู่ในสภาวะบัสว่าง จากลำดับขั้นการติดต่อสื่อสารข้างต้น สามารถนำมาเขียนเป็นโปรแกรมตัวอย่างเพื่ออ่านข้อมูลจากโมดูล CMPS03 โดยใช้เบสิกแสตมป์ 2SX หรือ i-Stamp ได้ดังแสดงในโปรแกรมที่ 2.3



รูปที่ 2.10 แสดงไทมิ่งไดอะแกรมของการติดต่อสื่อสารกับโมดูล CMPS03 ผ่านระบบบัส I^2C

โปรแกรมที่ 2.3 อ่านค่าจากโมดูล CMP03 ผ่านระบบบัส I^2C ของ i-Stamp

```
'{$STAMP BS2sx}
'{$PBASIC 2.5}
SDA             CON         6           ' I2C serial data line
SCL             CON         7           ' I2C serial clock line
WrCMPS03        CON         $C0         ' write to compass
RdCMPS03        CON         $C1         ' read from compass
Ack             CON         0           ' acknowledge bit
Nak             CON         1           ' no ack bit
i2cSDA          VAR         NIB         ' I2C serial data pin
i2cData         VAR         WORD        ' data to/from device
REGISTER        VAR         BYTE        ' register address
i2cWork         VAR         BYTE        ' work byte for TX routine
i2cAck          VAR         BIT         ' Ack bit from device
temp            VAR         WORD        ' for rj printing
digits          VAR         NIB
width           VAR         NIB
Init:
        PAUSE 250
        i2cSDA = SDA                    ' define SDA pin
        REGISTER = 0                    ' compass revision number
        GOSUB Read Byte
        DEBUG 2, 1, 1, "Revision Number = “, DEC2 i2cData
Main:
        REGISTER = 1                    ' Show Data 0-255 for 0-360 degree
```

โปรแกรมที่ 2.3 อ่านค่าจากโมดูล CMP03 ผ่านระบบบัส I^2C ของ i-Stamp (ต่อ)

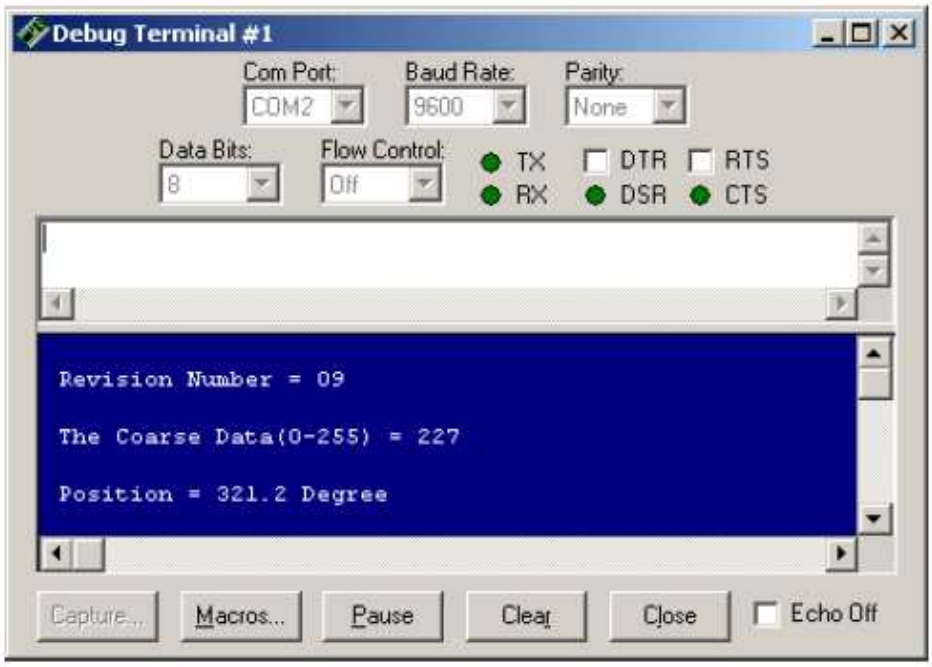
```
        GOSUB Read Byte                 ' Read Byte from I2C
        DEBUG 2, 1, 3, "The Coarse Data (0-255) = “, DEC i2cData
        REGISTER = 2            ' get Data in degrees, 0.0 - 359.9 Degree
        GOSUB Read Word
        DEBUG 2, 1, 5, "Position = ", DEC i2cData/10,".", DEC1 i2cData," Degree"
        PAUSE 250
        GOTO Main
‘Compass Access Subroutines
‘Writes low byte of i2cData to REGISTER
Write Byte:
        GOSUB I2C_Start
        i2cWork = WrCMPS03
        GOSUB I2C_TX_Byte           ' send device address
        i2cWork = REGISTER
        GOSUB I2C_TX_Byte           ' send register number
        i2cWork = i2cData.LOWBYTE
        GOSUB I2C_TX_Byte           ' send the data
        GOSUB I2C_Stop
        RETURN
‘Writes i2cData to REGISTER
Write Word:
        GOSUB I2C_Start
        i2cWork = WrCMPS03
        GOSUB I2C_TX_Byte           ' send device address
        i2cWork = REGISTER
        GOSUB I2C_TX_Byte '         send register number
        i2cWork = i2cData.HIGHBYTE
```

โปรแกรมที่ 2.3 อ่านค่าจากโมดูล CMP03 ผ่านระบบบัส I^2C ของ i-Stamp (ต่อ)

```
        GOSUB I2C_Stop
        RETURN
' Read i2cData (8 bits) from REGISTER
Read_Byte: GOSUB I2C_Start
        i2cWork = WrCMPS03
        GOSUB I2C_TX_Byte          ' send compass address
        i2cWork = REGISTER
        GOSUB I2C_TX_Byte          ' send register number
        GOSUB I2C_Start             repeat start (sets register)
        i2cWork = RdCMPS03
        GOSUB I2C_TX_Byte          ' send read command
        GOSUB I2C_RX_Byte_Nak
        GOSUB I2C_Stop
I       2cData = i2cWork           ' return the data
        RETURN
'Read i2cData (16 bits) from REGISTER
Read Word: GOSUB I2C_Start
        i2cWork = WrCMPS03
        GOSUB I2C_TX_Byte          ' send compass address
        i2cWork = REGISTER
        GOSUB I2C_TX_Byte          ' send register number
        GOSUB I2C_Start            ' repeat start (sets register)
        i2cWork = RdCMPS03
        GOSUB I2C_TX_Byte          ' send read command
        GOSUB I2C_RX_Byte
        i2cData.HIGHBYTE = i2cWork  ' read high byte of data
        GOSUB I2C_RX_Byte_Nak
```

โปรแกรมที่ 2.3 อ่านค่าจากโมดูล CMP03 ผ่านระบบบัส I^2C ของ i-Stamp (ต่อ)

```
'Low Level I2C Subroutines
'Start
I2C_Start:      INPUT i2Csda                        ' I2C start bit sequence
                INPUT SCL
                LOW i2cSDA                          ' SDA -> low while SCL high
Clock Hold:
        IF (INS.LOWBIT(SCL) = 0) THEN Clock Hold    ' device ready?
        RETURN
'Transmit
I2C_TX_Byte:
        SHIFTOUT i2cSDA, SCL, MSBFIRST,[i2cWork\8]      ' send byte to device
        SHIFTIN i2cSDA, SCL, MSBPRE,[i2cAck\1]          ' get acknowledge bit
        RETURN
'Receive
I2C_RX_Byte_Nak:    i2cAck = Nak                    ' no Ack = high
                    GOTO I2C_RX
I2C_RX_Byte:        i2cAck = Ack                    ' Ack = low
I2C_RX:
        SHIFTIN i2cSDA, SCL, MSBPRE, [i2cWork\8]        ' get byte from device
        SHIFTOUT i2cSDA, SCL, LSBFIRST,[i2cAck\1]       ' send ack or nak
        RETURN
'Stop
I2C_Stop:       LOW i2cSDA                          ' I2C stop bit sequence
                INPUT SCL
                INPUT i2cSDA                        ' SDA --> high while
SCL
RETURN
```



รูปที่ 2.11 แสดงหน้าจอของ Debug terminal แสดงข้อมูลที่อ่านได้จากโมดูล CMPS03

2.2.8 การทำงานของโปรแกรมที่ 2 ติดต่อกับโมดูล CMPS03 ผ่านระบบบัส I^2C ด้วย i-Stamp

เนื่องจากเบสิกแสตมป์ 2SX และ i-Stamp ไม่มีคำสั่งติดต่อกับระบบบัส I^2C ดังนั้น โปรแกรมติดต่อโมดูล CMPS03 ผ่านระบบบัส I^2C จึงค่อนข้างยาว เนื่องจากต้องสร้างโปรแกรมย่อยสำหรับการสื่อสารข้อมูลกับระบบบัส I^2C หลังจากนี้ผู้ใช้งานสามารถนำโปรแกรมย่อยนี้ไปประยุกต์ใช้งานกับอุปกรณ์ตัวอื่น ๆ ที่ได้ใช้การติดต่อระบบบัส I^2C ได้ทันที สำหรับขั้นตอนการทำงานหลัก ๆ ของโปรแกรมที่ 2 มีดังนี้

กำหนดรีจิสเตอร์เป็น 0 เพื่อติดต่ออ่านค่าเวอร์ชันของโมดูล CMPS03 จากนั้นอ่านค่ามาแสดงที่หน้าต่าง Debug terminal ให้โปรแกรมวนลูปที่โปรแกรมหลัก จากนั้นส่งค่ารีจิสเตอร์เท่ากับ 1 เพื่ออ่านค่าข้อมูลแบบหยาบออกมา แล้วนำมาแสดงผลที่หน้าต่าง ๆ Debug terminal ส่งค่ารีจิสเตอร์เท่ากับ 2 เพื่ออ่านค่าข้อมูลแบบละเอียด จากนั้นสั่งอ่านค่าข้อมูลจากบัส I^2C แบบเวิร์ด (อ่านข้อมูลออกมา 16 บิต)นำค่าข้อมูลที่ได้หารด้วย 10 ก่อนเพื่อแปลงค่าที่ได้เป็นองศา แสดงที่

หน้าต่าง Debug terminal จากนั้นนำค่าหลักสุดท้ายมาแสดง ซึ่งตำแหน่งสุดท้ายเป็นตำแหน่งของจุดทศนิยม

2.2.9 การปรับแต่งค่าของโมดูล CMPS03 ผ่านทางระบบบัส I^2C

การปรับแต่งค่าทำโดยการส่งค่า 0xFF ไปยังรีจิสเตอร์ 15 ของโมดูล CMPS03 โดยจะต้องส่งค่า 4 ครั้งและระบุทิศทางหลัก ๆ 4 ทิศทางเช่นเดียวกับการกำหนดค่าด้วยสวิตซ์โดยตรง มีขั้นตอนดังนี้

2.2.9.1 วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศเหนือ จากนั้นเขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

2.2.9.2 วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันออก เขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

2.2.9.3 วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศใต้ กดสวิตซ์ เขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

2.2.9.4 วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันตก กดสวิตซ์ เขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

หลังจากปรับแต่งค่าแล้วค่าที่ปรับแต่งจะเก็บไว้ที่หน่วยความจำอีอีพรอม ดังนั้นแม้ไม่จ่ายไฟให้กับบอร์ด ข้อมูลที่ปรับแต่งแล้ว จะยังคงอยู่ต่อไป

2.3 การควบคุมมอเตอร์ด้วยรีเลย์

รีเลย์เป็นสวิตช์ที่ทำงานโดยอาศัยแม่เหล็ก ทำให้เกิดการตัดหรือต่อวงจรไฟฟ้าเช่นใช้ในการควบคุมมอเตอร์เป็นต้น

รีเลย์ที่ใช้กันอยู่ในปัจจุบันสามารถแบ่งออกเป็นชนิดต่างๆตามหลักการทำงานได้ 2 ชนิด คือ Electromechanical Relay และ Solid State relay

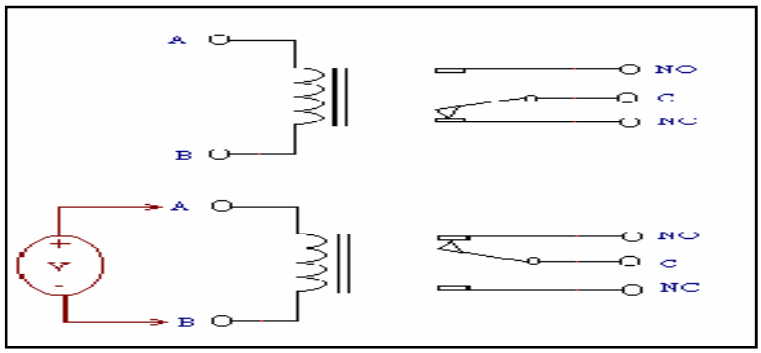
โดยโครงงานนี้ได้ศึกษาและใช้งานรีเลย์ชนิด Electromechanical Relay

2.3.1 Electromechanical Relay

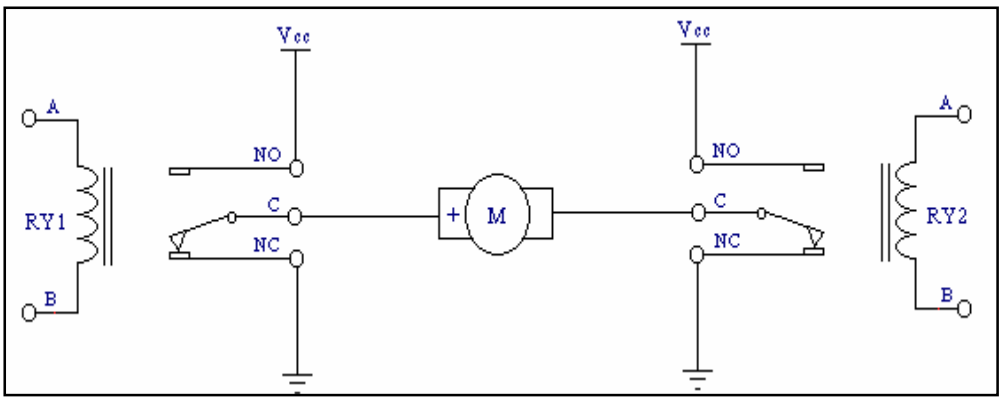
Electromechanical Relay คือรีเลย์ที่อาศัยกระแสไฟฟ้าสร้างแรงดึงดูดหรือแรงแม่เหล็กทำให้เกิดการเคลื่อนที่ทางกลของหน้าสัมผัส ภายในโครงสร้างของ รีเลย์ จะประกอบไปด้วยขดลวด (Coil) 1 ชุด และ หน้าสัมผัส (Contactor) ซึ่งในหน้าสัมผัส 1 ชุด จะประกอบไปด้วย

2.3.1.1 หน้าสัมผัสแบบปกติปิด (Normally Close หรือ NC.) ซึ่งในสภาวะปกติ ขานี้จะต่ออยู่กับขาร่วม (Common)

2.3.1.2 หน้าสัมผัสแบบปกติเปิด (Normally Open หรือ NO.) ขานี้จะต่อเข้ากับขาร่วม (Common) เมื่อขดลวดมีแรงดันตกคร่อม หรือกระแสไหลผ่าน (ในปริมาณที่เพียงพอ) ในรีเลย์ 1 ตัว อาจมีหน้าสัมผัสมากกว่า 1 ชุด เช่น 2 ชุด, 4 ชุด เป็นต้น เมื่อขดลวดได้รับแรงดันตกคร่อม (ขา A และ B) จะทำให้มีกระแสไหลผ่านขดลวด ซึ่งจะทำให้เกิดอำนาจสนามแม่เหล็ก ดึงดูดให้หน้าสัมผัส NOและ C ติดกัน



รูปที่ 2.12 โครงสร้างของรีเลย์วงจรของรีเลย์

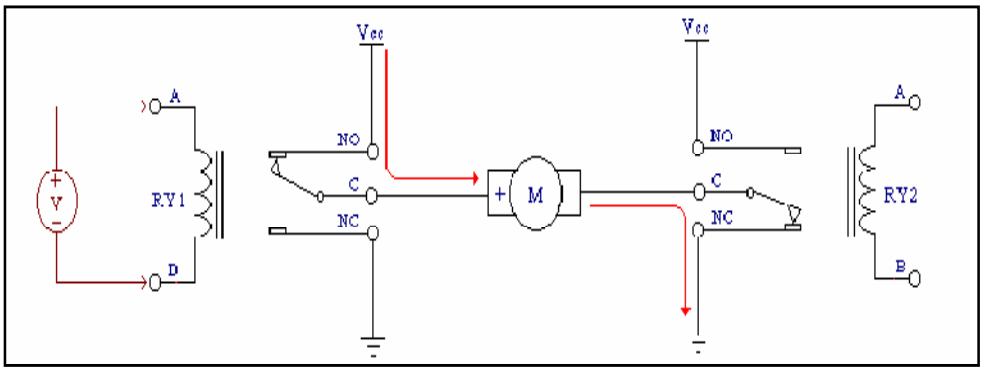


รูปที่ 2.13 วงจรของรีเลย์

วงจรที่เราจะทดลองกันในคราวนี้ จะประกอบไปด้วย รีเลย์ 2 ตัว คือ RY1 และ RY2 ซึ่ง Loadก็คือ DC-Motor ซึ่งต่ออยู่กับขาร่วม (C.) ของ RY1. และ RY2. โดยขั้วบวก (+) ของมอเตอร์ ต่ออยู่ที่ขา C. ของRY1 และขั้วลบ (-) ของมอเตอร์ ต่ออยู่ที่ขา C. ของ RY2 โดยที่ขา NO. ของ RY1 และ RY2 จะต่ออยู่กับขั้วบวกของแหล่งจ่ายไฟ ที่จะจ่ายให้มอเตอร์ (Vcc) และขา NC. ของ RY1 และ RY2 จะต่อลงกราวด์

การทำงานจะแสดงออกเป็น 2 กรณีคือ

กรณีที่ 1 RY 1 ทำงาน

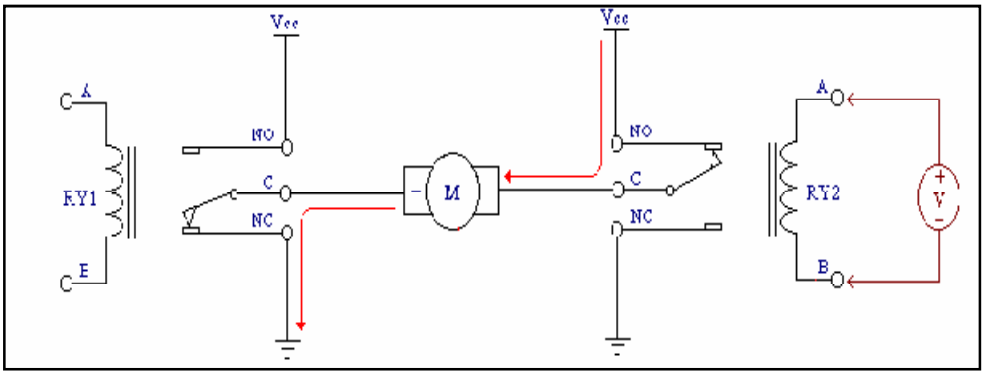


รูปที่ 2.14 ลักษณะการทำงานของ RY 1

เมื่อ RY1 ทำงาน (มีกระแสไหลผ่านขดลวดในปริมาณที่เพียงพอ) จะทำให้เกิดอำนาจสนามแม่เหล็กไฟฟ้า ดึงดูดให้ขา NO และขา C ของ RY1 ติดกัน ส่งผลให้มีกระแสไหลจาก

แหล่งจ่าย (Vcc) ผ่านเข้าสู่ขั้วบวก (+) ของมอเตอร์ ผ่านไปยังขา C ของ RY2 ซึ่งต่ออยู่ที่ NC และลงกราวด์ ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางบวก และครบวงจร จึงทำให้มอเตอร์สามารถหมุนในทิศทาง Forward ได้

กรณีที่ 2 RY 2 ทำงาน



รูปที่ 2.15 ลักษณะการทำงานของ RY 2

เมื่อ RY2 ทำงาน (มีกระแสไหลผ่านขดลวดในปริมาณที่เพียงพอ) จะทำให้เกิดอำนาจสนามแม่เหล็กไฟฟ้า ดึงดูดให้ขา NO และขา C ของ RY2 ติดกัน ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (Vcc) ผ่านเข้าสู่ขั้วลบ (-) ของมอเตอร์ ผ่านไปยังขา C ของ RY1 ซึ่งต่ออยู่ที่ NC และลงกราวด์ ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางลบ และครบวงจร จึงทำให้มอเตอร์สามารถหมุนในทิศทาง Reward ได้

2.4 โฟโต้ทรานซิสเตอร์ (Photo Transistor)

โฟโต้ทรานซิสเตอร์ มีโครงสร้างแตกต่างจากโฟโต้ไดโอด คือบริเวณที่รับแสงคือรอยต่อพี-เอ็นระหว่างเบสกับคอลเลกเตอร์ (B-C) และกระแสไหลผ่านทรานซิสเตอร์ระหว่างขั้วอิมิตเตอร์กับคอลเลกเตอร์ โฟโต้ทรานซิสเตอร์เปรียบเหมือนสวิตซ์แสง (Light Switch) เพราะเมื่อมีแสงตกกระทบรอยต่อระหว่าง B-C จะทำให้ทรานซิสเตอร์ทำงาน มีกระแสคอลเลตเตอร์ไหล(I_C) ตามสมการ

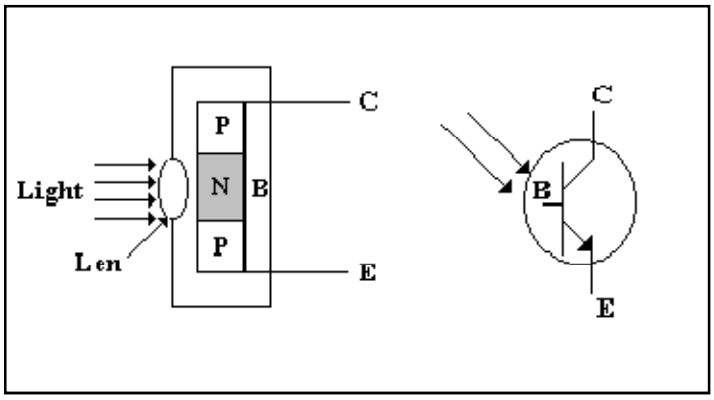
$$I_C = \beta I_B \qquad (1)$$

เมื่อ I_C กระแสคอลเลกเตอร์

β อัตราขยายกระแสของโฟโต้ทรานซิเตอร์

I_B กระแสเบสเกิดจากแสงสว่างตกกระทบระหว่างรอยต่อ B-C

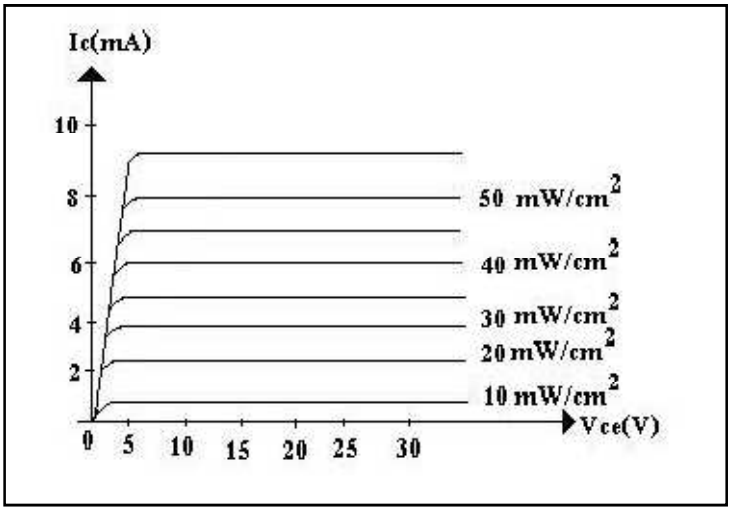
ลักษณะของโครงสร้างและสัญลักษณ์ของโฟโต้ทรานซิสเตอร์ ดังรูปที่ 2.18



รูปที่ 2.16 โครงสร้างและสัญลักษณ์ของโฟโต้ทรานซิสเตอร์

2.4.1 การทำงานของโฟโต้ทรานซิสเตอร์

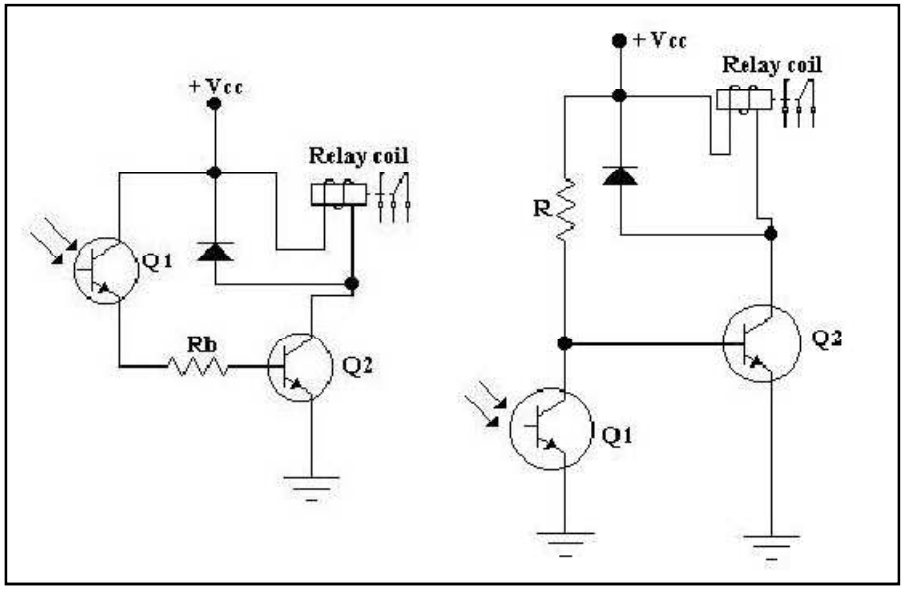
เมื่อไบอัสโฟโต้ทรานซิสเตอร์ โดยไบอัสแรงดันระหว่างขั้วอิมิตเตอร์กับคอลเลกเตอร์สำหรับเบสไม่ต้องให้ไบอัส ขณะที่ไม่มีแสงตกกระทบรอยต่อของเบส-คอลเลกเตอร์ ทรานซิสเตอร์จะไม่ทำงาน จะมีกระแสรั่วไหลระหว่างรอยต่อคอลเลกเตอร์กับอิมิตเตอร์จำนวนหนึ่ง เรียกว่า Dark Current เมื่อแสงที่มีความสว่างเล็กน้อยที่รอยต่อเบส-คอลเลกเตอร์จะทำให้กระแส I_C ไหลได้ และเพิ่มความสว่างของแสงให้มากขึ้น ค่ากระแส I_C จะสูงขึ้นตามไปด้วย ตามกราฟในรูปที่ 2.21



