การพัฒนาชุดทดลองสำหรับระบบ CAN

DEVELOPMENT OF BOARD FOR CAN SYSTEM

นายจรัสพงศ์ เขมาสิทธิ์

นายสุรพัฒน์ ศิริอุปถัมภ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมการวัดคุม

ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

DEVELOPMENT OF BOARD FOR CAN SYSTEM

JARASPONG KHAEMASITH

SURAPHAT SIRI-UPATHAM

A THESIS SUBMITTED INPARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2001

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	การพัฒนาชุดทดลองสำหรับระบบ CAN			
	DEVELOPMENT OF BOARD FOR CAN SYSTEM			
นักศึกษาผู้จัดทำ	นายจรัสพงศ์	เขมาสิทธิ์	รหัสประจำตัว	42015426
	นายสุรพัฒน์	ศิริอุปถัมภ์	รหัสประจำตัว	42015458
ปริญญา	วิศวกรรมศาสตรบัณฑิต			
สาขาวิชา	วิศวกรรมการวัดคุม			
ปีการศึกษา	2544			

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
ผศ. สักรียา ชิตวงศ์	

วัน/เดือน/ปี ที่สอบ วันพุธที่ 22 พฤษภาคม พ.ศ. 2545

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(ผศ. ประสิทธิ์ จุลเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	การพัฒนาชุดทดลองสำหรับระบบ CAN
	DEVELOPMENT OF BOARD FOR CAN SYSTEM
นักศึกษาผู้จัดทำ	นายจรัสพงศ์ เขมาสิทธิ์
	นายสุรพัฒน์ ศิริอุปถัมภ์
อาจารย์ที่ปรึกษา	ผศ.สักรียา ชิตวงศ์
ปีการศึกษา	2544

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เสนอเกี่ยวกับมาตรฐานและข้อกำหนดในโปรโตคอลของ CAN โดยรวมถึงการออกแบบและการสร้างชุดทดลองสำหรับทดสอบฟังก์ชันการทำงานของ CAN เพื่อให้เข้าใจถึงหลักการทำงานที่เกี่ยวกับโปรโตคอลของ CAN ได้ชัดเจนยิ่งขึ้น และเป็นข้อมูลให้แก่ผู้ที่มีความสนใจสามารถนำไปพัฒนา สำหรับการใช้ในงานควบคุมอัตโนมัติที่มีความซับซ้อนมากได้ โดยชุดทดลองจะถูกออกแบบให้มีลักษณะเป็นโมดูลของ CAN ที่มีความยืดหยุ่นสามารถนำไปประยุกต์ใช้งานได้ง่าย ซึ่งอาจนำไปต่อลงบนบอร์ดของ MCS-51 ทั่วไปได้ เพื่อความสะดวกในการเรียนรู้และพัฒนาต่อไป

Thesis Title	Development of Board for CAN System
Authors	Mr. Jaraspong Khaemasith
	Mr. Suraphat Siri-upathum
Thesis Advisor	Asst. Prof. Sakreya Chitwong
Year	2001

ABSTRACT

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

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี เพราะได้รับคำปรึกษาและความอนุเคราะห์จาก ผศ.สักรียา ชิตวงศ์ ซึ่งเป็นอาจารย์ผู้ควบคุมปริญญานิพนธ์ ที่ให้คำแนะนำและแนวคิดที่ดี ผู้จัดทำขอขอบพระคุณอย่างสูงที่ได้ให้การสนับสนุนและคำปรึกษาในงานวิจัยจนสำเร็จตามจุดประสงค์

ขอขอบพระคุณ อาจารย์ประจำภาควิชาวิศวกรรมการวัดคุมทุกท่าน ที่เอื้อเฟื้ออุปกรณ์และเครื่องมือในการทดลอง และทุกๆ ความช่วยเหลืออันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

และที่มิเคยลืม ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ที่ให้การสนับสนุนและเป็นกำลังใจมาโดยตลอดจนปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ผู้จัดทำขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญตาราง (ต่อ)

ตารางที่ หน้า

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุจูงใจของงานวิจัย

ในปัจจุบันวงการอุตสาหกรรมมีความก้าวหน้าอย่างมาก ทำให้กระบวนการควบคุมต่างๆ มีความสลับซับซ้อนมากยิ่งขึ้น และเสี่ยงต่อความผิดพลาดที่อาจเกิดขึ้นได้ ด้วยเหตุนี้จึงมีความต้องการระบบเครือข่ายที่มีความน่าเชื่อถือ ปลอดภัย และมีประสิทธิภาพสูง เพื่อช่วยแก้ปัญหาในเรื่องการสื่อสารข้อมูลระหว่างอุปกรณ์ในระบบเครือข่าย ได้แก่ ระบบ CAN ซึ่งเป็นบัสมาตรฐานที่ถูกพัฒนาขึ้นสำหรับใช้ในระบบอิเล็กทรอนิกส์ในรถยนต์ ต่อมาจึงมีการประยุกต์ใช้ในงานอื่นๆ โดยเฉพาะอย่างยิ่งในงานควบคุมอัตโนมัติในอุตสาหกรรม มีการใช้ CAN อย่างแพร่หลายทั่วโลก แต่ในประเทศไทยยังไม่มีการนำมาใช้อย่างจริงจังและอาจไม่เคยรับทราบมาก่อน เนื่องจากยังไม่มีหนังสือหรือบทความที่ให้ความรู้ในเรื่องนี้ ดั้งนั้นจึงเกิดแนวคิดที่จะศึกษาและทดสอบฟังก์ชันการทำงาน CAN เพื่อเป็นข้อมูลและแนวทางแก่ผู้ที่สนใจนำไปประยุกต์ใช้งานด้านต่างๆ โดยเฉพาะงานควบคุมอัตโนมัติ หรือการควบคุมกระบวนการอื่นที่มีความซับซ้อนมากกว่า และหวังว่าในอนาคตอาจจะมีการนำระบบ CAN มาใช้ในงานอุตสาหกรรมของไทยอย่างแพร่หลาย

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์นี้เป็นการศึกษาค้นคว้าเกี่ยวกับหลักการทำงานของ CAN และข้อกำหนดในโปรโตคอล รวมถึงการออกแบบสร้างชุดทดลองอย่างง่ายเพื่อทดสอบฟังก์ชันการทำงานของ CAN ซึ่งออกแบบให้มีความยืดหยุ่นในการนำไปประยุกต์ใช้งานต่างๆ ได้ง่าย และศึกษาการเขียนโปรแกรมคอมพิวเตอร์ภาษา C51 ของ Keil Software เพื่อให้สามารถเขียนโปรแกรมสำหรับการทดลองได้

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์นี้จะกล่าวถึงหลักการทำงานของ CAN และข้อกำหนดในโปรโตคอล ซึ่งได้จากการศึกษาค้นคว้าจากเว็บไซต์ที่ให้ข้อมูลเกี่ยวกับ CAN นอกจากนี้ยังกล่าวถึงการออกแบบสร้างชุดทดลองจากไอซี CAN คอนโทรลเลอร์ รวมถึงการโปรแกรมเพื่อทดสอบฟังก์ชันการทำงานของ CAN ที่เกี่ยวกับการรับและส่งข้อมูลบนบัสเพื่อให้มองเห็นการทำงานได้ชัดเจนยิ่งขึ้น

