การพัฒนาระบบเชื่อมต่อ CAN สำหรับงานอุตสาหกรรม

DEVELOPMENT OF INTERFACING SYSTEM BASED ON CAN FOR INDUSTRIAL

นายขนบสิทธิ์ เขียวรัมย์
นายชาติพิชิต สำราญกิจ
นายธีรพงศ์ เกื้อเกตุ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมการวัดคุม
ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

DEVELOPMENT OF INTERFACING SYSTEM BASED ON CAN FOR INDUSTRIAL

KHANOPSITT KHEAWRUM

CHATPICHIT SAMRANKIJ

TEERAPONG KUAKET

A THESIS SUBMITTED IN PARTIAL FLUFILLMENT

OF THE REQUIREMEN FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2004

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ การพัฒนาระบบเชื่อมต่อ CAN สำหรับงานอุตสาหกรรม

DEVELOPMENT OF INTERFACING SYSTEM BASED ON CAN FOR INDUSTRIAL

นักศึกษาผู้จัดทำ

นายขนบสิทธิ์	เขียวรัมย์	**รหัสประจำตัว**	45015590
นายชาติพิชิต	สำราญกิจ	**รหัสประจำตัว**	45015594
นายธีรพงศ์	เกื้อเกตุ	**รหัสประจำตัว**	45015600

ปริญญา วิศวกรรมศาสตรบัณฑิต

สาขา วิศวกรรมการวัดคุม

ปีการศึกษา 2547

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
รศ. ประภาษ อุคคกิมาพันธุ์	
อ. พิทยา ปาลนิล	

วัน/เดือน/ปี ที่สอบ วันพุธที่ 20 เมษายน พ.ศ. 2548

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(รศ.ประสิทธิ์ จุลเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	การพัฒนาระบบเชื่อมต่อ CAN สำหรับงานอุตสาหกรรม DEVELOPMENT OF INTERFACING SYSTEM BASED ON CAN FOR INDUSTRIAL
นักศึกษาผู้จัดทำ	นายขนบสิทธิ์ เขียวรัมย์ นายชาติพิชิต สำราญกิจ นายธีรพงศ์ เกื้อเกตุ
อาจารย์ที่ปรึกษา	รศ. ประภาษ อุดคกิมาพันธุ์ อ. พิทยา ปาลนิล
ปีการศึกษา	2547

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เสนอเกี่ยวกับมาตรฐานและข้อกำหนดต่าง ๆ ในโปรโตคอล CAN โดยรวมถึงการออกแบบ และสร้างชุดทดลองสำหรับทดสอบฟังก์ชันการทำงานของ CAN เพื่อให้เข้าใจถึงหลักการทำงานที่เกี่ยวกับโปรโตคอลของ CAN ได้ชัดเจนยิ่งขึ้น และเป็นข้อมูลให้แก่ผู้ที่มีความสนใจสามารถนำไปพัฒนาสำหรับการใช้ในงานควบคุมอัตโนมัติที่มีความซับซ้อนมาก ๆ ได้ โดยชุดทดลองจะถูกออกแบบให้มีลักษณะเป็นโมดูลของ CAN ที่มีความยืดหยุ่นสามารถนำไปประยุกต์ใช้งานได้ง่าย ซึ่งอาจนำไปต่อลงบนบอร์ดของ MCS-51 ทั่วไปได้ เพื่อสะดวกในการเรียนรู้และพัฒนาต่อไป

Thesis Title	Development Of Interfacing System Based On CAN For Industrial	
Authors	Mr. Khanopsitt	Kheawrum
	Mr. Chatpichit	Samrankij
	Mr. Teerapong	Kuaket
Thesis Advisor	Assoc.Prof. Prapart	Ukakimaparn
	Mr. Pittaya	Pannil
Year	2004	

ABSTRACT

This thesis presents concerning with standardization and specification of CAN protocol, including of designing and constructing the experimental board for testing operation functions of CAN , so as to better understand operation principle of CAN protocal and also being information for interesting person able to bring development. The experimental board designed as modular of CAN is used with more complicated automatic control system.By which it is designed asmodular, application of one is flexible and easy. For instance , general board of MCS-51 is used together with CAN , so as to study convenient and develop next time.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดีเพราะได้รับเมตตาจาก รองศาสตราจารย์ประภาษ อุคคกิมาพันธุ์ และ อาจารย์พิทยา ปาลนิล ที่ได้ให้คำแนะนำแก่ผู้วิจัยตลอดมา อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่าง ๆ ในการทำปริญญานิพนธ์นี้ ผู้วิจัยรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณอาจารย์ภาควิชาวิศวกรรมการวัดคุมทุกท่าน ที่ได้ให้ข้อเสนอแนะตลอดจนคำแนะนำอันเป็นประโยชน์ต่อการทำปริญญานิพนธ์นี้

และที่ลืมเสียไม่ได้คือ ขอกราบขอบพระคุณคุณพ่อคุณแม่ อันเป็นที่รักยิ่งที่สนับสนุนและเป็นแรงบันดาลใจในการทำปริญญานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุจูงใจ ของการวิจัย

เนื่องจากปัจจุบันโรงงานอุตสาหกรรมส่วนใหญ่ เริ่มที่จะปรับเปลี่ยนมาเป็นระบบอัตโนมัติมากขึ้นซึ่งมีความจำเป็นที่จะต้องมีการติดต่อสื่อสารซึ่งกันและกัน เพื่อนำข้อมูลที่ได้จากการวัดหรือการเปลี่ยนแปลงของกระบวนการนำไปวิเคราะห์และทำการคำนวณ เพื่อนำไปควบคุมกระบวนการอีกทีหนึ่งจึงจำเป็นที่ต้องศึกษาการรับส่งข้อมูล

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์นี้จะเป็นการศึกษาและออกแบบสร้างบอร์ดอินเตอร์เฟส CAN เพื่อนำไปใช้งานโดยต่อเข้ากับ Application ของเครื่องควบคุมหรือเครื่องมือวัดต่าง ๆ และสามารถทำการรับส่งระหว่างกันได้ โดยข้อมูลที่รับส่งกันนี้มีขนาดไม่มากนักและจะทำให้มีประสิทธิภาพดี อีกทั้งราคายังถูกกว่าการใช้อุปกรณ์รับส่งเดิมที่ใช้กันอยู่

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์เล่มนี้จะกล่าวถึง การสื่อสารข้อมูลโดยใช้โปรโตคอลของระบบ CAN และการออกแบบสร้างบอร์ดอินเตอร์เฟส CAN รวมถึงการทดสอบบอร์ดอินเตอร์เฟส CAN ซึ่งทดสอบส่งข้อมูลและรับข้อมูลระหว่างกันด้วย

1.4 ขั้นตอนการศึกษา

การทำโครงงานวิจัยในปริญญานิพนธ์ฉบับนี้ มีขั้นตอนการศึกษาเริ่มต้นจากการศึกษาการรับส่งข้อมูลด้วยโปรโตคอล CAN ว่ารูปแบบการรับส่งเป็นอย่างไรรวมถึงการศึกษาโครงสร้างในการที่จะสร้างบอร์ดอินเตอร์เฟส CAN ทั้งภาค Analog และ Digital

บทที่ 2
ลักษณะทั่วไปของ CAN

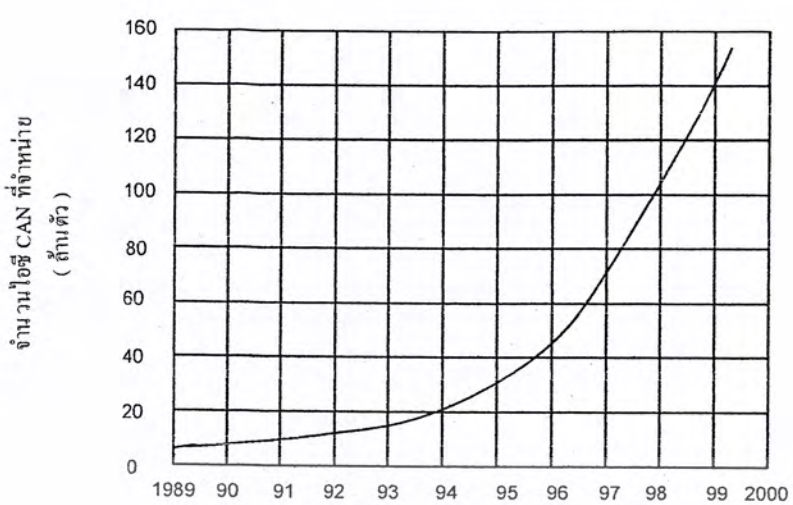
2.1 ความเป็นมาของ CAN

CAN หรือ Controller Area Network เป็นชื่อบัสสื่อสารข้อมูลแบบอนุกรมที่ออกแบบมาสำหรับใช้ในงานที่ใช้ในการควบคุมแบบเวลาจริง (Real Time) ที่มีจุดเด่นอยู่ที่ความสามารถในการสื่อสารข้อมูลที่เป็นเครือข่าย (Network) ระหว่างอุปกรณ์โดยไม่ต้องมีหมายเลขที่อยู่ของโนด (Node Addressing) อุปกรณ์ทุกตัวในเครือข่ายสามารถเรียกใช้บัสได้ (Multi–Master) และร้องขอใช้งานบัสได้พร้อมกันหลายตัว ด้วยความเร็วในการสื่อสารข้อมูลที่สูงถึง 1 เมกกะบิดต่อวินาที มีระบบป้องกันและทำการตรวจสอบความผิดพลาดที่มีประสิทธิภาพสูง

จุดเริ่มต้นของ CAN นั้นถูกพัฒนาขึ้งมาโดยบริษัท Robert Bosch ในประเทศเยอรมันเมื่อประมาณ 20 ปี ที่ผ่านมา สำหรับใช้ในระบบอิเล็คทรอนิกส์ภายในรถยนต์ เนื่องจากปัญหาในการพัฒนารถยนต์รุ่นใหม่ที่ต้องการสิ่งอำนวยความสะดวกและระบบรักษาความปลอดภัยเพิ่มขึ้น ส่งผลให้ระบบอิเล็กทรอนิกส์ภายในรถยนต์ที่มีความซับซ้อนมากยิ่งขึ้นและมีโมดูลของวงจรต่าง ๆ เป็นจำนวนมากที่ต้องมีการสื่อสารข้อมูลระหว่างกัน

ด้วยเหตุนี้จึงมีความต้องการบัสสื่อสารข้อมูลที่มีประสิทธิภาพ และมีความน่าเชื่อถือสูงในราคาที่เหมาะสม แทนที่ระบบบัสเดิมที่มีความยุ่งยากซับซ้อนไม่สะดวกที่จะนำมาใช้และมีค่าใช้จ่ายสูง ซึ่งภายหลังนอกจากจะใช้ในอุตสาหกรรมรถยนต์และยังได้มีการนำ CAN ไปประยุกต์ใช้ในงานอุตสาหกรรมอื่น ๆ อีกมากมาย เนื่องจากคุณสมบัติที่มีความโดดเด่นในด้านต่าง ๆ ของบัสนี้ในด้านความน่าเชื่อถือและความยืดหยุ่น รวมทั้งความง่ายต่อการใช้และมีค่าใช้จ่ายต่ำนั่นเอง

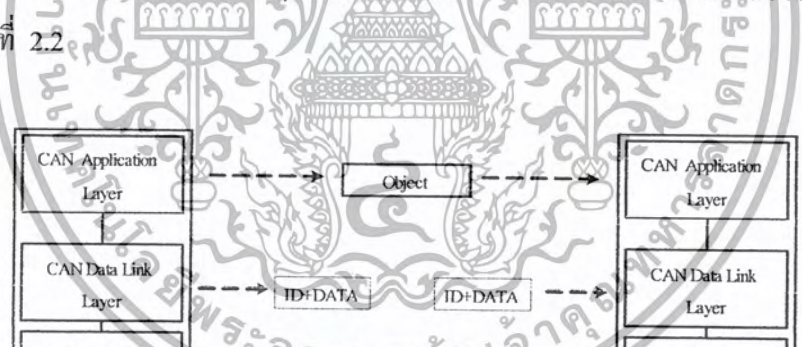
เนื่องจากความแพร่หลายในการใช้งานมีมากขึ้นเรื่อย ๆ ในภายหลังจึงได้มีการกำหนดให้ CAN เป็นบัสมาตรฐานระหว่างประเทศขึ้นมา นั่นคือ มาตรฐาน ISO 11898 (ความเร็วสูง) และ ISO 11519 (ความเร็วต่ำ) ซึ่งเป็นการรับประกันถึงความสามารถและยอมรับใน CAN ได้เป็นอย่างดี โดยมาตรฐานนี้ได้ครอบคลุมข้อกำหนดการทำงานของ CAN ในสองชั้น (Layer) แรกตามแบบจำลอง ISO/OSI ชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data Link Layer)



รูปที่ 2.1 แนวโน้มการใช้งาน ไอซี CAN

2.2 โครงสร้างการทำงาน

โครงสร้างการทำงานของ CANเมื่อเทียบกับแบบจำลอง ISO/OSI นอกจากจะประกอบด้วยสองชั้นแรกดังที่กล่าวไปแล้วยังประกอบด้วยชั้นที่ 7 คือชั้นประยุกต์ใช้งาน (Application Layer) โดยจะไม่มีส่วนประกอบของชั้นอื่น ๆ ที่เหลือคือชั้นที่ 3-6 ลักษณะของโครงสร้างในการทำงานจะเป็นดังรูปที่ 2.2



รูปที่ 2.2 โครงสร้างการทำงานของ CAN

ชั้นที่ 1 Physical Layer

โครงสร้างการทำงานในชั้นนี้ จะเป็นส่วนที่เกี่ยวข้องกับส่วนประกอบทางกายภาพของบัส ได้แก่ ตัวกลางที่ใช้ส่งผ่านสัญญาณข้อมูล คอนเน็กเตอร์และการเชื่อมต่อสายสัญญาณต่าง ๆ รวมทั้งคุณสมบัติทางไฟฟ้าของสัญญาณข้อมูล (แรงดัน / กระแส) ของสัญญาณ เป็นมาตรฐานที่เกี่ยวข้อง

กับการทำงานในชั้นนี้มีอยู่ 2 มาตรฐานคือ ISO 11898 สำหรับการสื่อสารข้อมูลความเร็วสูง สนับสนุนการสื่อสารข้อมูลด้วยความเร็วสูงสุดถึง 1 เมกะบิตต่อวินาที

ชั้นที่ 2 Data Link Layer

เกี่ยวข้องกับโปรโตรคอลและรูปแบบของข้อมูลที่ทำการสื่อสารถึงกันในเชิงตรรกะ รวมถึงเพื่อการป้องกันความผิดพลาดที่จะเกิดขึ้นกับข้อมูล เพื่อรับประกันว่าข้อมูลที่ได้มีความถูกต้องโดยข้อกำหนดของการทำงานในชั้นนี้ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วยและได้มีการแก้ไขและขยายข้อกำหนดของมาตรฐานในชั้นนี้เพิ่มเติมโดยจัดทำเป็น 2 เวอร์ชั่น คือ CAN 2.0 A (เดิม) และ CAN 2.0 B (ขยายเพิ่มเติม)

ชั้นที่ 7 Application Layer

เป็นส่วนของการนำ CAN ไปประยุกต์ใช้งานบนพื้นฐานการทำงานในสองชั้นแรก แม้ว่าการทำงานในชั้นนี้จะไม่ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วยแต่ได้มีการสร้างโปรโตคอลสำหรับการทำงานในส่วนของชั้นนี้ขึ้นมา ซึ่งมีอยู่ด้วยกันหลายรูปแบบ ตัวอย่างโปรโตคอลในชั้นนี้ที่มีการใช้งานกัน เช่น CAN open, Device Net, CAN Kingdom, OSEK / VDK, SDS และ J1939 เป็นต้น

2.3 คุณสมบัติของ CAN

- มาตรฐาน ISO 11898 พัฒนาขึ้นมาโดย Robert Bosch GmbH
- โครงสร้างการทำงานนั้นมี 2 ชั้น คือชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data Link Layer)
- สื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส
- สร้างเครือข่ายแบบ Multi – master / Broadcasting
- สนับสนุนการทำงานแบบเรียลไทม์
- ใช้หลักการ CSMA / CD ในการเข้าใช้บัส
- ไม่มีการกำหนดหมายเลขที่อยู่กำกับโนด
- ส่งข้อมูลครั้งละ 0 – 8 ไบต์ ด้วยอัตราเร็วสูงสุด 1 เมกะบิต ต่อ วินาที
- มีความปลอดภัยในการรับส่งข้อมูลสูง
- ตัวควบคุมโปรโตคอลสามารถบรรจุอยู่ในชิปเดียว
- ต้นทุนต่ำ ใช้งานสะดวก บำรุงรักษาง่าย
- มีการใช้งานอย่างแพร่หลายในงานอุตสาหกรรมและระบบอิเล็กทรอนิกส์ในยานยนต์

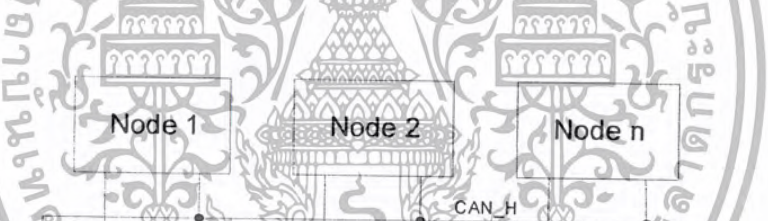
2.4 ลักษณะทางกายภาพ

ลักษณะทางกายภาพของ CAN นั้น เริ่มจากตัวกลางที่ใช้ส่งผ่านข้อมูลและลักษณะของสัญญาณที่ส่งออกไป

ตัวกลางที่ใช้ในการส่งผ่านข้อมูลในระบบ CAN นั้น มีอยู่ด้วยกันหลายรูปแบบตั้งแต่การใช้สายสัญญาณ 2 เส้น การใช้สายสัญญาณเส้นเดียว การใช้สายไฟเบอร์ออปติกหรือแม้แต่การส่งสัญญาณแบบไร้สาย แต่ที่นิยมใช้กันมากคือ การใช้สายสัญญาณ 2 เส้นแบบตีเกลียว (Twisted pair Cable) ซึ่งในที่นี้จะอ้างอิงการส่งผ่านข้อมูลโดยใช้ตัวกลางในรูปแบบนี้

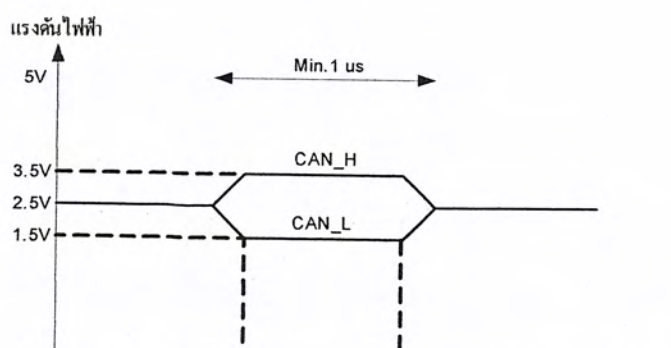
การส่งผ่านข้อมูลใน CAN มีรูปแบบของการส่งข้อมูลเป็นแบบอนุกรมที่ใช้สายสัญญาณเพียง 2 เส้น ซึ่งมีลักษณะการเชื่อมต่อสายสัญญาณ (Topology) ดังรูปที่ 2.3 โดยอุปกรณ์ทุกตัวจะต่ออยู่บนสายสัญญาณคู่เดียวกันและปิดปลายคู่สายสัญญาณทั้งสองข้างด้วยเทอร์มิเนชันอิมพีเดนซ์ (Termination Impedance)

สัญญาณข้อมูลที่ส่งในระบบ CAN ใช้วิธีการส่งแบบสัญญาณแรงดันผลต่าง (Differential Voltage Signal) สถานะของสัญญาณในสายส่งได้จากการเปรียบเทียบระดับแรงดันหรือศักย์ไฟฟ้าระหว่างสายสัญญาณทั้งสองเส้น คือ CAN_H (ศักย์สูง) CAN_L (ศักย์ต่ำ)



รูปที่ 2.3 ลักษณะการเชื่อมต่ออุปกรณ์ (Topology) ใน CAN

สถานะของสัญญาณข้อมูลในสายส่งของระบบ CAN จะมีสองสถานะคือสถานะรีเซสสีฟ (Recessive) และโดมิแนนต์ (Dominant) ระดับแรงดันไฟฟ้าของสัญญาณในแต่ละเส้น (เทียบกับกราวด์) และผลต่างแรงดันไฟฟ้าระหว่างสายคู่สัญญาณในแต่ละสถานะจะเป็นดังรูปที่ 2.4



รูปที่ 2.4 ลักษณะของสัญญาณข้อมูลแต่ละสถานะ

ถ้าผลต่างแรงดันระหว่าง CAN_H และ CAN_L เป็น 0 โวลต์ จะเป็นสถานะรีเซสสีฟซึ่งจะแทนข้อมูลที่มีลอจิกเป็น "1"

ถ้าผลต่างของแรงดันระหว่าง CAN_H และ CAN_L เป็น 2 โวลต์ จะเป็นสถานะโดมิแนนต์ซึ่งจะเป็นข้อมูลที่มีลอจิกเป็น "0"

การส่งผ่านข้อมูลโดยใช้ผลต่างแรงดันของคู่สัญญาณนั้นมีข้อดีคือ จะช่วยลดผลการรบกวนอันเนื่องมาจากผลรวมของ EMI ลงได้เป็นอย่างมากทำให้สามารถส่งสัญญาณได้ในอัตราเร็วที่สูงขึ้นและได้ในระยะทางที่ไกลมากขึ้น

อัตราเร็วในการส่งข้อมูลและระยะทางที่ส่งได้จะแปรผกผันกันดังกราฟในรูปที่ 2.5 ตามมาตรฐาน ISO 11898

รูปที่ 2.5 กราฟความสัมพันธ์ระหว่างอัตราเร็วของข้อมูลกับระยะทางที่ส่งได้

ตารางที่ 2.1 ความสัมพันธ์ของคุณสมบัติสายส่งกับอัตราเร็วข้อมูล

ความยาวสายส่ง	คุณสมบัติสายส่ง		เทอร์มิเนชันอิมพีแดนซ์	อัตราความเร็วข้อมูลสูงสุด
	ความต้านทาน	ขนาด		
0 – 40 m	70 m Ω/ m	0.25 – 0.34 mm^2 AWG23, AWG22	124Ω (1%)	1 Mbit/s ที่ 40 m
40 – 300 m	< 60 mΩ/ m	0.34 – 0.6 mm^2 AWG22, AWG20	127Ω (1%)	500 Kbit/s ที่ 100 m
300 – 600 m	< 40 mΩ/ m	0.5 – 0.6 mm^2 AWG20,	150Ω (1%) 300 Ω	1 Kbit/s ที่ 500 m
600 – 1 km	< 26 mΩ/ m	0.75 – 0.8 mm^2 AWG18	150Ω(1%) 300 Ω	50 Kbit/s ที่ 1 km

