การออกแบบบอร์ดเพื่อรับส่งข้อมูลภายใต้การสื่อสารแบบโปรโตคอล CAN

PROTOCAL CAN DATA COMMUNICATION BOARD DESIGN

นายรตน วงศ์วิรัติ

นายวรัทย์ พัฒนอางกูล

นางสาวสุนิสา ภู่วรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมการวัดคุม

ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

PROTOCAL CAN DATA COMMUNICATION BOARD DESIGN

RATANA WONGWIRAT

WARAT PATTANAOANGKUN

SUNISA PHUWORN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTTATION ENGINEERING

DEPARTMENT OF INSTRUMENT ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2002

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	การออกแบบบอร์ดเพื่อรับส่งข้อมูลภายใต้การสื่อสารแบบโปรโตคอล CAN PROTOCAL CAN DATA COMMUNICATION BOARD DESIGN			
นักศึกษาผู้จัดทำ	นายรตน	วงศ์วิรัติ	รหัสประจำตัว	42010289
	นายวรัทย์	พัฒนอางกูล	รหัสประจำตัว	42010312
	นางสาวสุนิสา	ภู่วรณ์	รหัสประจำตัว	42010391
ปริญญา	วิศวกรรมศาสตรบัณฑิต			
สาขาวิชา	วิศวกรรมการวัดคุม			
ปีการศึกษา	2545			

อาจารย์ผู้ควบคุมปริญญานิพนธ์	**ลายมือชื่อ**
รศ. พิพัฒน์ เลาหสงคราม	

วัน/เดือน/ปี ที่สอบ วันอังคารที่ 25 มีนาคม พ.ศ. 2546

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(ผศ.ประสิทธ์ ฉุลเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	การออกแบบบอร์ดเพื่อรับส่งข้อมูลภายใต้การสื่อสารแบบโปรโตคอล CAN PROTOCAL CAN DATA COMMUNICATION BOARD DESIGN
นักศึกษาผู้จัดทำ	นายรตน วงศ์วิรัติ นายวรัทย์ พัฒนอางกุล นางสาวสุนิสา ภู่วรณ์
อาจารย์ที่ปรึกษา	รศ. พิพัฒน์ เลาหสงคราม
ปีการศึกษา	2545

บทคัดย่อ

ในระบบการควบคุมอัตโนมัติใน โรงงาน อุตสาหกรรม จำเป็นอย่างยิ่งที่จะต้องมี การส่งข้อมูลระหว่างอุปกรณ์ที่ใช้ในการวัดกระบวนการต่างๆ เพื่อนำมาวิเคราะห์และคำนวณ แล้วส่งค่ากลับไปยังอุปกรณ์ควบคุม ทำให้กระบวนการดำเนินงานไปตามความต้องการ ซึ่งการรับส่งข้อมูลแบบ CAN เป็นอีกสิ่งหนึ่งที่สามารถนำมาใช้ได้ด้วยคุณสมบัติที่ดีหลายๆ อย่าง เช่น ราคาถูก และสามารถรับส่งข้อมูลในพื้นที่ที่มีสัญญาณรบกวนได้ โดยในส่วนของบอร์ดอินเตอร์เฟสนี้ เป็นส่วนสำคัญของการรับส่งข้อมูลแบบ CAN และบอร์ดอินเตอร์เฟสนี้สามารถเชื่อมต่อกับส่วนที่เป็น Application ของอุปกรณ์ควบคุมต่างๆ ได้อย่างง่ายดาย

Thesis Title	Protocal CAN Data Communication Board Design
Authors	Mr. Ratana Wongwirat
	Mr. Warat Pattanaoangkun
	Miss. Sunisa Phuworn
Thesis Advisor	Assoc.Prof. Phiphat Laohasongkram
Year	2002

ABSTRACT

The automation factory need to communicate between instrument that detect procession for analysing and sending data to controller, after that the controller will analyse and send data back to the other controller that control procession. CAN is a equipment that can control the communication for transfer data. CAN is used because it is cheap and can work in noise place. CAN interfaced board is necessary to transfer data and connect to each Application.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดีเพราะได้รับความเมตตาจาก รองศาสตราจารย์ พิพัฒน์ เลาหสงคราม ที่ได้ให้คำแนะนำแก่ผู้วิจัยตลอดมา อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่างๆ ในการทำปริญญานิพนธ์นี้ ผู้วิจัยรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ อาจารย์ภาควิชาวิศวกรรมการวัดคุมทุกท่าน ที่ให้คำแนะนำอันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

และลืมเสียมิได้คือ ขอกราบขอบพระคุณคุณแม่ อันเป็นที่รักยิ่งที่สนับสนุนและเป็นแรงบันดาลใจในการทำปริญญานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

สารบัญ (ต่อ)

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูป

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุจูงใจของการวิจัย

เนื่องจากปัจจุบันโรงงานอุตสาหกรรมส่วนใหญ่เริ่มที่จะเปลี่ยนมาเป็นระบบอัตโนมัติมากขึ้น ซึ่งจำเป็นอย่างยิ่งที่จะต้องมีการติดต่อสื่อสารซึ่งกันและกัน เพื่อนำข้อมูลที่ได้จากการวัดการเปลี่ยนแปลงของกระบวนการไปวิคราะห์และทำการคำนวณ เพื่อควบคุมกระบวนการอีกทีหนึ่ง จึงจำเป็นที่ต้องศึกษาการรับส่งข้อมูล

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์นี้จะเป็นการศึกษาและออกแบบสร้างบอร์ดอินเตอร์เฟส CAN เพื่อนำไปต่อเข้ากับสู่ Application ของเครื่องควบคุมหรือเครื่องมือวัดต่างๆ และสามารถทำการรับส่งข้อมูลระหว่างกันได้ โดยข้อมูลที่รับส่งกันนี้มีขนาดไม่มากนักและมีประสิทธิภาพดี อีกทั้งราคายังถูกว่าการใช้อุปกรณ์รับส่งเดิมที่ใช้กันอยู่

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์เล่มนี้จะกล่าวถึง การสื่อสารข้อมูลโดยใช้โปรโตคอล CAN และการออกแบบสร้างบอร์ดอินเตอร์เฟส CAN รวมถึงการทดสอบบอร์ดอินเตอร์เฟส CAN ซึ่งทดสอบส่งข้อมูลและรับข้อมูลระหว่างกันด้วย

1.4 ขั้นตอนการศึกษา

การทำโครงงานวิจัยในปริญญานิพนธ์ฉบับนี้ มีขั้นตอนการศึกษาเริ่มจากการศึกษา การรับส่งข้อมูลด้วยโปรโตคอล CAN ว่ารูปแบบการรับส่งเป็นอย่างไร รวมถึงศึกษาโครงสร้างในการที่จะสร้างบอร์ดอินเตอร์เฟส CAN ทั้งภาค Analog และ Digital

บทที่ 2

โครงสร้างและลักษณะทั่วไปของ CAN

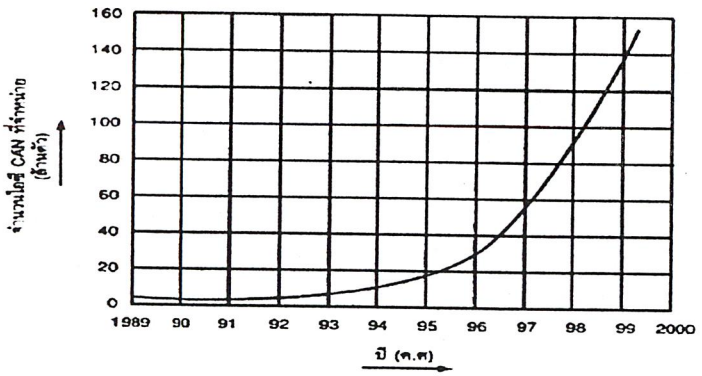
2.1 ความเป็นมาของ CAN

CAN หรือ Controller Area Network เป็นชื่อบัสสื่อสารข้อมูลแบบอนุกรมที่ออกแบบมาสำหรับใช้ในงาน ควบคุมแบบเรียลไทม์ ที่มีจุดเด่นอยู่ที่ ความสามารถใน การสื่อสารข้อมูล เป็นเครือข่าย(Network) ระหว่างอุปกรณ์ โดยไม่ต้องมีหมายเลขที่อยู่ของโหนด (Node Addressing) อุปกรณ์ทุกตัวในเครือข่ายสามารถเรียกใช้งานบัสได้ (Multi – master) และร้องขอใช้งานบัสได้พร้อมกันหลายตัว ด้วยความเร็วในการสื่อสารข้อมูลที่สูงถึง 1 เมกะบิตต่อวินาที มีระบบป้องกันและตรวจสอบความผิดพลาดที่มีประสิทธิภาพสูง

จุดเริ่มต้นของ CAN นั้นถูกพัฒนาขึ้นมาโดยบริษัท Robert Bosch ในประเทศเยอรมันเมื่อประมาณ 20 ปี ที่ผ่านมา สำหรับใช้ในระบบอิเล็กทรอนิกส์ภายในรถยนต์ เนื่องจากปัญหาในการพัฒนารถยนต์รุ่นใหม่ที่ต้องการสิ่งอำนวยความสะดวกและระบบรักษาความปลอดภัยเพิ่มขึ้น ส่งผลให้ระบบอิเล็กทรอนิกส์ภายในรถยนต์มีความซับซ้อนมากขึ้น มีโมดูลของวงจรต่างๆ เป็นจำนวนมากที่ต้องมีการสื่อสารข้อมูลระหว่างกัน

ด้วยเหตุนี้จึงมีความต้องการบัสสื่อสารข้อมูลที่มีประสิทธิภาพและมีความน่าเชื่อถือสูงในราคาที่เหมาะสม แทนที่ระบบบัสเดิมที่มีความยุ่งยากซับซ้อนไม่สะดวกที่จะนำมาใช้และมีค่าใช้จ่ายสูง ซึ่งภายหลังนอกจากจะใช้ใน อุตสาหกรรมรถยนต์ แล้วยังได้มีการนำ CAN ไปประยุกต์ใช้ในอุตสาหกรรมอื่นๆ อีกมากมาย เนื่องจากคุณสมบัติที่โดดเด่นของบัสนี้ในด้านความน่าเชื่อถือและความยืดหยุ่น รวมทั้งความง่ายต่อการใช้งานและมีค่าใช้จ่ายต่ำนั่นเอง

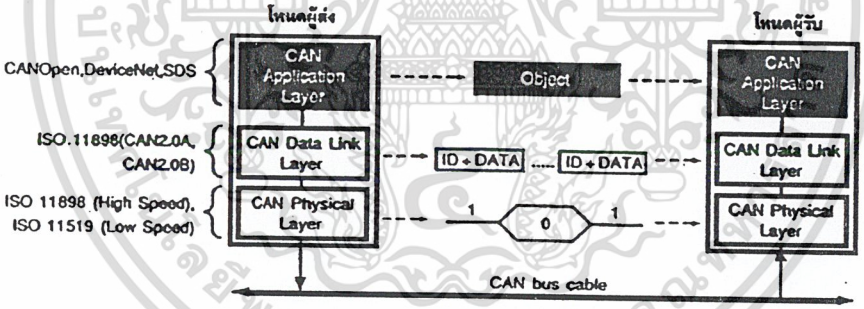
เนื่องจากความแพร่หลายในการใช้งานบัสนี้ภายหลังจึงได้มีการกำหนดให้CANเป็นบัสมาตรฐานระหว่างประเทศขึ้นมา นั่นคือมาตรฐาน ISO 11898 (ความเร็วสูง) และ ISO 11519 (ความเร็วต่ำ) ซึ่งเป็นการรับประกันถึงความสามารถและการยอมรับใน CAN ได้เป็นอย่างดี โดยมาตรฐานนี้ได้ครอบคลุมข้อกำหนดการทำงานของ CAN ในสองชั้น (Layer) แรกตามแบบจำลอง ISO/OSI คือ ชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data Link Layer)



รูปที่ 2.1 แนวโน้มการใช้งาน ไอซี CAN

2.2 โครงสร้างการทำงาน

โครงสร้างการทำงานของ CAN เมื่อเทียบกับแบบจำลอง ISO/OSI นอกจากจะประกอบด้วยสองชั้นแรกดังที่กล่าวไปแล้วยังประกอบด้วยชั้นที่ 7 คือชั้นประยุกต์ใช้งาน(Application Layer) โดยจะไม่มีส่วนประกอบของชั้นที่เหลือ คือชั้นที่ 3 – 6 ลักษณะโครงสร้างการทำงาน จะเป็นดังรูปที่ 2.2



รูปที่ 2.2 โครงสร้างการทำงานของ CAN

ชั้นที่ 1 Physical Layer

โครงสร้างการทำงานในชั้นนี้จะเป็นส่วนที่เกี่ยวข้องกับส่วนประกอบทางกายภาพของบัส ได้แก่ ตัวกลางที่ใช้ส่งผ่านสัญญาณข้อมูล คอนเน็กเตอร์และการเชื่อมต่อสายสัญญาณ รวมทั้งคุณสมบัติทางไฟฟ้าของสัญญาณข้อมูล (แรงดัน / กระแส) ของสัญญาณ มาตรฐานที่เกี่ยวข้องกับการทำงานในชั้นนี้มีอยู่ 2 มาตรฐาน คือ ISO 11519 สำหรับการสื่อสารข้อมูลความเร็วต่ำ คือมีอัตราเร็วอยู่ในช่วง 5 ถึง 125 กิโลบิต ต่อ วินาที และ ISO 11898 สำหรับการสื่อสารข้อมูลความเร็วสูง สนับสนุนการสื่อสารข้อมูลด้วยความเร็วสูงสุดถึง 1 เมกะบิต ต่อ วินาที

ชั้นที่ 2 Data Link Layer

เกี่ยวข้องกับโปรโตคอลและรูปแบบของข้อมูลที่สื่อสารกันในเชิงตรรกะรวมถึงการป้องกันความผิดพลาดที่จะเกิดขึ้นกับข้อมูลเพื่อรับประกันว่าข้อมูลที่ได้มีความถูกต้องโดยข้อกำหนดของการทำงานในชั้นนี้ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วย และได้มีการแก้ไขและขยาย ข้อกำหนดของมาตรฐานในชั้นนี้เพิ่มเติม โดยจัดทำเป็น 2 เวอร์ชั่น คือ CAN 2.0 A (เดิม) และ CAN 2.0 B (ขยายเพิ่มเติม)

ชั้นที่ 7 Application Layer

เป็นส่วนของการนำ CAN ไปประยุกต์ใช้งานบนพื้นฐานการทำงานในสองชั้นแรก แม้ว่าการทำงานในชั้นนี้จะไม่ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วย แต่ก็ได้มีการสร้างโปรโตคอลสำหรับการทำงานในชั้นนี้ขึ้นมา ซึ่งมีอยู่ด้วยกันหลายรูปแบบ ตัวอย่างโปรโตคอลในชั้นนี้ที่มีการใช้งานกัน เช่น CAN open , DeviceNet , CAN Kingdom , OSEK / VDX , SDS และ J1939 เป็นต้น

2.3 คุณสมบัติของ CAN

- มาตรฐาน ISO 11898 พัฒนาขึ้นมาโดย Robert Bosch GmbH
- โครงสร้างการทำงานมี 2 ชั้น คือ ชั้นกายภาพ (Physical Layer) และ ชั้นเชื่อมโยงข้อมูล (Data Link Layer)
- สื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส
- สร้างเครือข่ายแบบ Multi – master / Broadcasting
- สนับสนุนการทำงานแบบเรียลไทม์
- ใช้หลักการ CSMA / CD ในการเข้าใช้บัส
- ไม่มีการกำหนดหมายเลขที่อยู่กำกับโหนด
- ส่งข้อมูลครั้งละ 0 – 8 ไบต์ ด้วยอัตราเร็วสูงสุด 1 เมกะบิต ต่อ วินาที
- มีความปลอดภัยในการส่งข้อมูลสูง
- ตัวควบคุมโปรโตคอลสามารถบรรจุอยู่ในชิปเดียว
- ต้นทุนต่ำ ใช้งานสะดวก บำรุงรักษาง่าย
- มีการใช้งานอย่างแพร่หลายในงานอุตสาหกรรมและระบบอิเล็กทรอนิกส์ในยานยนต์