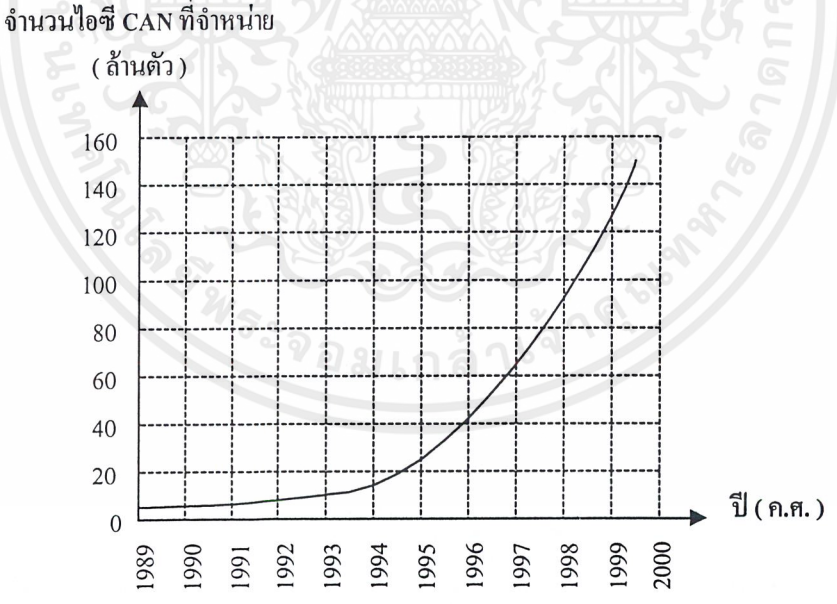
1.4 ขั้นตอนการศึกษา

ในตอนแรกจะเป็นการศึกษาค้นคว้าหาข้อมูลเกี่ยวกับ CAN ทั้งหมดเท่าที่สามารถจัดหาได้ เนื่องจากแหล่งข้อมูลแต่ละที่จะให้ข้อมูลที่แตกต่างกันแต่ก็จะอ้างอิงมาตรฐานเดียวกัน รวมถึงการศึกษาข้อกำหนดในโปรโตคอลของ CAN จากนั้นเป็นการจัดหาไอซี CAN คอนโทรลเลอร์ และอุปกรณ์ต่อร่วมเพื่อจัดทำชุดทดลอง สุดท้ายเป็นการศึกษาถึงการเขียนโปรแกรมภาษา C51 สำหรับการทดลอง

บทที่ 2
หลักการและทฤษฎีของ CAN

2.1 ความเป็นมาของ CAN

ในปัจจุบันนี้มีเทคโนโลยีที่เกี่ยวกับการสื่อสารข้อมูลนั้นเกิดขึ้นมามากมาย แต่ถ้ากล่าวถึงการสื่อสารข้อมูลระหว่างอุปกรณ์ในระบบอิเล็กทรอนิกส์แล้ว พบว่ามีบัสสื่อสารข้อมูลมาตรฐานอยู่มากมายหลายชนิด ทั้งที่ใช้ในงานอิเล็กทรอนิกส์อุตสาหกรรมและใช้งานในอุปกรณ์ทั่วไป แต่ถ้าพูดถึงการสื่อสารข้อมูลระหว่างอุปกรณ์อิเล็กทรอนิกส์ที่ใช้ในงานอุตสาหกรรม โดยเฉพาะในระบบอิเล็กทรอนิกส์สำหรับยานยนต์แล้ว พบว่ามีบัสมาตรฐานหนึ่งที่นิยมใช้กันอย่างกว้างขวางก็คือ CAN ซึ่งมีการคิดค้นและใช้งานกันมาเป็นเวลามากกว่า 10 ปีมาแล้ว โดยมีการใช้งานกันมากทั่วโลกและมีแนวโน้มว่าจะมีการใช้งานเพิ่มขึ้นเรื่อย ๆ ทราบได้จากยอดจำหน่ายไอซี CAN ภายในปี 1989 – 2000 ดังกราฟในภาพที่ 2.1 เนื่องจากมีการใช้งานแพร่หลายทั่วโลกจึงได้มีการปรับปรุงและพัฒนาชิปคอนโทรลเลอร์ที่สามารถจัดการเรื่องโปรโตคอลได้ในชิปเดียว พร้อมด้วยฟังก์ชันที่อำนวยความสะดวกในการโปรแกรมและการประยุกต์ใช้งานให้มีประสิทธิภาพมากขึ้น

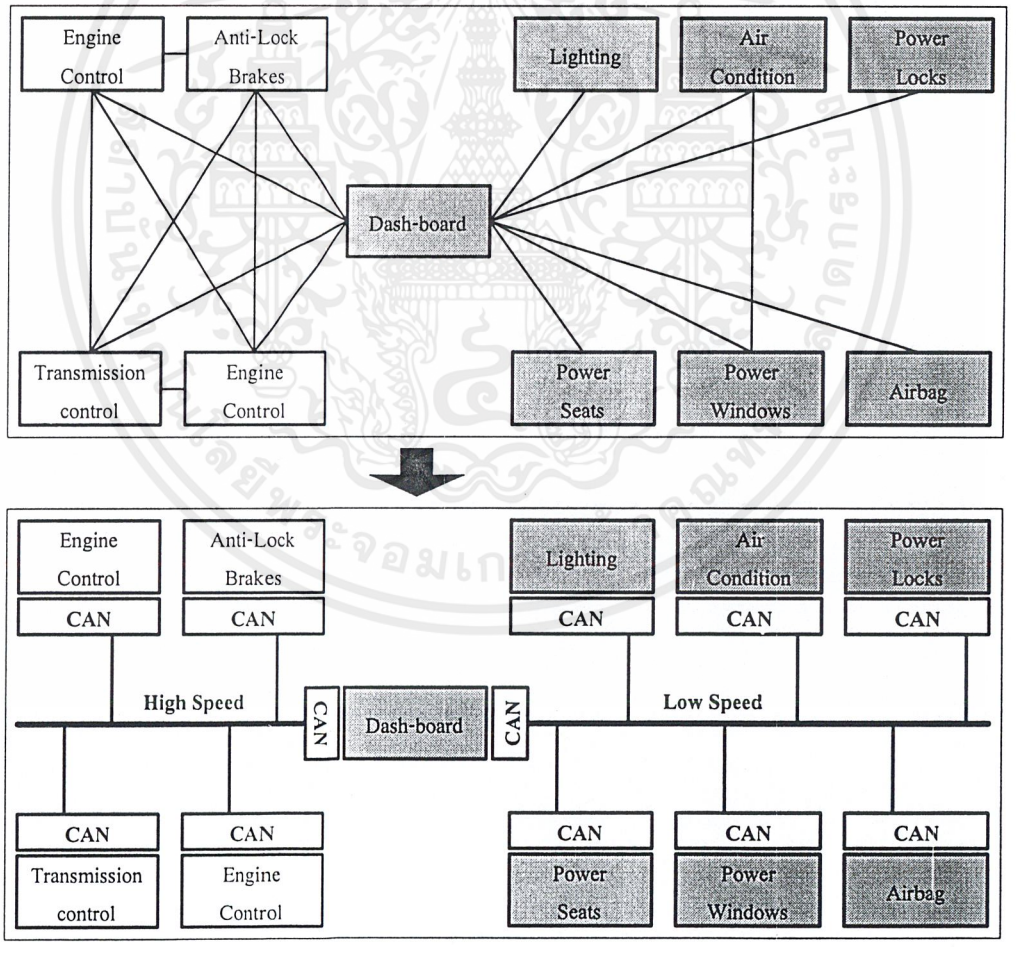


ภาพที่ 2.1 แสดงกราฟของแนวโน้มการใช้งานไอซี CAN

CAN หรือ Controller Area Network เป็นบัสสื่อสารแบบอนุกรมที่ออกแบบมาสำหรับใช้ในงานควบคุมแบบเรียลไทม์ ซึ่งจะมีจุดเด่นอยู่ที่ความสามารถในการสื่อสารข้อมูลที่เป็นเครือข่ายระหว่างอุปกรณ์ โดยไม่จำเป็นต้องมีหมายเลขที่อยู่ของโหนด (Node Addressing) อุปกรณ์ทุกตัว