อัตราเร็วในการส่งข้อมูลสูงสุดคือ 1 เมกะบิตต่อวินาที คิดที่ความยาวของสายส่งไม่เกิน 40 เมตร สำหรับค่าความต้านทานและขนาดของสายส่ง รวมทั้งขนาดของเทอร์มิเนชันอิมพีแดนซ์ที่แนะนำให้ใช้ดูได้จากตารางที่ 2.1 ตัวอย่างเช่น ที่อัตราเร็วข้อมูล 1 เมกะบิตต่อวินาที ควรใช้สายส่งที่มีความต้านทานขนาด 70 มิลลิโอห์มต่อเมตร และมีขนาดของเส้นผ่าศูนย์กลางสายส่งมากกว่า 0.25 ตารางมิลลิเมตร โดยต้องต่อเทอร์มิเนชันอิมพีแดนซ์ที่มีค่า 124 โอห์ม ไว้ที่ปลายทั้งสองด้านของสายส่ง

ข้อควรระวังในการต่อสายสัญญาณก็คือ ในกรณีที่มีการต่อสัญญาณพ่วงจากบัสกลางสายพ่วงควรมีความยาวไม่เกิน 2 เมตร เมื่อมีอัตราเร็วในการส่งข้อมูลเป็น 250 กิโลบิตต่อวินาที และไม่ควรเกิน 30 เซนติเมตร ถ้าอัตราเร็วของข้อมูลสูงกว่านั้น ทั้งนี้ความยาวของสายพ่วงทุกรวมกันไม่ควรเกิน 30 เมตร

บทที่ 3
โปรโตคอลในการส่งข้อมูลของ CAN

3.1 หลักการเบื้องต้น

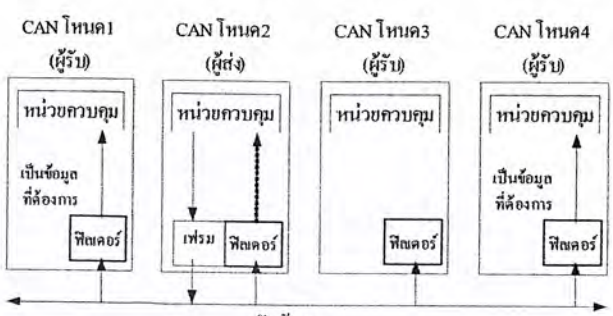
ในงานควบคุมอัตโนมัติในอุตสาหกรรมเพื่อให้ระบบ CAN มีความน่าเชื่อถือ ปลอดภัย มีประสิทธิภาพในการทำงานมากยิ่งขึ้น ช่วยลดความยุ่งยากซับซ้อนจากจำนวนสายสัญญาณในการเชื่อมต่อระหว่างอุปกรณ์ในส่วนต่าง ๆ ลง ได้กำหนดเป็นมาตรฐาน ISO 11898 (CAN ความเร็วสูง) และ ISO 11519 (CAN ความเร็วต่ำ) ขึ้นมา

รูปแบบการสื่อสารข้อมูลของ CAN เป็นแบบอนุกรมอะซิงโครนัส โดยที่อุปกรณ์แต่ละตัวหรือแต่ละโนด (Node) สามารถส่งข้อมูลหรือข้อความ (Message) ไปยังโนดอื่น ๆ ได้ เหตุที่เรียกว่าเป็นการสื่อสารข้อมูลแบบอะซิงโครนัส เนื่องจากไม่มีการส่งสัญญาณนาฬิกาเพื่อทำการเข้าจังหวะ แต่จะอาศัยการเข้าจังหวะจากจุดเริ่มต้นของข้อความที่ส่งไป (มีลักษณะเดียวกับการส่งข้อมูลแบบ UART ใน RS232)

ในส่วนของโปรโตคอลในการสื่อสารข้อมูลในชั้นเชื่อมโยงข้อมูลของ CAN นั้นมีหลักการสำคัญที่ทำให้ CAN เป็นบัสข้อมูลที่ทำงานได้อย่างมีประสิทธิภาพ ได้แก่การสื่อสารแบบบรอดคาส (Broadcast Communication) การเข้าใช้บัสจากหลายโนด (Multiple Bus Access) การสื่อสารที่อิงจากข้อความ (Message-based Communication) และระบบการป้องกันความผิดพลาด (Error Protetion)

3.1.1 การสื่อสารแบบบรอดคาส

การส่งข้อมูลใน CAN จะใช้หลักการส่งข้อมูลแบบบรอดคาส คือทุกโนดที่อยู่บนบัสสามารถรับข้อมูลที่ส่งมาจากโนดผู้ส่งได้ หลังจากที่ได้รับข้อมูลมาแล้วจะเป็นหน้าที่ของแต่ละโนดที่จะต้องตรวจสอบเองว่าเป็นข้อมูลที่ต้องการหรือไม่ ซึ่งการตรวจสอบตรงจุดนี้จะอาศัยหลักการของแอกเซปแตนซ์ฟิลเตอร์ (Acceptance Filter) ทำหน้าที่สกัดกั้นข้อมูลที่ไม่ต้องการทิ้งไป และยอมรับเฉพาะข้อมูลที่ต้องการ โดยดูจากค่ารหัสประจำตัว (Identifier) ของข้อมูลหรือข้อความนั้น ลักษณะการส่งข้อมูลแบบบรอดคาสจะเป็นดังรูปที่ 3.1



รูปที่ 3.1 ลักษณะการสื่อสารแบบบรอดคาสใน CAN

รูปแบบการส่งข้อมูลแบบบรอดคาสใน CAN นี้ อาจจะเปรียบเทียบได้กับการส่งข่าวสารจากสถานีวิทยุ เช่น ข้อมูลการจราจรผู้ฟังหรือผู้ใช้รถจะตัดสินใจเองว่าข้อมูลที่ได้รับมีความสำคัญกับตนหรือไม่ โดยจะดูจากเส้นทางที่ต้องการใช้เป็นหลัก ซึ่งในที่นี้ค่ารหัสประจำตัวของข้อมูลก็คือชื่อของถนนที่บอกข้อมูลในการจราจร ส่วนแอกเซปแตนซ์ฟิลเตอร์ก็คือเส้นทางที่ต้องการใช้เพื่อให้ไปถึงเป้าหมายนั้นเอง

การสื่อสารข้อมูลใน CAN นอกจากจะสื่อสารข้อมูลแบบบรอดคาสธรรมดาคือแต่ละโนดจะรอรับข้อมูลจากโนดที่ต้องการส่งแล้ว ยังสามารถร้องขอข้อมูล (Remote Request) จากโนดที่มีข้อมูลดังกล่าวได้อีกด้วย โดยการส่งค่ารหัสประจำตัวของข้อมูลที่ต้องการจะทราบออกไป หากโนดใดมีค่ามูลดังกล่าวก็จะส่งข้อมูลนั้นตอบกลับมา ซึ่งนอกจากโนดที่ร้องขอข้อมูลนี้ไปแล้วโนดอื่น ๆ ที่เหลือก็จะสามารถรับส่งข้อมูลดังกล่าวได้อย่างถูกต้อง (เนื่องจากเป็นการส่งข้อมูลแบบบรอดคาสนั่นเอง)

3.1.2 การเข้าใช้บัสจากหลายโนด

จุดเด่นการส่งโปรโตคอลการส่งข้อมูลที่ใช้ใน CAN อย่างหนึ่งก็คือการยอมให้มีการเข้าใช้บัสข้อมูลจากหลายโนดได้พร้อม ๆ กัน ในกรณีที่มีโนดมากกว่าหนึ่งโนดต้องการส่งข้อมูลในเวลาเดียวกัน จะมีการตัดสินชี้ขาดว่าโนดใดจึงจะมีสิทธิที่จะเข้าใช้บัสข้อมูลในเวลานั้นเพียงโนดเดียวในขณะที่โนดอื่นจะต้องหยุดทำการส่งและรอที่จะส่งข้อมูลในภายหลังเมื่อบัสว่างไม่ได้ถูกใช้งานแล้ว

การเข้าใช้บัสใน CAN จะใช้วิธีการที่เรียกว่า Carrier Sense Multiple Access แบบ Collision Cetection (CSMA/CA) กล่าวคือ เมื่อโนดใดต้องการใช้บัส จะต้องทำการตรวจสอบจนพบว่าบัสว่างไม่ได้ถูกใช้งานอยู่เป็นระยะเวลาหนึ่งจึงเริ่มทำการส่งข้อมูลออกไป (CS) ซึ่งขณะที่บัส ว่างอยู่นี้โนดแต่ละโนดมีสิทธิที่จะเข้าใช้บัสโอกาสที่เท่ากัน (MA) แต่ถ้ามีการส่งข้อมูลออกมาพร้อม ๆ กันโนดจะทราบได้ว่ามีการชนกันของข้อมูลเกิดขึ้น (CD) และดำเนินการแก้ไขการชนของข้อมูลที่เกิดขึ้นต่อไป

เมื่อมีการเข้าใช้บัสพร้อมกัน วิธีการที่นำมาใช้ใน CAN เพื่อตัดสินว่าโนดใดจะมีสิทธิได้เข้าใช้บัสจะอาศัยค่าลำดับความสำคัญข้อมูลเป็นตัวตัดสิน (Arbitration on Message Priority : AMI) ซึ่งค่าลำดับความสำคัญ (Priority) ของข้อมูลซึ่งจะถูกระบุอยู่ในค่ารหัสประจำตัว (Identifier) ของข้อมูลนั้น โดยการตัดสินชี้ขาดจะเกิดขึ้นโดยไม่ทำลายโอกาสของการใช้บัสของข้อมูลที่มีค่าลำดับความสำคัญสูงสุด (Non-destructive Arbitration) และการตัดสินจะเกิดขึ้นทันทีในระดับบิตข้อมูล (Bitwise Arbitration)

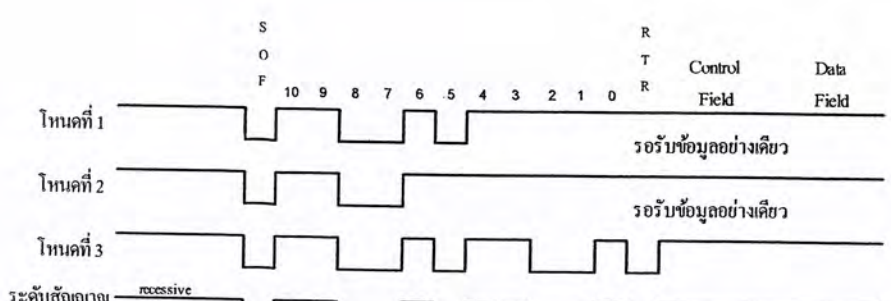
กระบวนการเช่นนี้จะเกิดขึ้นได้ต้องอาศัยคุณสมบัติที่สำคัญ 2 ประการ คือ

- สถานะของสัญญาณข้อมูลที่เป็นแบบโดมิแนนต์ (Dominant) และรีเซสสีฟ (Recessive) โดย CAN กำหนดให้สถานะโดมิแนนต์แทนลอจิก "0" และสถานะรีเวสสีฟแทนลอจิก "1" ในการใช้งานเมื่อสัญญาณทั้งสองสถานะถูกส่งมาที่บัสพร้อมกัน บัสจะมีสถานะโดมิแนนต์ (คืออ่านค่าได้เป็นลอจิก "0")

- โนดผู้ส่งต้องสามารถตรวจสอบสถานะของบัสอยู่ตลอดเวลา ว่าข้อมูลที่อยู่ในบัสตรงกับข้อมูลที่ส่งออกไปหรือไม่ หากไม่ตรงกันแสดงว่าเกิดการชนกันของข้อมูลหรือเกิดความผิดพลาดในการส่งข้อมูลเกิดขึ้น

รูปที่ 3.2 โครงสร้างอย่างง่ายของวงจรที่แต่ละโนดใช้เชื่อมต่อเข้ากับบัส CAN

โดยลักษณะของวงจรที่ใช้เชื่อมต่อเข้ากับบัส CAN จะแสดงวงจรไว้ดังรูปที่ 3.2 ในการส่งข้อมูลแต่ละโนดจะเริ่มส่งค่ารหัสประจำตัวของข้อมูลในแต่ละตัวออกมาก่อน ในขณะเดียวกันนั้นก็จะคอยตรวจสอบว่าค่าที่ได้รับจากบัสมีค่าตรงกับที่ส่งออกไปหรือไม่ ในกรณีที่มีการรับส่งข้อมูลพร้อมกันมากกว่า 1 โนด ดังตัวอย่างในรูปที่ 3.3 มีการส่งข้อมูลพร้อมกัน 3 โนด ขณะที่บิตข้อมูล (ของค่ารหัสประจำตัว) ที่แต่ละโนดสามารถจะส่งออกมามีค่าตรงกันอยู่ การส่งข้อมูลในแต่ละโนดก็จะดำเนินการต่อไปตามปกติ



รูปที่ 3.3 ตัวอย่างของสัญญาณในบัส CAN กรณีที่มีโนดส่งข้อมูลออกมาพร้อมกัน (3 โนด)

จนกระทั่งเมื่อค่าที่ส่งออกมาไม่ตรงกัน เช่น ในบิตที่ 5 โนดที่ 1 และโนดที่ 3 ส่งสัญญาณสถานะโดมิแนนต์ (ลอจิก "0") ในขณะที่โนดที่ 2 ส่งสัญญาณสถานะรีเซสสีฟ (ลอจิก "1") แต่อ่านค่าจากบัสกลับมาได้เป็นโดมิแนนต์ ซึ่งทำให้โนดที่ 2 หมดสิทธิในการส่งข้อมูลในรอบนี้ไปและเปลี่ยนมาเป็นโนดผู้รับแทน เช่นเดียวกับในบิตที่ 2 โนดที่ 1 ส่งสัญญาณสถานะรีเซสสีฟแต่อ่านค่ากลับมาได้เป็นโดมิแนนต์ทำให้โนดที่ 1 ต้องหยุดส่งข้อมูลไป สุดท้ายจึงเหลือเพียงแต่โนดที่ 3 เพียงโนดเดียวที่ส่งข้อมูลได้ในรอบนี้

จากตัวอย่างนี้กล่าวได้ว่าข้อมูลจากที่ส่งโดยโนดที่ 3 ที่มีค่ารหัสประจำตัวต่ำกว่าจะมีค่าระดับความสำคัญสูงกว่าข้อมูลจากโนดที่ 1 และ 2 ซึ่งจะมีสิทธิส่งข้อมูลได้ก่อน และไม่ต้องเริ่มทำการส่งข้อมูลใหม่เมื่อมีการส่งข้อมูลชนกับโนดอื่น ในขณะที่โนดที่ 1 และ 2 ซึ่งมีค่าระดับความสำคัญต่ำกว่าจะเริ่มทำการส่งข้อมูลได้ใหม่หลังจากที่โนดที่ 3 ส่งข้อมูลเรียบร้อยแล้ว

3.1.3 การสื่อสารที่อิงจากข้อความ

การส่งข้อมูลใน CAN จะไม่มีการระบุที่อยู่ของโนดผู้รับและโนดผู้ส่งเหมือนอย่างที่ใช้กันอยู่ทั่วไป แต่จะใช้การระบุด้วยค่าประจำตัว (Identifier) ของข้อมูลหรือข้อความแทนเพื่อเป็นการบ่งบอกถึงชนิดและระดับความสำคัญของข้อมูลนั้น โดยอาศัยวิธีการส่งข้อมูลแบบบรอดคาสดังเช่นที่กล่าวมาแล้วข้างต้น คือเมื่อมีโนดใดโนดหนึ่งซึ่งทำการส่งข้อมูลออกมาที่บัส แต่ละโนดจะได้รับการแจ้งว่ามีการส่งข้อมูลเกิดขึ้นและจะตัดสินใจกันเองว่าจะรับข้อมูลที่ส่งมานั้นหรือไม่

ตัวอย่างเช่น โนด A ทำการส่งข้อมูล (สมมติว่าเป็นค่าที่วัดได้จากเซนเซอร์) ด้วยค่ารหัสประจำตัวของข้อมูล (สมมติว่าเท่ากับ 123) ออกไปที่บัส หากโนดใดต้องการใช้ข้อมูลดังกล่าวก็จะทำการรับข้อมูลนี้ไว้ จะเห็นว่าการส่งข้อมูลด้วยวิธีนี้ไม่จำเป็นต้องมีการระบุหมายเลขที่อยู่หรือรหัสประจำตัวของโนดทั้งโนดผู้รับและโนดผู้ส่ง ซึ่งมีข้อดีคือ สามารถทำการเพิ่มโนดเข้ามาในระบบได้โดยไม่จำเป็นต้องทำการโปรแกรมทุกโนดใหม่ เพื่อให้รับรู้ว่ามีการเพิ่มโนดเข้ามา ในขณะ

เดียวกันโนดที่เพิ่มเข้ามาใหม่นี้ก็จะสามารถรับข้อมูลที่มีการส่งกันในบัสได้โดยทันที โดยพิจารณาจากค่ารหัสประจำตัวของข้อมูลที่ส่งมานั้นเอง

นอกจากการรอรับข้อมูลที่โนดอื่นส่งมาให้แล้ว (ซึ่งบางทีอาจต้องรอเป็นเวลานานกว่าที่จะได้รับข้อมูลที่ต้องการ) แต่ละโนดยังสามารถเป็นฝ่ายร้องขอข้อมูลจากโนดอื่นได้ด้วย เพื่อให้โนดที่มีข้อมูลดังกล่าวส่งข้อมูลนั้นออกมาให้โดยทันที โดยเรียกรูปแบบการร้องขอข้อมูลนี้ว่า Remote Transmission ๗equest (RTR) มีข้อดีคือข้อมูลบางอย่างที่ไม่ได้มีการใช้งานเป็นประจำ ก็ไม่จำเป็นต้องส่งออกมาอยู่ตลอดเวลาโดยรอให้มีการร้องขอข้อมูลนั้นมาก่อนจึงค่อยส่งข้อมูลนั้นออกไปซึ่งจะช่วยลดปริมาณการใช้บัสลงได้

3.1.4 การป้องกันความผิดพลาด

เนื่องจากแรกเริ่มเดิมที่นั้น CAN ถูกออกแบบมาเพื่อใช้ในระบบอิเล็กทรอนิกส์ในรถยนต์ ซึ่งปัญหาที่พบส่วนใหญ่ก็คือเรื่องสัญญาณรบกวน ดังนั้นการออกแบบโปรโตคอลเพื่อให้สามารถใช้ส่งข้อมูลได้อย่างมีประสิทธิภาพและถูกต้อง จะต้องสามารถส่งข้อมูลได้อย่างไม่มีข้อผิดพลาดในขณะที่ความเร็วในการส่งข้อมูลต้องเป็นที่ยอมรับหรือรองรับการใช้งานได้อย่างกว้างขวาง

ทั้งนี้นอกจาก CAN จะสามารถส่งข้อมูลได้ที่ความเร็วสูงถึง 1 เมกบิตต่อวินาที (ในเวอร์ชัน CAN 2.0) จุดเด่นที่สำคัญอีกอย่างหนึ่งก็คือ การตรวจสอบความผิดพลาดของข้อมูลเพื่อให้แน่ใจได้ว่าข้อมูลที่ได้มีความถูกต้องสมบูรณ์ไม่มีความผิดพลาดเกิดขึ้น โดยการสื่อสารข้อมูลใน CAN มีวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้นหลายแบบ ซึ่งจะขอกล่าวถึงรายละเอียดในภายหลัง

3.2 โปรโตคอลการส่งข้อมูล

โปรโตคอลการส่งข้อมูลใน CAN จะแบ่งข้อมูลที่ส่งออกเป็นแพ็กเก็ต (Packets) หรือเฟรม (Frames) ซึ่งมีใช้อยู่ด้วยกัน 4 ชนิด ได้แก่ เฟรมข้อมูล (Data Frame) ใช้ในการส่งข้อมูลตามปกติ เฟรมร้องขอข้อมูล (Remote Frame) ใช้ในการร้องขอข้อมูลจากโนดอื่น เฟรมแสดงความผิดพลาด (Error Frame) ใช้แสดงว่ามีความผิดพลาดเกิดขึ้น และเฟรมโอเวอร์โหลด (Overload Frame) ใช้เพื่อบอกว่าต้องการเวลาในการประมวลผลข้อมูลที่ได้รับเพิ่มขึ้น

ในที่นี้จะขอกล่าวถึงรายละเอียดเฉพาะส่วนเฟรมข้อมูลและเฟรมร้องขอข้อมูลเท่านั้น ส่วนรายละเอียดของเฟรมแสดงความผิดพลาดและเฟรมโอเวอร์โหลดจะไม่ขอกล่าวถึง โดยสามารถหาอ่านเพิ่มเติมได้จากข้อมูลอ้างอิงและเพิ่มเติมในตอนท้าย

3.2.1 เฟรมข้อมูล

ในเฟรมข้อมูล 1 เฟรมจะประกอบด้วยส่วนย่อย ๆ ที่เรียกว่าฟิลด์ (Fields) ดังในรูปที่ 3.4 และรูปที่ 3.5 ซึ่งแสดงส่วนประกอบของเฟรมข้อมูลแบบมาตรฐาน (Standard Data Frame) และข้อมูลแบบขยาย (Extended Data Frame) ตามลำดับเฟรมข้อมูลทั้งสองแบบจะมีข้อแตกต่างกันที่ขนาดของรหัสประจำตัวของข้อมูล (Identifier หรือ ID)

ในตอนแรกนี้จะขอกล่าวถึงส่วนประกอบต่าง ๆ ในเฟรมข้อมูลแบบมาตรฐานก่อน ส่วนรายละเอียดในข้อแตกต่างที่มีในเฟรมข้อมูลแบบขยายจะขอกล่าวถึงในส่วนถัดไป

เฟรมข้อมูลแบบมาตรฐาน (CAN2.0A) ส่วนประกอบต่าง ๆ ในเฟรมข้อมูลแบบมาตรฐานมีดังนี้

- บิตเริ่มต้นเฟรม (Start of Frame : SOF) มีสถานะเป็นโดมิแนนต์หรือลอจิก "0" ซึ่งเป็นจุดที่ทุกโนดในบัสใช้ในการเริ่มต้นเข้าจังหวะสำหรับการรับ-ส่งข้อมูลระหว่างกัน

- ฟิลด์แสดงรหัสข้อมูล (Arbitration Field) มีขนาด 12 บิต ประกอบด้วยหมายเลข ID (Identifier) ขนาด 11 บิต (และมีขนาด 29 บิต สำหรับเฟรมข้อมูลแบบขยาย) และบิต RTR (Remote Transmission Request) 1 บิต

ตารางที่ 3.1 ค่าของบิตต่าง ๆ ที่ใช้ในเฟรม (* = มีเฉพาะในเฟรมแบบขยาย)