รูปที่ 2.17 กราฟแสดงลักษณะสมบัติของโฟโต้ทรานซิสเตอร์

2.4.2 การประยุกต์ใช้งานโฟโต้ทรานซิสเตอร์

นิยมนำโฟโต้ทรานซิสเตอร์ไปใช้เป็นสวิตซ์แสงหรือตรวจจับแสง(Lighting Sensor) เมื่อแสงตกกระทบที่ตัวโฟโต้ทรานซิสเตอร์บริเวณเลนส์ด้านบน จะทำให้ทรานซิสเตอร์ทำงานได้ในลักษณะเปิด-ปิด (On-Off Control) สามารถควบคุมการเปิด-ปิดของรีเลย์หรืออุปกรณ์ไฟฟ้าอื่น ๆ ได้ ดังรูปที่ 2.22

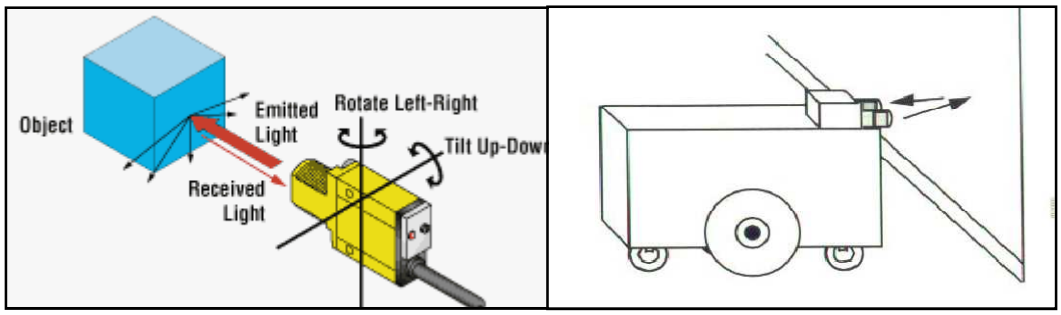


รูปที่ 2.18 การนำโฟโต้ทรานซิสเตอร์ ประยุกต์เป็นวงจรควบคุมการเปิด-ปิด

2.5 เซนเซอร์อินฟราเรด

เซนเซอร์อินฟราเรดเป็นเซนเซอร์ไม่อาศัยการสัมผัสที่นิยมมากในงานตรวจจับวัตถุ โดยเฉพาะอย่างยิ่งในงานหุ่นยนต์เคลื่อนที่ เซนเซอร์อินฟราเรดทำงานโดยการปล่อยแสงอินฟราเรดและรอตรวจจับว่ามีแสงอินฟราเรดสะท้อนกลับมาหรือไม่ ถ้ามี ก็แสดงว่ามีวัตถุขวางในระยะใกล้พอ (รูปที่ 2.21) ระยะทำงานของเซนเซอร์อินฟราเรดอยู่ประมาณ 50-100 ซ.ม. ในทางปฏิบัติแสงอินฟราเรดที่ปล่อยออกไปจะถูกเข้ารหัสด้วยการมอดูเลตกับความถี่ต่ำๆ เช่น 100 เฮิรตซ์ ทั้งนี้เพื่อจะได้ไม่สับสนกับแสงอินฟราเรดที่เกิดจากแสงแดด หรือหลอดไฟฟลูออเรสเซนซ์ (หลอดวาวแสง) ระดับความเข้มของแสงอินฟราเรดที่สะท้อนกลับไม่ได้ขึ้นกับระยะทางอย่างเดียว (ยิ่งห่างก็สะท้อนกลับน้อย) แต่ขึ้นกับสีและลักษณะของพื้นผิว เช่นวัตถุสีดำไม่สะท้อนแสงอินฟราเรด วัตถุผิวขรุขระสะท้อนแสงได้น้อยกว่าวัตถุผิวเรียบมัน ข้อจำกัดนี้ทำให้ยากที่จะใช้เซนเซอร์อินฟราเรดใน

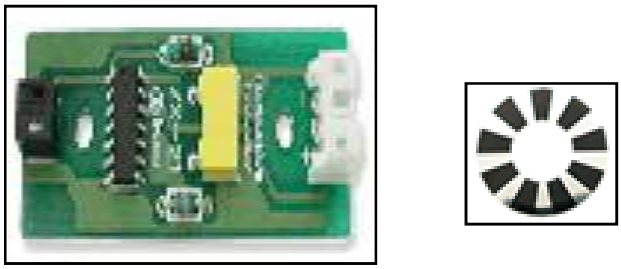
การวัดระยะห่างกับวัตถุที่ตรวจจับได้ นั่นก็คือบางครั้งเซนเซอร์อินฟราเรดอาจไม่สามารถตรวจจับวัตถุที่ขวางอยู่ได้ แต่อย่างไรก็ตามหากเซนเซอร์อินฟราเรดตรวจจับวัตถุขวางหน้าอยู่ได้ ก็เป็นการแน่นอนว่ามีวัตถุขวางอยู่จริง (เป็นไปได้ยากที่จะมีสัญญาณหลอกเกิดขึ้น) และถึงแม้ความเข้มของแสงอินฟราเรดที่รับได้จะบอกได้บ้างถึงระยะห่างจากวัตถุกีดขวาง แต่ข้อมูลนี้ไม่มีความแม่นยำนัก รวมทั้งระยะทำงานของเซนเซอร์อินฟราเรดค่อนข้างสั้น จึงเป็นการดีที่เริ่มทำการหลบหลีกหรือปรับการเคลื่อนที่ให้เหมาะสมทันทีที่ตรวจจับวัตถุได้จากเซนเซอร์อินฟราเรด



รูปที่ 2.19 การทำงานของเซนเซอร์อินฟราเรด

2.5.1 แผงวงจรตรวจจับรหัสล้อ ZX-21

อาศัยหลักการสะท้อนของแสงอินฟราเรดแผงวงจรนี้จะประกอบด้วยวงจรที่ช่วยจัดสัญญาณดิจิตอล เมื่อตัวโฟโต้ทรานซิสเตอร์ได้รับแสงอินฟราเรดสะท้อนกลับน้อยจะได้เอาต์พุตเป็น "1" หากได้รับแสงอินฟราเรดสะท้อนกลับมามากจะให้เอาต์พุตเป็น "0"



รูปที่ 2.20 แสดงแผงวงจรตรวจจับรหัสล้อ ZX-21

2.6 ไมโครคอนโทรลเลอร์ตระกูล 51

2.6.1 คุณสมบัติของไมโครคอนโทรลเลอร์ MCS-51

2.6.1.1 เป็นไมโครคอนโทรเลอร์ขนาด 8 บิต

2.6.1.2 มีหน่วยความจำภายในแบบแฟลตขนาด 4 กิโลไบต์ หรือ 8 กิโลไบต์ ที่โปรแกรมภายในวงจรสามารถเขียนและลบได้ถึงพันครั้ง

2.6.1.3 มีสายสัญญาณสำหรับต่อกับอินพุต/เอาต์พุตได้ 32 เส้น (แบบ 2 สองทิศทาง)

2.6.1.4 มีหน่วยความจำชั่วคราว (ROM) ภายในขนาด 128 ไบต์หรือ 256 กิโลไบต์

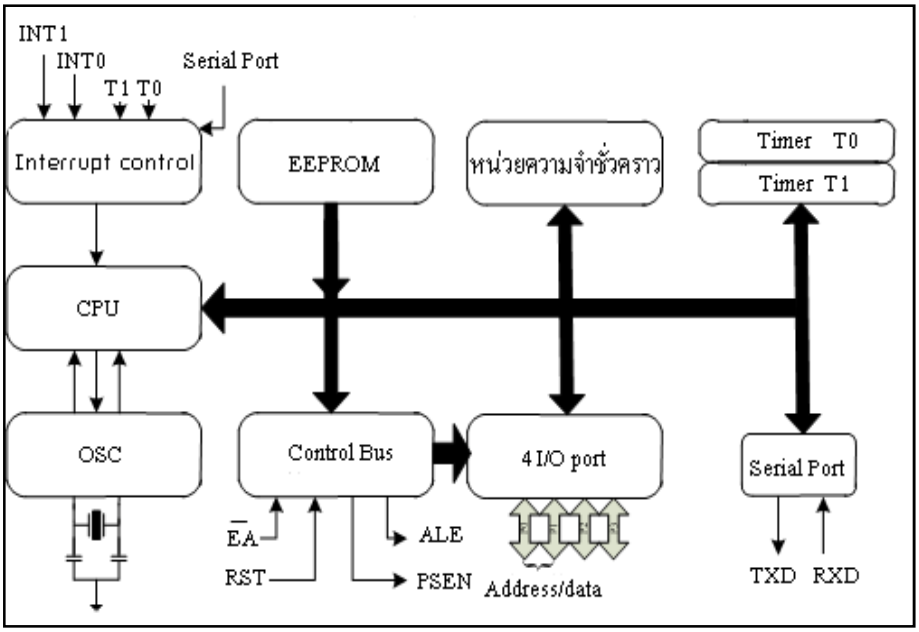
2.6.1.5 ใช้ความถี่สัญญาณนาฬิกาตั้งแต่ 0 Hz จนถึง 24 Hz

2.6.1.6 มีวงจรตั้งเวลาและนับสัญญาณเวลาขนาด 16 บิต จำนวน 2 หรือ 3 ชุด

2.6.1.7 มีวงจรรับสัญญาณอินเตอร์รัพท์ได้ไม่ต่ำกว่า 6 ชนิด

2.6.1.8 สามารถต่อขยายหน่วยความจำภายนอกได้สูงสุด 64 กิโลไบต์

มีวงจรสื่อสารแบบสื่อสาร 2 ทางเต็มอัตรา และมีคำสั่งที่ใช้ภาษาแอสแซมบลีทั้งหมด 111 คำสั่ง โครงสร้างพื้นฐานของ MCS-51 มีส่วนประกอบต่างๆ ดังในรูปที่ 2.25



รูปที่ 2.21 โครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์

รายการไอซีไมโครคอนโทรลเลอร์ตระกูล MCS-51 ที่ผู้ผลิตได้สร้างขึ้นมาหลายรุ่นนั้นก็เพื่อให้เหมาะสมกับการประยุกต์ใช้งานแต่ละประเภท ดังรูปที่ 2.2 แสดงจำนวนของหน่วยความจำภายใน วงจรตั้งเวลา/นับเวลา และระดับของการอินเตอร์รัพท์ของแต่ละรุ่น

ตารางที่ 2.2 แสดงรายละเอียดของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรเลอร์	**หน่วยความจำภายใน (internal memory)**		**ตั้งเวลา/นับเวลา (time/counter)**	**สัญญาณอินเตอร์รัพท์**
	หน่วยความจำภายในแบบ EPROM , EEPROM	ข้อมูล RAM		
8051	4 kb ✕ 8 Rom	128 ✕8 bit	2 ✕ 16 bit	6
8051AH	4 kb ✕ 8 Rom	128 ✕8 bit	2 ✕ 16 bit	5
8051AH	8 kb ✕ 8 Rom	256 ✕ 8 bit	3 ✕ 16 bit	6
8031AH	ไม่มี	128 ✕8 bit	2 ✕ 16 bit	5
8032AH	ไม่มี	256 ✕ 8 bit	3 ✕ 16 bit	5
8031	ไม่มี	128 ✕8 bit	2 ✕ 16 bit	5
8751H	4 kb ✕ 8 Rom	128 ✕8 bit	2 ✕ 16 bit	5
8751H-12	4 kb ✕ 8 Rom	128 ✕8 bit	2 ✕ 16 bit	5

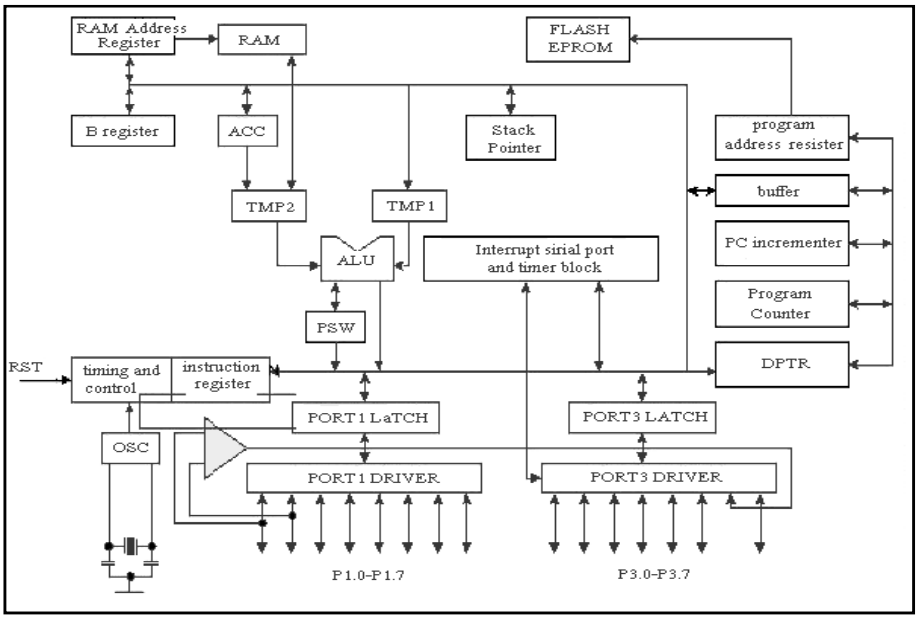
นอกจากนี้ยังมีอีกหลายบริษัทที่ทำการผลิตไมโครคอนโทรลเลอร์MCS-51พร้อมทั้งมีการพัฒนาเพิ่มเติมขึ้น ในส่วนของความเร็วและหน่วยความจำภายในโดยใช้หน่วยความจำแบบแฟลช (Flash Memory) ทำให้ประยุกต์ใช้งานได้ง่ายและเป็นที่นิยมใช้กันมากขึ้น ส่วนโครงสร้างและคำสั่งของโปรแกรมก็ยังใช้เหมือนเดิม แสดงตัวอย่างดังตารางที่ 2.3

ตารางที่ 2.3 ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ที่ผลิตโดยบริษัท ATMEL

ไมโครคอนโทรเลอร์	**หน่วยความจำภายใน (internal memory)**		**ตั้งเวลา/นับเวลา (time/counter)**	**สัญญาณอินเตอร์รัพท์**
	หน่วยความจำภายในแบบ EPROM,EEPROM	ข้อมูล RAM		
AT89C1051	1 kb × 8	64 ×8 bit	2 × 16 bit	6
AT89C2051	2 kb × 8	128 ×8 bit	2 × 16 bit	6
AT89C4051	4 kb × 8	128 × 8 bit	2 × 16 bit	6
AT89C51	4 kb × 8	128 ×8 bit	2 × 16 bit	6
AT89C52	8 kb × 8	256 × 8bit	3 × 16 bit	8
AT89S52	8 kb × 8	256 × 8bit	3 × 16 bit	8
AT89C55	20 kb × 8	256 × 8 bit	3 × 16 bit	8
AT89S8252	8 KB × 8 (2kb EEPROM)	256 × 8 bit	3 × 16 bit	9
AT89S53	12 kb × 8	256 × 8 bit	3 × 16 bit	9

2.6.2 โครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51

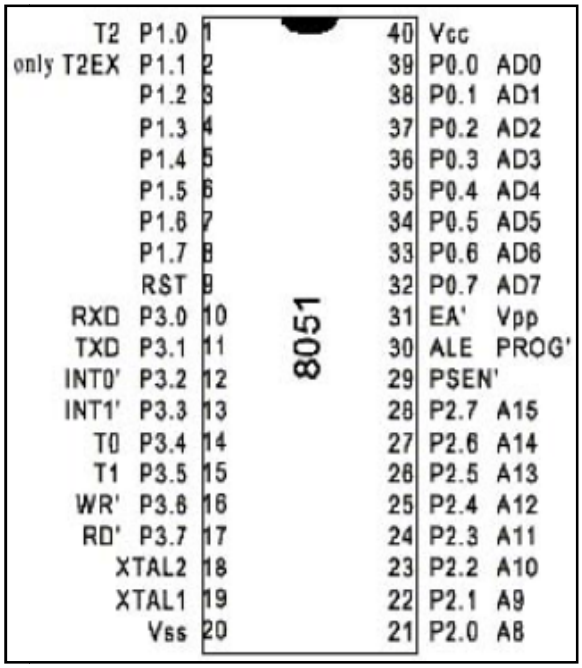
วงจรภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51 ประกอบด้วยวงจรอินพุตและเอาต์พุตทั้งหมด 4 พอร์ต แต่ละพอร์ตจะเป็น 8 บิต หน่วยความจำโปรแกรมภายใน (EPROM , EEPROM และ Flash) หน่วยความจำที่เป็นข้อมูลนั้น (RAM) ซึ่งอยู่จะในวงจรหลักของไมโครคอนโทรลเลอร์ตระกูล MCS-51 ตลอดจนวงจรการคำนวณทางคณิตศาสตร์และลอจิก (ALU) วงจรรีจีสเตอร์ทั่วไป และรีจีสเตอร์ฟังก์ชั่นการใช้งานเฉพาะ แสดงดังรูปที่ 2.26



รูปที่ 2.22 โครงสร้างภายในของไมโครคอนโทรลเลอร์ MCS-51

2.6.3 การจัดตำแหน่งขาของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกเบอร์นั้นจะมีโครงสร้างและการใช้งานพื้นฐานเหมือนกันดังตัวอย่างนี้ เช่น แบบดิป (DIP) ซึ่งมีทั้งหมด 40 ขา ได้แบ่งการใช้งานออกเป็น อินพุต/เอาต์พุต (Input/output Port) ขาสัญญาณควบคุม ขาสัญญาณกำหนดตำแหน่งหน่วยความจำ และขาสัญญาณข้อมูล ดังรูป 2.27



รูปที่ 2.23 ตำแหน่งขาของไมโครคอนโทรลเลอร์ตระกูล MCS-51

2.6.4 ตำแหน่งของไมโครคอนโทรลเลอร์ตระกูล MCS-51และหน้าที่การทำงาน

2.6.4.1 P0.0-P0.7 (ขาที่ 32-39) พอร์ต 0 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอก ได้ 2 ทิศทาง สามารถรับข้อมูลอินพุตและส่งข้อมูลเอาต์พุตได้ มีขนาด 8 บิต การตั้งค่าให้พอร์ต 0 รับข้อมูลอินพุตทำได้โดยการส่งค่าสัญญาณ 1 ไปยังบิตที่ ต้องการให้รับข้อมูลอินพุต วงจรภายในจะทำให้บิตนั้นมีค่าความต้านทานสูงและ สามารถรับข้อมูลอินพุตได้ และยังใช้เป็นขาสัญญาณกำหนดตำแหน่งหน่วยหน่วยความจำ (A0-A7) และขาสัญญาณข้อมูล (D0-D7)โดยการใช้ตัวแยก สัญญาณ (D-latch 74LS373) ทำหน้าที่เป็นมัลติเพล็กซ์ (Multiplex) โดยเลือกช่วงเวลาของ สัญญาณกำหนดตำแหน่งหน่วยความจำและสัญญาณข้อมูลออก จากกัน ในขณะที่ใช้เป็นอินพุตและเอาต์พุต วงจรภายในจะไม่ม่วงจรเพิ่ม กระแสไฟฟ้า (Pull up) จึงจำเป็นต้อง ต่อวงจรเพิ่มกระแสไฟฟ้าจากภายนอกเข้าไป

2.6.4.2 P1.0-P1.7 (ขาที่ 1.8) พอร์ต 1 ทำหน้าที่เป็นสัญญาณควบคุมการอุปกรณ์ภายนอก ได้ 2 ทิศทาง สามารถปรับได้ทั้งอินพุตและเอาต์พุต มีขนาด 8 บิต สามารถอ้างอิง ถึงการทำงานได้ที่ละบิต และวงจรภายในมีตัวต้านทานเพิ่มกระแสไฟฟ้า (Pull up) ในกรณีที่ต้องการให้รับข้อมูลอินพุตก็สามารถทำได้เหมือนพอร์ต 0

2.6.4.3 P2.0-P2.7 (ขาที่ 21-28) พอร์ต 2 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอก ได้ทั้ง 2 ทิศทาง คือ เป็นได้เป็นทั้งอินพุตและเอาต์พุต มีขนาด 8 บิต สามารถใช้เป็นขาสัญญาณที่กำหนดตำแหน่งหน่วยความจำ (A8-A15) และมี วงจรเพิ่มกระแสไฟภายใน การกำหนดให้เป็นขาอินพุตทำได้โดยการส่งข้อมูล สถานะ 1 ไปยังบิตที่ต้องการให้เป็นอินพุต ก็จะสามารถทำการรับค่าข้อมูลอินพุตได้

2.6.4.4 P3.0-P3.7 (ขาที่ 10-17) พอร์ต 3 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอก อินพุตและเอาต์พุต 2 ทิศทาง มีขนาด 8 บิต คุณสมบัติทั่วไปจะ เหมือนกับพอร์ตอื่นๆ แต่จะมีคุณสมบัติที่ต่างออกไป คือ ใช้ทำหน้าที่พิเศษเป็น สัญญาณควบคุมการทำงานต่างๆ ของไมโครคอนโทรลเลอร์ ดังตารางที่ 2.4

ตารางที่ 2.4 ตารางหน้าที่พิเศษของ พอร์ท 3

บิตของพอร์ต	สัญญาณ	หน้าที่การทำงาน
P3.0	RXD	รับข้อมูลจากพอร์ตอนุกรม (serial input port)
P3.1	TXD	ส่งข้อมูลจากพอร์ตอนุกรม (serial output port)
P3.2	$\overline{INT0}$	รับสัญญาณอินเตอร์รัปต์หมายเลข 0 (external interrupt 0)
P3.3	$\overline{INT1}$	รับสัญญาณอินเตอร์รัปต์หมายเลข 1 (external interrupt 1)
P3.4	T0	ใช้ตั้งเวลานับเวลาตัวที่ 0 (Timer 0 external input)
P3.5	T1	ใช้ตั้งเวลานับเวลาตัวที่ 1 (Timer 1 external input)
P3.6	$\overline{WR}$	เป็นสัญญาณเขียนข้อมูลหน่วยความจำหรืออุปกรณ์ภายนอก (external data memory write strobe)
P3.7	$\overline{RD}$	เป็นสัญญาณอ่านข้อมูลหน่วยความจำหรืออุปกรณ์ภายนอก (external data memory write strobe)

2.6.4.5 $\overline{PSEN}$ (Program Store Store Enable ขาที่ 29) ขาที่ทำงานที่สภาวะลอจิกเป็น “0” ไมโครคอนโทรลเลอร์ต้องอ่านค่าจากหน่วยความจำภายนอกที่เป็นข้อมูล โดย โปรแกรมจะเก็บในหน่วยความจำถาวร (ROM , EPROM , EEPROM) ส่วนมาก ใช้ต่อเป็นขาเลือกทำงาน (Enable:OE) แต่ถ้าไมโครคอนโทรลเลอร์ใช้ $\overline{\text{---}}$ หน่วยความจำภายใน ขานี้ก็จะไม่ได้ใช้งาน และมีค่าลอจิกเป็น “1”

2.6.4.6 ขา ALE (Address Latch Enable ขาที่ 30) ทำหน้าที่ควบคุมการทำงานของสัญญาณกำหนดตำแหน่งกับสัญญาณข้อมูล โดยใช้การเลือกเส้นทาง (data select หรือ multiplex) โดยปกติเมื่อไมโครคอนโทรลเลอร์ทำงานจะส่งสัญญาณกำหนด ตำแหน่งออกมาก่อน พร้อมกับส่ง

สัญญาณให้ขา ALE ทำงาน เพื่อเลือกให้ สัญญาณกำหนดตำแหน่ง (A0-A7) ผ่านไอซี (74LS373) ที่ทำหน้าที่เลือกเส้นทาง ถ้าส่งสัญญาณข้อมูลออกมา ไอซี (74LS373) จะไม่ทำงาน ข้อมูลก็จะถูกส่งไปที่ สายสัญญาณข้อมูล