ในเครือข่ายสามารถเรียกใช้งานบัสได้ (Multi-master) และสามารถร้องขอใช้งานบัสได้พร้อมกันได้หลายตัว ด้วยความเร็วในการสื่อสารที่สูงถึง 1 เมกะบิตต่อวินาที พร้อมด้วยระบบป้องกันและตรวจสอบความผิดพลาดที่มีประสิทธิภาพสูง

จุดเริ่มต้นของ CAN นั้นเริ่มจากการที่ CAN ถูกพัฒนาขึ้นมาครั้งแรกในปี 1980 โดยบริษัท Robert Bosch ในประเทศเยอรมัน สำหรับใช้งานในระบบอิเล็กทรอนิกส์ภายในรถยนต์ เนื่องจากปัญหาในการพัฒนารถยนต์รุ่นใหม่ที่ต้องการสิ่งอำนวยความสะดวกและระบบรักษาความปลอดภัยเพิ่มขึ้น จึงส่งผลให้ระบบอิเล็กทรอนิกส์ภายในรถยนต์มีความซับซ้อนมากขึ้น และจะมีโมดูลของวงจรต่าง ๆ เป็นจำนวนมากที่ต้องการสื่อสารข้อมูลระหว่างกัน ด้วยเหตุผลดังกล่าวจึงได้มีความต้องการบัสสื่อสารข้อมูลที่มีประสิทธิภาพและความน่าเชื่อถือสูงในราคาที่เหมาะสม แทนที่ระบบบัสเดิมที่มีความสลับซับซ้อนไม่สะดวกที่จะนำมาใช้และมีค่าใช้จ่ายสูง ดังนั้น CAN จึงถูกนำมาใช้สำหรับปรับปรุงและแก้ไขปัญหาในระบบเครือข่าย ดังภาพที่ 2.2

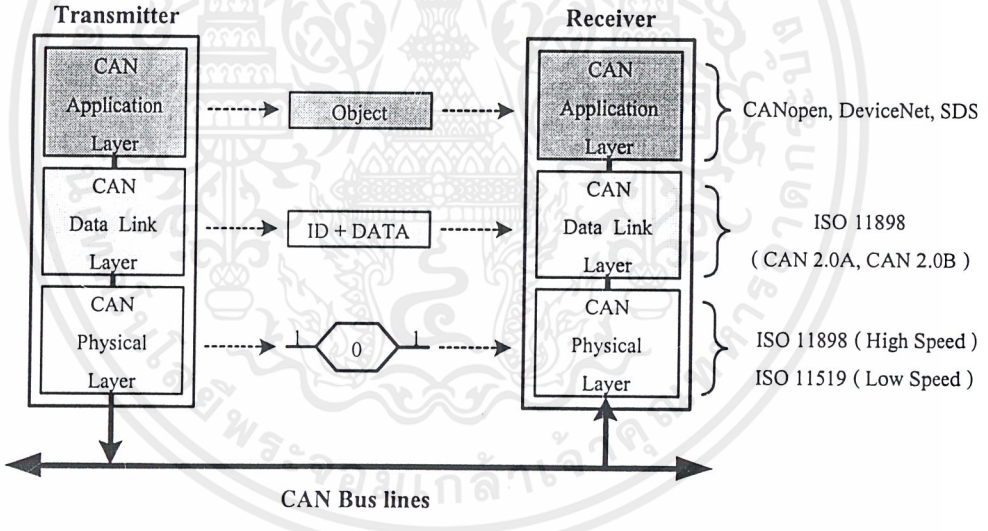


ภาพที่ 2.2 แสดงการนำ CAN มาใช้ปรับปรุงและแก้ไขปัญหาในระบบเครือข่าย

ภายหลังนอกจากจะใช้งานในอุตสาหกรรมรถยนต์แล้วยังได้มีการนำ CAN ไปประยุกต์ใช้งานในอุตสาหกรรมอื่น ๆ อีกมากมาย เนื่องจากคุณสมบัติที่โดดเด่นของบัสนี้ในด้านความน่าเชื่อถือและความยืดหยุ่น รวมทั้งความง่ายต่อการใช้งานและมีค่าใช้จ่ายต่ำนั่นเองเนื่องจากความแพร่หลายในการใช้งานของบัสนี้ ด้วยเหตุนี้จึงได้มีการกำหนด CAN ให้เป็นบัสมาตรฐานระหว่างประเทศขึ้นมา นั่นคือมาตรฐาน ISO 11898 (ความเร็วสูง) และ ISO 11519 (ความเร็วต่ำ) ซึ่งจะเป็นการรับประกันถึงความสามารถและความยอมรับใน CAN ได้เป็นอย่างดี โดยมาตรฐานนี้จะครอบคลุมข้อกำหนดการทำงานของ CAN ในสองชั้นแรกตามแบบจำลอง ISO/OSI คือ ชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data Link Layer)

2.2 โครงสร้างการทำงาน

โครงสร้างการทำงานของ CAN เมื่อเทียบกับแบบจำลอง ISO/OSI นอกจากจะประกอบด้วยสองชั้นแรกแล้วยังประกอบด้วยชั้นที่ 7 คือ ชั้นประยุกต์ใช้งาน (Application Layer) โดยจะไม่มีส่วนประกอบของชั้นที่เหลือ คือ ชั้นที่ 3-6 มีลักษณะโครงสร้างการทำงานเป็นดังภาพที่ 2.3



ภาพที่ 2.3 แสดงโครงสร้างการทำงานของ CAN

1. ชั้นที่ 1 Physical Layer โครงสร้างการทำงานในชั้นนี้จะเป็นส่วนที่เกี่ยวข้องกับส่วนประกอบทางกายภาพของบัส ได้แก่ ตัวกลางที่ใช้ส่งผ่านสัญญาณข้อมูล คอนเน็กเตอร์ และการเชื่อมต่อสายสัญญาณ รวมทั้งคุณสมบัติทางไฟฟ้าของสัญญาณข้อมูล (แรงดัน/กระแส) โดยมีมาตรฐานที่เกี่ยวข้องกับการทำงานในชั้นนี้มีอยู่ 2 มาตรฐาน คือ ISO 11519 สำหรับการสื่อสาร

ข้อมูลความเร็วต่ำ ซึ่งมีอัตราเร็วอยู่ในช่วง 5 ถึง 125 กิโลบิตต่อวินาที และ ISO 11898 สำหรับการสื่อสารข้อมูลความเร็วสูง ที่สนับสนุนความเร็วสูงสุดถึง 1 เมกะบิตต่อวินาที

2. ชั้นที่ 2 Data Link Layer โครงสร้างการทำงานในชั้นนี้จะเกี่ยวข้องกับโปรโตคอลและรูปแบบของข้อมูลที่สื่อสารกันในเชิงตรรกะ รวมถึงการป้องกันความผิดพลาดที่จะเกิดขึ้นกับข้อมูล เพื่อรับประกันว่าข้อมูลที่ได้มีความถูกต้อง โดยข้อกำหนดของการทำงานในชั้นนี้ได้รวมอยู่ในมาตรฐาน ISO 11898 ด้วย และได้มีการแก้ไขและขยายข้อกำหนดของมาตรฐานในชั้นนี้เพิ่มเติม โดยจัดทำเป็น 2 เวอร์ชัน คือ CAN 2.0A และ CAN 2.0B

3. ชั้นที่ 7 Application Layer ในชั้นนี้จะเป็นส่วนของการนำ CAN ไปประยุกต์ใช้งานบนพื้นฐานการทำงานในสองชั้นแรก ถึงแม้ว่าการทำงานในชั้นนี้ไม่ได้กำหนดรวมอยู่ในมาตรฐาน ISO 11898 ด้วย แต่มีการสร้างโปรโตคอลสำหรับการทำงานในชั้นนี้ขึ้นมา ซึ่งมีอยู่ด้วยกันหลายรูปแบบ ยกตัวอย่างโปรโตคอลในชั้นนี้ที่มีการใช้งานกัน เช่น CANopen, DeviceNet, SDS, OSEK/VDX, CAN Kingdom และ J1939 เป็นต้น โดยแต่ละโปรโตคอลจะมีหลักการทำงานอยู่บนแนวความคิดเดียวกัน