ชื่อบิต	ค่าที่ใช้	ค่าที่ใช้
	'0' หรือโดมิแนนต์	'1' หรือรีเซสสีฟ
SOF	/	
RTR	เฟรมข้อมูล	เฟรมร้องขอข้อมูล
IDE	เฟรมแบบมาตรฐาน	เฟรมแบบขยาย
RBO	/	
CRC Del		/
ACK Del		/
SRR *		/
RBI *	/	

ตารางที่ 3.2 ความสัมพันธ์ของค่า Data Length Code (DLC) กับจำนวนข้อมูลในเฟรมนั้น (d = โดมิแนนต์ หรือ หรือ '0', r = รีเซสสีฟ หรือ '1')

จำนวนข้อมูล (ไบต์)	Data Length Code (DLC)			
	DLC 3	DLC 2	DLC 1	DLC 0
0	d	D	D	d
1	d	D	D	r
2	d	D	R	d
3	d	D	R	r
4	d	R	D	d
5	d	R	D	r
6	d	R	R	d
7	d	R	R	r
8	r	D/r	d/r	d/r

การส่งข้อมูลโดยใช้เฟรมข้อมูลแบบมาตรฐานซึ่งจะมีหมายเลข ID ขนาด 11 บิต และจะมีหมายเลข ID ที่สามารถใช้งานได้จำนวน 2^{11} = 2,048 หมายเลขที่แตกต่างกัน โดยจะมีการสำรองไว้ 16 หมายเลขสำหรับใช้งานในฟังก์ชันพิเศษเฉพาะอย่าง จึงมีเหลือให้ใช้งานได้ 2,032 หมายเลขซึ่งหมายถึงจำนวนข้อมูลที่สามารถรองรับได้นั่นเอง

บิต RTR ซึ่งอยู่ถัดจากหมายเลข ID เป็นบิตที่ใช้แสดงชนิดของเฟรมข้อมูลโดยถ้าบิต RTR เท่ากับ "0" หรือโดมิแนนต์แสดงว่าเป็นเฟรมข้อมูล แต่ถ้าเป็นบิต RTR = "1" รีเซสสีฟ แสดงว่าเป็นเฟรมขอข้อมูล (ซึ่งจะขอกล่าวถึงเฟรมร้องขอข้อมูลในภายหลัง) ซึ่งในที่นี้เป็นเฟรมข้อมูลดังนั้นบิต RTR นี้จึงมีค่าเป็น "0" หรือ โดมิแนนต์

- ฟิลด์ควบคุม (Control Field) มีขนาด 6 บิต ประกอบด้วยบิต IDE (Identifier Ectension) 1 บิต , บิต RBO (Reserved Bit 0) 1 บิต และ DLC (Data Length Code) จำนวน 4 บิต

บิต IDE ใช้ในการบ่งบอกว่าเฟรมข้อมูลนี้เป็นเฟรมข้อมูลแบบใด โดยถ้าบิต IDE = '0' หรือโดมิแนนต์จะเป็นเฟรมข้อมูลแบบมาตรฐาน (ID = 11 บิต) ถ้าบิต IDE = '1' หรือรีเซสสีฟจะเป็นเฟรมข้อมูลแบบขยาย (ID = 29 บิต) สำหรับในที่นี้เป็นเฟรมข้อมูลแบบมาตรฐาน ดังนั้น IDE นี้จะมีค่าเป็น '0' หรือ โดมิแนนต์

บิต RBO เป็นบิตที่สำรองไว้สำหรับการใช้งานในอนาคต ในการงานจะให้บิตนี้เป็น '0' หรือโดมิแนนต์

ส่วนสุดท้ายในฟิลด์ควบคุมคือ DLC มีขนาด 4 บิต ใช้บอกขนาดข้อมูลที่ถูกส่งมาโดยข้อกำหนดของ CAN กำหนดให้ข้อมูลที่ส่งมาในแต่ละเฟรมมีขนาดได้ตั้งแต่ 0-8 ไบต์

- ฟิลด์ข้อมูล (Data Field) เป็นส่วนของข้อมูลที่ต้องการส่ง มีขนาดไม่เกิน 8 ไบต์

- ฟิลด์ตรวจสอบ (CRC Field) มีขนาด 16 บิต เป็นส่วนที่ใช้ในการตรวจสอบความถูกต้องของข้อมูลที่ส่งในเฟรมนั้น ประกอบด้วยค่า CRC (Cyclic Redundancy Check) ขนาด 15 บิต และ บิต CRC Delimiter 1 บิต

การส่งข้อมูลในแต่ละเฟรมจะมีการคำนวณค่า CRC เริ่มตั้งแต่บิตเริ่มต้นเฟรมไปจนถึงฟิลด์ข้อมูล โดยเราสามารถตรวจพบความผิดพลาดที่เกิดขึ้นในช่วงดังกล่าวได้ หากความผิดพลาดเกิดขึ้นไม่เกิน 5 บิต

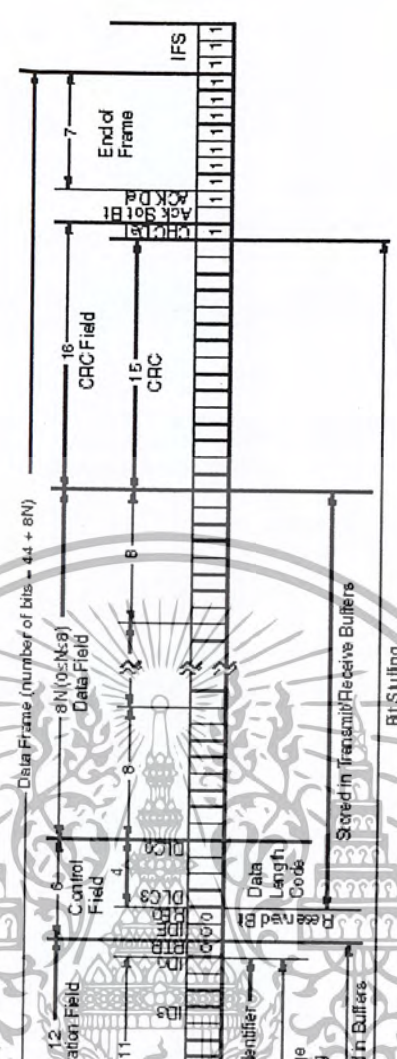
เมื่อส่งค่า CRC ครบทั้ง 15 บิต แล้วจะปิดท้ายฟิลด์ตรวจสอบนี้ด้วยบิต CRC Delimiter โดยจะส่งเป็นค่า '1' หรือรีเซสสีฟ่

- ฟิลด์ตอบสนอง (Aclnowledge Field) มีขนาด 2 บิต เป็นส่วนที่โนดผู้รับใช้ตอบกลับไปยังโนดผู้ส่งว่าข้อมูลที่ส่งมาได้รับถูกต้องหรือไม่ประกอบด้วย ACK Slot 1 บิต และ ACK Delimiter 1 บิต

โนดผู้ส่งจะส่งบิต ACK Slot นี้เป็น '1' หรือรีเซสสีฟออกไป ในขณะที่โนดผู้รับตรวจสอบแล้วว่าข้อมูลที่ได้รับถูกต้อง ก็จะส่งบิต ACK Slot เป็น '0' หรือโดมิแนนต์ตอบกลับมา ซึ่งจะทำให้สถานะของบัสเป็นโดมิแนนต์ ดังนั้นเมื่อโนดผู้ส่งอ่อนค่าบิต ACK Slot กลับมาได้เป็น '0' หรือโดมิแนนต์แสดงว่ามีอย่างน้อย 1 โนดที่ได้รับข้อมูลได้อย่างถูกต้อง

ปิดท้ายฟิลด์ตอบสนองด้วยบิต ACK Delimiter ซึ่งจะมีค่าเป็น "1" หรือ รีเซสสีฟเสมอ

- ฟิลด์สิ้นสุดเฟรม (End of Frame: EOF Field) ประกอบด้วยบิต "1" หรือรีเซสสีฟ จำนวน 7 บิตติดต่อกันหรือมีค่าเท่ากับ "1111111" เป็นส่วนที่ใช้แสดงว่าได้สิ้นสุดเฟรมนี้แล้ว



รูปที่ 3.4 ส่วนประกอบในเฟรมข้อมูลแบบมาตรฐาน

และเพื่อให้เวลากับโนดผู้รับ สำหรับการประมวลผลหรือจัดการกับข้อมูลที่ได้รับมาให้เป็นที่เรียบร้อยเสียก่อน ดังนั้นหนังจากที่สิ้นสุดเฟรมแล้ว จึงกำหนดว่าจะต้องส่งสภาวะเป็น "1" หรือรีเซสสีฟต่อท้ายฟิลด์สิ้นสุดเฟรมอย่างน้อย 3 บิต จึงจะสามารถส่งเฟรมข้อมูลใหม่ได้ต่อไป

เฟรมข้อมูลแบบขยาย (CAN 2.0B) ส่วนประกอบต่าง ๆ ในเฟรมข้อมูลแบบขยายนั้นจะต่างจากเฟรมข้อมูลแบบมาตรฐานตรงที่ฟิลด์แสดงรหัสข้อมูลและฟิลด์ควบคุม ส่วนฟิลด์ที่เหลือนั้นจะเหมือนกัน สำหรับรายละเอียดในส่วนที่แตกต่างมีดังนี้

- ฟิลด์แสดงรหัสข้อมูล จะมีขนาด 32 บิต โดยแบ่งเป็นหมายเลข ID (Identifier) ขนาด 29 บิต (ในขณะที่เฟรมข้อมูลแบบมาตรฐานจะมีหมายเลข ID ขนาด 11 บิต) บิต SRR บิต IDE และบิต RTR

หมายเลข ID 29 บิตนี้จะแบ่งเป็น 2 ส่วนคือ ส่วนแรกมีขนาด 11 บิตอยู่ถัดจากบิตเริ่มต้นเฟรมเช่นเดียวกับหมายเลข ID ในเฟรมข้อมูลมาตรฐาน และส่วนที่สองมีขนาด 18 บิตอยู่ถัดจากบิต IDE เหตุที่ต้องแบ่งหมายเลข ID ออกเป็น 2 ส่วน โดยส่วนแรกมีขนาด 11 บิตเช่นนี้ก็เพื่อให้สอดคล้องกับหมายเลข ID ที่ใช้ในเฟรมข้อมูลแบบมาตรฐานนั่นเอง

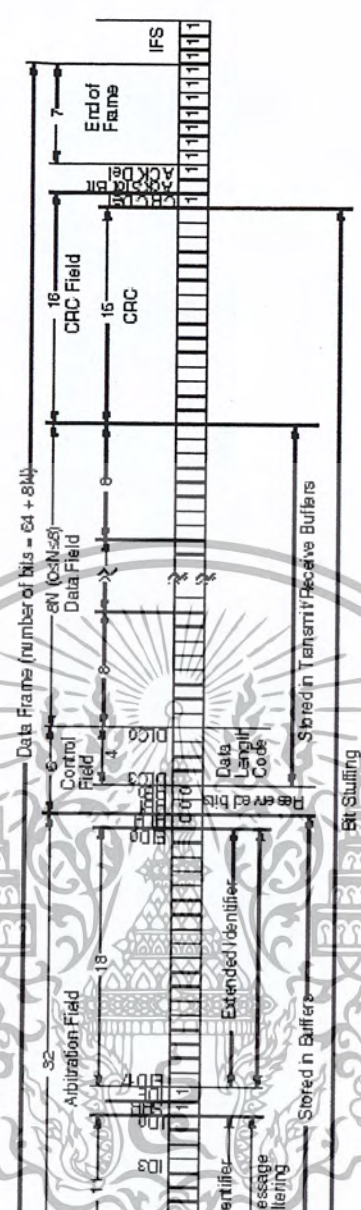
สาเหตุที่ต้องมีการขยายจำนวนหมายเลข ID เพิ่มขึ้นเนื่องจากจำนวนหมายเลข ID ในเฟรมข้อมูลแบบมาตรฐานที่มีอยู่ 2,048 หมายเลขนั้น แม้จะดูเหมือนมากแต่ในการใช้งานบางครั้งอาจจะไม่เพียงพอ จึงได้มีการกำหนดเฟรมข้อมูลแบบขยาย (CAN 2.0B) ขึ้นมาเพิ่มเติม โดยกำหนดให้หมายเลข ID มีขนาด 29 บิต ซึ่งจะรองรับการใช้งานได้มากถึง 2^{29} = 536,870,912 หมายเลขด้วยกัน

บิต SRR (Substitute Remote Request) จะอยู่ในตำแหน่งเดียวกับบิต RTR ที่อยู่ในเฟรมข้อมูลมาตรฐานเพื่อใช้แทนบิต RTR โดยจะมีค่าเป็น '1' หรือรีเซสสีฟ

ถัดจากบิต SRR ก็จะเป็นบิต IDE ซึ่งอยู่ตำแหน่งเดียวบิต IDE ในเฟรมข้อมูลแบบมาตรฐานและมีหน้าที่เดียวกัน โดยในที่นี้จะมีค่าเป็น '1'หรือรีเซสสีฟเนื่องจากเป็นเฟรมข้อมูลแบบขยาย

ส่วนบิต RTR จะอยู่ต่อจากหมายเลข ID ส่วนที่สอง (18 บิต) และมีค่าเป็น "0" หรือโดมิแนนต์เนื่องจากเป็นเฟรมข้อมูลนั่นเอง

- ฟิลด์ควบคุม มีขนาด 6 บิต เช่นกัน ส่วนที่แตกต่างไปจากเฟรมข้อมูลแบบมาตรฐานก็คือ RBI ซึ่งอยู่ในตำแหน่งเดียวกับบิต IDE ในฟิลด์ควบคุมของเฟรมข้อมูลแบบมาตรฐาน เนื่องจากบิต IDE ในเฟรมข้อมูลแบบขยายได้ถูกย้ายไปอยู่ในส่วนของฟิลด์แสดงรหัสข้อมูลเรียบร้อยแล้ว ดังนั้นที่ตำแหน่งนี้ในฟิลด์ควบคุมของเฟรมข้อมูลแบบขยายจึงได้กำหนดให้เป็นบิต RBI ซึ่งมีการสำรองไว้ใช้งานในอนาคต โดยกำหนดให้มีค่าเป็น "0" หรือโดมิแนนต์เอาไว้



รูปที่ 3.5 ส่วนประกอบในเฟรมข้อมูลแบบขยาย

3.2.2 เฟรมร้องขอข้อมูล

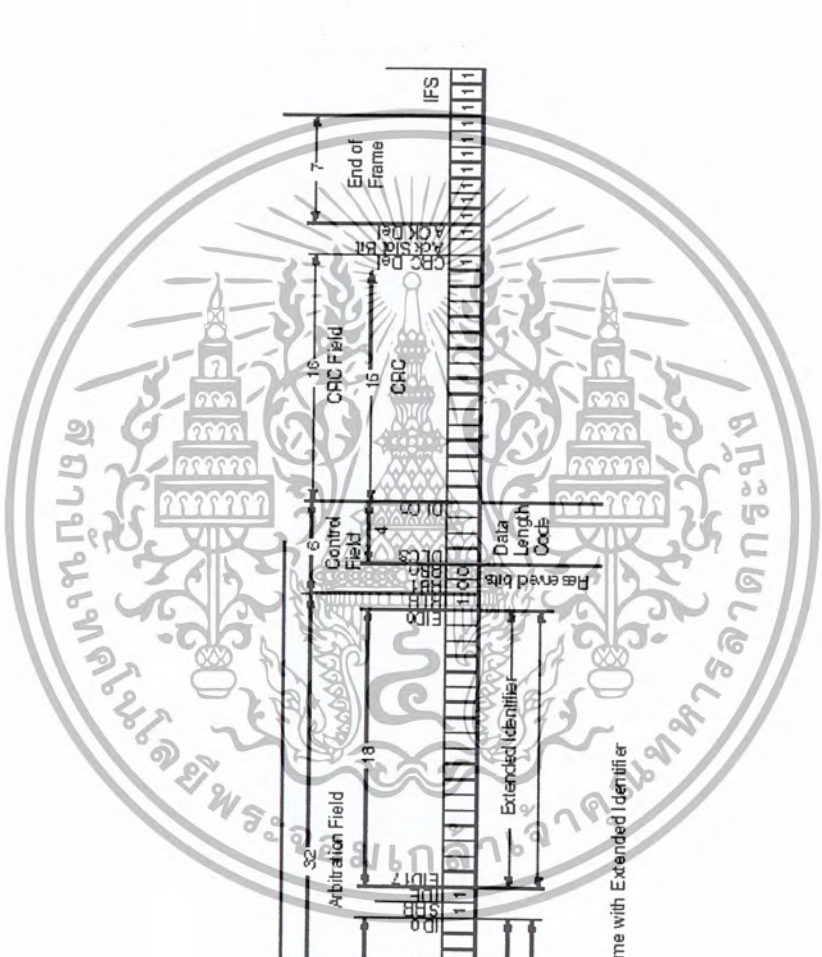
โนดผู้รับสามารถร้องขอข้อมูลจากโนดที่มีข้อมูลที่ต้องการอยู่ เพื่อให้โนดนั้นจัดส่งข้อมูลที่ต้องการนั้นมาให้ได้ โดยการส่งเฟรมร้องขอข้อมูลที่มีหมายเลข ID ของข้อมูลที่ต้องการนั้นออกไป หากโนดใดมีข้อมีที่มีหมายเลข ID ตรงกับที่ถูกร้องขอมา ก็จะทำการส่งข้อมูลนั้นตอบกลับไป

ลักษณะแฟรมร้องขอข้อมูลจะคล้ายกับเฟรมข้อมูล แต่จะมีส่วนที่แตกต่างจากเฟรมข้อมูลอยู่สองประการด้วยกันคือ

- บิต RTR ในเฟรมร้องขอข้อมูลจะมีค่าเป็น “1” หรือรีเซสสีฟ ในขณะที่ในเฟรมข้อมูลจะมีค่าเป็น “0” หรือโดมิแนนท์

- ในเฟรมร้องขอข้อมูลจะไม่มีฟิลด์ข้อมูล

ข้อสังเกตอีกอย่างหนึ่งก็คือ ในกรณีเฟรมร้องขอข้อมูลและเฟรมข้อมูลมีที่มีหมายเลข ID ตรงกันถูกส่งออกมาพร้อมกัน เฟรมข้อมูลจะมีสิทธิ์ได้ใช้บัสเนื่องจากบิต RTR จะมีค่าเป็น “0” หรือโดมิแนนท์ ในขณะที่โนดที่ส่งเฟรมร้องขอข้อมูลมาก็จะได้รับข้อมูลจากเฟรมข้อมูลที่ส่งมานั้นในทันที (เนื่องจากมีหมายเลข ID ตรงกัน)



รูปที่ 3.6 ส่วนประกอบในเฟรมร้องขอข้อมูล (แบบขยาย)

3.2.3 การเติมสตัฟฟ์บิต

เนื่องจากการส่งสัญญาณข้อมูลใน CAN จะใช้วิธีการส่งสัญญาณแบบ NRZ (Non-Return to Zero) ที่ระดับสัญญาณในแต่ละบิตมีค่าคงที่ตลอดช่วงระยะเวลาของบิตนั้นและเป็นการส่งข้อมูลแบบอะซิงโครนัส (Asyncronous) ที่ต้องอาศัยการเข้าจังหวะสัญญาณนาฬิกาจากสัญญาณข้อมูลที่ส่งมา การส่งข้อมูลด้วยวิธีนี้จะเกิดปัญหาขึ้นในกรณีที่สัญญาณข้อมูลมีค่าเดียวกันติด ๆ กันหลายบิต ซึ่งจะทำให้การเข้าจังหวะกับสัญญาณข้อมูลนั้นมีความผิดพลาดเกิดขึ้นได้มาก

ดังนั้นเพื่อการป้องกันไม่ให้เหตุการณ์เช่นนี้เกิดขึ้นใน CAN จึงได้กำหนดให้มีการเติมสตัฟฟ์บิต (Stuff Bit) คือถ้ามีการส่งบิตข้อมูลที่มีค่าเดียวกันติดกัน 5 บิตจุต้องคั่นด้วยบิตที่ตรงข้ามกับ 5 บิตนั้น 1 บิต และเมื่อโนดผู้รับได้รับข้อมูลที่มีค่าเดียวกันติดกัน 5 บิต ก็จะไม่นับบิตที่ตามมา 1 บิตว่าเป็นบิตข้อมูล แต่จะเริ่มรับบิตข้อมูลต่อในบิตถัดไป

การเติมสตัฟฟ์บิตจะเกิดขึ้นในเฟรมข้อมูลและเฟรมร้องขอข้อมูล ระหว่างบิตเริ่มต้นเฟรมกับบิต CRC Delimiter เท่านั้น ซึ่งนอกจากการเติมสตัฟฟ์บิตจะช่วยในการเข้าจังหวะสัญญาณข้อมูลแล้ว ยังใช้ตรวจสอบความผิดพลาดของข้อมูลได้อีกด้วย

3.3 การตรวจสอบและแก้ไขความผิดพลาด

จุดเด่นที่สำคัญอย่างหนึ่งของ CAN ก็คือ การตรวจสอบความผิดพลาดของข้อมูลที่เกิดขึ้น เพื่อให้เห็นภาพว่า CAN มีการตรวจสอบค่าความผิดพลาดของข้อมูลที่มีประสิทธิภาพมากเพียงใด จะขอยกตัวอย่างดังต่อไปนี้สมมติว่ามีการส่งข้อมูลด้วยอัตราความเร็ว 500 กิโลบิตต่อวินาที โดยที่ทุก ๆ 0.7 วินาที จะมีความผิดพลาดเกิดขึ้น 1 บิต และมีการใช้งาน 8 ชั่วโมงต่อหนึ่งวัน ด้วยระบบการตรวจสอบความผิดพลาดใน CAN รับประกันได้ว่าในระยะเวลา 1,000 ปี จะมีความผิดพลาดที่ไม่สามารถตรวจสอบได้เกิดขึ้นเพียง 1 ครั้งเท่านั้น จะเห็นได้ว่าโอกาสที่ข้อมูลที่ส่งนั้นจะเกิดความผิดพลาดขึ้นโดยที่ตรวจไม่พบนั้นมีน้อยมาก

ความผิดพลาดที่ตรวจสอบได้มีอยู่ 5 อย่างด้วยกันคือ

- CRC Error ในขณะที่โนดผู้ส่งทำการส่งข้อมูลจะคำนวณค่า CRC ของข้อมูลที่ส่งไปด้วยและส่งไปกับเฟรมข้อมูลนั้นในฟิลด์ตรวจสอบ ที่โนดผู้รับจะทำการคำนวณค่า CRC ของข้อมูลที่ได้รับด้วยวิธีเดียวกันกับโนดผู้ส่งแล้วนำไปเปรียบเทียบกับค่า CRC ที่ส่งมา หากไม่ตรงกันแสดงว่าข้อมูลที่ได้รับมามีความผิดพลาดเกิดขึ้น จะทำให้ที่โนดนั้นทำการส่งเฟรมแสดงความผิดพลาดเพื่อแจ้งกลับไปบอกว่าข้อมูลที่ได้รับนั้นไม่ถูกต้อง เมื่อโนดที่ส่งข้อมูลนั้นได้รับการแจ้งว่ามีค่าความผิดพลาดเกิดขึ้น ก็จะทำการส่งข้อมูลนั้นออกไปใหม่อีกครั้งหนึ่ง