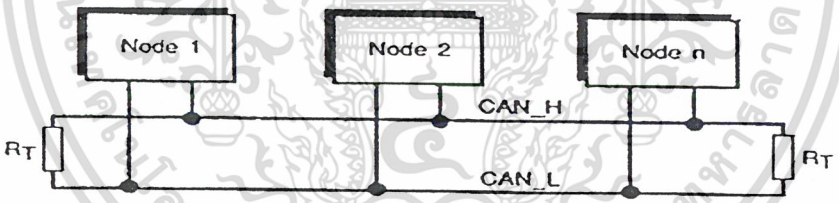
2.4 ลักษณะทางกายภาพ

ลักษณะทางกายภาพของ CAN นั้น เริ่มจากตัวกลางที่ใช้ส่งผ่านข้อมูลและลักษณะของสัญญาณที่ส่งออกไป

ตัวกลางที่ใช้ในการส่งผ่านข้อมูลใน CAN นั้น มีอยู่ด้วยกันหลายรูปแบบ ตั้งแต่การใช้สายสัญญาณ 2 เส้น การใช้สายสัญญาณเส้นเดียว การใช้สายไฟเบอร์ออปติกหรือแม้แต่การส่งสัญญาณแบบไร้สาย แต่ที่นิยมใช้กันมากคือ การใช้สายสัญญาณ 2 เส้นแบบตีเกลียว (Twisted – pair Cable) ซึ่งในที่นี้จะอ้างอิงการส่งผ่านข้อมูลโดยใช้ตัวกลางในรูปแบบนี้

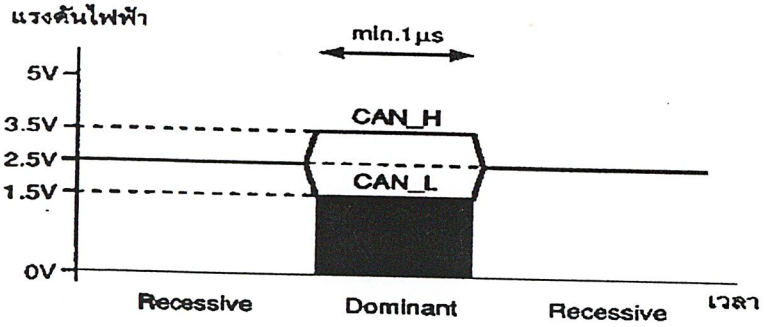
การส่งผ่านข้อมูลใน CAN มีรูปแบบของการส่งข้อมูลเป็นแบบอนุกรมที่ใช้สายสัญญาณเพียง 2 เส้น ซึ่งมีลักษณะการเชื่อมต่อสายสัญญาณ (Topology) ดังรูปที่ 2.3 โดยอุปกรณ์ทุกตัวจะต่ออยู่บนสายสัญญาณคู่เดียวกัน และปิดปลายคู่สายสัญญาณทั้งสองข้างด้วยเทอร์มิเนชันอิมพีแดนซ์ (Termination Impedance)

สัญญาณข้อมูลที่ส่งจะใช้วิธีการส่งแบบสัญญาณแรงดันผลต่าง (Differential Voltage Signal) สถานะของสัญญาณในสายส่งได้จากการเปรียบเทียบระดับแรงดันหรือศักย์ไฟฟ้าระหว่างสายสัญญาณทั้งสองเส้น คือ CAN_H (ศักย์สูง) และ CAN_L (ศักย์ต่ำ)



รูปที่ 2.3 ลักษณะการเชื่อมต่ออุปกรณ์ (Topology) ใน CAN

สถานะของสัญญาณข้อมูลในสายส่ง CAN จะมีอยู่สองสถานะคือ สถานะรีเซสสีฟ (Recessive) และสถานะโดมิแนนต์ (Dominant) ระดับแรงดันไฟฟ้าของสัญญาณในแต่ละเส้น (เทียบกับกราวด์) และผลต่างแรงดันไฟฟ้าระหว่างสายคู่สัญญาณในแต่ละสถานะจะเป็นดังรูปที่ 2.4



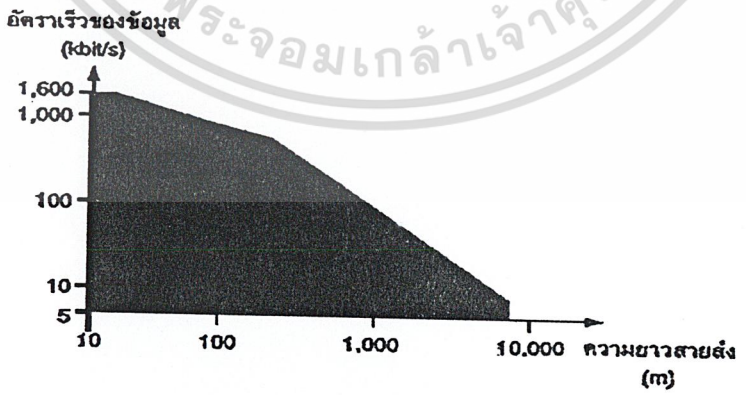
รูปที่ 2.4 ลักษณะของสัญญาณข้อมูลแต่ละสถานะ

ถ้าผลต่างแรงดันระหว่าง CAN_H และ CAN_L เป็น 0 โวลต์ จะเป็นสถานะรีเซสสีฟซึ่งจะแทนข้อมูลที่มีลอจิกเป็น " 1 "

ถ้าผลต่างของแรงดันระหว่าง CAN_H และ CAN_L เป็น 2 โวลต์ จะเป็นสถานะโดมิแนนต์ซึ่งจะแทนข้อมูลที่มีลอจิกเป็น " 0 "

การส่งผ่านข้อมูลโดยใช้ผลต่างแรงดันของคู่ส่งสัญญาณมีข้อดีคือ จะช่วยลดผลการรบกวนอันเนื่องมาจากผลรวมของ EMI ลงได้เป็นอย่างมาก ทำให้สามารถส่งสัญญาณได้ในอัตราเร็วที่สูงขึ้นและได้ในระยะทางที่ไกลมากขึ้น

อัตราเร็วในการส่งข้อมูลและระยะทางที่ส่งได้จะแปรผกผันกันดังกราฟในรูปที่ 2.5 ตามมาตรฐาน ISO 11898



รูปที่ 2.5 กราฟความสัมพันธ์ระหว่างอัตราเร็วของข้อมูลกับระยะทางที่ส่งได้

ตารางที่ 2.1 ความสัมพันธ์ของคุณสมบัติสายส่งกับอัตราเร็วข้อมูล

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

อัตราเร็วในการส่งข้อมูลสูงสุดคือ 1 เมกะบิตต่อวินาที ที่ความยาวของสายส่งไม่เกิน 40 เมตร สำหรับค่าความต้านทานและขนาดของสายส่ง รวมทั้งขนาดของเทอร์มิเนชันอิมพีแดนซ์ที่แนะนำให้ใช้ ดูได้จากตารางที่ 2.1 ตัวอย่างเช่น ที่อัตราเร็วข้อมูล 1 เมกะบิต ต่อวินาที ควรใช้สายส่งที่มีความต้านทานขนาด 70 มิลลิโอห์ม ต่อเมตร และมีขนาดเส้นผ่านศูนย์กลางของสายส่งมากกว่า 0.25 ตารางมิลลิเมตร โดยต่อเทอร์มิเนชันอิมพีแดนซ์ที่มีค่า 124 โอห์ม ไว้ที่ปลายของสายส่งทั้งสองด้าน

ข้อควรระวังในการต่อสายสัญญาณก็คือ ในกรณีที่มีการต่อสายสัญญาณพ่วงจากบัสกลาง สายพ่วงควรมีความยาวไม่เกิน 2 เมตร เมื่อมีอัตราเร็วในการส่งข้อมูลเป็น 250 กิโลบิต ต่อวินาที และไม่ควรเกิน 30 เซนติเมตร ถ้าอัตราเร็วของข้อมูลสูงกว่านั้น ทั้งนี้ความยาวของสายพ่วงทุกเส้นรวมกันไม่ควรเกิน 30 เมตร

บทที่ 3
โปรโตคอลในการส่งข้อมูลของ CAN

3.1 หลักการเบื้องต้น

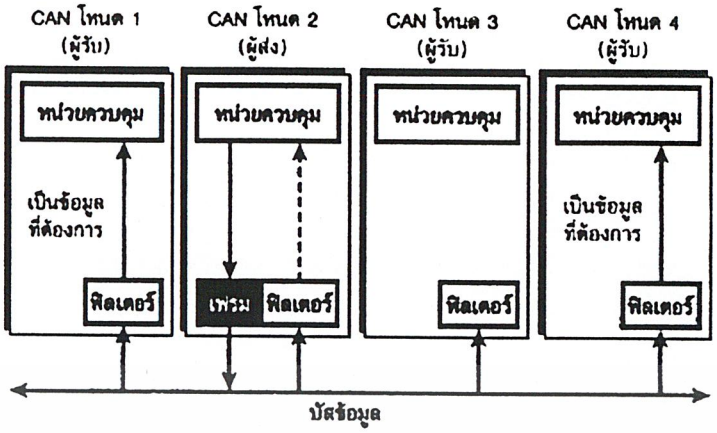
ในงานควบคุมอัตโนมัติในอุตสาหกรรม เพื่อให้ระบบ CAN มีความน่าเชื่อถือ ปลอดภัย และมีประสิทธิภาพในการทำงานมากขึ้น และลดความยุ่งยากซับซ้อนและจำนวนสายสัญญาณ ในการเชื่อมต่อระหว่างอุปกรณ์ในส่วนต่างๆลง ได้กำหนดเป็นมาตรฐาน ISO11898 (CAN ความเร็วสูง) และ ISO11519 (CAN ความเร็วต่ำ) ขึ้นมา

รูปแบบการสื่อสารข้อมูลของ CAN เป็นแบบอนุกรมอะซิงโครนัส โดยที่อุปกรณ์แต่ละตัวหรือแต่ละโหนด (Node) สามารถส่งข้อมูลหรือข้อความ (Message) ไปยังโหนดอื่นๆได้ เหตุที่เรียกว่าเป็นการสื่อสารข้อมูลแบบอะซิงโครนัสเนื่องจากไม่มีการส่งสัญญาณนาฬิกาเพื่อใช้ในการเข้าจังหวะ แต่อาศัยการเข้าจังหวะจากจุดเริ่มต้นของแต่ละข้อความที่ส่งไป (ลักษณะเดียวกับการส่งข้อมูลแบบ UARTใน RS232)

ในส่วนของระเบียบวิธีหรือโปรโตคอลในการสื่อสารข้อมูลในชั้นเชื่อมโยงข้อมูลของ CAN นั้น มีหลักการสำคัญที่ทำให้ CAN เป็นบัสข้อมูลที่ทำงานได้อย่ามีประสิทธิภาพ ได้แก่ การสื่อสารแบบบรอดคาส (Broadcast Communication), การเข้าใช้บัสจากหลายโหนด (Multiple Bus Access), การสื่อสารที่อิงจากข้อความ (Message-based Communication) และการป้องกันความผิดพลาด (Error Protetion)

3.1.1 การสื่อสารแบบบรอดคาส

การส่งข้อมูลใน CAN จะใช้หลักการส่งข้อมูลแบบบรอดคาส คือ ทุกโหนดที่อยู่บนบัสสามารถรับข้อมูลที่ส่งมาจากโหนดผู้ส่งได้ หลังจากที่ได้รับข้อมูลแล้วเป็นหน้าที่ของแต่ละโหนดที่จะต้องตรวจสอบเองว่าเป็นข้อมูที่ต้องการหรือไม่ซึ่งการตรวจสอบตรงจุดนี้จะอาศัยแอกเซปแตนซ์ฟิลเตอร์ (Acceptance Fillter) ทำหน้าที่สกัดกั้นข้อมูลที่ไม่ต้องการทิ้งไปและยอมรับเฉพาะข้อมูลที่ต้องการเท่านั้น โดยดูจากค่ารหัสประจำตัว (Identifier) ของข้อมูลหรือข้อความนั้น ลักษณะการส่งข้อมูลแบบ บรอดคาสจะเป็นดังรูปที่ 3.1



รูปที่ 3.1 ลักษณะการสื่อสารข้อมูลแบบบรอดคลาสใน CAN

รูปแบบกรส่งข้อมูลแบบบรอดคาสใน CAN นี้ อาจจะเปรียบเทียบได้กับการส่งข่าวสารจากสถานีวิทยุ เช่น ข้อมูลการจราจร ผู้ฟังหรือผู้ใช้รถจะตัดสินใจเองว่าข้อมูลที่ได้รับมีความสำคัญกับตนหรือไม่ โดยดูจากเส้นทางที่ต้องการใช้เป็นหลัก ซึ่งในที่นี้ค่ารหัสประจำตัวของข้อมูลก็คือชื่อของถนนที่บอกข้อมูลการจราจร ส่วนแอกเซปแตนซ์ฟิลเตอร์ก็คือเส้นทางที่ต้องการใช้เพื่อให้ไปถึงเป้าหมายนั่นเอง

การสื่อสารข้อมูลใน CAN นอกจากจะสื่อสารข้อมูลแบบบรอดคาสธรรมดา คือ แต่ละโหนดจะรอรับข้อมูลจากโหนดที่ต้องการส่งแล้ว ยังสามารถร้องขอข้อมูล (Remote Request) จากโหนดที่มีข้อมูลดังกล่าวได้อีกด้วย โดยการส่งค่ารหัสประจำตัวของข้อมูลที่ต้องการจะทราบออกไป หากโหนดใดมีค่ามูลดังกล่าวก็จะส่งข้อมูลนั้นตอบกลับมา ซึ่งนอกจากโหนดที่ร้องขอข้อมูลนี้ไปแล้ว โหนดอื่นๆ ที่เหลือก็สามารถรับข้อมูลดังกล่าวได้ด้วย (เนื่องจากเป็นการส่งข้อมูลแบบบรอดคาสนั่นเอง)

3.1.2 การเข้าใช้บัสจากหลายโหนด

จุดเด่นการส่งโปรโตคอลการส่งข้อมูลที่ใช้ใน CAN อย่างหนึ่งก็คือ การยินยอมให้มีการเข้าใช้บัสข้อมูลจากหลายโหนดได้พร้อมกัน ในกรณีที่มีโหนดมากกว่าหนึ่ง โหนดต้องการส่งข้อมูลในเวลาเดียวกัน จะมีการตัดสินชี้ขาดว่าโหนดใดจึงจะมีสิทธิที่จะบัสข้อมูลในเวลานั้นเพียงโหนดเดียว ในขณะที่โหนดอื่นจะต้องหยุดทำการส่งและรอที่จะส่งข้อมูลนั้นในภายหลังเมื่อบัสว่างไม่ได้ถูกใช้งานแล้ว

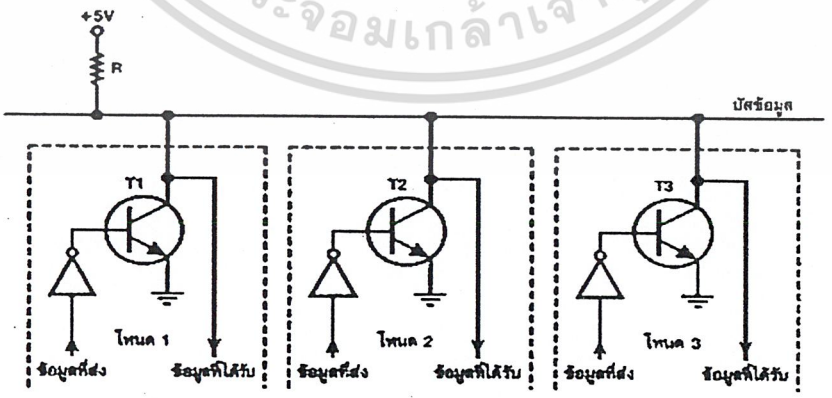
การเข้าใช้บัสใน CAN จะใช้วิธีการที่เรียกว่า Carrier Sense Multiple Access แบบ Collision Detection (CSMA/CA) กล่าวคือ เมื่อโหนดใดต้องการใช้บัส จะต้องทำการตรวจสอบจนพบว่าบัสว่างไม่ได้ถูกใช้งานอยู่เป็นระยะเวลาหนึ่งจึงเริ่มทำการส่งข้อมูลออกไป(CS) ซึ่งขณะที่บัส ว่างอยู่นี้แต่ละโหนดมีสิทธิที่จะเข้าใช้บัสด้วยโอกาสที่เท่ากัน (MA) แต่ถ้ามีการส่งข้อมูลออกมาพร้อมกัน โหนดจะทราบได้ว่ามีการชนกันของข้อมูลเกิดขึ้น (CD) และดำเนินการแก้ไขการชนกันของของข้อมูลที่เกิดขึ้นต่อไป