2.3 คุณสมบัติของ CAN

หลักการทำงานโดยทั่วไปของ CAN นั้นจะมีลักษณะตามข้อกำหนดในโปรโตคอล และสามารถกำหนดเป็นคุณสมบัติได้ คือ

1. มาตรฐาน ISO 11898 พัฒนาขึ้นมาในปี 1980 โดย Robert Bosch GmbH
2. โครงสร้างการทำงานมีเพียง 2 ชั้น คือ ชั้นกายภาพ (Physical Layer) และชั้นเชื่อมโยงข้อมูล (Data Link Layer) ตามแบบจำลอง ISO/OSI
3. ใช้การสื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส
4. มีการสร้างเครือข่ายแบบ Multi-master / Broadcasting
5. สามารถสนับสนุนการทำงานแบบเรียลไทม์
6. ไม่มีการจำกัดจำนวนโหนดที่ต่อร่วมบนบัสในเครือข่าย
7. ไม่มีการกำหนดหมายเลขที่อยู่กำกับโหนด แต่จะใช้การกำหนดหมายเลข ID ของข้อมูลที่ทำการส่งแทน
8. ใช้หลักการของ CSMA/CD + AMP ในการตัดสินเข้าใช้บัส
9. ตัดสินการเข้าใช้บัสที่ระดับความสำคัญข้อมูล ที่ถูกระบุด้วยหมายเลข ID ซึ่งจะใช้การตัดสินในระดับบิตข้อมูล
10. สามารถส่งข้อมูลได้ครั้งละ 0-8 ไบต์ ด้วยอัตราเร็วสูงสุดถึง 1 เมกะบิตต่อวินาที
11. ใช้การเข้ารหัสบิตข้อมูลแบบ NRZ (Non-Return-to-Zero)
12. มีระบบป้องกันและตรวจสอบความผิดพลาดที่มีประสิทธิภาพสูง

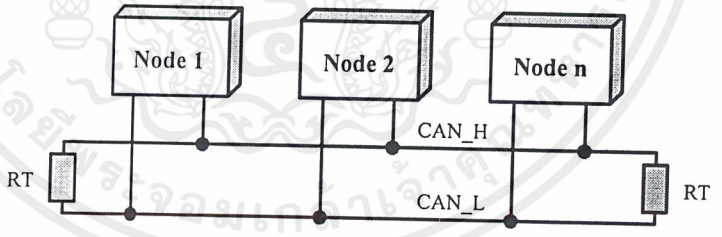
13. ตัวควบคุมโปรโตคอลสามารถบรรจุอยู่ในชิปเดียว
14. มีประสิทธิภาพและความน่าเชื่อถือสูง
15. ต้นทุนต่ำ ใช้งานสะดวก และบำรุงรักษาง่าย
16. มีการใช้งานอย่างแพร่หลายในอุตสาหกรรม และระบบอิเล็กทรอนิกส์ภายในยานยนต์

2.4 ลักษณะของสายสัญญาณที่ใช้ใน CAN

ก่อนที่จะไปดูรายละเอียดของโครงสร้างการสื่อสารข้อมูลหรือโปรโตคอลที่ใช้นั้น เพื่อให้มองเห็นภาพของ CAN มากขึ้น ต้องรู้จักว่าลักษณะทางกายภาพของ CAN นั้นเป็นอย่างไรเสียก่อน โดยเริ่มจากตัวกลางที่ใช้ส่งผ่านข้อมูลและลักษณะของสัญญาณที่ส่งออกไป

สำหรับตัวกลางที่ใช้ส่งผ่านข้อมูลใน CAN นั้นมีอยู่ด้วยกันหลายรูปแบบ ตั้งแต่การใช้สายสัญญาณ 2 เส้น การใช้สายสัญญาณเส้นเดียว การใช้สายไฟเบอร์ออปติก หรือแม้แต่การใช้สายสัญญาณ 2 เส้นแบบตีเกลียว (Twisted-pair Cable) ซึ่งในที่นี้จะอ้างอิงการส่งผ่านข้อมูลโดยการใช้ตัวกลางในรูปแบบนี้

การส่งผ่านข้อมูลใน CAN จะมีรูปแบบของการส่งข้อมูลเป็นแบบอนุกรมที่ใช้สายสัญญาณเพียง 2 เส้น ซึ่งมีลักษณะการเชื่อมต่อสายสัญญาณ (Topology) ดังภาพที่ 2.4 โดยอุปกรณ์ทุกตัวจะต่ออยู่บนสายสัญญาณคู่เดียวกัน และมีเทอร์มิเนชันอิมพีแดนซ์ (Termination Impedance) สำหรับปิดปลายคู่สายสัญญาณทั้งสองข้าง

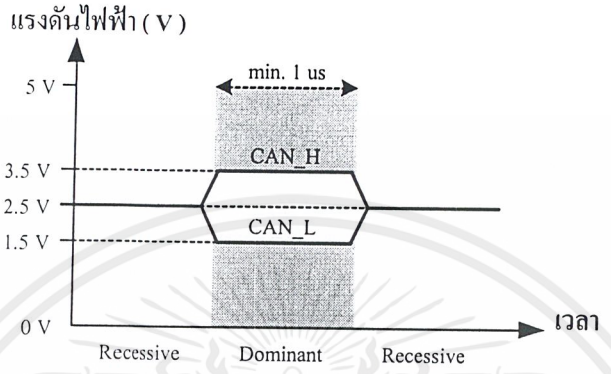


ภาพที่ 2.4 แสดงลักษณะการเชื่อมต่ออุปกรณ์ใน CAN

สัญญาณข้อมูลจะใช้วิธีการส่งแบบสัญญาณแรงดันผลต่าง (Differential Voltage Signal) สถานะของสัญญาณในสายส่งได้จากการเปรียบเทียบระดับแรงดัน หรือศักย์ไฟฟ้าระหว่างสายสัญญาณทั้งสองเส้น คือ CAN_H (ศักย์สูง) และ CAN_L (ศักย์ต่ำ) ส่วนสถานะของสัญญาณข้อมูลในสายส่ง CAN จะมีอยู่ 2 สถานะ คือ สถานะรีเซสสีฟ (Recessive) และสถานะโดมิเนนต์ (Dominant) ระดับแรงดันไฟฟ้าของสัญญาณในแต่ละเส้น (เทียบกับกราวด์) และผลต่างแรงดันไฟฟ้าระหว่างคู่สายสัญญาณในแต่ละสถานะจะเป็นดังภาพที่ 2.5

1. สถานะรีเซสสีฟ ในสถานะนี้จะมีผลต่างของแรงดันระหว่าง CAN_H และ CAN_L เป็น 0 โวลต์ (หรือ ≤ 0.5 โวลต์) ซึ่งจะแทนข้อมูลที่มีลอจิกเป็น '1'

2. สถานะโดมิแนนต์ ในสถานะนี้จะมีผลต่างของแรงดันระหว่าง CAN_H และ CAN_L เป็น 2 โวลต์ (หรือ ≥ 0.9 โวลต์) ซึ่งจะแทนข้อมูลที่มีลอจิกเป็น '0'

