การพัฒนาเครือข่ายระบบควบคุมโดยใช้ CAN

Control System Network Development Using CAN

โดย

นายกิตติชัย ศรียะวงษ์

นายธีรชัย ปรางค์ชัยภูมิ

อาจารย์ที่ปรึกษา

รศ.สมศักดิ์ มิตะถา

อ.อวัชริน นาชิน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเครือข่ายระบบควบคุมโดยใช้ CAN

Control System Network Development Using CAN

คณะผู้จัดทำ นายกิตติชัย ศรียะวงษ์ รหัส 43015351

นายธีรชัย ปรางค์ชัยภูมิ รหัส 43015360

_____ อาจารย์ที่ปรึกษา

(รศ.สมศักดิ์ มิตะถา)

_____ อาจารย์ที่ปรึกษา

(อ.อวัชริน นาชิน)

การพัฒนาเครือข่ายระบบควบคุมโดยใช้ CAN

นายกิตติชัย ศรียะวงษ์
นายธีรชัย ปรางค์ชัยภูมิ
รศ.สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษา
อ.อวัชริน นาชิน อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ได้ทำการออกแบบและพัฒนาเครือข่ายสำหรับระบบควบคุม โดยเริ่มจากการศึกษาการทำงานของ CAN โพรโตคอล, ศึกษาและกำหนด CAN High Layer Protocol จาก CAN High Layer Protocol ต่างๆที่มีอยู่ได้แก่ CANopen, DeviceNet, CAN Kingdom, OSEK/VDX, SDS และ J1939 จากนั้นทำการออกแบบเฟรมข้อมูลให้สนับสนุนตามข้อกำหนดของ CAN High Layer Protocol ที่ได้กำหนดไว้

เครือข่ายที่ได้พัฒนามีความสามารถเพิ่มเติมจาก CAN โพรโตคอลเดิมในหลายๆด้าน ได้แก่ การกำหนด ID ให้กับ Message เพื่อให้รับและส่งข้อมูลกันได้อย่างมีประสิทธิภาพ, สามารถส่งข้อมูลได้มากกว่า 8 ไบต์โดยใช้เทคนิคที่เรียกว่า Message on ID, การรายงานสถานะอุปกรณ์ในเครือข่ายซึ่งช่วยอำนวยความสะดวกให้ผู้ดูแลระบบในการตรวจสอบดูแลและสั่งงานผ่านระบบควบคุม, การตรวจสอบการนำส่งข้อมูล, สนับสนุนการเสริมสำรองเพื่อเพิ่มความน่าเชื่อถือและความยืดหยุ่นของระบบและเพิ่มความปลอดภัยในการรับส่งข้อมูล

Control System Network Development Using CAN

Mr.Kittichai Sriyawong
Mr.Teerachai Prangchaiyapoom
Assoc.Prof.Somsak Mitatha Advisor
Mr.Awacharin Nachin Advisor

ABSTRACT

In this thesis, we designed and implemented network for control system. We started by studying how is CAN protocol works. In addition, we studied and defined CAN High Layer Protocol from various available CAN High Layer Protocol such as CANopen, DeviceNet, CAN Kingdom, OSEK/VDX, SDS and J1939. Consequently, we designed data frames which supports the defined CAN High Layer Protocol specifications.

Developed network has many additional features over original CAN protocol in various ways such as message identification for the efficient data transmission, capability to transmit more than 8 bytes using message on ID technique, device status report for the convenience of network administrator to manage and control system, data transmission verification, support redundancy for improving the system liability, flexibility and security

กิตติกรรมประกาศ

การที่โครงงานนี้สามารถสำเร็จลงไปได้ด้วยดีนั้น เนื่องด้วยการให้ความสนับสนุนและคำแนะนำจากหลายๆฝ่าย ขอขอบพระคุณทุกๆท่านดังต่อไปนี้

ขอขอบพระคุณอาจารย์ทุกท่านในสถาบันที่ได้สั่งสอนคณะผู้จัดทำจนมีความรู้ ความสามารถจนถึงทุกวันนี้ รวมทั้งคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ ขอบคุณห้องวิจัยฮาร์ดแวร์ที่ได้เอื้อเฟื้อสถานที่ให้ผู้จัดทำได้ทำการวิจัยและช่วยอำนวยความสะดวกต่างๆ ขอขอบคุณ รศ.สมศักดิ์ มิตะถา, อ.ธนา หงสุวรรณ, อ.อวัชริน นาชิน, อ.เจริญ วงษ์ชุ่มเย็นและพี่อ๊อดที่ให้ความช่วยเหลือคณะผู้จัดทำในการทำงานตลอดเวลา ให้คำปรึกษาในยามที่เกิดปัญหาขึ้น ให้ความรู้และแนวคิดในการทำงาน และขอขอบคุณเพื่อนๆภาควิชาวิศวกรรมคอมพิวเตอร์ทุกคนที่เป็นกำลังใจมาโดยตลอด

ขอบพระคุณบริษัท วินนี่ไท จังหวัดระยอง ที่อนุญาตให้เข้าเยี่ยมชมระบบควบคุมที่ใช้งานอยู่จริง พร้อมทั้งคำแนะนำต่างๆเพื่อเป็นแนวทางในการทำโครงงาน

ที่สำคัญและขาดมิได้ คือ ต้องขอขอบพระคุณบิดา มารดาที่ได้ให้กำเนิด คอยสั่งสอน และให้การสนับสนุนการศึกษา กิจกรรมต่างๆ นับเป็นพระคุณที่หาใดเปรียบมิได้ ทางคณะผู้จัดทำขอกราบขอบพระคุณมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

หน้า

หน้า

สารบัญภาพ

ภาพที่ หน้า

ภาพที่ หน้า

ภาพที่ หน้า

สารบัญตาราง

ตารางที่ หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันนี้สิ่งประดิษฐ์ทางอิเล็กทรอนิกส์ต่างๆ ได้เข้ามาอยู่ในชีวิตประจำวันของเรามากขึ้น ล้วนแล้วแต่เพื่อเพิ่มความสะดวกให้กับการดำเนินชีวิตประจำวันทั้งสิ้น แต่เมื่อเริ่มมีอุปกรณ์เหล่านี้เพิ่มมากขึ้นก็จะทำให้เกิดความยุ่งยากในการควบคุมดูแลและการใช้งานอุปกรณ์เหล่านั้น

หากเรามีเครือข่ายที่สามารถเชื่อมโยงอุปกรณ์เหล่านั้นเข้าด้วยกัน และสามารถสั่งการได้จะเป็นการเพิ่มความสะดวกให้แก่ผู้ใช้งานยิ่งขึ้น ในปัจจุบันนี้เครือข่ายมีให้เลือกใช้กันหลายประเภทตั้งแต่เครือข่ายความสามารถสูงอย่างอีเธอร์เนต (Ethernet) ที่ใช้เป็นเครือข่ายในการแลกเปลี่ยนข้อมูลระหว่างเครื่องคอมพิวเตอร์ที่มีข้อมูลเป็นจำนวนมากและมีความเร็วสูง แต่เป้าหมายหลักของปริญญานิพนธ์นี้เป็นการสร้างเครือข่ายที่ใช้ในการควบคุมระบบในพื้นที่ โดยมีขนาดของข้อมูลไม่มากนัก ตัวอย่างของเครือข่ายอีกประเภทคือ RS-485 แต่เนื่องจาก RS-485 เป็นเพียงข้อกำหนดในการส่งสัญญาณผ่านสายนำสัญญาณเท่านั้น ไม่มีมาตรฐานในข้อตกลงในการสื่อสารจึงจำเป็นต้องกำหนดขึ้นเองทั้งหมด และได้พบว่ามีเครือข่ายอีกชนิดหนึ่งที่ใช้กันอย่างกว้างขวางในงานอุตสาหกรรมและยานยนต์คือ CAN (Control Area Network) ซึ่งเป็นบัสสื่อสารแบบอนุกรมที่ออกแบบมาสำหรับใช้ในงานควบคุมแบบเรียลไทม์ ที่มีจุดเด่นอยู่ที่ความสามารถในการสื่อสารข้อมูลเป็นเครือข่าย (Network) ระหว่างอุปกรณ์ โดยไม่ต้องมีหมายเลขที่อยู่ของโหนด (Node Addressing) อุปกรณ์ทุกตัวในเครือข่ายสามารถเรียกใช้งานบัสได้ (Multi-master) พร้อมกันหลายตัว ด้วยความเร็วในการสื่อสารข้อมูลถึง 1 เมกะบิตต่อวินาที มีระบบป้องกันและตรวจสอบความผิดพลาดที่มีประสิทธิภาพสูง อีกทั้งยังกำหนดให้ CAN เป็นบัสมาตรฐานระหว่างประเทศ นั่นคือมาตรฐาน ISO 11898 (ความเร็วสูง) และ ISO 11519 (ความเร็วต่ำ) ซึ่งเป็นการรับประกันถึงความสามารถและการยอมรับใน CAN เป็นอย่างดี

คุณสมบัติของเครือข่ายที่ต้องการในปริญญานิพนธ์ฉบับนี้

- เป็นเครือข่ายที่มีความเร็วในการสื่อสาร
- มีความน่าเชื่อถือและความยืดหยุ่น
- มีความปลอดภัยสำหรับข้อมูลในการส่งถ่ายข้อมูลโดยการเข้ารหัสข้อมูลก่อนนำส่ง
- มีการตรวจสอบการนำส่งข้อมูล
- มีการตรวจสอบความถูกต้องของข้อมูล
- สามารถเชื่อมต่อเข้ากับหน่วยประมวลผลหลักได้ตั้งแต่หน่วยประมวลที่มีความสามารถในการประมวลผลต่ำไปจนกระทั่งหน่วยประมวลผลที่มีความสามารถในการประมวลผลสูง

- มีความสลับซับซ้อนในการใช้งานต่ำเพื่อสนับสนุนการพัฒนาขึ้นใช้เองและง่ายในการพัฒนาเพิ่มเติม
- มีราคาไม่แพงจนเกินไป

จากคุณสมบัติทั้งหมด ในปริญญานิพนธ์ฉบับนี้เลือกที่จะใช้ CAN ในการสร้างเครือข่ายนี้ขึ้นมา เพื่อทำการทดลองและทดสอบ เนื่องจาก CAN เป็นที่นิยมใช้กันอย่างกว้างขวางโดยเฉพาะในงานอุตสาหกรรมและยานยนต์รวมไปถึงงานควบคุมเครื่องใช้ไฟฟ้าภายในอาคาร อีกทั้งยังมีความเหมาะสมกับงานสร้างเครือข่ายของอุปกรณ์อิเล็กทรอนิกส์ และยังมีความทนทานต่อสัญญาณรบกวน ถึงตัว CAN โพรโตคอลจะมีความสามารถหลายๆอย่างที่ต้องการให้อยู่แล้ว แต่ก่อนจะนำไปใช้งานจะต้องเพิ่มข้อกำหนดบางส่วนเข้าไปเนื่องจาก CAN โพรโตคอลครอบคลุมข้อกำหนดการทำงานในสองชั้น (Layer) แรกตามแบบจำลอง ISO/OSI เท่านั้นคือ ชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data link Layer) อีกทั้งต้องเข้ากับมาตรฐานเดิมที่มีอยู่และสามารถทำงานตามที่ต้องการได้โดยอาศัยพื้นฐานของ CAN เดิม

ดังนั้นในปริญญานิพนธ์ฉบับนี้ได้เริ่มจากการศึกษาการทำงานของ CAN โพรโตคอลถึงการทำงานและรูปแบบในการสื่อสาร จากนั้นจึงทำการ ศึกษาและกำหนด CAN High Layer Protocol จาก CAN High Layer Protocol ที่มีใช้งานอยู่ในปัจจุบัน แล้วจึงทำการสร้างวงจรเพื่อทำการทดสอบโพรโตคอลที่ออกแบบโดยใช้ไอซีเบอร์ MCP2510 ใช้ในการจัดการ CAN โพรโตคอลช่วยรับผิดชอบในการส่งข้อมูลในสองเลเยอร์แรกตามแบบจำลอง ISO/OSI ร่วมกับ CAN Transceiver เบอร์ UC5350 ทำหน้าที่ในการแปลงสัญญาณดิจิตอลที่รับมาจาก MCP2510 เป็นสัญญาณทางไฟฟ้าตามข้อกำหนดของ CAN โพรโตคอล และสุดท้ายพัฒนาโปรแกรมประยุกต์ (Application) ขึ้นมาทดสอบเครือข่าย โดยในที่นี้เลือกใช้ภาษาจาวาในการพัฒนาโปรแกรมบนเครื่องคอมพิวเตอร์ส่วนบุคคลและใช้ภาษาซีสำหรับการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์

ปริญญานิพนธ์ฉบับนี้จัดทำขึ้นเพื่อเป็นแนวทางในการสร้างเครือข่าย ในการควบคุมระบบโดยใช้ CAN โพรโตคอลและได้ทดลองเพิ่มเติมความสามารถต่างๆเข้าไป ตามวัตถุประสงค์ เพื่อเป็นตัวอย่างในการนำ CAN ไปใช้งาน

1.2 วัตถุประสงค์

1.2.1 ศึกษาในการทำงานของCANโพรโตคอลเพื่อใช้ในการประยุกต์กับการออกแบบเครือข่ายระบบควบคุม

1.2.2 ศึกษาและกำหนด CAN High Layer Protocol จาก CAN High Layer Protocol ต่างๆที่มีใช้งานอยู่ในปัจจุบัน

1.2.3 สร้างเครือข่ายที่มีความน่าเชื่อถือและความยืดหยุ่น มีความเร็วในการสื่อสาร มีความปลอดภัยสำหรับข้อมูลในการส่ง มีการตรวจสอบความถูกต้องของข้อมูล และตรวจสอบการนำส่งข้อมูล โดยใช้ CAN โพรโตคอล

1.2.4 สร้างโปรแกรมประยุกต์ใช้ในทดสอบโพรโตคอลที่ออกแบบ

1.3 ขอบเขต

1.3.1 กำหนด CAN High Layer Protocol และออกแบบเฟรมข้อมูลที่สนับสนุน CAN High Layer Protocol ที่กำหนด

1.3.2 ออกแบบสร้างวงจรฮาร์ดแวร์เพื่อใช้ในการทดสอบเครือข่ายที่ได้ออกแบบ

1.3.3 สร้างโปรแกรมประยุกต์เพื่อใช้ในการทดสอบเครือข่ายที่ได้ออกแบบ

1.4 ขั้นตอนการดำเนินงาน

1.4.1 ศึกษาการทำงานของ CAN โพรโตคอล

1.4.2 ศึกษาศึกษาและกำหนด CAN High Layer Protocol จาก CAN High Layer Protocol ต่างๆที่มีอยู่

1.4.3 ออกแบบฮาร์ดแวร์เชื่อมต่อกับคอมพิวเตอร์ส่วนบุคคลและฮาร์ดแวร์เชื่อมต่อกับไมโครคอนโทรลเลอร์

1.4.4 ออกแบบคลาสและกำหนดรายละเอียด (Specification) ของโปรแกรม

1.4.5 เขียนโปรแกรมทดสอบโพรโตคอลทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคล

1.4.6 เขียนโปรแกรมทดสอบโพรโตคอลทำงานบนไมโครคอนโทรลเลอร์

1.4.7 ทดสอบโปรแกรมและตรวจสอบข้อผิดพลาด

บทที่ 2

Control Area Network (CAN)

2.1 ความเป็นมาของ CAN

CAN หรือ Control Area Network เป็นบัสสื่อสารแบบอนุกรมที่ออกแบบมาสำหรับใช้ในงานควบคุมแบบเรียลไทม์ ที่มีจุดเด่นอยู่ที่ความสามารถในการสื่อสารข้อมูลเป็นเครือข่าย (Network) ระหว่างอุปกรณ์ โดยไม่ต้องมีหมายเลขที่อยู่ของโหนด (Node Addressing) อุปกรณ์ทุกตัวในเครือข่ายสามารถเรียกใช้งานบัสได้ (Multi-master) พร้อมกันหลายตัว ด้วยความเร็วในการสื่อสารข้อมูลถึง 1 เมกะบิตต่อวินาที มีระบบป้องกันและตรวจสอบความผิดพลาดที่มีประสิทธิภาพสูง

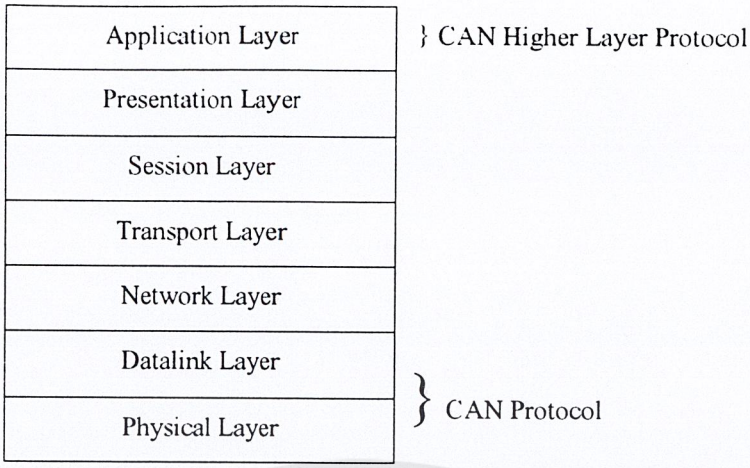
จุดเริ่มต้นของ CAN นั้นถูกพัฒนาขึ้นมาโดยบริษัท Robert Bosch ในประเทศเยอรมันเมื่อประมาณ 20 ปีผ่านมาสำหรับใช้ในระบบอิเล็กทรอนิกส์ภายในรถยนต์ เนื่องจากปัญหาในการพัฒนารถยนต์รุ่นใหม่ที่ต้องการสิ่งอำนวยความสะดวกและระบบรักษาความปลอดภัยเพิ่มขึ้น ส่งผลให้ระบบอิเล็กทรอนิกส์ภายในรถยนต์มีความซับซ้อนมากขึ้น มีโมดูลของวงจรต่างๆเป็นจำนวนมากที่ต้องมีการสื่อสารข้อมูลระหว่างกัน

ด้วยเหตุนี้จึงมีความต้องการบัสสื่อสารข้อมูลที่มีประสิทธิภาพและมีความน่าเชื่อถือสูงในราคาที่เหมาะสมแทนที่ระบบบัสเดิมที่มีความยุ่งยากซับซ้อนไม่สะดวกที่จะนำมาใช้และมีค่าใช้จ่ายสูง ซึ่งในภายหลังนอกจากจะใช้ในงานอุตสาหกรรมรถยนต์แล้วยังได้มีการนำ CAN ไปประยุกต์ใช้ในอุตสาหกรรมอื่นๆ อีกมากมาย เนื่องจากคุณสมบัติที่โดดเด่นของบัสนี้ในด้านความน่าเชื่อถือและความยืดหยุ่น รวมทั้งความง่ายต่อการใช้งานและมีค่าใช้จ่ายต่ำนั่นเอง

และเนื่องจากความแพร่หลายในการใช้บัสนี้ ภายหลังจึงได้มีการกำหนดให้ CAN เป็นบัสมาตรฐานระหว่างประเทศขึ้นมา นั่นคือ 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)

2.2 โครงสร้างการทำงาน

โครงสร้างการทำงานของ CAN เมื่อเทียบกับแบบจำลอง ISO/OSI นอกจากประกอบด้วยสองชั้นแรกดังที่กล่าวไปแล้ว ยังประกอบด้วยชั้นที่ 7 คือชั้นประยุกต์ใช้งาน (Application Layer) โดยจะไม่มีส่วนประกอบของชั้นที่เหลือ คือชั้นที่ 3-6 ลักษณะโครงสร้างการทำงานจะเป็นดังรูปที่ 2.1



รูปที่ 2.1 โครงสร้างการทำงานของ CAN

2.2.1 ชั้นที่ 1 Physical Layer โครงสร้างการทำงานในชั้นนี้จะเป็นส่วนที่เกี่ยวข้องกับส่วนประกอบทางกายภาพของบัส ได้แก่ ตัวกลางที่ใช้ส่งผ่านสัญญาณข้อมูล คอนเน็กเตอร์ และการเชื่อมต่อสายสัญญาณ รวมทั้งคุณสมบัติทางไฟฟ้าของสัญญาณข้อมูล (แรงดัน/กระแส) ของสัญญาณมาตรฐาน คือ ISO 11519 สำหรับการสื่อสารข้อมูลความเร็วต่ำ คือมีอัตราเร็วอยู่ในช่วง 5 ถึง 125 กิโลบิตต่อวินาที และ ISO 11898 สำหรับการสื่อสารความเร็วสูง สนับสนุนการสื่อสารข้อมูลด้วยความเร็วสูงสุดถึง 1 เมกะบิตต่อวินาที

2.2.2 ชั้นที่ 2 Data Link Layer จะเกี่ยวข้องกับโพรโตคอลและรูปแบบของข้อมูลที่สื่อสารกันในเชิงตรรกะ รวมถึงการป้องกันความผิดพลาดที่จะเกิดขึ้นกับข้อมูล เพื่อรับประกันว่าข้อมูลที่ได้มีความถูกต้องโดยข้อกำหนดของการทำงานในชั้นนี้ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วย และได้มีการแก้ไขขยายข้อกำหนดของมาตรฐานในชั้นนี้เพิ่มเติม โดยจัดเป็น 2 เวอร์ชัน คือ CAN2.0A (เดิม) และ CAN2.0B (ขยายเพิ่มเติม)

2.2.3 ชั้นที่ 7 Application Layer เป็นส่วนของการนำ CAN ไปประยุกต์ใช้งานบนพื้นฐานการทำงานในสองชั้นแรก แม้ว่าการทำงานในชั้นนี้จะไม่ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วย แต่ก็ได้มีการสร้างโพรโตคอลสำหรับการทำงานในชั้นนี้ขึ้นมาซึ่งมีอยู่ด้วยกันหลายรูปแบบ ตัวอย่างโพรโตคอลที่ใช้งานกัน เช่น CANopen, DeviceNet, CAN Kingdom, OSEK/VDX, SDS และ J1939 เป็นต้น โดยแต่ละโพรโตคอลจะทำงานอยู่บนแนวความคิดเดียวกัน

2.3 คุณสมบัติของ CAN

2.3.1 มาตรฐาน ISO11898 พัฒนาขึ้นโดย Robert Bosch GmbH

2.3.2 โครงสร้างการทำงานมีสองชั้น คือชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data Link Layer)

2.3.3 สื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส

2.3.4 สร้างเครือข่ายแบบ Multi-master/Broadcasting

2.3.5 สนับสนุนการทำงานแบบเรียลไทม์

2.3.6 ใช้หลักการ CSMA/CD ในการเข้าใช้บัส

2.3.7 ไม่มีการกำหนดหมายเลขที่อยู่กำกับโหนด

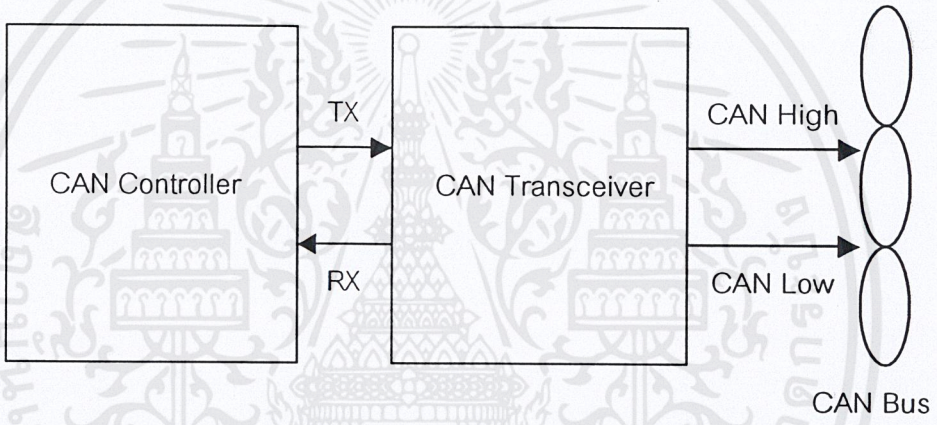
2.3.8 ส่งข้อมูลครั้งละ 0-8 ไบต์ ด้วยอัตราเร็วสูงสุด 1 เมกะบิตต่อวินาที

2.3.9 มีความปลอดภัยในการส่งข้อมูลสูง

2.3.10 ตัวควบคุมโพรโตคอลสามารถบรรจุอยู่ในชิปเดียว

2.3.11 ต้นทุนต่ำ ใช้งานสะดวก บำรุงรักษาง่าย

2.3.12 มีการใช้งานอย่างแพร่หลายในงานอุตสาหกรรมและระบบอิเล็กทรอนิกส์ในยานยนต์



รูปที่ 2.2 ลักษณะการเชื่อมต่ออุปกรณ์ (Topology) ใน CAN

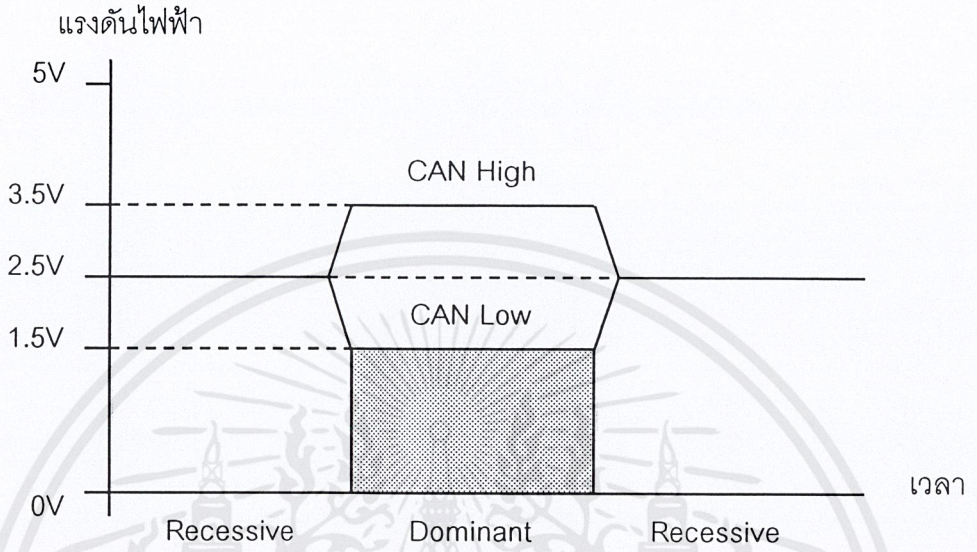
2.4 ลักษณะทางกายภาพ

ตัวกลางที่ใช้ส่งผ่านข้อมูลใน CAN นั้นมีอยู่ด้วยกันหลายรูปแบบตั้งแต่การใช้สายสัญญาณ 2 เส้น การใช้สายสัญญาณเส้นเดียว การใช้สายไฟเบอร์ออปติก หรือแม้แต่การส่งสัญญาณแบบไร้สาย แต่ที่นิยมใช้กันมากก็คือ การใช้สายสัญญาณ 2 เส้นแบบตีเกรียว (Twisted-pair Cable)

การส่งผ่านข้อมูลใน CAN มีรูปแบบของการส่งข้อมูลเป็นแบบอนุกรมที่ใช้สายสัญญาณเพียง 2 เส้น ซึ่งมีลักษณะการเชื่อมต่อสายสัญญาณ (topology) ดังรูปที่ 2.2 โดยอุปกรณ์ทุกตัวจะต่ออยู่บนสายสัญญาณคู่เดียวกัน และปิดปลายคู่สายสัญญาณทั้งสองข้างด้วยเทอร์มิเนชันอิมพิแดนซ์ (Termination Impedance)

สัญญาณที่ส่งจะใช้วิธีการส่งแบบแรงดันผลต่าง (Differential Voltage Signal) สถานะของสัญญาณในสายส่งได้จากการเปรียบเทียบระดับแรงดันหรือศักย์ไฟฟ้าระหว่างสายสัญญาณทั้งสองเส้น CAN_H (ศักย์สูง) และ CAN_L (ศักย์ต่ำ)

สถานะของสัญญาณข้อมูลในสายส่ง CAN จะมีอยู่ 2 สถานะคือ สถานะรีเซสสีฟ (Recessive) และสถานะโดมิแนนต์ (Dominant) ระดับแรงดันไฟฟ้าของสัญญาณในแต่ละเส้น (เทียบกราวนด์) และผลต่างแรงดันไฟฟ้าระหว่างคู่สายสัญญาณในแต่ละสถานะจะเป็นดังรูปที่ 2.3



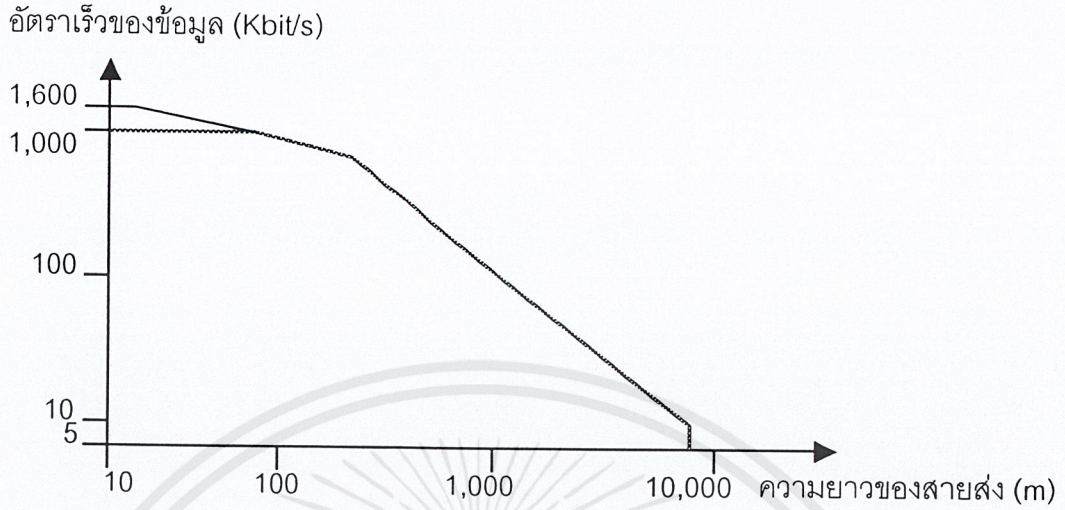
รูปที่ 2.3 ลักษณะของสัญญาณข้อมูลแต่ละสถานะ

2.4.1 ถ้าผลต่างระหว่าง CAN_H และ CAN_L เป็น 0 โวลต์ (หรือ $\leq$0.5 โวลต์) จะเป็น รีเซสสีฟ ซึ่งจะแทนข้อมูลที่มีลอจิกเป็น '1'

2.4.2 ถ้าผลต่างระหว่าง CAN_H และ CAN_L เป็น 2 โวลต์ (หรือ $\geq$0.9 โวลต์) จะเป็น โดมิแนนต์ ซึ่งจะแทนข้อมูลที่มีลอจิกเป็น '0'

การส่งผ่านข้อมูลโดยใช้ผลต่างแรงดันของคู่สายสัญญาณมีข้อดีคือ จะช่วยลดการรบกวนอันเนื่องมาจากผลของ EMI (Electromagnetic Interference) ลงได้เป็นอย่างมาก ทำให้สามารถส่งสัญญาณได้ในอัตราเร็วสูงขึ้นและได้ระยะทางไกลมากขึ้น

อัตราเร็วในการส่งผ่านข้อมูลและระยะทางที่ส่งได้จะแปรผกผันกันดังรูปที่ 2.4 ตามมาตรฐาน ISO 11898 การส่งข้อมูลสูงสุดคือ 1 เมกะบิตต่อวินาทีที่ความยาวของสายส่งไม่เกิน 40 เมตร สำหรับค่าความต้านทานและขนาดของสายส่ง รวมทั้งขนาดของเทอร์มิเนชันอิมพิแดนซ์ ตัวอย่างเช่น ที่อัตราเร็วข้อมูล 1 เมกะบิตต่อวินาที ควรใช้สายส่งที่มีความต้านทานขนาด 70 มิลลิโอห์มต่อเมตรและมีขนาดเส้นผ่านศูนย์กลางของสายส่งมากกว่า 0.25 ตารางมิลลิเมตร โดยต่อเทอร์มิเนชันอิมพีแดนซ์ที่มีค่า 124 โอห์มไว้ที่ปลายของสายทั้งสองด้าน



รูปที่ 2.4 กราฟความสัมพันธ์ระหว่างอัตราเร็วข้อมูลกับระยะทางที่ส่งได้

ข้อควรระวังในการต่อสายสัญญาณก็คือ ในกรณีที่มีการต่อสายสัญญาณพ่วงจากบัสกลาง สายพ่วงควรมีความยาวไม่เกิน 2 เมตรเมื่ออัตราเร็วในการส่งข้อมูลเป็น 250 กิโลบิตต่อวินาที และไม่ควรเกิน 30 เซนติเมตรถ้าอัตราเร็วของข้อมูลสูงกว่านั้น ทั้งนี้ความยาวของสายพ่วงทุกเส้นรวมกันไม่ควรเกิน 30 เมตร

2.5 หลักการเบื้องต้น

รูปแบบการสื่อสารข้อมูลของ CAN เป็นแบบอนุกรมอะซิงโครนัส โดยที่อุปกรณ์แต่ละตัวหรือโหนด (Node) สามารถส่งข้อมูลหรือข้อความ (Message) ไปยังโหนดอื่นๆได้ เหตุที่เรียกว่าเป็นการสื่อสารข้อมูลแบบอะซิงโครนัสเนื่องจากไม่มีการส่งสัญญานนาฬิกาเพื่อใช้ในการเข้าจังหวะ แต่อาศัยการเข้าจังหวะจากจุดเริ่มต้นของแต่ละข้อความที่ส่งไป (ลักษณะเช่นเดียวกับการส่งข้อมูลแบบ RS232)

ในส่วนของระเบียบวิธีหรือโพรโตคอลในการสื่อสารข้อมูลในชั้นเชื่อมโยงข้อมูลของ CAN นั้นมีหลักการสำคัญที่ทำให้ CAN เป็นบัสข้อมูลที่ทำงานได้อย่างมีประสิทธิภาพ ได้แก่ การสื่อสารแบบบรอดคาส (Broadcast Communication), การสื่อสารที่อิงจากข้อความ (Message-based Communication) และการป้องกันความผิดพลาด (Error Protection)

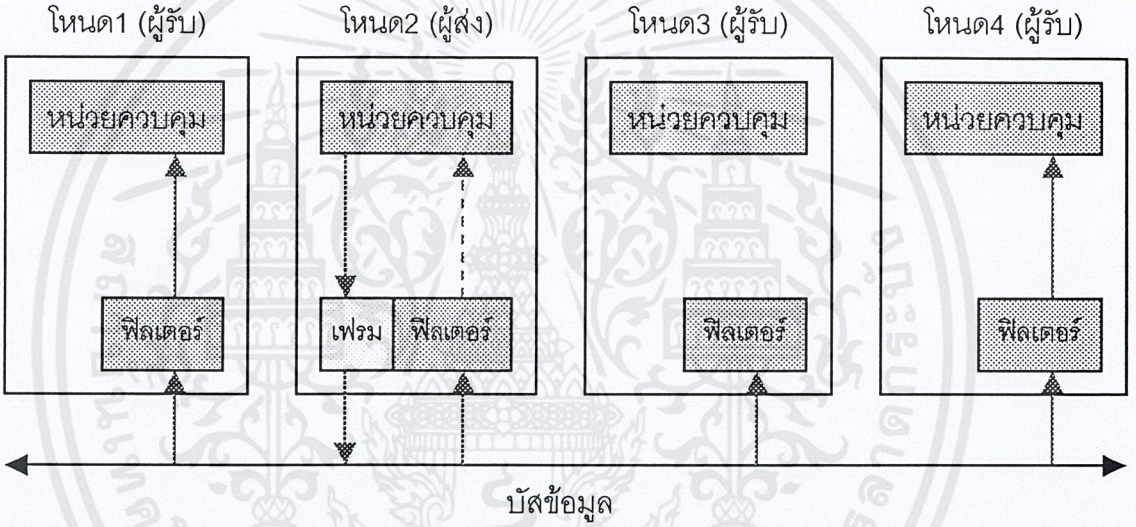
2.6 การสื่อสารแบบบรอดคาส

การส่งข้อมูลใน CAN จะใช้หลักการส่งข้อมูลแบบบรอดคาส คือ ทุกโหนดที่อยู่บนบัสสามารถรับข้อมูลที่ส่งมาจากโหนดผู้ส่งได้ หลังจากที่ได้รับข้อมูลแล้วเป็นหน้าที่ของแต่ละโหนดที่จะต้องตรวจสอบเองว่าเป็นข้อมูลที่ต้องการหรือไม่ ซึ่งการตรวจสอบตรงจุดนี้จะอาศัยการแอกเซปแตนซ์ฟิลเตอร์ (Acceptance Filter) ทำหน้าที่สกัดข้อมูลที่ไม่ต้องการทิ้งไปและยอมรับเฉพาะข้อมูลที่ต้องการเท่านั้น โดย

ดูจากค่ารหัสประจำตัว (Identifier) ของข้อมูลหรือข้อความนั้น ลักษณะการส่งข้อมูลแบบบรอดคาสจะเป็นดังรูปที่ 2.5

รูปแบบการส่งข้อมูลแบบบรอดคาสใน CAN นี้ อาจจะเปรียบได้กับการส่งข่าวสารจากสถานีวิทยุ เช่น ข้อมูลการจราจร

การสื่อสารข้อมูลใน CAN นอกจากจะสื่อสารข้อมูลแบบบรอดคาสแบบธรรมดาคือแต่ละโหนดจะรอรับข้อมูลจากโหนดที่ต้องการส่งแล้ว ยังสามารถร้องขอข้อมูล (Remote Request) จากโหนดที่มีข้อมูลดังกล่าวได้อีกด้วย โดยการส่งค่ารหัสประจำตัวของข้อมูลที่ต้องการจะทราบออกไป หากโหนดใดมีข้อมูลดังกล่าวก็จะส่งข้อมูลนั้นตอบกลับมาซึ่งนอกจากโหนดที่ร้องขอข้อมูลนี้ไปแล้วโหนดอื่นๆ ที่เหลือก็สามารถระบุข้อมูลดังกล่าวได้ด้วย



รูปที่ 2.5 ลักษณะการสื่อสารข้อมูลแบบบรอดคาสใน CAN

2.7 การเข้าใช้บัสจากหลายโหนด

จุดเด่นของโพรโตคอลการส่งข้อมูลที่ใช้ใน CAN อย่างหนึ่งก็คือ การยินยอมให้มีการใช้บัสข้อมูลจากหลายโหนดได้พร้อมกัน ในกรณีที่มีโหนดมากกว่าหนึ่งโหนดต้องการส่งข้อมูลในเวลาเดียวกัน จะมีการตัดสินชี้ขาดว่าโหนดใดจึงจะมีสิทธิ์ที่จะได้ใช้บัสข้อมูลในเวลานั้นเพียงโหนดเดียวในขณะที่โหนดอื่นจะต้องหยุดทำการส่งและรอที่จะส่งข้อมูลนั้นในภายหลังเมื่อบัสว่าง

การเข้าใช้บัสใน CAN จะใช้วิธีการที่เรียกว่า Carrier Sense Multiple Access แบบ Collision Detection (CSMA/CD) กล่าวคือ เมื่อโหนดใดต้องการใช้บัส จะต้องตรวจสอบจนพบว่าบัสว่าง ไม่ได้ถูกใช้งานอยู่เป็นระยะเวลาหนึ่งจึงเริ่มทำการส่งข้อมูลออกไป (CS) ซึ่งขณะที่บัสว่างอยู่นี้แต่ละโหนดมีสิทธิ์ที่จะใช้บัสด้วยโอกาสที่เท่ากัน (MA) แต่ถ้ามีการส่งข้อมูลออกมาพร้อมๆกัน โหนดจะทราบได้ว่ามีการชนกันของข้อมูลเกิดขึ้น (CD) และดำเนินการแก้ไขการชนกันของข้อมูลที่เกิดขึ้นต่อไป

เมื่อมีการเข้าใช้บัสพร้อมกัน วิธีการที่นำมาใช้ใน CAN เพื่อตัดสินว่าโหนดใดจะมีสิทธิเข้าใช้บัส จะอาศัยค่าลำดับความสำคัญของข้อมูลเป็นตัวตัดสิน (Arbitration on Message Priority : AMP) ซึ่งค่าลำดับความสำคัญ (Priority) ของข้อมูลจะถูกระบุอยู่ในค่ารหัสประจำตัว (Identifier) ของข้อมูลนั้น โดยการตัดสินชี้ขาดจะเกิดขึ้นโดยไม่ทำลายโอกาสในการใช้บัสของข้อมูลที่มีลำดับความสำคัญสูงสุด (Non-destructive Arbitration) และการตัดสินจะเกิดขึ้นทันทีในระดับบิตข้อมูล (Bitwise Arbitration)