- Acknowledge Error โนดผู้ส่งจะทำการตรวจสอบดูว่าที่บิต ACK Slot ในฟิลด์ตอบสนองจะมีค่าเป็น "0" หรือโดมิแนนต์หรือไม่ (ในขณะที่โนดผู้ส่งเองจะส่งค่า "1" หรือรีเซสสีฟออกไป) ถ้ามีค่าเป็นโดมิแนนต์แสดงว่าโนดอย่างน้อยหนึ่งโนดได้รับข้อมูลที่ถูกต้อง แต่ในทางตรงกันข้าม

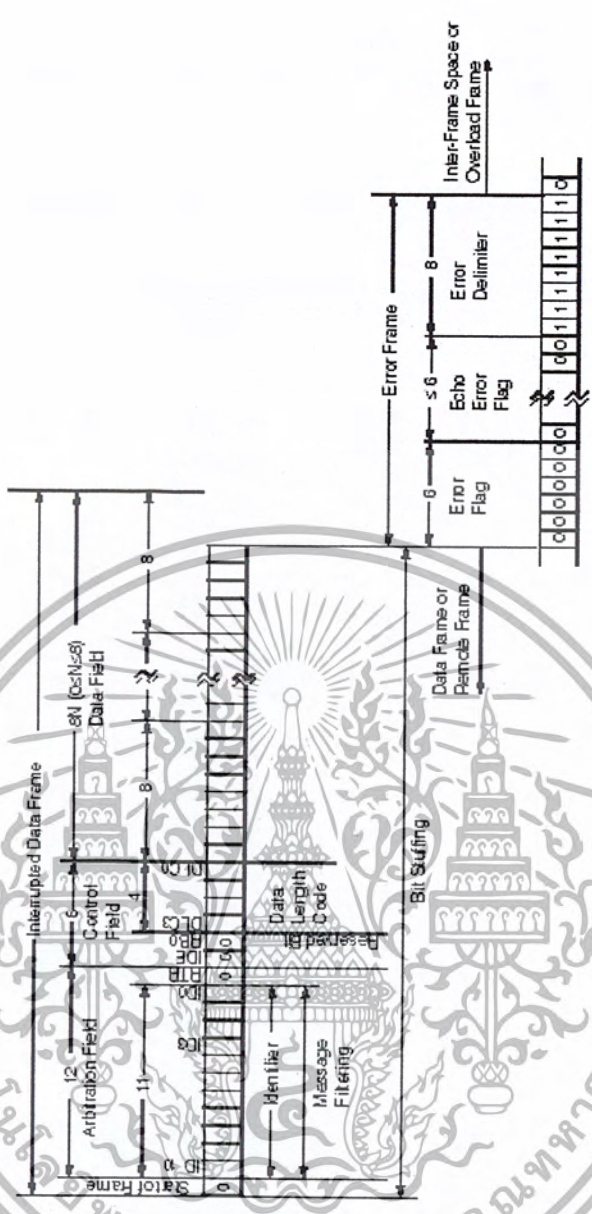
หากบิต ACK Slot มีค่าเป็นรีเซสสีฟ แสดงว่าไม่มีโนดใดที่ได้รับข้อมูลที่ถูกต้องเลย ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

- From Error ถ้าพบว่ามีสถานะโดมิแนนต์เกิดขึ้นในส่วนต่าง ๆ ต่อไปนี้ได้แก่ บิต CRC Delimiter, บิต ACK Delimiter และฟิลด์สิ้นสุดเฟรมแสดงว่ามีความผิดพลาดเกิดขึ้นในการรับส่งข้อมูลดังกล่าว (เนื่องจากส่วนต่าง ๆ เหล่านั้นต้องมีสถานะรีเซสสีฟเสมอ) ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

- Bit Error จะเกิดขึ้นเมื่อโนดผู้ส่งอ่านค่าจากบัสกลับมาได้ค่าไม่ตรงกับค่าที่ส่งออกไปคือค่าโดมิแนนต์แต่อ่านได้เป็นรีเซสสีฟ หรือส่งค่ารีเซสสีฟแต่อ่านได้เป็นโดมิแนนต์ สำหรับในกรณีที่ส่งค่ารีเซสสีฟแต่อ่านได้เป็นโดมิแนนต์ในฟิลด์แสดงรหัสข้อมูลและบิต ACK Slot นั้นไม่ถือว่าเป็นความผิดพลาดเนื่องจากสามารถเกิดขึ้นได้ เมื่อพบความผิดพลาดระบบจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

- Stuff Error จากที่กล่าวไปแล้วว่าการส่งข้อมูลใน CAN จะมีการเติมสตัฟฟ์บิตเมื่อข้อมูลที่ส่งมีค่าเดียวกันติดกัน 5 บิต ดังนั้นเมื่อข้อมูลที่ได้รับ (ที่อยู่ระหว่างบิตเริ่มต้นข้อมูลและบิต CRC Delimiter) มีค่าเดียวกันติดกันเกิน 5 บิตแสดงว่ามีความผิดพลาดเกิดขึ้น ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

นอกจากความสามารถในการตรวจสอบความผิดพลาดที่เกิดขึ้นได้แล้ว CAN ยังได้มีการกำหนดสถานะความผิดพลาดของแต่ละโนดในบัสขึ้น เพื่อควบคุมให้การใช้งานบัสมีความถูกต้องและประสิทธิภาพมากยิ่งขึ้น กล่าวคือในกรณีที่โนดใดเกิดความผิดพลาดมากกว่าเกณฑ์ที่กำหนดไว้ก็จะไม่อนุญาตให้ใช้บัสต่อได้ เนื่องจากหากปล่อยให้โนดดังกล่าวใช้บัสได้ตามปกติจะทำให้การใช้งานบัสอาจมีประสิทธิภาพต่ำลง เพราะจะเกิดความผิดพลาดขึ้นบ่อยครั้งซึ่งทุกครั้งจะต้องมีการส่งข้อมูลกันใหม่ ทำให้ปริมาณการใช้งานของบัสมีมากขึ้นและไปแย่งเวลากับโนดอื่นโดยไม่จำเป็น

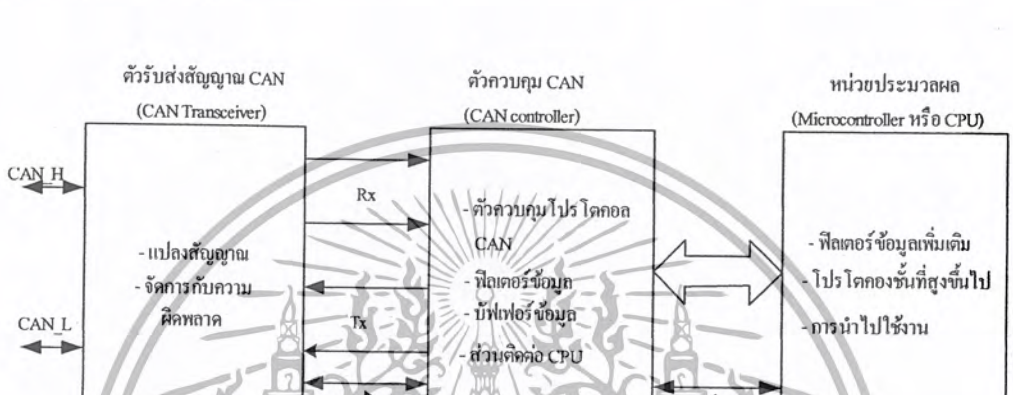


รูปที่ 3.7 ส่วนประกอบในเฟรมตรวจสอบความผิดพลาด

บทที่ 4
โครงสร้างของ CAN และการนำไปใช้

4.1 โครงสร้างโนด CAN

โมเดลการทำงานของโนดแต่ละโนดในระบบบัส CAN ซึ่งโดยทั่วไปแล้วจะมีลักษณะโครงสร้างดังแสดงในรูปที่ 4.1



รูปที่ 4.1 โมเดลการทำงานของแต่ละโนดในระบบบัส CAN

มีส่วนประกอบหลักๆอยู่ด้วยกัน 3 ส่วนด้วยกันคือ

- หน่วยควบคุมและประมวลผล
- ตัวควบคุม CAN
- ตัวรับส่งสัญญาณ CAN

ในส่วนของการนำ CAN ไปใช้งานนั้นปัจจุบันหลายบริษัทได้มีการออกแบบและพัฒนาตัวควบคุม CAN ให้อยู่ในรูปของวงจรรวมหรือไอซีโดยการทำงานของตัวควบคุม CAN ที่ได้ออกแบบขึ้นมาทั้งหมดนั้นจะอ้างอิงตามมาตรฐาน ISO 11898 ซึ่งเป็นส่วนของการทำงานที่อยู่ในชั้นการเชื่อมโยงข้อมูลที่ทำหน้าที่จัดการในเรื่องการที่สื่อสารข้อมูลขั้นพื้นฐาน เช่น การส่งข้อมูลและการร้องขอข้อมูล ให้เป็นไปอย่างถูกต้อง รวมไปถึงการตรวจสอบความผิดพลาดของข้อมูลด้วย ซึ่งจะอ้างอิงตามโมเดลการทำงานของตัวควบคุม CAN จากบริษัท Bosch ผู้ให้กำเนิด CAN ขึ้นมานั่นเอง

ส่วนประกอบอีกส่วนหนึ่งที่สำคัญที่ทำให้ CAN สามารถสื่อสารเป็นระบบบัสข้อมูลได้ ก็คือตัวรับส่งสัญญาณ CAN ซึ่งจะทำหน้าที่เป็นแปลงข้อมูลที่ต้องการส่งให้เป็นสัญญาณที่สามารถส่งผ่านบัสได้ตามข้อกำหนดที่มีขึ้นสำหรับการทำงานในชั้นกายภาพ

ตัวควบคุม CAN ที่มีลักษณะ-v'การทำงานที่รวมฟังก์ชันการควบคุมโปรโตคอลไว้ในไอซีตัวเดียวและแยกออกมาจากส่วนประมวลผลนั้น เราจะเรียกว่าเป็นตัวควบคุม CAN แบบ Stand-alone และในปัจจุบันได้มีบริษัทผู้ผลิตไอซีไมโครคอนโทรลเลอร์ หลายๆค่ายที่ออกแบบให้มีตัวควบคุม CAN รวมอยู่ในไอซีไมโครคอนโทรลเลอร์ ซึ่งจะเรียกตัวควบคุม CAN ที่มีลักษณะเช่นนี้ว่าเป็นตัวควบคุมแบบ Intergrated

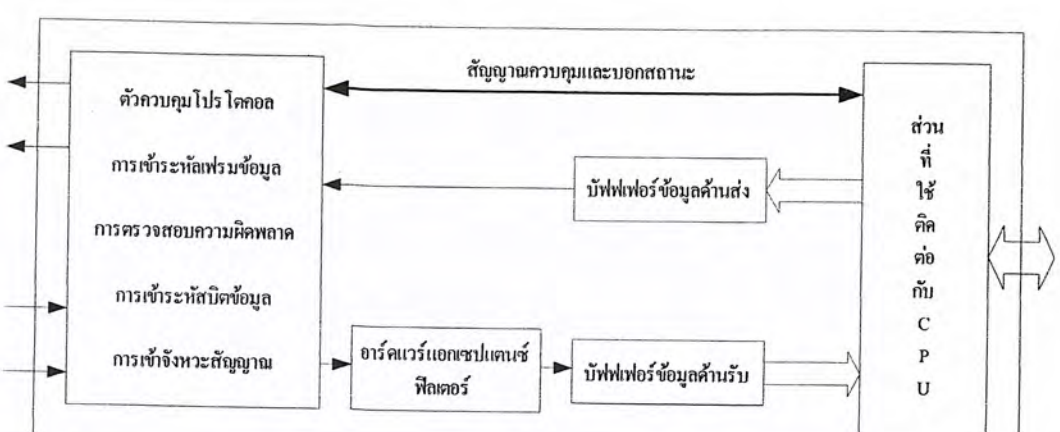
ตัวควบคุม CAN ทั้งสองแบบนี้มีข้อดีและมีจุดเด่นในการใช้งานที่แตกต่างกันออกไปโดยตัวควบคุม CAN แบบ Stand-alone นั้นมีข้อดี คือ สามารถนำไปใช้งานร่วมกับหน่วยประมวลผลหรือ CPU แบบใดก็ได้ตามความต้องการของผู้ใช้งานแต่ละความต้องการ ทำให้มีทางเลือกสำหรับการพัฒนาในส่วนของซอฟต์แวร์ที่หลากหลายกว่า ส่วนตัวควบคุม CAN แบบ Integrated นั้นมีข้อดีกว่าในเรื่องค่าใช้จ่ายในส่วนของการสร้างฮาร์ดแวร์ที่ต่ำกว่าแบบแรก ทั้งในด้านราคาของไอซีและค่าใช้จ่ายในการผลิตแผ่นวงจรพิมพ์ซึ่งมีขนาดและความซับซ้อนน้อยกว่า

นอกจากนี้การทำงานของ Integrated ยังมีข้อดีในเรื่องการใช้เวลาทำงานน้อยนอกจากนี้การใช้ทรัพยากรของ CPU ที่น้อยกว่าด้วย เนื่องจากการควบคุมจะมีการใช้การติดต่อผ่านบัสข้อมูลและแอดเดรสภายใน ในขณะนี้ที่ตัวควบคุมแบบ Stand-alone จะต้องอาศัยการถ่ายข้อมูลผ่านบัสข้อมูลและแอดเดรสภายนอก ซึ่งจะส่งผลให้การใช้ตัวควบคุมแบบ Stand-Alone สิ้นเปลืองทรัพยากรของ CPU มากกว่าและทำงานได้ช้ากว่าด้วย

ส่วนโมเดลการทำงานของตัวควบคุม CAN ในอีกรูปแบบนั้นจะเป็นการรวมส่วนประกอบทั้งหมดเข้ามาไว้ในไอซีเพียงตัวเดียว คือจะรวบรวมตัวรับส่งสัญญาณ CAN เข้ามาด้วยโดยเรียกตัวควบคุมแบบ Single-chip แนวโน้มของตัวควบคุม CAN ในลักษณะนี้เริ่มมีให้เห็นกันแล้วในปัจจุบันโดยเป็นการออกแบบตัวควบคุม CAN เพื่อใช้งานในระบบควบคุมสำหรับยานยนต์

4.2 โครงสร้างตัวควบคุม CAN

ตัวควบคุม CAN ถือเป็นส่วนประกอบสำคัญของโนดในระบบ CAN ที่ทำหน้าที่ควบคุมกลไกในการรับส่งข้อมูลทั้งหมดให้เป็นไปตามมาตรฐานต่าง ๆ ที่กำหนดไว้ โดยการทำงานภายในตัวควบคุม CAN นั้นจะมีลักษณะของโครงสร้างหลักที่เหมือนกันดังรูปที่ 4.2

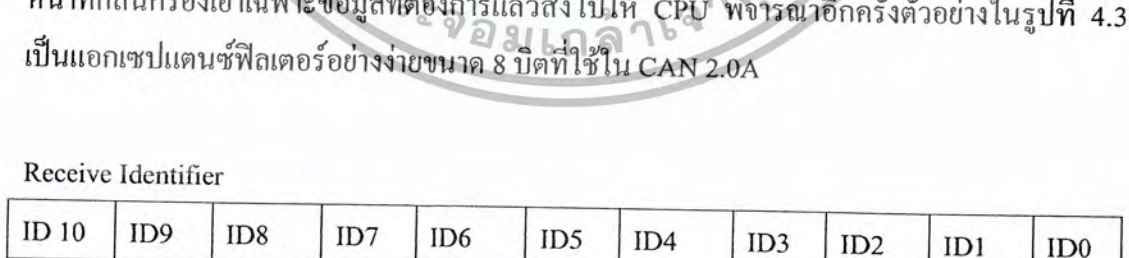


รูปที่ 4.2 โครงสร้างการทำงานของตัวควบคุม CAN

ประกอบด้วยตัวควบคุมโปรโตคอลCAN ฮาร์ดแวร์แอกเซฟแตนซ์ฟิลเตอร์ บัฟเฟอร์ข้อมูล และส่วนติดต่อกับ CPU ซึ่งแต่ละส่วนจะมีหน้าที่การทำงานดังนี้

- ตัวควบคุมโปรโตคอล CAN ทำหน้าที่ควบคุมกลไกการรับส่งข้อมูลทั้งหมดที่เกิดขึ้นในบัส CAN เช่น การสร้างเฟรมข้อมูล การตรวจสอบความผิดพลาด การเข้ารหัสบิตข้อมูล และการเข้าจังหวะสัญญาณเป็นต้น ให้เป็นไปตามโปรโตคอลที่ได้กำหนดไว้เพื่อให้ตัวควบคุม CAN ที่ไม่ว่าจะผลิตขึ้นมาจากบริษัทใด ๆ ก็ตามสามารถนำมาใช้งานร่วมกันได้โดยไม่มีปัญหา

- ฮาร์ดแวร์แอกเซปแตนซ์ฟิลเตอร์ เนื่องจากข้อมูลที่รับส่งใน CAN นั้น มีรหัสประจำตัวหรือหมายเลข ID อยู่เป็นจำนวนมาก (โดย CAN 2.0A มีหมายเลข ID = 2,048 หมายเลขในขณะที่ CAN 2.0B มีหมายเลข ID มากถึง 536,870,912 หมายเลขด้วยกัน) การที่จะยกให้เป็นภาระของ CPU ในการแยกแยะหมายเลข ID ของข้อมูลทั้งหมดนั้นจะไม่เหมาะสม แอกเซปแตนซ์ฟิลเตอร์นี้จะทำหน้าที่กลั่นกรองเอาเฉพาะข้อมูลที่ต้องการแล้วส่งไปให้ CPU พิจารณาอีกครั้งตัวอย่างในรูปที่ 4.3 เป็นแอกเซปแตนซ์ฟิลเตอร์อย่างง่ายขนาด 8 บิตที่ใช้ใน CAN 2.0A



รูปที่ 4.3 เป็นแอกเซปแตนซ์ฟิลเตอร์อย่างง่ายขนาด 8 บิตที่ใช้ใน CAN 2.0A

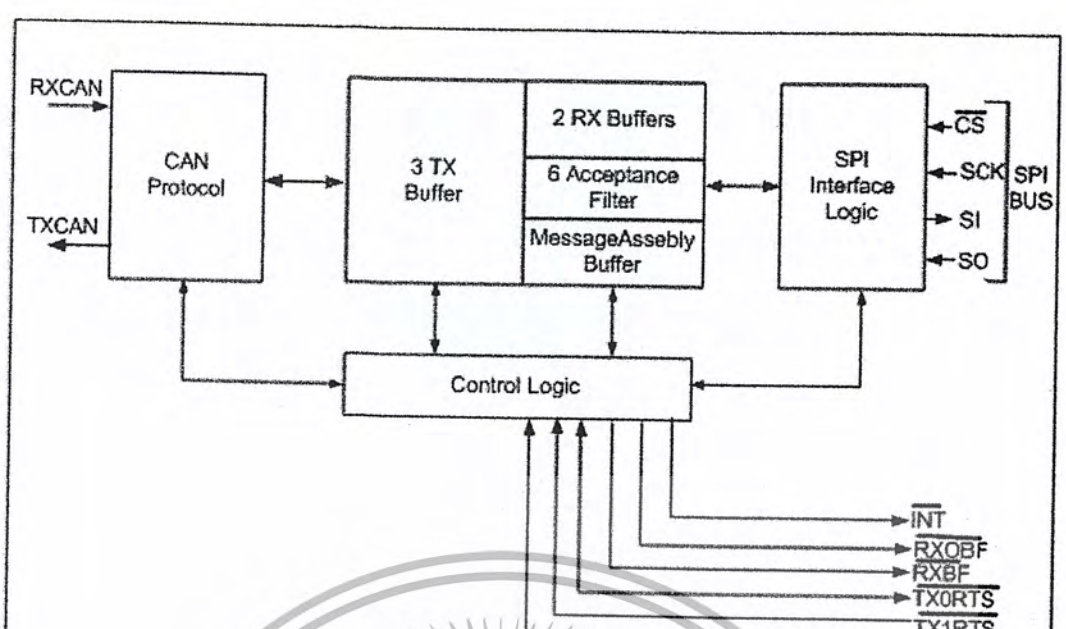
ซึ่งมีหมายเลข ID แบบมาตรฐานขนาด 11 บิต โดยมีแอกเซปแตนซ์มาสก์เป็นตัวระบุหมายเลข ID ของข้อมูลที่ต้องการ

- บัฟเฟอร์ข้อมูล เพื่อให้การรับส่งข้อมูลทำได้สะดวกโดยไม่ต้องคำนึงว่า CPU พร้อมที่จะทำงานหรือไม่ หรือมีโนดอื่น ๆ กำลังใช้งานบัสอยู่ และช่วยให้ข้อมูลไม่มีการสูญหายเมื่อ CPU ไม่พร้อมที่จะมาอ่านข้อมูลที่ได้รับในขณะนั้น จึงจะต้องมีบัฟเฟอร์ข้อมูลเพื่อทำหน้าที่เก็บข้อมูลไว้ชั่วคราว ทั้งข้อมูลที่ต้องการจะส่งและข้อมูลที่ได้รับมาโดยบัฟเฟอร์ของข้อมูลด้านส่งจะเก็บข้อมูลที่ต้องการส่งไว้ชั่วคราวจนกว่าบัสจะว่างพร้อมที่จะส่งข้อมูลไปได้โดยที่ CPU ไม่จำเป็นต้องมาเสียเวลาตรวจสอบบัสอยู่ตลอดเวลา ในขณะที่บัฟเฟอร์ข้อมูลด้านรับจะเก็บข้อมูลที่ได้รับมาไว้ชั่วคราวจนกระทั่ง CPU พร้อมที่จะอ่านข้อมูลไป หรือไม่ต้องการข้อมูลนั้นแล้ว ตัวอย่างในรูปที่ 4.4 เป็นบัฟเฟอร์ข้อมูลอย่างง่ายที่ใช้ใน CAN 2.0A ประกอบด้วยบัฟเฟอร์ข้อมูลขนาด 8 ไบต์ หมายเลข ID ขนาด 11 บิต ข้อมูล DLC ขนาด 4 บิต และ บิต RTR 1 บิต