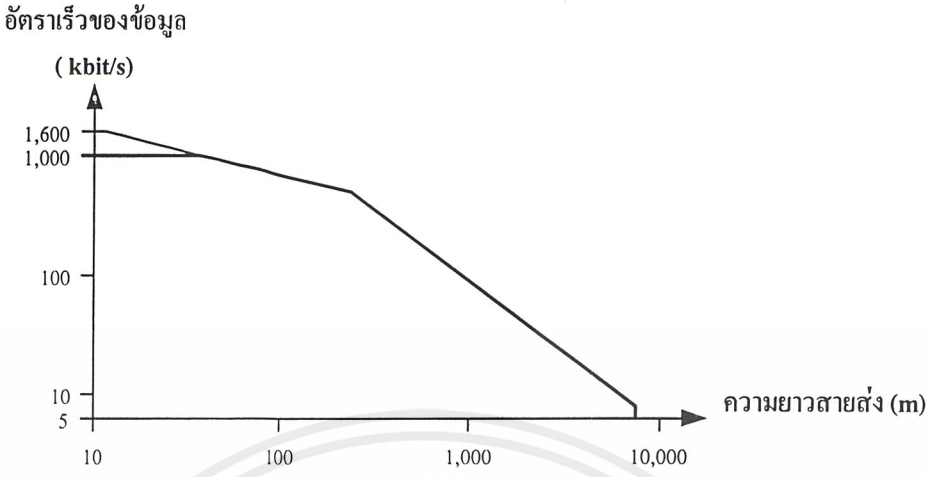


ภาพที่ 2.5 แสดงลักษณะของสัญญาณข้อมูลในแต่ละสถานะ

2.5 อัตราเร็วในการส่งข้อมูล

การส่งผ่านข้อมูลโดยใช้ผลต่างแรงดันของคู่สายสัญญาณมีข้อดีคือ จะช่วยลดการรบกวนอันเนื่องมาจากผลของ EMI (Electromagnetic Interference) ลงได้เป็นอย่างมาก ทำให้สามารถส่งสัญญาณได้ในอัตราเร็วที่สูงขึ้นและได้ระยะทางที่ไกลมากขึ้น โดยที่อัตราเร็วในการส่งข้อมูลและระยะทางที่ส่งได้นั้นจะแปรผกผันกันดังกราฟในภาพที่ 2.6 ตามมาตรฐาน ISO 11898 กล่าวคือ อัตราเร็วในการส่งข้อมูลสูงสุดคือ 1 เมกะบิตต่อวินาที ที่ความยาวของสายส่งไม่เกิน 40 เมตร สำหรับค่าความต้านทานและขนาดของสายส่ง รวมทั้งขนาดของเทอร์มิเนชันอิมพีแดนซ์ที่แนะนำให้ใช้ สามารถดูได้จากตารางที่ 2.1 ยกตัวอย่างเช่น ที่อัตราเร็วข้อมูล 1 เมกะบิตต่อวินาที ควรใช้สายส่งที่มีความต้านทานขนาด 70 มิลลิโอห์มต่อเมตร และมีขนาดเส้นผ่านศูนย์กลางของสายส่งมากกว่า 0.25 ตารางมิลลิเมตร โดยต่อเทอร์มิเนชันอิมพีแดนซ์ที่มีค่า 124 โอห์มไว้ที่ปลายของสายส่งทั้งสองด้าน

นอกจากนี้ยังมีข้อควรระมัดระวังที่สำคัญในการต่อสายสัญญาณก็คือ ในกรณีที่มีการต่อสายสัญญาณพ่วงจากบัสกลาง สายพ่วงดังกล่าวควรมีความยาวไม่เกิน 2 เมตร เมื่อมีอัตราเร็วในการส่งข้อมูลเป็น 250 กิโลบิตต่อวินาที และไม่ควรเกิน 30 เซนติเมตร เมื่อมีอัตราเร็วในการส่งข้อมูลสูงกว่านั้น ทั้งนี้ความยาวของสายพ่วงทุกเส้นรวมกันแล้วไม่ควรเกิน 30 เมตร



ภาพที่ 2.6 แสดงกราฟความสัมพันธ์ระหว่างอัตราเร็วข้อมูลกับระยะทางที่ส่งได้

ตารางที่ 2.1 แสดงความสัมพันธ์ของคุณสมบัติสายส่งกับอัตราเร็วข้อมูล

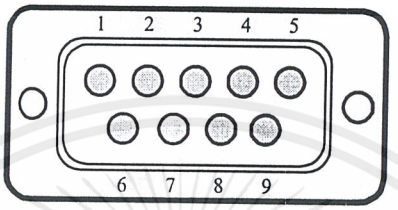
ความยาวสายส่ง	คุณสมบัติสายส่ง		เทอร์มิเนชันอิมพีแดนซ์	อัตราเร็วข้อมูลสูงสุด
	ความต้านทาน	ขนาด		
0 – 40 m	70 mΩ/m	0.25 – 0.34 mm^2 AWG23, AWG22	124 Ω (1%)	1 Mbit / s ที่ 40 m
40 – 300 m	< 60 mΩ/m	0.34 – 0.6 mm^2 AWG22, AWG20	127 Ω (1%)	500 kbit / s ที่ 100 m
300 – 600 m	< 40 mΩ/m	0.5 – 0.6 mm^2 AWG20	150 – 300 Ω	100 kbit / s ที่ 500 m
600 – 1000 m	< 26 mΩ/m	0.75 – 0.8 mm^2 AWG18	150 – 300 Ω	50 kbit / s ที่ 1000 m

2.6 มาตรฐานของคอนเน็กเตอร์ใน CAN

สำหรับมาตรฐานของขั้วต่อสายสัญญาณหรือคอนเน็กเตอร์นั้น รูปแบบหนึ่งที่นิยมใช้กันมากก็คือ คอนเน็กเตอร์แบบ DB9 โดยมีการเรียงสัญญาณในแต่ละขาดังภาพที่ 2.7 สัญญาณที่ใช้ในการสื่อสารข้อมูลจะอยู่ที่ขา 2 (CAN_L) และขา 7 (CAN_H) โดยที่สัญญาณกราวด์ของสัญญาณข้อมูลทั้งสองเส้นนี้จะอยู่ที่ขา 3 (CAN_GND) นอกจากนี้ยังมีขาอื่น ๆ สำหรับการใช้งานเพิ่มเติมซึ่งจะใช้หรือไม่ก็ได้ ได้แก่ ขา 5 สำหรับต่อสายชิลด์ (CAN_SHLD) ขา 6 สำหรับ

ต่อกราวด์ (GND) และขา 9 สำหรับต่อไฟเลี้ยงจากภายนอก (CAN_V+) ส่วนขาอื่น ๆ ที่เหลือไม่ได้ใช้งาน

นอกจากนี้ในแต่ละโปรโตคอลการใช้งาน (ในชั้นประยุกต์ใช้งาน) ยังได้มีการกำหนดรูปแบบของคอนเน็กเตอร์ในแบบต่าง ๆ เพิ่มขึ้นมาอีก ซึ่งสามารถดูรายละเอียดเพิ่มเติมได้จากข้อกำหนดของโปรโตคอลนั้น ๆ