กระบวนการเช่นนี้จะเกิดขึ้นได้ต้องอาศัยคุณสมบัติที่สำคัญ 2 ประการ คือ

2.7.1 สถานะของสัญญาณข้อมูลที่เป็นแบบโคมิแนนต์ (Dominant) และรีเซสสีฟ (Recessive) โดย CAN กำหนดให้สถานะโคมิแนนต์แทนลอจิก '0' และรีเซสสีฟแทนลอจิก '1' ในการใช้งานเมื่อสัญญาณทั้งสองสถานะถูกส่งมาพร้อมๆกัน บัสจะมีสถานะโคมิแนนต์(คืออ่านค่าได้เป็นลอจิก '0')

2.7.2 โหนดผู้ส่งต้องสามารถตรวจสอบสถานะของบัสอยู่ตลอดเวลา ว่าข้อมูลที่อยู่ในบัส ตรงกับข้อมูลที่ส่งออกไปหรือไม่หากไม่ตรงกันแสดงว่ามีการชนกันของข้อมูลหรือมีความผิดพลาดในการส่งข้อมูลเกิดขึ้น

ในการส่งข้อมูลแต่ละโหนดจะเริ่มต้นด้วยการส่งค่ารหัสประจำตัวของข้อมูลออกมาก่อน ในขณะเดียวกันก็จะคอยตรวจสอบดูว่าค่าที่ได้รับจากบัสมีค่าตรงกับค่าที่ส่งออกไปหรือไม่ ในกรณีที่มีการส่งข้อมูลพร้อมกันมากกว่า 1 โหนดดังตัวอย่างในรูปที่ 2.6 มีการส่งข้อมูลออกมาพร้อมกัน 2 โหนดขณะที่บิตข้อมูล (ค่ารหัสประจำตัว) ที่แต่ละโหนดส่งออกมามีค่าตรงกันอยู่ การส่งข้อมูลในแต่ละโหนดก็จะดำเนินการต่อไปตามปกติ

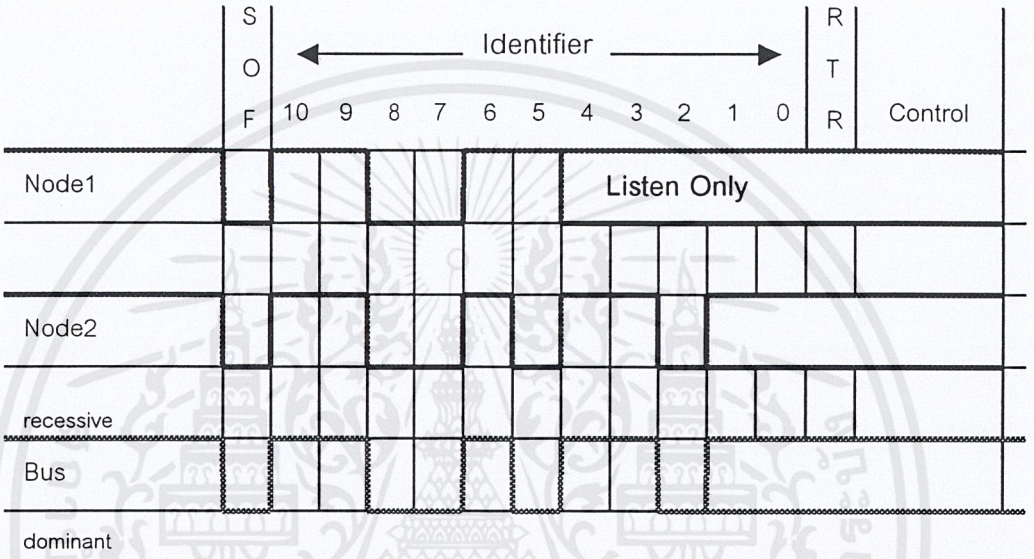
จนกระทั่งเมื่อค่าที่ส่งออกมาไม่ตรงกัน เช่น ในบิต 5 โหนดที่ 1 ส่งสัญญาณสถานะรีเซสสีฟ (ลอจิก '1') ในขณะที่โหนดที่ 2 ส่งสัญญาณสถานะโคมิแนนต์ (ลอจิก '0') แต่อ่านค่าจากบัสกลับมาได้เป็นรีเซสสีฟ ซึ่งทำให้โหนด 1 หมดสิทธิ์ในการส่งข้อมูลในรอบนี้ไปและเปลี่ยนมาเป็นโหนดผู้รับแทน สุดท้ายจึงเหลือแต่โหนด 2 เพียงโหนดเดียวที่ส่งข้อมูลได้ในรอบนี้

จากตัวอย่างนี้กล่าวได้ว่า ข้อมูลจากโหนดที่ 2 ที่มีค่าระดับความสำคัญสูงกว่าข้อมูลจากโหนดที่ 1 และ 2 ซึ่งจะมีสิทธิ์ส่งข้อมูลได้ก่อน และไม่ต้องเริ่มทำการส่งข้อมูลใหม่เมื่อมีการส่งข้อมูลชนกับโหนดอื่น ในขณะที่โหนดที่ 1 ซึ่งมีระดับความสำคัญต่ำกว่าจะเริ่มทำการส่งข้อมูลได้ใหม่หลังจากที่โหนดที่ 2 ส่งข้อมูลเสร็จเรียบร้อยแล้ว

2.8 การสื่อสารที่อิงจากข้อความ

การส่งข้อมูลใน CAN จะไม่มีการระบุที่อยู่ของโหนดผู้รับและโหนดผู้ส่งเหมือนดังที่ใช้กันอยู่ทั่วไป แต่จะใช้การระบุด้วยค่าประจำตัว (Identifier) ของข้อมูลหรือข้อความแทน เพื่อเป็นการบ่งบอกชนิดและระดับความสำคัญของข้อมูลนั้น โดยอาศัยวิธีการส่งข้อมูลแบบบรอดคาสดังเช่นที่กล่าวมาในขั้นต้น คือเมื่อมีโหนดใดโหนดหนึ่งทำการส่งข้อมูลออกมาที่บัสแต่ละโหนดจะได้รับการแจ้งว่ามีการส่งข้อมูลเกิดขึ้นและจะตัดสินใจกันเองว่าจะรับข้อมูลที่ส่งมานั้นหรือไม่

ดังตัวอย่างเช่น โหนด A ทำการส่งข้อมูล (สมมติว่าค่าที่เป็นค่าที่วัดได้จากเซนเซอร์) ด้วยค่ารหัสประจำตัวของข้อมูล (สมมติว่าเท่ากับ 123) ออกไปที่บัส หากโหนดใดต้องการใช้ข้อมูลดังกล่าวก็จะทำการรับข้อมูลนี้ไว้ จะเห็นว่าการส่งข้อมูลด้วยวิธีนี้ไม่จำเป็นต้องมีการระบุหมายเลขที่อยู่หรือรหัสประจำตัวของโหนดทั้งโหนดผู้รับและผู้ส่ง ซึ่งก็มีข้อดีคือ สามารถทำการเพิ่มโหนดเข้ามาในระบบได้โดยไม่จำเป็นต้องทำการโปรแกรมใหม่เพื่อให้รับรู้ว่ามีการเพิ่มโหนดเข้ามา ในขณะเดียวกันโหนดที่เพิ่มเข้ามาใหม่นี้ก็สามารถรับข้อมูลที่มีการส่งกันในบัสได้ทันทีโดยพิจารณาจากค่ารหัสประจำตัวของข้อมูลนั่นเอง



รูปที่ 2.6 ตัวอย่างของสัญญาณในบัส CAN กรณีที่มีโหนดส่งข้อมูลออกมาที่บัสพร้อมกัน

นอกจากการรอรับข้อมูลที่โหนดอื่นส่งมาให้แล้ว (ซึ่งบางทีอาจต้องรอนานกว่าที่จะได้รับข้อมูลที่ต้องการ) แต่ละโหนดก็ยังสามารถร้องขอข้อมูลจากโหนดอื่นได้ด้วย เพื่อให้โหนดที่มีข้อมูลดังกล่าวส่งข้อมูลมาให้ทันที โดยเรียกการร้องขอข้อมูลนี้ว่า Remote Transmission Request (RTR) มีข้อดีคือ ข้อมูลบางอย่างที่ไม่ได้มีการใช้งานเป็นประจำ ก็ไม่จำเป็นต้องส่งออกมาอยู่ตลอดเวลา รอให้มีการร้องขอข้อมูลนั้นมาก่อนจึงค่อยส่งข้อมูลนั้นออกไป ซึ่งจะช่วยลดปริมาณการใช้บัสลงได้

2.9 การป้องกันความผิดพลาด

เนื่องจากเริ่มแรกนั้น CAN ถูกออกแบบมาเพื่อใช้ในระบบอิเล็กทรอนิกส์ในรถยนต์ ซึ่งปัญหาที่พบส่วนใหญ่ก็คือเรื่องสัญญาณรบกวน ดังนั้นการออกแบบโพรโตคอลเพื่อให้สามารถใช้ส่งข้อมูลได้อย่างมีประสิทธิภาพ จะต้องส่งข้อมูลได้อย่างไม่มีข้อผิดพลาดในขณะที่ความเร็วในการส่งข้อมูลต้องเป็นที่ยอมรับหรือรับรองการใช้งานได้อย่างกว้างขวาง

ทั้งนี้นอกจาก CAN จะสามารถส่งข้อมูลได้ที่ความเร็วสูงถึง 1 เมกะบิตต่อวินาที (ในเวอร์ชัน CAN2.0) แล้ว จุดเด่นที่สำคัญอีกอย่างหนึ่งก็คือ การตรวจสอบข้อผิดพลาดของข้อมูลเพื่อให้แน่ใจว่าข้อมูล

ที่ได้มีความถูกต้องสมบูรณ์ไม่มีความผิดพลาดเกิดขึ้น โดยการสื่อสารข้อมูลใน CAN มีวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้นหลายแบบ ซึ่งจะขอกล่าวถึงรายละเอียดในภายหลัง

2.10 โพรโตคอลการส่งข้อมูล

โพรโตคอลการส่งข้อมูลใน CAN จะแบ่งข้อมูลที่ส่งออกเป็นแพ็กเก็ต (Packets) หรือเฟรม (Frames) ซึ่งมีใช้อยู่ 4 ชนิดได้แก่ เฟรมข้อมูล (Data Frame) ใช้ในการส่งข้อมูลตามปกติ, เฟรมร้องขอข้อมูลจากโหนดอื่น (Remote Frame) ใช้ในการร้องข้อมูลจากโหนดอื่น, เฟรมแสดงความผิดพลาด (Error Frame) ใช้แสดงว่าความผิดพลาดเกิดขึ้น และเฟรมโอเวอร์โหลด (Overload Frame) ใช้เพื่อบอกว่าต้องการเวลาในการประมวลผลข้อมูลที่ได้รับเพิ่มขึ้น

2.11 เฟรมข้อมูล

ในเฟรมข้อมูล 1 เฟรมจะประกอบไปด้วยส่วนย่อยๆที่เรียกว่าฟิลด์ (Fields) ดังในรูปที่ 2.7 และ 2.8 ซึ่งแสดงส่วนประกอบของเฟรมข้อมูลแบบมาตรฐาน (Standard Data Frame) และเฟรมข้อมูลแบบขยาย (Extended Data Frame) ตามลำดับเฟรมข้อมูลทั้งสองแบบจะมีข้อแตกต่างกันตรงที่ขนาดของรหัสประจำตัวของข้อมูล (Identifier หรือ ID)

2.12 เฟรมข้อมูลแบบมาตรฐาน (Can 2.0A)

ส่วนประกอบต่างๆในเฟรมข้อมูลแบบมาตรฐานมีดังนี้

2.12.1 บิตเริ่มต้นเฟรม (Start of Frame : SOF) มีสถานะเป็นโดมิแนนต์ หรือลอจิก ‘0’ ซึ่งเป็นจุดที่ทุกโหนดในบัสใช้ในการเริ่มต้นเข้าสู่จังหวะสำหรับการรับส่งข้อมูลระหว่างกัน

2.12.2 ฟิลด์แสดงรหัสข้อมูล (Arbitration Field) มีขนาด 12 บิต ประกอบด้วยหมายเลข ID (Identifier) ขนาด 11 บิต (และมีขนาด 29 บิตสำหรับเฟรมข้อมูลแบบขยาย) และบิต RTR (Remote Transmission Request) 1 บิต การส่งข้อมูลโดยใช้เฟรมข้อมูลแบบมาตรฐานซึ่งมีหมายเลข ID ขนาด 11 บิต จะมีหมายเลข ID ที่สามารถใช้งานได้จำนวน $2^{11} = 2,048$ หมายเลขที่แตกต่างกัน โดยมีการสำรองไว้ประมาณ 16 หมายเลขสำหรับใช้งานพิเศษเฉพาะอย่าง จึงมีเหลือใช้งานได้ 2,032 หมายเลขซึ่งหมายถึงจำนวนข้อมูลที่สามารถรองรับนั่นเอง บิต RTR ซึ่งอยู่ถัดจากหมายเลข ID เป็นบิตที่ใช้แสดงชนิดของเฟรมข้อมูล โดยถ้าบิต RTR = ‘0’ หรือโดมิแนนต์แสดงว่าเป็นเฟรมข้อมูล แต่ถ้าบิต RTR = ‘1’ หรือรีเซสซีฟแสดงว่าเป็นเฟรมร้องขอข้อมูล

2.12.3 ฟิลด์ควบคุม (Control Field) มีขนาด 6 บิต IDE (Identifier Extension) 1 บิต, บิต RB0 (Reserve bit 0) 1 บิต และ DLC (Data Length Code) จำนวน 4 บิต บิต IDE ใช้บ่งบอกว่าเฟรมข้อมูลนี้จะเป็นแบบใด โดยถ้าบิต IDE = ‘0’ หรือโดมิแนนต์จะเป็นเฟรมข้อมูลแบบมาตรฐาน (ID = 11 บิต) ถ้าบิต IDE = ‘1’ หรือรีเซสสีฟจะเป็นเฟรมข้อมูลแบบขยาย (ID =

29 บิต) สำหรับในที่นี้เป็นเฟรมข้อมูลมาตรฐาน ดั้งนั้นบิต IDE นี้จะมีค่าเป็น '0' หรือโดมิแนนต์ บิต RB0 เป็นบิตที่สำรองไว้สำหรับการใช้งานในอนาคต ในการใช้งานจะให้บิตนี้เป็น '0' หรือโดมิแนนต์ ส่วนสุดท้ายในฟิลด์ควบคุมคือ DLC มีขนาด 4 บิต ใช้บอกขนาดของข้อมูลที่ส่งมา โดยข้อกำหนดของ CAN กำหนดให้ข้อมูลที่ส่งมาในแต่ละเฟรมมีขนาดได้ตั้งแต่ 0-8 ไบต์

Number of data byte	Data length code			
	DLC3	DLC2	DLC1	DLC0
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

หมายเหตุ r = recessive , d = dominant

ตารางที่ 2.1 ความสัมพันธ์ของ Data length Code (DLC) กับจำนวนข้อมูลในเฟรมนั้น

2.12.4 ฟิลด์ข้อมูล (Data Field) เป็นส่วนของข้อมูลที่ต้องการส่ง มีขนาดไม่เกิน 8 ไบต์ ฟิลด์ตรวจสอบ (CRC Field) มีขนาด 16 บิต เป็นส่วนที่ใช้ในการตรวจสอบความถูกต้องของข้อมูลที่ส่งในเฟรมนั้น ประกอบไปด้วยค่า CRC (Cyclic Redundancy Check) ขนาด 15 บิต และบิต CRC Delimiter 1 บิต การส่งข้อมูลในแต่ละเฟรมจะมีการคำนวณค่า CRC เริ่มตั้งแต่บิตเริ่มต้นเฟรมไปจนฟิลด์ข้อมูล โดยสามารถตรวจสอบพบความผิดพลาดของข้อมูลที่เกิดขึ้นในช่วงดังกล่าวได้หากมีความผิดพลาดไม่เกิน 5 บิต เมื่อส่งค่า CRC ครบทั้ง 15 บิตแล้วจะปิดท้ายฟิลด์ตรวจสอบนี้ด้วยบิต CRC Delimiter โดยจะส่งค่า '1' หรือรีเซสสีฟ

2.12.5 ฟิลด์ตอบสนอง (Acknowledge Field) มีขนาด 2 บิต เป็นส่วนที่ผู้รับใช้ตอบกลับไปยังผู้ส่งว่าข้อมูลที่ได้รับมาถูกต้องหรือไม่ ประกอบไปด้วย ACK Slot 1 และ ACK Delimiter 1 บิต โหนดผู้ส่งจะส่งบิต ACK Slot นี้เป็น '1' หรือรีเซสสีฟออกไป ในขณะที่เมื่อโหนดผู้รับตรวจสอบข้อมูลที่ได้รับว่าถูกต้อง ก็จะส่งบิต ACK Slot เป็น '0' หรือโดมีแนนต์ตอบ

กลับมา ซึ่งจะทำให้สถานะของบัสเป็นโดมิแนนต์ ดังนั้นเมื่อโหนดผู้ส่งอ่านค่าบิต ACK Slot กลับมาได้เป็น '0' หรือโดมิแนนต์แสดงว่ามีอย่างน้อย 1 โหนดที่ได้รับข้อมูลได้อย่างถูกต้อง ปิดท้ายฟิลด์ตอบสนองด้วยบิต ACK Delimiter ซึ่งมีค่าเป็น '1' หรือรีเซสสีฟเสมอ

2.12.6 ฟิลด์สิ้นสุดเฟรม (End of Frame : EOF Field) ประกอบไปด้วยบิต '1' หรือรีเซสสีฟจำนวน 7 บิตติดต่อกันหรือมีค่าเท่ากับ "1111111" เป็นส่วนที่ใช้แสดงว่าได้สิ้นสุดเฟรมนี้แล้วและเพื่อให้เวลาสำหรับโหนดผู้รับสำหรับการประมวณผลหรือจัดการกับข้อมูลที่ได้รับมาให้เป็นที่เรียบร้อยเสียก่อน ดังนั้นหลังจากที่เฟรมสิ้นสุดแล้ว จึงกำหนดว่าจะต้องส่งค่า '1' หรือรีเซสสีฟสิ้นสุดอย่างน้อย 3 บิต จึงจะสามารถส่งเฟรมข้อมูลใหม่ได้ต่อไป

2.13 เฟรมข้อมูลแบบขยาย (CAN 2.0B)

ส่วนประกอบต่างๆในเฟรมข้อมูลแบบขยายจะต่างจากเฟรมข้อมูลแบบมาตรฐานตรงฟิลด์แสดงรหัสข้อมูลและฟิลด์ควบคุมเท่านั้น ส่วนฟิลด์ที่เหลือจะเหมือนกัน สำหรับรายละเอียดในส่วนที่แตกต่างกันมีดังนี้

2.13.1 ฟิลด์แสดงรหัสข้อมูล จะมีขนาด 32 บิต โดยแบ่งเป็นหมายเลข ID (Identifier) ขนาด 29 บิต (ในขณะที่เฟรมข้อมูลแบบมาตรฐานจะมีหมายเลข ID ขนาด 11 บิต), บิต SRR, บิต IDE และบิต RTR หมายเลข ID 29 บิตนี้จะแบ่งออกเป็น 2 ส่วน คือส่วนแรกมีขนาด 11 บิตอยู่ถัดจากบิตเริ่มต้นเฟรมเช่นเดียวกับหมายเลข ID ในเฟรมข้อมูลแบบมาตรฐาน และส่วนที่สองมีขนาด 18 บิตอยู่ถัดจากบิต IDE เหตุที่ต้องแบ่งออกเป็น 2 ส่วนโดยส่วนแรกมีขนาด 11 บิตเช่นนี้ ก็เพื่อให้สอดคล้องกับหมายเลข ID ที่ใช้ในเฟรมข้อมูลแบบมาตรฐานนั่นเอง สาเหตุที่ต้องมีการขยายจำนวนหมายเลข ID เพิ่มขึ้นก็เนื่องมาจากว่าจำนวนหมายเลข ID ในเฟรมข้อมูลแบบมาตรฐานที่มีอยู่ 2,048 หมายเลขนั้น แม้จะดูเหมือนมากแต่ในการใช้งานบางครั้งอาจไม่เพียงพอ จึงได้มีการกำหนดเฟรมข้อมูลแบบขยาย (CAN 2.0B) ขึ้นมาเพิ่มเติมโดยกำหนดให้หมายเลข ID มีขนาด 29 บิต ซึ่งจะรองรับการใช้งานได้มากถึง 2^{29} = 536,870,912 หมายเลขด้วยกัน บิต SRR (Substitute Remote Request) จะอยู่ตำแหน่งเดียวกับบิต RTR ที่อยู่ในเฟรมข้อมูลแบบมาตรฐานเพื่อใช้แทนบิต RTR โดยจะมีค่าเป็น '1' หรือรีเซสสีฟ ถัดจากบิต SRR ก็จะเป็นบิต IDE ซึ่งอยู่ตำแหน่งเดียวกับบิต IDE ในเฟรมข้อมูลมาตรฐานและมีหน้าที่เดียวกัน โดยในที่นี้จะมีค่าเป็น '1' หรือรีเซสสีฟเนื่องจากเป็นเฟรมข้อมูลแบบขยาย ส่วนบิต RTR จะอยู่ต่อจากหมายเลข ID ส่วนที่สอง (18 บิต) และมีค่าเป็น '0' หรือโดมิแนนต์เนื่องจากเป็นเฟรมข้อมูลนั่นเอง

2.13.2 ฟิลด์ควบคุม มีขนาด 6 บิตเช่นกัน ส่วนที่แตกต่างไปจากเฟรมข้อมูลแบบมาตรฐานก็คือบิต RB1 (Reserved Bit 1) ซึ่งอยู่ตำแหน่งเดียวกันกับบิต IDE ในควบคุมของเฟรมข้อมูลแบบมาตรฐาน เนื่องจากบิต IDE ในฟรมข้อมูลแบบขยายได้ถูกย้ายไปอยู่ในส่วนของฟิลด์แสดงรหัสข้อมูลแล้ว ดังนั้นที่ตำแหน่งนี้ในฟิลด์ของเฟรมควบคุมของเฟรมข้อมูลแบบ

ขยายจึงได้กำหนดให้เป็นบิต RB1 ซึ่งสำรองไว้ใช้งานในอนาคต โดยกำหนดให้มีค่าเป็น '0' หรือโดมิแนนต์เอาไว้

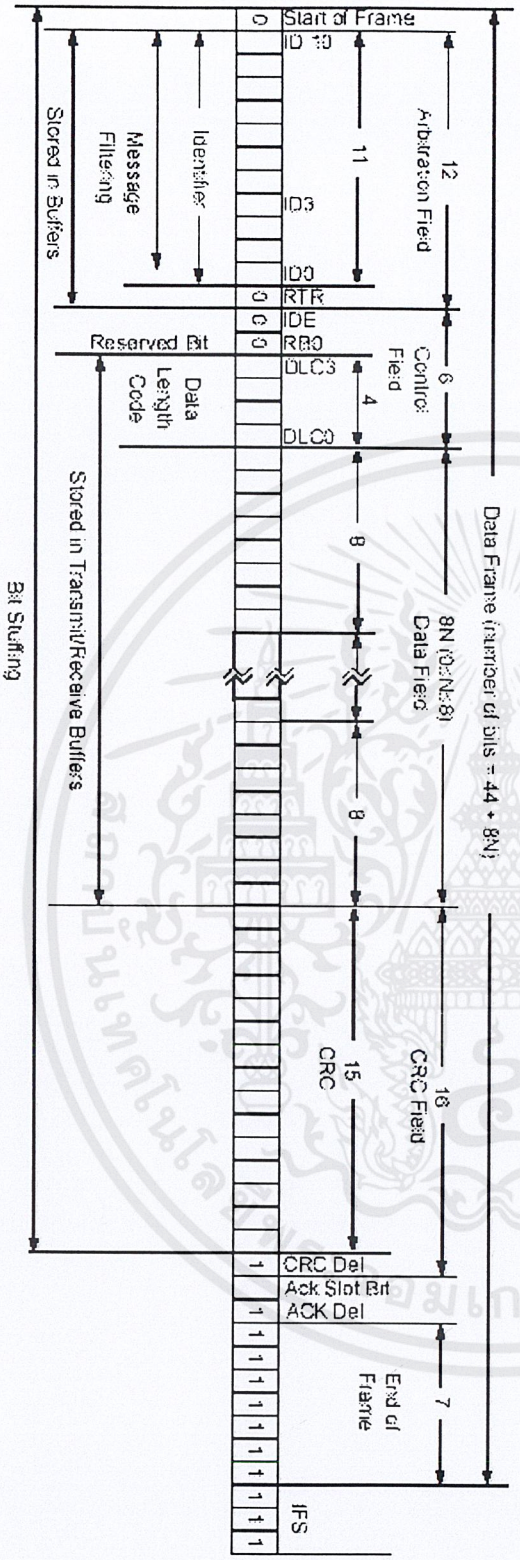
2.14 เฟรมร้องขอข้อมูล

โหนดผู้รับสามารถร้องขอข้อมูลจากโหนดที่มีข้อมูลนั้นอยู่ เพื่อให้โหนดนั้นส่งข้อมูลที่ต้องการมาให้ได้ โดยการส่งเฟรมร้องขอข้อมูลที่ต้องการนั้นออกไป หากโหนดใดมีข้อมูลที่มีหมายเลข ID ตรงกับที่ถูกร้องขอมา ก็จะทำการส่งข้อมูลนั้นตอบกลับไป ลักษณะของเฟรมร้องขอข้อมูลจะคล้ายกับเฟรมข้อมูล แต่จะมีส่วนที่แตกต่างกับเฟรมข้อมูลอยู่สองประการด้วยกันคือ

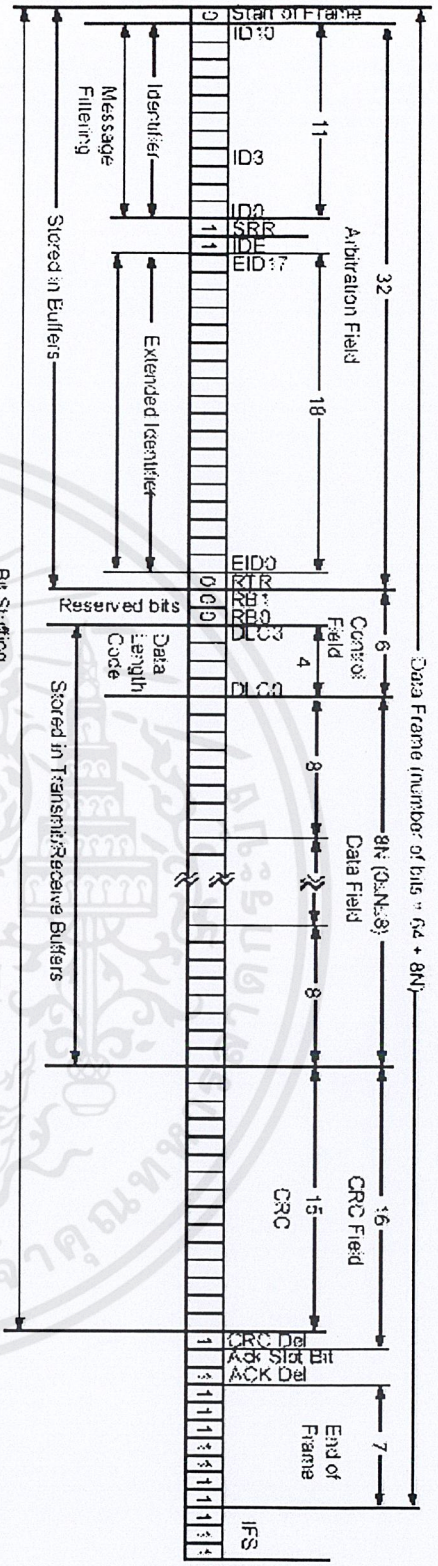
2.14.1 บิต RTR ในเฟรมร้องขอข้อมูลจะมีค่าเป็น '1' หรือรีเซสสีฟ ในขณะที่ในเฟรมข้อมูลจะเป็น '0' หรือโดมิแนนต์

2.14.2 ในเฟรมร้องขอข้อมูลจะไม่มีฟิลด์ข้อมูล

ข้อสังเกตอย่างหนึ่งก็คือ ในกรณีที่เฟรมร้องขอข้อมูลและเฟรมข้อมูลที่มีหมายเลข ID เดียวกันถูกส่งออกมาพร้อมกัน เฟรมข้อมูลจะมีสิทธิได้ใช้บัสเนื่องจากบิต RTR จะมีค่าเป็น '0' หรือโดมิแนนต์ในขณะที่โหนดที่ส่งเฟรมร้องขอออกมาก็จะได้รับข้อมูลทันที(เนื่องจากมีหมายเลข ID ตรงกัน)



รูปที่ 2.7 เฟรมข้อมูลแบบมาตรฐาน



รูปที่ 2.8 เฟรมข้อมูลแบบขยาย

2.15 การเติมสตัฟฟ์บิต

เนื่องจากการส่งสัญญาณข้อมูลใน CAN จะใช้วิธีการส่งสัญญาณแบบ NRZ (Non-Return-to-Zero) ที่ระดับของสัญญาณในแต่ละบิตมีค่าคงที่ในช่วงเวลาของบิตนั้น และเป็นการเป็นการส่งข้อมูลแบบอะซิงโครนัส (Asynchronous) ที่ต้องอาศัยการเข้าจังหวะสัญญาณข้อมูลที่ส่งมา การส่งข้อมูลด้วยวิธีนี้จะเกิดปัญญหาขึ้นในกรณีที่สัญญาณข้อมูลมีค่าเดียวกันติดๆกันหลายบิต ซึ่งจะทำให้การเข้าจังหวะนั้นมีความผิดพลาดเกิดขึ้นได้มาก

ดังนั้นเพื่อป้องกันไม่ให้เหตุการณ์เช่นนี้เกิดขึ้นใน CAN จึงได้มีการกำหนดให้มีการเติมสตัฟฟ์บิต (Stuff bit) คือถ้ามีการส่งบิตที่มีค่าเดียวกันเกิน 5 บิตจะต้องคั่นด้วยบิตที่มีค่าตรงกันข้ามกับบิต 5 นั้น 1 บิต และเมื่อโหนดผู้รับได้รับการข้อมูลที่มีค่าเดียวกันเกิน 5 บิต ก็จะไม่นับบิตที่ตามมา 1 บิตว่าเป็นบิตข้อมูล แต่จะเริ่มนับบิตข้อมูลในบิตถัดไป

การเติมสตัฟฟ์บิตจะเกิดขึ้นในเฟรมข้อมูลและเฟรมร้องขอข้อมูลระหว่างบิตเริ่มต้นเฟรมกับบิต CRC Delimiter เท่านั้น ซึ่งนอกจากการเติมสตัฟฟ์บิตจะช่วยในการเข้าจังหวะสัญญาณของข้อมูลแล้ว ยังใช้ตรวจสอบความผิดพลาดของข้อมูลได้อีกด้วย

2.16 การตรวจสอบและแก้ไขความผิดพลาด

จุดเด่นที่สำคัญอย่างหนึ่งของ CAN ก็คือมีการตรวจสอบข้อผิดพลาดของข้อมูลที่เกิดขึ้น ความผิดพลาดที่ตรวจสอบได้มีอยู่ 5 อย่างด้วยกัน คือ

2.16.1 CRC Error ในขณะที่โหนดผู้ส่งทำการส่งข้อมูลจะคำนวณค่า CRC ของข้อมูลที่ส่งด้วย และส่งไปกับเฟรมข้อมูลนั้นในฟิลด์ตรวจสอบ ที่โหนดผู้รับจะทำการคำนวณค่า CRC ของข้อมูลที่ได้รับด้วยวิธีเดียวกับโหนดผู้ส่งแล้วนำไปเปรียบเทียบกับค่า CRC ที่ส่งมา หากไม่ตรงกันแสดงว่าข้อมูลที่ได้รับมามีความผิดพลาดเกิดขึ้น ที่โหนดนั้นจะทำการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อบอกว่าข้อมูลที่ได้รับไม่ถูกต้อง เมื่อโหนดที่ส่งนั้นได้รับว่ามีการแจ้งว่ามีความผิดพลาดเกิดขึ้น ก็จะทำการส่งข้อมูลนั้นออกไปใหม่อีกครั้งหนึ่ง

2.16.2 Acknowledge Error โหนดผู้ส่งจะทำการตรวจสอบดูที่บิต ACK Slot ในฟิลด์ตอบสนองมีค่าเป็น '0' หรือโดมิแนนต์หรือไม่ (ในขณะที่โหนดผู้ส่งเองจะส่งค่า '1' หรือรีเซสสีฟออกไป) หากมีค่าเป็นโดมิแนนต์แสดงว่ามีโหนดอย่างน้อยหนึ่งโหนดที่ได้รับข้อมูลอย่างถูกต้อง แต่ในทางตรงกันข้ามหาก ACK Slot มีค่าเป็นรีเซสสีฟแสดงว่าไม่มีโหนดใดที่ได้รับข้อมูลที่ถูกต้องเลย ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

2.16.3 Form Error ถ้าพบว่าสถานะโดมิแนนต์เกิดขึ้นในส่วนต่างๆ ต่อไปนี้ได้แก่ บิต CRC Delimiter, บิต ACK Delimiter และเฟรมสิ้นสุดเฟรม แสดงว่ามีความผิดพลาดเกิดขึ้นใน

การส่งข้อมูลดังกล่าว (เนื่องจากส่วนต่างๆมีสถานะรีเซสสีฟเสมอ) จะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้งหนึ่ง

2.16.4 Bit Error จะเกิดขึ้นเมื่อโหนดผู้ส่งอ่านค่าจากบัสกลับมาได้ค่าไม่ตรงกับที่ส่งออกไป คือส่งค่าโดมิแนนต์แต่อ่านได้เป็นรีเซสสีฟ หรือส่งค่ารีเซสสีฟแต่อ่านได้เป็นโดมิแนนต์ ในฟิลด์แสดงรหัสข้อมูลและบิต ACK Slot นั้นไม่ถือว่าเป็นความผิดพลาดเนื่องจากสามารถเกิดขึ้นได้ เมื่อพบความผิดพลาดจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

2.16.5 Stuff Error จากที่กล่าวไปแล้วว่าการส่งข้อมูลใน CAN จะมีการเติมสตัฟฟ์บิตเมื่อข้อมูลที่ส่งมามีค่าเดียวกันติดต่อกันเกิน 5 บิต ดังนั้นเมื่อข้อมูลที่ได้รับ (ที่อยู่ระหว่างบิตเริ่มต้นข้อมูลและบิต CRC Delimiter) มีค่าเดียวติดกันเกิน 5 บิตแสดงว่ามีความผิดพลาดเกิดขึ้น ซึ่งจะมีการส่งเฟรมแสดงความผิดพลาดแจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

นอกจากความสามารถในการตรวจสอบความผิดพลาดที่เกิดขึ้นแล้ว CAN ยังได้มีการกำหนดสถานะความผิดพลาดของแต่ละโหนดในบัสขึ้นเพื่อควบคุมให้การใช้งานบัสมีประสิทธิภาพมากขึ้น กล่าวคือในกรณีที่มีโหนดใดมีความผิดพลาดเกิดขึ้นมากเกินกว่าเกณฑ์ที่กำหนดก็จะไม่อนุญาตให้ใช้บัสได้ เนื่องจากหากปล่อยให้โหนดดังกล่าวใช้บัสได้ตามปกติ จะทำให้การใช้งานบัสโดยรวมมีประสิทธิภาพต่ำลง เพราะจะเกิดความผิดพลาดเกิดขึ้นบ่อยครั้งซึ่งทุกครั้งจะต้องมีการส่งข้อมูลกันใหม่ ทำให้การใช้งานบัสมีมากและไปแย่งเวลากับโหนดอื่นโดยไม่จำเป็น

2.17 โครงสร้างโหนด CAN

โมเดลการทำงานของแต่ละโหนดในระบบบัส CAN โดยทั่วไปจะมีลักษณะโครงสร้างซึ่งมีส่วนประกอบหลักๆ 3 ส่วนด้วยกันคือ

2.17.1 หน่วยควบคุมประมวลผล(Microcontroller หรือ Microprocessor)

2.17.2 ตัวควบคุม CAN (CAN Controller)

2.17.3 ตัวรับส่งสัญญาณ CAN (CAN Transceiver)

ในส่วนของการนำ CAN ไปใช้งานนั้น ปัจจุบันหลายบริษัทได้มีการออกแบบและพัฒนาตัวควบคุม CAN ให้อยู่ในรูปของวงจรรวมไอซี (Integrated Circuit) โดยการทำงานของตัวควบคุม CAN ที่ได้ออกแบบขึ้นมาทั้งหมดนั้นจะอ้างอิงตามมาตรฐาน ISO11898 ซึ่งเป็นส่วนของการทำงานในชั้นเชื่อมโยงข้อมูล (Data link Layer) ที่ทำหน้าที่จัดการในเรื่องการสื่อสารข้อมูลขั้นพื้นฐาน เช่น การส่งข้อมูลและการร้องขอข้อมูล ให้เป็นไปอย่างถูกต้อง รวมไปถึงการตรวจสอบความผิดพลาดของข้อมูลด้วย ซึ่งจะอ้างอิงตามโมเดลการทำงานของตัวควบคุม CAN จากบริษัท Bosch ผู้ให้กำเนิด CAN ขึ้นมานั่นเอง

ส่วนประกอบอีกส่วนหนึ่งที่สำคัญที่ทำให้ CAN สามารถสื่อสารเป็นระบบบัสข้อมูลได้ก็คือ ตัวรับส่งสัญญาณ CAN ซึ่งจะทำหน้าที่แปลงข้อมูล (Data) ที่ต้องการส่งให้เป็นสัญญาณ (Signal) ที่สามารถส่งผ่านบัสได้ ตามข้อกำหนดที่มีขึ้นสำหรับการทำงานในชั้นกายภาพ (Physical Layer)

ตัวควบคุม CAN ที่มีลักษณะการทำงานที่รวมฟังก์ชันการควบคุมโพรโตคอลไว้ในไอซีตัวเดียวและแยกออกมาจากส่วนประมวลผลนั้น เราจะเรียกว่าเป็นตัวควบคุม CAN แบบ Stand-alone และในปัจจุบันได้มีบริษัทผู้ผลิตไอซีไมโครคอนโทรลเลอร์หลายค่ายที่ออกแบบให้มีตัวควบคุม CAN รวมอยู่ในตัวไอซีด้วยเลย ซึ่งจะเรียกตัวควบคุม CAN ที่มีลักษณะเช่นนี้ว่าเป็นตัวควบคุมแบบ Integrated

ตัวควบคุมCAN ทั้งสองแบบนี้มีข้อดีและจุดเด่นในการใช้งานที่แตกต่างกันออกไป โดยแบบ Stand-alone นั้นมีข้อดีคือ สามารถนำไปใช้งานร่วมกับหน่วยประมวลผลหรือ CPU แบบใดก็ได้ตามความต้องการของผู้ใช้งาน ทำให้มีทางเลือกในส่วนของการพัฒนาในด้านซอฟต์แวร์ที่หลากหลายกว่า ส่วนตัวควบคุม CAN แบบ Integrated นั้นมีข้อดีในเรื่องค่าใช้จ่ายในส่วนของการสร้างฮาร์ดแวร์ที่ต่ำกว่า ทั้งในด้านราคาของไอซีเองและค่าใช้จ่ายในการผลิตแผ่นวงจรพิมพ์ซึ่งมีขนาดและความซับซ้อนน้อยกว่า

นอกจากนี้การทำงานแบบ Integrated ยังมีข้อดีในเรื่องการกินเวลาทำงานและการใช้ทรัพยากรของ CPU ที่น้อยกว่าด้วย เนื่องจากการควบคุมและการติดต่อผ่านบัสข้อมูลและแอดเดรสภายใน ในขณะที่ตัวควบคุมแบบ Stand-alone จะต้องอาศัยการถ่ายข้อมูลผ่านบัสข้อมูลและแอดเดรสภายนอก ซึ่งส่งผลให้การควบคุมแบบ Stand-alone สิ้นเปลืองทรัพยากรของ CPU มากกว่าและทำงานได้ช้ากว่าด้วย