ID 10	ID 9	ID 8	ID 7	ID 6	ID 5	ID 4	ID 3
ID 2	ID 1	ID 0	RTR	DLC3	DLC2	DLC1	DLC0
Data Byte 1							
Data Byte 2							
Data Byte 3							
Data Byte 4							
Data Byte 5							
Data Byte 6							
Data Byte 7							
Data Byte 8							

รูปที่ 4.4 ตัวอย่างบัฟเฟอร์ข้อมูลอย่างง่ายที่ใช้ใน CAN 2.0A

- ส่วนติดต่อกับ CPU สำหรับตัวควบคุม CAN แบบ Stand-Alone แม้ว่าจะสามารถทำงานได้โดยลำพัง แต่ก็ยังจำเป็นที่จะต้องมีการติดต่อกับอุปกรณ์อื่นภายนอกในส่วนของการตั้งค่าเริ่มต้นในการทำงาน รวมทั้งการถ่ายโอนข้อมูลที่ต้องการรับส่งถึงกัน ซึ่งโดยทั่วไปแล้วมักจะนำตัวควบคุม CAN ไปใช้งานร่วมกับหน่วยประมวลผล คือ CAU ไมโครคอนโทรลเลอร์ต่าง ๆ จึงต้องมีส่วนที่ใช้ในการติดต่อกับ CPU เหล่านี้ วิธีการติดต่อนั้นมีอยู่หลายรูปแบบด้วยกัน มีทั้งแบบขนาดและแบบอนุกรม ทั้งนี้เพื่อความสะดวกในการเลือกแบบที่จะนำไปใช้นั่นเอง



รูปที่ 4.5 โครงสร้างการทำงานภายใน MCP 2510

รูปที่ 4.5 เป็นตัวควบคุมแบบ Stand-Alone คือไม่มี CPU อยู่ภายในไอซีใช้งานได้ทั้งกับ CAN 2.0A และ CAN 2.0B ที่ความเร็ว 1 เมกะบิตต่อวินาที โดยมีแอกเซปแตนซ์ฟิลเตอร์ 6 ชุด และแอกเซปแตนซ์มาสก์ 2 ชุด มีบัฟเฟอร์ข้อมูลด้านส่ง 3 ชุด และบัฟเฟอร์ข้อมูลด้านรับ 2 ชุด ที่ใช้ในการติดต่อกับ CPU ภายนอกผ่านการเชื่อมต่ออนุกรมแบบ SPI

พื้นที่รีจิสเตอร์ (Register Map)

REGISTER MAP ของ MCP 2510 แสดงดังตารางที่ 1 ตำแหน่งแอดเดรสของรีจิสเตอร์กำหนดโดยใช้ค่าหลัก (คำสั่ง 4 บิตสูง) และแถว (คำสั่ง 4 บิตต่ำ) รีจิสเตอร์ถูกจัดเรียงให้เหมาะสมกับการอ่านและเขียนข้อมูลอย่างต่อเนื่องบางชนิดของการควบคุมเฉพาะ และสถานะรีจิสเตอร์นั้นจะยอมทำตามคำสั่งที่แสดงตรงพื้นที่ shaded ในตารางที่ 4.1 ข้อสรุป Control Register ของ MCP 2510 แสดงพื้นที่ในตาราง 4.2

ตารางที่ 4.1 CAN Controller Register Map

Lower Address Bits	Higher Order Address Bits							
	x000 xxxx	x001 xxxx	x010 xxxx	x011 xxxx	x100 xxxx	x101 xxxx	x110 xxxx	x111 xxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID0	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

หมายเหตุ; พื้นที่ Shaded Register ชี้บอกส่วนของพื้นที่ที่ยอมให้ผู้ใช้แก้ไขบิตตามความต้องการ โดยใช้คำสั่ง “Bit Modify”

ตารางที่ 4.2 Control Register Summary

Register Name	Address (Hex)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	POR/RST Value
BFPCTRL	0C	–	–	B1BFS	B0BFS	B1BFE	B0BFE	B1BFM	B0BFM	--00 0000
TXRTSCTRL	0D	–	–	B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM	--xx x000
CANSTAT	xE	OPMOD2	OPMOD1	OPMOD0	–	ICOD2	ICOD1	ICOD0	–	100- 000-
CANCTRL	xF	REQOP2	REQOP1	REQOP0	ABAT	–	CLKEN	CLKPRE1	CLKPRE0	1110 -111
TEC	1C	Transmit Error Counter								0000 0000
REC	1D	Receive Error Counter								0000 0000
CNF3	28	–	WAKFIL	–	–	–	PHSEG22	PHSEG21	PHSEG20	-0-- -000
CNF2	29	BTLMODE	SAM	PHSEG12	PHSEG11	PHSEG10	PRSEG2	PRSEG1	PRSEG0	0000 0000
CNF1	2A	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	0000 0000
CANINTE	2B	MERRE	WAKIE	ERRIE	TX2IE	TX1IE	TX0IE	RX1IE	RX0IE	0000 0000
CANINTF	2C	MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF	0000 0000
EFLG	2D	RX1OVR	RX0OVR	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000
TXB0CTRL	30	–	ABTF	MLOA	TXERR	TXREQ	–	TXP1	TXP0	-000 0-00
TXB1CTRL	40	–	ABTF	MLOA	TXERR	TXREQ	–	TXP1	TXP0	-000 0-00
TXB2CTRL	50	–	ABTF	MLOA	TXERR	TXREQ	–	TXP1	TXP0	-000 0-00
RXB0CTRL	60	–	RXM1	RXM0	–	RXRTR	BUKT	BUKT	FILHIT0	-00- 0000
RXB1CTRL	70	–	RSM1	RXM0	–	RXRTR	FILHIT2	FILHIT1	FILHIT0	-00- 0000

บทที่ 5
การสร้างบอร์ดอินเตอร์เฟส CAN

5.1 โครงสร้างของบอร์ดอินเตอร์เฟส CAN

โครงสร้างของบอร์ดอินเตอร์เฟส CAN มีส่วนประกอบหลักอยู่สองส่วน ได้แก่ ตัวควบคุม CAN (CAN Controller) ในการทดลองใช้ไอซีเบอร์ MCP 2510 ตัวรับส่งสัญญาณ CAN (CAN Transceiver) ในการทดลองใช้ไอซีเบอร์ MCP 2551 และส่วนของไมโครคอลโทรลเลอร์ ในการทดลองใช้ไอซีเบอร์ AT89S53 โดยตัวควบคุม CAN จะทำการควบคุมกระบวนการให้เป็นไปตามโปรโตคอลที่กำหนดไว้ในชั้นเชื่อมโยงข้อมูล (Data Link Layer) ตัวรับส่งสัญญาณ CAN จะทำหน้าที่แปลงสัญญาณที่ต้องการส่ง (ลอจิก "0" หรือ "1") ให้เป็นสัญญาณที่ใช้ในบัส CAN (สถานะโดมิแนนต์หรือรีเซสสีฟ) ซึ่งจะสอดคล้องกับมาตรฐานที่กำหนดไว้ในชั้นกายภาพ (Physical Layer) ในขณะเดียวกันก็ทำการแปลงสัญญาณที่ได้รับจากบัส CAN กลับไปเป็นสัญญาณข้อมูลด้วย ส่วนไมโครคอลโทรลเลอร์นั้น ทำหน้าที่เป็นตัวประมวลผลข้อมูลที่ได้จากอุปกรณ์ภายนอกเพื่อส่งไปยังตัวควบคุม CAN และเป็นตัวประมวลผลสัญญาณจากตัวควบคุม CAN นำสัญญาณไปควบคุมและเชื่อมต่อกับอุปกรณ์ภายนอก

5.1.1 ตัวควบคุม CAN (CAN Controller)

ตัวควบคุม CAN หรือ CAN Controller ในการทดลองใช้ไอซีเบอร์ MCP 2510 ซึ่งถือว่าเป็นหัวใจหลักของการทำงานของบอร์ดอินเตอร์เฟส CAN ซึ่งทำหน้าที่เป็นตัวควบคุมโปรโตคอลในการรับส่งข้อมูลใน CAN โดยอาศัยการติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอกผ่านทางพอร์ต SPI (Serial Peripheral Interface) ที่มีอยู่ในตัวไอซีโดยจะติดต่อกับไมโครคอลโทรลเลอร์โดยผ่านทางพอร์ต SPI ได้โดยตรง

คำสั่งที่ใช้รับส่งข้อมูลด้วย SPI

ในการควบคุมบอร์ดอินเตอร์เฟส CAN นั้นจะใช้การควบคุมโดยการกำหนดค่ารีจีสเตอร์ภายในและการตรวจสอบสถานะการทำงานของไอซีควบคุม CAN (CAN controller) โดยใช้คำสั่งต่าง ๆ ดังแสดงในตารางที่ 5.1

1. คำสั่งการอ่าน (Read Instruction)

คำสั่งการอ่านจะเริ่มโดย ทำให้ขา CS เป็น Low คำสั่งการอ่านเมื่อส่งถึง MCP 2510 (CAN Controller) จะตามด้วยแอดเดรส 8 บิต (A0-A7) หลังจากคำสั่งอ่านและแอดเดรสเป็นการส่งข้อมูลในรีจิสเตอร์ที่ถูกเลือกแอดเดรสไว้ โดยจะถูกส่งออกที่ขา SO ตัวชี้แอดเดรสภายในจะเพิ่มขึ้น

อัตโนมัติที่แอดเดรสถัดไปหลังจากข้อมูลแต่ละไบต์ถูกส่งออกไป ดังนั้นมันสามารถอ่านแอดเดรสรีจิสเตอร์ค่าถัดไปได้อย่างต่อเนื่องโดยมี Clock Pulses บางตำแหน่งของรีจิสเตอร์แบบต่อเนื่องนั้นสามารถอ่านลำดับโดยใช้วิธีนี้ เมื่อการอ่านเสร็จสิ้นขา CS จะมีสภาเป็น High (รูปที่ 5.2 คำสั่งการอ่าน)

2. คำสั่งการเขียน (Write Instruction)

คำสั่งการอ่านเริ่มโดยให้ขา CS เป็น Low คำสั่งการอ่านจะส่งไปที่ MCP 2510 ตามด้วยแอดเดรสและข้อมูลอย่างน้อยที่สุด 1 ไบท์ มันจะเรียงลำดับรีจิสเตอร์โดยการต่อสัญญาณนาฬิกาลงในไบท์ข้อมูลคือการให้ CS เป็น Low ข้อมูลจะถูกเขียนถึงรีจิสเตอร์ที่ขอบขาขึ้นของขา SCK สำหรับบิต DO ค่าของ CS จะถูกทำให้เป็น High ก่อน 8 บิตนั้นก็จะเป็น Load การเขียนจะเกิดขึ้นก่อนข้อมูลไบท์นั้นคือก่อนไบท์คำสั่งจะมีการเขียน ดูจาก Timing Diagram ในรูปที่ 5.3 จะอธิบายรายละเอียดของไบท์ข้อมูลตามลำดับ

3. คำสั่งร้องขอให้เกิดการส่งข้อมูล (RTS)

คำสั่ง RTS ใช้ทำให้เกิดการส่งข้อมูล 1 หรือมากกว่าของบัฟเฟอร์การส่งการเลือกทำโดยให้ขา CS เป็น Low และส่งไบท์คำสั่ง RTS ส่งให้ MCP 2510 แสดงดังรูปที่ 9 โดย 3 บิตสุดท้ายของคำสั่งจะแสดงว่า บัฟเฟอร์ตัวส่งสามารถส่งได้ คำสั่งนี้จะเซ็ทบิต TxBnCTRL.TXREQ สำหรับการลำดับบัฟเฟอร์ 3 บิตสุดท้ายสามารถเซ็ทในคำสั่งเดียว ถ้าการส่งคำสั่ง RTS เท่ากับ nnn=000 คำสั่งจะไม่สนใจ (รูปที่ 5.4 คำสั่งร้องขอให้เกิดการส่งข้อมูล)

4. คำสั่งการอ่านสภาวะ

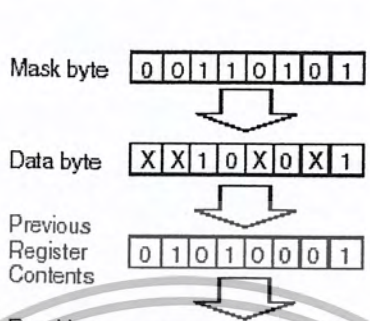
คำสั่งการอ่านสภาวะจะยอมให้คำสั่งเดียวสามารถเข้าถึงมักใช้สภาวะบิตของการรับและการส่งข้อมูลเลือกโดยให้ CS เป็น Low และอ่านค่าสภาวะไบท์คำสั่งดังรูปที่ 11 ส่งให้ MCP 2510 หลังจากส่งคำสั่งไบท์ MCP 2510 จะคืนค่าสภาวะของข้อมูล 8 บิต ค่า clock จะส่งหลังจาก 8 บิตแรก โดยจะต่อเนื่องจนถึงสภาวะเอาท์พุต คือให้ CS เป็น Low และจะเตรียม Clock บน SCK แต่ละสภาวะบิตจะคืนค่า ในคำสั่งนี้สามารถจะอ่านค่าโดยใช้คำสั่งมาตรฐานใช้กับรีจิสเตอร์ที่เหมาะสม (รูปที่ 5.6 คำสั่งการอ่านสถานะ)

5. คำสั่งแก้ไขบิต (Bit Modify)

คำสั่งแก้ไขบิตจะมีวิธีการสำหรับการเซ็ทหรือการเคลียร์แต่ละบิต ในสภาวะที่แน่นอนและคำสั่งนี้ไม่ได้เหมาะสมกับคอนโทรลรีจิสเตอร์ทุกตัว ดูที่ตารางที่ 1 (Register Map) ซึ่งจะมีการกำหนดรีจิสเตอร์ที่ยอมให้ใช้คำสั่งนี้ การเลือกใช้ส่วนนี้คือให้ขา CS เป็น Low แล้วส่งคำสั่งแก้ไขบิตให้ MCP 2510 ตามด้วยแมชไบท์และไบท์ข้อมูล ซึ่งแมชไบท์จะกำหนดบิตในรีจิสเตอร์โดยจะยอมให้เปลี่ยนเมื่อ A = 1 แมชไบท์จะยอมให้บิตในรีจิสเตอร์เปลี่ยนและ A=0 จะไม่ยอม ส่วนไบท์ข้อมูล A=1 จะเซ็ท A=0 จะเคลียร์ ดูจากรูปที่ 5.5

6. คำสั่งรีเซ็ท (RESET)

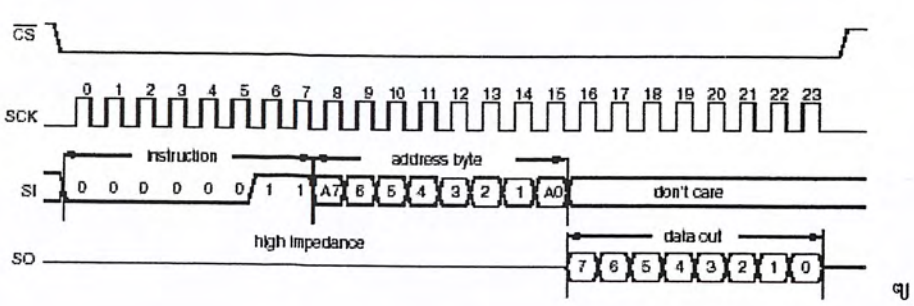
คำสั่งนี้สามารถใช้กับรีจิสเตอร์ในตอนแรกของ MCP 2510 โดยจะมีการเซ็ทค่าของฟังก์ชั่นในโหมด ผ่านส่วนติดต่อ SPI ทางขา Reset (ทำงานที่สภาวะ Low) คำสั่งรีเซ็ทเป็นคำสั่งเดี่ยว เลือกโดยให้ขาCS เป็น Low ส่งไบท์คำสั่ง เสร็จแล้วเพิ่มค่า CS ให้เป็น High (รูปที่ 5.7 คำสั่งรีเซ็ท)



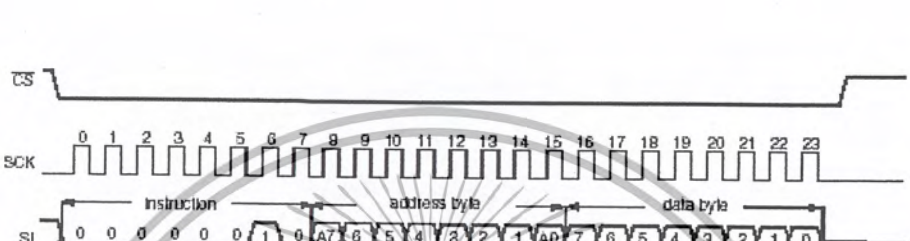
รูปที่ 5.1 ตัวอย่าง Bit Modify

ตารางที่ 5.1 การเซ็ทคำสั่ง SPI

ชื่อคำสั่ง	คำสั่ง	คำอธิบาย
รีเซ็ท	1100 0000	รีเซ็ทสถานะรีจิสเตอร์ภายใน,และเซ็ทค่าในโหมด
อ่าน	0000 0011	เริ่มอ่านข้อมูลจากรีจิสเตอร์ในตำแหน่งที่เลือกไว้
เขียน	0000 0010	เริ่มเขียนข้อมูลจากรีจิสเตอร์ในตำแหน่งที่เลือกไว้
RTS (ร้องขอให้เกิดการส่ง)	1000 0000	เซ็ทบิต TxBnCTRL.TXREQ สำหรับการส่งหนึ่งบัฟเฟอร์หรือมากกว่า 1000 0nnn (nnn=การร้องขอให้ส่งสำหรับ TXB2,TXB1,TXB0)
อ่านสถานะ		คำสั่งในการอ่านค่าสถานะของบิตเอาท์พุตของการรับและการส่งฟังก์ชั่น
แก้ไขบิต		เลือกรีจิสเตอร์ในการแก้ไขบิต

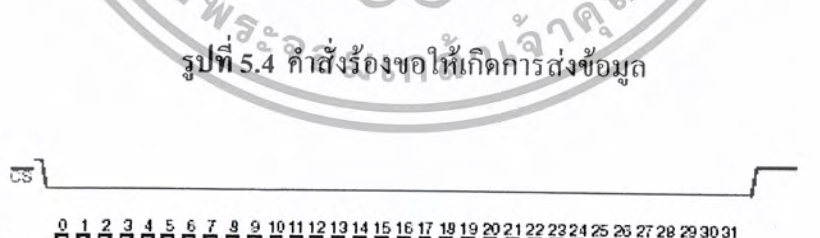


รูปที่ 5.2 คำสั่งการอ่าน



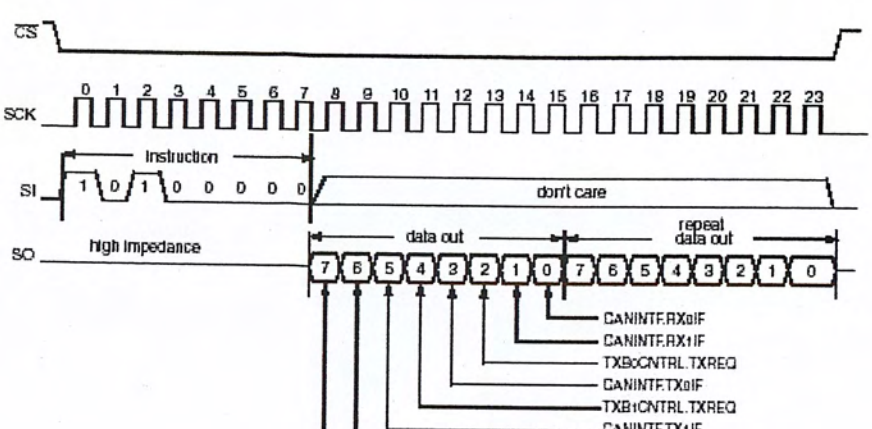
รูปที่ 5.3 คำสั่งการเขียน

รูปที่ 5.4 คำสั่งร้องขอให้เกิดการส่งข้อมูล

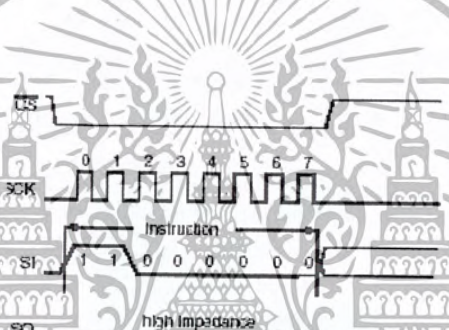


หมายเหตุ รีจิสเตอร์บางตัวจะไม่สามารถใช้คำสั่งนี้ โดยดูจาก Register Map ในตารางที่ 4.1 สำหรับรีจิสเตอร์ที่สามารถใช้ได้