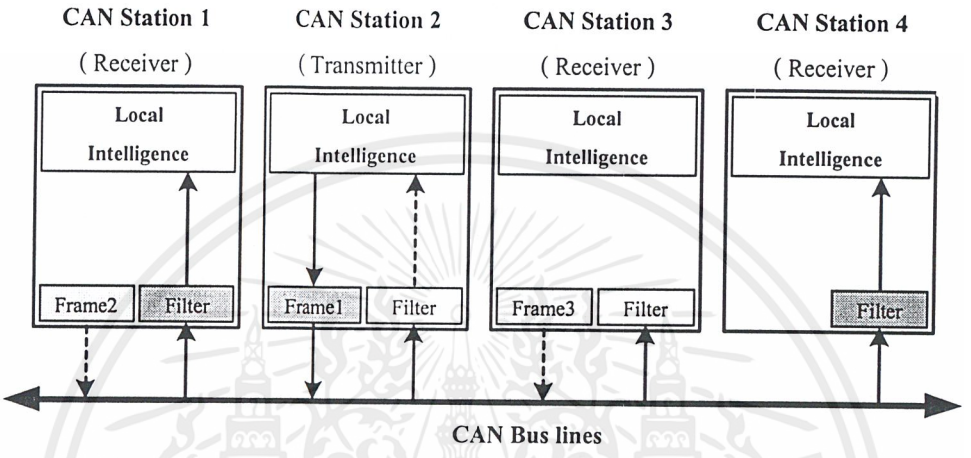
ขา	สัญญาณ	คำอธิบาย
1	-	สำรองไว้ ไม่ได้ใช้งาน
2	CAN_L	สัญญาณข้อมูลศักย์ต่ำ
3	CAN_GND	กราวด์ของสัญญาณข้อมูล
4	-	สำรองไว้ ไม่ได้ใช้งาน
5	(CAN_SHLD)	ชีลด์ (option)
6	GND	กราวด์ (option)
7	CAN_H	สัญญาณข้อมูลศักย์สูง
8	-	สำรองไว้ ไม่ได้ใช้งาน
9	(CAN_V+)	ไฟเลี้ยงจากภายนอก (option)

ภาพที่ 2.7 แสดงการเรียงสัญญาณในคอนเน็กเตอร์ที่ใช้ใน CAN

2.7 การบริการและการสื่อสารข้อมูลของ CAN

รูปแบบการสื่อสารข้อมูลของ CAN เป็นแบบอนุกรมอะซิงโครนัส โดยที่อุปกรณ์แต่ละตัวหรือแต่ละโหนดสามารถส่งข้อมูลหรือข้อความไปยังโหนดอื่น ๆ ได้ เหตุที่เรียกว่าเป็นการสื่อสารข้อมูลแบบอะซิงโครนัสเนื่องจากไม่มีการส่งสัญญาณนาฬิกาเพื่อใช้ในการเข้าจังหวะ แต่จะอาศัยการเข้าจังหวะจากจุดเริ่มต้นของแต่ละข้อความที่ส่งไป ส่วนโปรโตคอลในชั้นเชื่อมโยงข้อมูลจะมีหลักการสำคัญที่ทำให้ CAN เป็นบัสข้อมูลที่ทำงานได้อย่างมีประสิทธิภาพ คือ การสร้างเครือข่ายแบบ Multi-master และใช้การสื่อสารแบบ Broadcasting กล่าวคือ เมื่อระบบเครือข่ายอยู่ในช่วงบัสว่างจะส่งผลให้ทุกโหนดมีสิทธิในการเข้าใช้บัส และสามารถเริ่มต้นการส่งข้อมูลของตัวเองได้ โดยโหนดที่ทำการส่งด้วยข้อมูลที่มีลำดับความสำคัญสูงสุดจะได้รับสิทธิในการเข้าใช้บัสก่อน และทำการส่งข้อมูลในลักษณะของการกระจายข่าวสาร ซึ่งทุกโหนดที่อยู่บนบัสนั้นสามารถรับข้อมูล

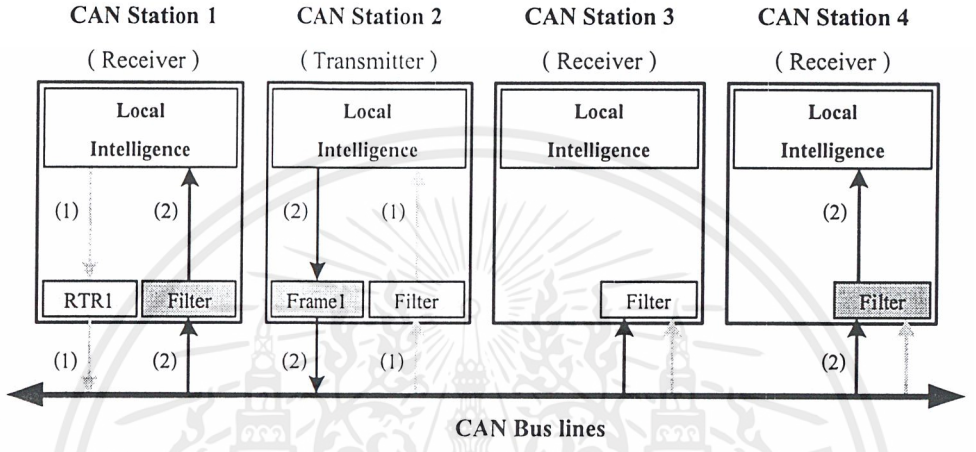
ดังกล่าวของโหนดผู้ส่งได้ ภายหลังจากที่ได้รับข้อมูลแล้วจะเป็นหน้าที่ของแต่ละโหนดที่จะต้องตรวจสอบเองว่าเป็นข้อมูลที่ต้องการหรือไม่ โดยอาศัยฟิลเตอร์เพื่อช่วยการตัดสินใจยอมรับข้อมูลดังกล่าวซึ่งจะทำหน้าที่สกัดกั้นข้อมูลที่ไม่ต้องการทิ้งไป และยอมรับเฉพาะข้อมูลที่ต้องการเท่านั้น โดยดูจากค่ารหัสประจำตัว (Identifier) ของข้อมูลหรือข้อความนั้น ลักษณะของการส่งข้อมูลแบบ Broadcasting จะเป็นดังภาพที่ 2.8



ภาพที่ 2.8 แสดงรูปแบบการติดต่อสื่อสารของ CAN

ในส่งข้อมูลของ CAN นั้นจะไม่มีการระบุที่อยู่ของโหนดผู้รับและโหนดผู้ส่ง แต่จะใช้การระบุด้วยค่ารหัสประจำตัวของข้อมูลหรือข้อความแทน เพื่อเป็นการบ่งบอกถึงชนิดและลำดับความสำคัญของข้อมูลนั้น โดยอาศัยวิธีการส่งข้อมูลแบบ Broadcasting ทำให้เมื่อมีโหนดใดโหนดหนึ่งทำการส่งข้อมูลออกที่บัส แต่ละโหนดจะได้รับการแจ้งว่ามีการส่งข้อมูลเกิดขึ้นและตัดสินใจกันเองว่าจะรับข้อมูลที่ส่งมานั้นหรือไม่ ยกตัวอย่างเช่น โหนด A ทำการส่งข้อมูลเป็นค่าที่วัดได้จากเซนเซอร์ด้วยค่ารหัสประจำตัวของข้อมูลเท่ากับ 123 ออกไปที่บัส หากโหนดใดต้องการใช้ข้อมูลดังกล่าวก็จะทำการรับข้อมูลนี้ไว้ ซึ่งจะเห็นได้ว่าการส่งข้อมูลด้วยวิธีนี้ไม่จำเป็นต้องระบุหมายเลขที่อยู่หรือรหัสประจำตัวของโหนดทั้งโหนดผู้รับและโหนดผู้ส่ง ทำให้มีข้อดีในการเพิ่มโหนดใหม่เข้ามาในระบบได้โดยไม่จำเป็นต้องทำการโปรแกรมทุกโหนดใหม่ เพื่อให้รับรู้ว่ามีการเพิ่มโหนดเข้ามา ในขณะเดียวกันโหนดที่เพิ่มเข้ามาใหม่นี้ก็สามารถรับข้อมูลที่มีการส่งกันในบัสได้ทันทีโดยพิจารณาจากค่ารหัสประจำตัวของข้อมูลที่ส่งมานั่นเอง นอกจากการที่แต่ละโหนดจะรอรับข้อมูลจากโหนดที่ต้องการส่งแล้ว ซึ่งบางทีอาจต้องรอเป็นเวลานานกว่าจะได้รับข้อมูลที่ต้องการ ดังนั้นแต่ละโหนดยังสามารถเป็นฝ่ายร้องขอข้อมูลจากโหนดอื่นได้ด้วย เพื่อให้โหนดที่มีข้อมูลดังกล่าวส่งข้อมูลนั้นมาทันที โดยเรียกการร้องขอข้อมูลแบบนี้ว่า Remote Transmission Requests (RTR) ซึ่งเป็นเหมือนกับคำถามให้กับทุกโหนดบนบัส โดยทำการส่งด้วยค่ารหัสประจำตัวของข้อมูลที่