เมื่อมีการเข้าใช้บัสพร้อมกัน วิธีการที่นำมาใช้ใน CAN เพื่อตัดสินว่าโหนดใดจะมีสิทธิได้เข้าใช้บัสจะอาศัยค่าลำดับความสำคัญข้อมูลเป็นตัวตัดสิน (Arbitration on Message Priority : AMI) ซึ่งค่าลำดับความสำคัญ (Priority) ของข้อมูลจะถูกระบุอยู่ในค่ารหัสประจำตัว (Identifier) ของข้อมูลนั้น โดยการตัดสินชี้ขาดจะเกิดขึ้นโดยไม่ทำลายโอกาสในการใช้บัสของข้อมูลที่มีค่าลำดับความสำคัญสูงสุด (Non-destructive Arbitration) และการตัดสินจะเกิดขึ้นทันทีในระดับบิตข้อมูล (Bitwise Arbitration)

กระบวนการเช่นนี้จะเกิดขึ้นได้ต้องอาศัยคุณสมบัติที่สำคัญ 2 ประการคือ

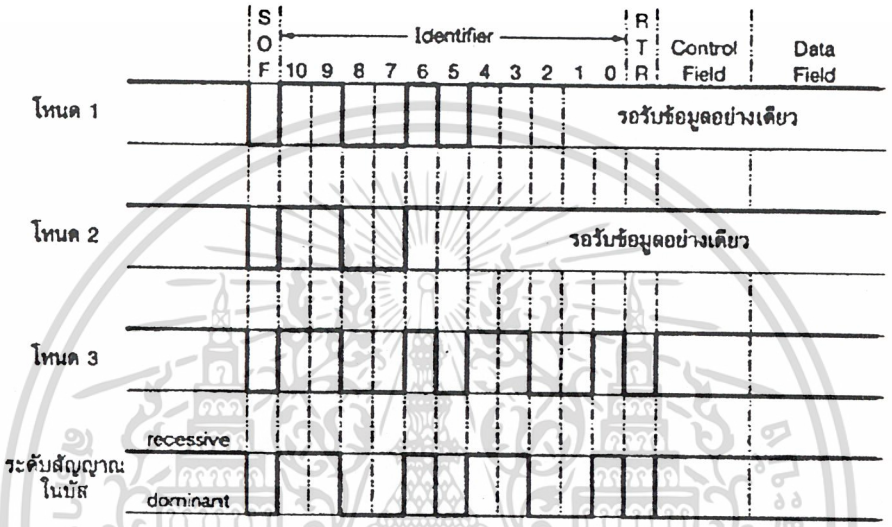
- สถานะของสัญญาณข้อมูลที่เป็นแบบโดมิแนนต์ (Dominant) และรีเซสสีฟ (Recessive) โดย CAN กำหนดให้สถานะโดมิแนนต์แทนลอจิก “0” และสถานะรีเวสสีฟแทนลอจิก “1” ในการใช้งานเมื่อสัญญาณทั้งสองสถานะถูกส่งมาที่บัสพร้อมกัน บัสจะมีสถานะโดมิแนนต์ (คืออ่านค่าได้เป็นลอจิก “0”)

- โหนดผู้ส่งต้องสามารถตรวจสอบสถานะของบัสอยู่ตลอดเวลา ว่าข้อมูลที่อยู่ในบัสตรงกับข้อมูลที่ส่งออกไปหรือไม่ หากไม่ตรงกันแสดงว่ามีการชนกันของข้อมูลหรือมีความผิดพลาดในการส่งข้อมูลเกิดขึ้น



รูปที่ 3.2 โครงสร้างอย่างง่ายของวงจรที่แต่ละโหนดใช้เชื่อมต่อเข้ากับบัส CAN

โดยลักษณะของวงจรที่ใช้เชื่อมต่อเข้ากับบัสจะแสดงได้ด้วยวงจรอย่างง่ายดังรูปที่ 3.2 ในการส่งข้อมูลแต่ละโหนดจะเริ่มต้นด้วยการส่งค่ารหัสประจำตัวของข้อมูลออกมาก่อน ในขณะเดียวกันก็จะคอยตรวจสอบว่าค่าที่ได้รับจากบัสมีค่าตรงกับค่าที่ส่งออกไปหรือไม่ ในกรณีที่มีการส่งข้อมูลพร้อมกันมากกว่า 1 โหนด ดังตัวอย่างในรูปที่ 3.3 มีการส่งข้อมูลพร้อมกัน 3 โหนด ขณะที่บิตข้อมูล (ของค่ารหัสประจำตัว) ที่แต่ละโหนดส่งออกมามีค่าตรงกันอยู่ การส่งข้อมูลในแต่ละโหนดก็จะดำเนินการต่อไปตามปกติ



รูปที่ 3.3 ตัวอย่างของสัญญาณในบัส CAN กรณีที่มีโหนดส่งข้อมูลออกมาที่บัสพร้อมกัน (3โหนด)

จนกระทั่งเมื่อค่าที่ส่งออกมาไม่ตรงกัน เช่นในบิตที่ 5 โหนดที่ 1 และ 3 ส่งสัญญาณสถานะโดมิแนนต์ (ลอจิก “0”) ในขณะที่โหนดที่ 2 ส่งสัญญาณสถานะรีเซสสีฟ (ลอจิก “1”) แต่อาจค่าจากบัสกลับมาได้เป็นโดมิแนนต์ ซึ่งทำให้โหนดที่ 2 หมดสิทธิในการส่งข้อมูลในรอบนี้ไปและเปลี่ยนมาเป็นโหนดผู้รับแทน เช่นเดียวกับในบิตที่ 2 โหนดที่ 1 ส่งสัญญาณสถานะรีเซสสีฟแต่อ่านค่ากลับมาได้เป็นโดมิแนนต์ ทำให้โหนดที่ 1 ต้องหยุดส่งข้อมูลไป สุดท้ายจึงเหลือแต่โหนดที่ 3 เพียงโหนดเดียวที่ส่งข้อมูลได้ในรอบนี้

จากตัวอย่างนี้กล่าวได้ว่า ข้อมูลจากโหนดที่ 3 ที่มีค่าประจำตัวต่ำกว่าจะมีค่าระดับความสำคัญสูงกว่าข้อมูลจากโหนดที่ 1 และ 2 ซึ่งจะมีสิทธิส่งข้อมูลได้ก่อน และไม่ต้องเริ่มทำการส่งข้อมูลใหม่เมื่อมีการส่งข้อมูลชนกับโหนดอื่น ในขณะที่โหนดที่ 1 และ 2 ซึ่งมีค่าระดับความสำคัญต่ำกว่าจะเริ่มทำการส่งข้อมูลได้ใหม่หลังจากที่โหนดที่ 3 ส่งข้อมูลเรียบร้อยแล้ว

3.1.3 การสื่อสารที่อิงจากข้อความ

การส่งข้อมูลใน CAN จะไม่มีการระบุที่อยู่ของโหนดผู้รับและโหนดผู้ส่งเหมือนอย่างที่ใช้กันอยู่ทั่วไป แต่จะใช้การระบุด้วยค่าประจำตัว (Identifier) ของข้อมูลหรือข้อความแทน เพื่อเป็นการบ่งบอกถึงชนิดและระดับความสำคัญของข้อมูลนั้น โดยอาศัยวิธีการส่งข้อมูลแบบบรอดคาสดังเช่นที่กล่าวมาแล้วข้างต้น คือเมื่อมีโหนดในโหนดหนึ่งทำการส่งข้อมูลออกมาที่บัสแต่ละโหนดจะได้รับการแจ้งว่ามีการส่งข้อมูลเกิดขึ้นและจะตัดสินใจกันเองว่าจะรับข้อมูลที่ส่งมานั้นหรือไม่

ตัวอย่างเช่น โหนด A ทำการส่งข้อมูล (สมมติว่าเป็นค่าที่วัดได้จากเซนเซอร์) ด้วยค่ารหัสประจำตัวของข้อมูล (สมมติว่าเท่ากับ 123) ออกไปที่บัส หากโหนดใดต้องการใช้ข้อมูลดังกล่าวก็จะทำการรับข้อมูลนี้ไว้ จะเห็นว่าการส่งข้อมูลด้วยวิธีนี้ไม่จำเป็นต้องมีการระบุหมายเลขที่อยู่หรือรหัสประจำตัวของโหนดทั้งโหนดผู้รับและโหนดผู้ส่ง ซึ่งมีข้อดีคือ สามารถทำการเพิ่มโหนดเข้ามาในระบบได้โดยไม่จำเป็นต้องทำการโปรแกรมทุกโหนดใหม่ เพื่อให้รับรู้ว่ามีการเพิ่มโหนดเข้ามา ในขณะเดียวกันโหนดที่เพิ่มเข้ามาใหม่นี้ก็สามารถรับข้อมูลที่มีการส่งกันในบัสได้ทันที โดย พิจารณา จากค่ารหัสประจำตัวของข้อมูลที่ส่งมานั่นเอง

นอกจากการรอรับข้อมูลที่โหนดอื่นส่งมาให้แล้ว (ซึ่งบางทีอาจต้องรอเป็นเวลานานกว่าที่จะได้รับข้อมูลที่ต้องการ) แต่ละโหนดยังสามารถเป็นฝ่ายร้องขอข้อมูลจากโหนดอื่นได้ด้วย เพื่อให้โหนดที่มีข้อมูลดังกล่าวส่งข้อมูลนั้นมาให้ทันที โดยเรียกการร้องขอข้อมูลนี้ว่า Remote Transmission Request (RTR) มีข้อดีคือ ข้อมูลบางอย่างที่ไม่ได้มีการใช้งานเป็นประจำ ก็ไม่จำเป็นต้องส่งออกมาอยู่ตลอดเวลา รอให้มีการร้องขอข้อมูลนั้นมาก่อนจึงค่อยส่งข้อมูลนั้นออกไป ซึ่งจะช่วยลดปริมาณการใช้บัสลงได้

3.1.4การป้องกันความผิดพลาด

เนื่องจากแรกเริ่มเดิมที่นั้น CAN ถูกออกแบบมาเพื่อใช้ในระบบอิเล็กทรอนิกส์ในรถยนต์ ซึ่งปัญหาที่พบส่วนใหญ่ก็คือเรื่องสัญญาณรบกวน ดังนั้นการออกแบบโปรโตคอลเพื่อให้สามารถใช้ส่งข้อมูลได้อย่างมีประสิทธิภาพ จะต้องสามารถส่งข้อมูลได้อย่างไม่มีข้อผิดพลาดในขณะที่ความเร็วในการส่งข้อมูลต้องเป็นที่ยอมรับหรือรองรับการใช้งานได้อย่างกว้างขวาง

ทั้งนี้นอกจาก CAN จะสามารถส่งข้อมูลได้ที่ความเร็วสูงถึง 1 เมกะบิตต่อวินาที (ในเวอร์ชัน CAN2.0) แล้ว จุดเด่นที่สำคัญอีกอย่างหนึ่งก็คือ การตรวจสอบความผิดพลาดของข้อมูลเพื่อให้แน่ใจได้ว่าข้อมูลที่ได้มีความถูกต้องสมบูรณ์ไม่มีความผิดพลาดเกิดขึ้น โดยการสื่อสารข้อมูลใน CAN มีวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้นหลายแบบ ซึ่งจะขอกล่าวถึงรายละเอียดในภายหลัง

3.2 โปรโตคอลการส่งข้อมูล

โปรโตคอลการส่งข้อมูลใน CAN จะแบ่งข้อมูลที่ส่งออกเป็นแพ็กเก็ต(Packets) หรือเฟรม (Frames) ซึ่งมีใช้อยู่ด้วยกัน 4 ชนิด ได้แก่ เฟรมข้อมูล(Data Frame) ใช้ในการส่งข้อมูลตามปกติ, เฟรม ร้องขอข้อมูล (Remote Frame) ใช้ในการร้องขอข้อมูลจากโหนดอื่น, เฟรมแสดงความผิดพลาด (Error Frame) ใช้แสดงว่ามีความผิดพลาดเกิดขึ้น และ เฟรมโอเวอร์โหลด (Overload Frame) ใช้เพื่อบอกว่าต้องการเวลาในการประมวลผลข้อมูลที่ได้รับเพิ่มขึ้น

ในที่นี้จะขอกล่าวถึงรายละเอียดเฉพาะเฟรมข้อมูลและเฟรมร้องขอข้อมูลเท่านั้น ส่วนรายละเอียดของเฟรมแสดงความผิดพลาดและเฟรมโอเวอร์โหลดจะไม่ขอกล่าวถึงในที่นี้ โดยสามารถหาอ่านเพิ่มเติมได้จากข้อมูลอ้างอิงและเพิ่มเติมในตอนท้าย

3.2.1 เฟรมข้อมูล

ในเฟรมข้อมูล 1 เฟรมจะประกอบด้วยส่วนย่อยๆ ที่เรียกว่าฟิลด์ (Fields) ดังในรูปที่ 3.4 และ 3.5 ซึ่งแสดงส่วนประกอบของเฟรมข้อมูลแบบมาตรฐาน (Standard Data Frame) และเฟรมข้อมูลแบบขยาย (Extended Data Frame) ตามลำดับเฟรมข้อมูลทั้งสองแบบจะมีข้อแตกต่างกันตรงที่ขนาดของรหัสประจำตัวของข้อมูล (Identifier หรือ ID)

ในตอนแรกนี้จะขอกล่าวถึงส่วนประกอบต่างๆ ในเฟรมข้อมูลแบบมาตรฐานก่อน ส่วนรายละเอียดในข้อแตกต่างที่มีในเฟรมข้อมูลแบบขยายจะขอกล่าวถึงในส่วนถัดไป

เฟรมข้อมูลแบบมาตรฐาน (CAN2.0A) ส่วนประกอบต่างๆ ในเฟรมข้อมูลแบบมาตรฐานมีดังนี้

- บิตเริ่มต้นเฟรม (Start of Frame : SOF) มีสถานะเป็นโดมิแนนต์หรือลอจิก “0” ซึ่งเป็นจุดที่ทุกโหนดในบัสใช้ในการเริ่มต้นเข้าจังหวะสำหรับการรับ-ส่งข้อมูลระหว่างกัน

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

ตารางที่ 3.1 ค่าของบิตต่างๆ ที่ใช้ในเฟรม (* = มีเฉพาะในเฟรมแบบขยาย)

ชื่อบิต	ค่าที่ใช้	
	0 ' หรือโดมิแนนต์	1 ' หรือรีเซสสีฟ
SOF	/	
RTR	เฟรมข้อมูล	เฟรมร้องขอข้อมูล
IDE	เฟรมแบบมาตรฐาน	เฟรมแบบขยาย
RB0	/	
CRC Del		/
ACK Del		/
SRR *		/
RB1 *	/	

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

จำนวนข้อมูล	Data Length Code (DLC)			
(ไบต์)	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

การส่งข้อมูลโดยใช้เฟรมข้อมูลแบบมาตรฐานซึ่งมีหมายเลข ID ขนาด 11 บิต จะมีหมายเลข ID ที่สามารถใช้งานได้จำนวน 2^{11} = 2,048 หมายเลขที่แตกต่างกัน โดยมีการสำรองไว้ 16 หมายเลขสำหรับใช้งานในฟังก์ชันพิเศษเฉพาะอย่าง จึงมีเหลือให้ใช้งานได้ 2,032 หมายเลขซึ่งหมายถึงจำนวนข้อมูลที่สามารถรองรับได้นั่นเอง

บิต RTR ซึ่งอยู่ถัดจากหมายเลข ID เป็นบิตที่ใช้แสดงชนิดของเฟรมข้อมูลโดยถ้าบิต RTR เท่ากับ "0" หรือโดมิแนนต์แสดงว่าเป็นเฟรมข้อมูล แต่ถ้าเป็นบิต RTR = "1" หรือรีเซสสีฟแสดงว่าเป็นเฟรมขอข้อมูล (ซึ่งจะขอกล่าวถึงเฟรมร้องขอข้อมูลในภายหลัง) ในที่นี้เป็นเฟรมข้อมูลดังนั้นบิต RTR นี้จึงมีค่าเป็น "0" หรือ โดมิแนนต์