ส่วนโมเดลการทำงานของตัวควบคุม CAN ในอีกรูปแบบหนึ่งนั้น จะเป็นการรวมส่วนประกอบทั้งหมดเข้ามาไว้ในไอซีเพียงตัวเดียว คือจะรวมตัวรับส่งสัญญาณ CAN เข้ามาด้วย โดยเรียกตัวควบคุมในลักษณะนี้ว่าเป็นตัวควบคุมแบบ Single-chip แนวโน้มของตัวควบคุม CAN ในลักษณะนี้เริ่มมีให้เห็นกันแล้วในปัจจุบันโดยเป็นการออกแบบตัวควบคุม CAN เพื่อใช้งานในระบบควบคุมสำหรับรถยนต์

2.18 โครงสร้างตัวควบคุม CAN

ตัวควบคุม CAN ถือเป็นส่วนประกอบที่สำคัญของโหนดในระบบ CAN ที่ทำหน้าที่ควบคุมกลไกในการรับส่งข้อมูลทั้งหมดให้เป็นไปตามมาตรฐานที่ได้กำหนดไว้ โดยการทำงานภายในตัวควบคุม CAN นั้นจะมีลักษณะของโครงสร้างหลักที่เหมือนกัน คือประกอบด้วยตัวควบคุม CAN (CAN Protocol Controller), ฮาร์ดแวร์แอกเซปแตนซ์ฟิลเตอร์ (Hardware acceptance Filter), บัฟเฟอร์ข้อมูล (Message Buffer), และส่วนติดต่อกับ CPU (CPU Interface) ซึ่งแต่ละส่วนนี้จะมีหน้าที่การทำงานดังนี้

2.18.1 ตัวควบคุมโพรโตคอล CAN ทำหน้าที่ควบคุมกลไกรับส่งข้อมูลทั้งหมดที่เกิดขึ้นในบัส CAN เช่น การสร้างเฟรมข้อมูล, การตรวจสอบความผิดพลาด, การเข้ารหัสบิตข้อมูล และการเข้าจังหวะสัญญาณ ให้เป็นไปตามโพรโตคอลที่กำหนดไว้ เพื่อให้ตัวควบคุม CAN ที่ไม่ว่าจะผลิตขึ้นมาจากบริษัทใดก็ตามสามารถนำมาใช้งานร่วมกันได้โดยไม่มีปัญหา

2.18.2 ฮาร์ดแวร์แอกเซปแตนซ์ฟิลเตอร์ เนื่องจากข้อมูลที่ส่งใน CAN มีรหัสประจำตัวหรือหมายเลข ID อยู่เป็นจำนวนมาก (โดย CAN 2.0A มีหมายเลข ID จำนวน $2^{11} = 2,048$

หมายเลข CAN 2.0B มีหมายเลข ID จำนวน 2^{29} = 536,870,912 หมายเลข) การที่จะยกให้เป็นภาระของ CPU ในการแยกแยะหมายเลข ID ของข้อมูลทั้งหมดนั้นดูจะเป็นเรื่องที่ไม่เหมาะสมเท่าไรนัก ตรงส่วนนี้จะทำหน้าที่กลั่นกรองเอาเฉพาะข้อมูลที่ต้องการ (โดยพิจารณาจากหมายเลข ID) แล้วส่งไปให้ CPU พิจารณาอีกครั้งหนึ่ง

2.18.3 บัฟเฟอร์ข้อมูล เพื่อให้การรับส่งข้อมูลทำได้โดยสะดวก โดยไม่ต้องคำนึงถึงว่า CPU พร้อมที่จะทำงานหรือไม่ หรือมีโหนดอื่นๆกำลังใช้งานบัสอยู่และช่วยให้ข้อมูลไม่มีการสูญหายเมื่อ CPU ไม่พร้อมที่จะอ่านข้อมูลที่ได้รับในขณะนั้น จึงจำเป็นต้องมีบัฟเฟอร์ข้อมูลเพื่อทำหน้าที่เก็บข้อมูลไว้ชั่วคราวทั้งข้อมูลที่ต้องการจะส่งและข้อมูลที่ได้รับมา โดยบัฟเฟอร์ของข้อมูลด้านส่งจะเก็บข้อมูลที่ต้องการส่งไว้ชั่วคราวจนกว่าบัสจะว่างพร้อมที่จะให้ส่งข้อมูลไปได้ โดยที่ CPU ไม่จำเป็นต้องมาเสียเวลาตรวจสอบบัสอยู่ตลอดเวลา ในขณะที่บัฟเฟอร์ข้อมูลด้านรับจะเก็บข้อมูลที่ได้รับมาไว้ชั่วคราวจนกระทั่ง CPU พร้อมที่จะอ่านข้อมูลไป หรือไม่ต้องการข้อมูลนั้นแล้ว

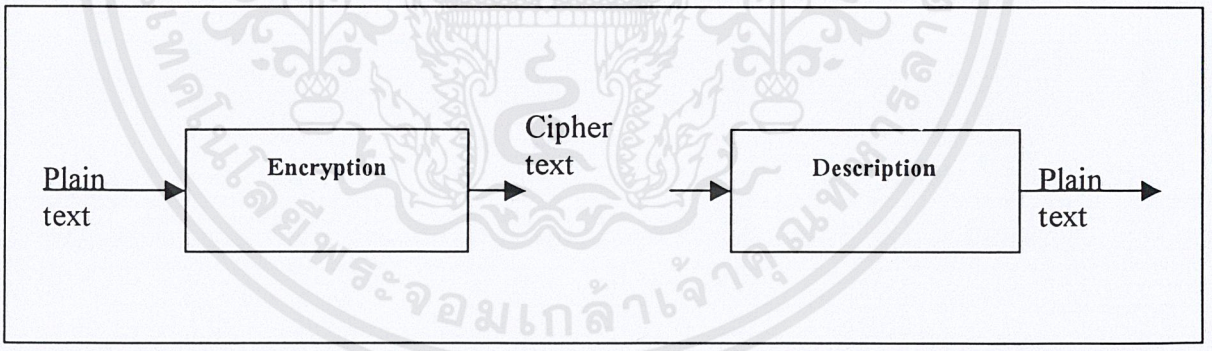
2.18.4 ส่วนติดต่อกับ CPU สำหรับควบคุม CAN แบบ Stand-alone แม้ว่าจะสามารถทำงานได้โดยลำพัง แต่ก็ยังจำเป็นต้องมีการติดต่อกับอุปกรณ์ภายนอกในส่วนของการตั้งค่าเริ่มต้นในการทำงานรวมทั้งการถ่ายโอนข้อมูลที่ต้องการรับส่งกันซึ่งโดยทั่วไปแล้วมักจะนำตัวควบคุม CAN ไปใช้งานร่วมกับหน่วยประมวลผล คือ CPU หรือไมโครคอนโทรลเลอร์ต่างๆจึงต้องมีส่วนที่ใช้ในการติดต่อกับ CPU เหล่านี้ วิธีการติดต่อนั้นก็มีอยู่หลายรูปแบบด้วยกัน มีทั้งการติดต่อแบบขนานและการติดต่อแบบอนุกรม ทั้งนี้ก็เพื่อความสะดวกในการเลือกนำไปใช้นั่นเอง

บทที่ 3
ทฤษฎีการเข้ารหัส

ในการส่งข้อมูลที่เป็นความลับ บางครั้งอาจจะมีการลักลอบอ่านและปลอมแปลงข้อมูลในระหว่างทางที่ส่งข้อมูลไป การเข้ารหัส (Encryption) เป็นวิธีที่จะช่วยไม่ให้มีบุคคลอื่นใดสามารถอ่านข้อมูลที่เราส่งไปได้ จึงทำให้ไม่สามารถที่จะปลอมแปลงข้อมูลของเราได้ การเข้ารหัสเป็นการรักษาความปลอดภัยของข้อมูลอย่างหนึ่ง โดยการเปลี่ยนแปลงข้อมูลโดยรู้กันระหว่างผู้รับและผู้ส่ง การเข้ารหัสมีพื้นฐานอยู่ 2 อย่างคือ

1. **การแทนที่** (Substitution) เป็นการแทนที่บิตใดๆด้วยข้อมูลอื่นทำให้ข้อมูลมีความสับสนยากต่อการถอดรหัส
2. **การสับเปลี่ยนตำแหน่ง** (Permutation) เป็นการสับเปลี่ยนตำแหน่งใดๆของข้อมูลเมื่อมีการสับเปลี่ยนตำแหน่งมากๆทำให้ข้อมูลมีความซับซ้อนยากต่อการเข้ารหัส

ในระบบการเข้ารหัส ข้อมูลที่เราสามารถอ่านได้เรียกว่าเพลนเท็กซ์ (Plain text) ข้อมูลที่เข้ารหัสแล้วจะกลายเป็นข้อมูลที่อ่านไม่ออกเรียกว่า ไซเฟอร์เท็กซ์ (Cipher text) วิธีในการเข้ารหัสเรียกว่า อัลกอริทึมการเข้ารหัส (Encryption Algorithm) ส่วนวิธีการถอดรหัสเรียกอัลกอริทึมการถอดรหัส (Decryption Algorithm) เมื่อนำเพลนเท็กซ์มาเข้ารหัสก็จะเป็นไซเฟอร์เท็กซ์ ซึ่งถ้าเราได้รับไซเฟอร์เท็กซ์มาต้องทำการถอดรหัสเพื่อให้ได้เพลนเท็กซ์ดังรูป 3.1



รูปที่ 3.1 แสดงการเข้าและถอดรหัส

จากรูปสามารถอธิบายได้ดังนี้ข้อมูลที่อยู่ในรูปอนุกรมดังนี้ P = [P1 P2 P3 P4 P5 P6…Pn] เมื่อเข้ารหัสแล้วข้อมูลไซเฟอร์เท็กซ์อยู่ในรูปอนุกรมดังนี้ C = [C1 C2 C3 C4 C5 C6…Cn] สามารถเขียนอยู่ในรูป สมการได้ดังนี้ C = E(P) และ P = D(C)

โดยที่

C = ไซเฟอร์เท็กซ์

P = เพลนเท็กซ์

E = อัลกอริทึมการเข้ารหัส

D = อัลกอริทึมการถอดรหัส

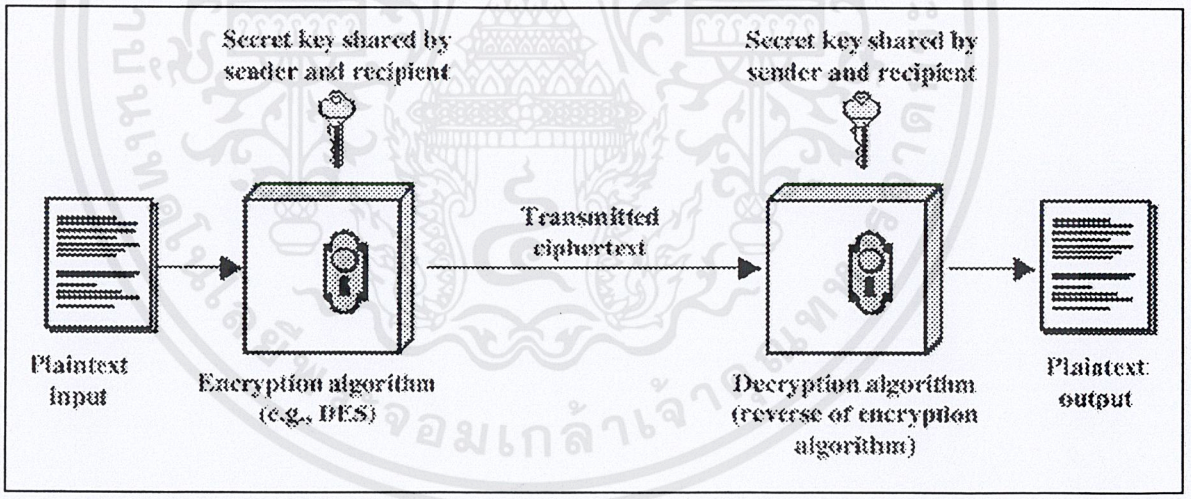
ระบบการเข้ารหัสและการถอดรหัส โดยอาศัยคีย์ในการแปลงข้อมูลมีอยู่ด้วยกัน 2 ประเภทคือ

1 **การเข้ารหัสแบบสมมาตร** (Asymmetric Cryptosystem) กระบวนการเข้าและถอดรหัสที่เป็นกระบวนการแบบเดียวกันในการเข้ารหัสแบบนี้บางทีเรียกว่า การเข้ารหัสแบบ Secret Key

2 **การเข้ารหัสแบบไม่สมมาตร** (Asymmetric Cryptosystem) เป็นการเข้ารหัสที่ใช้ 2 คีย์ที่มีความเกี่ยวข้องกันทางคณิตศาสตร์โดยประกอบด้วย คีย์สาธารณะ (Public Key) และคีย์ส่วนตัว (Private Key) โดยการเข้ารหัสทำด้วยการใช้คีย์สาธารณะการถอดรหัสต้องใช้คีย์ส่วนตัวที่เป็นคู่ของมันในการถอดรหัสเท่านั้น ในทางตรงกันข้ามถ้าใช้คีย์ส่วนตัวในการเข้ารหัสต้องใช้คีย์สาธารณะในการถอดรหัสเท่านั้น

3.1 พื้นฐานการเข้ารหัสแบบสมมาตร

การเข้ารหัสแบบสมมาตรทำงานโดย ทั้งการเข้ารหัสและการถอดรหัสจะทำโดยใช้รูปแบบเดียวกัน หรือจะเรียกว่าการเข้ารหัสแบบคีย์เดียว (Single Key Encryption) เพราะว่าทั้งการเข้าและถอดรหัสนั้นจะใช้คีย์เดียวกันดังรูปที่ 3.2



รูปที่ 3.2 แสดงการเข้าและถอดรหัสแบบสมมาตร

ซึ่งสามารถแสดงด้วยสมการได้ดังนี้

C = E(K,P) และ P = D(K,C) หรือได้ว่า P = D(K(K,C))

โดย C = ไซเฟอร์เท็กซ์

P = เพลนเท็กซ์

E = อัลกอริทึมการเข้ารหัส

D = อัลกอริทึมการถอดรหัส

K = คีย์ในการเข้าและถอดรหัส

ในการใช้งานการเข้ารหัสแบบสมมาตรนี้ จะมีความจำเป็น 2 ประการเพื่อให้การใช้งานได้ผลดีคือ

1 อัลกอริทึมในการเข้ารหัส จะต้องมีความแข็งแกร่งทั้งนี้เนื่องจากการเข้ารหัสแบบนี้อัลกอริทึมมักจะมีการเปิดเผยวิธีการอยู่แล้ว ดังนั้นผู้อื่นก็ย่อมที่จะรู้วิธีการเข้าและถอดรหัสเช่นเดียวกัน ดังนั้นจากส่วนประกอบของกระบวนการเข้ารหัสผู้อื่นก็จะรู้ถึงไซเฟอร์เท็กซ์ได้เพราะสามารถดักจับได้จากกระบวนการส่งและสามารถรู้ อัลกอริทึม แต่สิ่งที่ผู้อื่นไม่รู้ก็คือคีย์ ดังนั้นเพื่อให้ได้ข้อความต้นฉบับกลับมา ก็จะต้องหาคีย์ เพื่อจะได้ใช้ในการถอดรหัสนั้นได้ดังนั้นอัลกอริทึมต้องมีความแข็งแกร่งมากพอที่จะปิดบังคีย์เอาไว้ แม้ว่าจะได้ทั้งเพลนเท็กซ์และไซเฟอร์เท็กซ์ก็ไม่สามารถหาคีย์ได้

2 กระบวนการรักษาคีย์จะต้องมีความปลอดภัยโดยถือว่าผู้รับสารและผู้ส่งสารจะต้องมีช่องทางในการรับส่งคีย์อย่างปลอดภัยอยู่แล้วเพราะหากมีผู้อื่นได้คีย์ไปก็จะไม่เป็นความลับอีกต่อไป

3.2 ลักษณะที่สำคัญของการเข้ารหัสแบบสมมาตร

3.2.1 การแปลงเพลนเท็กซ์เป็นไซเฟอร์เท็กซ์ ใช้กระบวนการ 2 แบบคือ ทั้งการแทนที่ ใช้เพื่อการเปลี่ยนแปลงข้อมูลตำแหน่งใดๆในเพลนเท็กซ์ให้เป็นข้อมูลอื่น และใช้การเปลี่ยนตำแหน่งเพื่อการจัดเรียงข้อมูลใหม่ที่สำคัญคือข้อมูลต้องไม่หาย

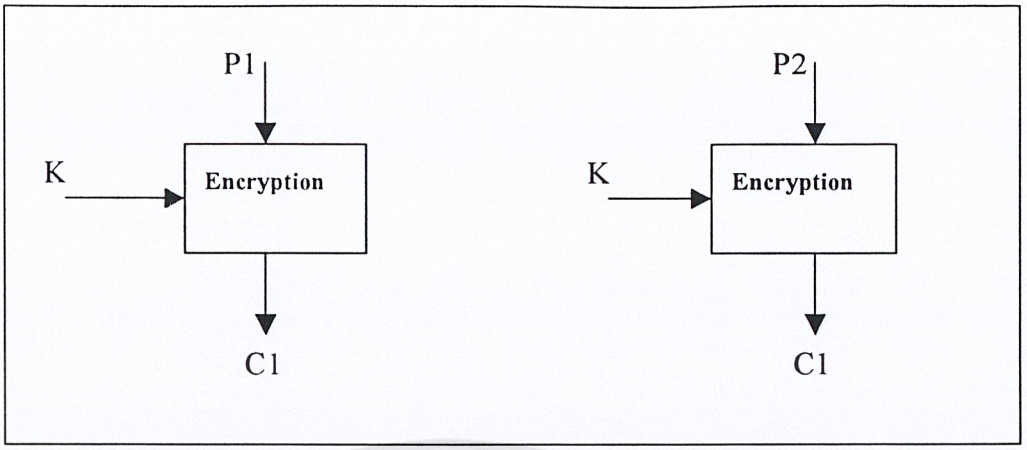
3.2.2 ใช้คีย์เพียงตัวเดียวทั้งผู้รับและผู้ส่ง

3.2.3 ใช้บล็อกไซเฟอร์ในการเข้ารหัส

3.3 ลักษณะของบล็อกไซเฟอร์

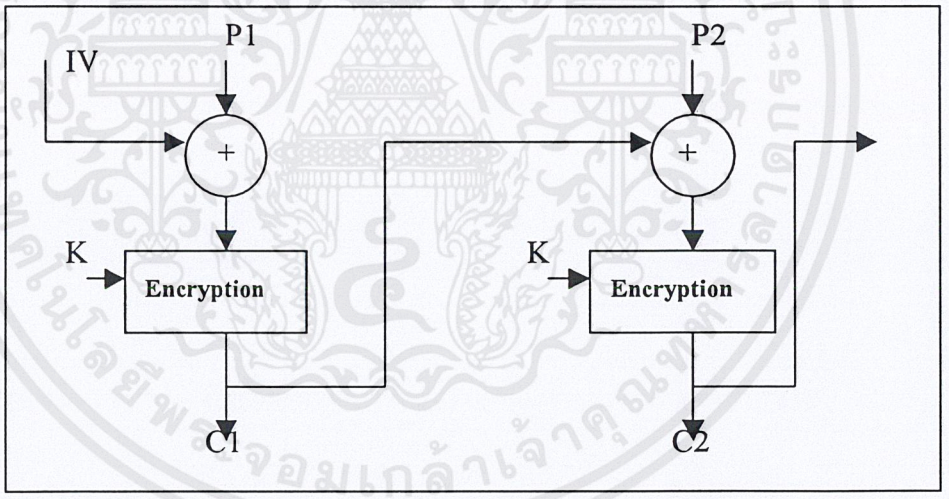
3.3.1 อิเล็กทรอนิกส์โค้ดบุ๊ก (Electronics Codebook : ECB) แบ่งเพลนเท็กซ์ออกเป็นบล็อก บล็อกละเท่าๆกันใช้คีย์และอัลกอริทึมเดียวกันดังรูปที่ 3.3

ข้อดีสำหรับบล็อกไซเฟอร์แบบอิเล็กทรอนิกส์โค้ดบุ๊ก คือสำหรับข้อมูลสั้นๆจะได้ใช้ไซเฟอร์เท็กซ์ที่ดีและสามารถทำได้ทีละหลายบล็อกทำให้เร็ว ถ้าข้อมูลยาวมากๆอาจมีข้อมูลซ้ำกัน เมื่อทำการเข้ารหัสจะทำให้ได้ไซเฟอร์เท็กซ์ที่มีลักษณะซ้ำกันได้



รูปที่ 3.3 แสดงการเข้ารหัสแบบอิเล็กทรอนิกส์โค้ดบุ๊ก

3.3.2 ไซเฟอร์บล็อกเชนนิ่ง (Cipher Block Chaining :CBC) แบ่งเพลนเท็กซ์ออกเป็นบล็อก บล็อกละเท่าๆกัน โดยก่อนการเข้ารหัส จะนำเพลนเท็กซ์มาเอ็กครูซีฟออร์ (XOR) กับไซเฟอร์เท็กซ์ก่อนหน้ามัน โดยเพลนเท็กซ์แรกจะทำการเอ็กครูซีฟออร์ กับเวคเตอร์เริ่มต้น (Initial Vector :IV) ดังรูปที่ 3.4

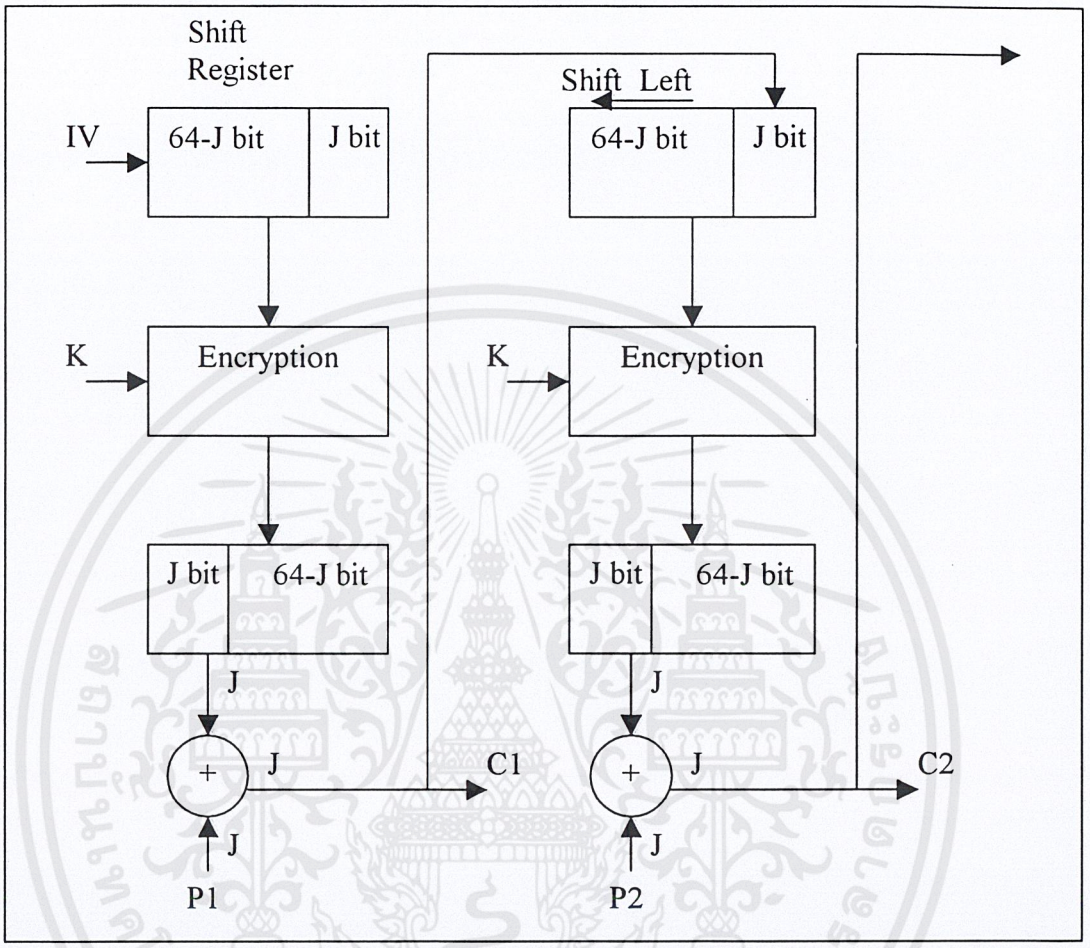


รูปที่ 3.4 แสดงการเข้ารหัสแบบไซเฟอร์บล็อกเชนนิ่ง

ข้อดีของบล็อกไซเฟอร์ลักษณะนี้คือสามารถแก้ปัญหาการซ้ำกันของข้อมูลก่อนการเข้ารหัสเพราะต้องนำเพลนเท็กซ์มาเอ็กครูซีฟออร์ก่อนที่จะนำมาเข้ารหัส แต่มีข้อเสียเนื่องจากการเข้ารหัสแต่ละบล็อกต้องรอ บล็อกข้างหน้ามันก่อนจึงจะทำการเข้ารหัสได้ไม่สามารถทำพร้อมกันได้ทำให้ช้า

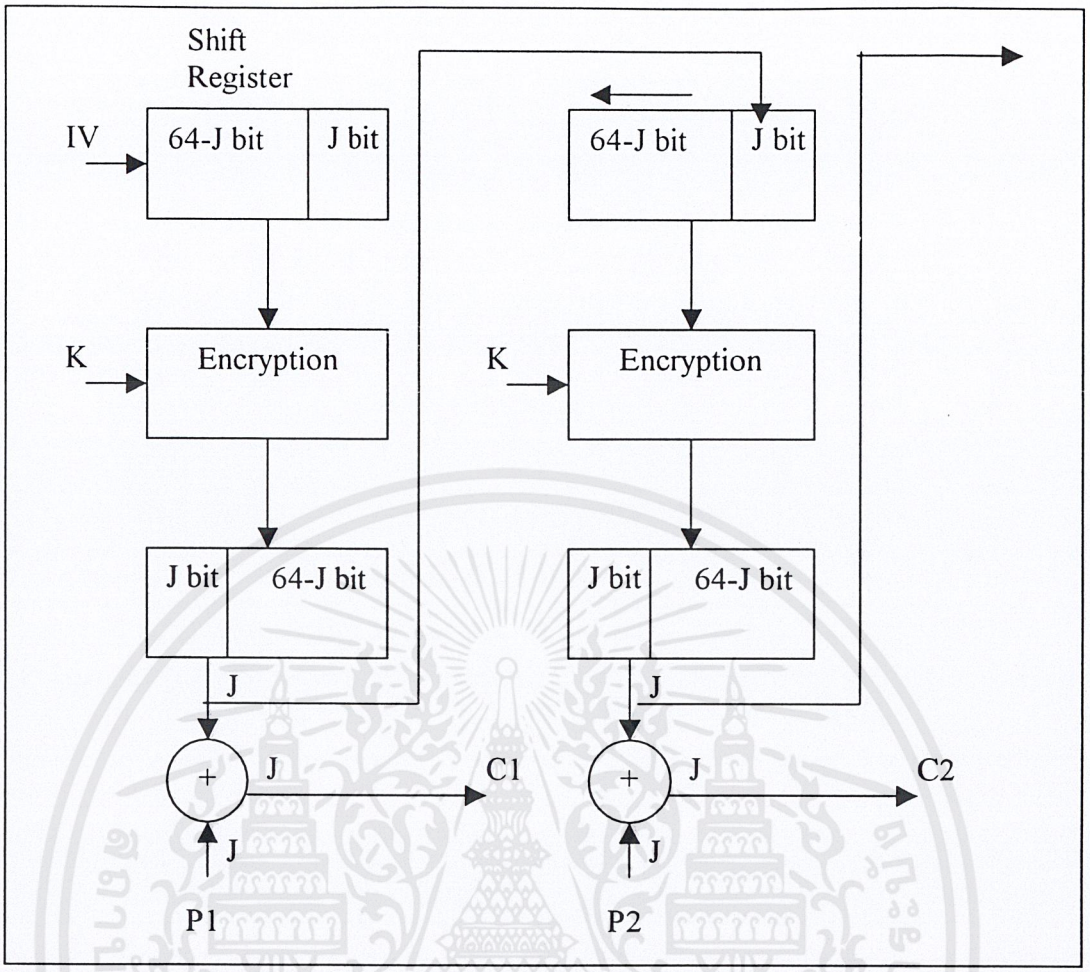
3.3.3 ไซเฟอร์ฟีดแบ็ก (Cipher Feed Back :CFB) ขั้นแรกมี ชิฟรีจิสเตอร์ (Shift Register) ขนาด 64 บิต โดยกำหนดค่าเริ่มต้นเป็นเวคเตอร์เริ่มต้น (IV) แล้วนำมาเข้ารหัสกับคีย์ (K) จากนั้นนำ J บิต หน้าสุดมาเอ็กครูซีฟออร์กับเพลนเท็กซ์แล้วนำไซเฟอร์เท็กซ์ที่ได้ไปใช้ใน ชิฟรี

จิสเตอร์ J บิตท้ายโดยเลื่อนไปด้านซ้าย ไป J บิตเพื่อนำไซเฟอร์เท็กเข้ามาดังรูป 3.5 โดยส่วนใหญ่จะทำทีละ 8 บิตซึ่งคือ 1 ตัวอักษรโดยสามารถทำงานแบบเรียลไทม์ได้



รูปที่ 3.5 แสดงการเข้ารหัสแบบไซเฟอร์ฟีดแบ็ก

3.3..4 เอาท์พุตท์ฟีดแบ็ก (Output Feedback :OFP) เป็นบล็อกไซเฟอร์ฟีดแบ็กเพียงแต่นำข้อมูลที่ได้จากการเข้ารหัส J บิคแรกไปใส่ในชิฟรีจิสเตอร์ J บิตสุดท้ายดังรูปที่ 3.6



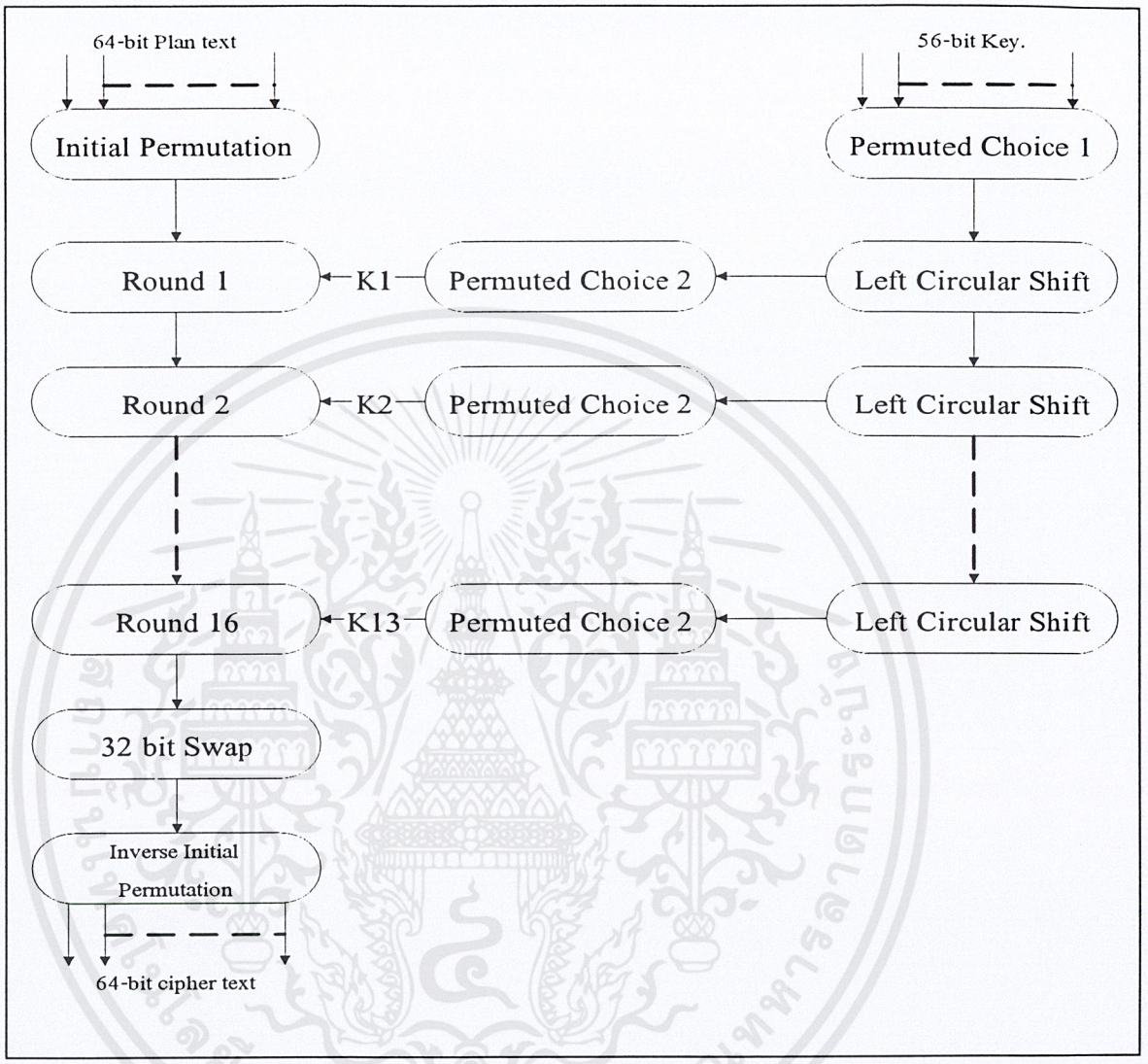
รูปที่ 3.6 การแสดงการเข้ารหัสแบบเอาท์พุตท์ฟีดแบ็ก

3.4 DES (Data Encryption Standard)

เป็นวิธีการเข้ารหัสที่ใช้กันอย่างแพร่หลายที่เป็นพื้นฐาน Data Encryption Standard (DES) ที่พัฒนาขึ้นในปี 1977 โดย National Bureau of Standards ซึ่งปัจจุบันคือ Federal Information Processing Standard 46 (FIPS PUB46) สำหรับ DES ข้อมูลจะถูกเข้ารหัสเป็นบล็อกขนาด 64 บิต ซึ่งใช้คีย์ 56 บิต โดยการจัดการกับข้อมูล 64 บิตที่เข้ามาเพื่อแปลงเป็นข้อมูล 64 บิตออกไป และใช้คีย์เดียวกันนี้ในการถอดรหัส

แม้ว่า DES ถูกนำมาใช้งานตั้งแต่ช่วงทศวรรษที่ 70 (คศ.1960-1970) และได้รับการตอบรับอย่างดีจากนักวิเคราะห์รหัส (Cryptanalysis) อย่างแพร่หลาย แต่ก็เป็นข้อถกเถียงกันเป็นอย่างมากถึงเรื่อง DES นั้นจะปลอดภัยหรือไม่ มีความปลอดภัยมากน้อยขนาดไหน แต่ปัจจุบันเราก็ยังไม่พบช่องโหว่ของ DES ตามเอกสารที่ตีพิมพ์เป็นสาธารณะ แม้ว่าจะใช้คีย์เพียงไม่กี่บิตก็ตาม ในทางตรงกันข้ามแนวคิดแบบ IDEA กลับใช้คีย์แบบ 128 บิต (ซึ่งมากกว่า 2 เท่าของ DES) และได้รับการตอบรับจากสาธารณะตั้งแต่ทศวรรษที่ 90 (คศ.1980-1990) (แต่ดีไม่เท่าตอนประกาศใช้ DES) IDEA มีความปลอดภัยมากกว่า DES และสามารถประมวลผลได้เร็วกว่า DES อย่างไรก็ตาม IDEA ยังต้องรอการตรวจสอบจากผู้เชี่ยวชาญอีกมากถึงเรื่องช่องโหว่ของความปลอดภัย

อัลกอริทึมของการเข้ารหัสแบบ DES



รูปที่ 3.7 General Depiction of DES Encryption Algorithm

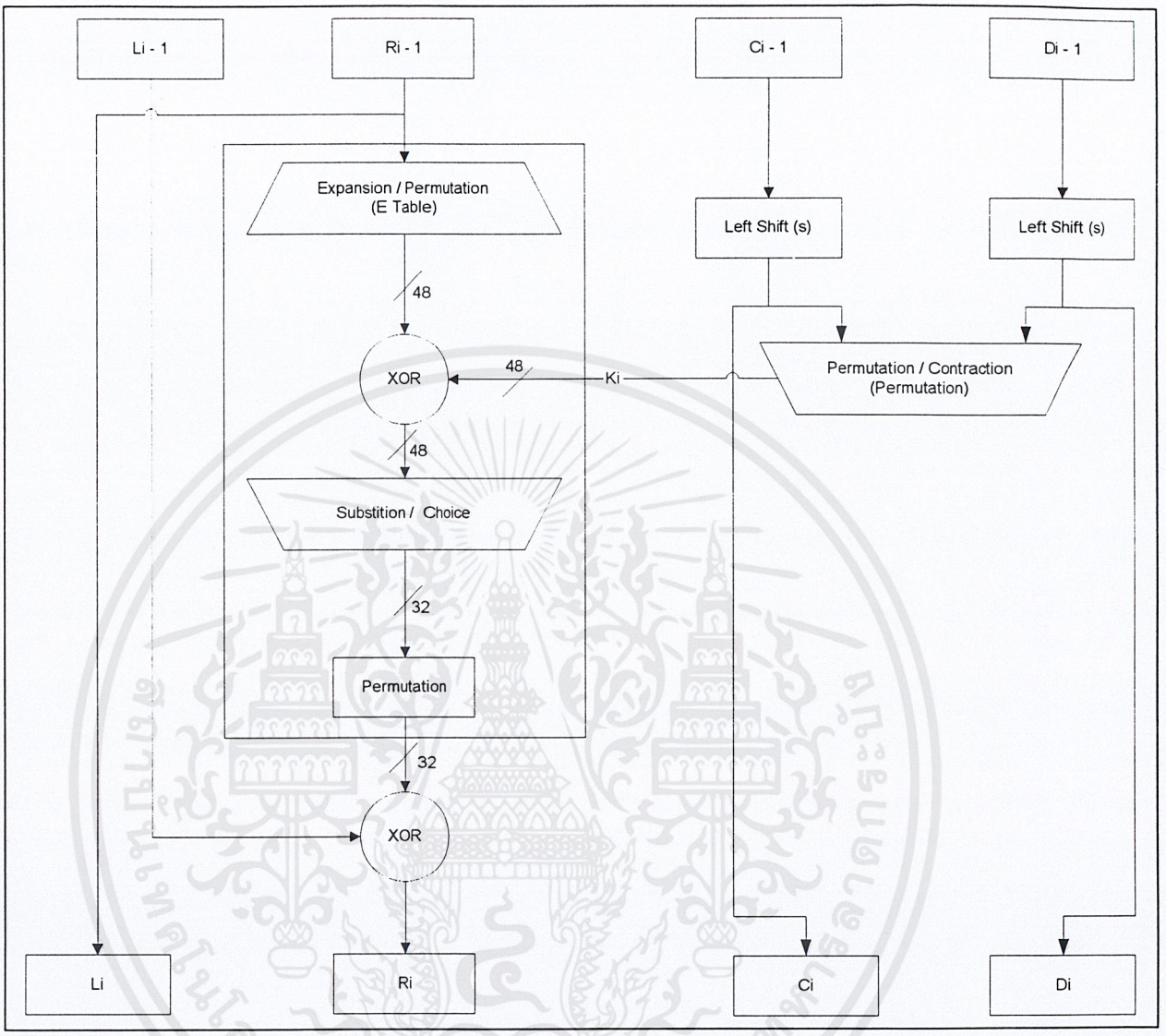
อัลกอริทึมการทำงานของ DES แสดงไว้ดังรูป 3.7 โดย DES จะใช้บล็อกข้อมูลขนาด 64 บิต และใช้คีย์ขนาด 56 บิต โดยหากข้อมูลมีขนาดใหญ่กว่า 64 บิตก็จะแบ่งเป็นบล็อกละ 64 บิต จากรูปทางด้านฝั่งซ้าย จะแสดงการนำบล็อกข้อมูลมาแบ่งออกเป็น 3 ช่วงย่อย โดยช่วงแรกจะเป็นการนำเอาบล็อกข้อมูลมาผ่านการสลับบิตขั้นต้น (Initial Permutation) ซึ่งจะสลับบิตทั้งหมดเสียใหม่ จากนั้นจะเข้าสู่ช่วงที่ 2 โดยประกอบด้วยการทำฟังก์ชัน Round จำนวน 16 ครั้ง และช่วงสุดท้าย จะประกอบไปด้วยการสลับกลุ่มข้อมูล 32 บิตซ้ายและขวา จากนั้นจะนำมาผ่านการสลับบิตย้อนกลับ (Reverse Initial Permutation) อีกครั้ง ก็จะได้ออกมาเป็นไซเฟอร์เท็กซ์ที่มีความยาวเท่ากับเพลนเท็กซ์ที่เข้าไปคือ 64 บิต