2.6.4.7 ขา $\overline{EA}$ (External Access ขาที่ 31) ทำหน้าที่เลือกการทำงานของหน่วยความจำ ถ้ามีค่าลอจิกเป็น “1” หมายถึง ใช้ข้อมูลจากหน่วยความจำภายในตัว ไมโครคอนโทรลเลอร์ แต่ถ้ามีค่าลอจิกเป็น “0” หมายถึง ใช้ข้อมูลจาก หน่วยความจำภายนอก

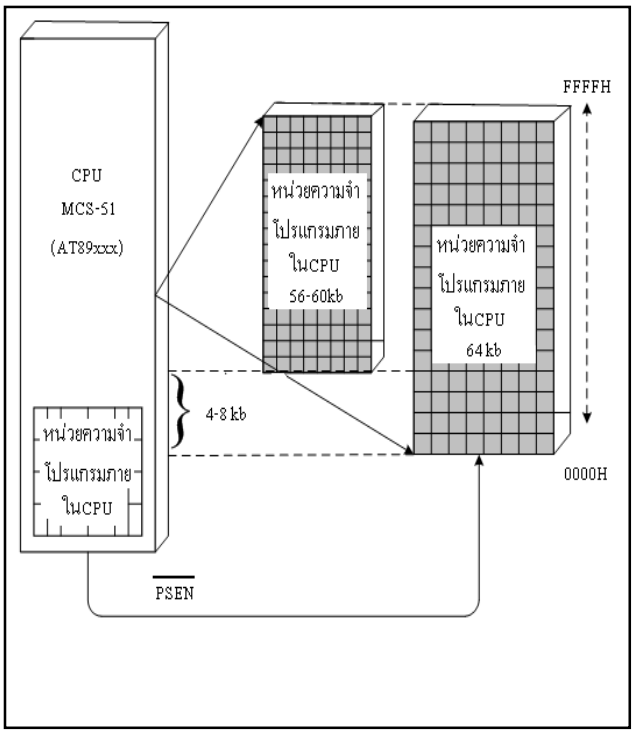
2.6.4.8 8.ขา RST (Reset ขาที่ 9) ทำหน้าที่เริ่มต้นการทำงานใหม่ของไมโครคอนโทรลเลอร์ การทำงานที่ค่าลอจิก “1” นี้จะทำให้ไมโครคอนโทรลเลอร์ เริ่มต้นทำงานที่ตำแหน่ง 0000 เพื่ออ่านข้อมูลโปรแกรมและจัดระบบการทำงาน

2.6.4.9 ขาสัญญาณนาฬิกา (ขาที่ 18-19) ซึ่งจะทำหน้าที่เป็นตัวกำหนดสัญญาณนาฬิกาให้กับไมโครคอนโทรลเลอร์ใช้เป็นฐานเวลาในการทำงาน โดยจะใช้แผ่นผลึก (crystal) ที่มีความถี่ตั้งแต่ 0-24 เมกกะเฮิรตซ์ (MHz) ร่วมกับตัวเก็บประจุขนาด 20-33 pF

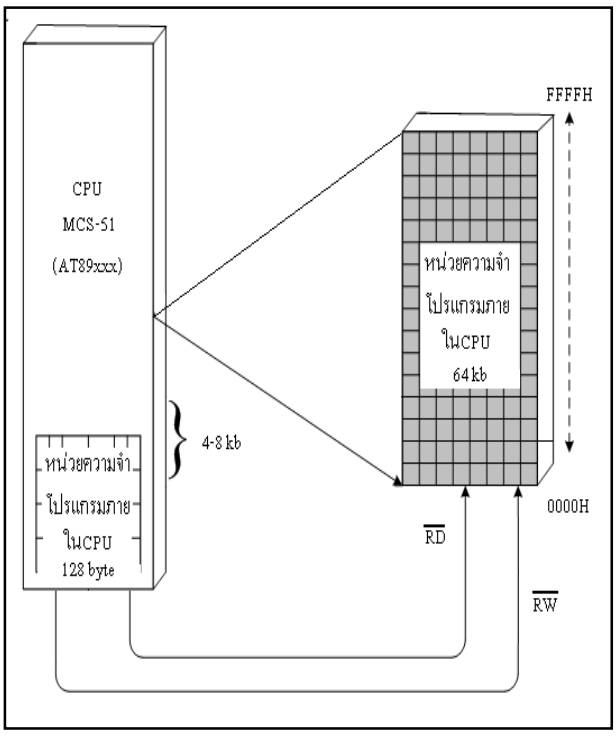
2.6.4.10 แหล่งจ่ายไฟ (Power supply) ขาที่ 20 จะเป็นขากราวด์ (Ground) และขาที่ 40 จะเป็นแหล่งจ่ายไฟบวกให้กับไมโครคอนโทรลเลอร์ ซึ่งใช้แหล่งจ่ายไฟขนาดไม่ เกิน 5 โวลต์

2.6.5 หน่วยความจำของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรเลอร์ MCS-51 ได้ออกแบบการจักการหน่วยความจำแต่ละประเภทแยกจากกันและกำหนดการทำงานเป็นแบบเฉพาะ คือ หน่วยความจำโปรแกรม “(program memory หรือ code memory)” และหน่วยความจำชั่วคราว (RAM) เรียกว่าหน่วยความจำข้อมูล (data memory) ซึ่งมีขนาดความจุ 64 กิโลไบต์เท่ากัน แต่จะถูกแยกการทำงานโดยคำสั่งทางซอฟแวร์และโครงสร้างทางฮาร์ดแวร์ ดังรูปที่ 2.28, 2.29



รูปที่ 2.24 การจัดการหน่วยความจำโปรแกรม (program memory)

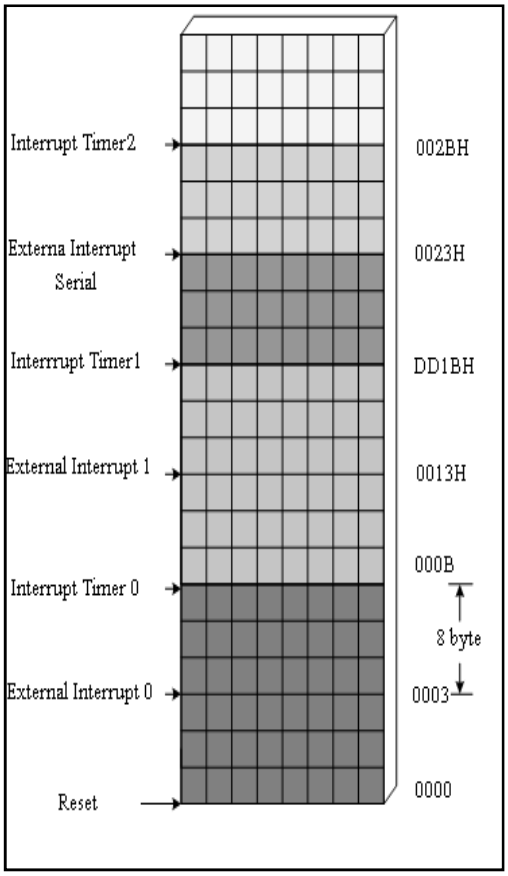


รูปที่ 2.25 การจัดการหน่วยความจำข้อมูล (data memory)

2.6.5.1 หน่วยความจำโปรแกรม (Program Memory)

ไมโครคอนโทรลเลอร์ MCS-51 ที่เริ่มต้นทำงานใหม่ หรือเมื่อรีเซตใหม่จะเริ่มทำงานที่ตำแหน่ง 0000 ทุกครั้ง โดยจะอ่านข้อมูลที่ตำแหน่ง 0000 แปลความหมายและปฏิบัติตามคำสั่ง ซึ่งเป็นตำแหน่งหน่วยความจำโปรแกรม (program memory) ที่ใช้เก็บประแกรมเบื้องต้นในการทำงานของไมโครคอนโทรลเลอร์ (monitor program) หรือระบบปฏิบัติการ (BIOS) ที่อาจอยู่ภายในหรือภายนอกไมโครคอนโทรลเลอร์ก็ได้

หลังจากโปรแกรมเริ่มทำงานจะมีการกำหนดตำแหน่งหน่วยความจำโปรแกรมเพื่อรับการอินเตอร์รัพท์ซึ่งมี 6 ประเภท แต่ละประเภทมีขนาด 8 ไบต์ ดังรูปที่ 2.30

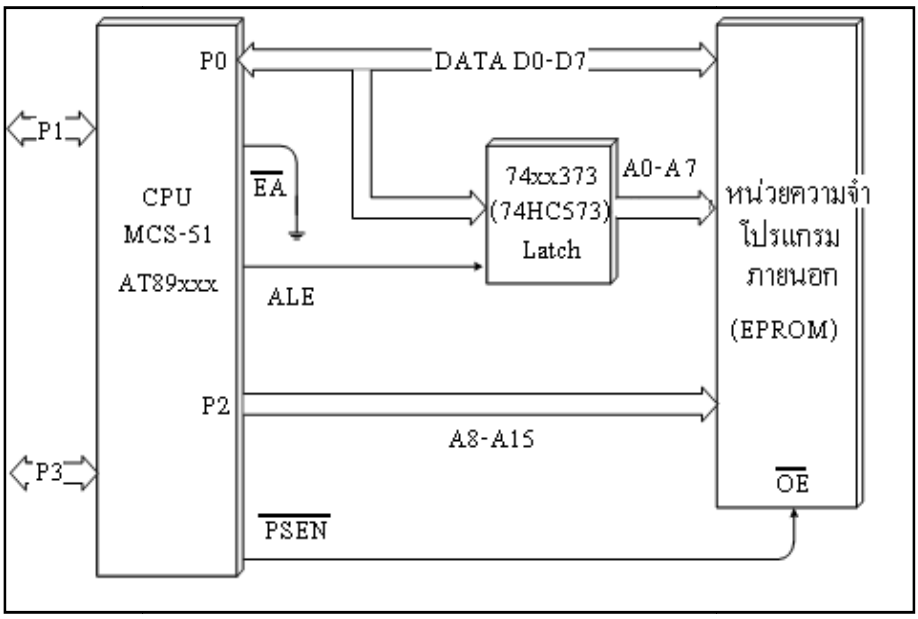


รูปที่ 2.26 หน่วยความจำโปรแกรมที่ตำแหน่งเริ่มต้นการทำงานและการอินเตอร์รัพท์

2.6.5.2 หน่วยความจำโปรแกรมภายนอก

งานควบคุมบางอย่างยังต้องใช้หน่วยความจำโปรแกรมเป็นจำนวนมาก ซึ่งหน่วยความจำโปรแกรมภายในอาจไม่เพียงพอ ก็สามารถใช้ไอซีหน่วยความจำมาต่อขยายได้ ซึ่งจะเริ่มจากตำแหน่งสุดท้ายของหน่วยความจำโปรแกรมภายใน เช่น ถ้ามีหน่วยความจำโปรแกรมภายในขนาด 4 กิโลไบต์ มีตำแหน่งหน่วยความจำข้อมูลอยู่ระหว่าง 0000-0FFFH การต่อหน่วยความจำภายนอกต้องเริ่มต้นที่ตำแหน่ง 1000H-FFFH

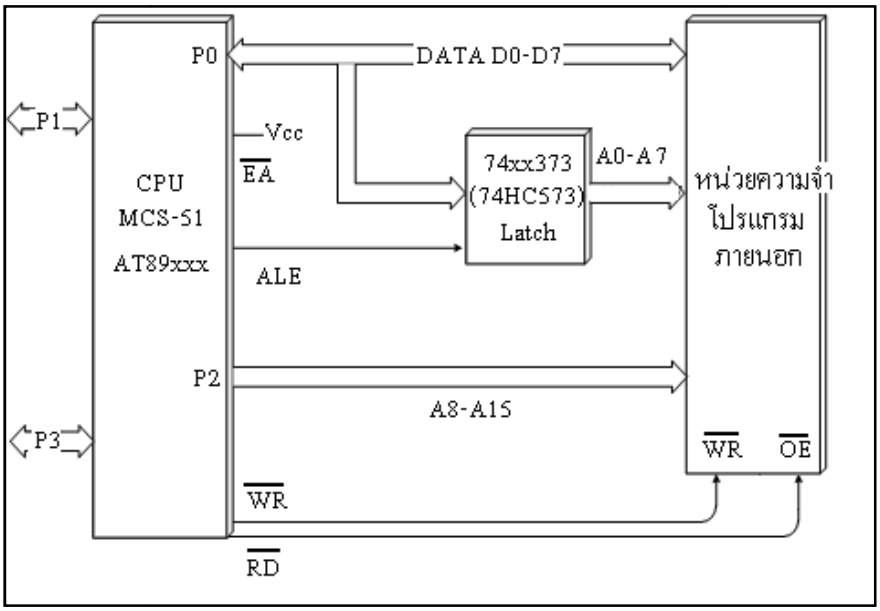
การที่จะต่อวงจรหน่วยความจำโปรแกรมภายนอกจะใช้พอร์ต 0 เป็นสายสัญญาณของข้อมูล (D0-D7) และสายสัญญาณกำหนดตำแหน่งหน่วยความจำ (A0-A7) โดยใช้เกตทีทีแอล 74xx373 (74HC573) ทำหน้าที่แยกสัญญาณการทำงานทั้งสองออกจากกันด้วยสัญญาณควบคุม ALE และใช้พอร์ต 2 ทำหน้าที่เป็นสัญญาณกำหนดตำแหน่งความจำ (A8-A15) ให้ครบ 16 บิต โดยใช้ขาควบคุม $\overline{\text{PSEN}}$ ทำหน้าที่ควบคุมการทำงานของไอซีหน่วยความจำ ดังรูป 2.31



รูปที่ 2.27 วงจรหน่วยความจำโปรแกรมภายนอก

2.6.5.3 หน่วยความจำข้อมูล (data memory)

การใช้งานจะมีอยู่ 2 แบบ คือ หน่วยความจำข้อมูลที่อยู่ภายในกับหน่วยความจำข้อมูลที่อยู่ภายนอกไมโครคอนโทรลเลอร์ สำหรับไมโครคอนโทรลเลอร์ตระกูล MCS-51 (AT89xxx) ขนาดของหน่วยความจำทั้งสองรวมกันจะไม่เกิด 64 กิโลไบต์ การติดต่อข้อมูลเพื่อทำการอ่าน/เขียนข้อมูลทำได้โดยใช้คำสั่ง MOVX เท่านั้น การต่อวงจรภายนอกจะใช้สัญญาณควบคุมการอ่านและการเขียนข้อมูล $\overline{RD}$ และ $\overline{WR}$ ดังรูปที่ 2.32

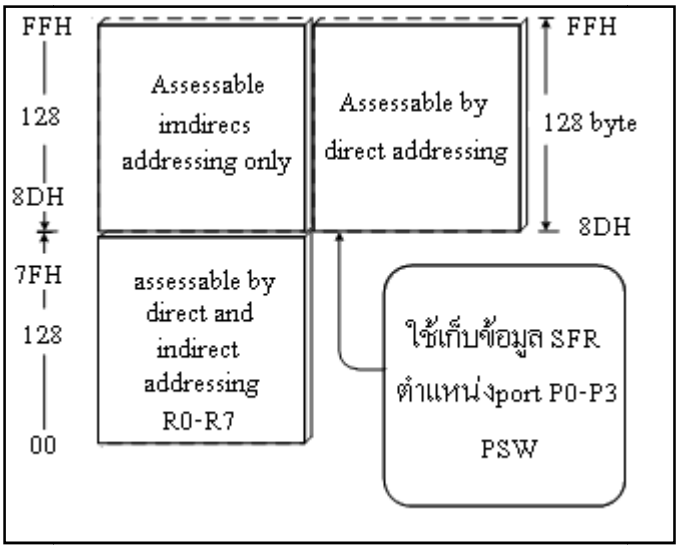


รูปที่ 2.28 วงจรหน่วยความจำข้อมูล

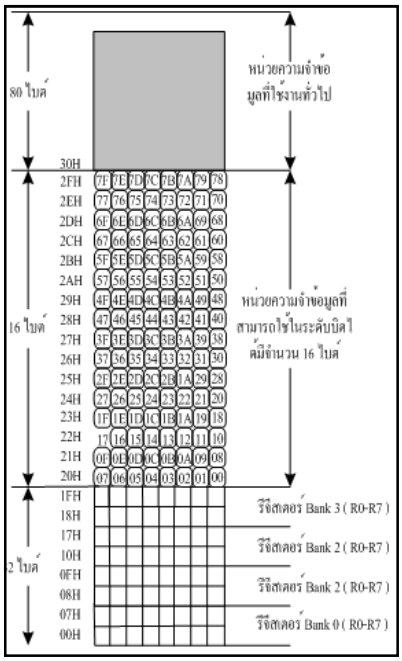
2.6.5.4 หน่วยความจำข้อมูลภายใน

โครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51(AT89xxx) จะมีหน่วยความจำข้อมูล (RAM) ขนาด 128 – 256 ไบต์ ขึ้นอยู่กับรุ่นของไมโครคอนโทรเลอร์ การใช้งานจะมีการแบ่งออกเป็นส่วนใหญ่ๆ คือหน่วยความจำตั้งแต่ตำแหน่ง 00-7FH (ส่วน = 128 ไบต์) และหน่วยความจำที่ตำแหน่ง 80H-FFH นอกจากนี้ยังได้มีการจัดระบบหน่วยความจำให้สามรถทำงานได้ 2 สภาวะ โดยใช้เป็นหน่วยความจำทั่วไปที่ใช้ได้โดย

ทางอ้อมแลใช้เป็นรีจีสเตอร์ที่ทำงานเฉพาะ(Special Function Register = SFR)อีกส่วนหนึ่ง ซึ่งดูเสมือนว่าหน่วยความจำภายในส่วนนี้จะมีขนาดทั้งหมด 384 กิโลไบต์ ดังรูปที่ 2.33



รูปที่ 2.29 การจัดพื้นที่งานของหน่วยความจำข้อมูลภายในไมโครคอนโทรลเลอร์



รูปที่ 2.30 พื้นที่ใช้งานของหน่วยความจำภายในที่ 128 ไบต์ส่วนล่าง

การจัดพื้นที่ใช้งานของหน่วยความจำข้อมูลที่อยู่ภายในถือว่าเป็นพื้นฐานสำคัญของการเรียนการเขียนโปรแกรมภาษาแอสเซมบลี ในตำแหน่ง 128 ไบต์ส่วนล่าง (00-7FH) ได้มีการจัดแบ่งหน้าที่แต่ละตำแหน่งออกเป็นส่วนย่อยๆ ซึ่งแต่ละส่วนได้กำหนดหน้าที่ให้ทำงานเฉพาะ เพื่อให้การเขียนโปรแกรมสามารถเรียกใช้ได้ง่ายขึ้น ดังรูปที่ 2.34

พื้นที่หน่วยความจำ 00-1FH มีขนาด 32 ไบต์ ได้กำหนดเป็นค่ารีจีสเตอร์ R0-R7 ขนาด8 บิต จำนวน 4 ชุด (Bank 0 – Bank 3) เป็นรีจีสเตอร์ที่ใช้งานทั่วไป การเรียกใช้งานแต่ละชุดทำได้โดยกำหนดบิตในรีจีสเตอร์ PSW (Program Status Word) การรีเซตหรือเริ่มต้นทำงานใหญ่ทุกครั้งจะกำหนดไว้ที่ตำแหน่ง รีจีสเตอร์ชุดที่ 1 (Bank 0)

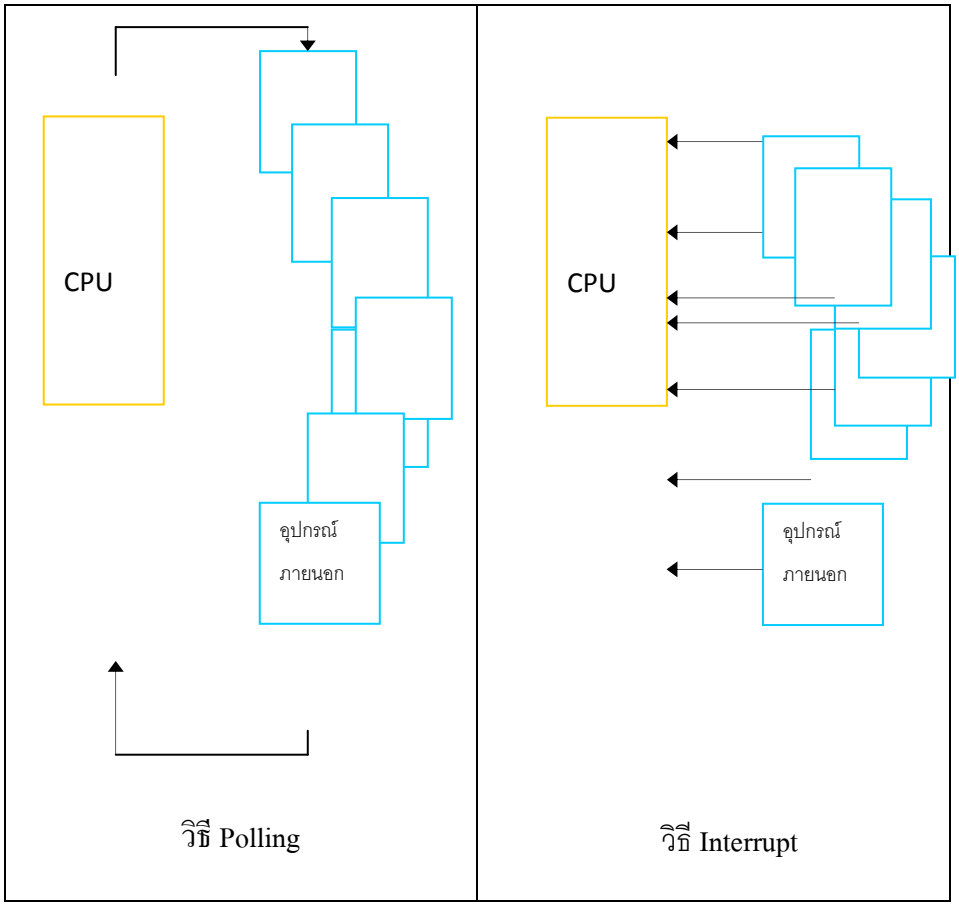
พื้นที่หน่วยความจำ 20H – 2FH มีขนาด 16 ไบต์ ออกแบบมาให้สามารถใช้ได้ถึงระดับบิต (bit addressable) ซึ่งเป็นหน่วยที่เล็กที่สุดของข้อมูล สามารถกระทำทางลอจิก (SET , CLEAR , AND และ OR) ได้ ทำให้ควบคุมการทำงานได้ละเอียดมากยิ่งขึ้นและในส่วนที่เหลืออีก 80 ไบต์นั้น (30H-7FH) ก็สามารถใช้งานได้ทั่วไป แต่ได้เป็นระดับไบต์ ซึ่งส่วนหนึ่งจะสำรองไว้เก็บข้อมูล สแตก (Stack Pointer = SP) ที่ใช้เก็บข้อมูลตำแหน่งที่กระโดดไปทำงานในส่วนของโปรแกรมย่อย

2.6.6 การควบคุมการอินเตอร์รัพท์

2.6.6.1 อินเตอร์รัพท์เบื้องต้น

จากที่เรารู้มาแล้วว่าองค์ประกอบของคอมพิวเตอร์ประกอบด้วย ซีพียู หรือหน่วยประมวลผลหน่วยความจำ และอุปกรณ์อินพุท เอาท์พุท วิธีที่ซีพียูใช้ติดต่อกับอุปกรณ์ภายนอก มีอยู่ 2 วิธีคือ การโพลลิ่ง (Polling) เป็นวิธีที่ซีพียูต้องคอยตรวจสอบอุปกรณ์อินพุทเอาท์พุทตลอดเวลาว่าอุปกรณ์ตัวไหนบ้างต้องการติดต่อ ทำให้เสียเวลามาก และอีกวิธีคือ การอินเตอร์รัพท์ (Interrupt) เป็นวิธีที่ซีพียูไม่ต้องคอยไปตรวจสอบอุปกรณ์อินพุทเอาท์พุท ถ้าอุปกรณ์ตัวไหนต้องการติดต่อ ก็เพียงแต่ส่งสัญญาณมาบอกซีพียู เรียกว่า สัญญาณร้องขออินเตอร์รัพท์ (Interrupt Request) เมื่อซีพียูได้รับสัญญาณ ก็ทำการตัดสินใจให้บริการอินเตอร์รัพท์ พร้อมกับส่งสัญญาณยอมให้มีการอินเตอร์รัพท์ (Interrupt Acknowledgement) กลับไปยังอุปกรณ์ เมื่ออุปกรณ์ได้รับก็ทำการติดต่อกับซีพียูต่อไป

ข้อแตกต่างระหว่างการโพลลิ่งและการอินเตอร์รัพท์ จึงขอยกตัวอย่างประกอบ วิธีโพลลิ่ง ซีพียูจะทำการตรวจสอบอุปกรณ์ที่ต้องการติดต่อ โดยเริ่มจากตัวแรกไปเรื่อยๆ จนกระทั่งตัวสุดท้าย ถ้าสมมุติว่าอุปกรณ์ตัวสุดท้ายต้องการติดต่อ ก็จะต้องรอจนกว่าซีพียูจะตรวจสอบมาถึง ทำให้อุปกรณ์ตัวนั้นเสียเวลาในการรอ