ต้องการออกไป หากโหนดใดมีข้อมูลดังกล่าวก็จะส่งข้อมูลนั้นตอบกลับมา และนอกจากโหนดที่ร้องขอข้อมูลนี้แล้วโหนดอื่น ๆ ที่เหลือก็สามารถที่จะรับข้อมูลดังกล่าวได้ด้วยถ้ามีความสนใจในค่ารหัสประจำตัวของข้อมูลนี้เช่นเดียวกัน การสื่อสารแบบนี้ทำให้เกิดข้อดี คือ ข้อมูลบางอย่างที่ไม่ได้มีการใช้งานเป็นประจำก็ไม่จำเป็นต้องส่งออกมาอยู่ตลอดเวลา จึงรอให้มีการร้องขอข้อมูลนั้นมาก่อนค่อยส่งข้อมูลนั้นออกไป ซึ่งจะช่วยลดปริมาณการใช้บัสลงได้ ดังภาพที่ 2.9



ภาพที่ 2.9 แสดงวิธีการร้องขอข้อมูลของ CAN

2.8 การตัดสินเข้าใช้บัสจากหลายโหนด

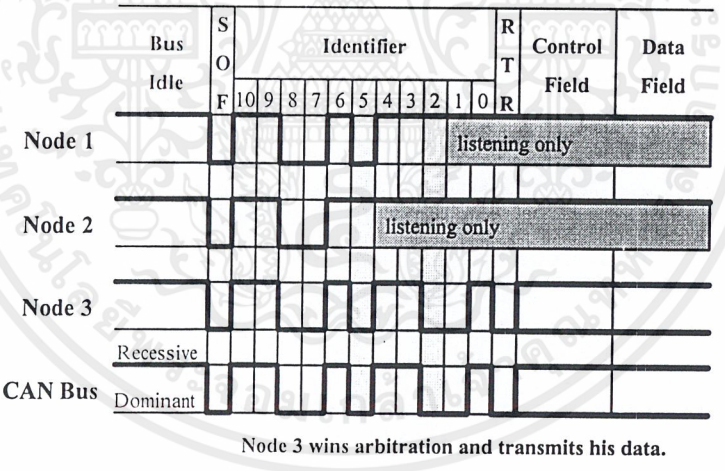
จุดเด่นของโปรโตคอลการส่งข้อมูลที่ใช้ใน CAN อย่างหนึ่ง คือ การยินยอมให้เข้าใช้บัสจากหลายโหนดได้พร้อมกัน ในกรณีที่มีมากกว่าหนึ่งโหนดนั้นต้องการส่งข้อมูลในเวลาเดียวกัน จึงจำเป็นต้องมีการตัดสินชี้ขาดว่าโหนดใดจะมีสิทธิเข้าใช้บัสในเวลานั้นเพียงโหนดเดียว ในขณะที่โหนดอื่นจะต้องหยุดทำการส่ง และรอเวลาที่จะส่งข้อมูลนั้นอีกครั้งในภายหลังเมื่อบัสว่างไม่ได้ถูกใช้งานแล้ว โดยการจัดการควบคุมการเข้าใช้บัสใน CAN จะใช้วิธีการของ CSMA/CD + AMP (Carrier Sense - Multiple Access with Collision Detection and Arbitration on Message Priority) กล่าวคือ เมื่อมีโหนดใดต้องการเข้าใช้บัสจะต้องตรวจสอบจนพบว่าบัสว่างไม่ได้ถูกใช้งานอยู่เป็นระยะเวลาหนึ่งจึงสามารถเริ่มทำการส่งข้อมูลออกไปได้ ซึ่งในขณะที่บัสว่างอยู่นี้แต่ละโหนดมีสิทธิที่จะเข้าใช้บัสด้วยโอกาสที่เท่ากัน แต่ถ้าหากมีการส่งออกมาพร้อมกันแล้วโหนดจะทราบได้ว่ามีการชนกันของข้อมูลเกิดขึ้น และจะดำเนินการแก้ไขการชนกันของข้อมูลโดยอาศัยค่าระดับความสำคัญของข้อมูลเป็นตัวตัดสินว่าโหนดใดจะมีสิทธิได้เข้าใช้บัส ซึ่งค่าระดับความสำคัญของข้อมูลจะถูกระบุอยู่ในค่ารหัสประจำตัวของข้อมูลนั้น สำหรับการตัดสินชี้ขาดนี้จะเกิดขึ้นโดยไม่เป็นการ

ทำลายโอกาสการเข้าใช้บัสของข้อมูลที่มีค่าระดับความสำคัญสูงสุด (Non-destructive Arbitration) และการตัดสินจะเกิดขึ้นทันทีในระดับบิตข้อมูล (Bit-wise Arbitration)

กระบวนการเช่นนี้จะเกิดขึ้นได้ต้องอาศัยคุณสมบัติที่สำคัญ 2 ประการ คือ

1. สถานะของสัญญาณข้อมูลบนบัสแบบโดมิแนนต์ (Dominant) และแบบรีเซสสีฟ (Recessive) โดย CAN จะกำหนดให้สถานะโดมิแนนต์แทนลอจิก '0' และให้สถานะรีเซสสีฟแทนลอจิก '1' ในการใช้งานเมื่อสัญญาณทั้งสองสถานะถูกส่งมาที่บัสพร้อมกัน บัสจะมีสถานะโดมิแนนต์ กล่าวคือ อ่านค่าได้เป็นลอจิก '0' ซึ่งเป็นผลลัพธ์จากวิธีการของ Wired-AND ที่ทำให้สถานะรีเซสสีฟสามารถถูกเขียนทับได้ด้วยสถานะโดมิแนนต์แทน

2. โหนดผู้ส่งจะต้องสามารถตรวจสอบสถานะของบัสอยู่ตลอดเวลา ว่าข้อมูลที่อยู่บนบัสตรงกับข้อมูลที่ส่งออกไปหรือไม่ หากตรวจพบว่าไม่ตรงกันแสดงว่ามีการชนกันของข้อมูลหรือมีความผิดพลาดในการส่งข้อมูลเกิดขึ้น กล่าวคือ ในการส่งข้อมูลแต่ละโหนดจะเริ่มต้นด้วยการส่งค่ารหัสประจำตัวของข้อมูลออกมาก่อน ในขณะเดียวกันก็จะคอยตรวจสอบดูว่าค่าที่ได้รับจากบัสมีค่าตรงกับค่าที่ส่งออกไปหรือไม่ โดยการอ่านค่ากลับคืนแบบบิตต่อบิตจากสัญญาณข้อมูลบนบัสและทำการเปรียบเทียบค่าในบิตที่ส่งกับค่าในบิตที่รับ