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

ตารางที่ 3.1 Initial Permutation และ Inverse Permutation

สำหรับทางด้านฝั่งขวา จะแสดงกระบวนการในการสร้างคีย์ย่อย โดยเริ่มต้นคีย์หลักที่มีความยาว 56 บิต จะผ่านฟังก์ชันการสลับบิต 1 จากนั้นจะนำผลลัพธ์ที่ได้ไปใช้ในการสร้างคีย์ย่อยจำนวน 16 คีย์ใน แต่ละครั้งของการสร้างคีย์ย่อยนั้น จะมีการทำ Circular Shift และนำผลลัพธ์ที่ได้ไปผ่านฟังก์ชันการสลับบิต 2 จากทั้งหมดที่ได้กล่าวมา สำหรับรายละเอียดของการเข้ารหัสในแต่ละรอบนั้น ได้แสดงไว้ในรูปที่ 3.8



รูปที่ 3.8 Single Round of DES Algorithm

จากรูปจะเห็นได้ว่าในแต่ละรอบการทำงานนั้น จะมีการแบ่งข้อมูลออกเป็น 64 บิตที่ได้จากผลของการทำงานในรอบก่อนหน้าออกเป็นข้อมูล 32 บิต 2 ชุด โดยจะเรียกว่าชุด L และชุด R โดยสมการการสร้างชุดข้อมูล L และ R ในรอบ I ได้ดังนี้

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_i\text{-}1, K_1)$$

สำหรับฟังก์ชัน F นั้นเป็นฟังก์ชันที่มีการทำงานในแบบที่มีการสลับบิต (Permutation) และการแทนที่ (Substitution) โดยผ่าน E Table และ S-BOX โดย E Table จะเป็นการสลับบิตในแบบที่มีการขยายข้อความยาวด้วย โดยตาราง E สามารถดูได้จากตารางข้างล่างนี้ จากนั้นจะนำผลลัพธ์ที่ขยายเป็น 48 บิตไป

XOR กับคีย์ย่อย จากนั้นเมื่อผ่าน S-BOX แล้วจะถูกลดความยาวลงเหลือ 32 บิตเท่าเดิม และนำไปผ่านฟังก์ชันสลับบิต P อีกครั้ง

E-bit Selection Table					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P Permutation			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

ตารางที่ 3.2 E-bit Selection Table และ P Permutation

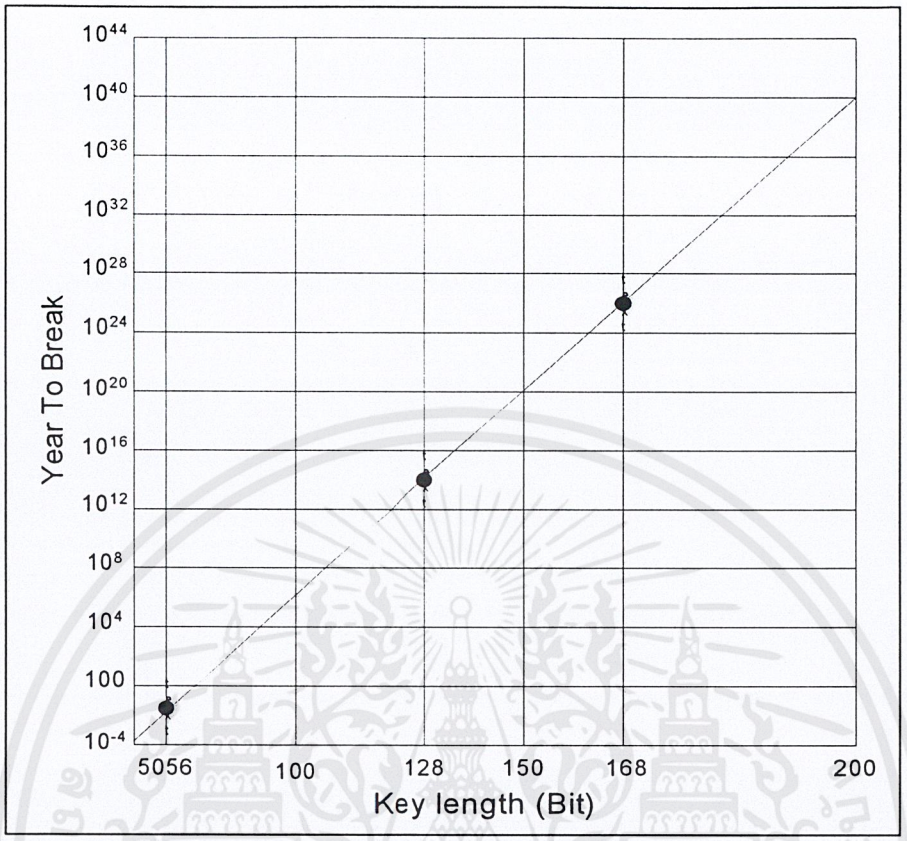
สำหรับกระบวนการในการถอดรหัส DES ก็จะมีลักษณะเช่นเดียวกับการเข้ารหัสทุกประการ ซึ่งถือว่าเป็นสิ่งสำคัญ โดยจะเป็นอัลกอริทึมเดียวกัน เพียงแต่เปลี่ยนอินพุตเป็นไซเฟอร์เท็กซ์และฟังก์ชันที่สร้างคีย์จะต้องสร้างออกมาในลำดับที่ย้อนกลับกันเท่านั้น ทั้งนี้เนื่องจากการเข้ารหัสนี้โดยภาพรวมแล้วจะเป็นการสลับบิตข้อมูลไปมา โดยผ่านตารางที่มีรูปแบบตายตัว ซึ่งเป็นกระบวนการที่ย้อนกลับได้ และผ่านฟังก์ชัน XOR ซึ่งเป็นกระบวนการที่ย้อนกลับได้เช่นเดียวกัน ดังนั้นการออกแบบอัลกอริทึมที่เหมาะสม ก็ทำให้การถอดรหัสง่าย (หากทราบคีย์) แต่จะการแกะจะทำได้ยาก เพราะข้อมูลมีการเปลี่ยนไปมาก

3.5 การพิจารณาถึงความแข็งแกร่งของDES

ในการพิจารณาถึงความแข็งแกร่งของ DES นั้นเราจะพิจารณากันใน 2 ด้านคือ ด้านของตัวอัลกอริทึมเองและความยาวของคีย์ สำหรับเรื่องของอัลกอริทึม นั้นหลังจากที่ DES ได้ประกาศออกมา ก็มีผู้ที่พยายามหาจุดอ่อนของ DES อยู่มากจนอาจกล่าวได้ว่า DES เป็นอัลกอริทึมการเข้ารหัสที่มีผู้ศึกษาค้นคว้ามากที่สุดในโลกก็ได้ แต่จนปัจจุบันจนถึงบัดนี้ก็ยังไม่มีผู้ที่ค้นหาจุดอ่อนของ DES ได้เลย ดังนั้นจึงอาจถือได้ว่ายังเป็นอัลกอริทึมที่ยังไม่มีจุดอ่อน

แต่จุดอ่อนที่น่าสนใจมากกว่าก็คือความยาว 56 บิตของ DES เพียงพอหรือไม่ เนื่องจาก DES เกิดมาในช่วงที่คอมพิวเตอร์ยังไม่มีความเร็วมากนัก แต่หลังจากนั้นก็ได้มีการพัฒนาความสามารถของคอมพิวเตอร์อย่างรวดเร็ว ทำให้ความเป็นไปได้ในการแกะคีย์ขนาด 56 บิตมีความเป็นไปได้มากขึ้น ใน ปี 1998 มีเหตุการณ์ที่ต้องบันทึกไว้ และถือเป็นเหตุการณ์ที่ทำให้อัลกอริทึม DES ถึงจุดจบอย่างเป็นทางการ เหตุการณ์ที่ว่านั้นเกิดจากหน่วยงานหนึ่งที่ชื่อว่า EFF (Electronics Frontier Foundation) ได้สร้างเครื่องคอมพิวเตอร์ขึ้นมาเครื่องหนึ่งเพื่อทำหน้าที่ในการแกะรหัส DES โดยเฉพาะโดยใช้ชื่อว่า “DES Cracker” โดยใช้เงินทุนไม่ถึง 250,000 เหรียญ โดยสามารถแกะคีย์ขนาด 56 บิตได้ในเวลา 3 วันเท่านั้น ยิ่งไปกว่านั้น EFF ได้เผยแพร่ผลงานตนเองออกสู่สาธารณะทำให้ทุกคนสามารถสร้างได้

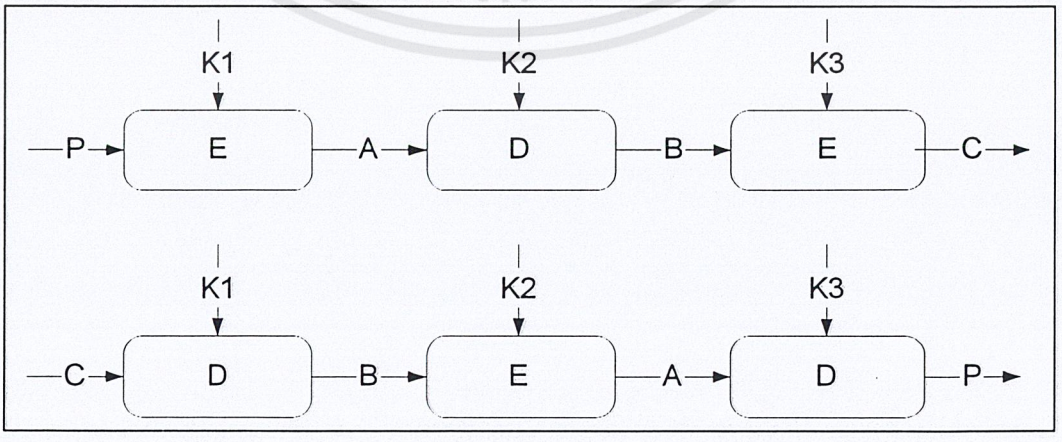
อย่างไรก็ตาม ในการสร้างการแกะรหัสนั้นจะเริ่มจากการใส่ไซเฟอร์เท็กซ์เข้าไปจากนั้นก็ใส่คีย์เข้าทีละตัวอย่าง ซึ่งจะทำให้เกิดเป็นเพลนเท็กซ์ออกมา แต่เพลนเท็กซ์จะเป็นเพลนเท็กซ์ที่ถูกต้องคือตรงกับต้นฉบับก็ต่อเมื่อคีย์ที่ใส่เข้าไปเป็นคีย์ที่ถูกต้อง คราวนี้จะรู้ได้อย่างไรว่าเพลนเท็กซ์ที่ถูกต้อง ซึ่งหมายถึงคีย์ที่ถูกต้องด้วย นั่นคือ การพิจารณาดูเนื้อความเช่น หากต้นฉบับเป็นภาษาอังกฤษเพลนเท็กซ์ที่ถูกต้องก็ต้องเป็นภาษาอังกฤษด้วย ดังนั้นก็จะใช้วิธีดูไปเรื่อยๆว่าหากผลลัพธ์ที่แกะออกมาเป็นภาษาที่อ่านได้ ก็หมายความว่าคีย์ที่ใช้เป็นคีย์ที่ถูกต้องแล้ว แต่ถ้าหากเป็นไฟล์ที่เข้ารหัสเป็นไฟล์แบบอื่น เช่น ไฟล์ที่มีตัวเลขอย่างเดียว หรือไฟล์ที่ผ่านการบีบข้อมูลมาแล้วก็ยิ่งจะหาคีย์ที่ถูกต้องได้ยากยิ่งขึ้นหรืออาจทำไม่ได้เลยก็ได้



รูปที่ 3.9 Time to break a code

3.6 Triple DES

อัลกอริทึม 3DES ได้รับการเสนอครั้งแรกโดย Tuchman โดยเริ่มแรกเป็นมาตรฐานของ ANSI หมายเลข X9.17 ในปี 1985 จากนั้น NIST ได้นำมาเป็นส่วนหนึ่งของมาตรฐานหมายเลข FIPS PUB 46-3 ในปี 1999 โดย 3DES จะใช้อัลกอริทึมเดียวกับ DES แต่จะใช้คีย์จำนวน 3 คีย์ และทำ DES จำนวน 3 ครั้ง ดังรูป 2.18



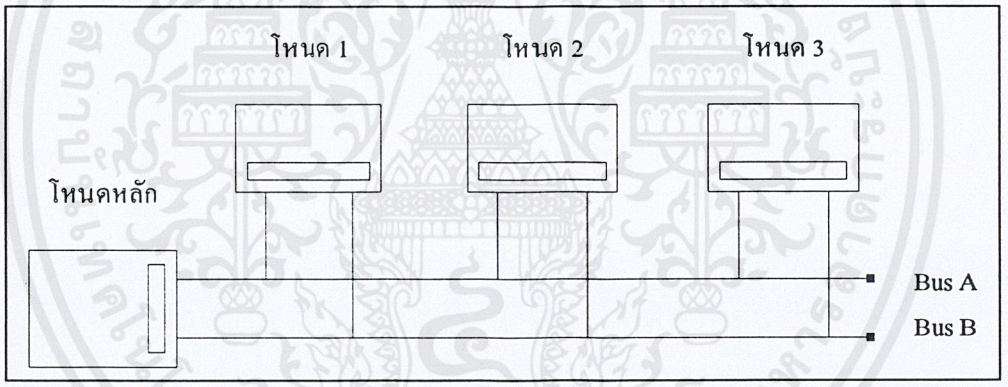
รูปที่ 3.10 Triple DES

ดังนั้นคีย์ทั้งหมดจะมีความยาวเท่ากับ 168 บิต อย่างไรก็ตาม FIPS PUB 46-3 ยอมให้ใช้เพียง 2 คีย์ คือกำหนดให้คีย์ K1 เท่ากับคีย์ K3 ได้ ดังนั้นความยาวของคีย์จะเหลือเท่ากับ 112 บิต กล่าวโดยสรุป 3DES เป็นการปรับปรุง DES ให้มีความปลอดภัยมากขึ้น สามารถใช้งานร่วมกับ DES ได้

บทที่ 4
การออกแบบ

4.1 ภาพรวมของระบบ

โพรโตคอลที่ได้ทำการออกแบบสนับสนุนรูปแบบของเครือข่าย (Topology) หลายประเภทด้วยกันไม่ว่าจะเป็น การควบคุมจากจุดศูนย์กลาง (Centralized) หรือว่าจะเป็น Master-Slave ระบบที่ได้นำเสนอถูกออกแบบให้มีลักษณะการสื่อสารแบบควบคุมจากจุดศูนย์กลาง กล่าวคือภายในระบบจะประกอบไปด้วยหลายๆโหนด แต่จะมีโหนดหนึ่งซึ่งทำหน้าที่เป็นโหนดหลักของเครือข่าย การสื่อสารที่เกิดขึ้นจะมีเฉพาะโหนดลูกกับโหนดหลักเท่านั้น จะไม่มีการสื่อสารระหว่างโหนดลูกกับโหนดลูกด้วยกันเอง การควบคุมจากจุดศูนย์กลางมีประโยชน์ในหลายอย่าง เช่นการตรวจสอบสถานะต่างๆของเครือข่าย, การควบคุมการทำงานหรือแม้แต่การควบคุมการสื่อสารสามารถดำเนินการได้ที่โหนดควบคุมหลัก สามารถแสดงได้ดังรูปที่ 4.1



รูปที่ 4.1 ภาพรวมของระบบ

4.1.1 การกำหนด ID ให้กับโหนด

เนื่องด้วยระบบเครือข่ายถูกออกแบบให้มีลักษณะการสื่อสารแบบควบคุมจากจุดศูนย์กลางและตรวจสอบการนำส่งข้อมูล ดังนั้นแต่ละโหนดจะมี ID ของ Message ที่ใช้ในการสื่อสารต่างกันเพื่อให้โหนดหลักเข้าใจว่ากำลังสื่อสารกับโหนดใด โดยใช้ขนาดของ ID เป็นแบบมาตรฐาน (Standard Identifier) คือ ID จะมีขนาด 11 บิต แต่เนื่องจากการออกแบบโพรโตคอลอาศัยเทคนิคที่เรียกว่า Message on ID กล่าวคือมีการนำข้อมูลบางส่วนไปเก็บไว้บน ID ทำให้ขนาด ID ที่ใช้งานจริงมีขนาดเล็กลง

Message on ID ดังที่ได้กล่าวไปประกอบไปด้วย More frame flag ขนาด 1 บิตและ Frame type ขนาด 2 บิต ทำให้ ID มีขนาดเท่ากับ 11 – 1 – 2 = 8 บิต ดังนั้นจำนวน ID ของ Message โดยที่

ไม่ซ้ำกันเลยจะมี 2^8 ID หรือเท่ากับ 256 Message ID การกำหนด ID ที่ใช้ในการสื่อสารระหว่างโหนดหลักกับโหนดลูกสามารถแสดงได้ดังตารางที่ 4.1

โหนด	ID จากโหนดลูกไปโหนดหลัก	ID จากโหนดหลักไปโหนดลูก
1	0000001	1000001
2	0000010	1000010
3	0000011	1000011

ตารางที่ 4.1 แสดงการกำหนด ID ให้กับ Message บนระบบ

4.1.2 จำนวนโหนดในเครือข่าย

เนื่องในปริญญานิพนธ์นี้เลือกใช้ CAN โพรโตคอลโดยใช้ขนาดของ ID เป็นแบบมาตรฐาน คือ ID จะมีขนาด 11 บิต และสงวนไว้ใช้งานพิเศษจำนวน 3 บิตดังนั้นจึงสามารถกำหนด ID ให้กับ Message ที่แตกต่างกันได้ถึง 2^8 Message ID ส่วนที่ถูกสงวนไว้ใช้งานพิเศษจำนวน 3 บิตประกอบไปด้วย More frame flag ขนาด 1 บิตและ Frame type ขนาด 2 บิต ทำให้ ID มีขนาดเท่ากับ $11 - 1 - 2 = 8$ บิต ดังนั้นจำนวน ID ของ Message บนเครือข่ายโดยที่ไม่ซ้ำกันเลยจะมี 2^8 Message ID หรือเท่ากับ 256 Message ID แต่หากเป็น CAN โพรโตคอลโดยใช้ ID แบบขยายจะมีขนาดทั้งหมด 29 บิต แต่สงวนไว้ใช้งานพิเศษจำนวน 3 บิต ทำให้ ID มีขนาดเท่ากับ $29 - 1 - 2 = 26$ บิต ดังนั้นจำนวน ID ของข่าวสารบนเครือข่ายโดยที่ไม่ซ้ำกันเลยจะมี 2^{26} Message ID หรือเท่ากับ 67,108,864 Message ID

เมื่อพิจารณาถึงโหนดที่สามารถมีอยู่ในเครือข่ายในข้อกำหนดที่ว่า แต่ละโหนดต้องการข่าวสารที่ต่างกัน จำนวนโหนดสูงสุดที่เป็นไปได้จะเท่ากับจำนวนของข่าวสารที่เป็นได้ คือ 256 โหนดบน CANโพรโตคอลแบบมาตรฐาน และ 67,108,864 โหนดในแบบขยาย แต่ทั้งนี้ขึ้นอยู่กับความเหมาะสมซึ่งต้องพิจารณาถึงความหนาแน่นของข้อมูลที่ส่งผ่านบนบัสด้วย

4.1.3 ความเร็ว

ความเร็วสูงสุดในการสื่อสารด้วย CAN โพรโตคอลสามารถสื่อสารได้ที่ 1 เมกะบิตต่อวินาที ที่ความยาวสายสัญญาณ 40 เมตร และสามารถส่งได้ไกลสุด 10 กิโลเมตรที่ความเร็ว 5 กิโลบิตต่อวินาที ซึ่งความเร็วจะสัมพันธ์กับความยาวสายสัญญาณดังตารางที่ 4.2

Max. Distance	Bit Rate	Type
10 m	1.6 Mbit/s	High - speed
40 m	1.0 Mbit/s	
130 m	500 Kbit/s	
270 m	250 Kbit/s	
530 m	125 Kbit/s	
620 m	100 Kbit/s	Low - speed
1,300 m	50 Kbit/s	
3,300 m	20 Kbit/s	
6,700 m	10 Kbit/s	
10,000 m	5 Kbit/s	

ตารางที่ 4.2 ความยาวสายและอัตราเร็วในการส่งข้อมูล

4.2 CAN High Layer Protocol

CAN High Layer Protocol เป็นการนำ CAN โพรโตคอลมาประยุกต์ใช้งานจากพื้นฐานการทำงานในชั้น Physical Layer และ Data Link Layer ซึ่งจะเพิ่มความสามารถของระบบที่นำ CAN ไปใช้งาน โดยจะการกำหนดในชั้นที่ 7 คือชั้นประยุกต์ใช้งาน (Application Layer) ตามแบบจำลอง ISO/OSI จากการศึกษา CAN High Layer Protocol หลายๆตัวได้แก่ CANopen, DeviceNet, CAN Kingdom, OSEK/VDX,SDS และ J1939 สามารถสรุปข้อกำหนดของ CAN High Layer Protocol ได้ดังนี้

1. การกำหนด ID ให้กับ Message
2. ส่งข้อมูลได้มากกว่า 8 ไบต์
3. ควบคุมลำดับการนำส่งข้อมูล
4. กำหนดหน้าที่ให้กับข้อมูล
5. รายงานสถานะอุปกรณ์
6. ตรวจสอบการนำส่งข้อมูล
7. สนับสนุนการเสริมสำรอง
8. ความปลอดภัยของข้อมูล

4.2.1 การกำหนด ID ให้กับ Message

การส่งข้อมูลใน CAN จะไม่มีการระบุที่อยู่ของโหนดผู้รับและโหนดผู้ส่งเหมือนดังที่ใช้กันอยู่ทั่วไป แต่จะใช้การระบุด้วยค่าประจำตัว (Identifier) ของข้อมูลหรือข้อความแทน เพื่อเป็นการบ่ง

บอกชนิดและระดับความสำคัญของข้อมูล ดังนั้นการกำหนดรูป ID จึงมีส่วนสำคัญที่จะทำให้โหนดบนเครือข่ายสามารถรับและส่งข้อมูลกันได้อย่างมีประสิทธิภาพ สิ่งที่ต้องพึงระวังในการรับส่งข้อมูลโดยใช้ ID คือการกำหนด ID ให้กับ Message เนื่องจากว่าถ้าหากมีการกำหนด ID ให้กับ Message ซ้ำซ้อนกันอาจทำให้เกิดข้อผิดพลาดในการรับส่งข้อมูลเนื่องจากการชนกันของข้อมูลได้

4.2.2 ส่งข้อมูลได้มากกว่า 8 ไบต์

เนื่องจาก CAN โพรโตคอลสามารถส่งข้อมูลครั้งละ 0-8 ไบต์ ดังนั้นระบบที่ต้องการส่งข้อมูลที่มีขนาดมากกว่า 8 ไบต์จะไม่สามารถส่งข้อมูลโดยใช้ CAN โพรโตคอลได้ เพราะฉะนั้นแล้วโพรโตคอลที่ได้พัฒนาขึ้นมาจึงออกแบบให้สามารถส่งข้อมูลได้มากกว่า 8 ไบต์ โดยใช้เทคนิคที่เรียกว่า Message on ID โดยจะเพิ่มข้อมูลพิเศษที่เรียกว่า More Frame Flag ไว้บน ID เพื่อบอกจุดสิ้นสุดของแพ็กเก็ต

4.2.3 ควบคุมลำดับการนำส่งข้อมูล

ถึงแม้ว่าการสื่อสารโดยใช้เทคโนโลยีบัสจะไม่มีโอกาสที่ลำดับของเฟรมจะไปถึงปลายทางโดยมีลำดับสลับกัน แต่เนื่องจากตัวควบคุม CAN ส่วนใหญ่จะมีบัฟเฟอร์ขารับมากกว่า 1 ชุด อีกทั้งยังมีความสามารถในการทำ Roll Over หรือการนำข้อมูลที่ได้รับมาใหม่ไปเก็บไว้ในบัฟเฟอร์อื่นในกรณีที่บัฟเฟอร์หลักไม่ว่าง ซึ่งจะเห็นได้ว่าอาจเกิดข้อผิดพลาดขึ้นในการประกอบข้อมูลกลับในกรณีที่ลำดับการอ่านข้อมูลจากบัฟเฟอร์ไม่แน่นอน ดังนั้นระบบที่ออกแบบจึงเพิ่มเติมในส่วนของ Sequence Number เพื่อใช้ในการบอกลำดับของเฟรมซึ่งจะทำให้การประกอบเฟรมข้อมูลกลับเป็นแพ็กเก็ตทำได้อย่างถูกต้อง

4.2.4 กำหนดหน้าที่ให้กับข้อมูล

นอกจากรับและส่งข้อมูลผ่านระบบเครือแล้วสิ่งที่สำคัญยิ่งกว่าคือการนำข้อมูลนั้นไปใช้งาน ดังนั้นการกำหนดหน้าที่ให้กับข้อมูลจึงมีความสำคัญมากเนื่องจากการรับส่งข้อมูลใดๆจะไม่มีประโยชน์เลยถ้าไม่มีการนำข้อมูลนั้นไปใช้งาน

เนื่องจากในการออกแบบโปรแกรมนั้นได้แสดงเฉพาะในส่วนของการทดสอบโพรโตคอลที่ได้ทำการออกแบบเท่านั้น ดังนั้นจึงไม่มีการกำหนดหน้าที่ให้กับข้อมูล

4.2.5 รายงานสถานะอุปกรณ์

การรายงานสถานะอุปกรณ์ในเครือข่ายเป็นส่วนที่จะช่วยอำนวยความสะดวกให้ผู้ดูแลระบบในการตรวจสอบ, ดูแลและสั่งงานผ่านระบบควบคุม

ระบบเครือข่ายที่ได้พัฒนาขึ้นมานี้สามารถรายงานสถานะของโหนดและ CAN ว่าอยู่ในสถานะ Active หรืออยู่ในสถานะ Standby โดยโหนดหลักจะทำการสอบถาม (Polling) สถานะของ

โหนดตามเวลาที่ผู้ใช้งานได้ตั้งไว้ จากนั้น โหนดที่ได้การสอบถามจะส่งข้อมูลสถานะกลับไปยังโหนดหลักเพื่อทำการแสดงผลและประมวลผลต่อไป

4.2.6 ตรวจสอบการนำส่งข้อมูล

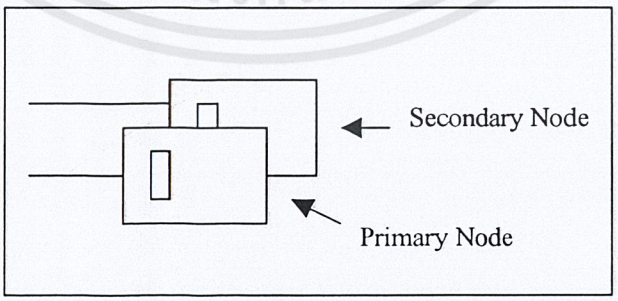
จุดเด่นอย่างหนึ่งของ CAN โพรโตคอลคือรับประกันว่าข้อมูลจะถูกส่งออกไปอย่างถูกต้อง โดยอาศัยฟิลด์ตอบสนอง (Acknowledge Field) ซึ่งเป็นส่วนที่โหนดผู้รับใช้ตอบกลับว่าข้อมูลที่ส่งมาได้รับถูกต้องหรือไม่ แต่ไม่ได้รับประกันว่าข้อมูลชุดนั้นจะถูกส่งไปถึงโหนดปลายทางจริงๆ ดังนั้น การตรวจสอบว่าข้อมูลจะถูกส่งโหนดปลายทางจริงๆจึงเป็นสิ่งที่จำเป็นเมื่อผู้ใช้งานต้องการมั่นใจว่าข้อมูลที่ส่งออกไปนั้นโหนดปลายทางจะได้รับ

การตรวจสอบว่าข้อมูลจะถูกส่งถึงปลายทางนั้นจะอาศัยการตอบกลับ (Acknowledge) จากโหนดปลายทางโดยอาศัยเฟรมในการรายงานสถานะ การตอบกลับของโหนดปลายทางจะกระทำเฉพาะกับเฟรมที่มี Flag More Frame เป็น 0 หรือเฟรมสุดท้ายเท่านั้น เนื่องจากถ้าตอบกลับทุกเฟรมอาจจะทำให้เกิดทราฟฟิกบนเครือข่ายสูง อีกทั้งไม่มีความจำเป็นที่จะต้องตอบกลับทุกเฟรม

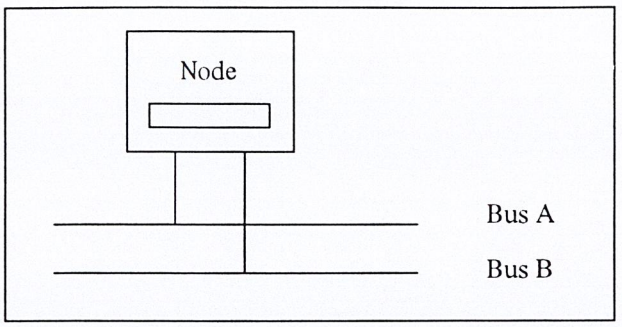
4.2.7 สนับสนุนการเสริมสำรอง

ในระบบที่ต้องการความน่าเชื่อถือและความยืดหยุ่นสูง การเสริมสำรองจะเป็นกลไกหนึ่งซึ่งถูกนำมาใช้ แต่สิ่งที่จะเกิดขึ้นตามมาอย่างหลีกเลี่ยงไม่ได้ก็คือค่าใช้จ่ายที่เพิ่มขึ้น ดังนั้นการออกแบบระบบที่ดีจะต้องเป็นระบบที่มีความน่าเชื่อถือและความยืดหยุ่นสูงโดยมีค่าใช้จ่ายต่ำ แต่ในทางกลับกันบางระบบกลับไม่ต้องการความน่าเชื่อถือสูงมากนักอันเนื่องมาจากมูลค่าของข้อมูลไม่สูงมากนั่นเอง ดังนั้นทางเลือกที่หนึ่งคือการเลือกอุปกรณ์เพื่อทำการเสริมสำรองเพียงบางส่วนจึงเป็นทางออกที่ดี อุปกรณ์ที่สนับสนุนการเสริมสำรองสามารถแสดงได้ดังนี้

1. การเสริมสำรองโหนด
2. การเสริมสำรองสายสัญญาณ



รูปที่ 4.2 ตัวอย่างการเสริมสำรองโหนด



รูปที่ 4.3 ตัวอย่างการเสริมสำรองสายสัญญาณ

ในปริญญานิพนธ์ฉบับนี้ได้เลือกการเสริมสำรองเฉพาะในส่วนของสายสัญญาณเท่านั้นเนื่องจากเป็นการสื่อสารโดยใช้สายสัญญาณ (Bus Topology) ดังนั้นสายสัญญาณเป็นส่วนที่มีความสำคัญกับระบบมากเนื่องมาจากหากสายสัญญาณมีปัญหาเกิดขึ้น อาจทำให้ทั้งระบบไม่สามารถใช้งานได้เลย

4.2.8 ความปลอดภัยของข้อมูล

ความปลอดภัยของข้อมูลก็เป็นอีกส่วนหนึ่งที่ควรจะพิจารณาร่วม สำหรับระบบที่ต้องการปลอดภัยในการสื่อสาร ระดับของความปลอดภัยนี้จะต้องมากน้อยแค่ไหนขึ้นอยู่กับความจำเป็นของระบบ เนื่องจากว่าในการเพิ่มเติมส่วนนี้เข้าไปนั้นถึงแม้จะทำให้ข้อมูลมีความปลอดภัยเพิ่มขึ้น แต่ก็เพิ่มภาระให้กับการประมวลผลด้วยเพราะวิธีการในการเข้ารหัสยิ่งซับซ้อนเท่าไรก็ยิ่งปลอดภัยมากขึ้นเช่นกัน

สำหรับระบบที่ออกแบบนี้เลือกที่จะใช้การเข้ารหัสแบบ DES ที่มีคีย์ขนาด 56 บิตทำการเข้ารหัสข้อมูลคราวละ 64 บิตโดยใช้คีย์ตายตัว หมายความว่าคีย์จะที่เก็บไว้ที่ตัวโหนดหลักกับโหนดลูกจะเป็นคีย์เดียวกันและไม่มีการแลกเปลี่ยนคีย์เนื่องจากว่าการแลกเปลี่ยนคีย์จะต้องใช้เส้นทางที่มีความปลอดภัยและการสื่อสารของระบบควบคุมโดยปกติแล้วส่วนของคาต้าจะไม่มีส่วนที่มีลักษณะเป็นภาษาที่สามารถอ่านได้ ดังนั้นการที่จะหาคีย์ที่ถูกต้องจะทำได้ยากหรืออาจทำไม่ได้เลย

4.3 การออกแบบเฟรมข้อมูล

จากการกำหนด CAN High Layer Protocol ในหัวข้อที่ 4.2 นั้น ทำให้ต้องมีการกำหนดเฟรมขึ้นมาใหม่ โดยที่เฟรมที่ได้ทำการออกแบบนั้นจะต้องสนับสนุนข้อกำหนด CAN High Layer Protocol อีกทั้งยังต้องมีความสอดคล้องกับรูปแบบของเฟรมในระดับ CAN โพรโตคอล

เฟรมที่ทำการออกแบบนั้นสามารถแบ่งออกได้เป็น 2 ประเภทได้แก่เฟรมที่ใช้ในการส่งข้อมูลและเฟรมที่ใช้ในการรายงานสถานะ (Status Report Frame) โดยที่เฟรมทั้งสองประเภทจะอาศัย CAN โพรโตคอลในการรับส่งข้อมูล

4.3.1 เฟรมที่ใช้ในการรับส่งข้อมูล

ID	Flag	Total length	Checksum	Data

รูปที่ 4.4 เฟรมสำหรับการรับส่งข้อมูล

เฟรมที่ใช้ในการรับส่งข้อมูลนี้จะทำหน้าที่ในการถ่ายโอนข้อมูลจากโหนดหนึ่งบนเครือข่ายไปยังอีกโหนดหนึ่งหรือหลายๆโหนด โดยเฟรมที่ใช้ในการรับส่งข้อมูลนี้จะรับประกันความถูกต้องของข้อมูลโดยอาศัยผลรวมซึ่งจะสร้างขึ้นโดยผู้ส่งและจะถูกตรวจสอบโดยผู้รับ อีกทั้งยังใช้เทคนิค Message On Identifier หมายความว่ามีการใช้พื้นที่บางส่วนของ ID ในเก็บข้อมูล ประกอบไปด้วยฟิลด์ต่างๆดังนี้

4.3.1.1 ID : เป็นฟิลด์ที่ใช้ในการระบุชนิดและระดับความสำคัญของข้อมูล มีขนาด 8 บิต

4.3.1.2 Flag : ใช้ในการระบุถึงชนิดของเฟรมและใช้ระบุว่าจุดสิ้นสุดของแพ็กเก็ต

4.3.1.3 Total length : ใช้ระบุขนาดของข้อมูล มีขนาด 8 บิต สามารถส่งข้อมูลได้สูงสุด 256 ไบต์

4.3.1.4 Checksum (ผลรวม) : เป็นฟิลด์ที่ใช้ในการตรวจสอบข้อผิดพลาดของเฮดเดอร์และข้อมูล มีขนาด 16 บิต การคำนวณผลตรวจสอบจะเริ่มต้นด้วยการให้ฟิลด์ผลรวมมีค่าเป็น 0 จากนั้นจึงบวกเฮดเดอร์และข้อมูลครั้งละ 8 บิต แบบ 1's complement เมื่อได้ผลลัพธ์แล้ว จะนำใส่ในฟิลด์ผลรวมที่โหนดปลายทางเมื่อได้รับข้อมูลแล้วก็เพียงแต่บวกเฮดเดอร์ทั้งหมดครั้งละ 8 บิต หากได้ค่าไม่เท่ากับศูนย์แสดงว่ามีข้อผิดพลาดเกิดขึ้น

4.3.1.5 Data : ข้อมูล มีขนาดตั้งแต่ 0 - 256 ไบต์

4.2.2 เฟรมที่ใช้ในการรายงานสถานะ

ID	Flag	Checksum	Type	Description

รูปที่ 4.5 เฟรมสำหรับรายงานสถานะ

เฟรมที่ใช้ในการรายงานสถานะจะทำหน้าที่ในการรายงานสถานะของอุปกรณ์, รายงานความผิดพลาดในการตรวจสอบความถูกต้องของข้อมูล, ใช้เป็นเฟรมตอบกลับ, ตลอดจนการร้องขอสถานะและร้องขอการเปลี่ยนแปลงสายสัญญาณ การใช้งานปกติจะใช้ควบคู่กับ Description เสมอ ประกอบไปด้วยฟิลด์ต่างๆดังนี้

4.2.2.1 ID : เป็นฟิลด์ที่ใช้ในการระบุชนิดและระดับความสำคัญของข้อมูล มีขนาด 8 บิต

4.2.2.2 Flag : ใช้ในการระบุถึงเฟรมและใช้ระบุว่าจุดสิ้นสุดของแพ็กเก็ต

4.2.2.3 Checksum (ผลรวม) : เป็นฟิลด์ที่ใช้ในการตรวจสอบข้อผิดพลาดของเฮดเดอร์และข้อมูล มีขนาด 16 บิต การคำนวณผลตรวจสอบจะเริ่มต้นด้วยการให้ฟิลด์ผลรวมมีค่าเป็น 0 จากนั้นจึงบวกเฮดเดอร์และข้อมูลครั้งละ 8 บิต แบบ 1's complement เมื่อได้ผลลัพธ์แล้ว จะนำใส่ในฟิลด์ผลรวมที่โหนดปลายทางเมื่อได้รับข้อมูลแล้วก็เพียงแต่บวกเฮดเดอร์ทั้งหมดครั้งละ 8 บิต หากได้ค่าไม่เท่ากับศูนย์แสดงว่ามีข้อผิดพลาดเกิดขึ้น

4.2.2.4 Type : กำหนดค่าความผิดพลาดและการรายงานสถานะโดยมีความหมายดังนี้

Type	ความหมาย
0	รายงานสถานะของโหนด
1	ข้อมูลผิดพลาด
2	เฟรมตอบกลับ
3	ร้องขอสถานะของโหนด
4	ร้องขอเปลี่ยนสายสัญญาณ

ตารางที่ 4.3 กำหนดค่าความผิดพลาดและการรายงานสถานะ

4.2.2.5 Description : ใช้ร่วมกับ Type โดยมีความหมายดังนี้

Type	หน้าที่ของ Description
0	ระบุถึงสถานะของอุปกรณ์
1	หมายเลขเฟรม
2	หมายเลขเฟรม
3	ไม่ใช้งาน
4	บัสที่ร้องขอ

ตารางที่ 4.4 ความหมายของ Description

4.2.3 รูปแบบของเฟรมในระดับ CAN โพรโตคอล

ID	Flag	Sequence Number	Data

รูปที่ 4.6 รูปแบบของเฟรมในระดับ CAN โพรโตคอล

รูปแบบของเฟรมในระดับของ CAN โพรโตคอลนั้นได้เพิ่มเติม Sequence Number เข้ามาเพื่อช่วยบอกลำดับของเฟรม อีกทั้งยังใช้ในเฟรมตอบกลับเพื่อยืนยันว่าข้อมูลได้ถูกส่งถึงปลายทางแล้ว

4.2.4 รูปแบบการกำหนด Message Identifier

10	9	8	7	6	5	4	3	2	1	0	Identity Usage
0	N							M	Frame type		Group 1
1	N							M	Frame type		Group 2

รูปที่ 4.7 การกำหนด Message Identifier

จากรูปที่ 4.7 แสดงถึงการกำหนด Message Identifier สามารถแบ่งออกได้ 2 กลุ่มได้แก่

4.2.4.1 กลุ่ม 1

กลุ่ม 1 เป็น ID ที่ใช้ในการส่งข้อมูลจากโหนดลูกบนเครือข่ายไปยังโหนดหลัก ประกอบไปด้วยฟิลด์ย่อยๆดังนี้

4.2.4.1.1 N (NODE Description) : เป็นฟิลด์ที่ใช้ในการระบุโหนดหรือกลุ่มของโหนด

4.2.4.1.2 M (More Fragment) : เป็นฟิลด์ที่ใช้ในการระบุลำดับเฟรม หากเป็น '1' หมาย ความว่ายังมีเฟรมอื่นตามมาอีก แต่ถ้าเป็น '0' จะหมายถึงเป็นเฟรมสุดท้าย