รูปที่ 2.31 แสดงการเปรียบเทียบระหว่างวิธี Polling กับวิธี Interrupt

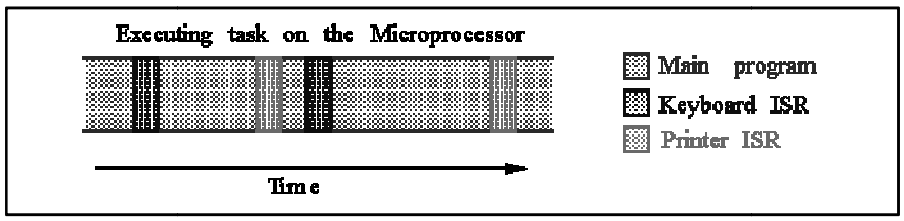
วิธีอินเตอร์รัพท์ ซีพียูไม่ต้องคอยไปตรวจสอบ ขณะที่ซีพียูกำลังทำงานอื่นอยู่ ถ้าอุปกรณ์ตัวไหนต้องการติดต่อก็จะส่งสัญญาณอินเตอร์รัพท์ไปบอกซีพียู ถ้าซีพียูต้องการติดต่อก็จะส่งสัญญาณตอบกลับไปยังอุปกรณ์ตัวนั้น ทำให้อุปกรณ์ไม่ต้องเสียเวลาในการรอจึงทำงานได้เร็ว และ

สมมุติว่าซีพียูต้องการติดต่อกับอุปกรณ์ตัวไหน เพียงแต่ส่งสัญญาณไปยังอุปกรณ์ตัวนั้น เช่น ต้องการอ่านข้อมูลจากแผ่นดิสก์ ก็จะส่งสัญญาณการอ่านไปยังดิสก์ไดร์ฟ เมื่อไดร์ฟได้รับสัญญาณแล้วก็จะทำการอ่านข้อมูลจากดิสก์เก็บไว้ใน Buffer ก่อน ช่วงระหว่างนี้ซีพียูสามารถไปทำงานอื่นได้ เมื่อดิสก์ไดร์ฟอ่านข้อมูลเสร็จ ก็จะส่งสัญญาณอินเตอร์รัพท์ไปบอกซีพียูให้มารับข้อมูลจาก Buffer ได้ จะเห็นว่าวิธีอินเตอร์รัพท์ จะช่วยให้ซีพียูทำงานสอดประสานกับกระบวนการอินพุทเอาท์พุทต่างๆได้ง่าย ถ้าปราศจากอินเตอร์รัพท์แล้ว หลายๆกระบวนการจะไม่สามารถทำงานได้หรือทำได้ยากมาก

2.6.6.2 อินเตอร์รัพท์

อินเตอร์รัพท์ คือเหตุการณ์ที่ไม่ปกติที่ต้องการให้ซีพียูหยุดการประมวลผลโปรแกรมที่ทำงานปัจจุบัน และไปทำงานบางอย่างที่เกี่ยวข้องกับเหตุการณ์ไม่ปกตินั้น หรือพูดง่ายๆ คือซีพียูกำลังทำงานบางอย่างอยู่ แต่อยู่ๆก็มีอุปกรณ์ภายนอกเข้ามาขอแทรกการทำงาน หรือขัดจังหวะการทำงาน ซีพียูก็ต้องพิจารณาว่าจะยอมให้อุปกรณ์นั้นขัดจังหวะหรือไม่ ถ้ายอมซีพียูก็ไปทำงานตามอุปกรณ์นั้น

ตัวอย่าง เครื่องคอมพิวเตอร์มีอุปกรณ์อินพุท คือ เครื่องพิมพ์ และอุปกรณ์เอาท์พุท คือ คีย์บอร์ด และอินเตอร์รัพท์เข้ามาช่วยทำงานได้อย่างไร พิจารณาการทำงานดังรูปที่ 2.36



รูปที่ 2.32 แสดงช่วงเวลาทำงานของซีพียูเมื่อติดต่อกับอุปกรณ์ภายนอก

จากรูปที่ 2.36 ซีพียูทำงานโปรแกรมหลักไปเรื่อยๆ และเมื่อมีคีย์บอร์ดอินเตอร์รัพท์เข้ามา (ผู้ใช้กดปุ่มคีย์บอร์ด) ซีพียูก็จะพักการทำงานโปรแกรมหลักไว้ก่อน และกระโดดไปยังโปรแกรมให้บริการอินเตอร์รัพท์คีย์บอร์ด (Keyboard ISR) เมื่อทำงานเสร็จซีพียูก็มาทำงาน

โปรแกรมหลักต่อ และเมื่อต้องการพิมพ์งานซีพียูก็จะส่งข้อมูลไปให้เครื่องพิมพ์ ขณะที่เครื่องพิมพ์กำลังพิมพ์งานอยู่ ซีพียูก็กลับมาทำงานโปรแกรมหลักต่อได้ เมื่อเครื่องพิมพ์ทำงานเสร็จก็ส่งสัญญาณอินเตอร์รัพท์มาบอกซีพียู และซีพียูก็หยุดทำงานโปรแกรมหลักและกระโดดไปยังโปรแกรมให้บริการอินเตอร์รัพท์เครื่องพิมพ์ (Printer ISR) เพื่อจัดการเกี่ยวกับเครื่องพิมพ์ เมื่อเสร็จก็กลับไปทำงานยังโปรแกรมหลักต่อไป

อินเตอร์รัพท์ถูกใช้ในหลายๆงาน ดังนี้

ซีพียูต้องรู้ว่าอุปกรณ์อินพุทเอาท์พุทตัวไหนพร้อมที่จะประมวลผล โดยจะต้องรู้สถานะของอุปกรณ์ต่างๆ ตลอดเวลา ซึ่งกลไกทางอินเตอร์รัพท์จะมาช่วยให้อุปกรณ์แจ้งซีพียูทราบว่ามันพร้อมที่จะส่งข้อมูลแล้ว ทำให้การทำงานของซีพียูมีประสิทธิภาพเมื่อเกิด Power fail จำเป็นต้องให้ซีพียูกระทำงานบางอย่างทันทีทันใด โดยกลไกอินเตอร์รัพท์จะเตรียมวิธีการที่จะสั่งให้ซีพียูเปลี่ยนจากการประมวลผลโปรแกรมปัจจุบัน ไปทำกิจกรรมนั้นทันทีเตรียมวิธีที่จะออกจากโปรแกรมที่ทำอยู่ปัจจุบัน กรณีเกิดข้อผิดพลาดทางด้านโปรแกรมขึ้น (software error) อัพเดตเวลาในแต่ละวัน ถ้าปราศจากอินเตอร์รัพท์แล้วซีพียูจะต้องใช้ program loop เพื่ออัพเดตเวลาปัจจุบัน ซึ่งจะทำให้ซีพียูไม่สามารถทำอย่างอื่นได้

2.6.6.3 ประเภทของอินเตอร์รัพท์

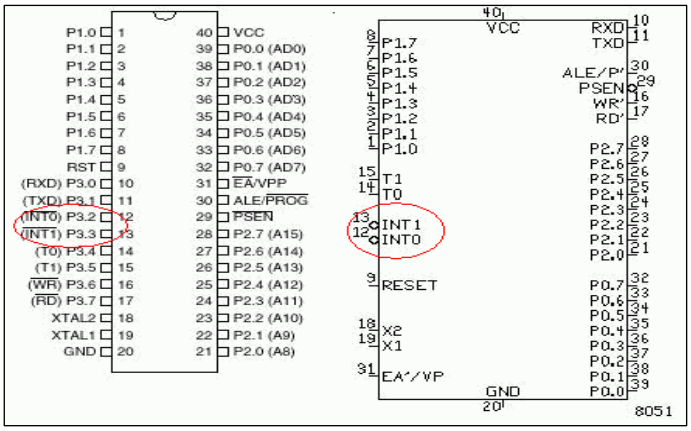
อินเตอร์รัพท์แบ่งออกเป็น 2 ประเภท คือ mask able interrupt และ non-mask able interrupt Mask able interrupt คืออินเตอร์รัพท์ที่ซีพียูสามารถที่จะยกเลิกได้ ส่วน Non-mask able interrupt คืออินเตอร์รัพท์ที่ซีพียูไม่สามารถยกเลิกได้ ซึ่งต้องตอบสนองทันทีทันใด เราสามารถกำหนดว่าจะให้ซีพียูทำงานอินเตอร์รัพท์แบบ Mask able interrupt หรือ non-mask able interrupt โดยการเซตบิต enable interrupt เมื่ออินเตอร์รัพท์ enable หรือบิตถูกเซต ซีพียูต้องให้บริการอินเตอร์รัพท์นั้น และถ้าอินเตอร์รัพท์ disable หรือบิตเคลียร์ ซีพียูจะ ก็จะยกเลิกอินเตอร์รัพท์นั้น

ในกรณีที่มีอินเตอร์รัพท์เข้าหลายๆ อินเตอร์รัพท์พร้อมกัน แต่ซีพียูจะให้บริการเพียงอินเตอร์รัพท์เดียวเท่านั้น การที่ซีพียูจะให้อินเตอร์รัพท์ไหนทำงานก่อน จำเป็นต้องมีการจัดลำดับ

ความสำคัญของอินเตอร์รัพท์ (Interrupt Priority) ตัวไหนจะมีลำดับความสำคัญสูงสุด ก็จะได้ก่อน ตัวอื่นก็จะได้ตามลำดับลงมา

2.6.6.4 อินเตอร์รัพท์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 มีการใช้งานอินเตอร์รัพท์อยู่ 2 ประเภท คือ อินเตอร์รัพท์จากภายนอก (External interrupt) จะเกิดเมื่อวงจรจากภายนอกส่งสัญญาณไปยังขาอินเตอร์รัพท์ของ MCS-51 ได้แก่ ขา INT0 และ INT1 และอินเตอร์รัพท์ภายใน (Internal interrupt) จะกำเนิดโดยฮาร์ดแวร์ที่อยู่ภายในชิป ได้แก่ไทเมอร์ 0 ไทเมอร์ 1 และพอร์ตสื่อสารอนุกรม เมื่ออินเตอร์รัพท์เกิดขึ้น ไมโครคอนโทรลเลอร์จะหยุดทำงานโปรแกรมปัจจุบัน และไปให้บริการอินเตอร์รัพท์ที่เข้ามา โดยการกระโดดไปยังตำแหน่งของโปรแกรมที่ให้บริการอินเตอร์รัพท์ (Interrupt Service Routine หรือ ISR) นั้นๆ ก่อนกระโดดไปจะต้องมีการเก็บค่าต่างๆที่จำเป็นที่ทำงานกับโปรแกรมปัจจุบันไว้ก่อน เช่นค่าในรีจิสเตอร์ PC เพราะเมื่อโปรแกรมให้บริการอินเตอร์รัพท์ทำงานเสร็จแล้ว จะได้กลับมาทำงานโปรแกรมเดิมต่อไปได้ จากนั้นก็กระโดดไปยังโปรแกรมให้บริการอินเตอร์รัพท์ เมื่อไมโครคอนโทรลเลอร์ทำงานเสร็จก็จะกระโดดกลับมาทำงานยังโปรแกรมปัจจุบันในตำแหน่งที่ต่อเนื่องจากเดิมก่อนที่จะเกิดอินเตอร์รัพท์ โดยการคืนค่าต่างๆ ที่บันทึกไว้กลับมา



รูปที่ 2.33 แสดงขา INT0 และ INT1

ไมโครคอนโทรลเลอร์ MCS-51 จะจองพื้นที่หน่วยความจำส่วนหนึ่งตั้งแต่แอดเดรสที่ 0003H ถึง 00FBH สำหรับเก็บแอดเดรสของโปรแกรมให้บริการอินเตอร์รัพท์ หรืออินเตอร์รัพท์เวกเตอร์ (Interrupt vector) ซึ่งแต่ละเวกเตอร์จะใช้พื้นที่ขนาด 8 ไบต์ ดังนั้นจึงมีจำนวนเวกเตอร์ทั้งหมด 32 เวกเตอร์ โดยแอดเดรส 0003H ถึง 0033H ถูกจองไว้สำหรับอินเตอร์รัพท์เวกเตอร์ของ MCS-51/52 ขณะที่ตั้งแต่แอดเดรสที่ 003BH ถึง 00FB เราสามารถกำหนดเวกเตอร์ของโปรแกรมให้บริการอินเตอร์รัพท์ที่เราสร้างขึ้นเองได้อย่างอิสระ เพื่อความเข้าใจง่ายจะใช้ตัวเลขแทนแอดเดรสของเวกเตอร์ เช่น อินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT0 (External INT0) จะอยู่ที่เวกเตอร์ 0 (0003H ถึง 000AH) หรือ อินเตอร์รัพท์ไทเมอร์ 0 อยู่ที่เวกเตอร์ 1 (000BH ถึง 0012H) เนื่องจากแต่ละเวกเตอร์มีขนาด 8 ไบต์ ดังนั้นเวลาคิดหมายเลขเวกเตอร์ จะเอาแอดเดรสเริ่มต้นมาลบด้วย 3 จากนั้นหารด้วย 8

ตารางที่ 2.5 แสดงหมายเลขอินเตอร์รัพท์ของ MCS-51/52

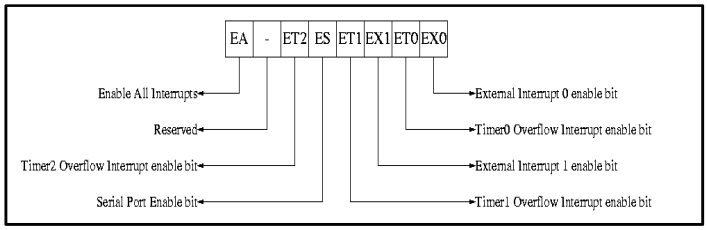
หมายเลขอินเตอร์รัพท์	แอดเดรส	หมายเลข อินเตอร์รัพท์	แอดเดรส
0 (อินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT0)	0003H	16	0083H
1 (อินเตอร์รัพท์ไทเมอร์ 0)	000BH	17	008BH
2 (อินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT1)	0013H	18	0093H
3 (อินเตอร์รัพท์ไทเมอร์ 1)	001BH	19	009BH
4 (อินเตอร์รัพท์พอร์ตอนุกรม)	0023H	20	00A3H
5 (อินเตอร์รัพท์ไทเมอร์ 2)	002BH	21	00ABH
6 (PCA)	0033H	22	00B3H

ตารางที่ 2.5 แสดงหมายเลขอินเตอร์รัพท์ของ MCS-51/52 (ต่อ)

หมายเลขอินเตอร์รัพท์	แอดเดรส	หมายเลข อินเตอร์รัพท์	แอดเดรส
7	003BH	23	00BBH
8	0043H	24	00C3H
9	004BH	25	00CBH
10	0053H	26	00D3H
11	005BH	27	00DBH
12	0063H	28	00E3H
13	006BH	29	00EBH
14	0073H	30	00F3H
15	007BH	31	00FBH

การใช้งานอินเตอร์รัพท์ของไมโครคอนโทรลเลอร์ MCS-51 จะเกี่ยวข้องกับรีจิสเตอร์ 3 ตัว คือ

รีจิสเตอร์ IE (Interrupt Enable) ไว้สำหรับควบคุมการ enable อินเตอร์รัพท์จากแหล่งต่างๆ ดังรูป



รูปที่ 2.34 แสดงรีจีสเตอร์ IE

EA ถ้ากำหนดเป็นลอจิก ‘1’ จะอินเตอร์รัพท์ทั้งหมด

ET2 ถ้ากำหนดเป็นลอจิก ‘1’ จะ enable ไทเมอร์ 2 (สำหรับ MCS-52)

ES ถ้ากำหนดเป็นลอจิก ‘1’ จะ enable พอร์ตอนุกรม

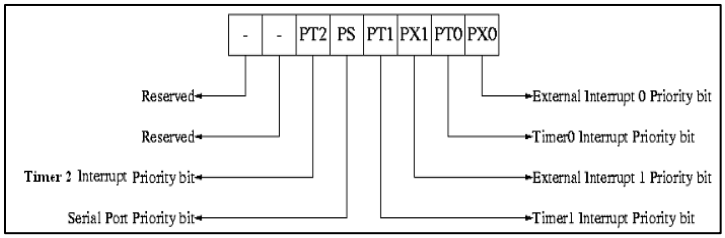
ET1 ถ้ากำหนดเป็นลอจิก ‘1’ จะ enable ไทเมอร์ 1

EX1 ถ้ากำหนดเป็นลอจิก ‘1’ จะ enable อินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT1

ET0 ถ้ากำหนดเป็นลอจิก ‘1’ จะ enable ไทเมอร์ 0

EX0 ถ้ากำหนดเป็นลอจิก ‘1’ จะ enable อินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT0

รีจิสเตอร์ IP (Interrupt Priority) สำหรับกำหนดลำดับความสำคัญของการอินเตอร์รัพท์ ในกรณีที่มีอินเตอร์รัพท์เกิดขึ้นจากหลายแหล่งพร้อมกัน อินเตอร์รัพท์ที่มีลำดับความสูงสุดจะได้รับบริการก่อน การกำหนดลำดับความสำคัญจะกำหนดโดยเซตบิตในรีจิสเตอร์ IP ดังรูป



รูปที่ 2.35 แสดงรีจีสเตอร์ IP

แต่ละบิตอธิบายดังนี้

PT2 บิตสำหรับกำหนดไทเมอร์ 2 เป็นลำดับสูงสุด

PS บิตสำหรับกำหนดพอร์ตอนุกรม เป็นลำดับสูงสุด

PT1 บิตสำหรับกำหนดไทเมอร์ 1 เป็นลำดับสูงสุด

PX1 บิตสำหรับกำหนดอินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT1 เป็นลำดับสูงสุด

PT0 บิตสำหรับกำหนดไทเมอร์ 0 เป็นลำดับสูงสุด

PX0 บิตสำหรับกำหนดอินเตอร์รัพท์ภายนอกที่เข้ามาทางขา INT0 เป็นลำดับสูงสุด

ถ้ามี 2 อินเตอร์รัพท์ที่มีระดับความสำคัญต่างกันเข้ามาพร้อมๆกัน ซีพียูจะให้บริการกับอินเตอร์รัพท์ที่มีลำดับความสำคัญสูงสุดก่อน แต่ถ้า 2 อินเตอร์รัพท์มีระดับความสำคัญเท่ากัน ก็จะใช้วิธีการโพลลิ่งภายใน (Internal Polling) นั่นคือจะมีการกำหนดโครงสร้างลำดับความสำคัญของอินเตอร์รัพท์อีกชั้นหนึ่ง เพื่อให้ซีพียูไว้ตรวจสอบว่าอินเตอร์รัพท์ไหนลำดับสูงกว่ากัน

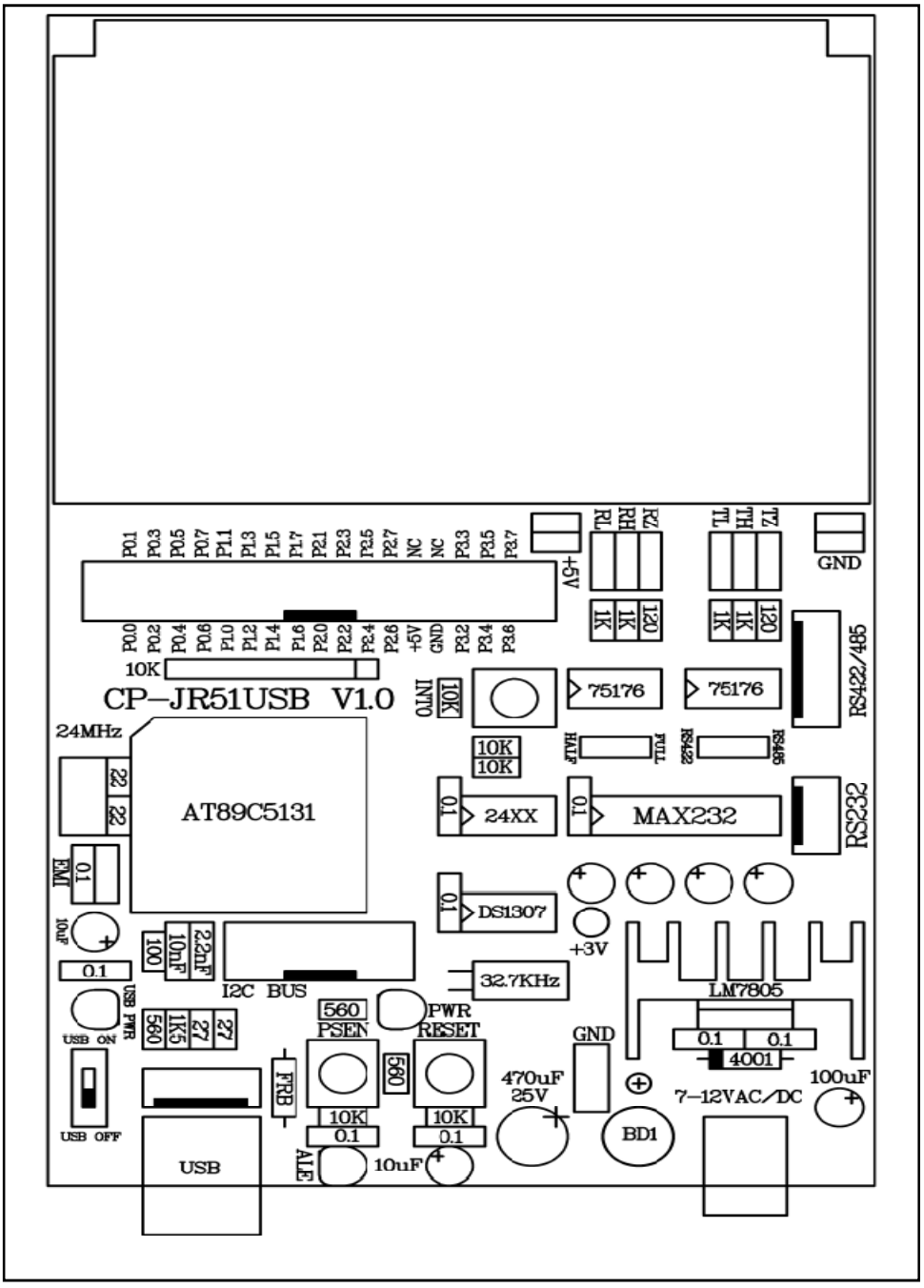
2.7 บอร์ด CP-JR51USB v1.0

2.7.1 ลักษณะทั่วไปของบอร์ด CP-JR51USB v1.0

บอร์ดไมโครคอนโทรลเลอร์รุ่น CP-JR51USB v1.0 ภายในบอร์ดใช้ชิปเบอร์ AT89C5131 ของบริษัท Atmel ซึ่งชิปเบอร์นี้มีคุณสมบัติเด่นในเรื่องของการเชื่อมต่อระบบบัส USB และ สามารถเลือกโหมดการทำงานเป็นแบบ 6clocks /1 Machine cycle หรือ 12 clocks /1 Machine cycle ได้ ซึ่ง จะไม่มีปัญหากับการสื่อสารอนุกรมเนื่องจากภายในชิป AT89C5131 นั้นได้มีส่วนของ Internal Baud Rate Generator อยู่ภายใน ซึ่งจะทำให้ผู้ใช้สามารถสร้างค่า Baud rate ใดๆ ได้ตามต้องการ

คุณสมบัติเด่นในส่วนของการเชื่อมต่อ USB, AT89C5131 ภายในได้บรรจุโมดูล Full-speed USBที่ใช้งานได้กับเวอร์ชั่น 1.1 และ 2.0 ซึ่งผู้อ่านสามารถส่ง หรือ รับข้อมูลผ่านทางบัส USB ด้วยอัตรา 12MHzได้ นอกจากนั้นชิป AT89C5131 ยังมีหน่วยความจำ Flash ถึง 32 Kbytes , Internal RAM 256 bytes ,ระดับความสำคัญของสัญญาณ Interrupt 4 – Level , 16-bit Timer/Counters มี 3 ตัว , Full duplex enhanced UART (EUART) , ERAM = 1024 bytes , Dual data point , แหล่งกำเนิดกระแสที่สามารถเลือกปริมาณกระแสเพื่อใช้ขับ LED 4 แหล่ง ฯลฯ

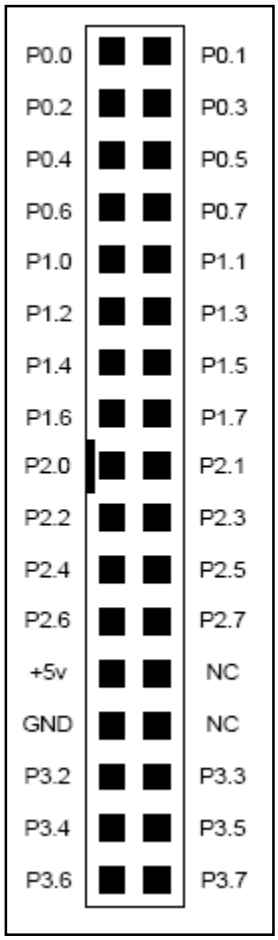
นอกจากนั้น AT89C5131 ยังมีโหมดเกี่ยวกับการประหยัดพลังงานอีก 2 โหมด คือ 1.) Idle mode และ 2.) Power-down mode ซึ่งจะทำให้การใช้พลังงานของ CPU ในขณะนั้นน้อยลง



รูปที่ 2.36 แสดงลักษณะโครงสร้างบอร์ด CP-JR51USB v1.