- ฟิลด์ควบคุม (Control Field) มีขนาด 6 บิต ประกอบด้วยบิต IDE (Identifier Extension) 1 บิต, บิต RB0 (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 ดังนั้นบิต IDE นี้จะมีค่าเป็น '0' หรือโดมิแนนต์

บิต RB0 เป็นบิตที่สำรองไว้สำหรับการใช้งานในอนาคต ในการงานจะให้บิตนี้เป็น '0' หรือโดมิแนนต์

ส่วนสุดท้ายในฟิลด์ควบคุมคือ DLC มีขนาด 4 บิต ใช้บอกขนาดข้อมูลที่ส่งมา โดยข้อกำหนดของ 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 และบิต CRC Delimiter 1 บิต

การส่งข้อมูลในแต่ละเฟรมจะมีการคำนวณค่า CRC เริ่มตั้งแต่บิตเริ่มต้นเฟรมไปจนถึงฟิลด์ข้อมูล โดยสามารถตรวจพบความผิดพลาดของข้อมูลที่เกิดขึ้นในช่วงดังกล่าวได้หากมีความผิดพลาดไม่เกิน 5 บิต

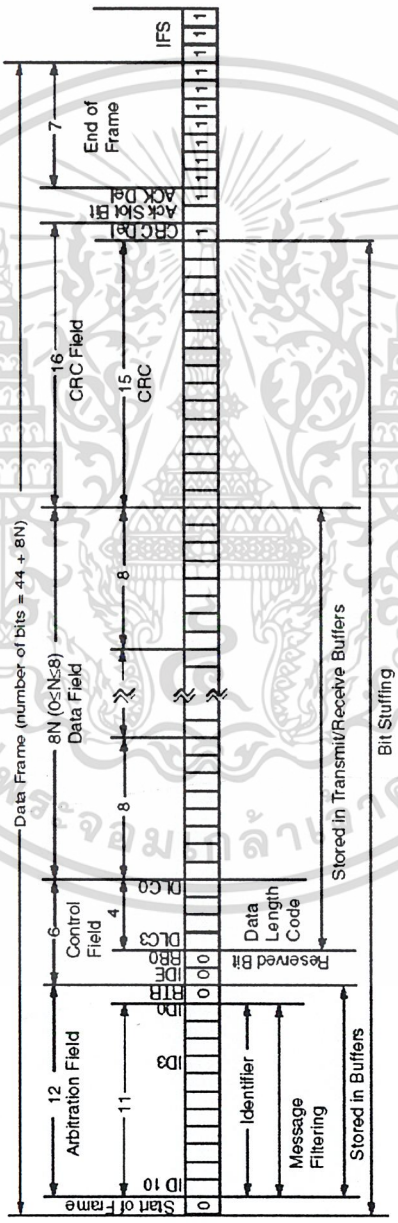
เมื่อส่งค่า CRC ครบทั้ง 15 บิต แล้วจะปิดท้ายฟิลด์ตรวจสอบนี้ด้วยบิต CRC Delimiter โดยจะส่งเป็นค่า '1' หรือรีเซสสีฟ

- ฟิลด์ตอบสนอง (Acknowledge Field) มีขนาด 2 บิต เป็นส่วนที่โหนดผู้รับใช้ตอบกลับไปยังโหนดผู้ส่งว่า ข้อมูลที่ส่งมาได้รับถูกต้องหรือไม่ ประกอบ ACK Slot 1 บิต และ ACK Delimiter 1 บิต

โหนดผู้ส่งจะส่งบิต ACK Slot นี้เป็น ‘1’ หรือรีเซสสีฟออกไป ในขณะที่โหนดผู้รับตรวจสอบแล้วว่าข้อมูลที่ได้รับถูกต้อง ก็จะส่งบิต ACK Slot เป็น ‘0’ หรือโดมิแนนต์ตอบกลับมา ซึ่งจะทำให้สถานะของบัสเป็นโดมิแนนต์ ดังนั้นเมื่อโหนดผู้ส่งอ่านค่าบิต ACK Slot กลับมาได้เป็น ‘0’ หรือโดมิแนนต์แสดงว่ามีอย่างน้อย 1 โหนดที่ได้รับข้อมูลได้อย่างถูกต้อง

ปิดท้ายฟิลด์ตอบสนองด้วยบิต ACK Delimiter ซึ่งมีค่าเป็น ‘1’ หรือ รีเซสสีฟเสมอ

- ฟิลด์สิ้นสุดเฟรม (End of Frame : EOF Field) ประกอบด้วยบิต ‘1’ หรือรีเซสสีฟจำนวน 7 บิตติดต่อกันหรือมีค่าเท่ากับ “1111111” เป็นส่วนที่ใช้แสดงว่าได้สิ้นสุดเฟรมนี้แล้ว



รูปที่ 3.4 ส่วนประกอบในเฟรมข้อมูลแบบมาตรฐาน

และเพื่อให้เวลากับโหนดผู้รับสำหรับการประมวลผลหรือจัดการกับข้อมูลที่ได้รับมาให้เป็นที่เรียบร้อยเสียก่อน ดังนั้นหลังจากที่สิ้นสุดเฟรมแล้ว จึงกำหนดว่าจะต้องส่งค่า '1' หรือรีเซสสีฟต่อท้ายฟิลด์สิ้นสุดเฟรมอย่างน้อย 3 บิต จึงจะสามารถส่งเฟรมข้อมูลใหม่ได้ต่อไป

เฟรมข้อมูลแบบขยาย (CAN 2.0B) ส่วนประกอบต่างๆ ในเฟรมข้อมูลแบบขยายจะต่างจากฟรมข้อมูลแบบมาตรฐานตรงฟิลด์แสดงรหัสข้อมูลและฟิลด์ควบคุมเท่านั้น ส่วนฟิลด์ที่เหลือจะเหมือนกัน สำหรับรายละเอียดในส่วนที่แตกต่างมีดังนี้

- ฟิลด์แสดงรหัสข้อมูล จะมีขนาด 32 บิต โดยแบ่งเป็นหมายเลข ID (Identifier) ขนาด 29 บิต (ในขณะที่เฟรมข้อมูลแบบมาตรฐานจะมีหมายเลข ID ขนาด 11บิต), 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 ส่วนโดยส่วนแรกมีขนาด 11 บิตเช่นนี้ ก็เพื่อให้สอดคล้องกับหมายเลข ID ที่ใช้ในเฟรมข้อมูลแบบมาตรฐานนั่นเอง

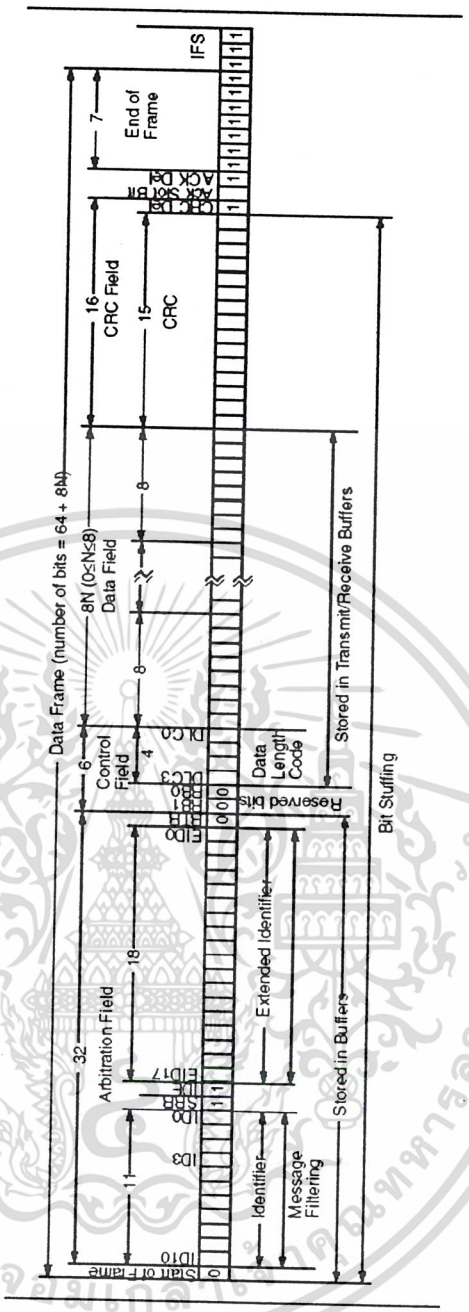
สาเหตุที่ต้องมีการขยายจำนวนหมายเลข ID เพิ่มขึ้น ก็เนื่องจากว่าจำนวนหมายเลข ID ในเฟรมข้อมูลแบบมาตรฐานที่มีอยู่ 2,048 หมายเลขนั้น แม้จะดูเหมือนมากแต่ในการใช้งานบางครั้งอาจจะไม่เพียงพอ จึงได้มีการกำหนดเฟรมข้อมูลแบบขยาย (CAN 2.0B) ขึ้นมาเพิ่มเติม โดยกำหนดให้หมายเลข ID มีขนาด 29 บิต ซึ่งจะรองรับการใช้งานได้มากถึง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

บิต SRR (Substitutc Rcmotc Rcquest) 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 RB1 (Reserved Bit 1) ซึ่งอยู่ในตำแหน่งเดียวกับบิต IDE ในฟิลด์ควบคุมของเฟรมข้อมูลแบบมาตรฐาน เนื่องจากบิต IDE ในเฟรมข้อมูลแบบขยายได้ถูกย้ายไปอยู่ในส่วนของฟิลด์แสดงรหัส ข้อมูลแล้ว ดังนั้นที่ตำแหน่งนี้ในฟิลด์ควบคุมของเฟรมข้อมูลแบบขยายจึงได้กำหนดให้เป็นบิต RB1 ซึ่งสำรองไว้ใช้งานในอนาคต โดยกำหนดให้มีค่าเป็น "0" หรือโดมิแนนต์เอาไว้



รูปที่ 3.5 ส่วนประกอบในเฟรมข้อมูลแบบขยาย

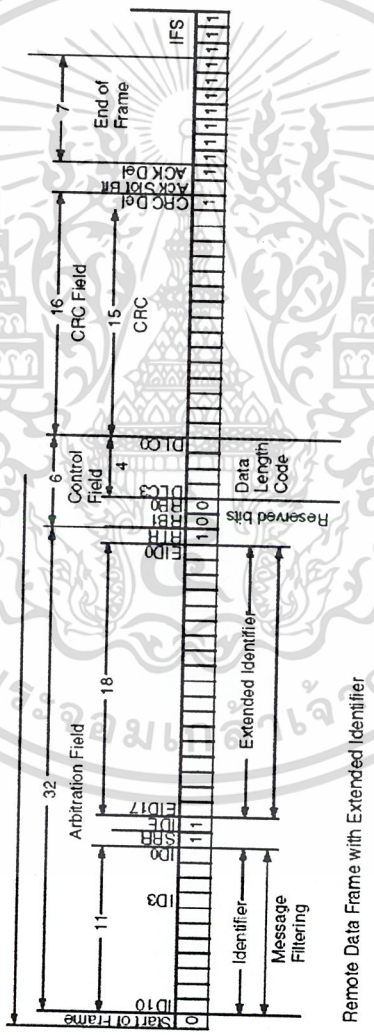
3.2.2 เฟรมร้องขอข้อมูล

โหนดผู้รับสามารถร้องขอข้อมูลจากโหนดที่มีข้อมูลนั้นอยู่ เพื่อให้โหนดนั้นส่งข้อมูลที่ต้องการนั้นมาให้ได้ โดยการส่งเฟรมร้องขอข้อมูลที่มีหมายเลข ID ของข้อมูลที่ต้องการนั้นออกไป หากโหนดใดมีข้อมูลที่มีหมายเลข ID ตรงกับที่ถูกร้องขอมา ก็จะทำการส่งข้อมูลนั้นตอบกลับไป

ลักษณะของเฟรมร้องขอข้อมูลจะคล้ายกับเฟรมข้อมูล แต่จะมีส่วนที่แตกต่างกับเฟรมข้อมูลอยู่สองประการด้วยกันคือ

- บิต RTR ในเฟรมร้องขอข้อมูลจะมีค่าเป็น “1” หรือรีเซสสีฟ ในขณะที่ในเฟรมข้อมูลจะมีค่าเป็น “0” หรือโดมิแนนต์
- ในเฟรมร้องขอข้อมูลจะไม่มีฟิลด์ข้อมูล

ข้อสังเกตอย่างหนึ่งก็คือ ในกรณีที่เฟรมร้องขอข้อมูลและเฟรมข้อมูลที่มีหมายเลข ID เดียวกันถูกส่งออกมาพร้อมกัน เฟรมข้อมูลจะมีสิทธิ์ได้ใช้บัสเนื่องจากบิต RTR จะมีค่าเป็น “0” หรือ โดมิแนนต์ ในขณะที่โหนดที่ส่งเฟรมร้องขอข้อมูลมาก็จะได้รับข้อมูลจากเฟรมข้อมูลนั้นทันที (เนื่องจากมีหมายเลข ID ตรงกัน)



รูปที่ 3.6 ส่วนประกอบในเฟรมร้องขอข้อมูล (แบบขยาย)

3.2.3 การเติมสตัฟฟ์บิต

เนื่องจากการส่งสัญญาณข้อมูลใน CAN จะใช้วิธีการส่งสัญญาณแบบ NRZ (Non-Return-to Zero) ที่ระดับสัญญาณในแต่ละบิตมีค่าคงที่ตลอดช่วงเวลาของบิตนั้น และเป็นการส่งข้อมูลแบบอะซิงโครนัส (Asyncronous) ที่ต้องอาศัยการเข้าจังหวะสัญญาณนาฬิกาจากสัญญาณข้อมูลที่ส่งมา การส่งข้อมูลด้วยวิธีนี้จะเกิดปัญหาขึ้นในกรณีที่สัญญาณข้อมูลมีค่าเดียวกันติดๆ กันหลายบิต ซึ่งจะทำให้การเข้าจังหวะกับสัญญาณข้อมูลนั้นมีความผิดพลาดเกิดขึ้นได้มาก

ดังนั้นเพื่อป้องกันไม่ให้เหตุการณ์เช่นนี้เกิดขึ้นใน CAN จึงได้กำหนดให้มีการเติมสตัฟฟ์บิต (Stuff Bit) คือถ้ามีการส่งบิตข้อมูลที่มีค่าเดียวกันติดกัน 5 บิตจุต้องคั่นด้วยบิตที่ตรงข้ามกับ 5 บิตนั้น 1 บิต และเมื่อโหนดผู้รับได้รับข้อมูลที่มีค่าเดียวกันติดกัน 5 บิต ก็จะไม่นับบิตที่ตามมา 1 บิตว่าเป็นบิตข้อมูล แต่จะเริ่มรับบิตข้อมูลต่อในบิตถัดไป

การเติมสตัฟฟ์บิตจะเกิดขึ้นในเฟรมข้อมูลและเฟรมร้องขอข้อมูลระหว่างบิตเริ่มต้นเฟรมกับบิต CRC Delimiter เท่านั้น ซึ่งนอกจากการเติมสตัฟฟ์บิตจะช่วยในการเข้าจังหวะสัญญาณข้อมูลแล้ว ยังใช้ตรวจสอบความผิดพลาดของข้อมูลได้อีกด้วย

3.3 การตรวจสอบและแก้ไขความผิดพลาด

จุดเด่นที่สำคัญอย่างหนึ่งของ CAN ก็คือ การตรวจสอบความผิดพลาดของข้อมูลที่เกิดขึ้น เพื่อให้เห็นภาพว่า CAN มีการตรวจสอบความผิดพลาดของข้อมูลที่มีประสิทธิภาพมากเพียงใด จะขอยกตัวอย่างดังต่อไปนี้ สมมติว่ามีการส่งข้อมูลด้วยอัตรา 500 กิโลบิตต่อวินาที โดยที่ทุก 0.7 วินาที จะมีความผิดพลาดเกิดขึ้น 1 บิต และมีการใช้งาน 8 ชั่วโมงต่อหนึ่งวัน ด้วยระบบการตรวจสอบความผิดพลาดใน CAN รับประกันได้ว่าในระยะเวลา 1,000 ปีจะมีความผิดพลาดที่ไม่สามารถตรวจสอบได้เกิดขึ้นเพียง 1 ครั้งเท่านั้น จะเห็นได้ว่าโอกาสที่ข้อมูลที่ส่งจะเกิดความผิดพลาดขึ้นโดยที่ตรวจไม่พบนั้นมีน้อยมาก