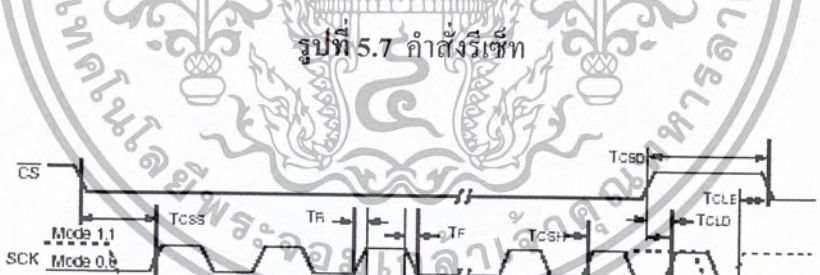
รูปที่ 5.5 คำสั่ง Bit Modify



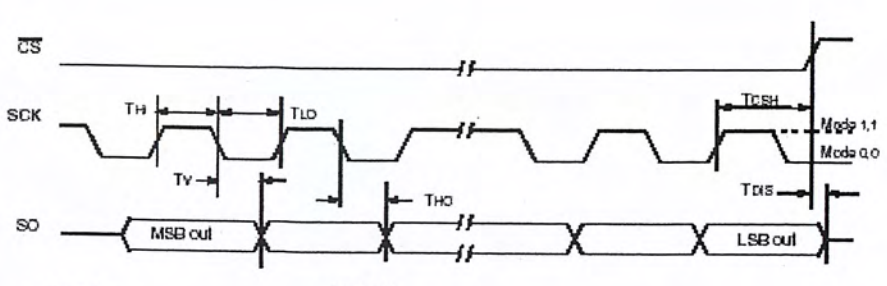
รูปที่ 5.6 คำสั่งการอ่านสภาวะ



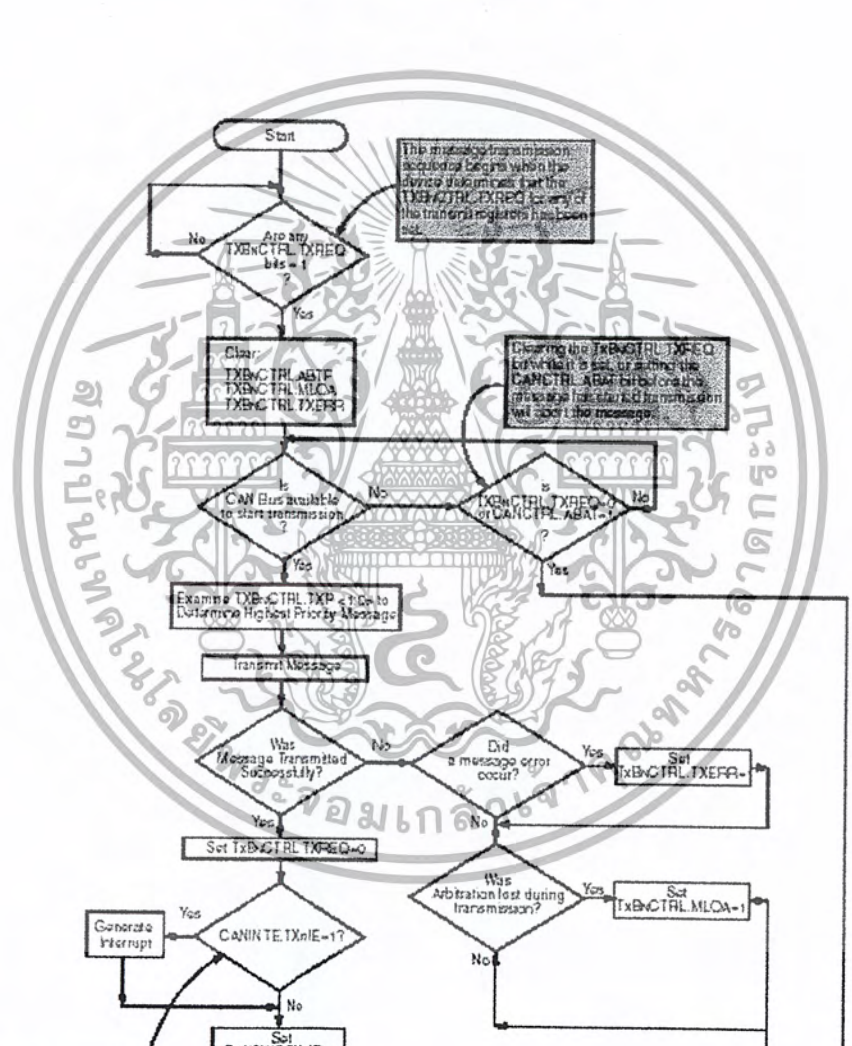
รูปที่ 5.7 คำสั่งรีเซ็ท



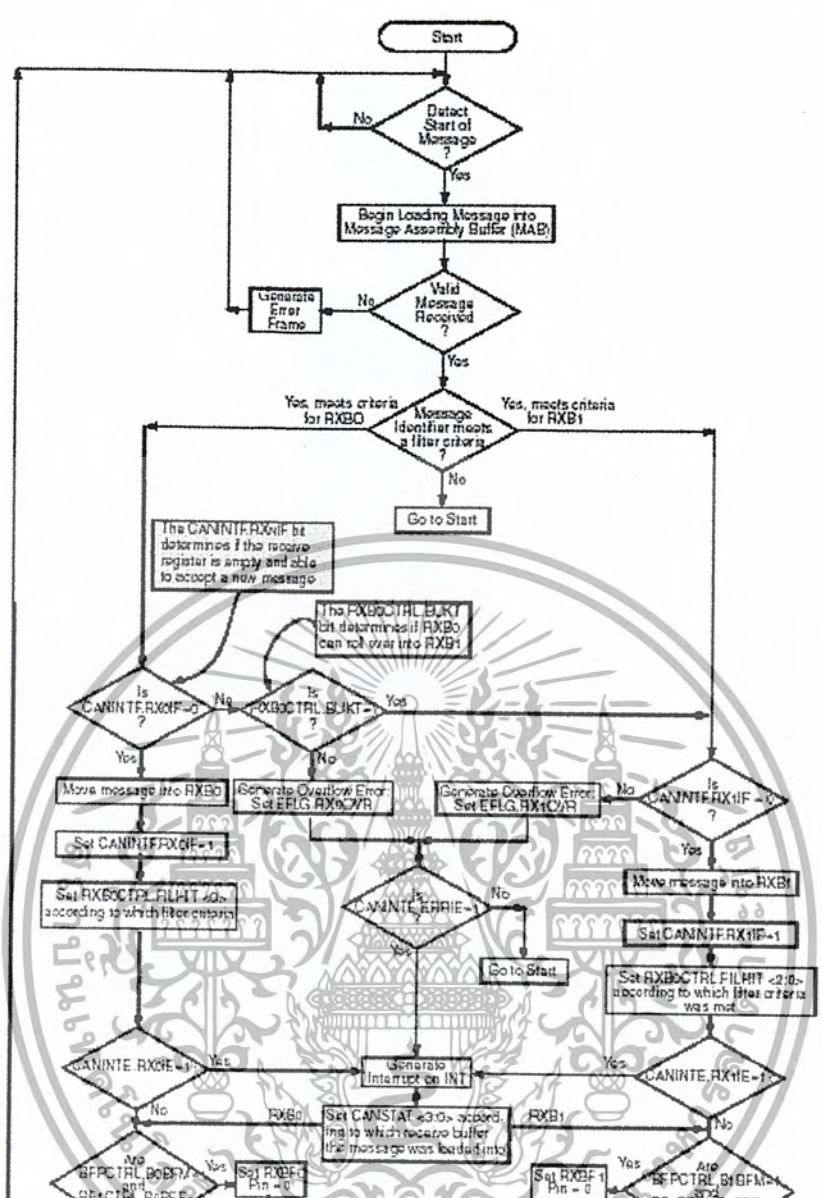
รูปที่ 5.8 SPI Input Timing



รูปที่ 5.9 SPI output Timing

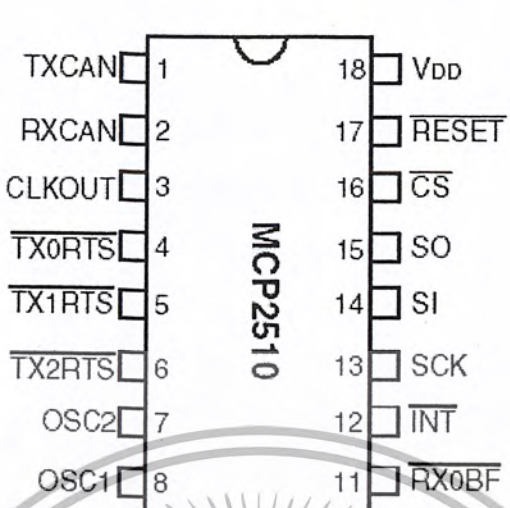


รูปที่ 5.10 Transmit Message Flow chart



รูปที่ 5.11 Message Reception Flow chart

รายละเอียดของขาไอซี MCP 2510



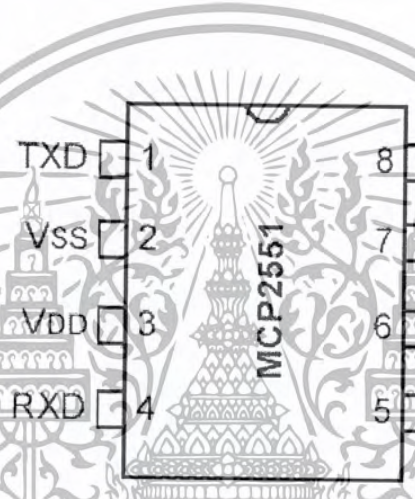
รูปที่ 5.12 แสดงขาไอซีของตัวควบคุม (CAN Controller)

- TXCAN เป็นขาส่งสัญญาณขาออก
- RXCAN เป็นขารับสัญญาณขาเข้า
- CLKOUT เป็นขาสัญญาณนาฬิกา
- $\overline{\text{TX0RTS}}$ ทำการส่งข้อมูลไปยัง TXB0 เมื่อมีการร้องขอ
- $\overline{\text{TX1RTS}}$ ทำการส่งข้อมูลไปยัง TXB1 เมื่อมีการร้องขอ
- $\overline{\text{TX2RTS}}$ ทำการส่งข้อมูลไปยัง TXB2 เมื่อมีการร้องขอ
- OSC2 เป็นขากำเนิดสัญญาณขาออก
- OSC1 เป็นขากำเนิดสัญญาณขาเข้า
- Vss ต่อลงกราวน์สำหรับอ้างอิงทางลอจิก
- $\overline{\text{RX1BF}}$ รับสัญญาณจาก RXB1
- $\overline{\text{RX0BF}}$ รับสัญญาณจาก RXB0
- $\overline{\text{INT}}$ เป็นขาอินเตอร์รัฟทางเอาท์พุท
- SCK เป็นขารับสัญญาณนาฬิกาเข้าในส่วน SPI
- SI เป็นขารับข้อมูลเข้าในส่วน SPI
- SO เป็นขาส่งข้อมูลออกในส่วน SPI
- $\overline{\text{CS}}$ เป็นขาเลือกสัญญาณเข้าในส่วนของ SPI
- $\overline{\text{RESET}}$ เป็นขาเคลียร์สถานะทำงานที่ลอจิก "0"

- Vdd ป้อนไฟบวก

5.1.2 ตัวรับส่งสัญญาณ CAN (CAN Transceiver)

MCP 2551 เป็นไอซีที่ใช้ในการอินเตอร์เฟสระหว่างไอซี MCP2510 กับบัส CAN ทำงานด้วยความเร็ว 1 Mb/s ซึ่งสามารถต่อโนดสูงสุด 112 โนด และสามารถเลือกใช้ได้ 3 สภาวะการทำงานได้แก่ โหมด High Speed ในการทำงานโหมดนี้จะต่อขา Rs กับขา Vss โหมด Slope control ในโหมดนี้จะทำการจำกัดความเร็วของขอบขาขึ้นและขอบขาลงของ CAN_H และ CAN_L และ Standby Mode ทำงานในสภาวะที่มีกระแสต่ำซึ่งทำให้ขา R XD ทำการส่งสัญญาณในอัตรการส่งที่มีความเร็วต่ำ



รูปที่ 5.13 แสดงขาไอซีของตัวรับส่งสัญญาณ (CAN Transceiver)

รายละเอียดขาของไอซี MCP 2551

- TXD จะทำงานสองสภาวะเมื่อ CAN_H และ CAN_L ซึ่งมีแรงดันแตกต่าง 2.5 V โดยที่ขา TXD อยู่ในสภาวะ “0” เมื่อ CAN_H และ CAN_L อยู่ในสภาวะโดมิแนนท์และ TXD อยู่ในสภาวะเป็น “1” เมื่อ CAN_H และ CAN_L อยู่ในสภาวะรีเซสสีฟ
- Vss ต่อกราวน์ของแหลงจ่ายไฟ
- Vdd ต่อไฟบวกของแหล่งจ่ายไฟ
- RXD จะมีสองสภาวะคือ “0” และ “1” จะทำงานสองสภาวะเมื่อ CAN_H และ CAN_L ซึ่งมีแรงดันที่แตกต่างกัน ขา RXD อยู่ในสภาวะ “0” เมื่อ CAN_H และ CAN_L อยู่ในสภาวะที่เป็นโดมิแนนท์และ RXD อยู่ในสภาวะเป็น “1” เมื่อ CAN_H และ CAN_L อยู่ในสภาวะรีเซสสีฟ
- Vref เป็นแหล่งจ่ายไฟอ้างอิง มีค่าเท่ากับ Vdd/2
- CANL ทำหน้าที่ขับสัญญาณทางด้านบัส CAN_L

- CANH ทำหน้าที่ขับสัญญาณทางด้านบัส CAN_H
- Rs ทำหน้าที่เลือกโหมดการทำงานของไอซี MCP 2551

5.1.3 ไมโครคอลโทรลเลอร์

คุณสมบัติของ SPI ในไมโครคอนโทรลเลอร์ AT89S53

- มีการจัดการสื่อสารข้อมูลแบบฟูลดูเพล็กซ์ ใช้สายสัญญาณในการถ่ายทอดข้อมูล 3 เส้น ในลักษณะซิงโครนัส (3-wire synchronous data transfer)
- สามารถทำงานเป็นได้ทั้งอุปกรณ์มาสเตอร์และ สเลฟ
- ความถี่ของการถ่ายทอดข้อมูลสูงสุด 1.5 MHz
- สามารถเลือกให้ถ่ายทอดข้อมูลในบิต LSB หรือ MSB ก่อนได้
- เลือกอัตราการถ่ายทอดข้อมูลได้ 4 อัตรา
- สามารถสร้างสัญญาณอินเตอร์รัพต์เมื่อการถ่ายทอดข้อมูลเสร็จสิ้นลง
- มีการป้องกันการเขียนข้อมูลชนกัน
- เมื่อมีการทำงานเป็นอุปกรณ์สเลฟสามารถออกจากโหมดประหยัดพลังงานแบบไอเดิล ได้เมื่อมีการร้องขอให้ติดต่อกับระบบ SPI

ขาสัญญาณของการเชื่อมต่อแบบ SPI

มีด้วยกัน 4 ขา คือ

1. ขา MOSI (Master Out Slave In) สำหรับอุปกรณ์มาสเตอร์ขานี้จะเป็นขาข้อมูลออก สำหรับส่งไปยังอุปกรณ์สเลฟ ในขณะที่ถ้าเป็นอุปกรณ์สเลฟขานี้จะจะเป็นขาข้อมูลเข้า
2. ขา MISO (Master In Save Out) สำหรับอุปกรณ์มาสเตอร์ ขานี้จะเป็นขาข้อมูลเข้าจากอุปกรณ์สเลฟ ในขณะที่เป็นอุปกรณ์สเลฟ ขานี้จะเป็นขาข้อมูลส่งออกไปยังอุปกรณ์มาสเตอร์
3. ขา SCK (SPI clock) สำหรับอุปกรณ์มาสเตอร์ ขานี้จะเป็นขาส่งสัญญาณนาฬิกาออกไปยังอุปกรณ์สเลฟเพื่อกำหนดจังหวะของการถ่ายทอดข้อมูลให้ตรงกัน ในขณะที่เป็นอุปกรณ์สเลฟ ขานี้จะเป็นขารับสัญญาณนาฬิกาจากอุปกรณ์มาสเตอร์
4. ขา SS (Slave Select) ใช้เป็นขาเลือกอุปกรณ์สเลฟในกรณีที่มีการต่ออุปกรณ์สเลฟพ่วงกันหลายตัว ทำงานที่ลอจิก “0”

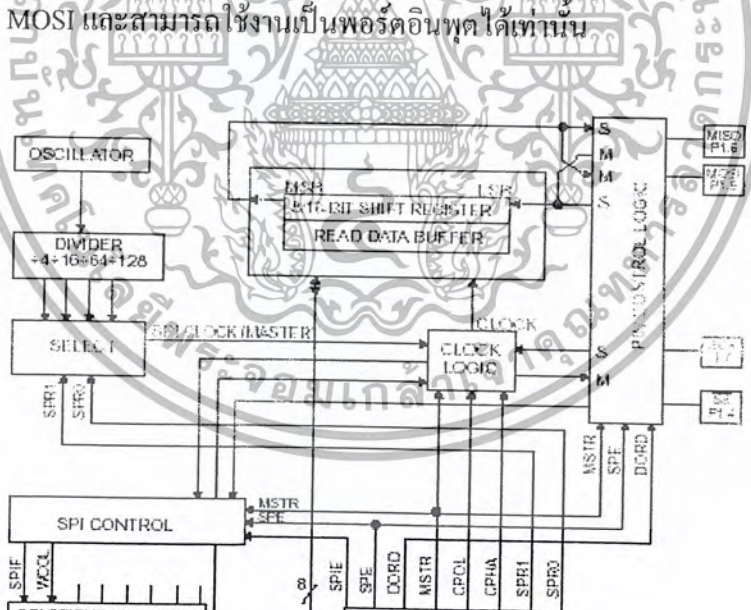
การทำงานของ SPI

ในรูปที่ 5.14 แสดงไดอะแกรมการทำงานของส่วนเชื่อมต่อ SPI ของไมโครคอลโทรลเลอร์แบบแฟลชในอนุกรม AT89S53 จะเห็นได้ว่าหัวใจการทำงานอยู่ที่รีจีสเตอร์ควบคุม SPI (Serial Peripheral Interface) สำหรับการเชื่อมต่อสายสัญญาณในระบบของ SPI ระหว่างอุปกรณ์มาสเตอร์และสเลฟแสดงดังรูปที่ 5.15

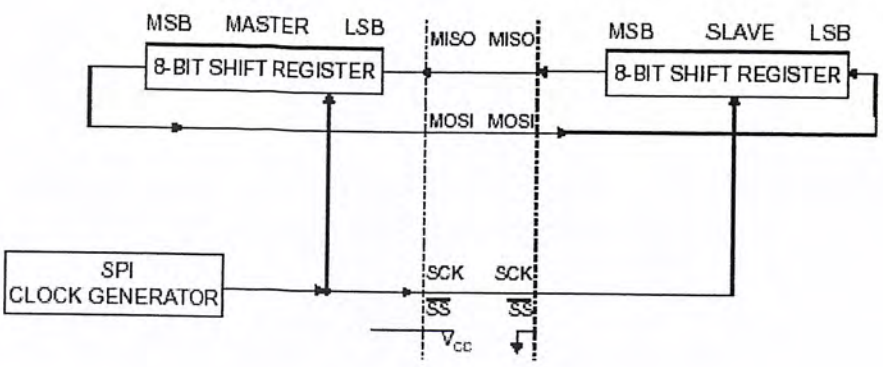
ในการอธิบายการทำงานของ SPI จะใช้รูปที่ 5.14 และรูปที่ 5.15 ประกอบกันเมื่อเริ่มต้นการทำงานขา SCK จะส่งสัญญาณนาฬิกาจากอุปกรณ์มาสเตอร์ไปยังอุปกรณ์สเลฟ โดยสัญญาณนาฬิกานี้ได้มาจากวงจรออสซิลเลเตอร์ภายในผ่านการหารค่าความถี่ด้วยค่า 4, 1, 64 หรือ 128 ขึ้นอยู่กับการกำหนดในบิต SPR0 และ SPR1 ในรีจีสเตอร์ควบคุม SPI จากนั้นสัญญาณนาฬิกาก็จะถูกควบคุมการรับเข้าหรือส่งออกโดยวงจรควบคุมสัญญาณนาฬิกา

การเขียนข้อมูลลงในรีจีสเตอร์ของอุปกรณ์สเลฟ เริ่มด้วยซีพียูทำการกระตุ้นให้วงจรกำเนิดสัญญาณนาฬิกาทำงาน ข้อมูลที่ต้องการเขียนจะออกจากอุปกรณ์มาสเตอร์ไปยังอุปกรณ์สเลฟ ซึ่งในที่นี้คือไมโครคอลโทรลเลอร์ในลักษณะอนุกรมต่อผ่านทางขา MOSI ไปเข้ายังขา MOSI ของอุปกรณ์สเลฟ หลังจากเลื่อนข้อมูลครบหนึ่งไบต์ วงจรกำเนิดสัญญาณนาฬิกาส่วนของ SPI จะหยุดทำงานแล้วทำการเซตบิต SPIF เพื่อแจ้งการถ่ายทอดข้อมูลสิ้นสุดลง ถ้าหากบิต SPIE ในรีจีสเตอร์ควบคุม SPI (SPCR) และบิต ES ในรีจีสเตอร์ SCON ได้รับการเซตไว้ด้วย จะเป็นการอีนาเบิลให้เกิดการอินเตอร์รัพต์ขึ้น

ขา SS เป็นขาสำหรับเลือกอุปกรณ์สเลฟ ถ้ามีการเชื่อมต่ออุปกรณ์สเลฟตัวใดเข้าสู่ระบบการติดต่อ SPI ที่ขานี้ต้องได้รับลอจิก "0" ในกรณีเป็นอุปกรณ์มาสเตอร์ต้องทำให้ที่ขานี้เป็นลอจิก "1" หากอุปกรณ์สเลฟตัวใดที่ขานี้ได้รับลอจิก "1" จะเป็นการดิสเอเบิลการติดต่อกับระบบ SPI ในทันทีและขา MOSI และสามารถใช้งานเป็นพอร์ตอินพุตได้เท่านั้น



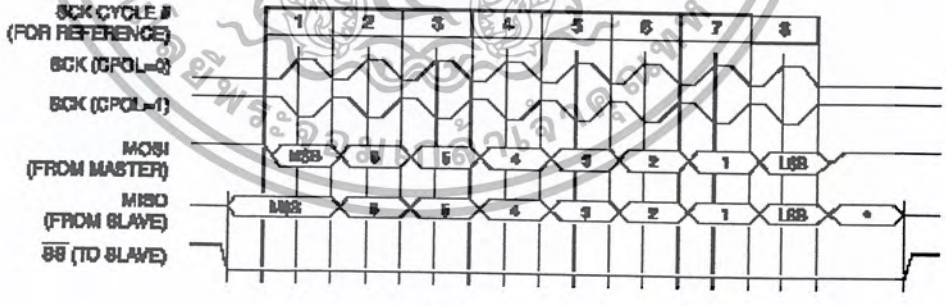
รูปที่ 5.14 ไดอะแกรมการทำงานของส่วน SPI ภายในไมโครคอลโทรลเลอร์ AT 89S53



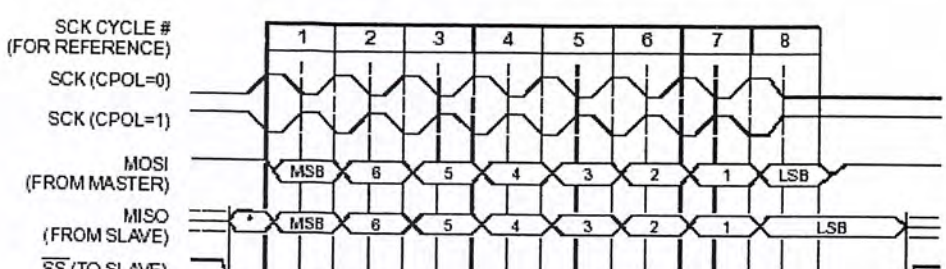
รูปที่ 5.15 การเชื่อมต่อสายสัญญาณใน SPI ระหว่างอุปกรณ์มาสเตอร์และสเลฟ

ในรูปที่ 5.16 และ 5.17 เป็นไดอะแกรมเวลาของรูปแบบการถ่ายทอดข้อมูลในส่วน SPI ซึ่งได้รับการกำหนดโดย CPHA และ CPOL ในรีจีสเตอร์ควบคุม SPI