2.7.2 พอร์ต 34 PIN

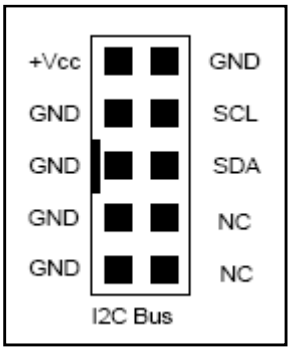
พอร์ตสื่อสาร 34 Pin เป็นพอร์ตที่รวมพอร์ต P0, P1, P2, P3 (บางขา) ของ CPU เข้าไว้ด้วยกัน โดยการต่อใช้งานผู้อ่านควรใช้สายแพร 34 Pin ในการต่อไปใช้งาน ในกรณีที่ผู้อ่านซื้อบอร์ด I/O ของทางบริษัท ETT ผู้อ่านสามารถต่อสายแพร 34 ได้โดยตรง หรือ ถ้าผู้อ่านออกแบบ Hardware เอง, ผู้อ่านควรจัดให้ขาให้อยู่ในรูปแบบของพอร์ต 34 Pin เพื่อจะได้สะดวกในการใช้งาน



รูปที่ 2.37 แสดงตำแหน่งขาคอนเน็คเตอร์ I/O พอร์ต 34 Pin

2.7.3 พอร์ต I2C BUS

การสื่อสารในแบบ Two Wire Interface (TWI), ที่ตำแหน่งขา P4.0 และ P4.1 ได้ต่อออกมายังคอนเน็คเตอร์ขนาด 10 Pin ซึ่งขั้วต่อนี้จะใช้เพิ่มเติมจำนวนอุปกรณ์ที่ติดต่อสื่อสารแบบ I2C ให้กับบอร์ดทดลอง



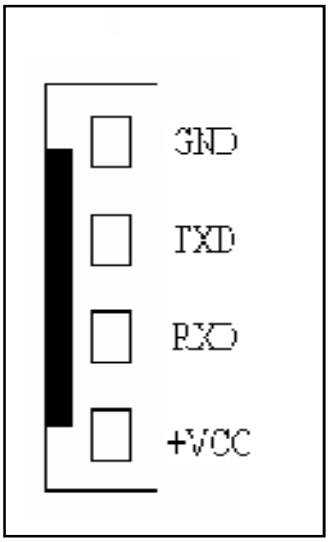
รูปที่ 2.38 แสดงตำแหน่งขาคอนเน็คเตอร์ I2C BUS, 10 Pin

2.7.4 พอร์ต RS232

การสื่อสารอนุกรม RS23ภายในได้บรรจุไอซี Line Driver ไว้เรียบร้อยแล้ว ซึ่งสามารถต่อใช้งานได้ทันที โดย ไอซี Line Driver มีหน้าที่ในการเปลี่ยนระดับแรงดัน TTL 5V. ให้เป็นแรงดันในแบบRS232 ที่ 12 V. ผลจะทำให้ผู้อ่านสามารถต่อสายสัญญาณได้ถึง 15 เมตรเลย ทีเดียวในแบบ ตัวต่อตัว (Point-to-Point) เท่านั้นสำหรับสายสัญญาณที่จะนำมาใช้สำหรับทำการสื่อสารแบบ RS232 นั้น จะใช้สัญญาณเพียง 2-3เส้น เท่านั้น ทั้งนี้ขึ้นอยู่กับความต้องการในการสื่อสารว่าต้องการสื่อสารแบบทิศทางเดียวหรือสองทิศทาง

การสื่อสาร RS232 แบบสองทิศทาง ซึ่งจะมีทั้งการรับข้อมูลและส่งข้อมูลไปมาระหว่างด้านรับและด้านส่ง โดยในกรณีนี้จะต้องใช้สายสัญญาณจำนวน 3 เส้น สัญญาณรับข้อมูล (RXD) สัญญาณส่งข้อมูล(TXD) และสัญญาณอ้างอิง (GND) โดยในการเชื่อมต่อสายนั้นจะต้องทำการสลับสัญญาณกับอุปกรณ์ปลายทางด้วย คือ สัญญาณส่ง (TXD) จากบอร์ด CPJR51USBv1.0 จะต้องต่อเข้ากับสัญญาณรับ (RXD) ของอุปกรณ์อื่น และ สัญญาณส่ง(TXD) จากอุปกรณ์อื่น ก็ต้องต่อกับสัญญาณรับ (RXD) ของบอร์ด ส่วนสัญญาณอ้างอิง(GND) จะต้องต่อตรงถึงกัน จึงจะสามารถทำการ รับ-ส่ง ข้อมูลกันได้

การสื่อสาร RS232 แบบทิศทางเดียว ซึ่งอาจเป็นการรอรับข้อมูลจากด้านส่งเพียงอย่างเดียว หรือ อาจเป็นการส่งข้อมูลออกไปยังปลายทางเพียงอย่างเดียว โดยไม่มีการโต้ตอบข้อมูลซึ่งกันและกัน ซึ่งวิธีนี้จะใช้สายสัญญาณเพียง 2 เส้น เท่านั้น โดยถ้าเป็นทางด้านส่ง ก็จะต่อเพียงสัญญาณส่ง (TXD) และสัญญาณอ้างอิง (GND) แต่ถ้าเป็นทางด้านรับ ก็จะต่อเพียงสัญญาณรับ (RXD) และ สัญญาณอ้างอิง (GND) เท่านั้นโดยขั้วต่อของสัญญาณ RS232 ของบอร์ด CP-JR51USB v1.0 นั้น จะเป็นจุดเชื่อมต่อของสัญญาณ รับ-ส่ง ข้อมูล ที่เปลี่ยนระดับสัญญาณเป็นแบบ RS232 แล้ว ซึ่งจะมีลักษณะเป็นแบบขั้ว CPAขนาด 4 PIN สำหรับใช้เป็นจุดเชื่อมต่อสัญญาณ รับ-ส่ง ข้อมูลกับอุปกรณ์ภายนอก โดยมีลักษณะการจัดเรียงสัญญาณดังนี้



รูปที่ 2.39 แสดงขั้วต่อสัญญาณ RS232 ของบอร์ด CP-JR51USB v1.0

ซึ่งจะเห็นได้ว่าขั้วต่อสัญญาณ RS232 ของบอร์ดนั้น จะมีทั้งหมด 4 เส้น แต่ในการ รับ-ส่งข้อมูลแบบปรกตินั้น จะใช้สัญญาณเพียงแค่ 3 เส้น คือ RXD,TXD และ GND เท่านั้น ส่วน +VCC ซึ่งเป็นไฟเลี้ยงวงจร +5V นั้น จะไม่จำเป็นต้องนำมาใช้ในการสื่อสารกันแต่อย่างใด โดย +VCC หรือ +5V นี้ จะออกแบบเผื่อไว้ในกรณีที่อุปกรณ์ปลายทางเป็นวงจรขนาดเล็กและไม่สะดวกที่จะหาแหล่งจ่ายไฟให้กับอุปกรณ์ปลายทางด้วย ก็อาจต่อไฟเลี้ยงวงจร +VCC นี้ออกไปให้กับอุปกรณ์ปลายทางด้วยก็ได้เช่นกัน

2.8 ไทเมอร์และเคาน์เตอร์ ใน MCS-51

2.8.1 การใช้งานไทเมอร์ในไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 มีไทเมอร์/เคาน์เตอร์ขนาด 16 บิต 2 ตัว คือ ไทเมอร์ 0 และไทเมอร์ 1 ส่วนในไมโครคอนโทรลเลอร์อนุกรม 8052 ซึ่งออกมาทีหลังจะมีไทเมอร์/เคาน์เตอร์ 3 ตัว นั้นคือมีไทเมอร์ 2 เพิ่มขึ้นมา ไทเมอร์/เคาน์เตอร์แต่ละตัวสามารถเลือกใช้งานเป็นไทเมอร์ หรือเคาน์เตอร์ก็ได้ และทำงานได้อย่างเป็นอิสระต่อกัน ในการทำงานเป็นไทเมอร์นั้นจะใช้หลักการเพิ่มค่ารีจิสเตอร์ไทเมอร์ ทุก ๆ Machine Cycle ซึ่งมีช่วงเท่ากับ 12 คาบสัญญาณนาฬิกาที่ถูกสร้างขึ้นจากคริสตอลที่ต่อใช้งานให้กับไมโครคอนโทรลเลอร์นั้นเอง ในการใช้งานไทเมอร์ 0 และ 1 ของ MCS-51 นั้น ต้องรู้จัก รีจิสเตอร์ที่สำคัญที่ใช้ควบคุมการทำงานของไทเมอร์เสียก่อน ซึ่งมีรายละเอียดดังนี้

รีจิสเตอร์ไทเมอร์: เป็นรีจิสเตอร์ขนาด 8 บิต มี 4 ตัวด้วยกันคือ TL0 อยู่ที่ตำแหน่ง 8AH, TH0 อยู่ที่ตำแหน่ง 8CH, TL1 อยู่ที่ตำแหน่ง 8BH และ TH1 อยู่ที่ตำแหน่ง 8DH โดยปกติจะทำงานเป็นคู่กัน นั้นคือไทเมอร์ 0 ใช้รีจิสเตอร์ TL0 และ TH0 ส่วนไทเมอร์ 1 ใช้รีจิสเตอร์ TL1 และ TH1 ซึ่งทำให้กลายเป็นรีจิสเตอร์ขนาด 16 บิต ในการทำงานเป็นไทเมอร์ เมื่อไทเมอร์ถูกเปิดค่าในรีจิสเตอร์ไทเมอร์จะเพิ่มขึ้นทุก ๆ Machine Cycle จนกว่าจะเกิดการโอเวอร์โฟลว์

รีจิสเตอร์ TCON: เป็นรีจิสเตอร์ขนาด 8 บิต อยู่ที่ตำแหน่ง 88H สามารถเข้าถึงได้ในระดับบิต มีโครงสร้างดังแสดงในรูปที่ 2.41

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

รูปที่ 2.40 โครงสร้างรีจิสเตอร์ TCON

TF1 (Timer 1 overflow Flag): เป็นบิตที่ใช้ในแสดงการเกิดโอเวอร์โฟลวของไทเมอร์ 1 โดยบิต TF1 จะถูกเซตเป็น “1” อัตโนมัติ เมื่อเกิดโอเวอร์โฟลวจากไทเมอร์ 1 และจะถูกเคลียร์โดยกระบวนการทางฮาร์ดแวร์ เมื่อเกิดการอินเตอร์รัพท์จากไทเมอร์ 1 ขึ้น

TR1 (Timer 1 run Control bit): ใช้ควบคุม เปิด ปิด การทำงานของไทเมอร์ 1 สามารถเซตหรือเคลียร์ได้โดยซอฟท์แวร์ โดยถ้าต้องการให้ไทเมอร์ 1 ทำงานต้องเซตบิตนี้ให้เป็น “1”

TF0 (Timer 0 overflow Flag): เป็นบิตที่ใช้ในแสดงการเกิดโอเวอร์โฟลวของไทเมอร์ 0 โดยบิต TF0 นี้จะถูกเซตเป็น “1” อัตโนมัติ เมื่อเกิดโอเวอร์โฟลวจากไทเมอร์ 0 และจะถูกเคลียร์โดยกระบวนการทางฮาร์ดแวร์ เมื่อเกิดการอินเตอร์รัพท์จากไทเมอร์ 0 ขึ้น

TR0 (Timer 0 run Control bit): ใช้ควบคุม เปิด ปิด การทำงานของไทเมอร์ 0 สามารถเซตหรือเคลียร์ได้โดยซอฟท์แวร์ โดยถ้าต้องการให้ไทเมอร์ 0 ทำงานต้องเซตบิตนี้ให้เป็น “1”

รีจิสเตอร์TMOD (Timer/Counter Mode Control Register): เป็นรีจิสเตอร์ขนาด 8 บิต อยู่ที่ตำแหน่ง 89H ไม่สามารถเข้าถึงได้ในระดับบิตได้ แบ่งออกเป็น 2 ส่วนคือ ส่วน 4

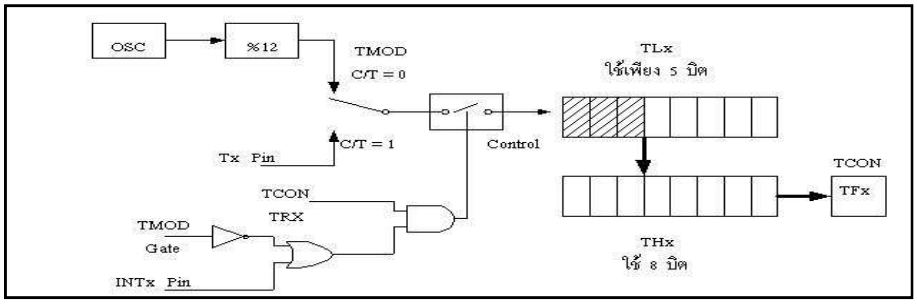
บิตล่างใช้กำหนดโหมดการทำงานของไทเมอร์ 0 และ 4 บิตบนใช้กำหนดการทำงานของไทเมอร์ 1 มีโครงสร้างดังแสดงในรูปที่ 2.42

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Gate	C/T	M1	M0	Gate	C/T	M1	M0
Timer 1				Timer 0			

รูปที่ 2.41 โครงสร้างรีจิสเตอร์ TMOD

GATE: ใช้เลือกการควบคุมการทำงานของไทเมอร์ โดยถ้าเป็น “0” จะเป็นการควบคุมทางซอฟท์แวร์ นั้นคือไทเมอร์จะทำงานเมื่อบิต TR ในรีจิสเตอร์ TCON เป็น “1” แต่ในกรณีที่บิต GATE เป็น “1” จะเป็นการควบคุมทางฮาร์ดแวร์ นั้นคือไทเมอร์จะทำงานเมื่อบิต TR ในรีจิสเตอร์ TCON เป็น “1” และ ขาอินพุต INT ของไมโครคอนโทลเลอร์มีสถานะลอจิกเป็นเป็น “1” ด้วย C/T (Timer or Counter selector) ใช้เลือกการทำงานของไทเมอร์/เคาน์เตอร์ โดยถ้าเป็น “0” จะทำงานเป็นไทเมอร์ แต่ถ้าเป็น“1”จะทำงานเป็นเคาน์เตอร์ M1, M0 (Mode selector bit) ใช้เลือกโหมดการทำงานของไทเมอร์/เคาน์เตอร์โดยถ้า เป็น“00” ทำงานในโหมด 0, “01” ทำงานในโหมด 1, “10” ทำงานในโหมด 2, “11” ทำงานในโหมด 3

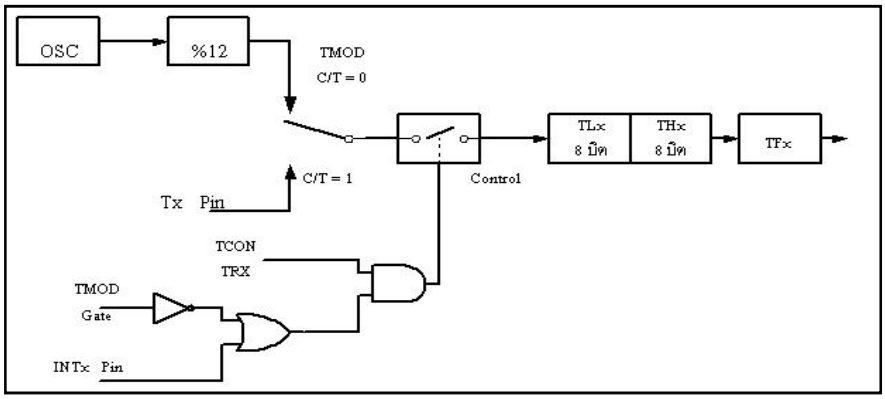
การทำงานในโหมด 0 เป็นการทำงานเป็นไทเมอร์/เคาน์เตอร์ 13 บิต นั้นคือใช้งานรีจิสเตอร์ TL 5 บิต และ TH 8 บิต ดังนั้นเมื่อค่าในรีจิสเตอร์ TL และ TH เพิ่มขึ้นจนเป็น “1” หมดทั้ง 13 บิต บิต TF ภายในรีจิสเตอร์ TCON ก็จะถูกเซต เพื่อแสดงการเกิดโอเวอร์โฟลว์



รูปที่ 2.42 แสดงการทำงาน ไทเมอร์/เคาน์เตอร์ ในโหมด 0

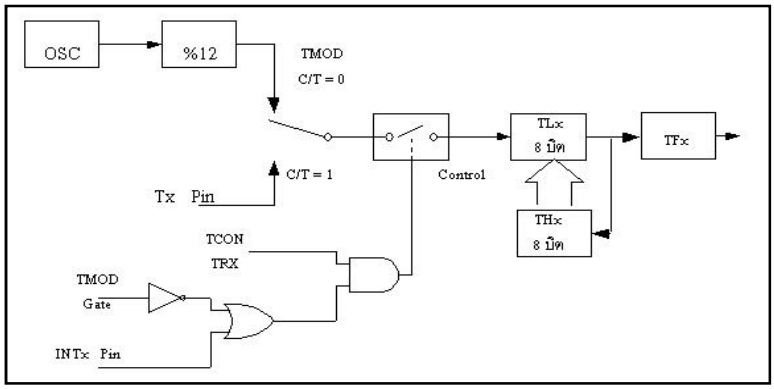
ใช้นับ 13 บิต โดย TLx 5 บิต รวมกับ THx อีก 8 บิตซึ่งอยู่ในช่วง 0000H ถึง 1FFFH (หรือ 0 - 8192) นั้นคือถ้านับเกิน 8192 ค่าจะเกิดค่าล้น (Overflow)ที่ TFx ในรีจิสเตอร์ TCON บอกให้รู้ว่านับครบแล้ว ในการนับจะเริ่มนับที่รีจิสเตอร์ TLX ในส่วน 5 บิตก่อนให้ครบ 31 ครั้ง (25 หรือ XXX00000B ถึง XXX11111B) ต่อจากนั้นจะเพิ่มค่าที่ THX ขึ้น 1 ค่า (ดังนั้นอัตราส่วนระหว่าง THXต่อTLXจึงเป็น1:32)

การควบคุมการตั้งเวลาโดยใช้ซอฟแวร์จะต้องกำหนดบิตที่ GATE, C/T ในรีจิสเตอร์ TMOD และ TRx ในรีจิสเตอร์ TCON ในส่วนของฮาร์ดแวร์การกำหนดสถานะลอจิกทางขา INTx จะเป็นการเปิด ปิด Control ที่จะรับอินพุตจาก 2 ทาง คือ ความถี่ภายในคอนโทรเลอร์ หาร ด้วย 12 ส่วนอีกทางมาจากขา Tx เมื่อเตรียมทุกบิตเรียบร้อยแล้ว ก็เริ่มให้คอนโทรเลอร์นับ หรือหยุดนับที่บิต TRx (ทางซอฟแวร์)



รูปที่ 2.43 แสดงการทำงาน ไทเมอร์/เคาน์เตอร์ ในโหมด 1

การทำงานในโหมด 1 เป็นการทำงานเป็นไทเมอร์/เคาน์เตอร์ 16 บิต นั้นคือใช้งานรีจิสเตอร์ TL และ TH ครบทั้ง 8 บิต ดังนั้นเมื่อค่าในรีจิสเตอร์ TL และ TH เพิ่มขึ้นจนเป็น “1” หมดทั้ง 16 บิต บิต TF ภายในรีจิสเตอร์TCONก็จะถูกเซตเพื่อแสดงการเกิดโอเวอร์โฟลว

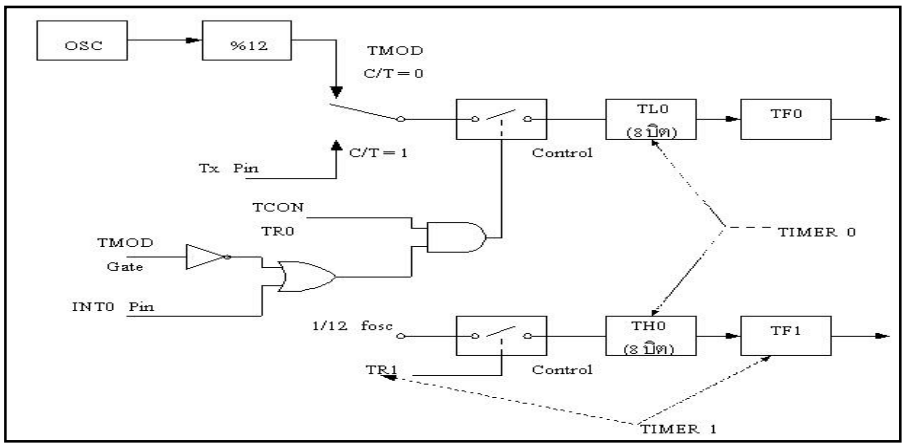


รูปที่ 2.44 แสดงการทำงาน ไทเมอร์/เคาน์เตอร์ ในโหมด 2

การทำงานในโหมด 2 เป็นการทำงานเป็นไทเมอร์/เคาน์เตอร์ 8 บิต แบบตั้งค่าอัตโนมัติ (Auto Reload) โดยรีจิสเตอร์ TL ทำงานเป็นตัวนับ ส่วนรีจิสเตอร์ TH ทำงานเป็นตัวเก็บค่าเริ่มต้นของการนับ เมื่อเริ่มต้นการทำงานค่ารีจิสเตอร์ TL จะถูกเซตให้เหมือนกับค่าในรีจิสเตอร์ TH และเริ่มการนับแบบ 8 บิต เมื่อนับจนเกิดโอเวอร์โฟลว (ค่าในรีจิสเตอร์ TL เป็น “1” หมดทุกบิต) จะทำให้บิต TF ภายในรีจิสเตอร์ TCON จะถูกเซต ส่วนรีจิสเตอร์ TL ก็จะถูกเซตให้เหมือนกับค่าในรีจิสเตอร์ TH และเริ่มต้นการนับใหม่อีกครั้งรีจิสเตอร์ TH และเริ่มต้นการนับใหม่อีกครั้ง

การทำงานในโหมด 3 จะแบ่ง Timer 0 ออกจาก Timer 1 การใช้งาน Timer 0 จะแยก TH0 กับ TL0 ให้มีตัวนับสองตัวทำงานอิสระต่อกันและทำงานแบบ 8 บิตเท่านั้น ที่รีจิสเตอร์ TL0 สามารถเลือกการทำงานทั้งโหมดตัวนับ (Counter) และตัวตั้งเวลา (Timer)ได้ตามปกติ จากรูป TL0 จะใช้บิตการทำงานของ Timer 0 โดยการกำหนดบิต C/T , TR0 , Gate , INT0 และ TF0 เป็นตัวควบคุมการนับเกิน

ส่วน Timer 1 จะไม่มีในโหมด 3 แต่จะใช้ TF1 และบิต TR1 เป็นตัวควบคุมการทำงานร่วมกับ TH0 (Timer 0) แทน และจะใช้เป็นตัวตั้งเวลาเท่านั้น (นับค่าแมชชีนไซเกิลจากความถีคอนโทรเลอร์)โดยการกำหนด TR1 ให้เป็น “1” ข้อมูลใน TH0 จะเริ่มนับถึง FFH แล้วกลับเป็นค่า 00H ใหม่จะทำให้ TF1 เกิด Overflow ขึ้น (เป็น “1”) นั้นก็คือการร้องขออินเตอร์รัพท์



รูปที่ 2.45 แสดงการทำงาน ไทเมอร์/เคาน์เตอร์ ในโหมด 3

2.9 โปรแกรม ภาษา Visual Basic

โปรแกรมภาษา Visual Basic เป็นโปรแกรมภาษา ที่ใช้พัฒนา Application ในเครื่อง Pc โปรแกรม Visual Basic เป็นโปรแกรมที่พัฒนามาจาก ภาษา Basic ซึ่งเป็นโปรแกรมภาษาที่ใช้พัฒนา Application ใน Dos และได้รับการพัฒนา จนกลายเป็น Visual Basic ที่ใช้งานร่วมใน Windows

โปรแกรม Visual Basic เป็นโปรแกรมที่ที่สนับสนุนการพัฒนา Application แบบ Component คือ เป็นการนำเอาวัตถุ (Object) ที่ได้รับการพัฒนาขึ้น โดยโปรแกรม Visual Basic เองมาประกอบเข้าด้วยกันและเขียนคำสั่งควบคุมการทำงาน ได้ตามความต้องการของผู้พัฒนา Application นั้น ลักษณะการทำงานดังกล่าวทำให้สามารถพัฒนา Application ได้สะดวกรวดเร็ว

2.10 โปรแกรม Keil C51

โปรแกรม Keil C51 เป็นโปรแกรมคอมไพเลอร์ (complier) ภาษาซีพัฒนาโดยบริษัท Keil Software ตัวโปรแกรมนี้มีชื่อว่า uVision 3 ซึ่งเป็นคอมฟ์เอร์ที่ช่วยในการพัฒนาโปรแกรมภาษาซี เป็น HEX ไฟล์ เพื่อใช้ในการให้โปรแกรมลงหน่วยความจำของไมโครคอลโทรเลอร์ ให้ทำงานตามที่ต้องการ

บทที่ 3
หลักการออกแบบ

3.1 ส่วนของฮาร์ดแวร์

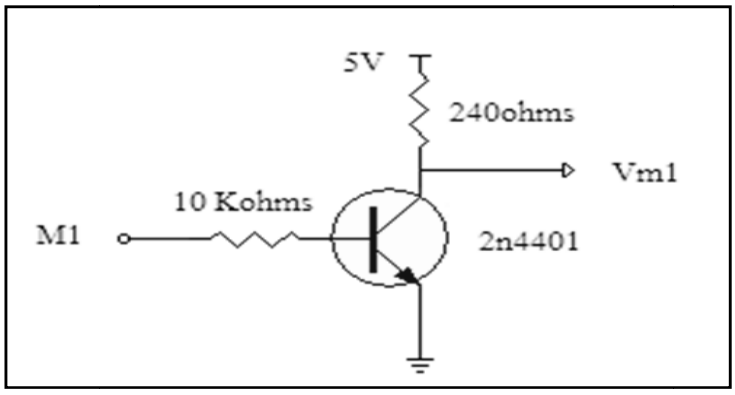
3.1.1 Encoder

ใช้ ZX-21 ติดไว้กับล้อหลัง-ซ้าย สัญญาณที่ได้จะเป็น 0สลับกับ 1 (pulse) โดยจะนำเข้าไปนับกับ Counter ของ MCU โดยใช้ ขา T0 และใช้ Timer0 เป็นตัวนับ