ความผิดพลาดที่ตรวจสอบได้มีอยู่ 5 อย่างด้วยกันคือ

- CRC Error ในขณะที่โหนดผู้ส่งทำการส่งข้อมูลจะคำนวณค่า CRC ของข้อมูลที่ส่งด้วยและส่งไปกับเฟรมข้อมูลนั้นในฟิลด์ตรวจสอบ ที่โหนดผู้รับจะทำการคำนวณค่า CRC ของข้อมูลที่ได้รับด้วยวิธีเดียวกันกับโหนดผู้ส่งแล้วนำไปเปรียบเทียบกับค่า CRC ที่ส่งมา หากไม่ตรงกันแสดงว่าข้อมูลที่ได้รับมีความผิดพลาดเกิดขึ้น ที่โหนดนั้นจะทำการส่งเฟรมแสดงความผิดพลาดแจ้ง กลับไปเพื่อบอกว่าข้อมูลที่ได้รับไม่ถูกต้อง เมื่อโหนดที่ส่งข้อมูลนั้นได้รับการแจ้งว่ามีความผิดพลาด เกิดขึ้น ก็จะทำการส่งข้อมูลนั้นออกไปใหม่อีกครั้งหนึ่ง

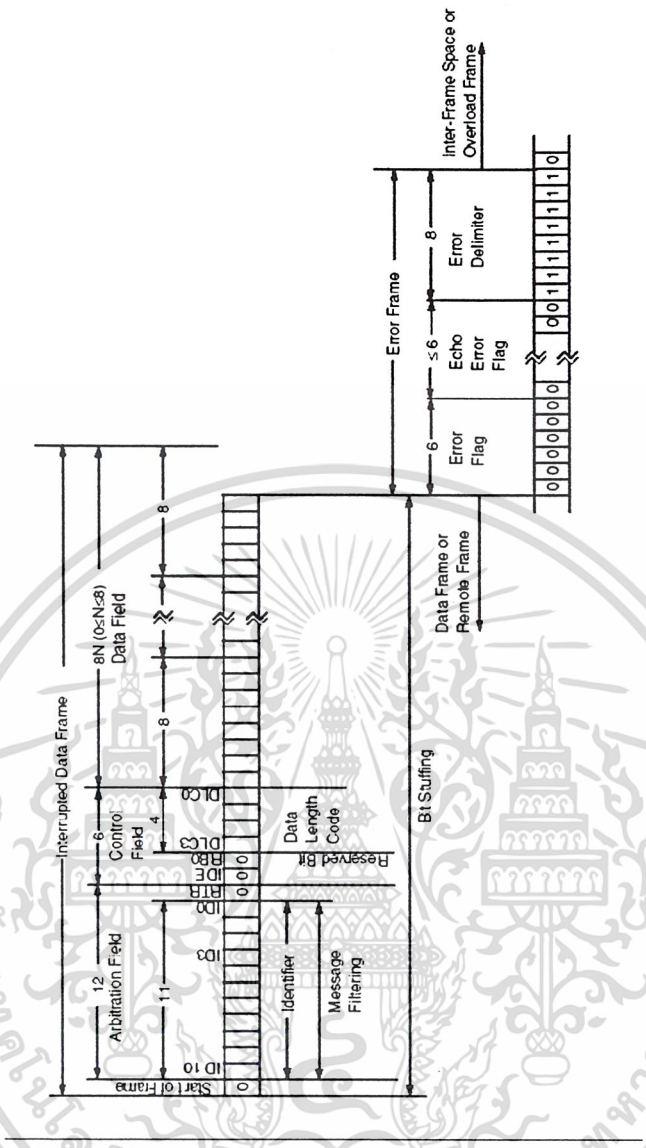
- Acknowledge Error โหนดผู้ส่งจะทำการตรวจสอบดูว่าที่บิต ACK Slot ในฟิลด์ตอบสนองมีค่าเป็น “0” หรือโดมิแนนต์หรือไม่ (ในขณะที่โหนดผู้ส่งเองจะส่งค่า “1” หรือรีเซสสีฟออกไป) หากมีค่าเป็นโดมิแนนต์แสดงว่ามีโหนดอย่างน้อยหนึ่งโหนดที่ได้รับข้อมูลถูกต้อง แต่ในทางตรงกันข้ามหากบิต ACK Slot มีค่าเป็นรีเซสสีฟแสดงว่าไม่มีโหนดใดที่ได้รับข้อมูลที่ถูกต้องเลย ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

- From Error ถ้าพบว่ามีสถานะโดมิแนนต์เกิดขึ้นในส่วนต่างๆ ต่อไปนี้ได้แก่ บิต CRC Delimiter, บิต ACK Delimiter และฟิลด์สิ้นสุดเฟรม แสดงว่ามีความผิดพลาดเกิดขึ้นในการส่งข้อมูลดังกล่าว (เนื่องจากส่วนต่างๆ เหล่านั้นต้องมีสถานะรีเซสสีฟเสมอ) ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

- Bit Error จะเกิดขึ้นเมื่อโหนดผู้ส่งอ่านค่าจากบัสกลับมาได้ค่าไม่ตรงกับค่าที่ส่งออกไป คือส่งค่าโดมิแนนต์แต่อ่านได้เป็นรีเซสสีฟ หรือส่งค่ารีเซสสีฟแต่อ่านได้เป็นโดมิแนนต์ สำหรับในกรณีที่ส่งค่ารีเซสสีฟแต่อ่านได้เป็นโดมิแนนต์ในฟิลด์แสดงรหัสข้อมูลและบิต ACK Slot นั้น ไม่ถือว่าเป็นความผิดพลาดเนื่องจากสามารถเกิดขึ้นได้ เมื่อพบความผิดพลาดจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

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

นอกจากความสามารถในการตรวจสอบความผิดพลาดที่เกิดขึ้นได้แล้ว CAN ยังได้มีการกำหนดสถานะความผิดพลาดของแต่ละโหนดในบัสขึ้นเพื่อควบคุมให้การใช้งานบัสมีประสิทธิภาพมากขึ้น กล่าวคือ ในกรณีที่โหนดใดมีความผิดพลาดเกิดขึ้นมากเกินกว่าเกณฑ์ที่กำหนดก็จะไม่อนุญาตให้ใช้บัสได้ เนื่องจากหากปล่อยให้โหนดดังกล่าวใช้บัสได้ตามปกติ จะทำให้การใช้งานบัสโดยรวมมีประสิทธิภาพต่ำลง เพราะจะเกิดความผิดพลาดขึ้นบ่อยครั้งซึ่งทุกครั้งจะต้องมีการส่งข้อมูลกันใหม่ ทำให้ปริมาณการใช้งานบัสมีมากและไปแย่งเวลากับโหนดอื่น โดยไม่จำเป็น

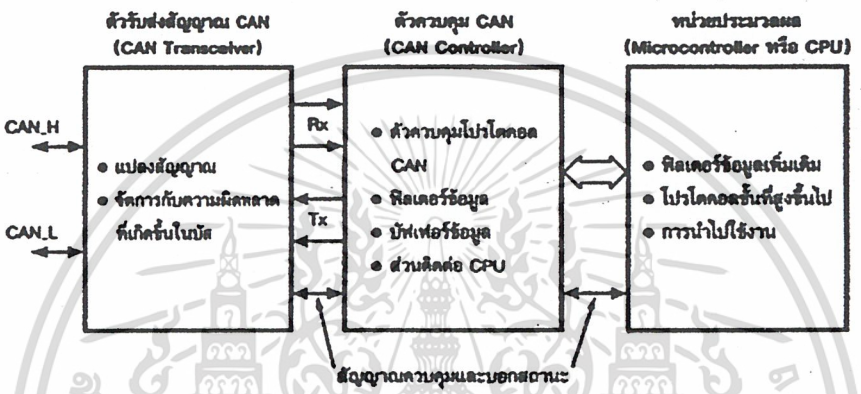


รูปที่ 3.7 ส่วนประกอบในเฟรมตรวจสอบความผิดพลาด

บทที่ 4
โครงสร้างของ CAN และการนำไปใช้

4.1 โครงสร้างโหนด CAN

โมเดลการทำงานของแต่ละโหนดในระบบบัส CAN โดยทั่งไปจะมีลักษณะโครงสร้างดังรูปที่ 4.1



รูปที่ 4.1 โมเดลการทำงานของแต่ละโหนดในระบบบัส CAN

ซึ่งมีส่วนประกอบหลักๆ อยู่ด้วยกัน 3 ส่วนด้วยกัน คือ

- หน่วยควบคุมและประมวลผล
- ตัวควบคุม CAN
- ตัวรับส่งสัญญาณ CAN

ในส่วนของการนำ CAN ไปใช้งานนั้น ปัจจุบันหลายบริษัทได้มีการออกแบบและพัฒนาตัวควบคุม CAN ให้อยู่ในรูปของวงจรรวมหรือไอซี โดยการทำงานของตัวควบคุม CAN ที่ได้ออกแบบขึ้นมาทั้งหมดนั้นจะอ้างอิงตามมาตรฐาน ISO 11898 ซึ่งเป็นส่วนของการทำงานในชั้นเชื่อมโยงข้อมูลที่ทำหน้าที่จัดการในเรื่องการสื่อสรข้อมูลขั้นพื้นฐาน เช่น การส่งข้อมูลและการร้องขอข้อมูล ให้เป็นไปอย่างถูกต้อง รวมไปถึงการตรวจสอบความผิดพลาดของข้อมูลด้วย ซึ่งจะอ้างอิงตามโมเดลการทำงานของตัวควบคุม CAN จากบริษัท Bosch ผู้ให้กำเนิด CAN ขึ้นมานั่นเอง

ส่วนประกอบอีกส่วนหนึ่งที่สำคัญที่ทำให้ CAN สามารถสื่อสารเป็นระบบบัสข้อมูลได้ก็คือ ตัวรับส่งสัญญาณ CAN ซึ่งจะทำหน้าที่แปลงข้อมูลที่ต้องการส่งให้เป็นสัญญาณที่สามารถส่งผ่านบัสได้ ตามข้อกำหนดที่มีขึ้นสำหรับการทำงานในชั้นกายภาพ

ตัวควบคุม CAN ที่มีลักษณะการทำงานที่รวมฟังก์ชันการควบคุมโปรโตคอลไว้ในไอซีตัวเดียว และแยกออกมาจากส่วนประมวลผลนั้น เราจะเรียกว่าเป็นตัวควบคุม CAN แบบ Stand-alone และในปัจจุบันได้มีบริษัทผู้ผลิตไอซีไมโครคอนโทรลเลอร์หลายค่ายที่ออกแบบให้มีตัวควบคุม CAN รวมอยู่ในไอซีไมโครคอนโทรลเลอร์เลย ซึ่งจะเรียกตัวควบคุม CAN ที่มีลักษณะเช่นนี้ว่าเป็นตัวควบคุมแบบ Intergrated

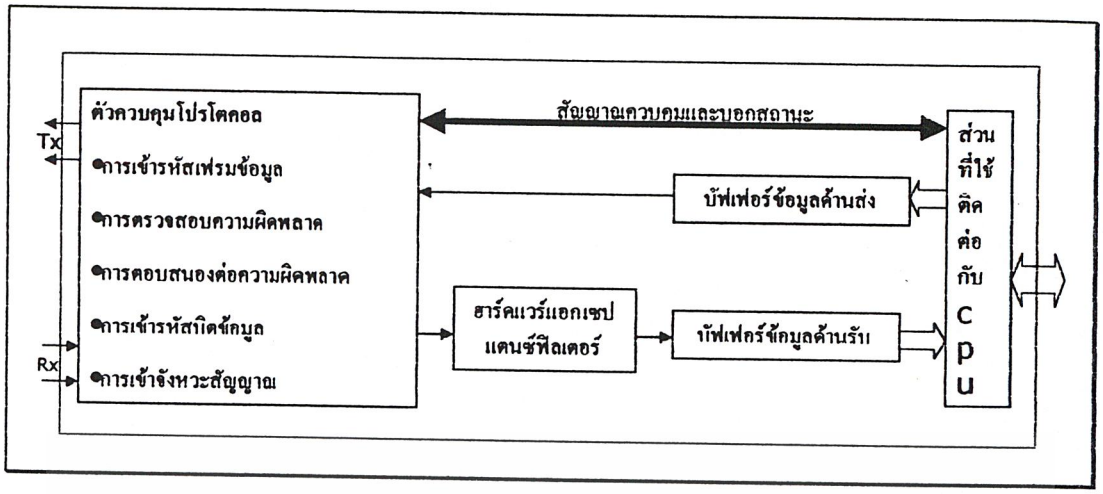
ตัวควบคุม CAN ทั้งสองแบบนี้มีข้อดีและจุดเด่นในการใช้งานที่แตกต่างกันออกไปโดยตัวควบคุม CAN แบบ Stand-alone นั้นมีข้อดีคือ สามารถนำไปใช้งานร่วมกับหน่วยประมวลผลหรือ CPU แบบใดก็ได้ตามความต้องการของผู้ใช้งานทำให้มีทางเลือกสำหรับการพัฒนาในส่วนของซอฟต์แวร์ที่หลากหลายกว่า ส่วนตัวควบคุม CAN แบบ Integrated นั้นมีข้อดีในเรื่องค่าใช้จ่ายในส่วนของการสร้างฮาร์ดแวร์ที่ต่ำกว่า ทั้งในด้านราคาของไอซีเองและค่าใช้จ่ายในการผลิตแผ่นวงจรพิมพ์ซึ่งมีขนาดและความซับซ้อนน้อยกว่า

นอกจากนี้การทำงานของ Integrated ยังมีข้อดีในเรื่องการใช้เวลาทำงานและการใช้ทรัพยากรของ CPU ที่น้อยกว่าด้วย เนื่องจากการควบคุมจะใช้การติดต่อผ่านบัสข้อมูล และแอดเดรสภายใน ในขณะที่ตัวควบคุมแบบ Stand-alone จะต้องอาศัยการถ่ายข้อมูลผ่านบัสข้อมูลและแอดเดรสภายนอก ซึ่งส่งผลให้การใช้ตัวควบคุมแบบ Stand-alone สิ้นเปลืองทรัพยากรของ CPU มากกว่าและทำงานได้ช้ากว่าด้วย

ส่วนโมเดลการทำงานของตัวควบคุม CAN ในอีกรูปแบบหนึ่งนั้นจะเป็นการรวมส่วนประกอบทั้งหมดเข้ามาไว้ในไอซีเพียงตัวเดียว คือจะรวมตัวรับส่งสัญญาณ CAN เข้ามาด้วย โดยเรียกตัวควบคุมแบบ Single-chip แนวโน้มของตัวควบคุม CAN ในลักษณะนี้เริ่มมีให้เห็นกันแล้วในปัจจุบันโดยเป็นการออกแบบตัวควบคุม CAN เพื่อใช้งานในระบบควบคุมสำหรับยานยนต์

4.2 โครงสร้างตัวควบคุม CAN

ตัวควบคุม CAN ถือเป็นส่วนประกอบสำคัญของโหนดในระบบ CAN ที่ทำหน้าที่ควบคุมกลไกในการรับส่งข้อมูลทั้งหมดให้เป็นไปตามมาตรฐานที่ได้กำหนดไว้ โดยการทำงานภายในตัวควบคุม CAN นั้นจะมีลักษณะของโครงสร้างหลักที่เหมือนกันดังรูปที่ 4.2

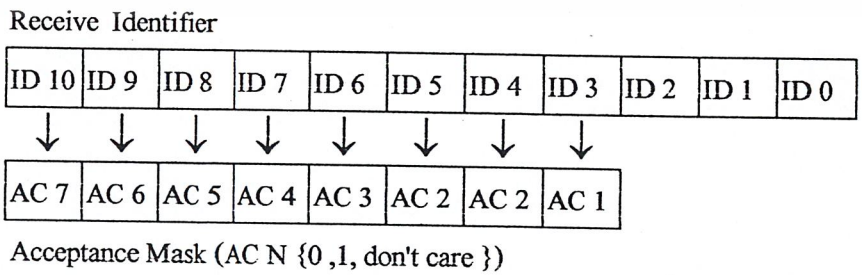


รูปที่ 4.2 โครงสร้างการทำงานของตัวควบคุม CAN

ประกอบด้วยตัวควบคุมโปรโตคอล CAN, ฮาร์ดแวร์แอกเซปแตนซ์ฟิลเตอร์ , บัฟเฟอร์ข้อมูล , และส่วนติดต่อกับ CPU ซึ่งแต่ละส่วนจะมีหน้าที่การทำงานดังนี้