4.2.4.1.3 Frame type : ใช้ในการระบุชนิดของเฟรมในระดับบน โดยมีความหมายดังนี้

ประเภทเฟรม	ความหมาย
0X	การส่งข้อมูล
1X	เป็นการรายงานสถานะ

ตารางที่ 4.5 ความหมายของเฟรม

4.2.4.1.4 Data : ข้อมูลที่ใช้ในการส่งมีขนาดตั้งแต่ 0 – 8 ไบต์

4.2.4.2 กลุ่ม 2

กลุ่ม 2 เป็น ID ที่ใช้ในการส่งข้อมูลจากโหนดหลักบนเครือข่ายไปยังโหนดลูกบนเครือข่าย ประกอบไปด้วยฟิลด์ย่อยๆดังนี้

4.2.4.2.1 N (NODE Description) : เป็นฟิลด์ที่ใช้ในการระบุโหนดหรือกลุ่มของโหนด

4.2.4.2.2 M (More Fragment) : เป็นฟิลด์ที่ใช้ในการระบุว่าลำดับเฟรมหากเป็น '1' หมายความว่ายังมีเฟรมอื่นตามมาอีก แต่ถ้าเป็น '0' จะหมายถึงเป็นเฟรมสุดท้าย

4.2.4.2.3 Frame type : ใช้ในการระบุชนิดของเฟรมในระดับบน โดยมีความหมายดังตารางที่ 4.5

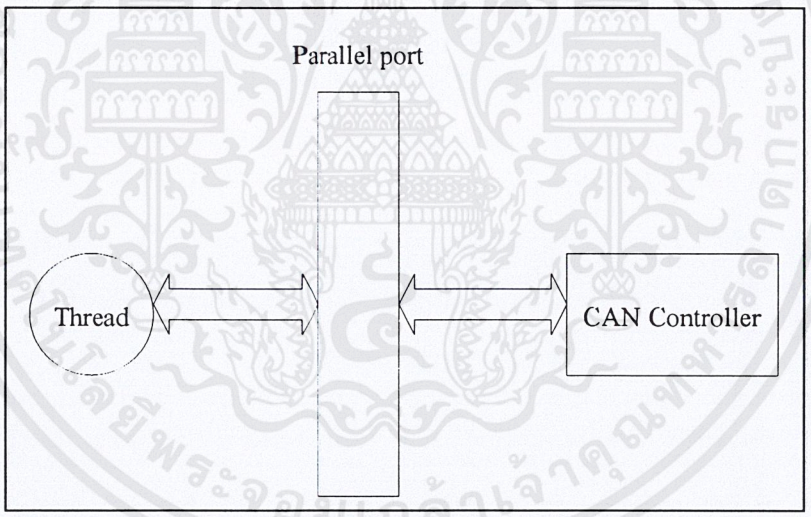
4.2.4.2.4 Data : ข้อมูลที่ใช้ในการส่งมีขนาดตั้งแต่ 0 – 8 ไบต์

4.4 การพัฒนาโปรแกรมของระบบเครือข่าย

การพัฒนาโปรแกรมของระบบเครือข่ายที่สามารถสนับสนุนโพรโตคอล สามารถแบ่งออกเป็น 2 ส่วนหลักๆ คือการพัฒนาโปรแกรมบนคอมพิวเตอร์ส่วนบุคคลและการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์ ในการพัฒนาโพรโตคอลบนคอมพิวเตอร์ส่วนบุคคลนั้นใช้ภาษาจาวาในการอิมพลีเมนต์ตัวโพรโตคอลขึ้นมาเนื่องจากเป็นภาษาที่มีความสามารถในการทำเธรดได้ดี ส่วนการพัฒนาโพรโตคอลบนไมโครคอนโทรลเลอร์นั้นได้ใช้ภาษาซีในการพัฒนาซึ่งช่วยทำให้การพัฒนาง่ายกว่าการใช้ภาษาแอสเซมบลีอีกทั้งช่วยย่นเวลาในการพัฒนาด้วย

4.4.1 การพัฒนาโปรแกรมบนคอมพิวเตอร์ส่วนบุคคล

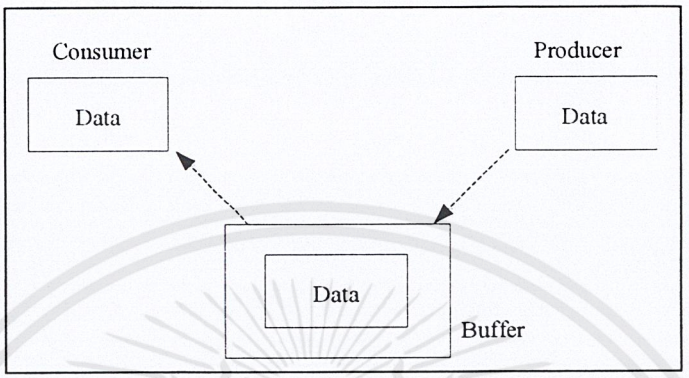
การเชื่อมต่อระหว่างคอมพิวเตอร์ส่วนบุคคลกับตัวควบคุม CAN นั้นเชื่อมต่อผ่านพอร์ตขนาน เนื่องจากภายในพอร์ตขนานเองนั้นไม่มีอินเทอร์รัปต์ให้ใช้งาน ดังนั้นในส่วนของโปรแกรมจึงต้องอาศัยการทำงานของเธรดในการวนตรวจสอบค่าแฟล็กต่างๆที่อยู่ในตัวควบคุม CAN โดยที่จังหวะในการวนตรวจสอบนั้นจะต้องเร็วเพียงพอต่อปริมาณข้อมูลที่จะรับเข้ามารวมไปถึงต้องมีความสม่ำเสมอในการวนตรวจสอบด้วย



รูปที่ 4.8 รูปแบบการทำงานของโปรแกรมบนคอมพิวเตอร์ส่วนบุคคล

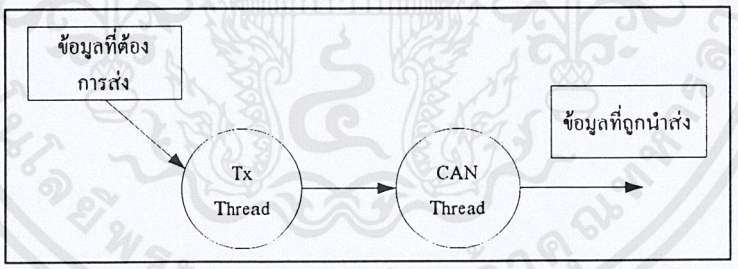
เนื่องจากการทำงานของโปรแกรมบนคอมพิวเตอร์ส่วนบุคคลนั้นใช้เธรดในการควบคุมการทำงาน เพราะฉะนั้นการควบคุมการทำงานของเธรดจึงเป็นกลไกหลักที่จะดำเนินการทำงานของโปรแกรม ประกอบกับจาวาเองสนับสนุนการซิงโครไนซ์เธรด หมายความว่าใช้กลไกสำหรับการควบคุมให้ที่เวลาหนึ่งทรัพยากรนั้นถูกใช้งานได้โดยเธรดเดียวเท่านั้น โดยนำกลไกของ Mutually Exclusive ของระบบปฏิบัติการมาใช้ นั่นคือ หากนำ Mutually Exclusive ไปครอบครองทรัพยากรใดจะจำกัดให้ที่เวลาหนึ่งมีเธรดเดียวเท่านั้นที่สามารถเข้าครอบครองและใช้งานทรัพยากรนั้น ระหว่างที่มีเธรดหนึ่งครอบครองอยู่ หากมีเธรดอื่นต้องการใช้งานทรัพยากรนั้นก็จะต้องหยุดรอจนกว่าเธรดที่ใช้งานอยู่จะยอมปล่อยทรัพยากรนั้นออกจากการครอบครอง อีกทั้งยังใช้หลักการผู้ผลิต (Producer)

กับผู้บริโภค (Consumer) คือมีอย่างน้อย 2 เธรดทำงานร่วมกันโดยเธรดหนึ่งเป็นผู้ผลิตซึ่งจะให้ผลลัพธ์ออกมาทีละตัวโดยเขียนค่าไปที่ตัวแปรที่เป็นบัฟเฟอร์ตัวหนึ่ง อีกเธรดหนึ่งผู้บริโภคซึ่งจะอ่านค่าจากบัฟเฟอร์แต่ละตัว โดยผู้ผลิตจะต้องเขียนค่าก่อนที่ผู้บริโภคจะอ่านค่า และผู้ผลิตต้องรอจนกว่าผู้บริโภคอ่านค่าเดิมไปใช้งานจึงจะเขียนค่าใหม่ได้ เป็นเงื่อนไขในการควบคุมการทำงานของเธรด

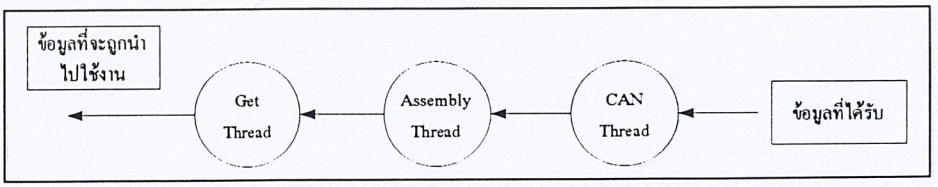


รูปที่ 4.9 แสดงกลไกการทำงานในลักษณะ Producer & Consumer

เมื่อการทำงานของโปรแกรมถูกควบคุมโดยเธรดดังนั้นการทำงานในแต่ละหน้าที่จะถูกแบ่งแยกกันอย่างชัดเจน ผลจากการทำงานของเธรดหนึ่งอาจจะมีผลกระทบกับอีกเธรดหนึ่ง และเพื่อให้เข้าใจลำดับขั้นตอนการทำงานของการรับและส่งข้อมูล จึงได้แสดงขั้นตอนการทำงานดังรูปที่ 4.10 และ 4.11



รูปที่ 4.10 ลำดับการทำงานในการส่งข้อมูล



รูปที่ 4.11 ลำดับการทำงานในการส่งข้อมูล

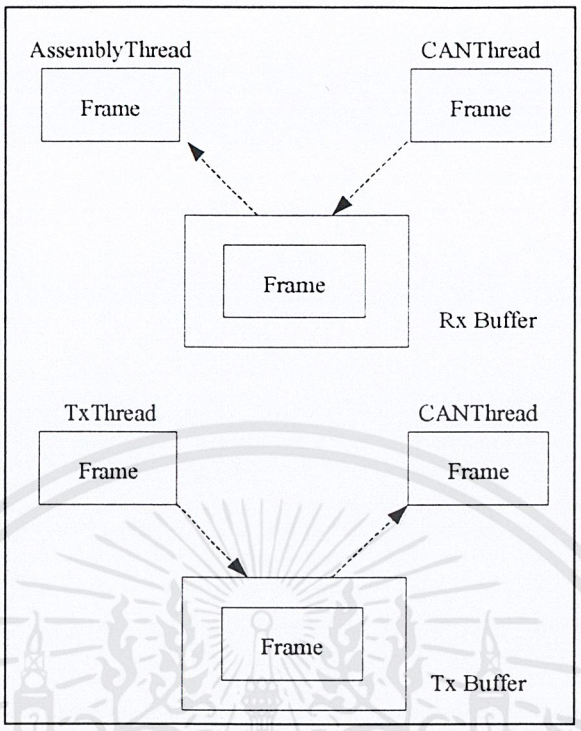
สามารถอธิบายการทำงานของเธรดได้ดังนี้

4.4.1.1 CANThread

CANThread เป็นเธรดที่ทำหน้าที่ในการเชื่อมต่อกับตัวควบคุม CAN รับผิดชอบการรับและส่งข้อมูลผ่าน CAN มีการทำงานเป็นอิสระและมีค่าลำดับความสำคัญสูงสุด (Priority) เนื่องจากต้องวนตรวจสอบข้อมูลอยู่ในบัฟเฟอร์ขาส่งและตรวจสอบสถานะแฟล็กต่างๆจากตัวควบคุม CAN ประกอบไปด้วยบัฟเฟอร์ที่สำคัญอยู่ 2 ชุดได้แก่

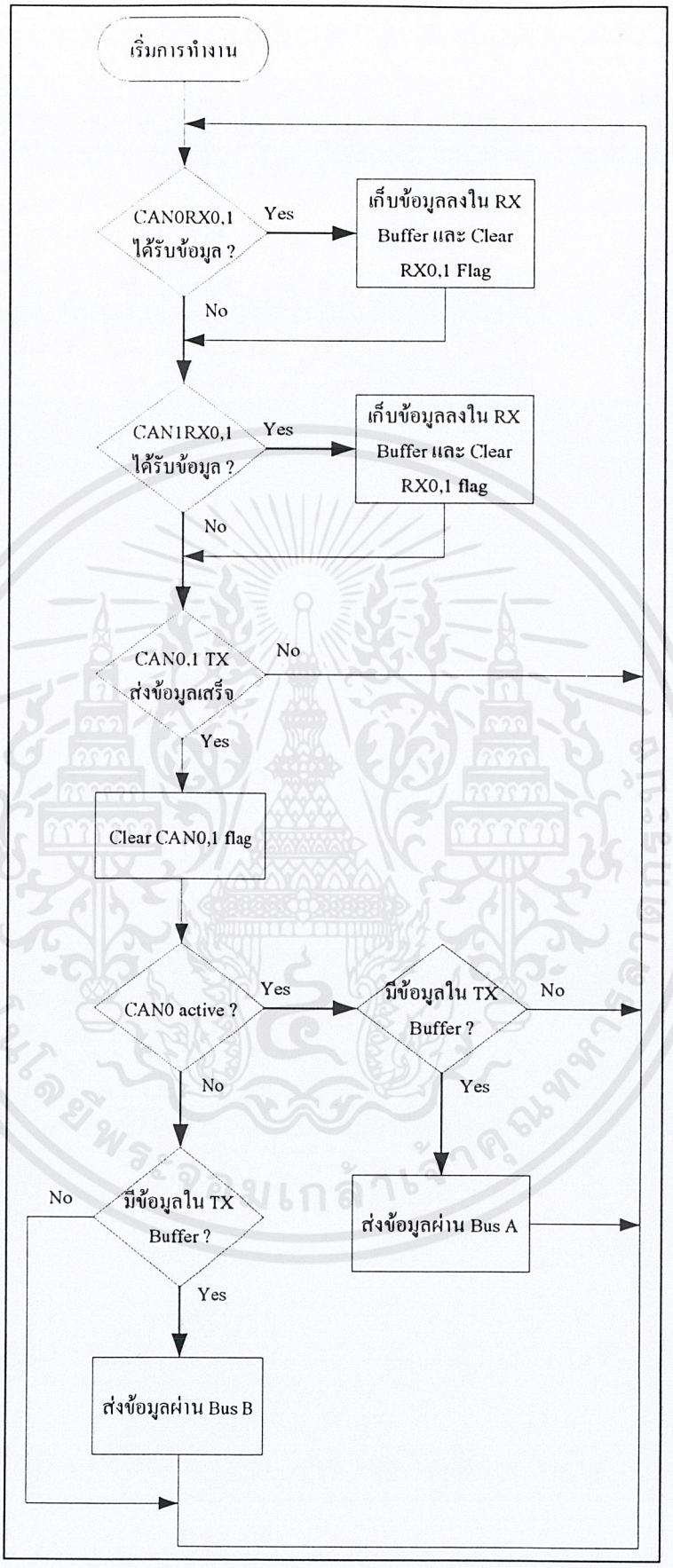
4.4.1.1.1 ***Rx Buffer*** เป็นแชร์บัฟเฟอร์ขนาด 8 ไบต์ ทำหน้าที่ในการเก็บเฟรมข้อมูลจากเครือข่าย ใช้การซิงโครไนซ์เธรดในการควบคุมการเขียนและอ่านค่าบัฟเฟอร์ Rx การเขียนบัฟเฟอร์ Rx นั้นรับผิดชอบโดย CANThread ส่วนการนำข้อมูลไปใช้งานจะรับผิดชอบโดย AssemblyThread ซึ่งจะกล่าวถึงในส่วนถัดไป

4.4.1.1.2 ***Tx buffer*** เป็นแชร์บัฟเฟอร์ขนาด 8 ไบต์ ทำหน้าที่เก็บเฟรมข้อมูลที่จะใช้ส่งไปบนเครือข่าย ใช้การซิงโครไนซ์เธรดในการควบคุมการเขียนและอ่านค่าบัฟเฟอร์ Tx การเขียนบัฟเฟอร์ Tx นั้นรับผิดชอบโดย TxThread ส่วนการนำข้อมูลส่งไปยังตัวควบคุม CAN จะรับผิดชอบโดย CANThread



รูปที่ 4.12 รูปแบบการทำงานของ CANThread

การทำงานของ CANThread จะเริ่มจากการตรวจสอบแฟล็ก RX0IF (Receive Buffer 0 Interrupt Flag) และ RX1IF (Receive Buffer 1 Interrupt Flag) ถ้ามีสถานะเป็น '1' หมายความว่าได้รับข้อมูล จากนั้นจะดำเนินการอ่านค่า ID และข้อมูลจากตัวควบคุม CAN มาเก็บไว้ที่บัฟเฟอร์ Rx เพื่อให้ AssemblyThread นำข้อมูลไปประกอบเป็นแพ็กเก็ตแล้วทำการเคลียร์แฟล็ก RX0IF หรือ RX1IF เพื่อให้บัฟเฟอร์สามารถรับข้อมูลใหม่ได้ จากนั้นจะทำการตรวจสอบก่อนว่ามีข้อมูลในบัฟเฟอร์ Tx หรือไม่ ถ้ามีข้อมูลอยู่ก็จะดำเนินการส่งข้อมูลนั้นไปยังตัวควบคุม CAN เพื่อดำเนินการส่งไปบนเครือข่ายโดยการเซตบิต TXB0RTS (Request To Send) แต่ถ้าไม่มีข้อมูลในบัฟเฟอร์ Tx ก็จะข้ามไปทำงานอย่างอื่น การเขียนบัฟเฟอร์ Tx นั้นจะรับผิดชอบโดย TxThread ส่วนการนำส่งข้อมูลรับผิดชอบโดย CAN Thread สามารถแสดงการทำงานได้ด้วยโฟลว์ชาร์ทดังรูปที่ 4.13



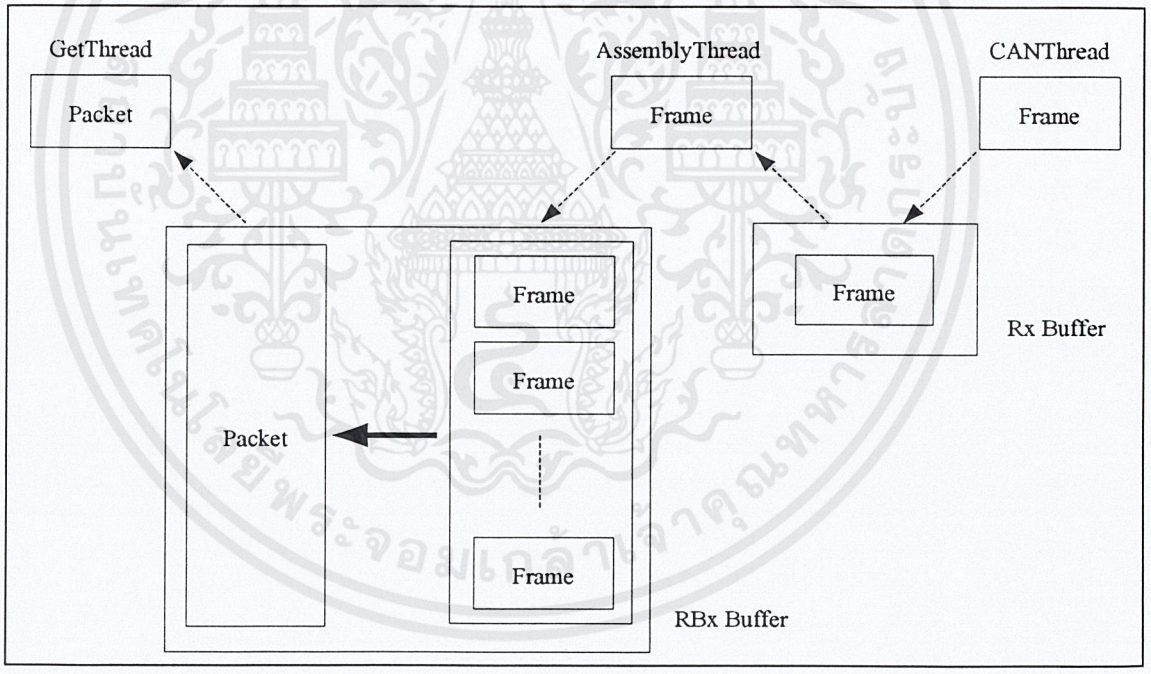
รูปที่ 4.13 แสดงการทำงานของ CANThread

4.4.1.2 Assembly Thread

ทำหน้าที่การประกอบเฟรมข้อมูลกลับเป็นแพ็กเก็ตประกอบไปด้วยบัฟเฟอร์ที่สำคัญ 2 ชุด ได้แก่

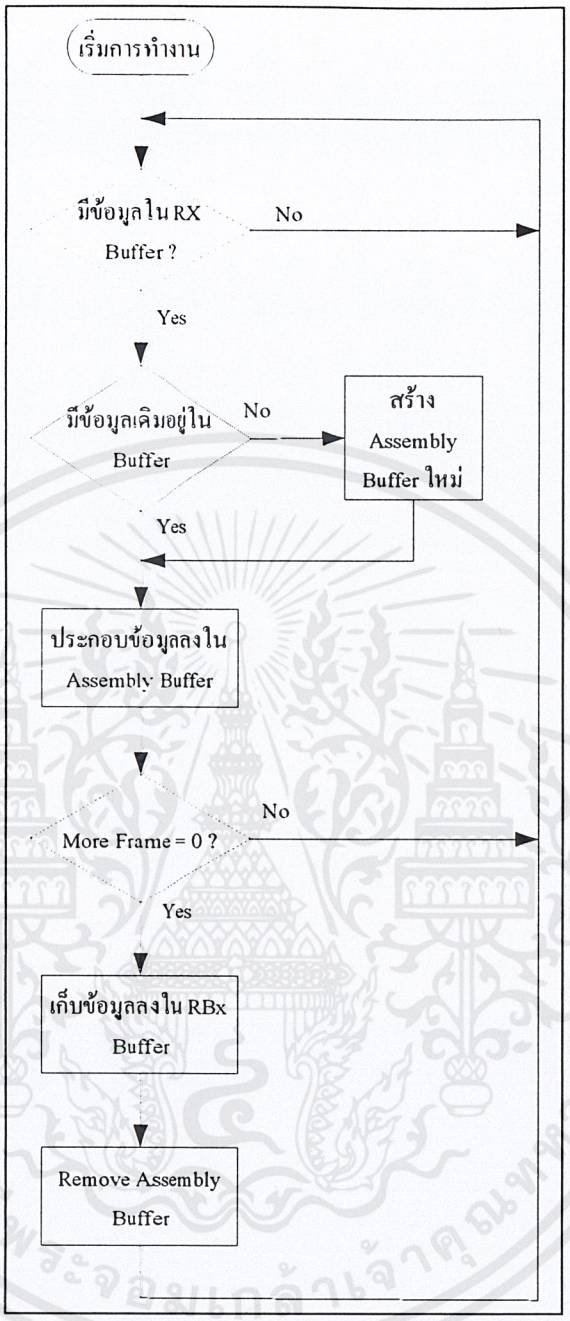
4.4.1.2.1 ***Rx Buffer*** เป็นแชร์บัฟเฟอร์ขนาด 8 ไบต์ ทำหน้าที่ในการเก็บข้อมูลจากเครือข่าย โดยใช้ ซิงโครไนซ์เธรดในการควบคุมการเขียนและอ่านค่าจากบัฟเฟอร์ Rx การเขียนบัฟเฟอร์ Rx นั้นรับผิดชอบโดย CANThread ส่วนการนำข้อมูลไปใช้งานจะรับผิดชอบโดย AssemblyThread

4.4.1.2.2 ***RBx High layer buffer*** เป็นแชร์บัฟเฟอร์ขนาด 256 ไบต์ ทำหน้าที่ในการเก็บข้อมูลที่ประกอบเป็นแพ็กเก็ตแล้ว โดยใช้ซิงโครไนซ์เธรดในการควบคุมการเขียนและอ่านค่าจากบัฟเฟอร์ RBx High layer การเขียนบัฟเฟอร์ RBx High layer นั้นรับผิดชอบโดย AssemblyThread ส่วนการนำข้อมูลไปใช้งานจะรับผิดชอบโดย GetThread ซึ่งจะกล่าวถึงในส่วนถัดไป



รูปที่ 4.14 รูปแบบการทำงานของ AssemblyThread

หน้าที่ของ AssemblyThread คือการประกอบเฟรมข้อมูลกลับเป็นแพ็กเก็ตโดยอาศัย More frame flag เป็นเงื่อนไขในการประกอบข้อมูล ถ้า More frame flag เป็น '1' AssemblyThread จะนำข้อมูลไปเก็บไว้ในบัฟเฟอร์เพื่อรอประกอบเฟรมถัดไป แต่ถ้าเป็น More frame flag '0' จะประกอบข้อมูลลงไปในบัฟเฟอร์แล้วดำเนินการก๊อบปี้ข้อมูลชุดนั้นไปเก็บไว้ในบัฟเฟอร์ RBx High layer เพื่อให้ GetThread นำไปประมวลผลต่อไป



รูปที่ 4.15 แสดงการทำงานของ AssemblyThread

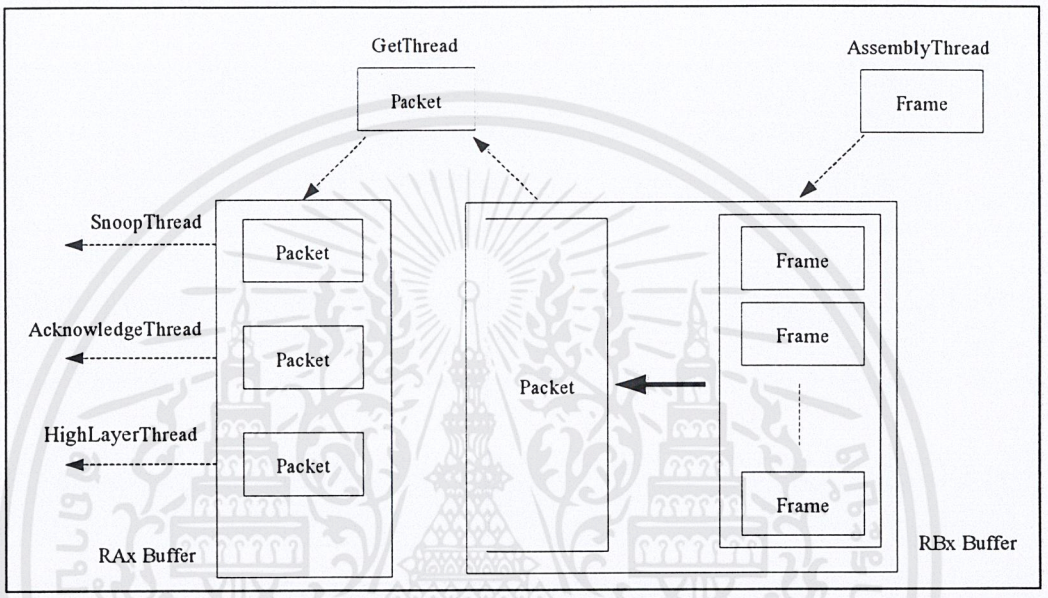
4.4.1.3 GetThread

ทำหน้าที่ในการนำข้อมูลนั้นส่งไปให้แอพพลิเคชันระดับบนใช้งาน ประกอบไปด้วยบัฟเฟอร์ที่สำคัญ 2 ชุดได้แก่

4.4.1.3.1 ***RBx High layer buffer*** เป็นแชร์บัฟเฟอร์ขนาด 256 ไบต์ ทำหน้าที่ในการเก็บข้อมูลที่ประกอบเป็นแพ็กเก็ตเรียบร้อยแล้ว โดยใช้การซิงโครไนซ์เธรดในการควบคุมการเขียนและอ่านค่าจากบัฟเฟอร์ RBx High layer การเขียนบัฟเฟอร์

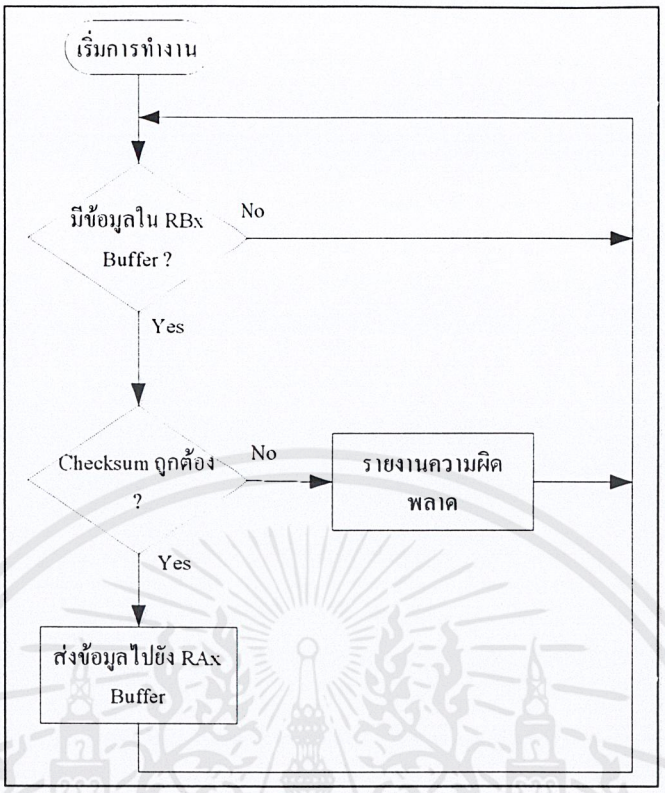
RBx High layer นั้นรับผิดชอบโดย AssemblyThread ส่วนการนำข้อมูลไปใช้งานจะรับผิดชอบโดย GetThread

4.4.1.3.2 ***RAx High layer buffer*** เป็น บัฟเฟอร์ขนาด 256 ไบต์ ทำหน้าที่เก็บข้อมูลในระดับแพ็กเก็ตเพื่อที่จะนำไปใช้งานประกอบไปด้วย SnoopThread, AcknowledgeThread และ HighLayerThread โดยอาศัยกลไกซิงโครไนซ์ทั้งหมด



รูปที่ 4.16 รูปแบบการทำงานของ GetThread

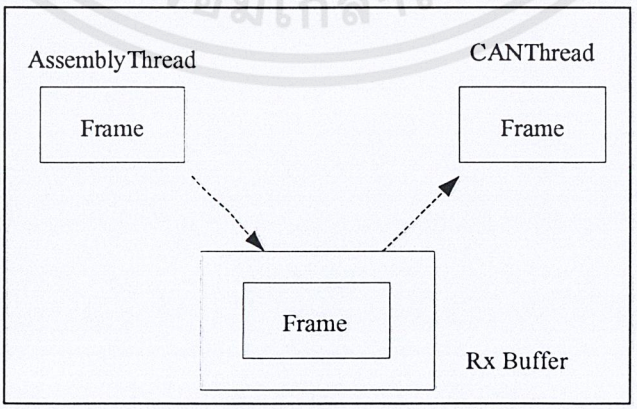
หน้าที่ของ GetThread คือนำส่งแพ็กเก็ตข้อมูลไปให้แอพพลิเคชันระดับบนได้แก่ การดักจับแพ็กเก็ต (SnoopThread), การตอบกลับ (AcknowledgeThread) และประมวลผลเฟรมรายงานสถานะ (HighLayerThread) อีกทั้งยังทำหน้าที่ในการดักจับเฟรมตอบกลับจากโหนดต่างๆ แล้วดำเนินการในการเคลียร์ข้อมูลใน Timeout Thread และยังทำหน้าที่ในการตรวจสอบผลรวมผ่านฟังก์ชัน TryChecksum



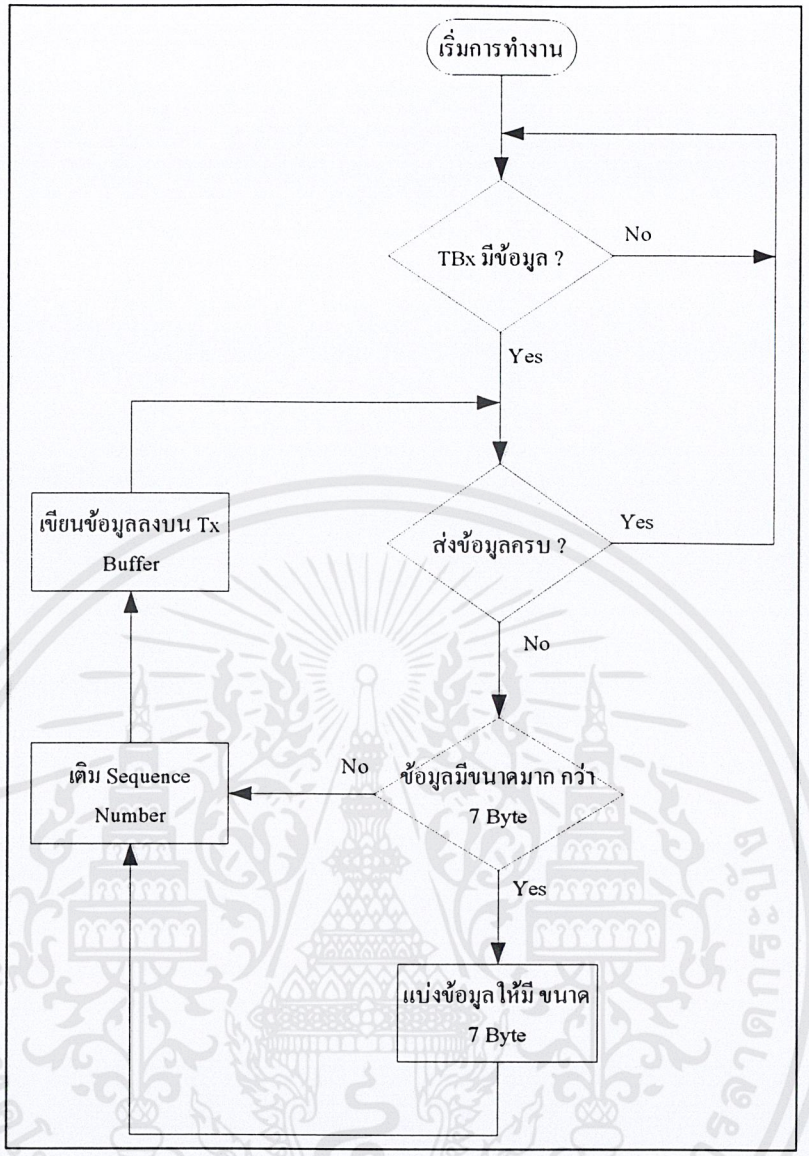
รูปที่ 4.17 แสดงการทำงานของ GetThread

4.4.1.4 TxThread

ทำหน้าที่ในการแบ่งแพ็กเก็ตออกเป็นเฟรมขนาดไม่เกิน 8 ไบต์ เพื่อให้สามารถส่งไปบน CAN โพรโตคอลได้แอพพลิเคชันระดับบนใดๆก็ตามที่ต้องการจะส่งข้อมูล จะต้องดำเนินการผ่าน TxThread นี้ โดย TxThread จะอัพเดตข้อมูลลงบนบัฟเฟอร์ Tx และข้อมูลจะถูกนำส่งโดย CANThread อีกทั้งยังทำหน้าที่ในการสร้าง Sequence Number ส่งไปให้กับ TimeoutThread เพื่อใช้ในการตรวจสอบการตอบกลับของโหนดบนเครือข่าย



รูปที่ 4.18 รูปแบบการทำงานของ TxThread



รูปที่ 4.19 แสดงการทำงานของ TxThread

นอกจากนี้แล้วยังมี Class และไฟล์อื่นๆ ได้แก่

4.4.1.5 AcknowledgeThread เป็นเธรดที่ทำหน้าที่ในการสร้างเฟรมตอบกลับเพื่อใช้ในการยืนยันการรับข้อมูลกลับไปยังโหนดที่ส่งข้อมูลมา

4.4.1.6 AssemblyBuffer เป็นบัฟเฟอร์ขนาด 256 ไบต์ ใช้สำหรับประกอบข้อมูล

4.4.1.7 AssemblyTimeoutManage เป็นคลาสที่ทำหน้าที่ในการจัดการดูแลการสร้างและทำลายไทม์เมอร์ที่ใช้จับเวลาในการประกอบข้อมูล

4.4.1.8 AssemblyTimeoutThread เป็นเธรดทำหน้าที่เป็นไทม์เมอร์คอยรายงานข้อผิดพลาดจากการประกอบแพ็กเก็ตที่ใช้เวลานานเกินกำหนด ซึ่งจะถูกสร้างขึ้นโดยคลาส AssemblyTimeoutManage

4.4.1.8 CANBuffer เป็นบัฟเฟอร์ขนาด 8 ไบต์ ใช้เป็นอาร์กิวเมนต์ในการรับส่งข้อมูลผ่านฟังก์ชันที่มีโมดิฟายเป็นซิงโครไนซ์

4.4.1.9 CANException เป็นคลาสที่ใช้ในการตรวจจับข้อผิดพลาดของโปรแกรม

4.4.1.10 CANShareBuffer เป็นบัฟเฟอร์ขนาด 8 ไบต์ ใช้ในการซิงโครไนซ์ระหว่างเธรดและรับส่งข้อมูลผ่าน CAN โพรโตคอล

4.4.1.11 DataBaseHandle เป็นคลาสที่ให้บริการในการเก็บและเรียกข้อมูลจากดาต้าเบส

4.4.1.12 HighLayerBuffer เป็นบัฟเฟอร์ขนาด 256 ไบต์ ใช้สำหรับเป็นอาร์กิวเมนต์ในการรับส่งข้อมูลผ่านฟังก์ชันที่มีโมดิฟายเป็นซิงโครไนซ์

4.4.1.13 HighLayerShareBuffer เป็นบัฟเฟอร์ขนาด 256 ไบต์ ที่ใช้ในการซิงโครไนซ์ระหว่างเธรดและรับส่งข้อมูลในระดับบน

4.4.1.14 HighLayerThread เป็นเธรดที่รับผิดชอบการประมวลผลในกรณีที่แพ็กเก็ตเป็นการรายงานสถานะ

4.4.1.15 MainFrame เป็นจุดเริ่มต้นของโปรแกรม ประกอบไปด้วยตัวออบเจ็กต์ต่างๆที่จำเป็นต้องอ้างถึงจากหลายๆคลาสได้แก่ RBx, TBx, RAx , Rx, Tx และเป็นส่วนที่ใช้สร้างและรันเธรดทั้งหมด

4.4.1.16 MCP2510 รับผิดชอบการให้บริการการเชื่อมต่อกับตัวควบคุม CAN รวมไปถึงการ Initial ตัวควบคุม CAN ซึ่งจะถูกดำเนินการทันทีที่สร้างอินสแทนซ์ขึ้นมา

4.4.1.17 RequestStatusThread เป็นเธรดที่รันโดยอิสระ ซึ่งจะทำงานตามเวลาที่ได้ระบุไว้ใน Time Request ทำหน้าที่ในการร้องขอสร้างเฟรมเพื่อร้องขอสถานะจากโหนดต่างๆบนเครือข่าย

4.4.1.18 SendStatus เป็นคลาสทำหน้าที่ในการสร้างเฟรมสำหรับรายงานสถานะ

4.3.1.19 SequenceCounter เป็นคลาสที่ทำหน้าที่สร้าง Sequence Number

4.4.1.20 SnoopThread เป็นเธรดที่ทำหน้าที่ในการดักจับแพ็กเก็ตมาแสดงผล

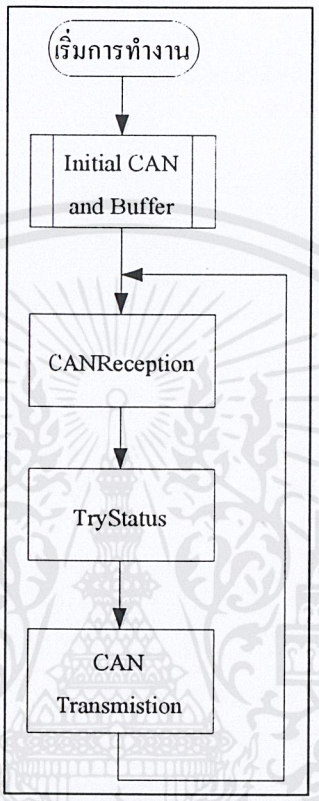
4.4.1.21 TimeoutManage เป็นคลาสที่ทำหน้าที่ในการจัดการดูแลการสร้างและทำลายไทม์เมอร์ที่ใช้สำหรับการส่งข้อมูล