จากการทดลองพบว่า ล้อหมุน 1 รอบจะได้ระยะ 46 cm ดังนั้น pulse 1 ลูกจะได้ 5.1 cm (pulse 1 ลูก ออกมาเมื่อล้อหมุนไป 40 องศา)

3.1.2 การควบคุมมอเตอร์

ตัวรถจะใช้รีเลย์ 4 ตัวเพื่อควบคุมมอเตอร์ให้เคลื่อนที่ มีการเคลื่อนที่ทั้งหมด 9 วิธี (จากการควบคุมที่รีโมท) ได้แก่ หยุดนิ่ง, ขึ้นทั้งคู่, ขึ้นซ้าย, ขึ้นขวา, ลงซ้าย, ลงขวา, ซ้ายขึ้น-ขวาลง, ซ้ายลง-ขวาขึ้น จากการทดลงได้ดักสัญญาณที่รีเลย์ จะพบว่ามีสัญญาณไฟบวกประมาณ 7V ออกมาที่รีเลย์ และ มีลักษณะต่างกันไปจึงได้ออกแบบวงจรทรานซิสเตอร์เพื่อสร้างสัญญาณดิจิตอลขึ้นมา ดังรูปที่ 3.1



รูปที่ 3.1 แสดงวงจรทรานซิสเตอร์ที่ใช้ในการสร้างสัญญาณดิจิตอล

รูปที่ 3.1 นำไปทดลองวัดค่าดิจิตอล จะได้ดังนี้ (เรียงจาก M1-M4) (ให้ดูในวงจรของรถจะเขียนบอก)

ซ้าย-ขึ้น	1110
ซ้าย-ลง	1101
ขวา-ขึ้น	0111
ขวา-ลง	1011
ขึ้นทั้งคู่	0110
ลงทั้งคู่	1001
ซ้ายลง-ขวาขึ้น	0101
ซ้ายขึ้น-ขวาลง	1010
หยุดนิ่ง	1111

ข้อมูลนี้จะนำไปตรวจว่ารถกำลังเดินหน้า หรือถอยหลังต่อไป

การต่อ M1 – M4 เข้า MCU จะต่อดังนี้

P2.0 = M1

P2.1 = M2

P2.2 = M3

P2.4 = M4

3.1.3 ระบบวัดระยะสิ่งกีดขวาง

ระบบตรวจจับสิ่งกีดขวางในโครงงานนี้ ได้ทำการพัฒนาระบบการตรวจจับสิ่งกีดขวางโดยการนำเอาอัลตร้าโซนิคทรานดิวเซอร์มาเป็นตัวตรวจจับสิ่งกีดขวางดังที่ได้กล่าวมาแล้วในข้างต้น

3.1.3.1 การทดสอบ SRF - 04

ในการทดลองนี้เป็นการทดสอบเพื่อแสดงการกำหนดระยะของ SRF04 ว่าสามารถตรวจจับวัตถุในระยะต่างๆได้ดีเพียงใดโดยจะใช้ SRF-04 โดยส่งสัญญาณ เข้า MCU ดังนี้

P0.0 = Trigger Input

P0.1 = Pulse Output

จากการทดลอง จะต้องติดตั้ง SRF-04 ที่ระยะสูงจากพื้น ประมาณ 20 cm จึงจะวัดได้ถูกต้อง และวัดระยะของวัตถุได้แม่นยำในระยะ 1.5 m หากเกินนี้จะเกิด Error มาก

3.1.4 โมดูลเข็มทิศดิจิตอล CMPS 03 Digital Compass Module

ส่วนของเข็มทิศ ใช้ CMPS03 โดยส่งสัญญาณ เข้า MCU ดังนี้

P1.0 = SCL

P1.1 = SDA

3.2 ส่วนของซอฟท์แวร์

3.2.1 ภาค MCU

3.2.1.1 การวัดระยะที่รถเคลื่อนที่

ใช้ Timer0 เป็น Counter โดยนับสัญญาณที่เปลี่ยนจาก 1 เป็น 0 ที่ขา T0 ก่อนการนับจะต้องเคลียร์ค่า TL0 และ TH0 ให้เป็น 0x00 ด้วย เมื่อนับได้ก็นำไปคูณกับ 5.11 จะเป็นระยะทางที่ได้

3.2.1.2 การควบคุม SRF-04

ใช้ทฤษฎีบทเรื่อง อัลตร้าโซนิคทรานสดิวเซอร์ SRF04 ที่ได้กล่าวมาแล้วข้างต้นโดยส่งพัลส์ในช่วงเวลาที่กำหนด และรอคอยเวลาที่เสียงจะสะท้อนกลับมา และนำความกว้างของสัญญาณที่ได้รับมาคำนวณหาระยะของวัตถุ ปัญหาของส่วนนี้คือความแม่นยำ และเวลาที่ใช้ในการทำงาน (ค่อนข้างนาน เนื่องจากต้องรอเสียง) โดยจะคำนวณ 2 ครั้ง และหาค่าเฉลี่ย

3.2.1.3 การควบคุมโมดูลเข็มทิศดิจิตอล CMPS 03 Digital Compass Module

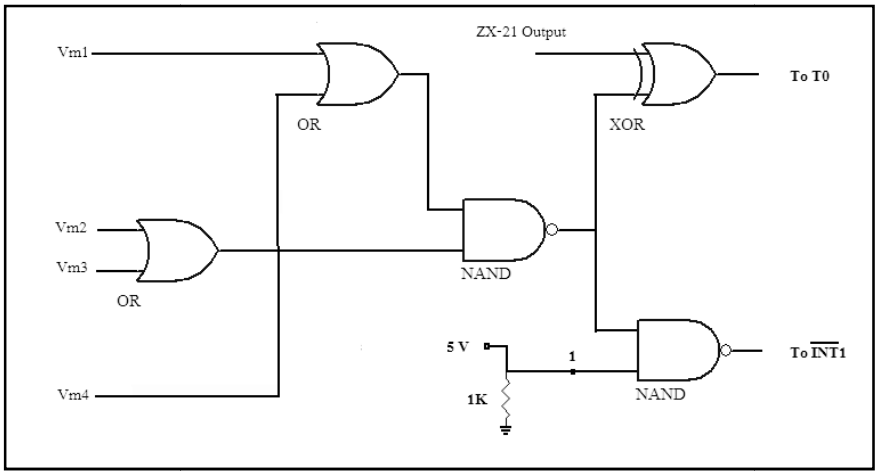
เขียนติดต่อแบบ I^2C โดยต้องส่งสัญญาณไปให้กับตัวโมดูล และรอรับข้อมูลกลับ รูปแบบการสื่อสารข้อมูลบัส I^2C บัส I^2C เชื่อมต่อกับไมโครคอนโทรลเลอร์โดยใช้สายสัญญาณ 2 เส้น ได้แก่ขา SDA (รับและส่งข้อมูล) และ SCL (ขาสัญญาณนาฬิกา) โดยขาสัญญาณทั้งสองจะต้องต่อตัวต้านทานพูลอัปต่อไว้เพื่อกำหนดสถานะลอจิก “1” ให้กับระบบบัส

ลำดับขั้นการติดต่อ ค่าแอดเดรสของโมดูล CMPS03 คือ $C0 สำหรับการส่งข้อมูล และ $C1 สำหรับการอ่านค่าข้อมูล โดยขั้นตอนการติดต่อกับโมดูล CMPS03 เพื่ออ่านข้อมูลดังนี้

ส่งบิตเริ่มต้นหรือ Start bit เพื่อแจ้งให้ระบบบัส I^2C เตรียมพร้อมรับข้อมูล ส่งค่าแอดเดรส $C0 เพื่อระบุว่าต้องการติดต่อเพื่อเขียนข้อมูลไปยังกับโมดูล CMPS03. ส่งค่าตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03 ที่ต้องการอ่านค่า ซึ่งมีรายละเอียดแสดง ในตารางที่ 1 ในทฤษฎีบทเรื่องโมดูลเข็มทิศที่ได้กล่าวมาแล้วข้างต้นส่งค่าแอดเดรส $C1 เพื่อระบุว่าต้องการอ่านค่าข้อมูลจากโมดูล CMP03 อ่านค่าข้อมูลจากโมดูล CMPS03 มาเก็บไว้ในหน่วยความจำ ส่งบิตหยุดหรือ stop เพื่อหยุดการสื่อสารข้อมูลและกำหนดให้บัสอยู่ในสภาวะบัสว่าง

3.2.1.4 การรับสัญญาณอินเตอร์รัพท์

เมื่อมีการ อินเตอร์รัพท์ จากวงจร จะเก็บข้อมูลของมอเตอร์ในขณะนั้น ว่ามีการเดินหน้า หรือถอยหลัง ซึ่งจะทำงานทันที ไม่ต้องรอคิว



รูปที่ 3.2 แสดงวงจรสร้างสัญญาณอินเตอร์รัพท์

3.2.1.5 ส่วนของการติดต่อกับคอมพิวเตอร์

ใช้การสื่อสารแบบ RS-232 โดยส่งที่อัตราบอด 9,600 บิตต่อวินาที (9600, n, 8, 1) โดยใช้ Timer2 ในการสร้าง baud rate ซึ่งได้กล่าวไปแล้วในข้างต้น

3.2.1.6 การส่งข้อมูล จะมีลำดับของข้อมูลดังนี้

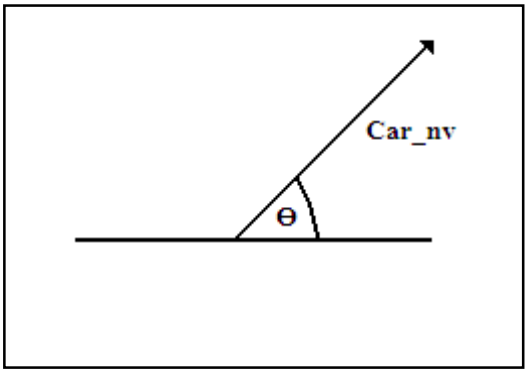
ตารางที่ 3.1 แสดงลำดับการส่งข้อมูล

Header	ระยะสิ่งกีดขวาง	องศาของรถ	ข้อมูลมอเตอร์	ระยะทางที่รถเคลื่อนที่

หมายเหตุ ในที่นี่ใช้ Header เป็น 0x7E ซึ่งอาจเปลี่ยนแปลงได้

3.2.2 ภาค VB (Visual Basic)

จะใช้ คอมโพเนนท์ชื่อ MSComm ที่ต้อง Add เพิ่มเข้ามาที่ Tool Bar ของ VB แนวคิดของการสร้างแผนที่คือการสร้าง เวกเตอร์ 1 หน่วย ซึ่งเป็นทิศทางของรถ (ตัวแปร car_nv) โดยรับค่ามาจาก cpms03 เมื่อได้เวกเตอร์ 1 หน่วยแล้ว จะนำไปวาดรูปสามเหลี่ยม โดยใช้หลักการของตรีโกณมิติ วาดเส้นออกมาตามทิศทางของเวกเตอร์นั้น

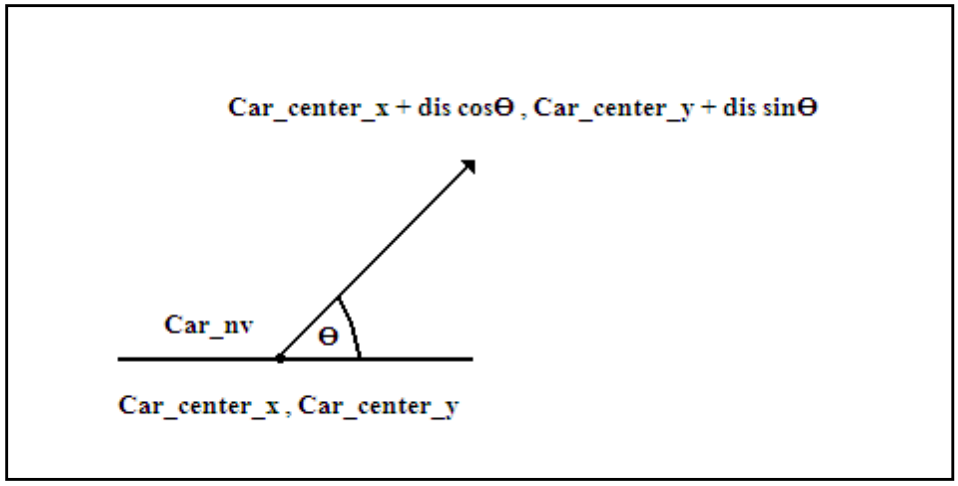


รูปที่ 3.3 แสดง เวกเตอร์ 1 หน่วย

เวกเตอร์ 1 หน่วยนี้ จะเป็นตัวที่ทำให้เกิด Real Time ของการแสดงผล โดยรับค่าใหม่ๆมาจาก MCU แล้วนำไปวาดรูปบน VB

ส่วนการวาดสิ่งกีดขวางนั้น จะใช้อาร์เรย์จำนวน 50 อันเก็บตำแหน่งของสิ่งกีดขวาง (ตัวแปร obs_x และ obs_y) และเมื่อจะวาดรูปจะใช้ตัว Shape ใน VB สำหรับวาด โดยใช้อาร์เรย์ของ Shape เช่นกัน ซึ่งค่า x ใน VB จะใช้ property ชื่อ left ค่า y จะใช้ top (x วัดจากขอบจอภาพด้านซ้าย, top จะวัดจากขอบภาพด้านบนลงมา) หน่วยเป็นพิกเซล

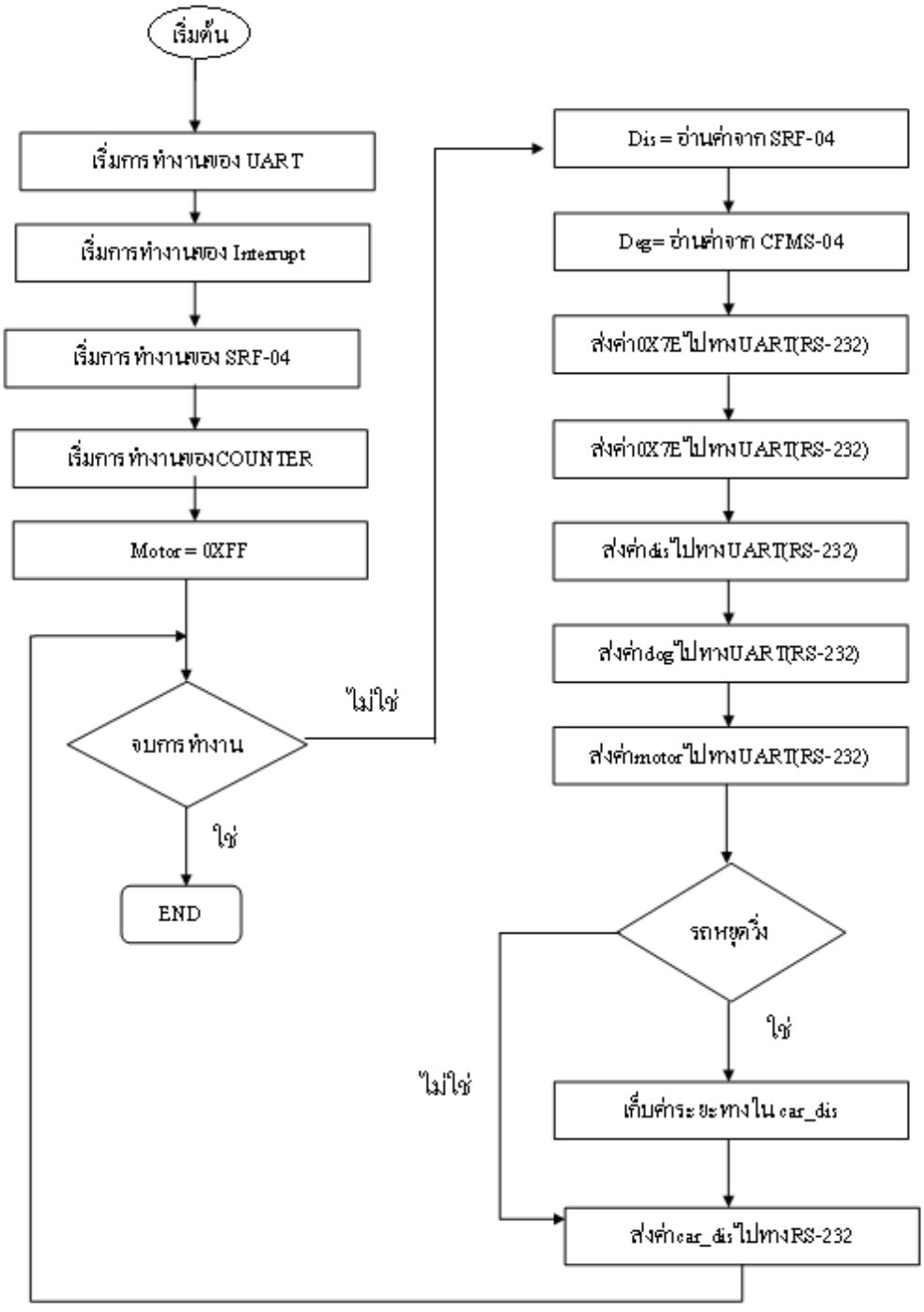
การคำนวณตำแหน่งของสิ่งกีดขวางจะใช้หลักการของตรีโกณมิติเช่นกัน



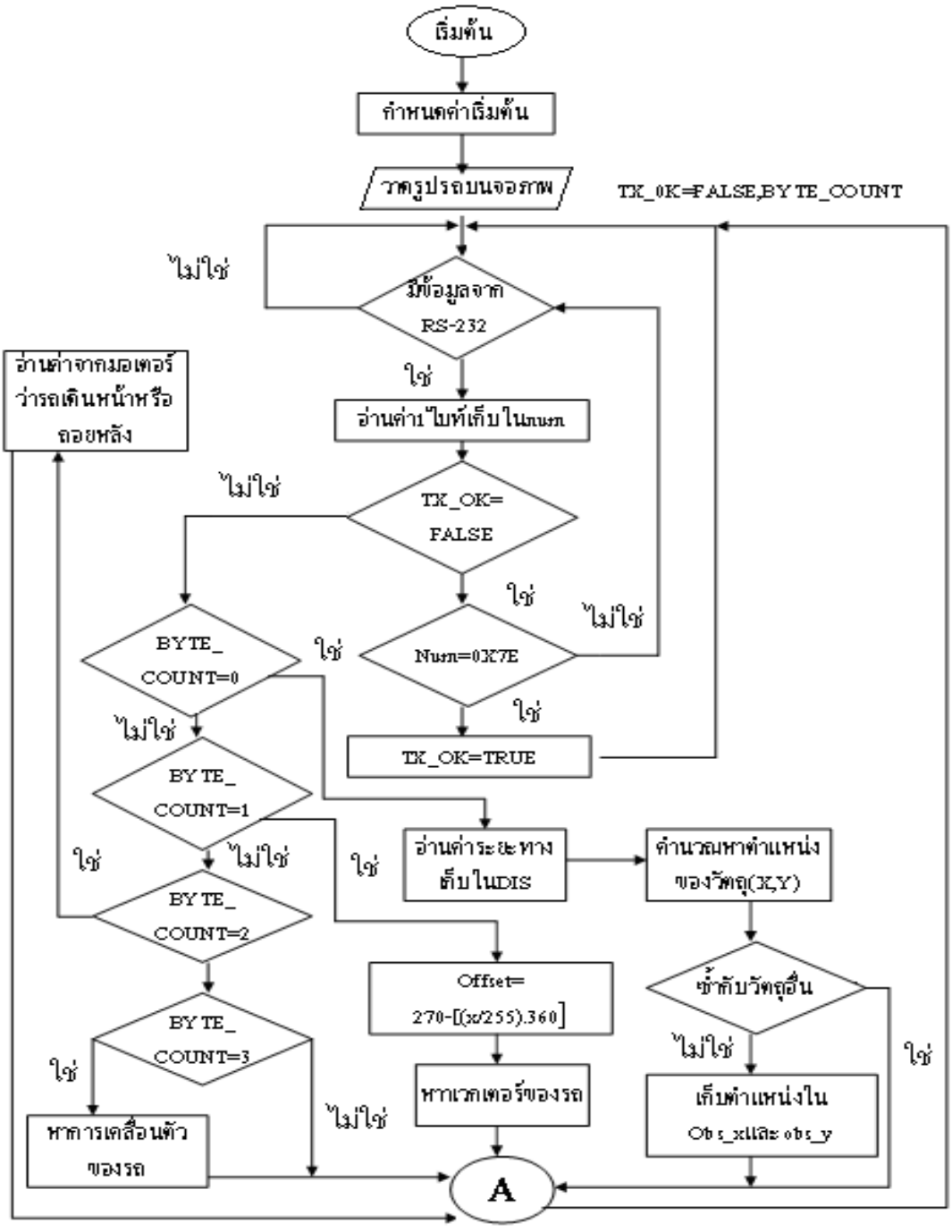
รูปที่ 3.4 การคำนวณจุดของสิ่งกีดขวาง

การเคลื่อนที่ของรถ จะใช้การเปลี่ยนตำแหน่งของ Car_center_x และ car_center_y ซึ่งเป็นจุด Sensor (SRF-04) ของรถ ได้แก่จุดที่จะวาดรูปสามเหลี่ยมนั่นเอง

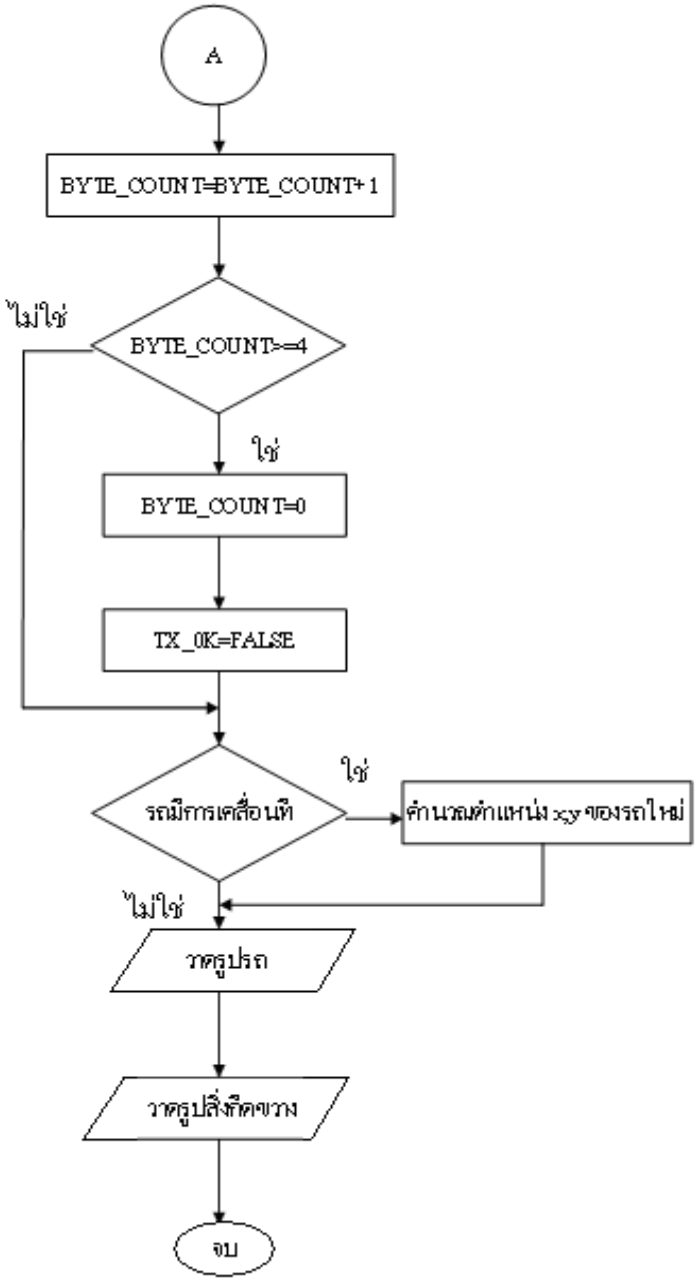
การเลี้ยวรถ ใช้หลักการเปลี่ยนองศาของ เวกเตอร์ 1 หน่วย โดยเปลี่ยนตามข้อมูลที่ได้จาก CMPS03



รูปที่ 3.5 FLOWCHARTการทำงานทั้งหมดของโปรแกรมบนบอร์ด



รูปที่ 3.6 FLOWCHART การทำงานทั้งหมดของโปรแกรม VB (Visual Basic)