เมื่อนำในไมโครคอลโทรลเลอร์แบบแฟลชในอนุกรม AT89S53 มาใช้งานในด้าน SPI พอร์ต 1.4-1.7 ซึ่งก็คือขา SS, MOSI, MISO และ SCK ไม่ควรนำมาใช้ในงานอื่นอีก ซึ่งอาจส่งผลกระทบต่อการจัดสรรพอร์ตของไมโครคอลโทรลเลอร์ MCS-51 บ้างแต่ก็ไม่มากนัก เนื่องจากในไมโครคอลโทรลเลอร์ตระกูลนี้มีจำนวนขาพอร์ตให้ใช้ถึง 32 บิต นำไปใช้ในงาน SPI เพียง 4 บิต ยังเหลือให้ใช้อีก 28 บิต ซึ่งน่าจะเพียงพอสำหรับการสร้างระบบควบคุมที่ทรงประสิทธิภาพโดยเฉพาะอย่างยิ่งการใช้หน่วยความจำโปรแกรมภายในไมโครคอลโทรลเลอร์ จะยิ่งทำให้การใช้งานพอร์ตสามารถทำได้อย่างเต็มที่



รูปที่ 5.16 ไดอะแกรมเวลาของการถ่ายทอดข้อมูลในส่วน SPI เมื่อบิต CPHA เป็น "0"



รูปที่ 5.17 ไดอะแกรมเวลาของการถ่ายทอดข้อมูลในส่วน SPI เมื่อบิต CPHA เป็น "1"

รีจีสเตอร์ที่เกี่ยวข้องกับการทำงานของส่วน SPI ในไมโครคอลโทรลเลอร์

การทำงานของส่วนเชื่อมต่ออุปกรณ์อนุกรม SPI ของไมโครคอลโทรลเลอร์แบบแฟลชอนุกรม AT89S53 มีรีจีสเตอร์ที่เกี่ยวข้องทั้งสิ้น 3 ตัว ดังมีรายละเอียดต่อไปนี้

1. รีจีสเตอร์ข้อมูล SPI หรือ SPDR (SPI Data Register)

มีแอดเดรดอยู่ที่ 86H ในพื้นที่รีจีสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิต โดยสามารถเข้าถึงในระดับบิต ใช้ในการเก็บข้อมูลที่เกิดการถ่ายทอดในส่วนการทำงาน SPI ชื่อเรียกบิตต่าง ๆ ภายในรีจีสเตอร์ SPDR เรียงลำดับจากบิต MSB คือ บิต SPD7-SPD0 รีจีสเตอร์ตัวนี้มีความพิเศษคือ เมื่อเกิดการรีเซต ค่าที่อยู่ในรีจีสเตอร์ SPDR นี้จะไม่มีการเปลี่ยนแปลง

2. รีจีสเตอร์ควบคุม SPI หรือ SPCR (SPI Control Register)

มีแอดเดรดอยู่ที่ D5H ในพื้นที่ของรีจีสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิต สามารถเข้าถึงในระดับบิต เมื่อเกิดการรีเซตค่าของรีจีสเตอร์ตัวนี้เป็น 000001XXB (เมื่อ X หมายถึงค่าเดิมก่อนหน้าเกิดการรีเซต) มีรายละเอียดดังนี้

Bit	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
	7	6	5	4	3	2	1	0

SPIE (SPI Interrupt Enable) : ใช้อีนาเบิลอินเตอร์รัพต์อันเนื่องมาจากการถ่ายทอดของ SPI เสร็จสิ้นลง บิตนี้ต้องทำงานร่วมกับ SPIF ในรีจีสเตอร์ SPFR และบิต ES เป็น "1" ทั้งหมด ถ้าบิต SPIE เป็น "0" จะเป็นการดิสเอเบิลการอินเตอร์รัพต์ในระบบ SPI

SPE (SPI Enable) : ใช้อีนาเบิลการทำงานของ SPI เมื่อบิตนี้เซตค่าเป็น "1" วงจรควบคุมภายในไมโครคอลโทรลเลอร์อนุกรม AT89S53 จะทำการเชื่อมต่อวงจร SS, MOSI, และ SCK เข้ากับขา P1.4, P1.5, P1.6 และ P1.7 ของพอร์ต "1" ทันที เพื่อเตรียมการทำงานของ SPI

DORD (Data Order) : ใช้เลือกการถ่ายทอดข้อมูลของ SPI

"0" เลือกการถ่ายทอดข้อมูลในบิต MSB ก่อน

"1" เลือกการถ่ายทอดข้อมูลในบิต LSB ก่อน

MSTR (Master/Slave Select) : ใช้เลือกโหมดการทำงาน SPI ของอุปกรณ์

"0" เลือกการทำงานเป็นอุปกรณ์สเลฟ

"1" เลือกการทำงานเป็นอุปกรณ์มาสเตอร์

CPOL (Clock Polarity) : ใช้ร่วมกับ บิต CPHA ในการกำหนดจังหวะการทำงานของสัญญาณนาฬิกา SCK

"0" ทำให้สัญญาณนาฬิกา SCK ของอุปกรณ์มาสเตอร์เป็นลอจิก "0" เมื่อไม่มีการส่งข้อมูล

"1" ทำให้สัญญาณนาฬิกา SCK เป็น "1" เมื่ออยู่ในสภาวะไอเดิล

CPHA (Clock Phase) : ใช้ร่วมกับ บิต CPOL ในการกำหนดรูปแบบของสัญญาณนาฬิกา SCK

SPR1-SPR0 (SPI Clock Rate Select) : ใช้กำหนดอัตราของสัญญาณนาฬิกา SCK การกำหนดนี้เองต้องกระทำที่อุปกรณ์มาสเตอร์เท่านั้น

"00" อัตราสัญญาณนาฬิกามาจาก ความถี่สัญญาณนาฬิกาหลักหาร 4

"01" อัตราสัญญาณนาฬิกามาจาก ความถี่สัญญาณนาฬิกาหลักหาร 16

"10" อัตราสัญญาณนาฬิกามาจาก ความถี่สัญญาณนาฬิกาหลักหาร 64

"11" อัตราสัญญาณนาฬิกามาจาก ความถี่สัญญาณนาฬิกาหลักหาร 128

3. รีจีสเตอร์สถานะ SPI หรือ SPSR (SPI Status register)

มีแอดเดรสอยู่ที่ AAH ในพื้นที่ของรีจีสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิตสามารถเข้าถึงในระดับบิต มีการใช้งานเพียง 2 บิต เมื่อเกิดการรีเซตค่าของรีจีสเตอร์ตัวนี้จะเป็น 00XXXXXXB (เมื่อ X หมายถึงค่าเดิมก่อนหน้าเกิดการรีเซต) รายละเอียดของการใช้งานมีดังนี้

	SPIF	WCOL	—	—	—	—	—	—
Bit	7	6	5	4	3	2	1	0

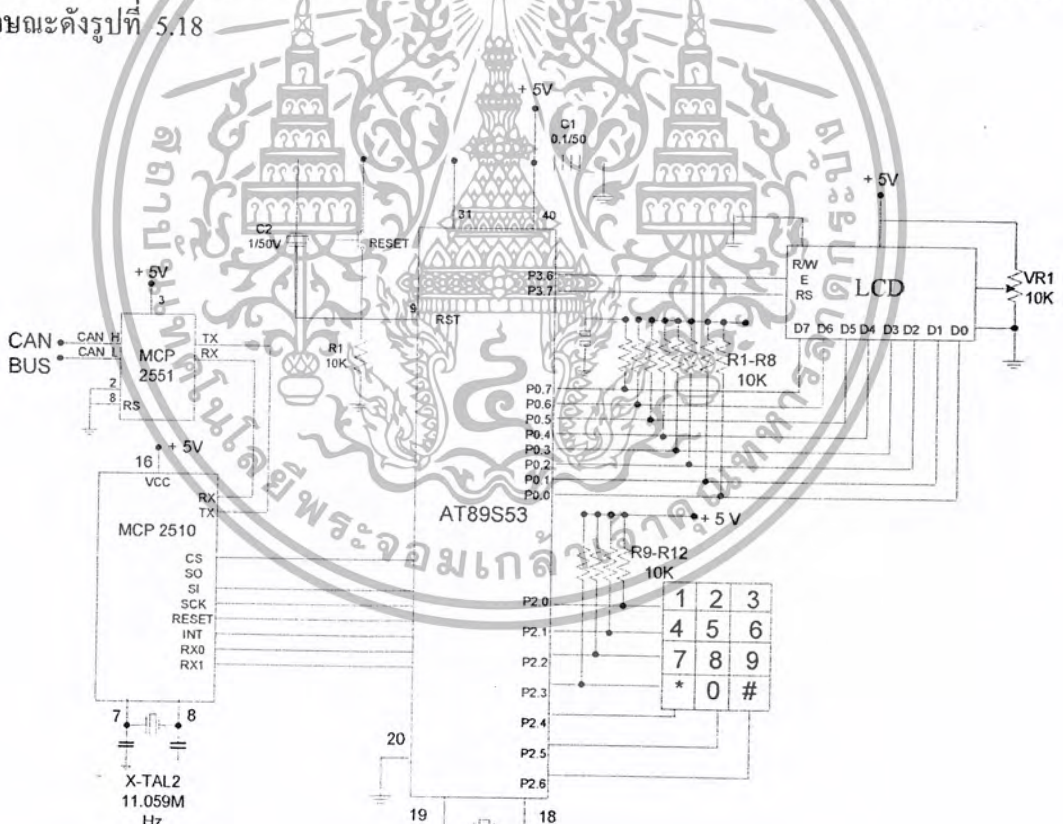
SPIF (SPI Interrupt Flag) : เมื่อเกิดการถ่ายทอดข้อมูลของ SPI เสร็จสิ้นลง บิตนี้จะเซต และจะทำให้เกิดการอินเตอร์รัพต์ขึ้นถ้าบิต SPIE ในรีจีสเตอร์ SPSR และบิต ES ในรีจีสเตอร์ IE เป็น "1" การเคลียร์บิตนี้ทำได้ด้วยกระบวนการทางซอฟต์แวร์โดยการอ่านค่าของรีจีสเตอร์ SPSR และการเข้าถึงข้อมูลในรีจีสเตอร์ข้อมูล SPI หรือ SPDR

WCOL (Write Collision Flag) : ใช้แสดงการชนกันของข้อมูล เหตุการณ์นี้จะเกิดก็ต่อเมื่อรีจีสเตอร์ข้อมูล SPI หรือ SPDR ได้รับการเขียนขณะมีการถ่ายทอดข้อมูลอยู่ทำให้ค่าของรีจีสเตอร์ SPDR ที่อ่านได้มีค่าไม่ถูกต้อง บิตนี้จะเซตเมื่อเกิดเหตุการณ์ดังกล่าวขึ้น สามารถเคลียร์ได้ด้วย

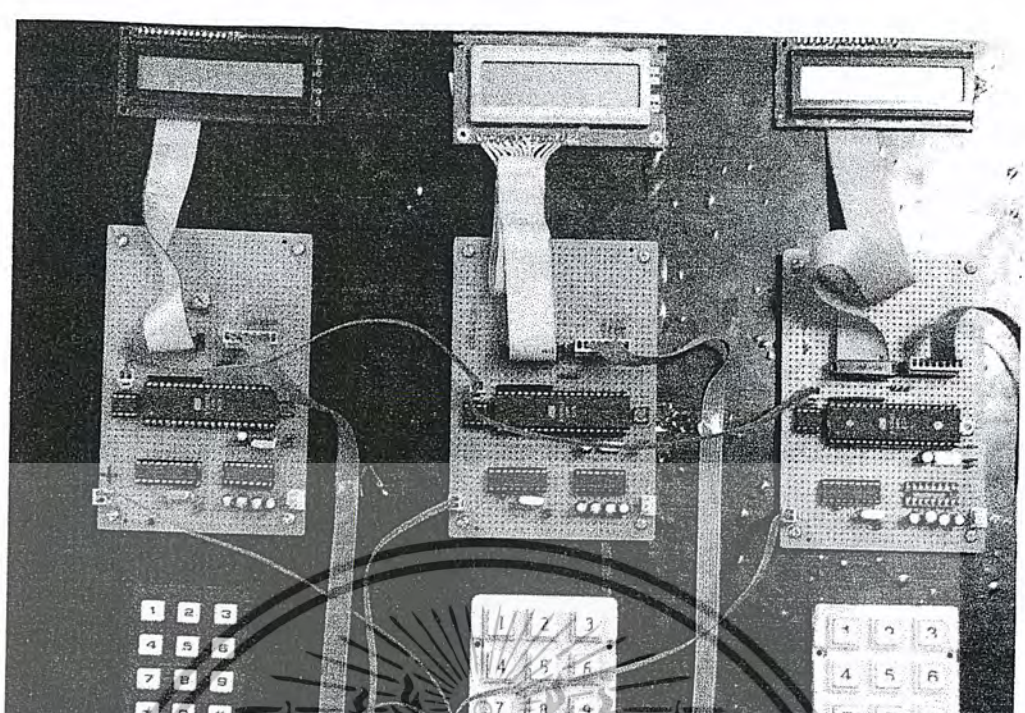
กระบวนการทางซอฟต์แวร์โดยการอ่านค่าจาก SPSR หลังเกิดการเซตบิตนี้ และโดยการเข้าถึงรีจีสเตอร์ SPDR

5.2 การสร้างบอร์ดอินเตอร์เฟส CAN

ในปัจจุบันมีบริษัทผู้ผลิตไอซีหลายรายผลิตไอซี CAN คอลโทรลเลอร์ออกมาจำหน่ายมากมายและได้พัฒนาให้สามารถจัดการในเรื่องโปรโตคอลได้อย่างมีประสิทธิภาพมากขึ้นรวมถึงได้เพิ่มฟังก์ชั่นการทำงานให้มีความยืดหยุ่นและสะดวกในการโปรแกรมอีกด้วย บอร์ดอินเตอร์เฟสที่ใช้ในการทดลองนี้จะใช้ไอซีเบอร์ MCP 2510 เป็นไอซีตัวควบคุม CAN ซึ่งมีส่วนติดต่อ SPI อยู่ในตัวไอซีซึ่งสามารถติดต่อกับไมโครคอลโทรลเลอร์ได้โดยตรงโดยไมโครคอลโทรลเลอร์ที่ใช้จะต้องมีส่วนติดต่อ SPI ซึ่งในการทดลองใช้ไอซีเบอร์ AT89S53 และไอซีตัวรับส่งสัญญาณ (CAN Transceiver) ใช้ไปซีเบอร์ MCP 2551 โดยบอร์ดอินเตอร์เฟสนี้ถูกออกแบบให้มีลักษณ์เป็นโมดูลของ CAN มีความยืดหยุ่นในการใช้งาน เพื่อความสะดวกในการศึกษาและพัฒนา วงจรเบื้องต้นจะมีลักษณะดังรูปที่ 5.18



รูปที่ 5.18 วงจรของบอร์ดอินเตอร์เฟส CAN



รูปที่ 5.19 รูปของบอร์ดอินเตอร์เฟส CAN

5.2.1 การทดลองบอร์ดอินเตอร์เฟส CAN

บอร์ดการทดลองจะใช้การรับค่าหมายเลข ID จาก Key Pad โดยกำหนดให้มี ID สามหมายเลขคือ ID=0000 มีข้อมูลเป็น AA ID=0001 มีข้อมูลเป็น BB และID=0002 มีข้อมูลเป็น CC โดยการทดลองจะกำหนดให้มีโนดใดโนดหนึ่งเป็นมาสเตอร์ ส่วนโนดที่เหลือก็จะเป็นสเลฟ เมื่อเริ่มเปิดเครื่องจะปรากฏข้อความว่า

CAN MODULE V1.1

ID (0-2047) :

โปรแกรมจะให้ป้อนหมายเลข ID โดยป้อนค่า ID ทั้งสามข้างต้น เมื่อป้อยหมายเลข ID เรียบร้อยจะปรากฏข้อความที่มาสเตอร์ว่า

ID : XXXX

TX DATA : XX

โดยโนดที่เป็นสเลฟที่มีหมายเลข ID ตรงกันจะปรากฏข้อความว่า

ID : XXXX

RX DATA : XX

5.2.2 ผลการทดลอง

- เมื่อทำการทดลองป้อนหมายเลข ID=0000 จะปรากฏข้อความที่มาสเตอร์ (โนดที่ 1) ว่า

ID : 0000

TX DATA : 55

โนดที่เป็นสเลฟ (โนดที่ 2 และ 3) ที่มีหมายเลข ID ตรงกันจะปรากฏข้อความว่า

ID : 0000

RX DATA : 55

- เมื่อทำการทดลองป้อนหมายเลข ID=0001 จะปรากฏข้อความที่มาสเตอร์ (โนดที่ 2) ว่า

ID : 0001

TX DATA : 66

โนดที่เป็นสเลฟ (โนดที่ 1 และ 3) ที่มีหมายเลข ID ตรงกันจะปรากฏข้อความว่า

ID : 0001

RX DATA : 66

- เมื่อทำการทดลองป้อนหมายเลข ID=0002 จะปรากฏข้อความที่มาสเตอร์ (โนดที่ 3) ว่า

ID : 0002

TX DATA : 77

โนดที่เป็นสเลฟ (โนดที่ 1 และ 2) ที่มีหมายเลข ID ตรงกันจะปรากฏข้อความว่า

ID : 0002

RX DATA : 77

บทที่ 6
สรุปผลการทดลอง

การสื่อสารโดยใช้ระบบเครือข่าย CAN เหมาะสำหรับการควบคุมอุปกรณ์อิเล็กทรอนิกส์ที่ต่อบนบัส ซึ่งมีการทำงานแบบเวลาจริง (Realtime) และต้องการความปลอดภัยของข้อมูลสูง เพื่อให้ระบบมีความน่าเชื่อถือและมีประสิทธิภาพในการทำงานมากขึ้น ในขณะที่เป็นการลดความยุ่งยากซับซ้อน และลดจำนวนสายสัญญาณในการเชื่อมต่อระหว่างอุปกรณ์ในส่วนต่าง ๆ ลง ด้วยเหตุนี้ CAN จึงเป็นที่นิยมอย่างแพร่หลายในเวลาอันสั้น นอกจากการใช้งานในระบบอิเล็กทรอนิกส์ในรถยนต์แล้วยังมีการประยุกต์ใช้งานอื่น ๆ โดยเฉพาะในงานควบคุมอัตโนมัติในอุตสาหกรรม

จากการทดลองเพื่อทดสอบสมรรถภาพในการสื่อสารข้อมูลในระบบ CAN ทำให้เข้าใจถึงหลักการทำชัดเจนขึ้น ทั้งในด้านคุณสมบัติและข้อกำหนดโปรโตคอล จากการทดลองจะได้ข้อมูลที่สำคัญที่เกี่ยวกับการเขียนโปรแกรมสำหรับสื่อสารข้อมูลในระบบ CAN ซึ่งเป็นอัลกอลิทึมเบื้องต้นในการโปรแกรมควบคุมโปรโตคอลของ CAN เพื่อเป็นข้อมูลและแนวทางแก่ผู้ที่สนใจที่จะนำ CAN ไปพัฒนาต่อไป

บรรณานุกรม

- http://www.es.co.th : เป็นเว็บไซต์ที่ให้ข้อมูลเกี่ยวกับไมโครคอลโทรลเลอร์ (AT89S53)
- http://www.keil.com: เป็นเว็บไซต์ที่ให้ข้อมูลเกี่ยวกับ CAN ในเรื่องของการคอมไพเลอร์ และตัวอย่างการเขียนโปรแกรมภาษา C51
- http://www.microchip.com : เป็นเว็บไซต์ที่ให้ข้อมูลเกี่ยวกับ CAN ในเรื่องของตัวควบคุม CAN (CAN Controller) MCP 2510 และตัวรับส่ง CAN (CAN Transceiver) MCP 2551