4.4.1.22 TimeoutThread เป็นเธรดทำหน้าที่เป็นไทม์เมอร์คอยรายงานข้อผิดพลาดจากการรอคอยการส่งข้อมูลที่ใช้เวลานานเกินกำหนดซึ่งจะถูกสร้างขึ้นโดยคลาส TimeoutManage

4.4.1.23 TimeThread เป็นเธรดซึ่งทำหน้าที่ในการแสดงเวลาปัจจุบันบนแท็ป Status

4.4.2 การพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์

การพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์นี้จะมีลักษณะการทำงานเป็นลำดับขั้นตอนที่ตายตัวและทำงานวนลูปซ้ำไปเรื่อยๆ โดยไม่ใช้คุณสมบัติในการอินเทอร์รัปต์สามารถแสดงการทำงานได้ดังรูปที่ 4.20

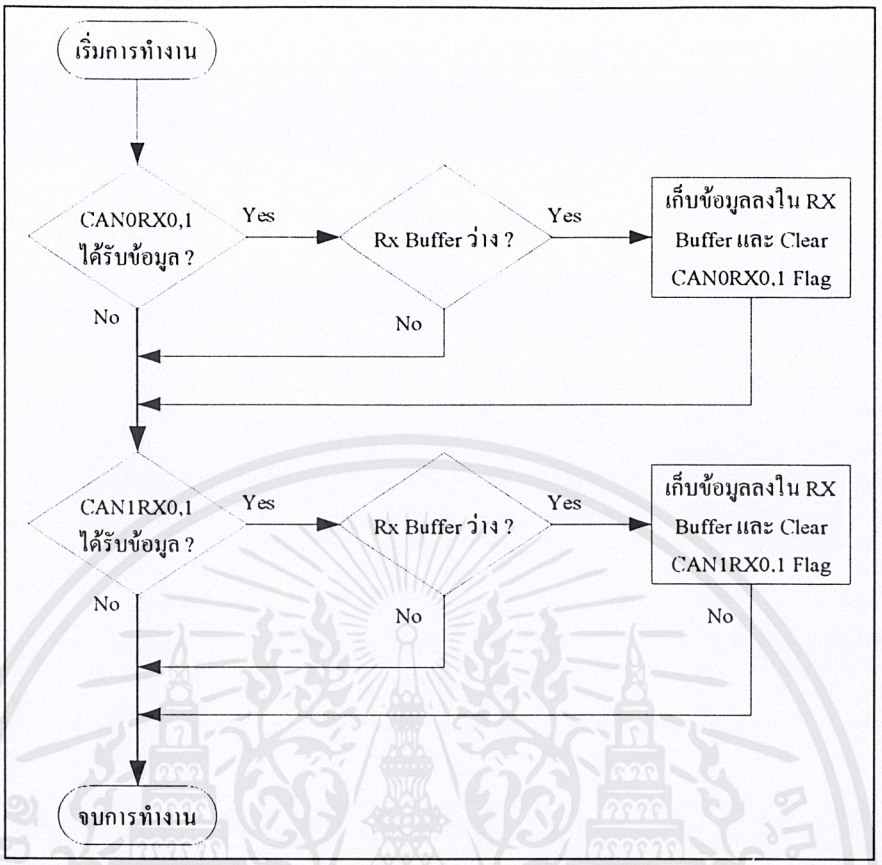


รูปที่ 4.20 แสดงการทำงานหลักของไมโครคอนโทรลเลอร์

การทำงานหลักของโปรแกรมบนไมโครคอนโทรลเลอร์ประกอบไปด้วยการ Initial ตัวควบคุม CAN, การตรวจสอบข้อมูลที่ได้รับ (CANReception) , การตรวจสอบความถูกต้องและแสดงสถานะ (Status) และการนำส่งข้อมูล (CANTransmission)

4.4.2.1 ฟังก์ชัน CANReception

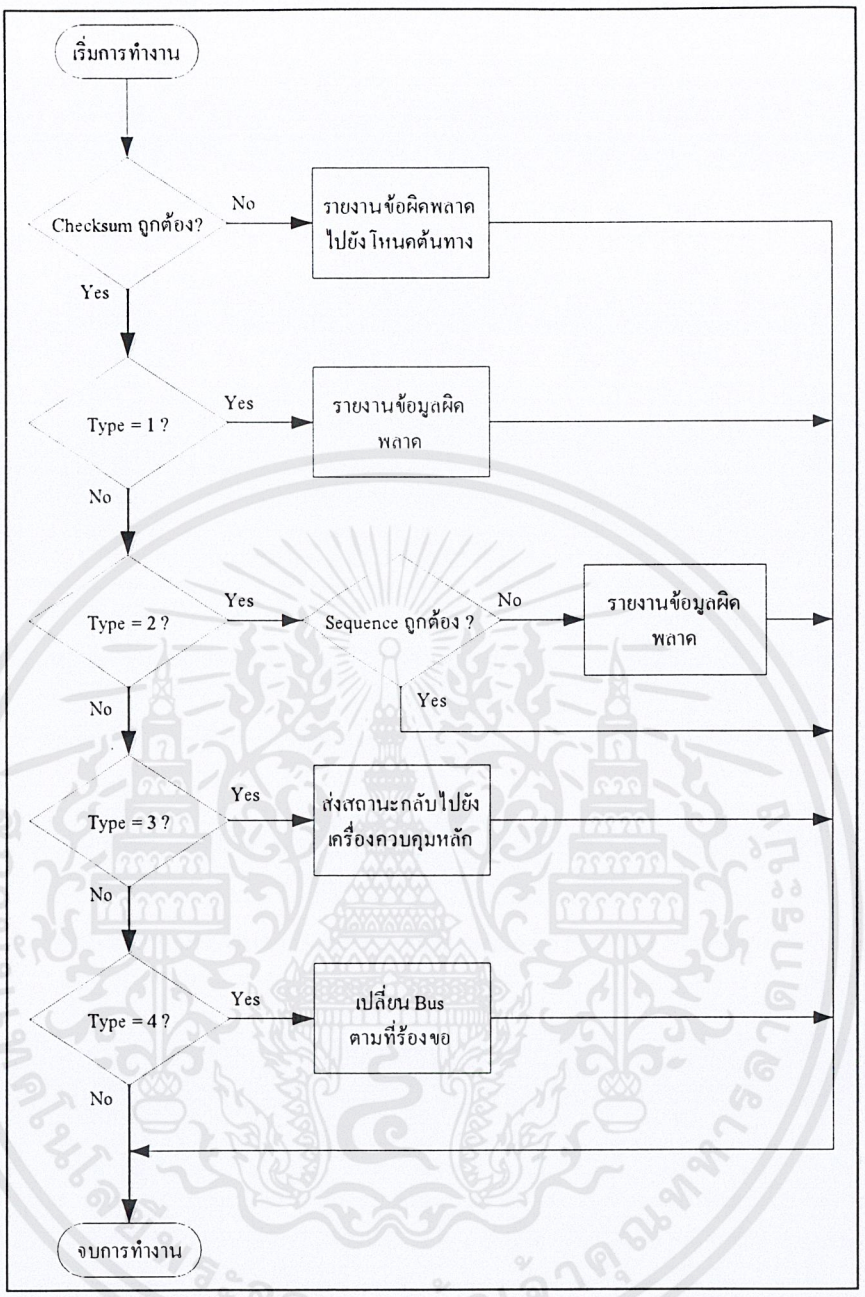
ฟังก์ชัน CANReception ทำหน้าที่หลักในการรับข้อมูลจากเครือข่ายโดยจะทำการตรวจสอบแฟล็ก RX0IF (Receive Buffer 0 Interrupt Flag) และ RX1IF (Receive Buffer 1 Interrupt Flag) ถ้ามีสถานะเป็น '1' หมายความว่ามีข้อมูลอยู่ในบัฟเฟอร์จากนั้นจะดำเนินดำเนินการอ่านค่า ID และข้อมูลจากตัวควบคุม CAN มาเก็บไว้ที่บัฟเฟอร์ Rx เพื่อนำข้อมูลไปใช้งานแล้วทำการเคลียร์แฟล็ก RX0IF หรือ RX1IF เพื่อให้บัฟเฟอร์สามารถรับข้อมูลใหม่ได้



รูปที่ 4.21 แสดงการทำงานของฟังก์ชัน CANReception

4.4.2.2 ฟังก์ชัน TryStatus

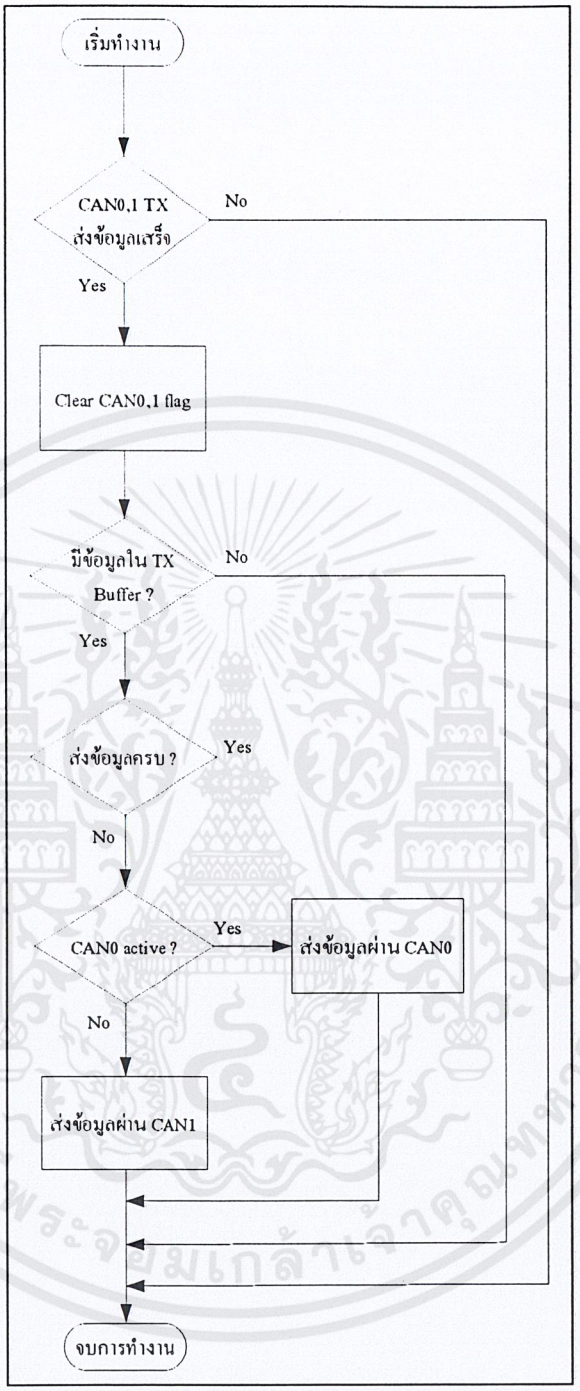
ฟังก์ชัน TryStatus ทำหน้าที่ในการตรวจสอบและแสดงผลในกรณีที่แพ็กเก็ตที่รับเข้ามาเป็นแพ็กเก็ตที่ใช้ในการรายงานสถานะ โดยจะดำเนินการตรวจสอบตามรูปแบบที่กำหนดเอาไว้แล้วทำการแสดงผลหรือตอบกลับไปยังโหนดที่ส่งข้อมูลมา



รูปที่ 4.22 แสดงการทำงานของฟังก์ชัน TryStatus

4.4.2.3 ฟังชัน CANTransmission

ฟังก์ชัน CANTransmission ทำหน้าที่ในการนำส่งข้อมูลไปบนเครือข่าย โดยการทำงานจะเริ่มจากการตรวจสอบบัฟเฟอร์ Tx ก่อนว่ามีข้อมูลที่ต้องการจะส่งหรือไม่ ถ้าบัฟเฟอร์ Tx มีข้อมูลที่ต้องการจะส่งก็แบ่งข้อมูลให้มีขนาดไม่เกิน 8 ไบต์ เพื่อให้สามารถส่งไปบน CAN โพรโตคอลได้ จากนั้นจะทำการตรวจสอบ TX0,1IF แฟล็กที่ใช้ในการแสดงว่าข้อมูลได้ถูกนำส่งเรียบร้อยแล้ว จากนั้นจะทำการ เคลียร์แฟล็ก TX0,1IF เพื่อที่จะใช้ในการส่งข้อมูลถัดไป



รูปที่ 4.23 แสดงการทำงานของฟังก์ชัน CAN Transmission

4.5 การออกแบบฮาร์ดแวร์ของระบบเครือข่าย

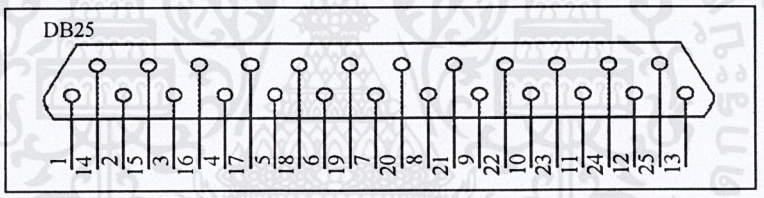
การจำลองเครือข่ายที่ได้ทำการออกแบบสามารถแบ่งออกเป็น 2 ส่วนได้แก่ การจำลองการทำงานบนคอมพิวเตอร์ส่วนบุคคลและการจำลองการทำงานบนไมโครคอนโทรลเลอร์ เหตุผลที่ต้องมีการจำลองการทำงานบนคอมพิวเตอร์ส่วนบุคคล เนื่องมาจากสามารถแสดงผลการทำงานได้ดีกว่าเมื่อเทียบกับการแสดงผลบนไมโครคอนโทรลเลอร์ แต่ทั้งนี้ทั้งนั้นโพรโตคอลถูกออกแบบสำหรับ

อุปกรณ์ประมวณผลขนาดเล็กอย่างไมโครคอนโทรลเลอร์ ดังนั้นจึงต้องติดตั้งโพรโตคอลลงบนไมโครคอนโทรลเลอร์เพื่อที่จะแสดงให้เห็นว่าสามารถใช้งานได้จริง โดยทั้ง 2 ส่วนต้องสามารถสื่อสารหรือส่งถ่ายข้อมูลได้โดยใช้โพรโตคอลที่ได้ออกแบบ

4.5.1 การออกแบบฮาร์ดแวร์เชื่อมต่อกับคอมพิวเตอร์ส่วนบุคคล

ในส่วนของการออกแบบฮาร์ดแวร์เชื่อมต่อผ่านคอมพิวเตอร์ส่วนบุคคลนั้นเนื่องจากว่าการเชื่อมต่อ MCP2510 นั้นสามารถเชื่อมต่อผ่าน Serial Peripheral Interface (SPI) หรือการเชื่อมต่ออนุกรมแบบซิงโครนัส ดังนั้นจึงไม่สามารถเชื่อมต่อผ่านพอร์ตอนุกรมได้เนื่องจากพอร์ตอนุกรมของคอมพิวเตอร์ส่วนบุคคลมีลักษณะการทำงานแบบอะซิงโครนัส ดังนั้นการเชื่อมต่อระหว่างคอมพิวเตอร์ส่วนบุคคลกับ MCP2510 จึงต้องเชื่อมต่อผ่านพอร์ตขนานโดยใช้ขาของพอร์ตขนานเป็นทั้งขาของข้อมูลและขาคล็อก โดยตัว MCP2510 นี้สามารถเชื่อมต่อผ่าน Serial Peripheral Interface (SPI) ได้ด้วยความเร็วสูงสุด 5 เมกะบิตต่อวินาที

4.5.1.1 การใช้งานของพอร์ตขนาน



รูปที่ 4.24 ลักษณะของคอนเน็กเตอร์แบบ DB25

พอร์ตขนาน (Parallel Port) มีลักษณะเป็นหัวต่อคอนเน็กเตอร์ชนิดตัวเมีย มีรูขาสัญญาณจำนวน 25 ขาสัญญาณ (Connector DB25 Female Type) ดังรูปที่ 4.24 ประกอบไปด้วยขาสัญญาณเอาท์พุตที่เป็นบิตข้อมูล (Data Bit) จำนวน 8 ขาสัญญาณ ขาสัญญาณเอาท์พุตที่เป็นบิตควบคุม (Control Bit) จำนวน 4 ขาสัญญาณ และขาสัญญาณอินพุตที่เป็นบิตสถานะ (Status Bit) จำนวน 5 ขาสัญญาณ ดังนี้

ขาสัญญาณ	หน้าที่
1	ขา Strobe (Control Bit0)
2	ขา Data 0 (Data bit 0)
3	ขา Data 1 (Data bit 1)
4	ขา Data 2 (Data bit 2)
5	ขา Data 3 (Data bit 3)
6	ขา Data 4 (Data bit 4)
7	ขา Data 5 (Data bit 5)
8	ขา Data 6 (Data bit 6)
9	ขา Data 7 (Data bit 7)
10	ขา ACK (Status bit 6)
11	ขา Busy (Status bit 7)
12	ขา Paper Empty (Status bit 5)
13	ขา Select (Status bit 4)
14	ขา Auto Feed (Control bit 1)
15	ขา Error (Status bit 3)
16	ขา Initial Printer (Control bit 2)
17	ขา Select Input (Control bit 3)
18 - 25	ขา Ground (Ground signal)

ตารางที่ 4.6 หน้าที่ของขาต่างๆบนพอร์ตขนาน

พรินเตอร์พอร์ตถูกกำหนดให้มีพอร์ตสื่อสารตามมาตรฐานของไบออส (BIOS) อยู่ในส่วนโลว์เมมโมรี่ (Low Memory) เริ่มต้นที่ตำแหน่ง 0040 : 0008 โดยระบุตำแหน่งของพรินเตอร์พอร์ตที่คอมพิวเตอร์แต่ละเครื่องสนับสนุนการใช้งาน ประกอบไปด้วยข้อมูลชนิด 16 บิตจำนวน 3 ชุด ใช้ระบุหมายเลขพอร์ต LPT1, LPT2 และ LPT3 พรินเตอร์พอร์ตจะประกอบไปด้วยรีจิสเตอร์ 8 บิต จำนวน 3 ชุด (พอร์ต) ใช้สำหรับสื่อสารกับอุปกรณ์ต่อเชื่อมภายนอก ประกอบไปด้วย

รีจิสเตอร์ชนิดดาต้า (Data Register) มีตำแหน่งพอร์ตอยู่ที่ IOBase + 0 มีความกว้างของข้อมูลเป็นขนาด 1 ไบต์ (8 บิต) สามารถทำการอ่านหรือเขียนข้อมูลได้ (Read/Write Access) โดยข้อมูลแต่ละบิตจะเชื่อมโยงกับขาสัญญาณของคอนเน็กเตอร์ (DB25) ที่ขาสัญญาณ 2-9 คือ บิตที่ 0 เชื่อมโยงกับขาสัญญาณที่ 2, บิตที่ 1 เชื่อมโยงกับขาสัญญาณที่ 3 และบิตที่ 7 เชื่อมโยงกับขาสัญญาณที่ 9 ตามลำดับ

รีจิสเตอร์ชนิดสถานะ (Status Register) มีตำแหน่งพอร์ตอยู่ที่ IOBase + 1 มีความกว้างของข้อมูลเป็นขนาด 1 ไบต์ (ใช้จริง 5 บิต) สามารถทำการอ่านข้อมูล (Read Access Only) จากรีจิสเตอร์นี้ได้อย่างเดียว

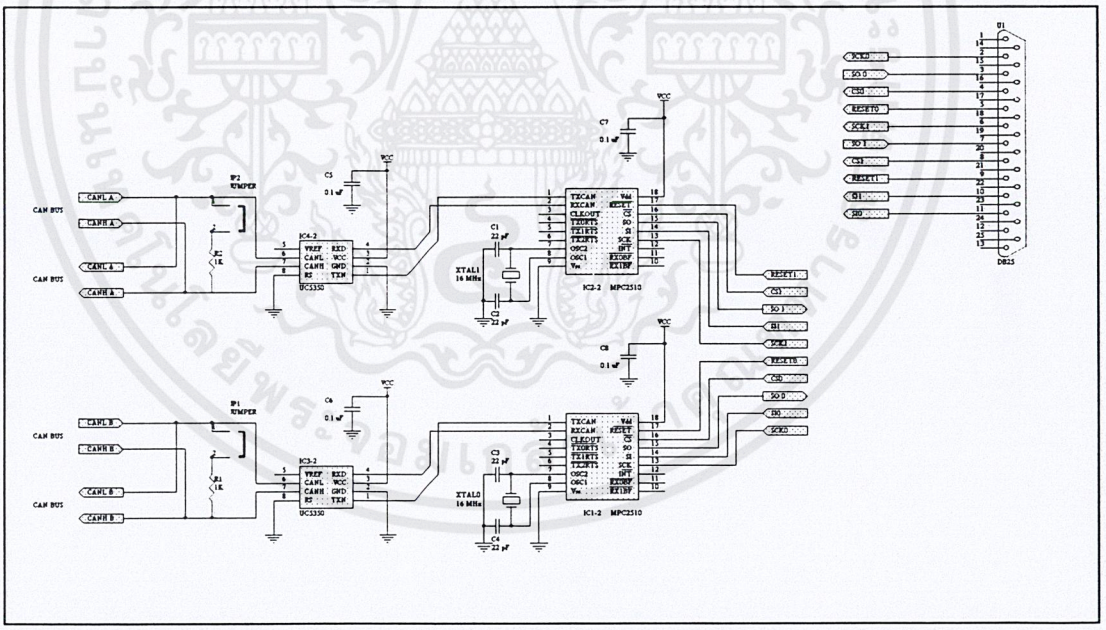
รีจิสเตอร์สำหรับควบคุม (Control Register) มีตำแหน่งพอร์ตอยู่ที่ IOBase + 2 มีความกว้างขนาดข้อมูล 1 ไบต์ (ใช้จริง 4 บิต) สามารถทำการอ่านหรือเขียนข้อมูลได้ (Read/Write Access) แต่โดยปกติจะใช้เขียนข้อมูลอย่างเดียว ข้อมูลแต่ละบิตจะเชื่อมโยงกับขาสัญญาณของคอนเน็กเตอร์ (DB25) โดยมี 3 บิตเป็นชนิด Reversed กล่าวคือตำแหน่งบิตที่ 0, 1 และ 3 จะมีสถานะปกติของลอจิกเป็น 1 (สถานะปกติของบิตทั่วไปจะเป็น '0')

โครงสร้างของบอร์ดอินเทอร์เฟซ CAN มีส่วนประกอบหลักอยู่สองส่วน ได้แก่ตัวควบคุม CAN (MCP2510) และตัวรับส่งสัญญาณ CAN (CAN Transceiver เบอร์ UC5350) โดยตัวควบคุม CAN จะทำหน้าที่หลักในการควบคุมกระบวนการสื่อสารให้เป็นไปตามโพรโตคอลที่กำหนดไว้ในดาต้าลิงก์เลเยอร์ ส่วนตัวรับส่งสัญญาณ CAN จะทำหน้าที่แปลงสัญญาณข้อมูลที่ต้องการส่ง (ลอจิก '0' หรือ '1') ให้เป็นสัญญาณที่ใช้งานในบัส CAN (สถานะโดมิแนนต์หรือ รีเซสสีฟ) ซึ่งจะสอดคล้องกับมาตรฐานที่ได้กำหนดไว้ในฟิสิคอลเลเยอร์ในขณะเดียวกันก็แปลงสัญญาณที่ได้รับจากบัส CAN กลับเป็นสัญญาณข้อมูลด้วย

เมื่อได้สร้างส่วนประกอบหลักของวงจรทางฮาร์ดแวร์ที่ใช้ในการควบคุมกระบวนการทำงานให้เป็นไปตาม CAN โพรโตคอลจึงนำมาเชื่อมต่อเข้ากับเครื่องคอมพิวเตอร์ส่วนบุคคลเพื่อทดสอบ โดยเชื่อมต่อผ่านทางพอร์ตขนานดังตารางที่ 4.7 เป็นตารางสรุปหน้าที่และตำแหน่งของขาในการเชื่อมต่อระหว่างพอร์ตขนานกับ MCP2510 จะใช้งานทั้งหมด 10 ขาสัญญาณ และได้แสดงลักษณะการเชื่อมต่อวงจรตารางที่ 4.7 เป็นรูปแสดงการเชื่อมต่อ CAN ร่วมกับ เครื่องคอมพิวเตอร์ส่วนบุคคล

ขาสัญญาณ	หน้าที่
2 (Data bit 0)	Serial Clock CAN 0
3 (Data bit 1)	Serial Out CAN 0
4 (Data bit 2)	Chip Select CAN 0
5 (Data bit 3)	Reset CAN 0
6 (Data bit 4)	Serial Clock CAN 1
7 (Data bit 5)	Serial Out CAN 1
8 (Data bit 6)	Chip Select CAN 1
9 (Data bit 7)	Reset CAN 1
10 (Status bit 6)	Serial In CAN 1
11 (Status bit 7)	Serial In CAN 0

ตารางที่ 4.7 หน้าที่ของขาต่างๆ ในการเชื่อมต่อระหว่างตัวควบคุม CAN MCP2510 กับเครื่องคอมพิวเตอร์ส่วนบุคคล



รูปที่ 4.25 แสดงวงจรของบอร์ดอินเทอร์เฟซ CAN เชื่อมต่อกับเครื่องคอมพิวเตอร์ส่วนบุคคล

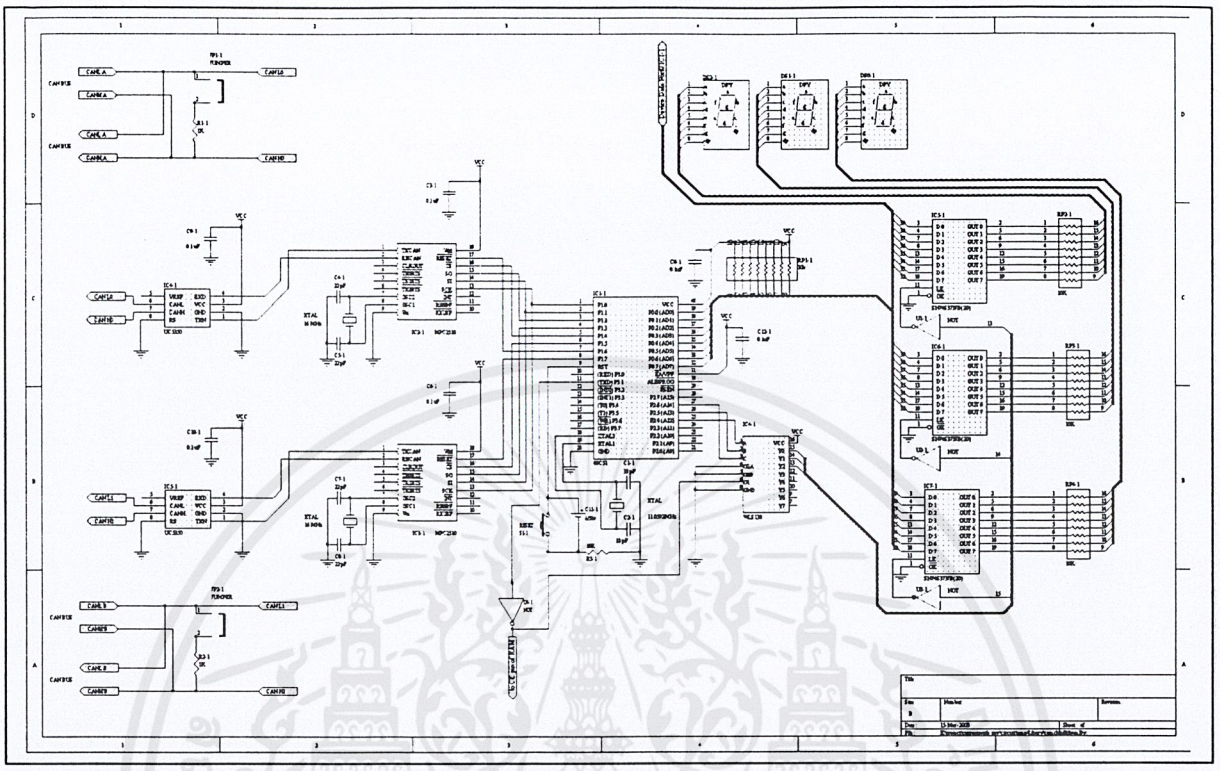
4.5.2 การออกแบบฮาร์ดแวร์เชื่อมต่อกับไมโครคอนโทรลเลอร์

วงจรทางฮาร์ดแวร์อีกส่วนหนึ่งคือการเชื่อมต่อ CAN อินเทอร์เฟซร่วมกับไมโครคอนโทรลเลอร์ MCS51 เบอร์ AT89C55 การเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับ MCP2510 จะเชื่อมต่อผ่านพอร์ต 2 ของไมโครคอนโทรลเลอร์เลอร์เบอร์ AT89C55 โดยมีรายละเอียดในการเชื่อมต่อดังนี้

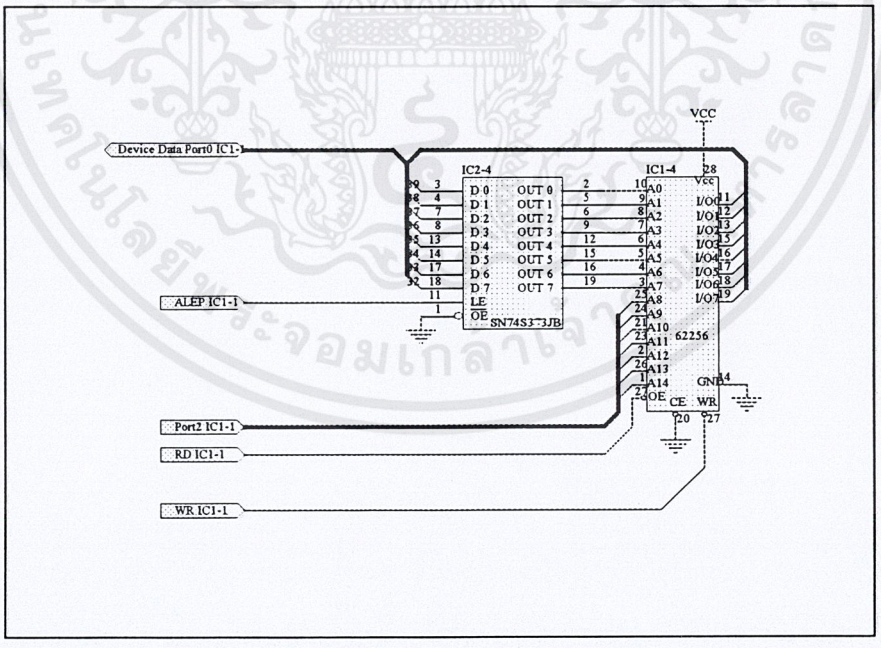
ขาสัญญาณ	หน้าที่
1	Serial Clock CAN 0, 1
2	Serial In CAN 0, 1
3	Serial Out CAN 0
4	Serial Out CAN 1
5	Chip Select CAN 0
6	Chip Select CAN 1
7	Reset CAN 0
8	Reset CAN 1

ตารางที่ 4.8 หน้าที่ของขาต่างๆ ในการเชื่อมต่อระหว่างตัวควบคุม CAN เบอร์ MCP2510 กับ ไมโครคอนโทรลเลอร์ MCS51 เบอร์ AT89C55 ผ่านทางพอร์ต 2

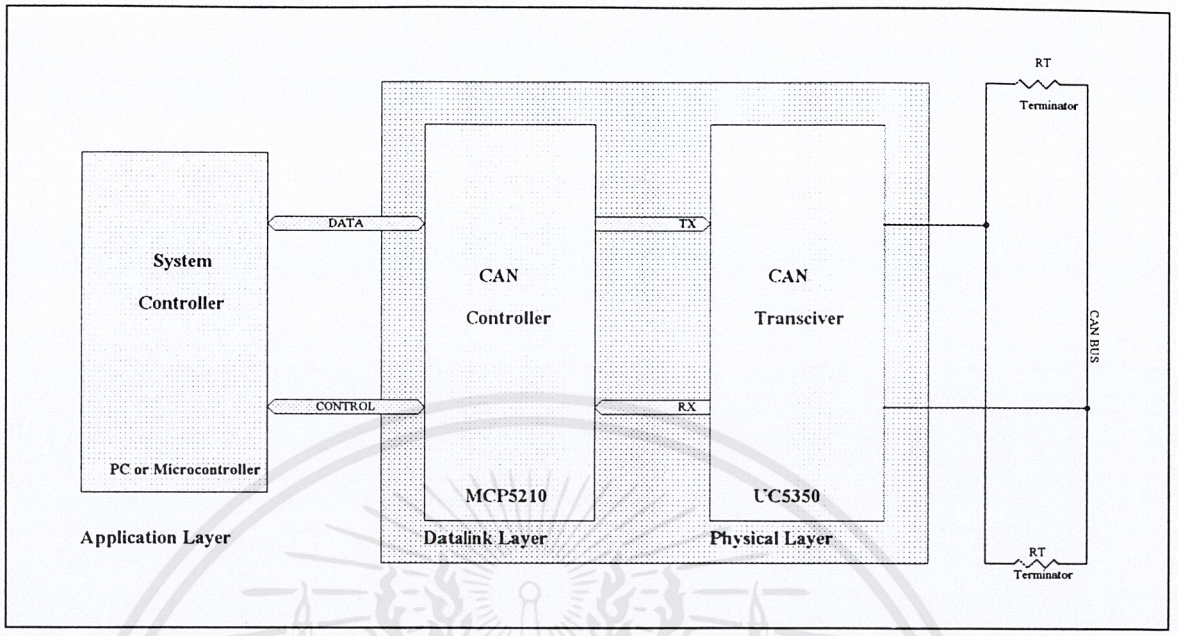
นอกจากนั้นยังมีส่วนในการแสดงผลเป็นเซเวนต์เซกเมนต์ (LED 7-Segment) ซึ่งใช้สำหรับตรวจสอบการทำงานของตัวไมโครคอนโทรลเลอร์เองและการเชื่อมต่อกับหน่วยความจำข้อมูลภายนอก (RAM) อีกด้วยดังแสดงในรูปวงจรที่ 4.26 และการเชื่อมต่อหน่วยความจำข้อมูลภายนอกในรูปที่ 4.27



รูปที่ 4.26 วงจรการเชื่อมต่อตัวควบคุม CAN ร่วมกับไมโครคอนโทรลเลอร์ MCS51 เบอร์ AT89C55



รูปที่ 4.27 การเชื่อมต่อ หน่วยความจำภายนอกเบอร์ 62256 ร่วมกับไมโครคอนโทรลเลอร์



รูปที่ 4.28 โครงสร้างการทำงาน

เนื่องจากไอซีตัวควบคุม CAN นั้นมีให้เลือกใช้อยู่หลายเบอร์ด้วยกันดังตารางที่ 4.9 สาเหตุที่เลือกใช้เบอร์ MCP2510 ก็เนื่องจากสามารถหาซื้อได้ภายในประเทศรวมไปถึงตัวรับส่งสัญญาณ CAN เบอร์ UC5350 ก็เช่นเดียวกัน ถึงแม้ CAN โพรโตคอลจะถูกคิดค้นขึ้นมานานแล้วแต่ไม่ค่อยเป็นที่รู้จักและแพร่หลายในประเทศไทยนักจึงหาซื้อได้ยาก

ภาพรวมของการทำงาน หากเป็นการส่งข้อมูลไปยังบัส CAN การทำงานจะเริ่มต้นด้วยการส่งสัญญาณไปควบคุม CAN เพื่อจัดการกับข้อมูลที่ต้องการส่งนั้นไปยังไอซี MCP2510 แล้วทำการจัดส่งข้อมูลออกไปโดยให้เป็นไปตาม CAN โพรโตคอลข้อมูลที่ส่งออกมาจาก MCP2510 จะถูกส่งผ่านไปยัง UC5350 ซึ่งเป็นตัวรับส่งสัญญาณ ทำหน้าที่แปลงสัญญาณข้อมูลที่จะส่งให้เป็นสัญญาณตามข้อกำหนดของ CAN โพรโตคอลสามารถแสดงภาพรวมได้ดังรูปที่ 4.28

สำหรับการรับข้อมูลจากบัส CAN นั้นสัญญาณจากบัส CAN จะถูกส่งมาที่ไอซี UC5350 ถูกแปลงเป็นสัญญาณข้อมูลดิจิตอลก่อนที่สัญญาณข้อมูลจะถูกส่งไปที่ไอซี MCP2510 เพื่อจัดการข้อมูลที่ได้รับตาม CAN โพรโตคอลต่อไป

จะเห็นว่าส่วนทำงานหลักๆของการรับส่งข้อมูลจะอยู่ที่ไอซี MCP2510 ที่ทำหน้าที่เป็นตัวควบคุมโพรโตคอลในการรับส่งข้อมูลใน CAN ทำงานในชั้นดาต้าลิงค์เลเยอร์ และอีกส่วนที่สำคัญมากๆคือส่วนของ System Controller ชั้นแอพพลิเคชันเลเยอร์ซึ่งเป็นส่วนที่เราจะได้ทำการเพิ่มคุณสมบัติการทำงานเพิ่มเติมให้กับ CAN โพรโตคอลเดิม ซึ่งจะแยกออกได้เป็นการทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคลและบนไมโครคอนโทรลเลอร์

ทางด้านสัญญาณเอาต์พุต CAN จากไอซี UC5350 ขา CAN_H และ CAN_L จะมีการต่อ RT เป็นตัวต้านทานเทอร์มิเนเตอร์ (Terminator Resistance) ซึ่งจะช่วยในเรื่องการแมตชิ่งอิมพีแดนซ์ โดย

มีจัมเปอร์ JP1 และ JP2 เป็นตัวเลือกว่าจะต่อตัวต้านทานหรือไม่ สำหรับขนาดของเทอร์มิเนชันอิมพีแดนซ์นี้ในหลายบทความแนะนำให้ใช้ค่าประมาณ 120 โอห์ม แต่จากที่ได้ทดลองแล้วปรากฏว่ามีค่าต่ำเกินไปซึ่งจะโหลดกระแสไอซีตัวรับส่ง CAN มากเกินไปทำให้สัญญาณเอาต์พุต CAN ไม่ได้ขนาดตามข้อกำหนด แต่หากไม่ต่อเทอร์มิเนชันอิมพีแดนซ์ ก็จะทำให้สัญญาณเอาต์พุต CAN ไม่เสถียรภาพ โดยจากการทดลองพบว่าขนาดของเทอร์มิเนชันอิมพีแดนซ์ ที่เหมาะสมจะมีค่าประมาณ 200 โอห์ม

# NO.	Company	CAN Ver.	Object Storage Capacity	Acceptance Filtering Capability	Remarks
A_CAN	Bosch	2.0B	3 Tx / Rx buffers	1 full-bit mask	Synthesizable IP module, CAN module with 3 configurable message buffers, flexible interface for 8-bit, 16-bit, or 32-bit CPU
CAN Core	Bosch	2.0B	n.a	n.a	Synthesizable IP module, CAN core cell with protocol controller and shift register
CC750	Bosch	2.0B	14 Tx/Rx buffers + 1 double Rx buffer	2 global masks + 1 mask for Rx double buffer	function compatible with Intel 82527, interface to CPU by SPI only, SOIC16-W package
CC770	Bosch	2.0B	14 Tx/Rx buffers + 1 double Rx buffer	2 global masks + 1 mask for Rx double buffer	pin and function compatible with Intel 82527, interface is 8/16-bit multiplexed/nonmultiplexed
C_CAN	Bosch	2.0B	32 Tx / Rx buffers	32 full-bit masks	Synthesizable IP module, CAN module with message RAM for storage of 32 (or more) message objects, flexible interface for 8-bit, 16-bit, or 32-bit CPU
TTCAN	Bosch	ISO 11898 –1,4	32 Tx / Rx buffers	32 full-bit masks	Synthesizable IP module, CAN module with message RAM for storage of 32 message objects, flexible interface for 8-bit, or 16-bit CPU, TTCAN Level 2 support
SAE81C90/91	Infineon	2.0Bp	16 Tx/Rx buffers	16 full-bit masks	2 x 8-bit I/O ports
AN82527	Intel	2.0B	14 Tx/Rx buffers + 1 double Rx buffer	15 full-bit masks + 1 global mask	
AS82527	Intel	2.0B	14 Tx/Rx buffers + 1 double Rx buffer	15 full-bit masks + 1 global mask	
MCP2510	Microchip	2.0B	3 Tx buffers + 2 Rx buffers	6 full-bit masks + 2 global masks	Stand-alone CAN Controller, SPI Interface, PDIP/SOIC18, TSSOP20
MSM9225	OKI Electric	2.0B	16 Tx/Rx buffers	16 full-bit masks	
SJA1000	Philips	2.0B	1 Tx buffer + 64 RxFIFO	1 global 32-bit mask or 2 global 16-bit masks	replaces 82C200
TC190C580	Toshiba	2.0B	15 Tx/Rx buffers + 1 Rx buffer	15 full-bit masks + 1 global mask	