- ตัวควบคุมโปรโตคอล CAN ทำหน้าที่ควบคุมกลไกการรับส่งข้อมูลทั้งหมดที่เกิดขึ้นในบัส CAN เช่น การสร้างเฟรมข้อมูล , การตรวจสอบความผิดพลาด , การเข้ารหัสบิตข้อมูล และการเข้าจังหวะสัญญาณ เป็นต้น ให้เป็นไปตามโปรโตคอลที่ได้กำหนดไว้ เพื่อให้ตัวควบคุม CAN ที่ไม่ว่าจะผลิตขึ้นมาจากบริษัทใดๆก็ตามสามารถนำมาใช้งานร่วมกันได้โดยไม่มีปัญหา

- 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 ของข้อมูลทั้งหมดนั้น ดูจะไม่เป็นเรื่องที่เหมะสมเท่าไรนัก แอกเซปแตนซ์ฟิลเตอร์นี้จะทำหน้าที่กลั่นกรองเอาเฉพาะข้อมูลที่ต้องการ (โดยพิจารณาจากหมายเลข ID) แล้วส่งไปให้ CPU พิจารณาอีกครั้งหนึ่ง ตัวอย่างในรูปที่ 4.3 เป็นแอกเซปแตนซ์ฟิลเตอร์อย่างง่ายขนาด 8 บิตที่ใช้ใน CAN 2.0 A



รูปที่ 4.3 เป็นแอกเซปแตนซ์ฟิลเตอร์อย่างง่ายขนาด 8 บิตที่ใช้ใน CAN 2.0 A

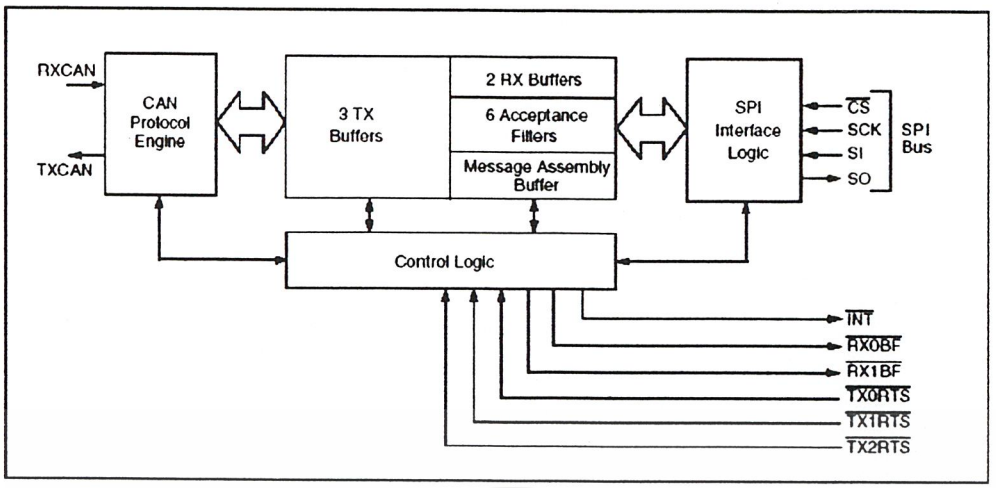
ซึ่งมีหมายเลข ID แบบมาตรฐานขนาด 11 บิต โดยมีแอกเซปแตนซ์มาสก์ เป็นตัวระบุหมายเลข ID ของข้อมูลที่ต้องการ

- บัฟเฟอร์ข้อมูล เพื่อให้การรับส่งข้อมูลทำได้โดยสะดวก โดยไม่ต้องคำนึงว่า CPU พร้อมที่จะทำงานหรือไม่ หรือมีโหนดอื่นๆ กำลังใช้งานบัสอยู่ และช่วยให้ข้อมูลไม่มีการสูญหายเมื่อ CPU ไม่พร้อมที่จะมาอ่านข้อมูลที่ได้รับในขณะนั้น จึงจำเป็นต้องมีบัฟเฟอร์ข้อมูลเพื่อทำหน้าที่เก็บข้อมูลไว้ชั่วคราว ทั้งข้อมูลที่ต้องการจะส่งและข้อมูลที่ได้รับมา โดยบัฟเฟอร์ของข้อมูลด้านส่งจะเก็บข้อมูลที่ต้องการส่งไว้ชั่วคราวจนกว่าบัสจะว่างพร้อมที่จะส่งข้อมูลไปได้ โดยที่ CPU ไม่จำเป็นต้องมาเสียเวลาตรวจสอบบัสอยู่ตลอดเวลา ในขณะที่บัฟเฟอร์ข้อมูลด้านรับจะเก็บข้อมูลที่ได้รับมาไว้ชั่วคราวจนกระทั่ง CPU พร้อมที่จะอ่านข้อมูลไป หรือไม่ต้องการข้อมูลนั้นแล้ว ตัวอย่างในรูปที่ 4.4 เป็นบัฟเฟอร์ข้อมูลอย่างง่ายที่ใช้ใน CAN 2.0 A ประกอบด้วยบัฟเฟอร์ข้อมูลขนาด 8 ไบต์ , หมายเลข ID ขนาด 11 บิต , ข้อมูล DLC ขนาด 4 บิต และบิต 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 2.0 A

- ส่วนติดต่อกับ CPU สำหรับตัวควบคุม CAN แบบ Stand-alone แม้ว่าจะสามารถทำงานได้โดยลำพัง แต่ก็ยังจำเป็นจะต้องมีการติดต่อกับอุปกรณ์ภายนอกในส่วนของการตั้งค่าเริ่มต้นในการทำงาน รวมทั้งการถ่ายโอนข้อมูลที่ต้องการรับส่งถึงกัน ซึ่งโดยทั่วไปแล้วมักจะนำตัวควบคุม CAN ไปใช้งานร่วมกับหน่วยประมวลผล คือ CPU หรือ ไมโครคอนโทรลเลอร์ต่างๆ จึงต้องมีส่วนที่ใช้ในการติดต่อกับ CPU เหล่านี้ วิธีการติดต่อนั้นมีอยู่หลายรูปแบบด้วยกัน มีทั้งแบบขนานและแบบอนุกรม ทั้งนี้เพื่อความสะดวกในการเลือกแบบที่จะนำไปใช้นั่นเอง



รูปที่ 4.5 โครงสร้างการทำงานภายใน MCP 2510 ตัวควบคุม CAN ของบริษัท Microchip

รูปที่ 4.5 เป็นตัวควบคุมแบบ Stand-alone คือไม่มี CPU อยู่ภายในไอซี ใช้งานได้ทั้งกับ CAN 2.0 A และ CAN 2.0 B ที่ความเร็ว 1 เมกะบิต ต่อวินาที มีแอกเซปแตนซ์ฟิลเตอร์ 6 ชุด และแอกเซปแตนซ์มาสก์ 2 ชุด มีบัฟเฟอร์ข้อมูลด้านส่ง 3 ชุด และบัฟเฟอร์ข้อมูลด้านรับ 2 ชุด และใช้ในการติดต่อกับ CPU ภายนอกผ่านการเชื่อมต่ออนุกรมแบบ SPI

บทที่ 5
การสร้างบอร์ดอินเตอร์เฟส CAN

5.1 โครงสร้างของบอร์ดอินเตอร์เฟส CAN

โครงสร้างของบอร์ดอินเตอร์เฟส CAN มีส่วนประกอบหลักอยู่สองส่วน ได้แก่ ตัวควบคุม CAN (CAN Controller) และตัวรับส่งสัญญาณ CAN (CAN Transceiver) โดยตัวควบคุม CAN จะทำหน้าที่หลักในการควบคุมกระบวนการทำงานให้เป็นไปตามโปรโตคอลที่กำหนดไว้ในชั้นเชื่อมโยงข้อมูล (Data Link Layer) ส่วนตัวรับส่งสัญญาณ CAN จะทำหน้าที่แปลงสัญญาณข้อมูลที่ต้องการส่ง (ลอจิก "0" หรือ "1") ให้เป็นสัญญาณที่ใช้งานในบัส CAN (สถานะโดมิแนนต์หรือรีเซสสีฟ) ซึ่งจะสอดคล้องกับมาตรฐานที่ได้กำหนดไว้ในชั้นกายภาพ (Physical Layer) ในขณะเดียวกันก็แปลงสัญญาณที่ได้รับจากบัส CAN กลับไปเป็นสัญญาณข้อมูลด้วย

สำหรับการออกแบบบอร์ดอินเตอร์เฟส CAN มีส่วนประกอบที่เพิ่มเข้ามาอีกส่วนหนึ่งคือ ตัวแยกวงจรด้วยแสงหรือออปโตคัปเปลอร์ (Opto-coupler) โดยส่วนนี้จะทำหน้าที่ป้องกันสัญญาณจากบัส CAN ซึ่งเป็นสัญญาณในภาคอะนาลอกไม่ให้เข้ามา รบกวนการทำงานของตัวควบคุม CAN ซึ่งเป็นการทำงานในภาคดิจิตอลนั่นเอง

5.2 คุณสมบัติของบอร์ดอินเตอร์เฟส CAN

CAN Controller: MCP2510

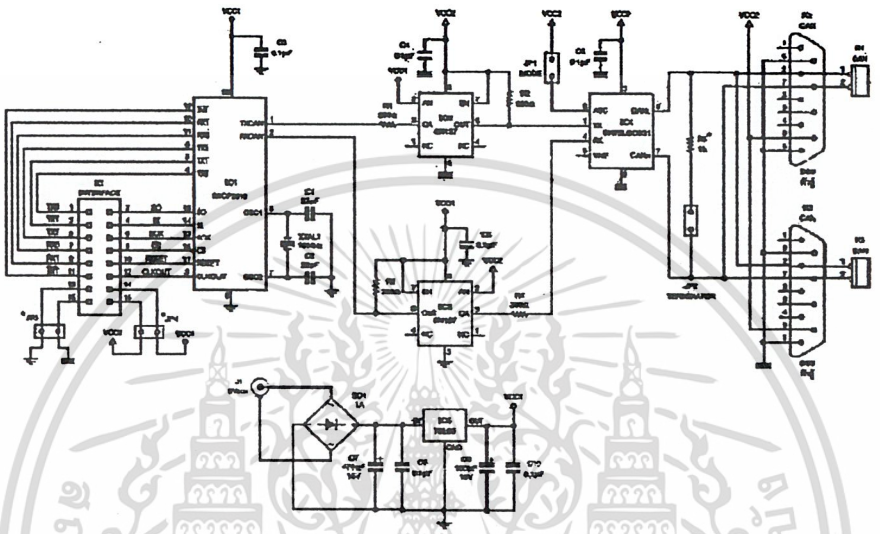
- ใช้งานได้ทั้ง CAN2.0A และ CAN2.0B ที่ความเร็วสูงสุด 1 MB/s
- บัฟเฟอร์ข้อมูลด้านส่ง 3 ชุด และด้านรับ 2 ชุด
- แอกแซปแตนซ์ฟิลเตอร์ 6 ฟุตและด้านรับ 2 ชุด
- รับส่งข้อมูลแบบอนุกรม SPI

CAN Transceiver: SN75LBC031

- รับส่งสัญญาณ CAN ตามมาตรฐาน ISO/DIS 11898
- รองรับความเร็วสูงสุด 500 kBaud

5.3 การทำงานของวงจร

วงจรของตัวอย่างบอร์ดอินเตอร์เฟส CAN ที่ได้ออกแบบขึ้นมานี้ไม่ได้มีความซับซ้อนแต่อย่างใด เนื่องจากวงจรในแต่ละส่วนได้ออกแบบและผลิตขึ้นมาอยู่ในรูปของวงจรรวมหรือไอซีเรียบร้อยแล้ว โดยรายละเอียดของวงจรบอร์ดอินเตอร์เฟส CAN นี้สามารถดูได้จากรูปที่ 5.1



รูปที่ 5.1 วงจรของบอร์ดอินเตอร์เฟส CAN

ไอซีตัวควบคุม CAN และตัวรับส่งสัญญาณ CAN นั้นมีให้เลือกใช้งานอยู่หลายแบบจากหลายๆ บริษัทที่ได้มีการผลิตออกมา แต่เหตุผลสำคัญที่เลือกใช้ไอซีต่างๆ ดังที่เห็นอยู่ในวงจรรูปที่ 5.1 นั้นก็เพราะว่าเป็นไอซีเบอร์ที่สามารถหามาใช้ได้นั่นเอง ไม่ว่าจะเป็นการสั่งซื้อหรือการขอตัวอย่างไอซีจากต่างประเทศ

ภาพรวมการทำงานของวงจรจะเป็นดังนี้คือ หากเป็นการส่งข้อมูลไปยังบัส CAN การทำงานจะเริ่มต้นด้วยการส่งข้อมูลที่ต้องการผ่านคอนเน็กเตอร์ K1 มาให้ IC1 ซึ่งทำหน้าที่เป็นตัวควบคุม CAN เพื่อจัดการกับข้อมูลที่ต้องการส่งนั้นแล้วทำการจัดส่งข้อมูลออกไปโดยให้เป็นไปตามโปรโตคอลของ CAN ข้อมูลที่ส่งออกมาจาก IC1 จะส่งผ่าน IC2 ซึ่งจะทำการแยกวงจรด้วยแสง (Opto-isolate) เพื่อป้องกันสัญญาณมารบกวนการทำงานของ IC1 ก่อนที่จะส่งสัญญาณข้อมูลนี้ไปยัง IC4 ซึ่งเป็นตัวรับส่งสัญญาณ CAN โดย IC4 จะทำหน้าที่แปลงสัญญาณข้อมูลที่ส่งให้เป็นสัญญาณที่สามารถส่งไปบนบัส CAN ได้

สำหรับการรับข้อมูลจากบัส CAN นั้น สัญญาณจากบัส CAN จะถูกส่งมาที่ IC4 ผ่านคอนเน็กเตอร์ K2, K3, K4 หรือ K5 สัญญาณจากบัส CAN จะถูกแปลงสัญญาณข้อมูล แล้วทำการแยกวงจรด้วย IC3 ก่อนที่สัญญาณข้อมูลจะถูกส่งไปยัง IC1 เพื่อจัดการกับข้อมูลที่ได้รับตามโปรโตคอลของ CAN ต่อไป

หัวใจหลักในการทำงานของบอร์ดอินเตอร์เฟส CAN นี้อยู่ที่ IC1 ที่ทำหน้าที่เป็นตัวควบคุมโปรโตคอลการรับส่งข้อมูลใน CAN โดยอาศัยการติดต่อเพื่อรับส่งข้อมูลกับอุปกรณ์ภายนอกทางพอร์ต SPI (Serial Peripheral Interface) นอกจากนี้ยังมีสัญญาณควบคุมต่างๆ เช่นสัญญาณร้องขอให้ส่งข้อมูลในบัฟเฟอร์ TXn (n = 1, 2 หรือ 3) และสัญญาณอินเตอร์รัปต์ เนื่องจากได้รับข้อมูลที่บัฟเฟอร์ RXn (n = 1 หรือ 2) เป็นต้น โดยสัญญาณควบคุม IC1 ทั้งหมดนี้จะเชื่อมต่อกับอุปกรณ์ภายนอกผ่านทางคอนเน็กเตอร์ K1 ซึ่งมีรายละเอียดดังตารางที่ 5.1

ทางด้านตัวรับส่งสัญญาณ CAN นั้น จะมีจั๊มเปอร์ JP1 ต่ออยู่ระหว่างที่ขา 8 ของ 14 IC4 กับไฟเลี้ยง VCC2 ใช้สำหรับเลือกว่าจะทำการส่งสัญญาณ CAN ในโหมดความเร็วสูงหรือความเร็วต่ำ โดยในโหมดความเร็วสูงจะปล่อยให้ขา 8 ของ IC4 ลอย ซึ่งจะสามารถส่งสัญญาณ CAN ได้ที่ความเร็วสูงสุด 500 กิโลบิตต่อวินาที ส่วนในโหมดความเร็วต่ำจะทำการต่อขา 8 ของ IC4 นี้เข้ากับไฟเลี้ยง โดยมีความเร็วในการส่งข้อมูลสูงสุดอยู่ที่ 125 กิโลบิตต่อวินาที ซึ่งการต่อขา 8 ของ IC4 เข้ากับไฟเลี้ยงในโหมดความเร็วต่ำนี้จะช่วยลดผลของ EMI ลง