ภาคผนวก

Source Program

```
/*************************CAN.c*******************/
#include <scankey4x3_h.h>
#include <xlcd.h>
#include <can_x.h>
#include <AT89S53.H>
bit scr_normal=1;
unsigned char *init="CAN MODULE V1.1";
unsigned char *enter="ID(0-2047):";
unsigned char *id_print="ID:";
unsigned char *data_r="RX DATA:";
unsigned char *data_t="TX DATA:";
unsigned char sidh;
unsigned char sidl;
unsigned char aa = 1 ;
unsigned char id_buf[4];
unsigned char data_buf[2];
/*****************************************************************************/
/*****************************Function Delay Time *****************************/
/************** convert ascii to LCD ********************/
void sent_to_lcd(unsigned char value)
{
        unsigned char buf=0;
        buf = value & 0xf0;
        buf = (buf>>4)|(0x30);
        lcd_text(buf);
        buf=value & 0x0f;
        buf=buf|0x30;
        lcd_text(buf);
        }
```

```
/***************/
void scr_init(void)
{
unsigned char i;
lcd_clear();
//        lcd_command(0x0c);
lcd_command(0x80);
for(i=0;i<15;i++)
lcd_text(*(init+i));
lcd_jumporigin();
//        lcd_command(0x0c);
lcd_command(0xc0);
for(i=0;i<11;i++)
        lcd_text(*(enter+i));
}
void scr_RX_id(void)
{
unsigned char i;
lcd_clear();
lcd_command(0x0c);
lcd_command(0x80);
for(i=0;i<3;i++)
{
lcd_text(*(id_print+i));
}
lcd_command(0x0c);
lcd_command(0x84);
sent_to_lcd(sidh);
lcd_command(0x0c);
lcd_command(0x86);
sent_to_lcd(sidl);
}
```

```
void scr_TX_id(void)
        {
        unsigned char i;
//      lcd_clear();
//      lcd_command(0x0c);
        lcd_command(0x80);
        for(i=0;i<3;i++)
        lcd_text(*(id_print+i));
//      lcd_command(0x0c);
        lcd_command(0x84);
        sent_to_lcd(sidh);
//      lcd_command(0x0c);
        lcd_command(0x86);
        sent_to_lcd(sidl);
        //      lcd_command(0x0c);
        lcd_command(0xc0);
        for(i=0;i<8;i++)
                lcd_text(*(data_t+i));
                delay(20);
        }

        //*************************************************************************//
        //*************************Function main*********************************//
        //*************************************************************************//

        void main(void)
        {
        unsigned char intf;
        unsigned char _data,_data1,i;
        unsigned char push_key,key;
        unsigned char countto_4 = 1;
        unsigned char countto_2 = 1;
```

```
unsigned char count_f2  = 1;
unsigned char count_tokey = 0;
unsigned char count_tokey_f2 = 0;
unsigned char datk,x1,x2,x3;
unsigned char countloop2 = 1;
unsigned char loop_1 = 1;
unsigned char send_data = 0;
P0=0x00;
can_init();
lcd_init();
scr_init();
can_modify();
        while(countto_4)
        {
                push_key=scankey();
                if(push_key!=0xff)
                {
                    count_tokey++;
                    if(count_tokey == 1)
                    {
                            id_buf[1] = push_key;
                            datk = push_key;
                            x1   = datk << 4;
                            lcd_command(0xcc);
                            push_key = push_key | 0x30;
                            lcd_text(push_key);
                    }
                    if(count_tokey == 2)
                    {
                            id_buf[2] = push_key;
                            sidh = x1 | push_key;
                            lcd_command(0xcd);
```

```
                push_key = push_key | 0x30;
                lcd_text(push_key);
            }
            if(count_tokey == 3)
            {
                id_buf[3] = push_key;
                datk = push_key;
                x2   = datk << 4;
                lcd_command(0xce);
                push_key = push_key | 0x30;
                lcd_text(push_key);
            }
            if(count_tokey == 4)
            {
                id_buf[4] = push_key;
                sidl = x2 | push_key;
                lcd_command(0xcf);
                push_key = push_key | 0x30;
                lcd_text(push_key);
                countto_4 = 0;
            }
        }
    }
//*****************************************************************//
//*************************send*************************************//

write_byte(TXB0SIDH,sidh);
write_byte(TXB0SIDL,sidl);

//*****************************************************************///
//***********************Total loop********************************//
//*****************************************************************//
```

```
while(1)
{
scr_RX_id();
count_tokey=0;
push_key = scankey();
if(push_key == 0xff)
{
        intf=read_byte(CANINTF);
        if((intf&0x01)==0x01)
        {
                _data = read_byte(RXB0DB0);
                lcd_command(0x0c);
                lcd_command(0xc0);
                for(i=0;i<8;i++)
                lcd_text(*(data_r+i));
                lcd_command(0x0c);
                lcd_command(0xc9);
                sent_to_lcd(_data);
                delay(3000);
                bit_modify(CANINTF,0x01,0);
        }
        if((intf&0x04)==0x04)
        {
                bit_modify(CANINTF,0x04,0);
        }
}
else
{
        scr_TX_id();
        countto_2=1;
        while(countto_2)
        {
```

```
key = scankey();
if(key != 0xff)
{
        count_tokey++;
                if(count_tokey == 1)
                {
                        data_buf[1] = key;
                        datk = key;
                        x3 = datk << 4;
                        lcd_command(0xc9);
                        key =key | 0x30;
                        lcd_text(key);
                }

                if(count_tokey == 2)
                {
                        data_buf[2] = key;
                        lcd_command(0xca);
                        key = key | 0x30;
                        lcd_text(key);
                        countto_2 = 0;
                        _data1 = x3 | key;
                        for(i=0;i<3;i++)
                {
                        write_byte(TXB0DB0,_data1);
                        request_to_send(0x81);
                }
        }
}

}
}
```

```
}
}

/*-------------------------------------------------------------------
AT89S53.H

Header file for the Atmel AT89S53.
Copyright 1996-1997 Keil Elektronik GmbH and Keil Software, Inc.
All rights reserved.
-------------------------------------------------------------------*/

#ifndef AT89S53_HEADER_FILE
#define AT89S53_HEADER_FILE 1

/*-------------------------------------------------
Byte Registers

Note: Only Registers located on addresses that
are evenly divisible by 8 are bit addressable.
All other registers use bit masks.
-------------------------------------------------*/
sfr P0    = 0x80;   /* Port 0 */
sfr SP    = 0x81;   /* Stack Pointer */
sfr DPL   = 0x82;   /* Data Pointer Low Byte */
sfr DP0L  = 0x82;   /* Alternate Definition */
sfr DPH   = 0x83;   /* Data Pointer High Byte */
sfr DP0H  = 0x83;   /* Alternate Definition */
sfr DP1L  = 0x84;   /* Data Pointer 1 Low Byte */
sfr DP1H  = 0x85;   /* Data Pointer 1 High Byte */
sfr SPDR  = 0x86;   /* SPI Data Register */
sfr PCON  = 0x87;   /* Power Control Register */
sfr TCON  = 0x88;   /* Timer Control Register */
```

```
sfr TMOD   = 0x89;   /* Timer Mode Control Register */
sfr TL0    = 0x8A;   /* Timer 0 Low Byte */
sfr TL1    = 0x8B;   /* Timer 1 Low Byte */
sfr TH0    = 0x8C;   /* Timer 0 High Byte */
sfr TH1    = 0x8D;   /* Timer 1 High Byte */
sfr P1     = 0x90;   /* Port 1 */
sfr WMCON  = 0x96;   /* Watchdog and Memory Control Register */
sfr SCON   = 0x98;   /* Serial Port Control */
sfr SBUF   = 0x99;   /* Serial Port Buffer */
sfr P2     = 0xA0;   /* Port 2 */
sfr IE     = 0xA8;   /* Interrupt Enable Register 0 */
sfr SPSR   = 0xAA;   /* SPI Status Register */
sfr P3     = 0xB0;   /* Port 3 */
sfr IP     = 0xB8;   /* Interrupt Priority Register */
sfr T2CON  = 0xC8;   /* Timer 2 Control */
sfr T2MOD  = 0xC9;   /* Timer 2 Mode */
sfr RCAP2L = 0xCA;   /* Timer 2 Capture Low Byte */
sfr RCAP2H = 0xCB;   /* Timer 2 Capture High Byte */
sfr TL2    = 0xCC;   /* Timer 2 Low Byte */
sfr TH2    = 0xCD;   /* Timer 2 High Byte */
sfr SPCR   = 0xD5;   /* SPI Control Register */
sfr ACC    = 0xE0;   /* Accumulator */
sfr B      = 0xF0;   /* B Register */

/*-----------------------------------------------
P0 (0x80) Bit Registers
-----------------------------------------------*/
sbit P0_0 = 0x80;
      sbit P0_1 = 0x81;
      sbit P0_2 = 0x82;
      sbit P0_3 = 0x83;
      sbit P0_4 = 0x84;
```

```
sbit P0_5 = 0x85;
sbit P0_6 = 0x86;
sbit P0_7 = 0x87;

#define T0_CT_   0x04   /* Timer 0 Counter/Timer Select: 0=Counter, 1=Timer */
#define T0_GATE_ 0x08   /* Timer 0 Gate Control */
#define T1_M0_   0x10   /* Timer 1 Mode Bit 0 */
#define T1_M1_   0x20   /* Timer 1 Mode Bit 1 */
#define T1_CT_   0x40   /* Timer 1 Counter/Timer Select: 0=Counter, 1=Timer */
#define T1_GATE_ 0x80   /* Timer 1 Gate Control */

#define T1_MASK_ 0xF0   /* Timer 0 Mask */
#define T0_MASK_ 0x0F   /* Timer 1 Mask */

/*--------------------------------------
P1 (0x90) Bit Registers
--------------------------------------*/
sbit P1_0 = 0x90;
sbit P1_1 = 0x91;
sbit P1_2 = 0x92;
sbit P1_3 = 0x93;
sbit P1_4 = 0x94;
sbit P1_5 = 0x95;
sbit P1_6 = 0x96;
sbit P1_7 = 0x97;
sbit T2   = 0x90;      /* External input to Timer/Counter 2, clock out */
sbit T2EX = 0x91;      /* Timer/Counter 2 capture/reload trigger & dir ctl */
sbit SS   = 0x94;      /* SPI: SS - Slave port select input */
sbit MOSI = 0x95;      /* SPI: MOSI - Master data output, slave data input */
sbit MISO = 0x96;      /* SPI: MISO - Master data input, slave data output */
sbit SCK  = 0x97;      /* SPI: SCK - Master clock output, slave clock input */
```

```
/*---------------------------------------
WMCON (0x96) Bit Values
---------------------------------------*/
#define WDTEN_  0x01
#define WDTRST_ 0x02  /* Watchdog Timer Reset and EEPROM Ready,/Busy Flag*/
#define EERDY_  0x02  /* Watchdog Timer Reset and EEPROM Ready,/Busy Flag */
#define DPS_    0x04  /* Data Pointer Select: 0=DP0, 1=DP1 */
#define EEMEN_  0x08  /* Internal EEPROM Access Enable: 1=Enabled */
#define EEMWE_  0x10  /* Internal EEPROM Write Enable: 1=Enabled */
#define PS0_    0x20  /* Prescaler bit 0 for the Watchdog Timer */
#define PS1_    0x40  /* Prescaler bit 1 for the Watchdog Timer */
#define PS2_    0x80  /* Prescaler bit 2 for the Watchdog Timer */
                      /* 000 =  16ms Timeout */
                      /* 001 =  32ms Timeout */
                      /* 010 =  64ms Timeout */
                      /* 011 = 128ms Timeout */
                      /* 100 = 256ms Timeout */
                      /* 101 = 512ms Timeout */
                      /* 110 = 1024ms Timeout */
                      /* 111 = 2048ms Timeout */

/*---------------------------------------
SCON (0x98) Bit Registers
---------------------------------------*/
sbit RI  = 0x98;      /* Receive Interrupt Flag */
sbit TI  = 0x99;      /* Transmit Interrupt Flag */
sbit RB8 = 0x9A;      /* 9th data bit received */
sbit TB8 = 0x9B;      /* 9th data bit to be transmitted in modes 2 & 3 */
sbit REN = 0x9C;      /* Receive Enable */
sbit SM2 = 0x9D;      /* Serial Port Mode Bit 2 */
sbit SM1 = 0x9E;      /* Serial Port Mode Bit 1 */
sbit SM0 = 0x9F;      /* Serial Port Mode Bit 0 */
```

```
/*---------------------------------------------
P2 (0xA0) Bit Registers
---------------------------------------------*/
sbit P2_0 = 0xA0;
sbit P2_1 = 0xA1;
sbit P2_2 = 0xA2;
sbit P2_3 = 0xA3;
sbit P2_4 = 0xA4;
sbit P2_5 = 0xA5;
sbit P2_6 = 0xA6;
sbit P2_7 = 0xA7;

/*---------------------------------------------
IE (0xA8) Bit Registers
---------------------------------------------*/
sbit EX0  = 0xA8;   /* External Interrupt 0 Enable: 1=Enabled */
sbit ET0  = 0xA9;   /* Timer 0 Interrupt Enable: 1=Enabled */
sbit EX1  = 0xAA;   /* External Interrupt 1 Enable: 1=Enabled */
sbit ET1  = 0xAB;   /* Timer 1 Interrupt Enable: 1=Enabled */
sbit ES   = 0xAC;   /* SPI and UART Interrupt Enable: 1=Enabled */
sbit ET2  = 0xAD;   /* Timer 2 Interrupt Enable: 1=Enabled */
sbit EA   = 0xAF;   /* Global Interrupt Enable: 0=Disable all interrupts */

/*---------------------------------------------
SPSR (0xAA) Bit Values - Reset Value = 0000.0000
---------------------------------------------*/
#define WCOL_  0x40   /* SPI Write Collision Flag: 1=Collision */
#define SPIF_  0x80   /* SPI Interrupt Flag */
/*---------------------------------------------
P3 (0xB0) Bit Registers (Mnemonics & Ports)
---------------------------------------------*/
sbit P3_0 = 0xB0;
```

```
sbit P3_1 = 0xB1;
sbit P3_2 = 0xB2;
sbit P3_3 = 0xB3;
sbit P3_4 = 0xB4;
sbit P3_5 = 0xB5;
sbit P3_6 = 0xB6;
sbit P3_7 = 0xB7;
sbit RXD  = 0xB0;     /* Serial data input */
sbit TXD  = 0xB1;     /* Serial data output */
sbit INT0 = 0xB2;     /* External interrupt 0 */
sbit INT1 = 0xB3;     /* External interrupt 1 */
sbit T0   = 0xB4;     /* Timer 0 external input */
sbit T1   = 0xB5;     /* Timer 1 external input */
sbit WR   = 0xB6;     /* External data memory write strobe */
sbit RD   = 0xB7;     /* External data memory read strobe */

/*-----------------------------------------
IP (0xB8) Bit Registers
-----------------------------------------*/
sbit PX0  = 0xB8;     /* External Interrupt 0 Priority Bit */
sbit PT0  = 0xB9;     /* Timer 0 Interrupt Priority Bit */
sbit PX1  = 0xBA;     /* External Interrupt 1 Priority Bit */
sbit PT1  = 0xBB;     /* Timer 1 Interrupt Priority Bit */
sbit PS   = 0xBC;     /* Serial Port Interrupt Priority Bit */
sbit PT2  = 0xBD;     /* Timer 2 Interrupt Priority Bit */

/*-----------------------------------------
T2CON (0xC8) Bit Registers
-----------------------------------------*/
sbit CP_RL2= 0xC8;    /* 0=Reload, 1=Capture select */
sbit C_T2 = 0xC9;     /* 0=Timer, 1=Counter */
sbit TR2  = 0xCA;     /* 0=Stop timer, 1=Start timer */
```

```
sbit EXEN2= 0xCB;      /* Timer 2 external enable */
sbit TCLK = 0xCC;      /* 0=Serial clock uses Timer 1 overflow, 1=Timer 2 */
sbit RCLK = 0xCD;      /* 0=Serial clock uses Timer 1 overflow, 1=Timer 2 */
sbit EXF2 = 0xCE;      /* Timer 2 external flag */
sbit TF2  = 0xCF;      /* Timer 2 overflow flag */

/*-------------------------------------------
T2MOD (0xC9) Bit Values
-------------------------------------------*/
#define DCEN_   0x01   /* 1=Timer 2 can be configured as up/down counter */
#define T2OE_   0x02   /* Timer 2 output enable */

/*-------------------------------------------
PSW (0xD0) Bit Registers
-------------------------------------------*/
sbit P    = 0xD0;      /* Parity Flag */
sbit FL   = 0xD1;      /* User Flag */
sbit OV   = 0xD2;      /* Overflow Flag */
sbit RS0  = 0xD3;      /* Register Bank Select Bit 0 */
sbit RS1  = 0xD4;      /* Register Bank Select Bit 1 */
sbit F0   = 0xD5;      /* User Flag 0 */
sbit AC   = 0xD6;      /* Auxiliary Carry Flag */
sbit CY   = 0xD7;      /* Carry Flag */

/*-------------------------------------------
SPCR (0xD5) Bit Values - Reset Value = 0000.01XX
-------------------------------------------*/
#define SPR0_   0x01   /* SPI Clock Rate Select bit 0 */
#define SPR1_   0x02   /* SPI Clock Rate Select bit 1 */
                       /* 00 = Fosc / 4   */
                       /* 01 = Fosc / 16  */
                       /* 10 = Fosc / 64  */
```

```
                /* 11 = Fosc / 128 */
#define CPHA_   0x04    /* SPI Clock Phase */
#define CPOL_   0x08    /* SPI Clock Polarity */
#define MSTR_   0x10    /* SPI Master/Slave Select: 0=Slave, 1=Master */
#define DORD_   0x20    /* SPI Data Order: 0=MSB First, 1=LSB First */
#define SPE_    0x40    /* SPI Enable: 0=Disabled, 1=Enabled */
#define SPIE_   0x80    /* SPI Interrupt Enable: 0=Disabled, 1=Enabled */

/*-------------------------------------------
Interrupt Vectors:
Interrupt Address = (Number * 8) + 3
-------------------------------------------*/
#define IE0_VECTOR    0  /* 0x03 External Interrupt 0 */
#define TF0_VECTOR    1  /* 0x0B Timer 0 */
#define IE1_VECTOR    2  /* 0x13 External Interrupt 1 */
#define TF1_VECTOR    3  /* 0x1B Timer 1 */
#define SIO_VECTOR    4  /* 0x23 Serial port */

#define TF2_VECTOR    5  /* 0x2B Timer 2 */
#define EX2_VECTOR    5  /* 0x2B External Interrupt 2 */

        /*-------------------------------------------


        -------------------------------------------*/
        #endif

        /*************************xlcd.h*********************************/
        sbit      e = P3^6;
        sbit      rs = P3^7;
        void delay(int tick)
        {
                unsigned int i,j;
```

```
        for(i=0;i<tick;i++)
        for(j=0;j<250;j++);
}
void lcd_command(unsigned char com)
{
rs=0;
e=1;
P0=com;
delay(1);
e=0;
delay(1);
}
void lcd_text(unsigned char text)
{
rs=1;
#include <scankey4x3.h>
#include <xlcd.h>
#include <can.h>
#include <at89s53.h>
bit scr_normal=1;
unsigned char *init="CAN MODULE V1.1";
unsigned char *enter="ID(0-2047):";
unsigned char *id_print="ID:";
unsigned char *data_r="RX DATA:";
unsigned char *data_t="TX DATA:";
unsigned char sidh;
unsigned char sidl;
unsigned char aa = 1 ;
unsigned char id_buf[4];
unsigned char data_buf[2];
```

```
/*****************************************************************************
************************************/
/****************************** Function Delay time
*************************************************************/
/*****************************************************************************
************************************/

/***************** convert ascii to LCD ********************/
void sent_to_lcd(unsigned char value)
{
        unsigned char buf=0;
        buf = value & 0xf0;
        buf = (buf>>4)|(0x30);
        lcd_text(buf);

        buf=value & 0x0f;
        buf=buf|0x30;
        lcd_text(buf);
        }

        /***************/
        void scr_init(void)
        {
        unsigned char i;

        lcd_clear();
//      lcd_command(0x0c);
        lcd_command(0x80);
        for(i=0;i<15;i++)
        lcd_text(*(init+i));
        lcd_jumporigin();
//      lcd_command(0x0c);
```

```
lcd_command(0xc0);
for(i=0;i<11;i++)
        lcd_text(*(enter+i));
}
void scr_RX_id(void)
{
unsigned char i;
lcd_clear();
lcd_command(0x0c);
lcd_command(0x80);
for(i=0;i<3;i++)
{
lcd_text(*(id_print+i));
}
lcd_command(0x0c);
lcd_command(0x84);
sent_to_lcd(sidh);
lcd_command(0x0c);
lcd_command(0x86);
sent_to_lcd(sidl);
}
void scr_TX_id(void)
{
unsigned char i;
//  lcd_clear();
//  lcd_command(0x0c);
    lcd_command(0x80);
    for(i=0;i<3;i++)
        lcd_text(*(id_print+i));


//  lcd_command(0x0c);
```

```
        lcd_command(0x84);
        sent_to_lcd(sidh);
//      lcd_command(0x0c);
        lcd_command(0x86);
        sent_to_lcd(sidl);

//      lcd_command(0x0c);
        lcd_command(0xc0);
        for(i=0;i<8;i++)
                lcd_text(*(data_t+i));
                delay(20);
}

//*****************************************************************************
*****//
//*************************Function
main*************************************//
//*****************************************************************************
**//

void main(void)
{

unsigned char intf;
unsigned char _data,_data1,i;
unsigned char push_key,key;
unsigned char countto_4 = 1;
unsigned char countto_2 = 1;
unsigned char count_f2  = 1;
unsigned char count_tokey = 0;
unsigned char count_tokey_f2 = 0;
unsigned char datk,x1,x2,x3;
```

```
unsigned char countloop2 = 1;
unsigned char loop_1 = 1;
unsigned char send_data = 0;


P0=0x00;
can_init();

lcd_init();
scr_init();
can_modify();
        while(countto_4)
        {
                push_key=scankey();
                if(push_key!=0xff)
                {
                count_tokey++;
                if(count_tokey == 1)
                {
                        id_buf[1] = push_key;
                        datk = push_key;
                        x1  = datk << 4;
                        lcd_command(0xcc);
                        push_key = push_key | 0x30;
                        lcd_text(push_key);
                }
                if(count_tokey == 2)
                {
                        id_buf[2] = push_key;
                        sidh = x1 | push_key;
                        lcd_command(0xcd);
                        push_key = push_key | 0x30;
```

```
                lcd_text(push_key);
            }
            if(count_tokey == 3)
            {
                id_buf[3] = push_key;
                datk = push_key;
                x2   = datk << 4;
                lcd_command(0xce);
                push_key = push_key | 0x30;
                lcd_text(push_key);
            }
            if(count_tokey == 4)
            {
                id_buf[4] = push_key;
                sidl = x2 | push_key;
                lcd_command(0xcf);
                push_key = push_key | 0x30;
                lcd_text(push_key);
                countto_4 = 0;
            }
        }
    }
//*****************************************************************//
//*************************send*************************************//

write_byte(TXB0SIDH,sidh);
write_byte(TXB0SIDL,sidl);

//*****************************************************************///
//**********************Total loop*********************************//
//*****************************************************************//
```

```
while(1)
{
scr_RX_id();
count_tokey=0;
push_key = scankey();
if(push_key == 0xff)
{

        intf=read_byte(CANINTF);
        if((intf&0x01)==0x01)
        {

                _data = read_byte(RXB0DB0);
                lcd_command(0x0c);
                lcd_command(0xc0);
                for(i=0;i<8;i++)
                lcd_text(*(data_r+i));

                lcd_command(0x0c);
                lcd_command(0xc9);
                sent_to_lcd(_data);
                delay(3000);
                bit_modify(CANINTF,0x01,0);
        }
        if((intf&0x04)==0x04)
        {
                bit_modify(CANINTF,0x04,0);
        }
}
else
{
        scr_TX_id();
```

```
countto_2=1;
while(countto_2)
{
        key = scankey();
        if(key != 0xff)
        {

            count_tokey++;

                if(count_tokey == 1)
                {
                    data_buf[1] = key;
                    datk = key;
                    x3 = datk << 4;
                    lcd_command(0xc9);
                    key =key | 0x30;
                    lcd_text(key);
                }

                if(count_tokey == 2)
                {
                    data_buf[2] = key;
                    lcd_command(0xca);
                    key = key | 0x30;
                    lcd_text(key);
                    countto_2 = 0;
                    _data1 = x3 | key;
                    for(i=0;i<3;i++)
                {
                write_byte(TXB0DB0,_data1);
                request_to_send(0x81);
                }
```

```
                                        }
                                    }


                                }
                            }
                        }
                }
                e=1;
                P0=text;
                delay(1);
                e=0;
                delay(1);
}
void lcd_clear()
{
        lcd_command(0x01);
}
void lcd_jumporigin()
{
        lcd_command(0x02);
}
void lcd_init()
{
        delay(250);
        delay(250);
lcd_command(0x38);
lcd_command(0x0c);
lcd_command(0x01);
}
```

```
/*************************scankey4*3.h*****************************/
#include <AT89S53.H>
```

```
sbit c0=P2^6;
sbit c1=P2^5;
sbit c2=P2^4;
sbit r0=P2^0;
sbit r1=P2^1;
sbit r2=P2^2;
sbit r3=P2^3;

void delay_db(int id)
{
        do
        {
                id--;
        }while(id>0);
        }
        unsigned char scankey(void)
        {
        unsigned char ret =0xff;
        c0=0;
        if(r0==0)
        {
                delay_db(30000);
                ret=0x01;
        }
        if(r1==0)
        {
                delay_db(30000);
                ret=0x04;
        }
        if(r2==0)
        {
                delay_db(30000);
```

```
            ret=0x07;
    }
    if(r3==0)
    {
            delay_db(30000);
            ret=0x0a;
    }
    c0=1;
    c1=0;
    if(r0==0)
    {
            delay_db(30000);
            ret=0x02;
    }
    if(r1==0)
    {
            delay_db(30000);
            ret=0x05;
    }
    if(r2==0)
    {
            delay_db(30000);
            ret=0x08;
    }
    if(r3==0)
    {
            delay_db(30000);
            ret=0x00;
    }
    c1=1;
    c2=0;
    if(r0==0)
```