ตารางที่ 4.9 รายชื่อตัวอย่างไอซีตัวควบคุม CAN

4.6 ข้อมูลจำเพาะ MCP2510

4.6.1 รายละเอียด

MCP2510 ผลิตขึ้นโดยบริษัท Microchip Technology Inc. เป็น Full Control Area Network (CAN) โพรโตคอลตามข้อกำหนด 2.0 A/B สนับสนุนโพรโตคอลเวอร์ชัน CAN 1.2, CAN 2.0A, CAN 2.0B Passive และ CAN 2.0 Active รับและส่งข้อมูลได้ทั้งแบบมาตรฐานและขยายและยัง ประกอบไปด้วยแอกเซปแตนซ์ฟิลเตอร์และ Message Management ภายในประกอบไปด้วยบัฟเฟอร์ขาส่ง 3 ชุด บัฟเฟอร์ขารับ 2 ชุด ดูแลจัดการด้วย Microcontroller Management (MCU) การสื่อสารจะเชื่อมต่อผ่าน Industry Standard Serial Peripheral Interface (SPI) ด้วยอัตราเร็วสูงถึง 5 เมกะบิตต่อวินาที

4.6.2 คุณสมบัติ

4.6.2.1 Full CAN V2.0A and V2.0B ที่1 เมกะบิตต่อวินาที :

- **4.6.2.1.1** สนับสนุนขนาดข้อมูล 0 ถึง 8 ไบต์
- **4.6.2.1.2** เฟรมข้อมูลมีสองแบบ เฟรมมาตรฐานและเฟรมขยาย
- **4.6.2.1.3** สามารถโปรแกรมให้มี อัตราเร็วในการสื่อสารได้ถึง 1 เมกะบิตต่อวินาที
- **4.6.2.1.4** สนับสนุนการใช้งานรีโมทเฟรม
- **4.6.2.1.5** มีบัฟเฟอร์ขารับ 2 ชุดที่สามารถจัดลำดับความสำคัญได้
- **4.6.2.1.6** มีแอคเซฟแตนท์ฟิลเตอร์ 6 ชุด
- **4.6.2.1.7** มีแอคเซฟแตนท์ฟิลเตอร์มาร์ส 2 ชุด
- **4.6.2.1.8** มีบัฟเฟอร์ขาส่ง 3 ชุดที่ที่สามารถจัดลำดับความสำคัญได้
- **4.6.2.1.9** มีโหมดลูปแบค (Loop Back) สำหรับทดสอบการทำงานในตัวเองโดยไม่ต้องเชื่อมต่อกับโหนดอื่น

4.6.2.2 คุณสมบัติทางฮาร์ดแวร์ :

- **4.6.2.2.1** มีการเชื่อมต่อแบบ SPI ความเร็วสูง (5 MHz at 4.5 V)
- **4.6.2.2.2** สนับสนุน SPI โหมด 0, 0 และ 1, 1
- **4.6.2.2.3** มีขาสัญญาณนาฬิกาเอาต์พุต สามารถเลือกได้ว่าจะอีนาเบิลหรือไม่
- **4.6.2.2.4** 'Buffer full' มีขาเอาต์พุตของสัญญาณอินเทอร์รัปต์สำหรับแต่ละบัฟเฟอร์ขารับ
- **4.6.2.2.5** ขา 'Request to send' เป็นขาอินพุตใช้ควบคุมการส่งข้อความในทันทีของแต่ละบัฟเฟอร์ขาส่ง
- **4.6.2.2.6** โหมดประหยัดพลังงาน 'Sleep mode'

4.6.2.3 ใช้พลังงานต่ำด้วยเทคโนโลยี CMOS :

4.6.2.3.1 ทำงานที่ระดับแรงดัน 3.0 โวลต์ ถึง 5.5 โวลต์

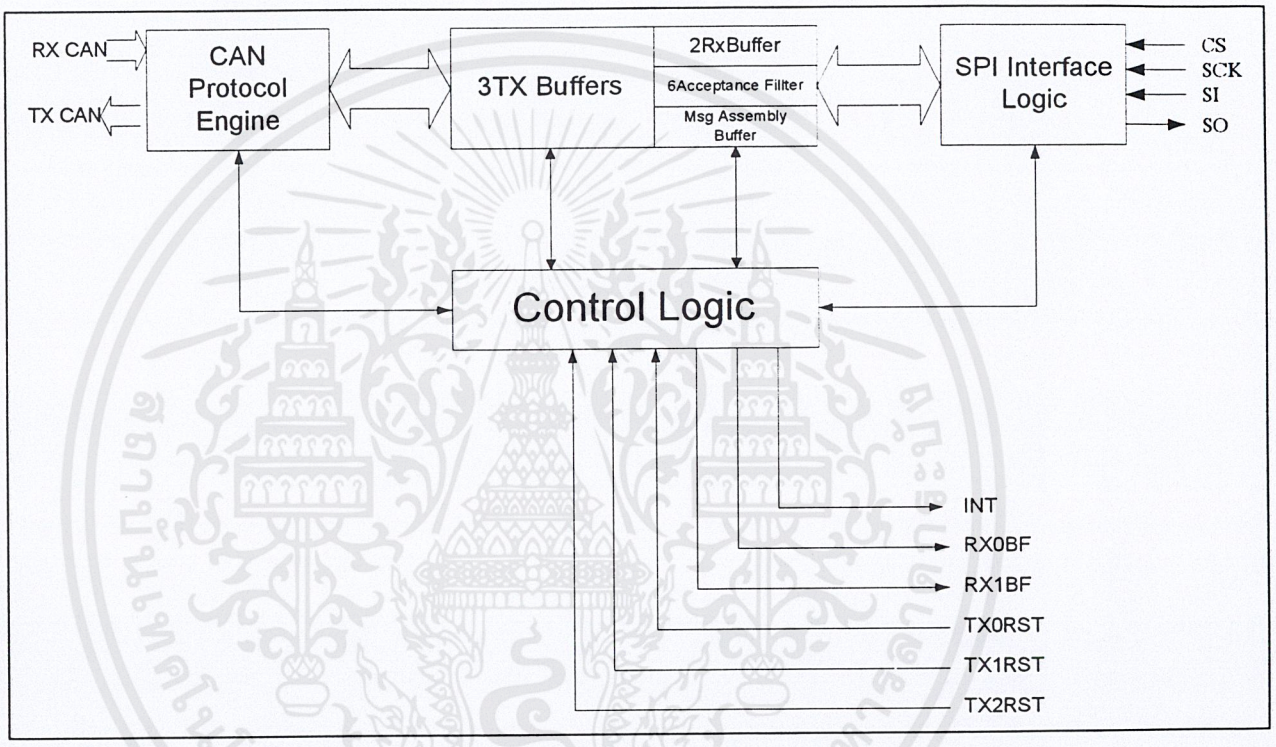
4.6.2.3.2 สภาวะทำงานปกติใช้กระแส 5 มิลลิแอมป์

4.6.2.3.3 ในโหมด Standby ที่แรงดัน 5.5 โวลต์กินกระแส10 ไมโครแอมป์

4.6.2.4 อุณหภูมิที่สนับสนุน :

4.6.2.4.1 Industrial (I) : -40°C to +85°C

4.6.2.4.2 Extended (E) : -40°C to +125°C



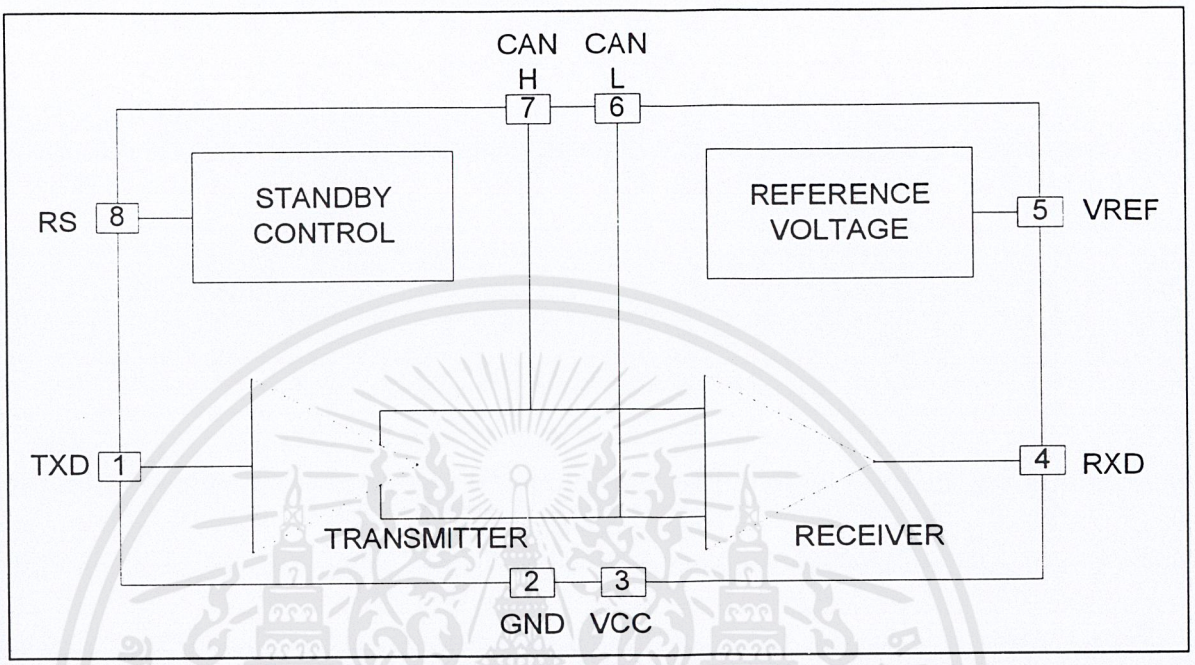
รูปที่ 4.29 บล็อกไดอะแกรมภายในไอซีตัวควบคุม CAN เบอร์ MCP2510

4.7 ข้อมูลจำเพาะ UC5350

ไอซีตัวรับส่ง CAN เบอร์ UC5350 (CAN Transceiver) ถูกออกแบบมาสำหรับใช้ประโยชน์ในงานอุตสาหกรรมการสื่อสารระดับฟิสิคอลเลเยอร์ตามมาตรฐาน ISO 11898 ออกแบบมาให้สามารถทำงานได้ที่ความเร็วสูงสุดถึง 1 เมกกะบิตต่อวินาที (Mb/s.) ถูกออกแบบมาให้ทนต่อสภาพแวดล้อมหรือสถานการณ์เช่นการต่อสายครอส (Cross wrie) , แรงดันเกิน, และป้องกันอุณหภูมิเกิน

การเชื่อมต่อเป็นเชื่อมต่อระหว่าง ตัวควบคุม CAN และบัสสัญญาณที่พบในงานอุตสาหกรรมและยานยนต์นั้น จะทำงานอยู่ที่ระดับแรงดัน –7 โวลต์ ถึง 12 โวลต์ในโหมดใช้งานปกติ และสามารถทนได้ถึงระดับแรงดัน –25 ถึง 18 โวลต์ จากการทดสอบในห้องแล็ป มีดิฟเฟอร์เรนต์เชียลอินพุตอิมพีแดนซ์สูง วงจรไดร์ฟสัญญาณแบบซิมเมทริกคอล มี Propagation delay ต่ำ เพิ่มแบนด์วิดธ์และความยาวของสายได้ด้วยการลดการสะท้อนของสัญญาณและการ Distortion

สามารถทำงานได้ที่ช่วงอุณหภูมิ –40 ถึง 85 องศาเซลเซียส บรรจุอยู่ในแพ็คเกจ ขนาด 8 ขา แบบ Dual inline package



รูปที่ 4.30 บล็อกไดอะแกรมภายในไอซีเบอร์ UC5350

บทที่ 5

ผลการทดลอง

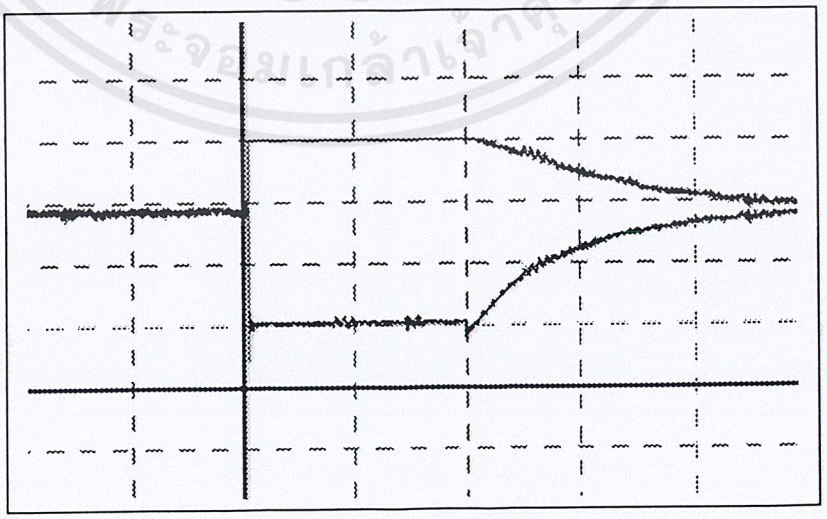
5.1 การทดลองเพื่อเลือกค่าความต้านทานเทอร์มิเนชั่นอิมพีแดนซ์ที่เหมาะสม

เนื่องจากการสื่อสารในเครือข่ายด้วย CAN โพรโตคอลที่เราเลือกใช้นี้เป็นการสื่อสารด้วยการเชื่อมต่อสายสัญญาณแบบบัส (Bus Topology) ซึ่งอุปกรณ์ทุกตัวจะอยู่บนสายสัญญาณคู่เดียวกันและต้องปิดปลายสายสัญญาณทั้งสองด้วยเทอร์มิเนชั่นอิมพีแดนซ์ สำหรับค่าตัวต้านทานที่เหมาสมนั้นเป็นไปดังตารางที่ 5.1

ความยาวสาย	คุณสมบัติสายส่ง		เทอร์มิเนชันอิมพีแดนซ์	อัตราเร็วข้อมูลสูงสุด
	ความต้านทาน	ขนาด		
0 – 40 m	70 mΩ/m	0.25 – 0.34 mm^2 AWG23,AWG22	124 Ω(1%)	1Mbit/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 300 Ω	100 Kbit/s ที่ 500 m
600 – 1 Km	<26 mΩ/m	0.75 – 0.8 mm^2 AWG18	150 Ω ถึง 300 Ω	50 Kbit/s ที่ 1 m

ตารางที่ 5.1 ความสัมพันธ์ของคุณสมบัติของสายส่งกับอัตราเร็วข้อมูล

5.1.1 การทดลองโดยไม่ต่อเทอร์มิเนชันอิมพิแดนซ์

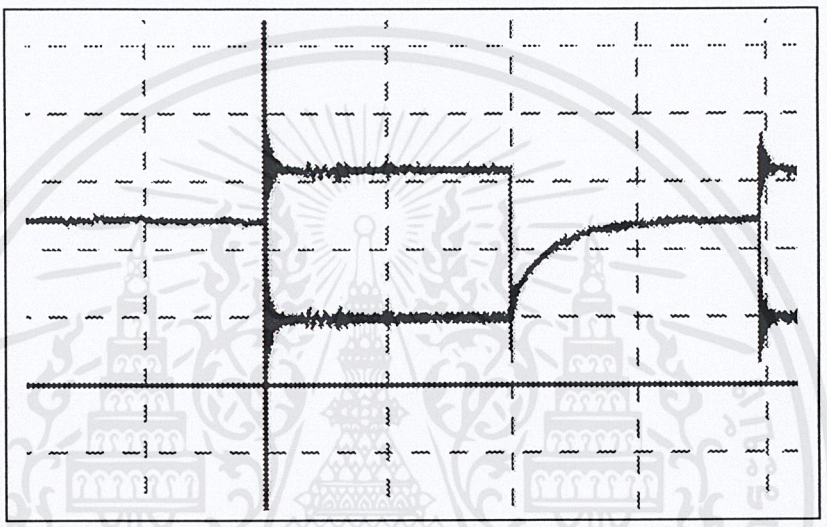


รูปที่ 5.1 สัญญาณเมื่อไม่ต่อเทอร์มิเนชันอิมพิแดนซ์

ผลการทดลอง

จากการทดลองวัดสัญญาณจากตัวรับส่ง CAN โดยไม่ต่อเทอร์มิเนชันอิมพิแดนซ์ ปรากฏว่าระดับสัญญาณบนสายบัสทั้งสองเส้นนั้นรูปร่างของสัญาณคลาดเคลื่อนไปจากมาตรฐานของ CAN โพรโตคอล

5.1.2 การทดลองโดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 200 Ω (5%)

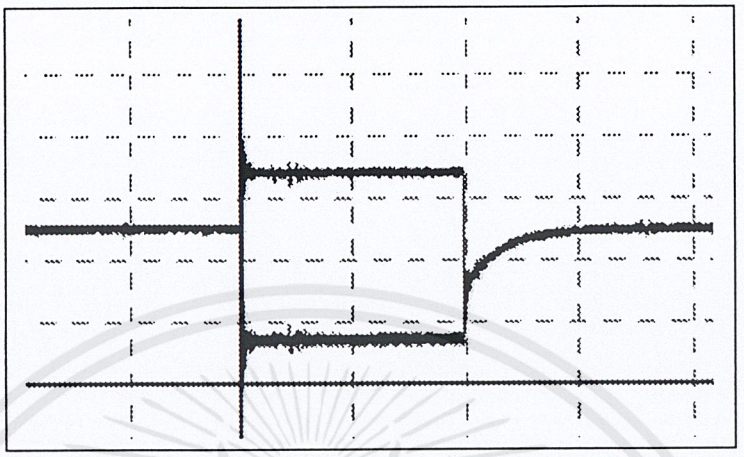


รูปที่ 5.2 สัญญาณเมื่อต่อเทอร์มิเนชันอิมพิแดนซ์ขนาด 200 Ω (5%)

ผลการทดลอง

จากการทดลองวัดสัญญาณจากตัวรับส่ง CAN โดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 100 Ω (5%) ปรากฏว่ารูปร่างของสัญญาณเป็นไปตามมาตรฐานโดยมีระดับแรงดันในสถานะรีเซสสีฟ 2.5 โวลต์ และมีค่าความต่างศักย์ระหว่าง CAN High กับ CAN Low 2.1 โวลต์

5.1.3 การทดลองโดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 400 Ω (5%)

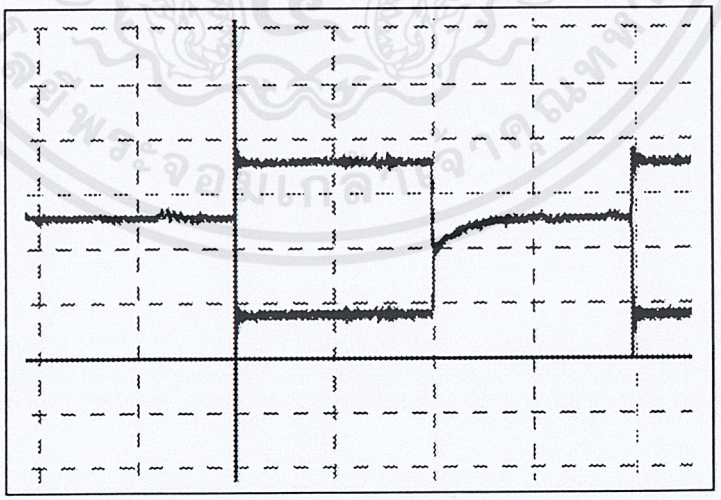


รูปที่ 5.3 สัญญาณเมื่อต่อเทอร์มิเนชันอิมพิแดนซ์มีขนาด 400 Ω (5%)

ผลการทดลอง

จากการทดลองวัดสัญญาณจากตัวรับส่ง CAN โดยให้เทอร์มิเนชั่นอิมพีแดนซ์มีขนาด 400 Ω (5%) ปรากฏว่ารูปร่างของสัญญาณเป็นไปตามมาตรฐานโดยมีระดับแรงดันในสถานะรีเซสสีฟ 2.5 โวลต์ และมีค่าความต่างศักย์ระหว่าง CAN High กับ CAN Low 2.6 โวลต์

5.1.4 การทดลองโดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 1k Ω (5%)

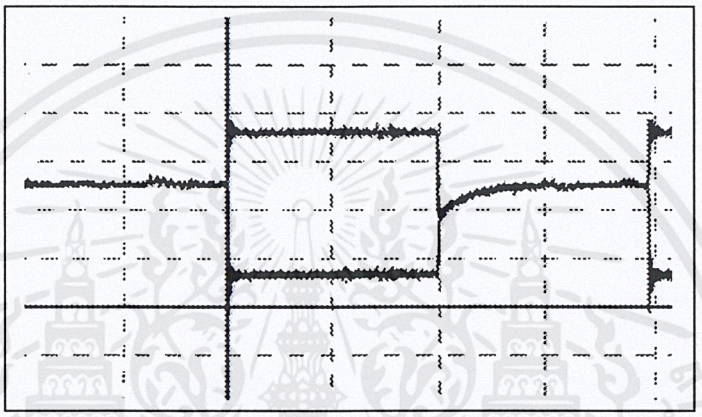


รูปที่ 5.4 สัญญาณเมื่อต่อเทอร์มิเนชันอิมพิแดนซ์ขนาด 1KΩ (5%)

ผลการทดลอง

จากการทดลองวัดสัญญาณจากตัวรับส่ง CAN โดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 1 kΩ (5%) ปรากฏว่ารูปร่างของสัญญาณเป็นไปตามมาตรฐานโดยมีระดับแรงดันในสถานะรีเซสสีฟ 2.5 โวลต์ และมีค่าความต่างศักย์ระหว่าง CAN High กับ CAN Low 2.8 โวลต์

5.1.5 การทดลองโดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 2K Ω (5%)



รูปที่ 5.5 สัญญาณเมื่อต่อเทอร์มิเนชันอิมพิแดนซ์ขนาด 2KΩ (5%)

ผลการทดลอง

จากการทดลองวัดสัญญาณจากตัวรับส่ง CAN โดยให้เทอร์มิเนชันอิมพีแดนซ์มีขนาด 2 kΩ (5%) ปรากฏว่ารูปร่างของสัญญาณเป็นไปตามมาตรฐานโดยมีระดับแรงดันในสถานะรีเซสสีฟ 2.5 โวลต์ และมีค่าความต่างศักย์ระหว่าง CAN High กับ CAN Low 3 โวลต์

ข้อสรุปในการทดลองวัดสัญญาณจากตัวรับส่ง CAN

จากการทดลองจะพบว่าสัญญาณจากตัวรับส่ง CAN มีรูปร่างของสัญญาณเป็นไปตามมาตรฐานเมื่อใช้ค่าความต้านทานตั้งแต่ 200 Ω ถึง 2 kΩ แต่มีค่าความแตกต่างของระดับแรงดัน CAN High และ CAN Low ในสถานะรีเซสสีฟที่ต่างกัน ดังนั้นค่าความต้านทานที่เหมาะสมในการใช้งานจึงอยู่ที่ประมาณ 200 Ω เพราะมีค่าความแตกต่างของระดับแรงดัน CAN High และ CAN Low ในสถานะรีเซสสีฟอยู่ 2.1 โวลต์ ซึ่งใกล้เคียงกับมาตรฐานของ CAN มากที่สุด

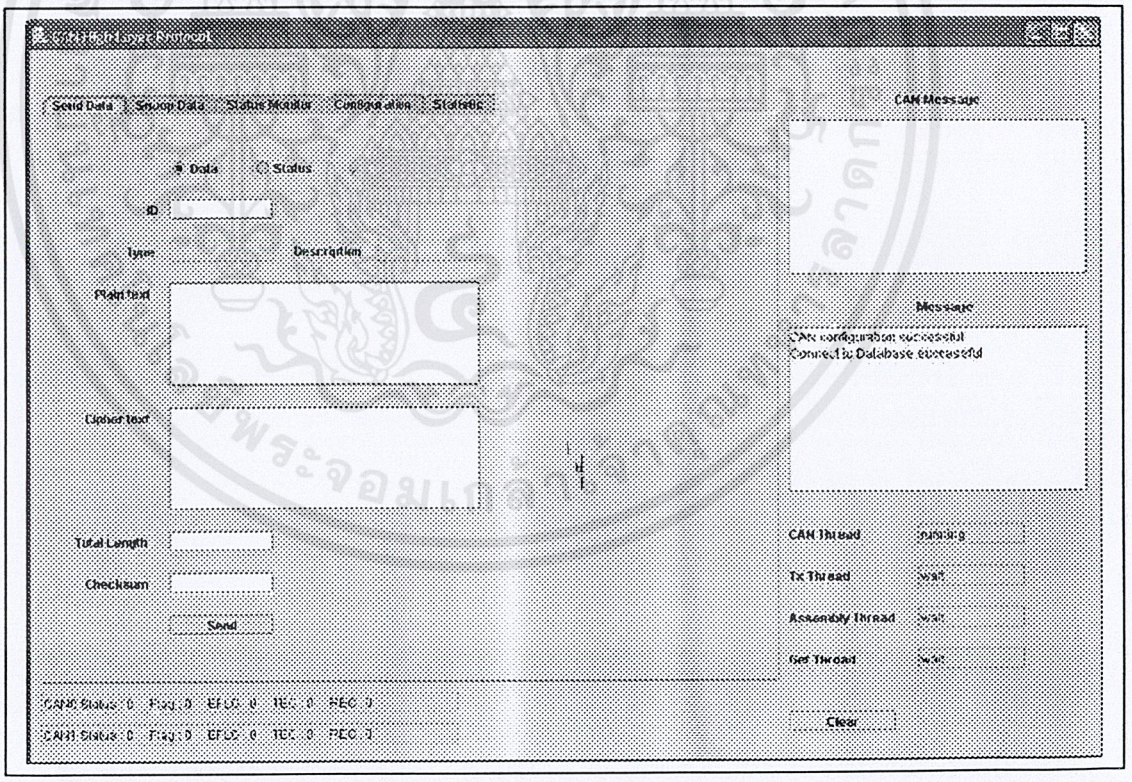
5.2 แนะนำโปรแกรมใช้ทดสอบโพรโตคอล

การออกแบบโปรแกรมเพื่อใช้ในการนำเสนอการทดสอบโพรโตคอลตามที่ได้ออกแบบไว้มีองค์ประกอบดังนี้

5.2.1 ส่วนที่ใช้ในการนำส่งข้อมูล

ส่วนที่ใช้ในการนำส่งข้อมูลนี้เป็นส่วนที่อนุญาตให้ผู้ใช้งานได้ทดสอบการส่งข้อมูลไปยังโหนดที่ต้องการ โดยผู้ใช้งานจะต้องป้อนข้อมูลที่สำคัญ ได้แก่ ID และเพลนเท็กซ์ในกรณีที่ผู้ใช้เลือกที่จะส่งข้อมูลโดยใช้เฟรมสำหรับรับส่งข้อมูลในกรณีที่ผู้ใช้ต้องการทดสอบการส่งสถานะโดยใช้เฟรมรายงานสถานะผู้ใช้จะต้องเลือก Radio Button ไปที่ Status จากนั้นผู้ใช้จะต้องป้อน ID, Type และ Description

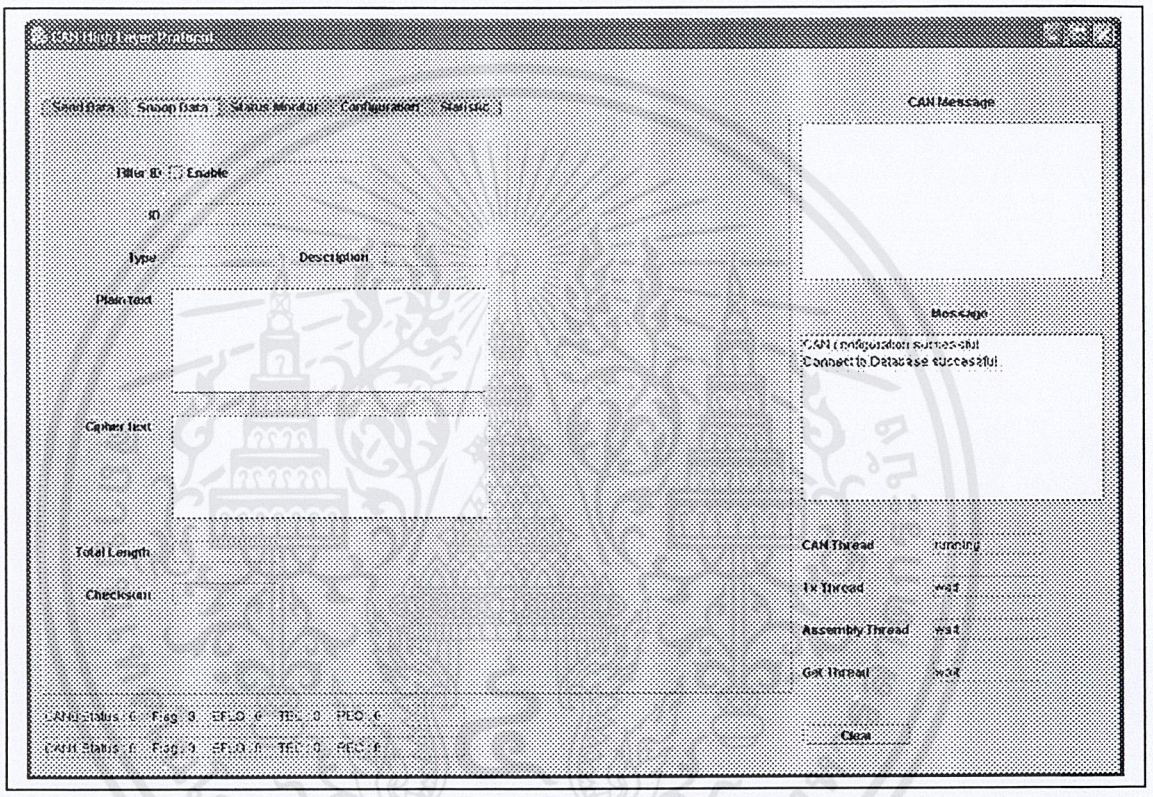
ในกรณีที่ผู้ใช้เลือกการส่งข้อมูลโดยใช้เฟรมสำหรับรับส่งข้อมูลเมื่อผู้ใช้ได้ป้อนข้อมูลตามข้อกำหนดแล้วโปรแกรมจะทำการคำนวณขนาดของเพลนเท็กซ์และผลรวมให้อัตโนมัติ ในกรณีที่ผู้ใช้เลือกการส่งข้อมูลโดยใช้เฟรมรายงานสถานะโปรแกรมจะทำการคำนวณเฉพาะส่วนของผลรวมเท่านั้น



รูปที่ 5.6 แสดงอินเทอร์เฟซที่ใช้ในการส่งข้อมูล

5.2.2 ส่วนที่ใช้ในการดักจับแพ็กเก็ต

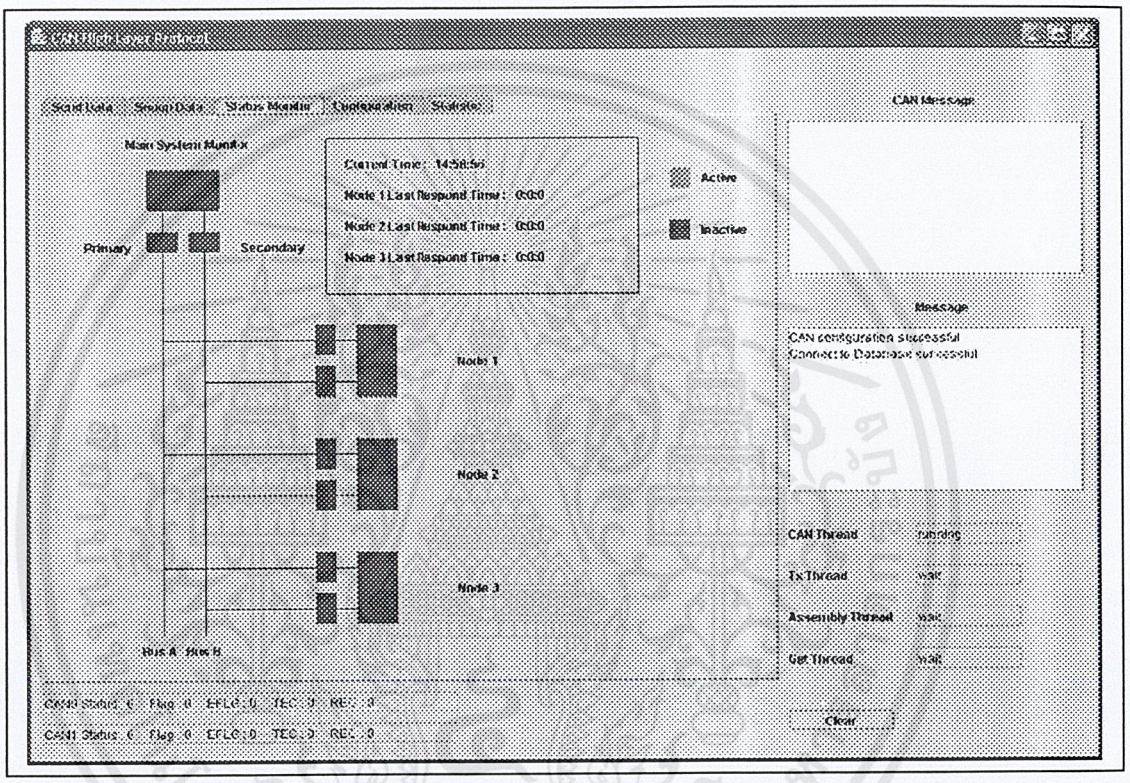
ส่วนที่ใช้ในการดักจับแพ็กเก็ตเป็นส่วนที่อนุญาตให้ตรวจสอบการรับส่งข้อมูลผ่านเครือข่าย โดยสามารถเลือกการกรองข้อมูลที่ต้องการได้โดยเลือกที่ Filter ID จากนั้นป้อน ID ของข้อความที่ต้องการ โปรแกรมจะทำการแสดงผลโดยกรองเอาเฉพาะข้อมูลที่มี ID ที่ตรงตามเลือกและแสดงผลตามประเภทของเฟรมที่ได้รับ



รูปที่ 5.7 แสดงอินเทอร์เฟซที่ใช้ในการดักจับแพ็กเก็ต

5.2.3 ส่วนที่ใช้ในการแสดงสถานะของระบบ

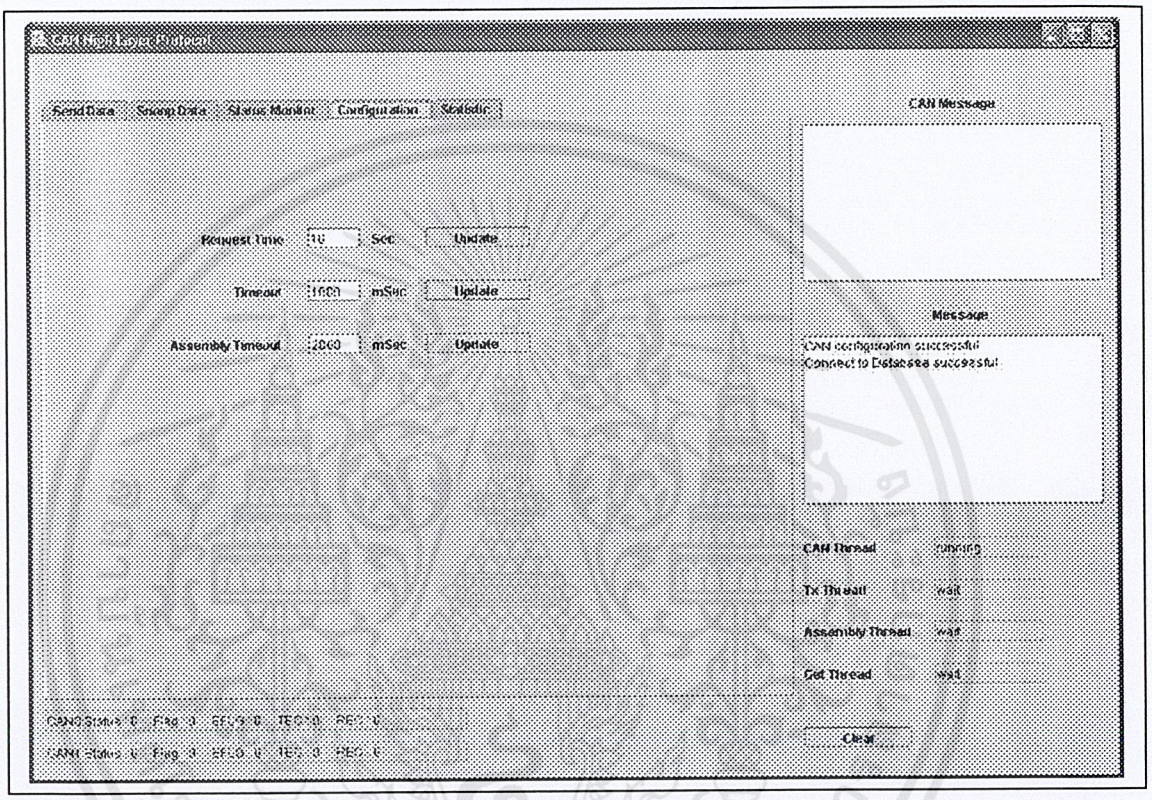
ส่วนที่ใช้ในการแสดงสถานะของระบบเป็นส่วนที่อนุญาตให้ตรวจสอบดูสถานะของโหนดและตัวควบคุม CAN ต่างๆที่อยู่บนเครือข่าย โปรแกรมจะรายงานสถานะของโหนดเป็นภาพกราฟฟิกส์ โดยที่สีเขียวจะแทนโหนดและตัวควบคุม CAN ที่อยู่ในสถานะ Active ส่วนสีแดงจะแทนโหนดและตัวควบคุม CAN ที่อยู่ในสถานะ Standby



รูปที่ 5.8 แสดงอินเทอร์เฟซที่ใช้ในการตรวจสอบสถานะ

5.2.4 ส่วนที่ใช้ในการตั้งค่าระบบ

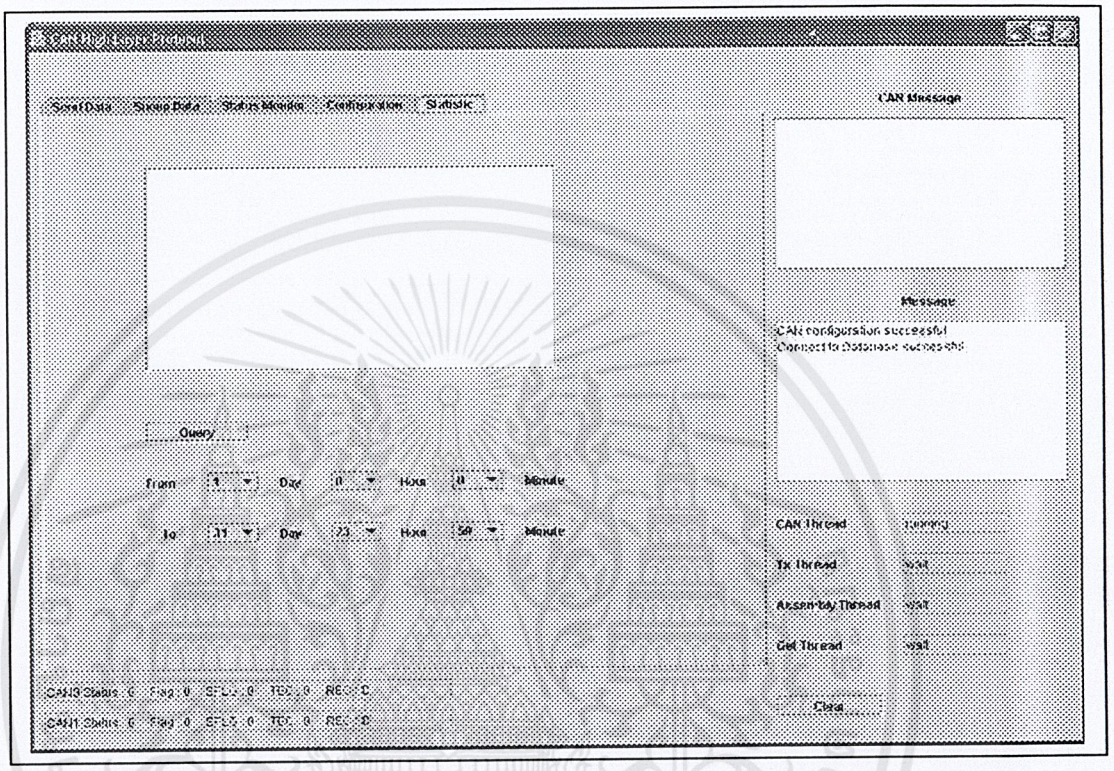
ส่วนที่ใช้ในการตั้งค่าระบบเป็นส่วนที่อนุญาตให้ทดลองตั้งค่าต่างๆของระบบเพื่อตรวจสอบดูการเปลี่ยนแปลงที่เกิดขึ้น โดยสามารถที่จะตั้งค่าที่ใช้ในเวลาที่ใช้ในการร้องขอสถานะของระบบ, ค่า Time out ในการส่งข้อมูลและค่าเวลา Time out ที่ใช้ในการประกอบข้อมูล