ทางด้านสัญญาณเอาท์พุต CAN ของ IC4 ที่นำไปต่อกับบัส CAN นั้น จะมีการต่อ R5 เพื่อทำหน้าที่เป็นตัวต้านทานเทอร์มิเนเตอร์ (Terminator Resistance) ซึ่งจะช่วยในเรื่องการแมตชิ่งอิมพิแดนซ์ โดยมีจั๊มเปอร์ JP2 เป็นตัวเลือกว่าจะต่อตัวต้านทานเทอร์มิเนเตอร์นี้หรือไม่

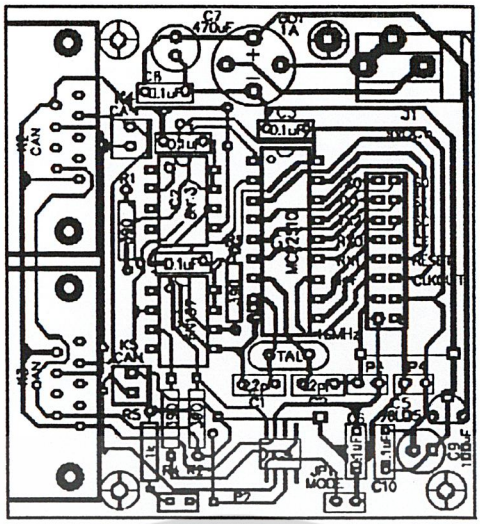
หมายเหตุ : หากไม่ต่อตัวต้านทานเทอร์มิเนเตอร์จะทำให้สัญญาณเอาท์พุต CAN ไม่เสถียรภาพ โดยจากผลการทดลองพบว่าขนาดของตัวต้านทานเทอร์มิเนเตอร์ที่เหมาะสมจะมีค่าประมาณ 1 กิโลโอห์ม

ตารางที่ 5.1 ชื่อและหน้าที่ของขาต่างๆ ในคอนเน็กเตอร์ K1 (INTERFACE)

ขา	ชื่อ	หน้าที่
1	TX0	ร้องขอให้ส่งข้อมูลในบัฟเฟอร์ TX0
2	SO	บิตข้อมูลเอาต์พุตของ SPI
3	TX1	ร้องขอให้ส่งข้อมูลในบัฟเฟอร์ TX1
4	SI	บิตข้อมูลอินพุตของ SPI
5	TX2	ร้องขอให้ส่งข้อมูลในบัฟเฟอร์ TX2
6	SCK	สัญญาณนาฬิกาเลื่อนข้อมูลของ SPI
7	RX0	อินเตอร์รัปท์จากบัฟเฟอร์ RX0
8	CS	สัญญาณควบคุมการทำงานของ SPI
9	RX1	อินเตอร์รัปท์จากบัฟเฟอร์ RX1
10	RESET	สัญญาณรีเซตยุปกรณ์
11	INT	อินเตอร์รัปรวมของระบบ
12	CLKOUT	สัญญาณนาฬิกาเอาต์พุต
13	GND1	กราวด์ที่ 1
14	VCC1	ไฟเลี้ยงที่ 1
15	GND2	กราวด์ที่ 2
16	VCC2	ไฟเลี้ยงที่ 2

ในส่วนของไฟเลี้ยงที่ใช้ในวงจรนั้น เนื่องจากออกแบบให้แยกวงจรภาคอนาลอกจากวงจรภาคดิจิตอล เพื่อป้องกันการรบกวนการทำงานระหว่างกัน จึงออกแบบให้ใช้ไฟเลี้ยง 2 ชุด โดยไฟเลี้ยงชุดแรกคือ VCC1-GND1 ใช้สำหรับเลี้ยงวงจรภาคดิจิตอล (IC1) ส่วนไฟเลี้ยงชุดที่สองคือ VCC2-GND2 ใช้สำหรับเลี้ยงวงจรภาคอนาลอก (IC4) ซึ่งปกติจะให้ VCC1 และ VCC2 มีขนาด 5 โวลต์ โดยต่อไฟเลี้ยงทั้งสองเข้ากับบอร์ดที่ขา 14 และ 16 และต่อ GND1 และ GND2 ที่ขา 13 และ 15 ของคอนเน็กเตอร์ K1 ตามลำดับ

ในกรณีที่ไม่มีแหล่งจ่ายไฟ 2 ชุด ก็สามารถใช้ไฟเลี้ยงชุดเดียวจากอะแด็ปเตอร์ไฟตรงขนาด 9 โวลต์ ต่อเข้าที่แจ็ค J1 โดยมีบริดจ์ไดโอด BD1 ทำหน้าที่ป้องกันการต่อไฟเลี้ยงกลับขั้ว ไฟเลี้ยงนี้จะถูกเรกกูเลตด้วย IC5 ให้เหลือขนาดแรงดัน 5 โวลต์ซึ่งต่ออยู่กับ VCC1 ไฟเลี้ยงที่ได้นี้สามารถนำไปเลี้ยงวงจรทั้งหมดได้โดยการเสียบจั๊มเปอร์ JP3 และ JP4



รูปที่ 5.2 การวางอุปกรณ์บนบอร์ด และลายทองแดง

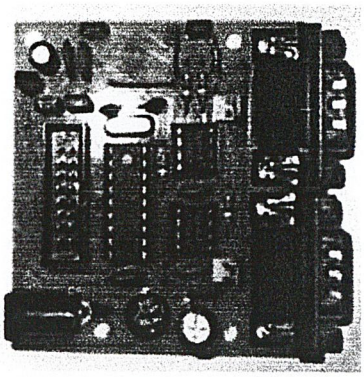
5.4 การสร้างบอร์ดอินเตอร์เฟส CAN

ปัญหาสำคัญที่พบในการสร้างบอร์ดอินเตอร์เฟส CAN ขึ้นมานั้น ไม่ได้อยู่ที่ความยากในการออกแบบวงจร เพราะมีหลายบริษัทได้ผลิตไอซีสำเร็จรูปขึ้นมารองรับการทำงานไว้หมดแล้ว แต่อยู่ที่ความยากในการหาอุปกรณ์หรือไอซีนั้นมาใช้ จากที่ได้พยายามเสาะหาดูตามร้านอิเล็กทรอนิกส์ในบ้านเรา ปรากฏว่าไม่มีการนำไอซีเหล่านี้เข้ามาจำหน่ายแต่อย่างใด สุดท้ายจึงต้องลงเอยด้วยการสั่งซื้อรวมทั้งการขอตัวอย่างไอซีจากต่างประเทศ ในที่สุดก็ได้ไอซีต่างๆ ขนครบ

โดยไอซี MCP2510 ซึ่งเป็นไอซีตัวควบคุม CAN ของบริษัท Nicrochip และไอซี 6N137 ซึ่งเป็นไอซีออปโตคัปเปลอร์ความเร็วสูงของบริษัท Fairchild Semiconductor ส่วนไอซี SN75LBC031 ซึ่งเป็นไอซีตัวรับส่งสัญญาณ CAN ของบริษัท Texas Instrument นั้น สามารถขอตัวอย่างไอซีได้ที่เว็บไซต์ http://www.ti.com

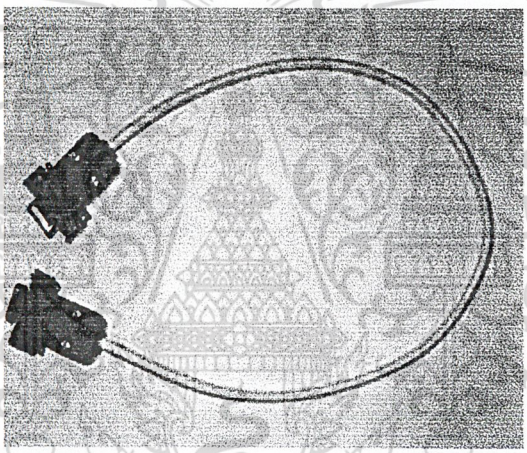
สำหรับแผ่นวงจรพิมพ์ของบอร์ดนั้น มีลายทองแดงและการวางอุปกรณ์ดังรูปที่ 5.2 โดยจุดที่ต้องระมัดระวังกันนอกจากเรื่องของการวางอุปกรณ์กลับขั้วแล้ว ที่ต้องระวังเป็นพิเศษก็คือการบัดกรี IC4 ซึ่งเป็นไอซีแบบ SMD (Surface Mount Device) เนื่องจากตัวถังมีขนาดเล็กบัดกรียากและต้องวางอุปกรณ์ไว้ด้านเดียวกับลายทองแดง

หลังจากที่สร้างบอร์ดอินเตอร์เฟส CAN เสร็จเรียบร้อยแล้ว อีกสิ่งหนึ่งที่ต้องสร้างขึ้นมาก็คือ สายอินเตอร์เฟสที่ใช้ในการทดสอบบอร์ดอินเตอร์เฟส CAN ที่สร้างขึ้น โดยนำมาทดสอบใช้งานร่วมกับคอมพิวเตอร์ผ่านทางพอร์ตพรินเตอร์

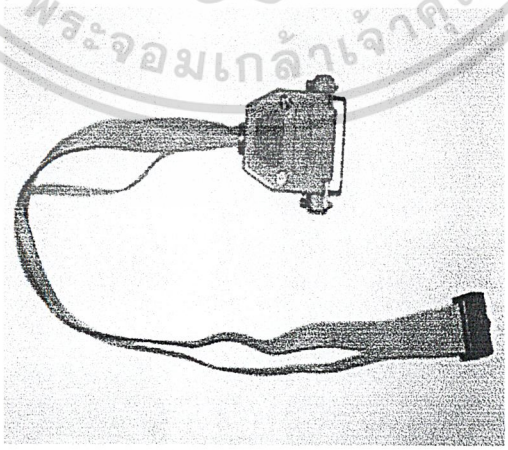


รูปที่ 5.3 บอร์ดอินเตอร์เฟส CAN ที่สร้างเสร็จเรียบร้อย

รูปที่ 5.4 สายอินเตอร์เฟส CAN ที่สร้างเสร็จเรียบร้อย



(ก) สายอินเตอร์เฟส ระหว่างโหนด CAN กับ โหนด CAN



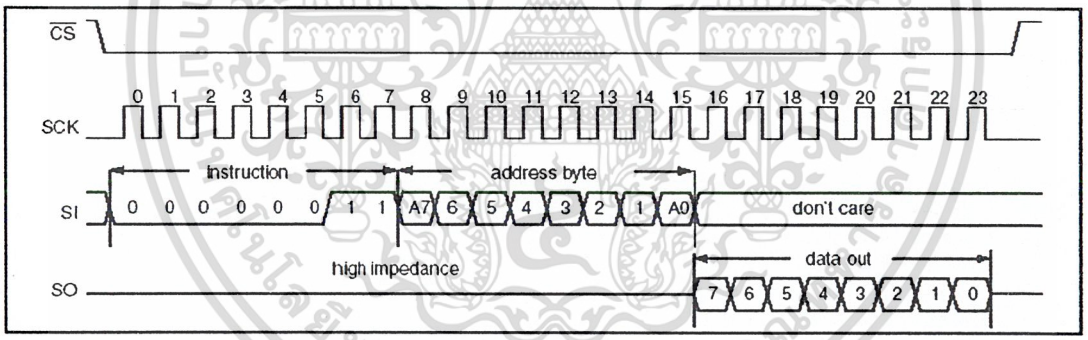
(ข) สายอินเตอร์เฟส ระหว่างโหนด CAN กับ Computer

5.5 การทดสอบการใช้งาน

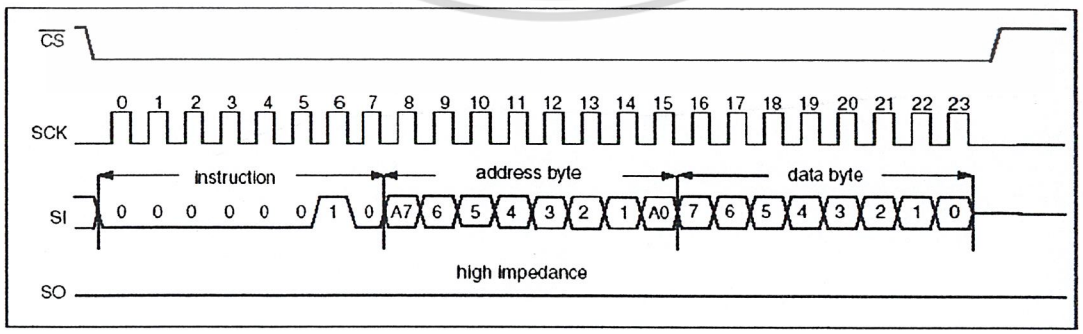
ในการควบคุมบอร์ดอินเตอร์เฟส CAN จะใช้การควบคุมโดยการกำหนดค่ารีจิสเตอร์ภายในและการตรวจสอบสถานะการทำงานของ IC1 ซึ่งทั้งหมดนี้จะใช้การควบคุมการรับส่งข้อมูลอนุกรมแบบ SPI ผ่านทางคอนเน็กเตอร์ K1 (Interface) ซึ่งจะมีคำสั่งต่างๆ ดังตารางที่ 5 โดยแต่ละคำสั่งจะมีการกำหนดสัญญาณควบคุมต่างๆ ดังรูปที่ 5.5

สำหรับการทดสอบการทำงานเบื้องต้นของบอร์ดอินเตอร์เฟส CAN ที่ได้สร้างขึ้นมานั้น เพื่อความสะดวกจึงได้ออกแบบการทดสอบโดยใช้คอมพิวเตอร์ผ่านทางพอร์ตพรินเตอร์ โดยการทดสอบเบื้องต้นนี้จะเป็นการทดสอบการรับส่งข้อมูลและการใช้คำสั่งต่างๆ ทางพอร์ต SPI และการสื่อสารสัญญาณบนบัส CAN ระหว่างบอร์ดอินเตอร์เฟส CAN จำนวน 3 บอร์ด ว่าสามารถรับส่งข้อมูลผ่านบัส CAN ระหว่างกันได้ถูกต้อง (หมายความว่า ต้องมีเครื่องคอมพิวเตอร์ที่มีพอร์ตพรินเตอร์อยู่ 3 เครื่อง)

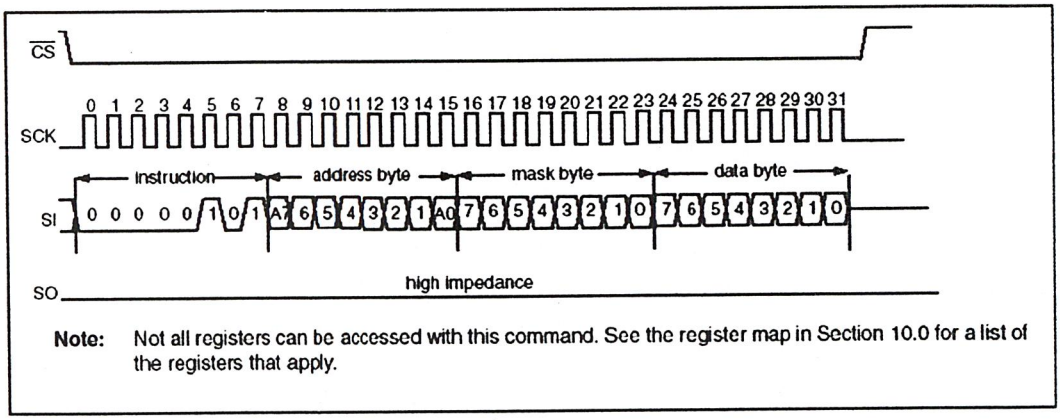
รูปที่ 5.5 รูปแบบของสัญญาณในการควบคุม IC1 ด้วย SPI



(ก) คำสั่ง READ

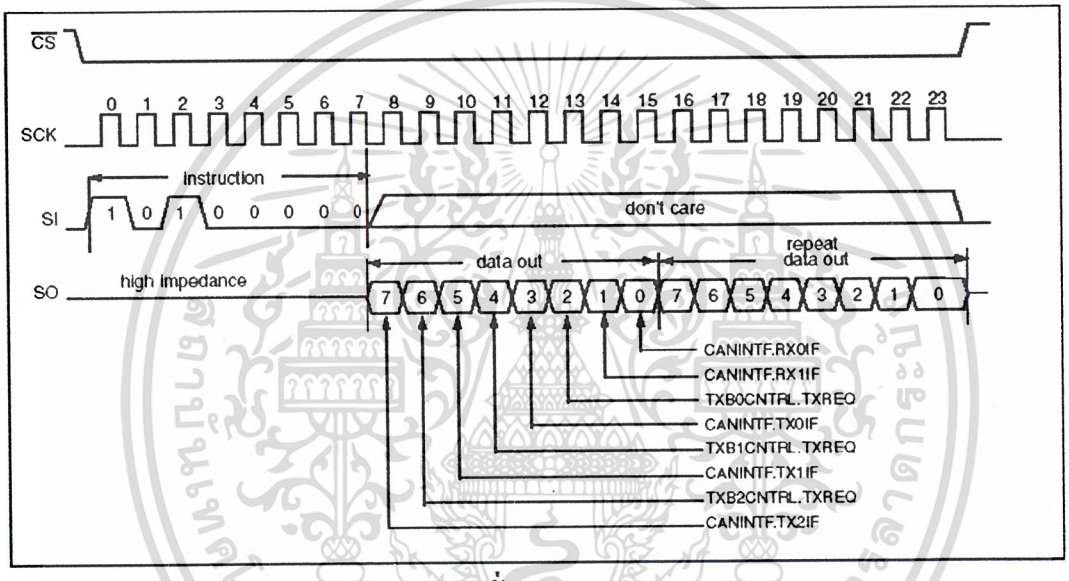


(ข) คำสั่ง WRITE

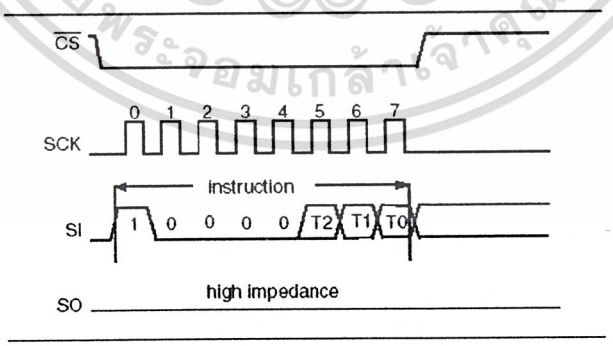


Note: Not all registers can be accessed with this command. See the register map in Section 10.0 for a list of the registers that apply.