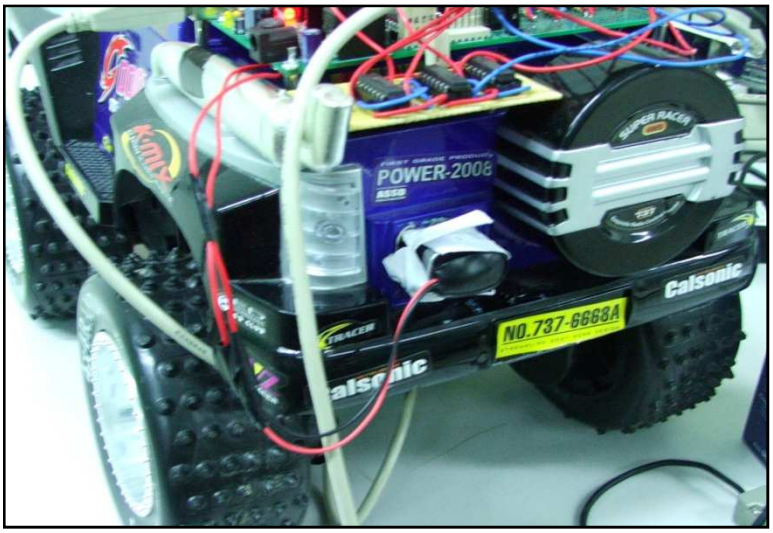
รูปที่ 3.6 FLOWCAHART การทำงานทั้งหมดของโปรแกรม VB (Visual Basic) (ต่อ)

บทที่ 4

ผลการทดลอง

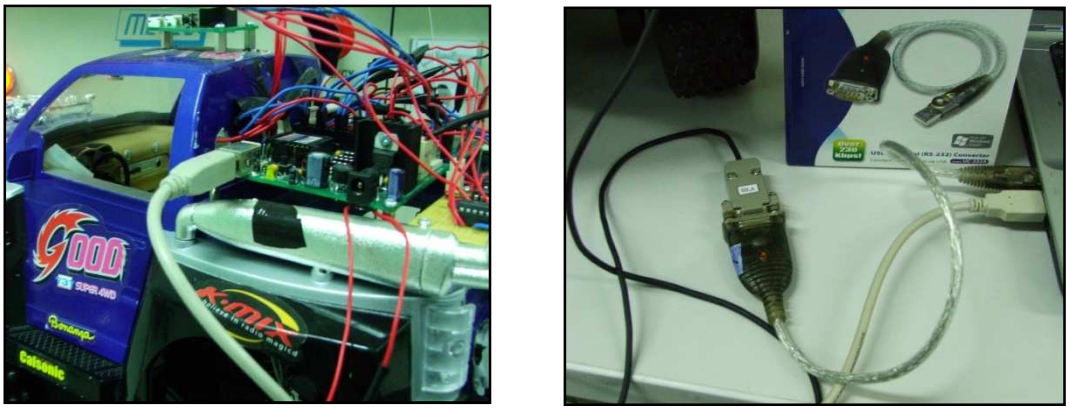
4.1 การทดสอบส่วนฮาร์ดแวร์และซอฟท์แวร์

ขั้นตอนแรกเริ่มจากการจ่ายไฟให้กับบอร์ดทดลอง ดังรูปที่ 4.1



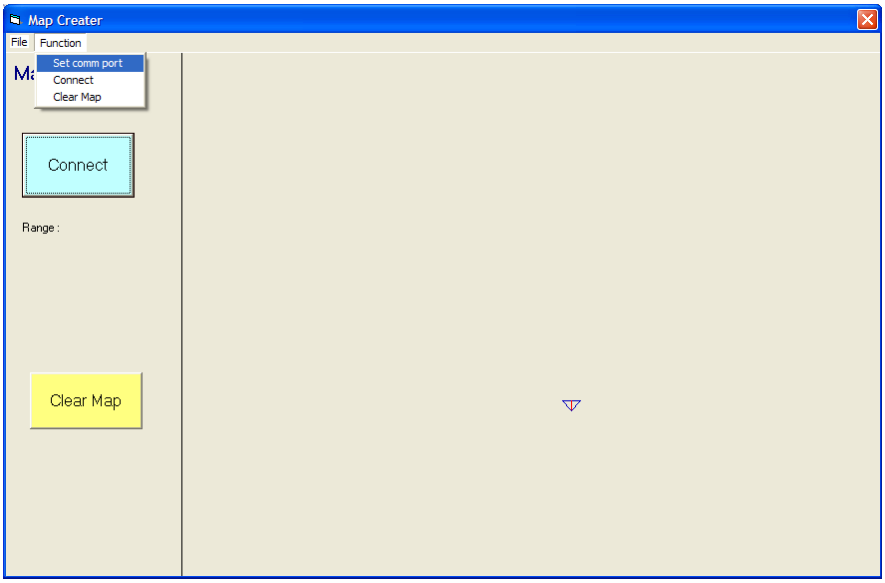
รูปที่ 4.1 แสดงการจ่ายไฟให้กับบอร์ดทดลอง

จากนั้นทำการเชื่อต่อพอร์ต RS232 กับเครื่องคอมพิวเตอร์โน้ตบุ๊ก ซึ่งคอมพิวเตอร์โน้ตบุ๊กนี้ไม่มีพอร์ต serial จึงได้ใช้อุปกรณ์พิเศษเพิ่มเติมคือ USB-to-serial (RS-232) converter เพื่อทำการเชื่อมต่อระหว่างภาค MCUกับคอมพิวเตอร์โน้ตบุ๊ก ดังรูปที่ 4.2



รูปที่ 4.2 การเชื่อต่อพอร์ต RS232 กับเครื่องคอมพิวเตอร์โน้ตบุ๊ก

จากนั้นจึงทำการเรียกโปรแกรมที่เขียนด้วย Visual Basicที่สมบรูณ์แล้ว (Map Maker) ดังรูปที่ 4.3



รูปที่ 4.3 แสดงโปรแกรม Map Maker

เลือกพอร์ต Serial ให้ตรงกับเครื่องของตัวเอง (ในที่นี้เป็นพอร์ต com 6) ดังรูปที่ 4.4

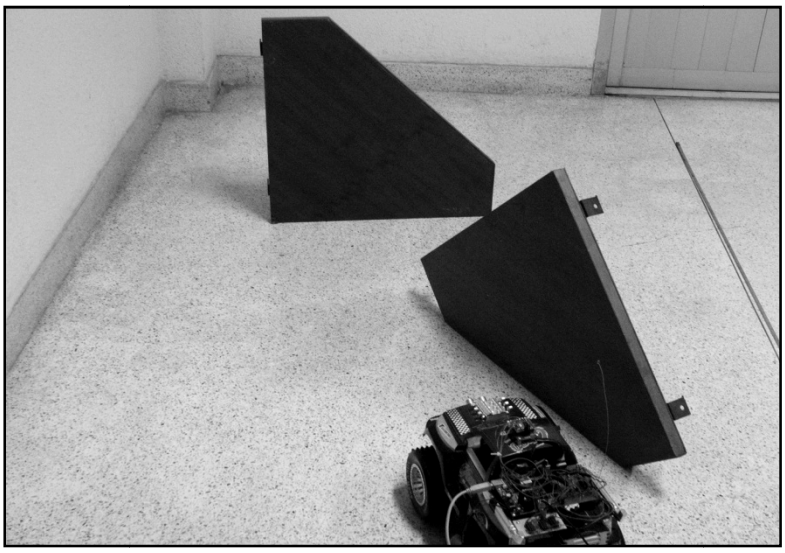
รูปที่ 4.4 แสดงการเลือกพอร์ต com

เลือกคำสั่ง Connect เพื่อพร้อมที่จะทำการสำรวจจุดที่ต้องการดังที่แสดงดังรูป 4.5

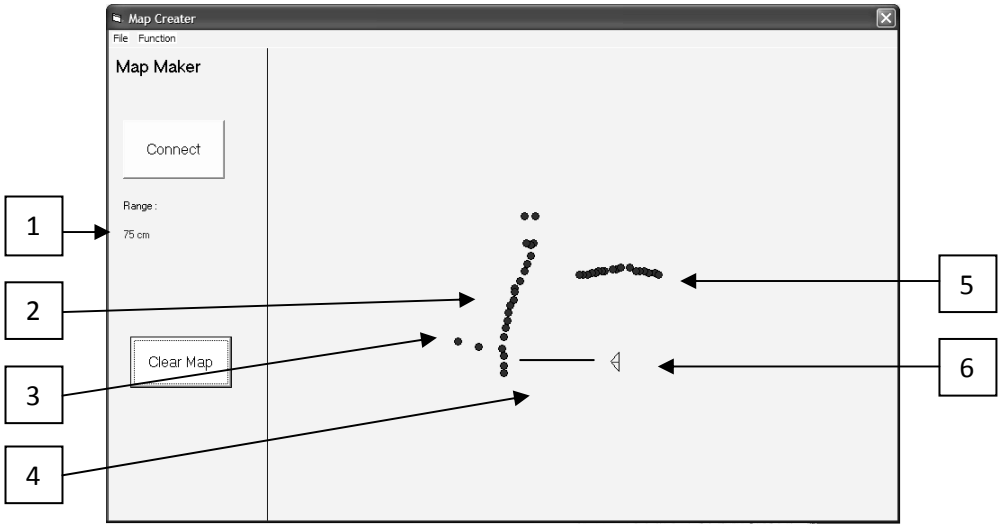
รูปที่ 4.5 แสดงการเลือกคำสั่ง Connect

หลังจากนั้นได้ทำการนำรถสำรวจไปสำรวจสถานจำลองต่างๆ 4 ที่ดังนี้

จุดที่ทำการสำรวจที่ 1



รูปที่ 4.6 แสดงจุดที่ใช้สำรวจที่ 1



รูปที่ 4.7 แสดงภาพของโปรแกรมที่ได้จากการสำรวจในจุดที่ 1

ลักษณะของสถานที่1 คือ ด้านซ้ายมีกำแพงขวางอยู่ ด้านขวามีไม้กระดาน 2 แผ่น ขวางอยู่ จากการทดลองโปรแกรม ปรากฏดังนี้

หมายเลข 1 คือ ค่าระยะห่างระหว่างรถกับกำแพงที่ปรากฏบนจอ ซึ่งเป็นจุดสุดท้าย มีระยะห่าง 75 เซนติเมตร

หมายเลข 2 คือ กำแพง

หมายเลข 3 คือ จุดที่เกิดจากการผิดพลาด

หมายเลข 4 คือ ระยะห่างระหว่างรถกับกำแพง

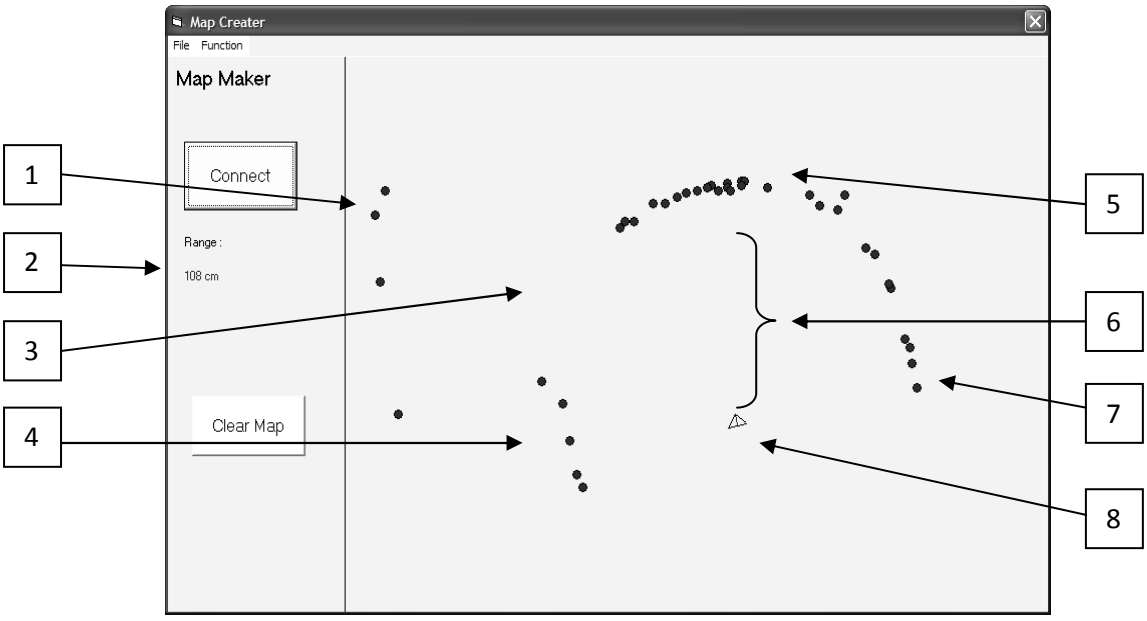
หมายเลข 5 คือ กระดาน

หมายเลข 6 คือ ตัวรถซึ่ง หันหน้ารถเข้าหากำแพง

จากการประมวลผลของโปรแกรม มีจุดผิดพลาด เกิดขึ้น 2 จุดจาก จากจุด 50 จุด แสดงว่ามีค่าความผิดพลาด 4 เปอร์เซ็นต์

จุดที่ทำการสำรวจที่ 2

รูปที่ 4.8 แสดงจุดที่ใช้สำรวจที่ 2



รูปที่ 4.9 แสดงภาพของโปรแกรมที่ได้จากการสำรวจในจุดที่ 2

ลักษณะของสถานที่ 2 คือ ข้างหน้ามีถังดับเพลิงวางอยู่ 2 ถัง ด้านข้างทางขวาเป็นกำแพง ทางด้านซ้ายเป็นที่โล่งต่อมาเป็นกำแพง จากการทดลองโปรแกรมปรากฏดังรูป

หมายเลข 1 คือ จุดที่เกิดจากการผิดพลาด

หมายเลข 2 คือ ค่าระยะห่างระหว่างรถกับกำแพงที่ปรากฏบนจอ ซึ่งเป็นจุดสุดท้าย มีระยะห่าง 108 เซนติเมตร

หมายเลข 3 คือ ที่โล่ง

หมายเลข 4 คือ กำแพงด้านซ้าย

หมายเลข 5 คือ ถังดับเพลิง

หมายเลข 6 คือ ระยะห่างระหว่างรถกับถังดับเพลิง

หมายเลข 7 คือ กำแพงด้านขวา

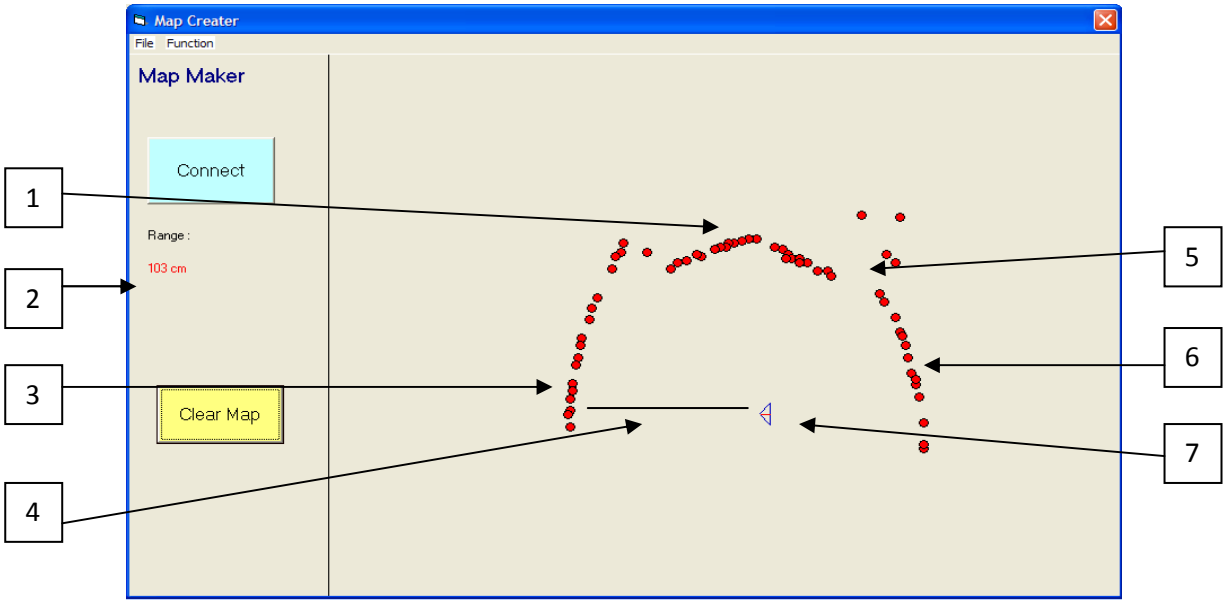
หมายเลข 6 คือ ตัวรถซึ่ง หันหน้ารถเข้าหาถังดับเพลิง

จากการประมวลผลของโปรแกรม มีจุดผิดพลาด เกิดขึ้น 10 จุดจาก จากจุด 50 จุด แสดงว่ามีค่าความผิดพลาด 20 เปอร์เซ็นต์

สาเหตุของค่าความผิดพลาดเกิดจากการสะท้อนกลับของวัตถุผิวโค้ง ดังเช่นถังดับเพลิงที่มีผิวโค้งและอยู่ใกล้กัน

จุดที่ทำการสำรวจที่ 3

รูปที่ 4.10 แสดงจุดที่ใช้สำรวจที่ 2



รูปที่ 4.11 แสดงภาพของโปรแกรมที่ได้จากการสำรวจในจุดที่ 3

ลักษณะของสถานที่ 3 คือ ข้างหน้ามีเก้าอี้หักมุมขวางอยู่ 1 ตัว ด้านข้างทางขวาเป็นกำแพง ทางด้านซ้ายเป็นกำแพง ระหว่างเก้าอี้กับกำแพงมีช่องว่างอยู่ จากการทดลองโปรแกรมปรากฏดังรูป

หมายเลข 1 คือเก้าอี้หักมุม

หมายเลข 2 คือ ค่าระยะห่างระหว่างรถกับกำแพงที่ปรากฏบนจอ ซึ่งเป็นจุดสุดท้าย มีระยะห่าง 103 เซนติเมตร

หมายเลข 3 คือ กำแพงด้านซ้าย

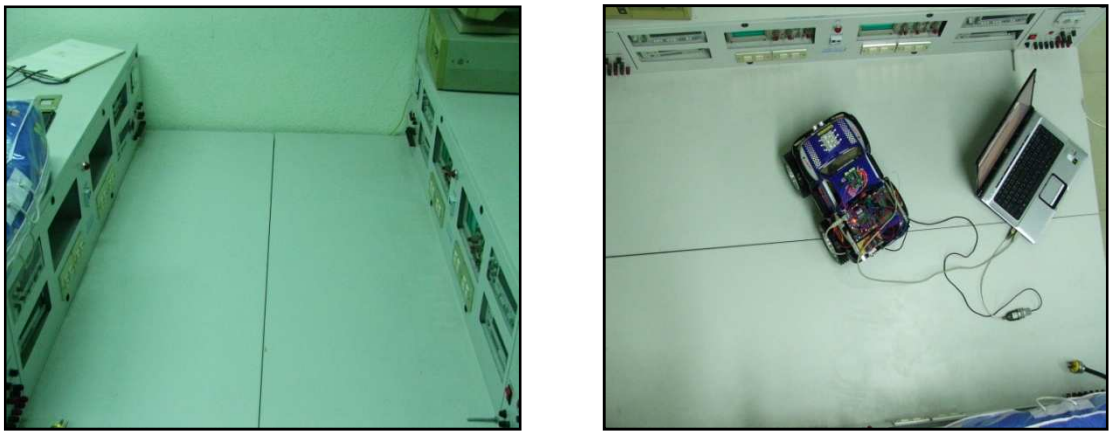
หมายเลข 4 คือ ระยะห่างระหว่างรถกับกำแพง

หมายเลข 5 คือ เก้าอี้หักมุม

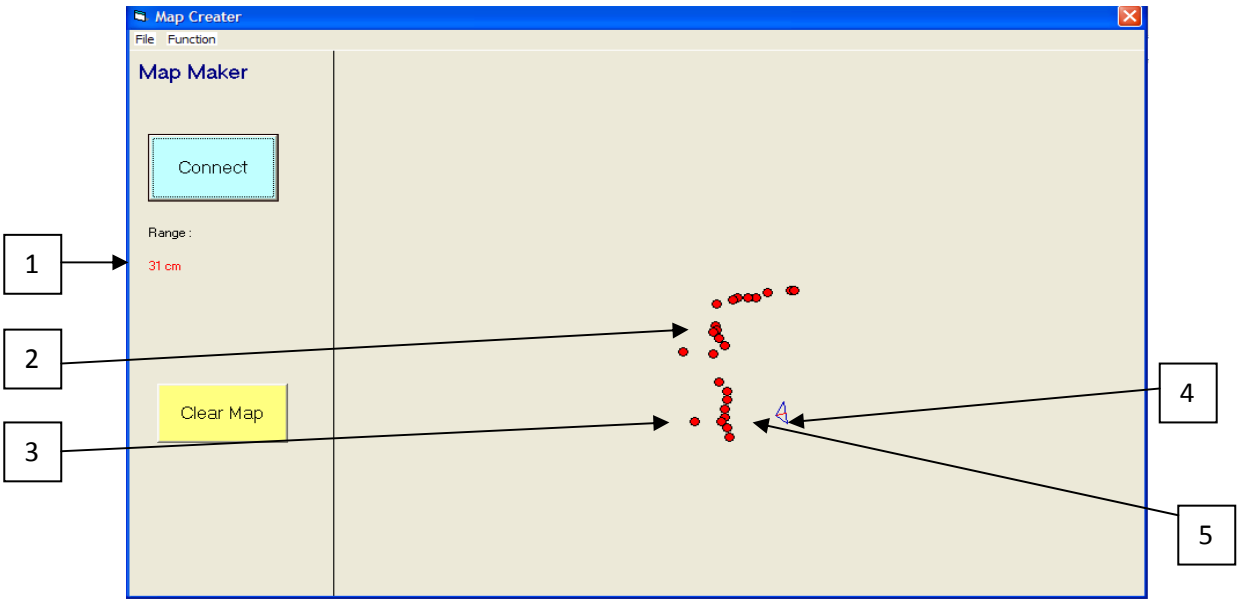
หมายเลข 6 คือ กำแพงด้านขวา

หมายเลข 7 คือ ตัวรถซึ่ง หันหน้ารถเข้าหากำแพงด้านซ้าย

จุดที่ทำการสำรวจที่ 4



รูปที่ 4.12 แสดงจุดที่ใช้สำรวจที่ 4



รูปที่ 4.13 แสดงภาพของโปรแกรมที่ได้จากการสำรวจในจุดที่ 4

\

ลักษณะของสถานที่ 4 คือ มีลักษณะอุปกรณ์ทางไฟฟ้าอยู่ทางด้านข้าง ข้างหน้าเป็นกำแพง

หมายเลข 1 คือ ค่าระยะห่างระหว่างรถกับอุปกรณ์ทางไฟฟ้าที่ปรากฏบนจอ ซึ่งเป็นจุดสุดท้าย มีระยะห่าง 31 เซนติเมตร

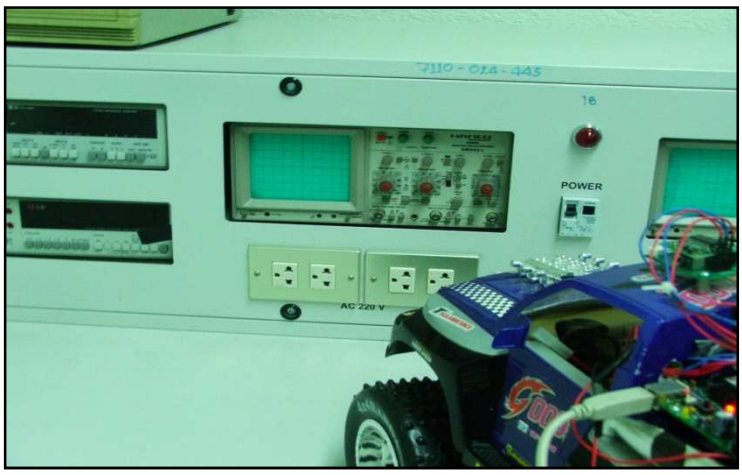
หมายเลข 2 คือ จุดผิดพลาด

หมายเลข 3 คือ อุปกรณ์ทางไฟฟ้า

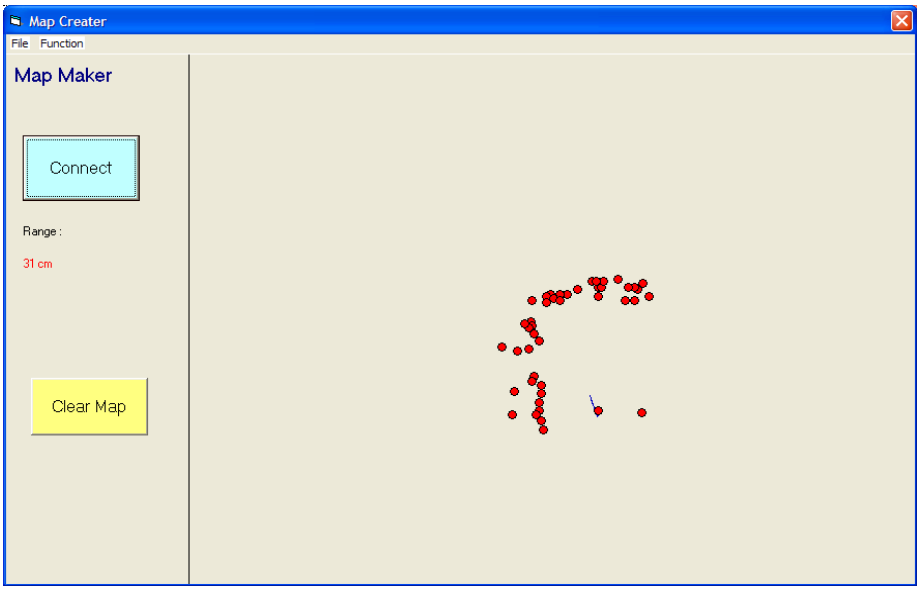
หมายเลข 4 คือ ตัวรถซึ่ง หันหน้ารถเข้าหาอุปกรณ์ทางด้านซ้าย