รูปที่ 5.9 แสดงอินเทอร์เฟซที่ใช้ในการตั้งค่าระบบ

5.2.5 ส่วนที่ใช้ในการตรวจสอบข้อมูลย้อนหลัง

ส่วนที่ใช้ในการตรวจสอบข้อมูลย้อนหลังเป็นส่วนที่อนุญาตให้ตรวจสอบดูสถานะของระบบย้อนหลังเพื่อใช้การวิเคราะห์ระบบ

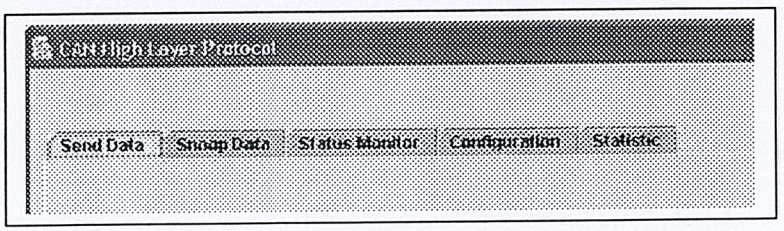


รูปที่ 5.10 แสดงอินเทอร์เฟซที่ใช้ในการตรวจสอบข้อมูลย้อนหลัง

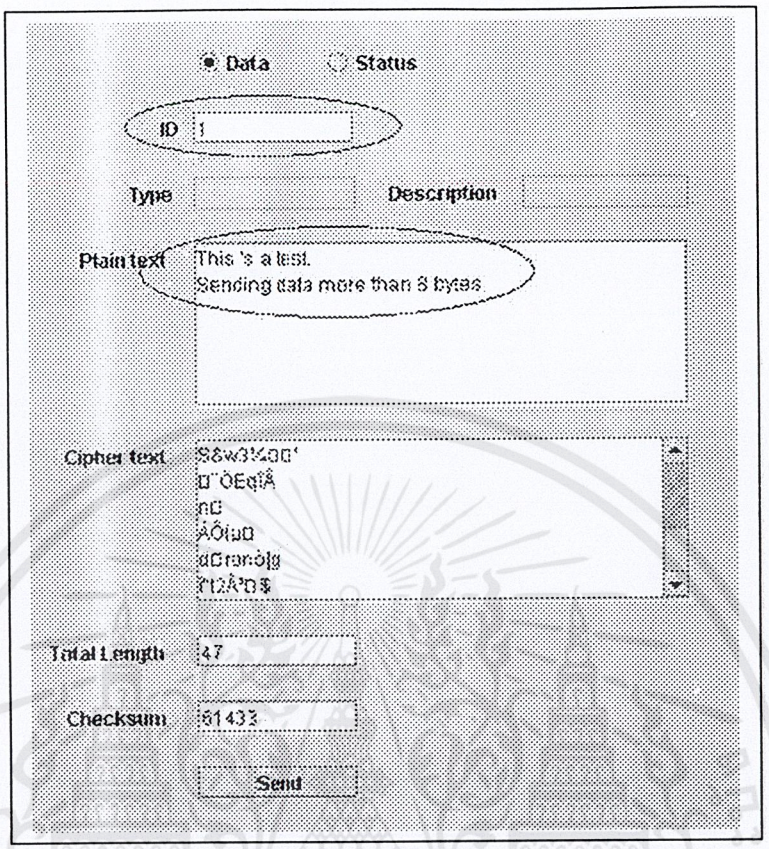
5.3 การทดสอบระบบ

การทดสอบระบบจะเป็นส่วนที่ใช้ในการทดสอบโพรโตคอลที่ได้ทำการออกแบบ โดยมีการทดสอบดังนี้

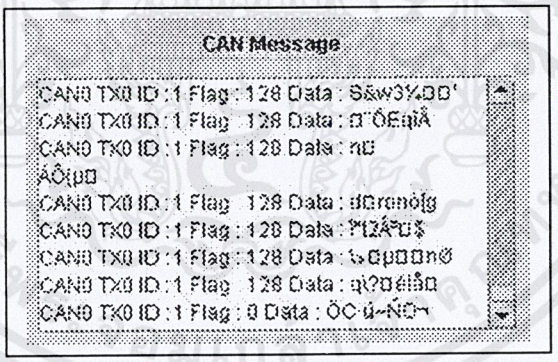
5.3.1 การทดสอบการนำส่งข้อมูลที่มีขนาดมากกว่า 8 ไบต์



(ก)



(ข)



(ค)

รูปที่ 5.11 การทดลองการส่งข้อมูล

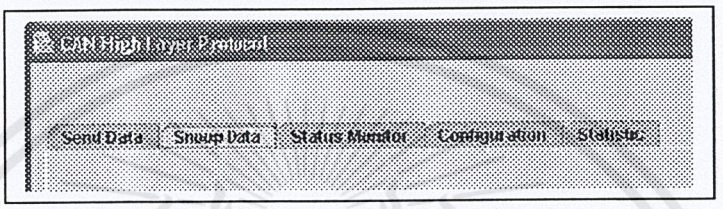
ขั้นตอนการทดลอง

1. เลือกที่แท็ป Send Data ดังรูปที่ 5.11 (ก)
2. กรอก ID และข้อมูลที่ต้องการส่งในช่อง Plain text ดังรูปที่ 5.11 (ข)
3. กดปุ่ม Send เพื่อส่งข้อมูล

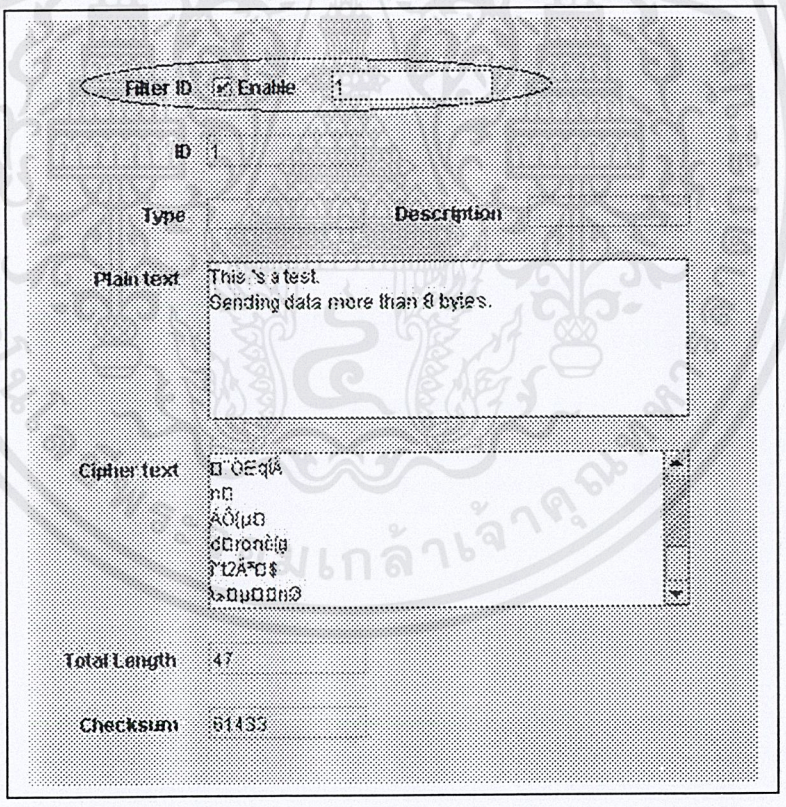
ผลการทดลอง

จากการทดลองเมื่อกรอกข้อมูลแล้วโปรแกรมจะทำการคำนวณขนาดของข้อมูลและผลรวม เมื่อกดปุ่ม Send เพื่อส่งข้อมูลโปรแกรมจะทำการแบ่งข้อมูลให้มีขนาด 8 ไบต์แล้วทำการเข้ารหัสดังรูปที่ 5.11 (ข) จากนั้นโปรแกรมจะนำส่งข้อมูลที่เข้ารหัสแล้วดังรูปที่ 5.11 (ค)

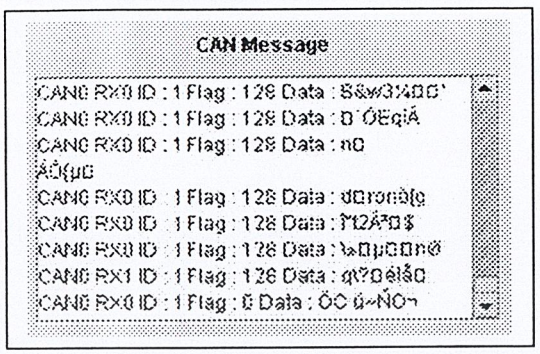
5.3.2 ทดสอบการดักจับแพ็กเก็ต



(ก)



(ข)



(ค)

รูปที่ 5.12 การทดลองการดักจับแพ็กเก็ต

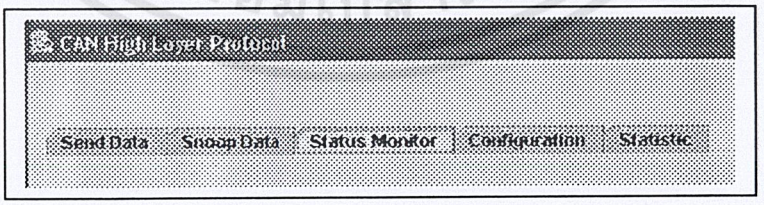
ขั้นตอนการทดลอง

1. เลือกที่แท็ป Snoop Data ดังรูปที่ 5.12 (ก)
2. คลิกที่ Filter ID เพื่อกรองข้อมูลโดยใช้ ID จากนั้นใส่ ID ของ Message ที่ต้องการดังรูปที่ 5.12 (ข)

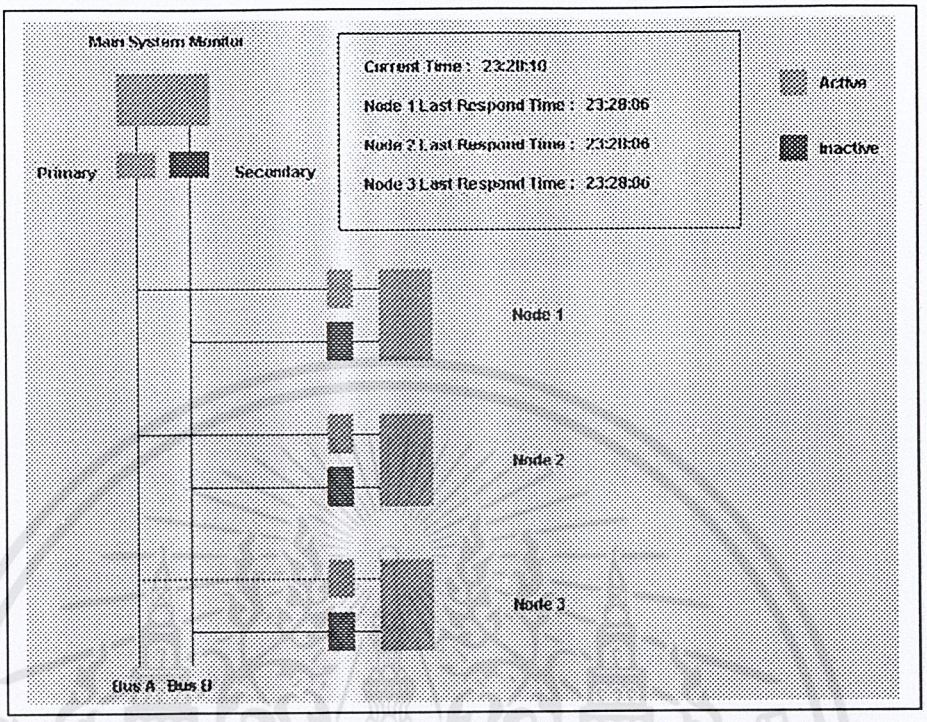
ผลการทดลอง

จากการทดลองเมื่อมีการส่งข้อมูลบนเครือข่ายและมี ID ของ Message ตรงกับที่ต้องการ โปรแกรมจะแสดงผลตามประเภทของเฟรมที่ได้รับเข้ามาดังรูปที่ 5.12 (ข) โดยใช้ ID เป็นเงื่อนไขในการดักจับแพ็กเก็ตและแสดงผล รูปที่ 5.12 (ค) แสดงข้อมูลในระดับ CAN โพรโตคอล

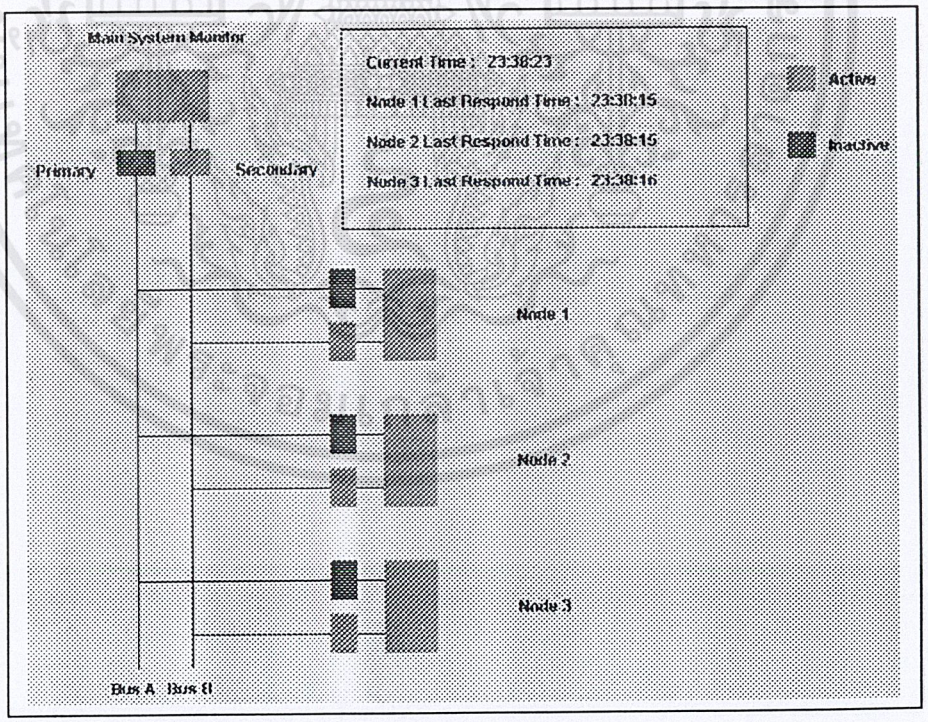
5.3.3 การแสดงสถานะของระบบ



(ก)



(ก)



(ข)

รูปที่ 5.13 การตรวจสอบสถานะของระบบ

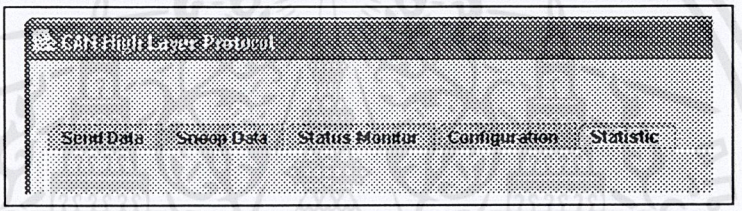
ขั้นตอนการทดลอง

1. เลือกที่แท็ป Status Monitor เพื่อดูสถานะปัจจุบันดังรูปที่ 5.13 (ก)
2. รูปที่ 5.13 (ข) แสดงสถานะปัจจุบันของระบบ จากนั้นทำการถอดสายสัญญาณที่ Active อยู่ออก

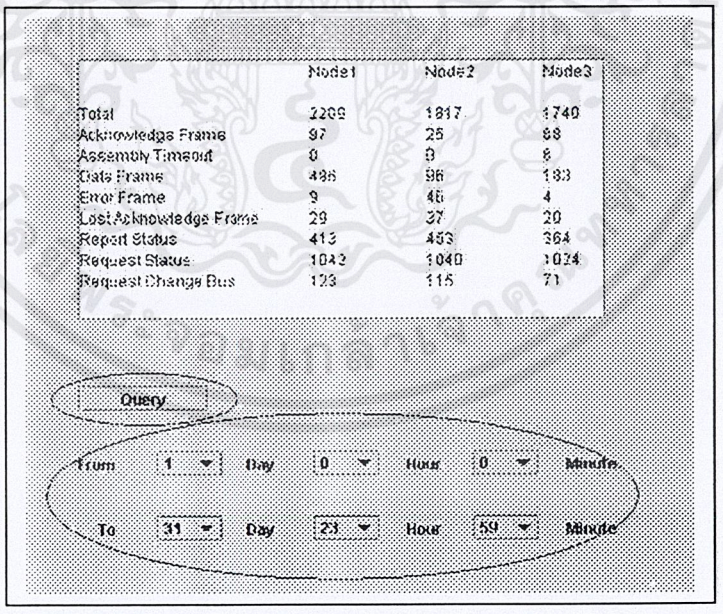
ผลการทดลอง

จากการทดลองเมื่อทำการถอดสายสัญญาณที่ Active ออกเมื่อโปรแกรมจะทำการส่งข้อมูลและพบว่าไม่สามารถส่งข้อมูลได้จะสลับไปใช้งานสายสัญญาณอีกเส้นหนึ่ง ดังรูปที่ 4.13 (ค)

5.3.4 การตรวจสอบข้อมูลย้อนหลัง



(ก)



(ข)

รูปที่ 5.14 การตรวจสอบข้อมูลย้อนหลัง

ขั้นตอนการทดลอง

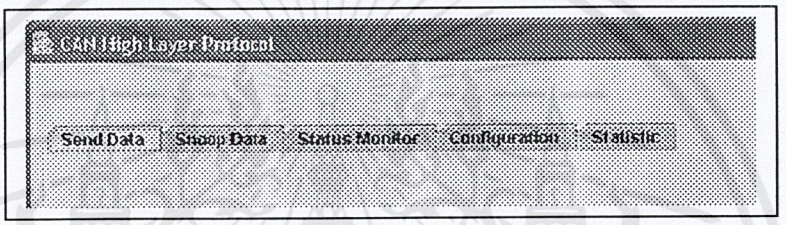
1. เลือกที่แท็ป Statistic ดังรูปที่ 5.14 (ก)

2. เลือกช่วงเวลาที่ต้องการแล้วกดปุ่ม Query เพื่ออ่านข้อมูลจากดาต้าเบสดังรูปที่ 5.14 (ข)

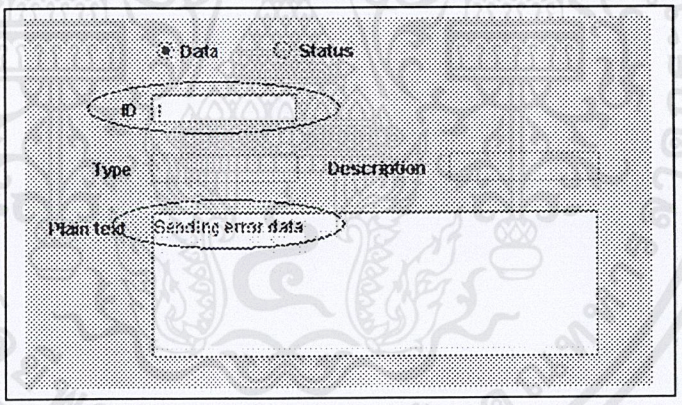
ผลการทดลอง

จากการทดลอง หลังจากที่เลือกช่วงเวลาที่ต้องการและกดปุ่ม Query แล้วโปรแกรมจะทำการอ่านข้อมูลจากดาต้าเบสตามช่วงเวลาที่ผู้ใช้เลือกและทำการแสดงผลดังรูปที่ 5.14 (ข)

5.3.5 การตรวจสอบ Packet ผิดปกติ

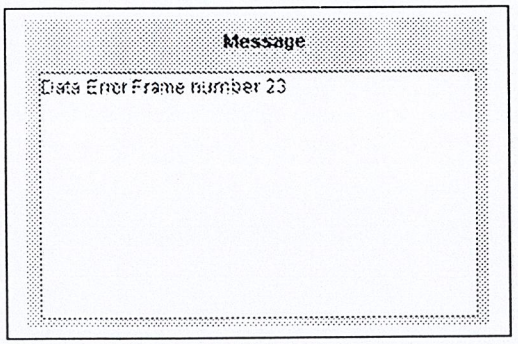


(ก)



(ข)

(ค)



(ง)

รูปที่ 5.15 การตรวจสอบ Packet ผิดปกติ

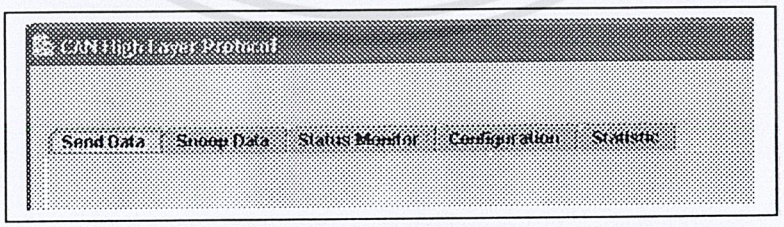
ขั้นตอนการทดลอง

1. เลือกที่แท็ป Send Data ดังรูปที่ 5.15 (ก)
2. กรอก ID และข้อมูลที่ต้องการส่งดังรูปที่ 5.15 (ข)
3. ทำการเปลี่ยนแปลงค่าผลรวมจาก 63778 เป็น 63775 แล้วกดปุ่ม Send เพื่อส่งข้อมูลดังรูปที่ 5.15 (ค)

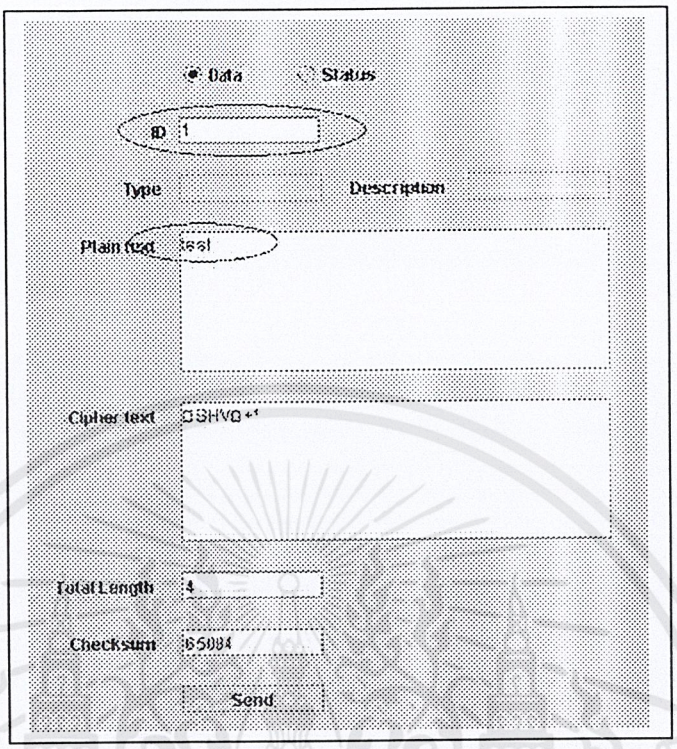
ผลการทดลอง

หลังจากที่ได้ทำการส่งข้อมูลไปที่โหนดปลายทางแล้ว โหนดปลายทางจะทำการถอดรหัสและคำนวณผลรวมแต่ผลลัพธ์ไม่ได้เท่ากับศูนย์แสดงว่ามีความผิดพลาดของข้อมูลเกิดขึ้นจึงแสดงข้อความรายงานข้อผิดพลาดดังรูปที่ 5.15 (ง)

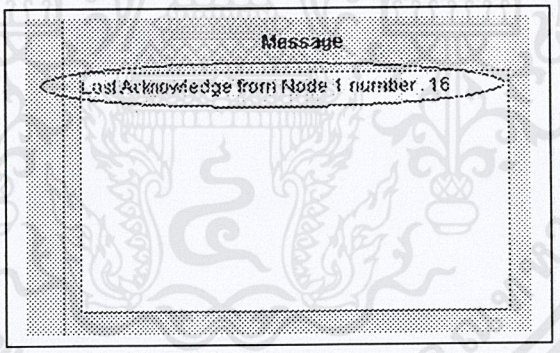
5.3.6 การตรวจสอบการนำส่งข้อมูล



(ก)



(ข)



(ค)

รูปที่ 5.16 การตรวจสอบการนำส่งข้อมูล

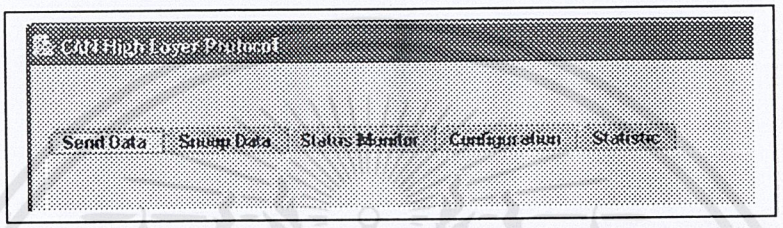
ขั้นตอนการทดลอง

1. เลือกที่แท็ป Send Data ดังรูปที่ 5.16 (ก)
2. กรอก ID และข้อมูลที่ต้องการส่งดังรูปที่ 5.16 (ข)
3. ปิดโปรแกรมจำลองโหนดที่ 1 จากนั้นกด Send เพื่อส่งข้อมูล

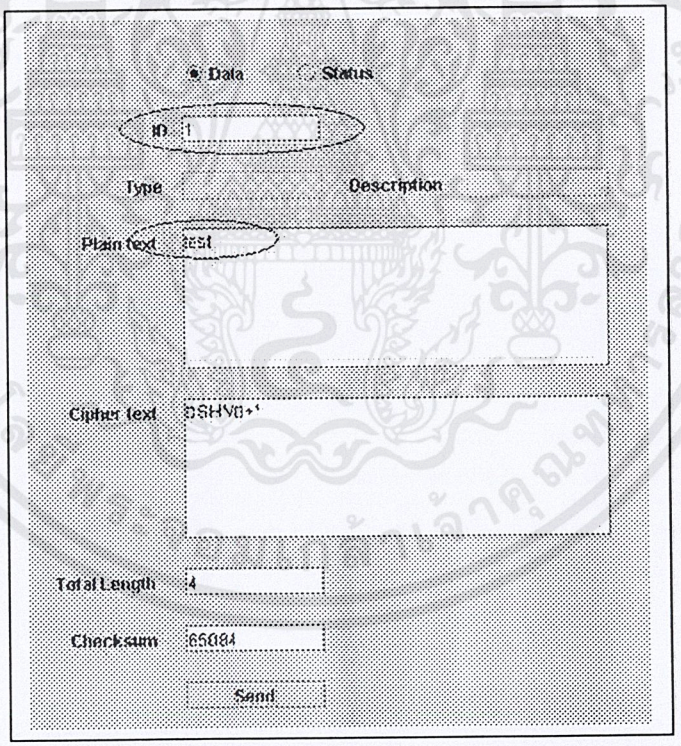
ผลการทดลอง

จากการทดลองหลังจากที่ได้ทำการส่งข้อมูลไปที่โหนดปลายทางแล้ว ไม่มีการตอบกลับหรือ Acknowledge จากโหนดปลายทางภายในระยะเวลาที่กำหนดโปรแกรมจะทำการแจ้งข้อความแสดงข้อผิดพลาดดังรูปที่ 5.16 (ค)

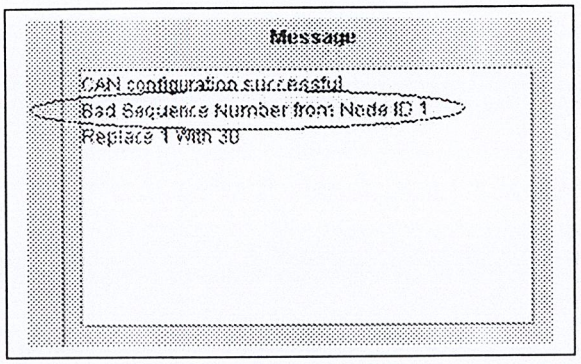
5.3.7 การตรวจสอบ Sequence ผิดปกติ



(ก)



(ข)



(ค)

รูปที่ 5.17 การตรวจสอบ Sequence ผิดปกติ

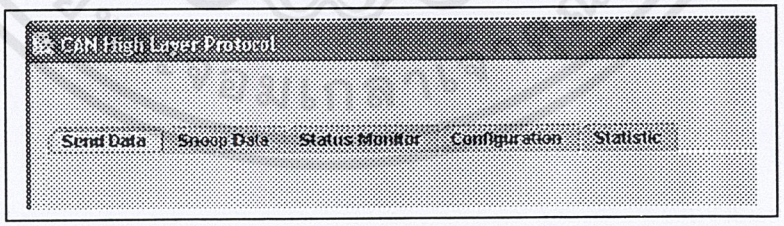
ขั้นตอนการทดลอง

1. ทำการสื่อสารตามปกติ จากนั้นปิดโปรแกรมจำลอง
2. รันโปรแกรมจำลองใหม่แล้วทำการเลือกที่แท็ป Send Data ดังรูปที่ 5.17 (ก)
3. กรอก ID และข้อมูลที่ต้องการส่งจากนั้นกดปุ่ม Send เพื่อส่งข้อมูลดังรูปที่ 5.17 (ข)

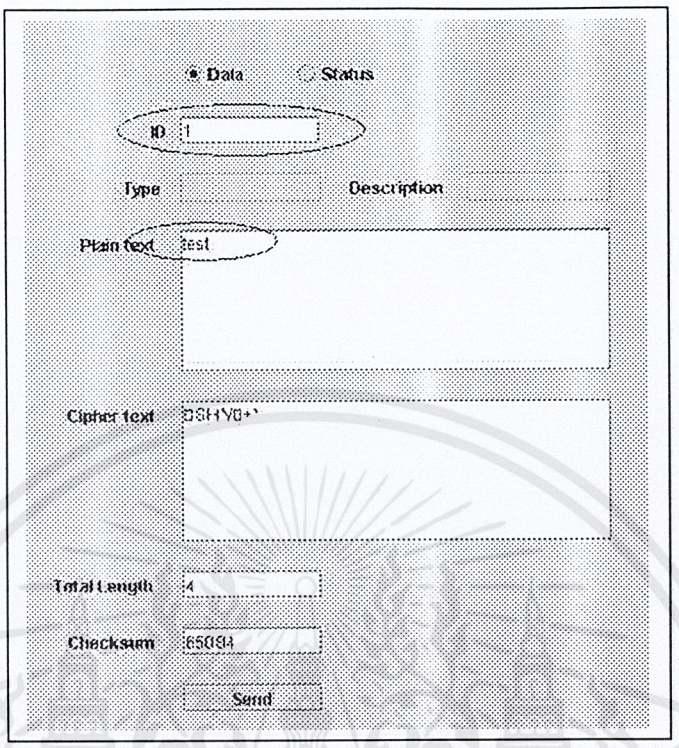
ผลการทดลอง

หลังจากที่ได้ทำการปิดและรันโปรแกรมใหม่แล้วทำการส่งข้อมูล โหนดที่ได้รับข้อมูลทำการตรวจสอบ Sequence Number แล้วพบว่ามีหมายเลขที่ตรงกับ Sequence Number ที่ได้รับครั้งล่าสุดโปรแกรมจะทำการแสดงข้อความแสดงข้อผิดพลาดดังรูปที่ 5.17 (ค)

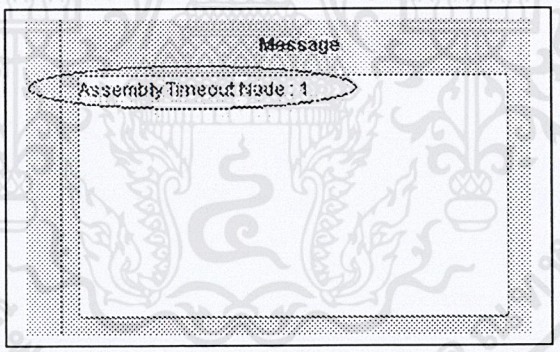
5.3.8 การทดลองนำส่งแพ็กเก็ตที่ไม่มีจุดสิ้นสุด



(ก)



(ข)



(ค)

รูปที่ 5.18 การตรวจสอบแพ็กเก็ตที่ไม่มีจุดสิ้นสุด

การทดลอง

1. ทำการแก้ไขโปรแกรมเพื่อเซ็ตให้บิต More Frame เป็น '1' ในเฟรมสุดท้ายของแพ็กเก็ต
2. รันโปรแกรมจำลองแล้วทำการเลือกที่แท็ป Send Data ดังรูปที่ 5.18 (ก)
3. กรอก ID และข้อมูลที่ต้องการส่งจากนั้นกดปุ่ม Send เพื่อส่งข้อมูลดังรูปที่ 5.18 (ข)

ผลการทดลอง

หลังจากที่ส่งแพ็กเก็ตที่ไม่มีจุดสิ้นสุดออกไปยังโหนดปลายทาง โหนดปลายทางจะทำการจับเวลาที่ใช้ในการประกอบเฟรมเป็นแพ็กเก็ต ถ้าเฟรมสุดท้ายขาดหายไปจะทำให้ไม่

สามารถประกอบข้อมูลได้ เมื่อถึงเวลาที่กำหนดโปรแกรมจะการละทิ้งแพ็กเก็ตนั้นแล้วแสดงข้อความแสดงข้อผิดพลาดดังรูปที่ 5.18 (ค)

บทที่ 6
บทสรุป

ปริญญานิพนธ์ฉบับนี้ได้ทำการออกแบบและพัฒนาเครือข่ายสำหรับระบบควบคุม โดยเริ่มจากการศึกษาการทำงานของ CAN โพรโตคอล, ทำการศึกษาและกำหนด CAN High Layer Protocol จาก CAN High Layer Protocol ต่างๆที่มีอยู่ได้แก่ CANopen, DeviceNet, CAN Kingdom, OSEK/VDX, SDS และ J1939 จากนั้นทำการออกแบบเฟรมข้อมูลให้สนับสนุนตามข้อกำหนดของ CAN High Layer Protocol ที่ได้กำหนดไว้ การกำหนด CAN High Layer Protocol นี้จะเป็นการประยุกต์ใช้งานจากพื้นฐานการทำงานในชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data link Layer) ซึ่งจะเพิ่มความสามารถของระบบที่นำ CAN ไปใช้งาน

จากการทดลองเครือข่ายที่ได้พัฒนาขึ้นผลปรากฏว่ามีความสามารถเพิ่มเติมจาก CAN โพรโตคอลเดิมในหลายๆด้านได้แก่ การกำหนด ID ให้กับ Message เพื่อให้รับและส่งข้อมูลกันได้อย่างมีประสิทธิภาพ, ออกแบบให้สามารถส่งข้อมูลได้มากกว่า 8 ไบต์โดยใช้เทคนิคที่เรียกว่า Message on ID ทำให้ระบบมีความยืดหยุ่นตัวสูงขึ้น, การรายงานสถานะอุปกรณ์ในเครือข่ายซึ่งช่วยช่วยอำนวยความสะดวกให้ผู้ดูแลระบบในการตรวจสอบดูแลและสั่งงานผ่านระบบควบคุม, การตรวจสอบการนำส่งข้อมูลซึ่งช่วยให้ผู้ใช้งานตรวจสอบได้ว่าข้อมูลถูกส่งถึงปลายทางจริงสำหรับระบบที่ต้องการความน่าเชื่อถือ, สนับสนุนการเสริมสำรองเพื่อเพิ่มความน่าเชื่อถือและความยืดหยุ่นของระบบและเพิ่มความปลอดภัยในการรับส่งข้อมูล

ปริญญานิพนธ์นี้ถูกจัดทำขึ้นมาโดยมีประโยชน์กับทางผู้จัดทำได้เข้าใจถึงการออกแบบโพรโตคอลภาพใต้ข้อจำกัดต่างๆและยังมีประโยชน์กับผู้ที่สนใจอ่านปริญญานิพนธ์ฉบับนี้ โดยจะเป็นแนวทางในการนำ CAN โพรโตคอลไปประยุกต์ใช้งานจริง

6.1 ปัญหาและอุปสรรค

6.1.1 ใช้เวลาในการศึกษา CAN โพรโตคอลนาน

6.1.2 ไอซีควบคุม CAN เบอร์ MCP2510 ไม่เป็นที่นิยมมากนักจึงทำให้หาตัวอย่างการใช้งานได้ยาก

6.1.3 เครื่องมือที่ใช้ในการแปลงจากภาษาซีไปเป็นภาษาแอสแซมบลี้ยังมีข้อผิดพลาดทำให้การเขียนโปรแกรมบนไมโครคอนโทรลเลอร์ใช้เวลานาน

6.2 ขอบเขตและข้อจำกัดของโครงงาน

6.2.1 โปรแกรมที่ได้ทำการพัฒนาเป็นเพียงเครื่องมือที่ใช้ในการทดสอบการทำงานของโพรโตคอลเท่านั้นไม่สามารถนำไปใช้งานได้จริง

6.2.2 เฟรมที่ออกแบบใช้งานได้กับ CAN โพรโตคอลเท่านั้น

6.3 แนวทางการประยุกต์และพัฒนา

6.3.1 นำโพรโตคอลที่ได้ทำการออกแบบไปพัฒนาและไปประยุกต์ใช้งานจริง

6.3.2 ปรับปรุงด้านความปลอดภัยและการยืนยันตน

บรรณานุกรม

หนังสืออ้างอิง

[1] อุดมศักดิ์ ใช้ศรีทอง. 2545. "CAN บัสข้อมูลอัจฉริยะ สำหรับงานอุตสาหกรรมและยานยนต์ ตอนที่ 1 โครงสร้างและลักษณะทั่วไป". **เซมิอิเล็กทรอนิกส์**. (231) : 155 –159.

[2] อุดมศักดิ์ ใช้ศรีทอง. 2545. "CAN บัสข้อมูลอัจฉริยะ สำหรับงานอุตสาหกรรมและยานยนต์ ตอนที่ 2 โปรโตคอลในการส่งข้อมูล". **เซมิอิเล็กทรอนิกส์**. (234) : 176 –183.

[3] อุดมศักดิ์ ใช้ศรีทอง. 2545. "CAN บัสข้อมูลอัจฉริยะ สำหรับงานอุตสาหกรรมและยานยนต์ ตอนจบ การสร้างบอร์ดอินเตอร์เฟส CAN". **เซมิอิเล็กทรอนิกส์**. (237) : 190 –198.

[4] วรพจน์ กรแก้ววัฒนกุล และชัยวัฒน์ ลิ้มพรจิตรวิไล. 2540. **เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช**. กรุงเทพฯ : อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด

[5] วีระศักดิ์ ซึงถาวร. 2543. "Java Programming Volume 1". **ซีเอ็ดยูเคชั่น.**

[6] วีระศักดิ์ ซึงถาวร. 2545. "Java Programming Volume 2". **ซีเอ็ดยูเคชั่น.**

[7] Deitel. 2000. "Java How to program". third edition. **Prentice Hall.**

เว็บไซต์อ้างอิง

[8] Robert Bosch GmbH. "CAN **Homepage**" [Online]. Available : http://www.can.bosch.com.

[9] CIA. **"CANopen protocol – Introduction"** [Online]. Available : http://www.can-cia.org/canopen/

[10] Warwick Control Technology. **"Technical - Introduction to CAN"** [Online]. Available : http://www.warwickcontrol.com/TICAN.htm

[11] Microchip. **"MCP2510 Datasheet"** [Online]. Available : http://www.microchip.com/download/lit/pline/analog/interfce/can/21291e.pdf

[12] Taxus Instruments. **"UC5350 Datasheet"** [Online]. Available : http://www-s.ti.com/sc/ds/uc5350.pdf