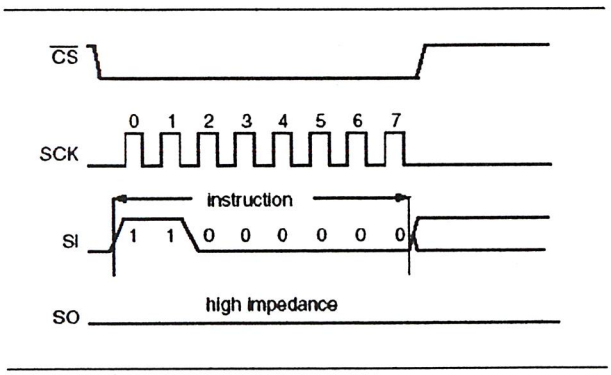
(ค) คำสั่ง Bit Modify



(ง) คำสั่ง Read Status



(จ) คำสั่ง Request To Send



(ฉ) คำสั่ง RESET

ข้อกำหนดในการทดสอบเบื้องต้นมีดังนี้

- ใช้หมายเลข ID แบบมาตรฐานขนาด 11 บิต (CAN2.0A)
- ไม่ได้ทดสอบในส่วนของแอกเซปแตนซ์ฟิลเตอร์และแอกเซปแตนซ์มากส์
- ใช้บัฟเฟอร์ข้อมูลด้านรับและด้านส่งเพียงด้านละหนึ่งชุด คือ TXB0 และ RXB0
- ส่งข้อมูลเพียงครั้งละ 1 ไบต์เท่านั้น
- ความเร็วในการส่งสัญญาณบนบัส CAN เท่ากับ 100 kBaud
- สำหรับโปรแกรมบนคอมพิวเตอร์ที่ใช้ทดสอบนั้นเขียนด้วยภาษา C

ก่อนที่จะเปิดโปรแกรมขึ้นมาจะต้องทำการเซตฮาร์ดแวร์หรือบอร์ดให้พร้อมเสียก่อน เริ่มจากการเซตจั๊มเปอร์ทั้งหมด ยกเว้น จั๊มเปอร์ JP2 ที่จะจับช็อตเพียง 2 บอร์ดที่อยู่ด้านปลายของบัส CAN ก็พอ เมื่อเซตจั๊มเปอร์เรียบร้อยก็ทำการต่อไปเลี้ยงจากอะแดปเตอร์เข้าบอร์ดที่แจ็ค J1 และต่อสายอินเตอร์เฟสกับพอร์ตพรินเตอร์ที่คอนเน็กเตอร์ K1 จากนั้นก็ต่อบอร์ดทั้งหมดเข้าด้วยกันบนบัส CAN ที่คอนเน็กเตอร์ตัวใดตัวหนึ่งระหว่างคอนเน็กเตอร์ DB9 ตัวผู้ K2, K3 หรือคอนเน็กเตอร์ 2 ขา K4, K5 (โดยที่ K4, K5 จะใช้สายสัญญาณเพียง 2 เส้นเท่านั้น)

ตารางที่ 5.2 คำสั่ง ที่ใช้ในการรับส่งข้อมูลด้วย SPI

Instruction Name	Instruction Format	Description
RESET	1100 0000	Resets internal registers to default state, set configuration mode
READ	0000 0011	Read data from register beginning at selected address
WRITE	0000 0010	Write data to register beginning at selected address
RTS (Request To Send)	1000 0nnn	Sets TXBnCTRL.TXREQ bit for one or more transmit buffers 1000 0nnn Request to send for TXB2 — Request to send for TXB0 Request to send for TXB1
Read Status	1010 0000	Polling command that outputs status bits for transmit/receive functions
Bit Modify	0000 0101	Bit modify selected registers

ตารางที่ 5.3 ความหมายของการเซ็ตจั๊มเปอร์ต่างๆ

ตำแหน่งจั๊มเปอร์	ช็อต	ไม่ช็อต
JP1	รับส่งสัญญาณ CANในโหมดความเร็วต่ำ (125 kBaud)	รับส่งสัญญาณ CAN ในโหมดความเร็วสูง (500 kBaud)
JP2	มีตัวต้านทานเทอร์มิเนเตอร์ (R5)	ไม่มีตัวต้านทานเทอร์มิเนเตอร์
JP3 , JP4	ใช้ไฟเลี้ยงเดียว (VCC1 = VCC2)	ใช้ไฟเลี้ยง สองชุด (Isolate)

เมื่อเซตบอร์ดเรียบร้อยแล้ว ก็เปิดโปรแกรม Testcan.exe ที่ใช้ทดสอบบนคอมพิวเตอร์ทั้ง 3 เครื่องขึ้นมา เริ่มต้นจะปรากฏข้อความว่า

CAN Module Tester V1.0

Enter ID of TX data (0-2047):

โปรแกรมจะให้กรอกหมายเลข ID ของข้อมูลที่ต้องการส่ง ซึ่งมีค่าตั้งแต่ 0-2,047 (เลขฐานสิบ) สมมติว่ากรอก “237” เมื่อกรอกหมายเลข ID เสร็จแล้วให้กดปุ่ม Enter จากนั้นโปรแกรมจะรอว่ามีการกดคีย์บอร์ดเพื่อส่งข้อมูลหรือมีข้อมูลบนบัส CAN จากบอร์ดอื่นส่งมาหรือไม่

หากมีการกดคีย์บอร์ดเพื่อส่งข้อมูลเช่นกด “0” บนหน้าจอคอมพิวเตอร์ เครื่องนั้นจะปรากฏข้อความว่า

TX data = 0

โดยโปรแกรมจะส่งค่าแอสกีของตัวอักษรที่กดนั้น (ในที่นี้คือ 30H) ไปที่บอร์ดเพื่อส่งค่าแอสกีนี้ไปที่บัส CAN ซึ่งบอร์ดอื่นๆ ที่เหลือก็จะได้รับค่าแอสกีนั้น เมื่อคอมพิวเตอร์ตรวจพบว่าบอร์ดได้รับข้อมูลก็จะอ่านข้อมูลนั้นและแสดงข้อมูลที่ได้รับพร้อมทั้งหมายเลข ID ของข้อมูลนั้นบนหน้าจอเป็นข้อความดังนี้

RX data = 0 ID = 237

หากการทดสอบให้ผลลัพธ์ดังที่ได้กล่าวมาข้างต้น แสดงว่าบอร์ดอินเตอร์เฟส CAN สามารถรับส่งข้อมูลได้อย่างถูกต้อง สามารถนำไปประยุกต์ใช้งานต่อไปได้

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<stdlib.h>

#define PORTA       0x378
#define PORTB       0x379

#define TXRTSCTRL   0x0d
#define BFPCTRL         0x0c
#define CANSTAT         0x0e
#define CANCTRL         0x0f
#define CNF1        0x2a
#define CNF2        0x29
#define CNF3        0x28

#define RXB0CTRL    0x60
#define RXB0SIDH    0x61
#define RXB0SIDL    0x62
#define RXB0EID8    0x63
#define RXB0EID0    0x64
#define RXB0DLC         0x65
#define RXB0DB0         0x66
#define RXB1CTRL    0x70
#define RXB1SIDH    0x71
#define RXB1SIDL    0x72
#define RXB1EID8    0x73
#define RXB1EID0    0x74
#define RXB1DLC     0x75
#define RXB1DB0     0x76
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
#define RXF0SIDH    0x00
#define RXF0SIDL    0x01
#dcfinc RXF0EID8    0x02
#define RXF0EID0    0x03

#define RXM0SIDH    0x20
#define RXM0SIDL    0x21
#define RXM0EID8    0x22
#define RXM0EID0    0x23

#define TXB0CTRL    0x30
#define TXB0SIDH    0x31
#define TXB0SIDL    0x32
#define TXB0EID8    0x33
#define TXB0EID0    0x34
#define TXB0DLC     0x35
#define TXB0DB0     0x36

#define CANINTE     0x2b
#define CANINTF     0x2c

unsigned char pbuf;

void shift_out8(unsigned char dbuf)
{
        unsigned char i,x;

        for(i=0;i<=7;i++)
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
        {
            x=dbuf;
            x=x&0x80;
            pbuf=pbuf&0x7f;
            pbuf=pbuf|x;                    // send si
            outportb(PORTA,pbuf);
            pbuf=pbuf&0xbf;                 // sck=0
            outportb(PORTA,pbuf);
            pbuf=pbuf|0x40;                 // sck=1
            outportb(PORTA,pbuf);
            dbuf=dbuf<<1;                   // shift data 1 bit
        }
}

unsigned char shift_in8(void)
{
        unsigned char i,x,dbuf;

        dbuf=0;

        for(i=0;i<=7;i++)
        {
            pbuf=pbuf&0xbf;                 // sck=0
            outportb(PORTA,pbuf);
            pbuf=pbuf|0x40;                 // sck=1
            outportb(PORTA,pbuf);
            x=inportb(PORTB);// receive so
            x=x&0x80;
            x=x^0x80;
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
        x=x>>i;                         // shift data 1 bit
        dbuf=dbuf|x;
    }
    return(dbuf);
}

unsigned char read_byte(unsigned char addr)
{
    unsigned char dbuf;

    pbuf=pbuf&0xdf;                     // cs=0
    outportb(PORTA,pbuf);
    shift_out8(0x03);                   // send instruction
    shift_out8(addr);           // send address
    dbuf=shift_in8();                   // receive data
    pbuf=pbuf|0x20;                     // cs=1
    outportb(PORTA,pbuf);
    return(dbuf);
}

void write_byte(unsigned char addr, unsigned char data)
{
    pbuf=pbuf&0xdf;                     // cs=0
    outportb(PORTA,pbuf);
    shift_out8(0x02);                   // send instruction
    shift_out8(addr);           // send address
    shift_out8(data);           // send data
    pbuf=pbuf|0x20;                     // cs=1
    outportb(PORTA,pbuf);
}
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
void bit_modify(unsigned char addr, unsigned char mask, unsigned char data)
{
        pbuf=pbuf&0xdf;                         // cs=0
        outportb(PORTA,pbuf);
        shift_out8(0x05);               // send instruction
        shift_out8(addr);       // send address
        shift_out8(mask);               // send mask
        shift_out8(data);       // send data
        pbuf=pbuf|0x20;                 // cs=1
        outportb(PORTA,pbuf);
}

void request_to_send(unsigned char data)
{
        pbuf=pbuf&0xdf;                         // cs=0
        outportb(PORTA,pbuf);
        shift_out8(data);       // send instruction
        pbuf=pbuf|0x20;                 // cs=1
        outportb(PORTA,pbuf);
}

unsigned char read_status(void)
{
        unsigned char dbuf;

        pbuf=pbuf&0xdf;                         // cs=0
        outportb(PORTA,pbuf);
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
        shift_out8(0xa0);               // send instruction
        dbuf=shift_in8();               // receive data
        dbuf=shift_in8();               // receive data again
        pbuf=pbuf|0x20;                 // cs=1
        outportb(PORTA,pbuf);

        return(dbuf);
}

void reset(void)
{
        pbuf=pbuf&0xdf;                 // cs=0
        outportb(PORTA,pbuf);
        shift_out8(0xc0);               // send instruction
        pbuf=pbuf|0x20;                 // cs=1
        outportb(PORTA,pbuf);
}

void main()
{
        unsigned char i,key,intf,data,sidh,sidl;
        unsigned int id;
        clrscr();

        pbuf=0x67;                      // reset=0
        outportb(PORTA,pbuf);
        delay(20);                      // delay 20 ms
        pbuf=0x77;                      // reset=1
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
outportb(PORTA,pbuf);

printf("CAN Module Tester V1.0\n");

/*** modify internal register ***/
write_byte(CANCTRL,0x83);
write_byte(CNF1,0x04);
write_byte(CNF2,0xf1);
write_byte(CNF3,0x05);
write_byte(TXRTSCTRL,0x00);
write_byte(BFPCTRL,0x0f);
write_byte(RXB0CTRL,0x00);
write_byte(RXB1CTRL,0x00);
write_byte(RXF0SIDH,0xff);
write_byte(RXF0SIDL,0xe0);
write_byte(RXM0SIDH,0x00);
write_byte(RXM0SIDL,0x00);
write_byte(RXM0EID8,0x00);
write_byte(RXM0EID0,0x00);
write_byte(TXB0CTRL,0x03);
write_byte(CANINTE,0x05);
write_byte(CANINTF,0x00);
write_byte(CANCTRL,0x03);
write_byte(TXB0DLC,0x01);
printf("Enter ID of TX data (0-2047): ");
scanf("%d",&id);
sidh=id/8;
sidl=id%8;
sidl=sidl*32;
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

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

key=0;

while(key!=0x1b)
{
    intf=read_byte(CANINTF);
    if((intf&0x01)==0x01)
    {
        data=read_byte(RXB0DB0);
        printf("RX data = %c  ",data);
        data=read_byte(RXB0SIDH);
        id=data*8;
        data=read_byte(RXB0SIDL);
        data=data&0xe0;
        id=id+(data/32);
        printf("ID = %d\n",id);
        bit_modify(CANINTF,0x01,0);
    }
    intf=read_byte(CANINTF);
    if((intf&0x04)==0x04)
    {
        bit_modify(CANINTF,0x04,0);
    }
    if(kbhit())
```

ซอร์สโคดของ โปรแกรมทดสอบบอร์ด อินเตอร์เฟส CAN (ต่อ)

```
        {
            key=getch();
            writc_bytc(TXB0DB0,kcy);
            request_to_send(0x81);
            printf("TX data = %c\n",key);
        }
    }
}
```

บรรณานุกรม

- Data sheet ของ CAN Controller เบอร์ MCP 2510 จากบริษัท Microship
- Data sheet ของ CAN Transceiver เบอร์ SN75LBC031 จากบริษัท Texas Instruments
- Data sheet ของ Opto-couper เบอร์ 6N137 จากบริษัท Fairchild Semiconductor
- Website http://www.microchip.com
- Website http://www-s.ti.com
- Website http://www.fairchildsemi.com
- Website อื่นๆ ที่ให้ข้อมูลเกี่ยวกับ CAN

ภาคผนวก

Lower Address Bits	Higher Order Address Bits							
	x000 xxxx	x001 xxxx	x010 xxxx	x0011 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

CAN Controller Register Map

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

Control Register Summary