หมายเลข 5 คือ ระยะห่างระหว่างรถกับกำแพง

ในการทดลองที่จุดสำรวจที่ 4 นี้ จะเกิดส่วนที่ผิดพลาดขึ้นเนื่องมาจาก ทิศทางที่หน้ารถเกิดบังเอิญหันไปหา เครื่องมือหรืออุปกรณ์ที่มีคลื่นแม่เหล็กไฟฟ้าสูง จะทำให้เกิดความผิดปกติของโปรแกรมขึ้นดังรูปที่ 4.14



รูปที่ 4.14 แสดงทิศทางที่หน้ารถหันไปหา เครื่องมือหรืออุปกรณ์ต่างๆที่มีคลื่นแม่เหล็กไฟฟ้าสูง



รูปที่ 4.15 แสดงภาพของโปรแกรมที่เกิดการผิดพลาด (EROR)

บทที่ 5
สรุปผลและข้อเสนอแนะ

5.1 สรุปผลของการทดสอบส่วนฮาร์ดแวร์และซอฟท์แวร์

จากการทดลองโปรแกรม เขียนแผนที่ เมื่อรถสำรวจเคลื่อนที่ไปในจุดที่ต้องการสำรวจ มันสามารถบ่งบอกว่ามีสิ่งกีดขวางระยะห่างจากตัวรถเท่าใด ดังจะเห็นได้จากการที่เรานำรถเข้าไปในจุดที่ต้องการสำรวจ มันสามารถแสดงผลออกมาทำให้ทราบว่ามีสิ่งกีดขวางระยะห่างจากตัวรถเท่าใด แม้จะมีระยะทางที่ผิดพลาดอยู่บ้างอันเกิดมาจากการที่อุปกรณ์ ultrasonic เองที่มีการส่งสัญญาณไปกลับ ซึ่งในการที่มันสะท้อนกลับมานั้นจะมีขอบเขตที่กว้าง ซึ่งคณะผู้จัดทำได้ทำการแก้ไขโดยการแหงนอุปกรณ์ ultrasonic ขึ้นเพื่อให้การสะท้อนกลับมาของสัญญาณโดนอุปกรณ์น้อยลง จึงทำให้มีการผิดพลาดที่ลดลง แต่ก็มีปัญหาข้อใหม่เกิดขึ้นคือสิ่งกีดขวางที่อยู่ใกล้แต่มีขนาดไม่สูงมันจะมองไม่เห็น แต่นั้นก็เป็นการผิดพลาดที่น้อยเมื่อเทียบกับของเดิม ต่อมาเกี่ยวกับอุปกรณ์โมดูลเข็มทิศ ซึ่งเป็นสิ่งที่บ่งบอกถึงทิศทางสามารถทำหน้าที่ได้เป็นอย่างดีมีข้อผิดพลาดที่น้อยมาก แต่ปัญหาสิ่งเดียวของโมดูลเข็มทิศจากการทดลองคือเมื่อมันอยู่ในห้องที่มีอุปกรณ์ไฟฟ้าทำให้มีการการแผ่ของสนามแม่เหล็กเกิดขึ้น โมดูลเข็มทิศซึ่งอาศัยสนามแม่เหล็กโลก จะเกิดการผิดเพี้ยนอย่างมาก อุปกรณ์ต่อมาที่มีการทดลองคือ encoder ซึ่งจะมีข้อผิดพลาดพอสมควร กล่าวคือ เมื่อมีการขับเคลื่อนไม่ครบรอบของการสะท้อนมันจะไม่แสดงผลออกมา แต่ในความเป็นจริงแล้วรถมีการเคลื่อนที่

ในส่วนของโปรแกรม VB การทำงานของมันจะมีข้อจำกัด คือ เมื่อกำหนดกรอบหรือขอบเขตของการแสดงผล ซึ่งถ้าหากมีพื้นที่กว้าง การแสดงผลต้องกำหนดจุดไว้มาก มันจะแสดงผลช้า แต่ถ้ากำหนดพื้นที่แคบถึงแม้ว่ามันจะแสดงผลเร็วแต่มันจะเขียนแผนที่ได้ไม่ครอบคลุม

นอกจากนี้ยังมี โดยภาพรวมแล้วจากการทดลองนี้หากพื้นที่ ไม่เกินสองคูณสองเมตร มันจะมีการปัญหาอีกก็คือ การบังคับจากรีโมทเป็นสิ่งที่เกิดขึ้นรวดเร็วมาก ทำให้ MCU รับข้อมูลไม่ทันว่าตอนนี้บังคับให้รถวิ่งหรือหยุด วิ่งไปข้างหน้าหรือข้างหลัง ดังนั้นจะออกแบบวงจรให้เข้า

ไป Interrupt ตัว MCU เพื่อให้ทำงานได้ทัน วงจรนี้จะใช้ ลอจิกเกท โดยสัญญาณมาจาก M1-M4 โดยมันจะส่ง INT. เมื่อรถเดินหน้าหรือถอยหลังเท่านั้น กรณีอื่นจะไม่ INT.

5.2 ข้อเสนอแนะ

5.2.1 ในการส่งผลออกมาทางจอ ควรปรับปรุงจากการนำผลมาตามสาย เพื่อความสะดวก ควรเป็นอุปกรณ์ที่ไร้สาย เช่น wireless เป็นต้น

5.2.2 ในการนำไปใช้ ไม่ควรมีสนามแม่เหล็กอื่นมารบกวน

เอกสารอ้างอิง

ธีรวัฒน์ ประกอบผล. (2542). **การประยุกต์ใช้งานไมโครคอนโทรเลอร์.** พิมพ์ครั้งที่ 3. กรุงเทพฯ : สมาคมส่งเสริมไทย-ญี่ปุ่น

อุทัย สุขสิงห์. (2547). **ไมโครโพรเซสเซอร์และไมโครคอนโทรเลอร์ MCS-51.** พิมพ์ครั้งที่ 1.กรุงเทพฯ : สมาคมส่งเสริมไทย-ญี่ปุ่น

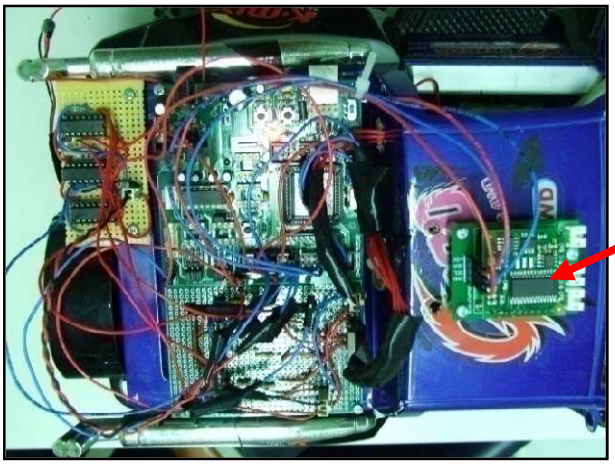
ธาริน สิทธิธรรมชารี. (2549). **คู่มือเรียนเขียนโปรแกรม Visual Basic 2005.** พิมพ์ครั้งที่ 1. กรุงเทพฯ : ซัคเซส มีเดีย

โชคชัย เตชพรรุ่ง. (2539). **เจาะแก่น Visual Basic.** พิมพ์ครั้งที่ 1.กรุงเทพฯ : ซีเอดยูเคชั่น

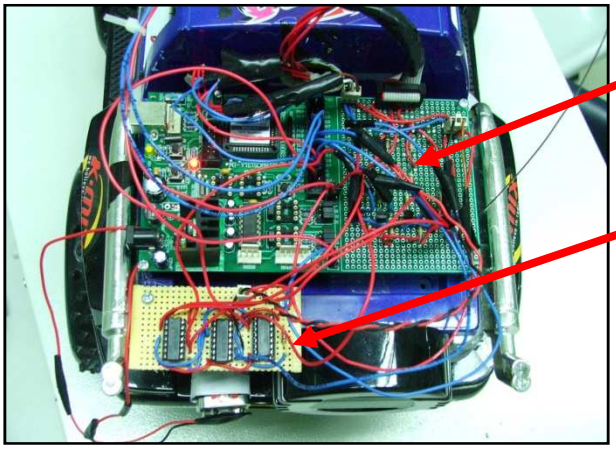
ภาคผนวก

ภาพถ่ายในส่วนฮาร์ดแวร์แสดงอุปกรณ์ต่างๆที่ใช้

1

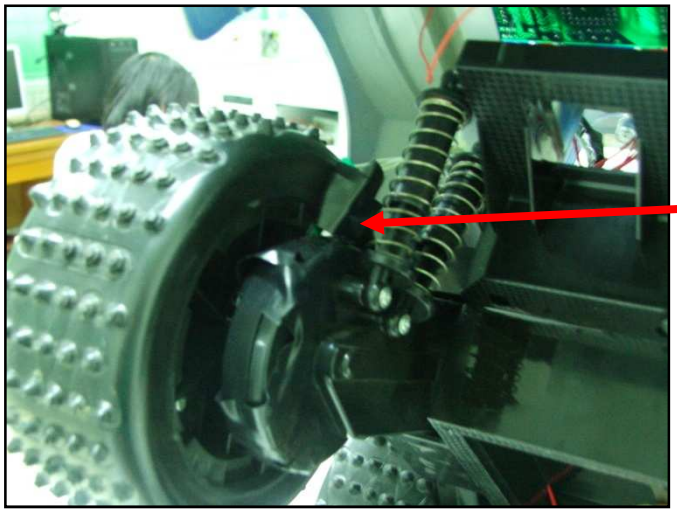


2



3

4



5

6

7

8

9

อธิบายอุปกรณ์

1. โมดูลตรวจจับและวัดระยะทางด้วยอัตร้าโซนิค SRF-04 Ultrasonic Distance Detector Module
2. โมดูลเข็มทิศดิจิตอล CMPS03 Digital Compass Module
3. บอร์ดทดลอง ET-CP-JR51-USB-V1
4. วงจรสัญญาณอินเตอร์รัพท์
5. แผงวงจรตรวจจับรหัสล้อ ZX-21
6. ภาพรถสำรวจที่ถ่ายจากมุมด้านหน้า (Front view)
7. ภาพรถสำรวจที่ถ่ายจากมุมด้านข้าง (Side view)
8. ภาพรถสำรวจที่ถ่ายจากมุมบน(Top view)

Code Broad Program

```
*/
#include <at89c5131.h>
#include <intrins.h>
#include "config.h"
sbit srf_trigger = P0^0;
sbit srf_echo = P0^1;
sbit SCL = P1^0;
sbit SDA = P1^1;
sbit test_bit = P1^3;
unsigned char motor;
void init_int() {
EA = 1;  //enable all int.
EX1 = 1;              //enable external int. of INT1
IT1 = 1;  //trigger at high to low change of INT1
}
void init_counter() {
//use T0 to count pulse to calculate distance
TMOD |= 0x05;  //use T0 as counter (pin 3.4) and use mode 1 (16 bit)
TL0 = TH0 = 0;
TR0 = 1;   //switch on counter 0}
void i2c_delay() {
unsigned char i;
for(i=0;i<8;i++)
;
}
void txd(unsigned char val) {
SBUF = val;
while(!TI)
;
```

```
TI = 0;
}
void timer1_service() interrupt 3 {
motor = P2;
}
void i2c_high() {
SCL = 1;
i2c_delay();
}
void i2c_low() {
SCL = 0;
i2c_delay();
}
void i2c_start() {
SDA = 1;
i2c_high();
SDA = 0;
i2c_delay();
i2c_low();
SDA = 1;
}
void i2c_stop() {
SDA = 0;
i2c_high();
SDA = 1;
}
bit i2c_write(unsigned char dat) {
unsigned char i;
bit outbit;
for(i=0;i<8;i++) {
```

```
outbit = dat & 0x80;
SDA = outbit;
dat <<= 1;
i2c_high();
i2c_low();
}
SDA = 1;
i2c_high();
outbit = SDA;
i2c_low();
return outbit;
}
unsigned char i2c_read() {
unsigned char i,dat;
bit inbit;
dat = 0;
for(i=0;i<8;i++) {
i2c_high();
inbit = SDA;
dat <<= 1;
dat = dat | inbit;
i2c_low();
}
SDA = 1;
i2c_high();
inbit = SDA;
i2c_low();

return dat;
}
```

```
unsigned char cpms_read() {
bit ok;
unsigned char dat;
i2c_start();
ok = i2c_write(0xC0);
if(ok != 0)
txd(0xFA);
ok = i2c_write(0x01);       //register
if(ok != 0)
txd(0xFB);
i2c_start();
ok = i2c_write(0xC1);
if(ok != 0)
txd(0xFC);
dat = i2c_read();
i2c_stop();
return dat;
}
void delay_ms(unsigned char m) {
int i;
for(i=0;i<m * 2000;i++)
;
}
void init_uart() {
//use T2 to generate baud rate at 9,600 bps
SCON = 0x52;
T2CON = 0x30;
RCAP2H = 0xFF;
RCAP2L = 0xB1;
TR2 = 1;            //start baud rate gen.
```

```
}
void init_srf()
{ //use T1 to count width of echo signal     TMOD |= 0x10; }
unsigned int srf() {
unsigned char i;
unsigned int dat;
TL1 = 0x00;
TH1 = 0x00;
TR1 = 0;
srf_trigger = 0;
srf_echo = 1;
srf_trigger = 1;    //send trigger at least 10 us

for(i=0;i<20;i++) {
_nop_();                    //delay about 10 us (1 machine cycle = 0.5 us)
}
srf_trigger = 0;
while(!srf_echo);
TR1 = 1;     //start timer 1
while(srf_echo);
TR1 = 0;            //stop timer 1

dat = TH1;
dat <<= 8;
dat |= TL1;

return dat;
}
unsigned char srf_cal() {
unsigned char i;
```

```
unsigned int sum,val;
sum = 0;
for(i=0;i<2;i++) {
val = srf();
sum = sum + (val/2);
delay_ms(10);
}
sum /= 114;

if(sum > 250)
return 0xFF;
else {
i = sum & 0x00FF;
return i;
}
}
void  main() {
unsigned char dis,deg,m;
unsigned char car_dis;
init_uart();
init_int();
motor = 0xFF;
init_srf();
init_counter();
car_dis = 0;
while(1) {
dis = srf_cal();
deg = cpms_read();
txd(0x7E);
txd(dis);//obsruct distance
```

```
txd(deg);          //orientation of car
txd(motor);        //data of motor
m = P2;
m &= 0x0F;
car_dis = 0;
if(m == 0x0F) {    //if car stop FW/BW so cal. for dis
TR0 = 0;
car_dis = TL0;
TH0 = TL0 = 0x00;
motor = 0xFF;
TR0 = 1;
}
txd(car_dis);
}
}
```

Visaul Basic code

```
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim myscale As Integer
Dim org_x As Integer
Dim org_y As Integer
Dim org_view_y As Integer
Dim PI As Double

Dim dis As Integer
```

```
Dim theta As Byte
Dim x1 As Integer
Dim y1 As Integer
Dim x2 As Integer
Dim y2 As Integer
Dim x3 As Integer
Dim y3 As Integer

Dim car_center_x As Integer  'x of car real
Dim car_center_y As Integer  'x
Dim car_y_view As Integer  'y of car that will draw in screen
Dim car_nv As Double 'car normal vector theta
Dim car_w As Integer 'half of car width = 11 cm (car width = 22 cm)
Dim tri_l As Integer 'length of triangle of car
Dim offset As Double

Dim byte_count As Byte

Dim obs_x(50) As Integer  'point of obstruct
Dim obs_y(50) As Integer
Dim obs_count As Integer

Dim tx_ok As Boolean 'check header of comm.
Dim vel As Integer  'scale in cm
Dim first As Boolean

Dim comm_port As Integer


Private Function check_cos(ByVal deg As Double) As Double
```

```
If deg >= 0 And deg <= 90 Then
check_cos = Cos(deg)
ElseIf deg > 90 And deg <= 180 Then
check_cos = -Cos(deg)
ElseIf deg > 180 And deg <= 270 Then
check_cos = -Cos(deg)
ElseIf deg > 270 And deg <= 360 Then
check_cos = Cos(deg)
End If
End Function

Private Function check_sin(ByVal deg As Double) As Double
If deg >= 0 And deg <= 90 Then
check_sin = Sin(deg)
ElseIf deg > 90 And deg <= 180 Then
check_sin = Sin(deg)
ElseIf deg > 180 And deg <= 270 Then
check_sin = -Sin(deg)
ElseIf deg > 270 And deg <= 360 Then
check_sin = -Sin(deg)
End If
End Function


Private Sub draw_car()
Dim theta1 As Double      'direction of car
Dim theta2 As Double    'angle to right base of triangle
Dim theta3 As Double    'angle to left base of triangle
theta1 = car_nv
theta2 = theta1 - 90
```

```
If theta2 < 0 Then
theta2 = 360 - Abs(theta2)
End If

theta3 = theta1 + 90

x1 = car_center_x + CInt(car_w * check_cos(theta2 * PI / 180))
y1 = car_y_view + CInt(car_w * check_sin(theta2 * PI / 180))
x2 = car_center_x + CInt(tri_l * check_cos(theta1 * PI / 180))
y2 = car_y_view + CInt(tri_l * check_sin(theta1 * PI / 180))
x3 = car_center_x + CInt(car_w * check_cos(theta3 * PI / 180))
y3 = car_y_view + CInt(car_w * check_sin(theta3 * PI / 180))

Line1.x1 = x1
Line1.y1 = y1
Line1.x2 = x2
Line1.y2 = y2

Line2.x1 = x2
Line2.y1 = y2
Line2.x2 = x3
Line2.y2 = y3

Line3.x1 = x3
Line3.y1 = y3
Line3.x2 = x1
Line3.y2 = y1

Line4.x1 = car_center_x
Line4.y1 = car_y_view
```

```
Line4.x2 = x2
Line4.y2 = y2

End Sub

Private Sub init_data()
PI = 3.14159
myscale = Int(Form1.Width / Form1.ScaleWidth)
org_x = ((Form1.Width - Line5.x1) / 2) / myscale
org_x = org_x + 70   'calibrate to center of screen
org_y = (Form1.Height / 2) / myscale
org_view_y = org_y
car_w = 5 'width of car in cm
tri_l = 5 'use to draw triangle shape of car sensor

car_center_x = org_x
car_center_y = org_y
car_y_view = car_center_y
car_nv = 90

x1 = org_x + car_w
y1 = org_y
x2 = org_x
y2 = org_y - tri_l
x3 = org_x - car_w
y3 = org_y

obs_count = 0
first = True
tx_ok = False
```

```
comm_port = -1
End Sub

Private Sub init_comm()
If MSComm1.PortOpen = True Then
i = MsgBox("Port is connected already", vbExclamation)
Exit Sub
End If

If comm_port = -1 Then
comm_port = 3  'default comm port
End If

MSComm1.CommPort = comm_port
MSComm1.Settings = "9600,n,8,1"
MSComm1.PortOpen = True
MSComm1.RThreshold = 1
MSComm1.InputLen = 1
End Sub

Private Sub move_car(ByVal dir As Byte)   'F,B,FL,FR,BL,BR
Select Case dir
Case 0  'F
car_center_x = car_center_x + CInt(vel * check_cos(car_nv * PI / 180))
car_center_y = car_center_y + CInt(vel * check_sin(car_nv * PI / 180))
car_y_view = car_y_view + CInt(vel * check_sin(car_nv * PI / 180))
Case 1  'B
car_center_x = car_center_x - CInt(vel * check_cos(car_nv * PI / 180))
car_center_y = car_center_y - CInt(vel * check_sin(car_nv * PI / 180))
car_y_view = car_y_view - CInt(vel * check_sin(car_nv * PI / 180))
```

```
End Select
End Sub

Private Sub clear_Click()
Call clear_map
End Sub

Private Sub Command1_Click()
Call clear_map
End Sub

Private Sub Command2_Click()
Call init_comm
End Sub

Private Sub connect_Click()
Call init_comm
End Sub

Private Sub Exit_Click()
End
End Sub

Private Sub Form_Load()
byte_count = 0
theta = 90
Call init_data
Call draw_car
End Sub
```

```
Private Sub draw_obstruct()
Dim i As Integer

For i = 0 To obs_count - 1
If obs_x(i) <> -1 Then
obs(i).Visible = True
obs(i).Left = obs_x(i)
obs(i).Top = obs_y(i)
End If
Next i
If obs_count >= 50 Then
obs_count = 0
End If
End Sub

Private Sub clear_map()
Dim i As Integer

For i = 0 To 49
obs_x(i) = -1
obs_count = 0
obs(i).Visible = False
Next i

End Sub

Private Function check_dup(ByVal x As Integer, ByVal y As Integer) As Boolean
Dim i As Integer
For i = 0 To obs_count
If Abs(obs_x(i) - x) < 2 And Abs(obs_y(i) - y) < 2 Then
```

```
check_dup = True
Exit Function
End If
Next i
check_dup = False
End Function



Private Sub MSComm1_OnComm()
Dim d As Integer
Dim num As Variant
Dim x As Integer
Dim y As Integer
Dim car_dir As Integer

Select Case MSComm1.CommEvent
Case comEvReceive
num = MSComm1.Input

If tx_ok = False Then   'check for protocol header
If CInt(Asc(num)) = &H7E Then
tx_ok = True
End If
Exit Sub
End If

car_dir = 3
If byte_count = 0 Then
```

```
If first = False Then
dis = Asc(num)
If dis <> 255 Then  'cannot see in range of SRF-04
x = car_center_x + CInt(dis * check_cos(car_nv * PI / 180))
y = car_center_y + CInt(dis * check_sin(car_nv * PI / 180))
If check_dup(x, y) = False Then
obs_x(obs_count) = x
obs_y(obs_count) = y
obs_count = obs_count + 1
End If
End If
End If
ElseIf byte_count = 1 Then
If first = True Then
'first = False
offset = 270 - CInt((Asc(num) / 255) * 360)
End If

car_nv = CInt((Asc(num) / 255) * 360) + offset
car_nv = CDbl(car_nv)
ElseIf byte_count = 2 Then
d = Asc(num)   'm1-m4
d = d And &HF
'check for direction now
Select Case d
Case &H6
car_dir = 0
Case &H9
car_dir = 1
Case Else
```

```
car_dir = 3
End Select
ElseIf byte_count = 3 Then
d = Asc(num)
d = CInt(d * 5.11)
End If
here:

byte_count = byte_count + 1
If byte_count >= 4 Then
byte_count = 0
tx_ok = False
If first = True Then
first = False
Exit Sub
End If
End If

If dis = 255 Then
Label3.Caption = "..."
Else
Label3.Caption = dis & " cm"
End If

If car_dir <> 3 Then
vel = d
Call move_car(car_dir)
End If
Call draw_car
Call draw_obstruct
```

```
End Select
End Sub

Private Sub set_port_Click()
Dim port As String
port = InputBox("Enter comm port", "Message", 0, 200, 300)
comm_port = CInt(port)
End Sub
```

ประวัติย่อนิสิตผู้ทำโครงงาน

ชื่อ-สกุล	นายสาธิต โยคเสนะกุล
วัน เดือน ปีเกิด	30 ตุลาคม 2528
สถานที่เกิด	จังหวัดนครราชสีมา
สถานที่อยู่ปัจจุบัน	301/14 ต.ในเมือง อ.เมือง จ.นครราชสีมา 30000
โทรศัพท์	081-2825035

ประวัติการศึกษา

พ.ศ. 2544	โรงเรียนราชสีมาวิทยาลัย
พ.ศ. 2547	โรงเรียนราชสีมาวิทยาลัย
ปี ปัจจุบัน	กำลังศึกษาระดับปริญญาตรี คณะวิศวกรรมศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ

ประวัติย่อนิสิตผู้ทำโครงงาน

ชื่อ-สกุล	นางสาวเณริศศา บรรลือ
วัน เดือน ปีเกิด	2 กันยายน 2528
สถานที่เกิด	จังหวัดแพร่
สถานที่อยู่ปัจจุบัน	126 ม.4 ต.ทุ่งโฮ้ง อ.เมือง
	จ.แพร่ 54000
โทรศัพท์	089-1294323

ประวัติการศึกษา

พ.ศ. 2544	โรงเรียนนารีรัตน์จังหวัดแพร่
พ.ศ. 2547	โรงเรียนนารีรัตน์จังหวัดแพร่
ปี ปัจจุบัน	กำลังศึกษาระดับปริญญาตรี คณะวิศวกรรมศาสตร์
	มหาวิทยาลัยศรีนครินทรวิโรฒ

ประวัติย่อนิสิตผู้ทำโครงงาน

ชื่อ-สกุล	นายศิริชัย ปานพรหมมินทร์
วัน เดือน ปีเกิด	17 กรกฎาคม 2528
สถานที่เกิด	จังหวัดชุมพร
สถานที่อยู่ปัจจุบัน	11/6 ถ.หลังสวน ต.หลังสวน อ.หลังสวน
	จ.ชุมพร 86110
โทรศัพท์	086-9500021

ประวัติการศึกษา

พ.ศ. 2544	โรงเรียนสวนศรีวิทยา
พ.ศ. 2547	โรงเรียนศรียาภัย
ปี ปัจจุบัน	กำลังศึกษาระดับปริญญาตรี คณะวิศวกรรมศาสตร์
	มหาวิทยาลัยศรีนครินทรวิโรฒ