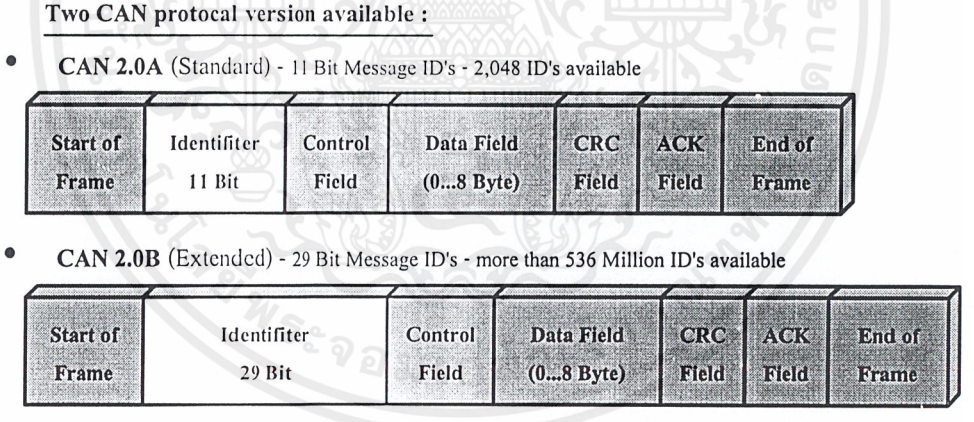
ภาพที่ 2.10 แสดงการตัดสินเข้าใช้บัสและสถานะของสัญญาณบนบัส

ในกรณีที่มีมากกว่าหนึ่งโหนดเริ่มต้นการส่งข้อมูลพร้อมกัน ดังตัวอย่างในภาพที่ 2.10 มีการส่งข้อมูลพร้อมกัน 3 โหนด ในขณะที่บิตข้อมูลของค่ารหัสประจำตัวที่แต่ละโหนดส่งออกมามีค่าตรงกันอยู่ การส่งข้อมูลในแต่ละโหนดก็จะดำเนินการต่อไปได้ตามปกติ จนกระทั่งเมื่อค่าที่ส่งออกมาไม่ตรงกัน โดยในบิตที่ 5 โหนดที่ 1 และ 3 ส่งสัญญาณสถานะโดมิแนนต์ (ลอจิก '0') ส่วนโหนดที่ 2 ส่งสัญญาณสถานะรีเซสสีฟ (ลอจิก '0') แต่อ่านค่าจากบัสกลับมาเป็นโดมิแนนต์ ทำให้โหนดที่ 2 เป็นฝ่ายแพ้ในการตัดสินเข้าใช้บัส (Losses Abritation) จึงหมดสิทธิในการส่ง

ข้อมูลในรอบนี้และเปลี่ยนมาเป็นโหนดผู้รับแทน ซึ่งทำให้ปรับเข้าสู่โหมดของการรับฟังอย่างเดียว (Listening Only Mode) เช่นเดียวกับในบิตที่ 2 โหนดที่ 1 ส่งสัญญาณสถานะรีเซสสีฟแต่อ่านค่าจากบัสกลับมาเป็นโดมิแนนต์ทำให้โหนดที่ 1 ต้องหยุดส่งข้อมูลไป สุดท้ายจึงเหลือเพียงโหนดที่ 3 เพียงโหนดเดียวที่ส่งข้อมูลในรอบนี้ จากตัวอย่างนี้จึงกล่าวได้ว่า ข้อมูลจากโหนดที่ 3 จะต้องมีค่ารหัสประจำตัวต่ำกว่าจึงมีระดับความสำคัญสูงกว่าข้อมูลจากโหนดที่ 1 และ 2 ซึ่งจะมีสิทธิส่งข้อมูลได้ก่อนและไม่ต้องเริ่มทำการส่งข้อมูลใหม่เมื่อมีการส่งข้อมูลชนกับโหนดอื่น ในขณะที่โหนดที่ 1 และ 2 ซึ่งมีระดับความสำคัญต่ำกว่าจะเริ่มทำการส่งข้อมูลได้ใหม่หลังจากโหนดที่ 3 นั้นทำการส่งข้อมูลเรียบร้อยแล้ว

2.9 โปรโตคอลของ CAN

โปรโตคอลของ CAN นั้นมีการกำหนดรูปแบบของข้อมูลที่ใช้ในการสื่อสารในระบบ ซึ่งมีการจัดทำออกมาเป็น 2 เวอร์ชัน คือ CAN 2.0A และ CAN 2.0B รูปแบบเฟรมสื่อสารทั้งสองเวอร์ชันนี้มีข้อแตกต่างกันตรงที่ขนาดของค่ารหัสประจำตัวของข้อมูล หรือเรียกว่า ค่าหมายเลข ID แสดงรายละเอียดดังภาพที่ 2.11



ภาพที่ 2.11 แสดงรูปแบบเฟรมสื่อสารในโปรโตคอลของ CAN

โดยแรกเริ่มนั้นในข้อกำหนดของ CAN จะเป็นเวอร์ชัน CAN 2.0A ซึ่งรู้จักกันดีว่าเป็น " Standard CAN " กล่าวคือ มีลักษณะเป็นรูปแบบเฟรมมาตรฐาน (Standard Frame Format) หรือเรียกว่า รูปแบบ SFF ที่มีค่าหมายเลข ID ขนาด 11 บิต ทำให้เฟรมนี้มีหมายเลข ID ที่สามารถใช้งานได้จำนวน 2^{11} = 2,048 หมายเลขที่แตกต่างกัน ได้แก่ หมายเลข ID ที่ 0 – 2,047 แต่จะมีการสำรองไว้อยู่ 16 หมายเลขสำหรับใช้งานในฟังก์ชันพิเศษเฉพาะอย่าง ได้แก่ หมายเลข ID ที่ 2,032 – 2,047 ดังนั้นจึงมีเหลือให้ใช้งานได้ 2,032 หมายเลขเป็นจำนวนข้อมูลที่สามารถรองรับได้

ต่อมาภายหลังได้มีการปรับปรุงให้ทันสมัยขึ้นเป็นเวอร์ชัน CAN 2.0B เพื่อช่วยแก้ปัญหาในเรื่องข้อจำกัดของจำนวนหมายเลข ID ที่สามารถใช้งานได้ ซึ่งเรียกได้ว่าเป็น " Extended CAN " กล่าวคือ มีลักษณะเป็นรูปแบบเฟรมเพิ่มเติม (Extended Frame Format) หรือเรียกว่า รูปแบบ EFF ที่มีหมายเลข ID ขนาด 29 บิต ประกอบด้วยหมายเลข ID ขนาด 11 บิตที่เรียกว่า " Base ID " และเพิ่มเติมด้วยหมายเลข ID ขนาด 18 บิตที่เรียกว่า " Extended ID " ส่วนสาเหตุที่ต้องทำการแบ่งหมายเลข ID ออกเป็น 2 ส่วนโดยให้ส่วนแรกมีขนาด 11 บิตเช่นนี้ เพื่อต้องการให้สอดคล้องกับหมายเลข ID ที่ใช้อยู่ในรูปแบบ SFF และสาเหตุที่ต้องมีการขยายจำนวนหมายเลข ID เพิ่มขึ้นนั้นเนื่องจากจำนวนหมายเลข ID ในรูปแบบ SFF ที่มีอยู่ 2,048 หมายเลขนั้น แม้จะดูเหมือนมากแต่ในการใช้งานบางครั้งอาจจะไม่เพียงพอ จึงได้กำหนดเป็นเวอร์ชัน CAN 2.0B ขึ้นมาเพิ่มเติม โดยมีหมายเลข ID ขนาด 29 บิต ซึ่งรองรับการใช้งานได้มากถึง 2^{29} = 536,870,912 หมายเลขด้วยกัน

นอกจากนี้โปรโตคอลของ CAN ยังมีการกำหนดลักษณะโมดูลของ CAN อีกด้วย เพื่อความเหมาะสมในการใช้งาน สามารถแบ่งออกได้เป็น 3 ชนิด ได้แก่

1. CAN 2.0A ที่จะพิจารณาหมายเลข ID ขนาด 29 บิตว่าเป็นความผิดพลาด
2. CAN 2.0B Passive ที่จะทำการละเลยต่อหมายเลข ID ขนาด 29 บิต
3. CAN 2.0B Active ที่สามารถจัดการได้ทั้งหมายเลข ID ขนาด 11 บิต และหมายเลข ID ขนาด 29 บิตได้

ตารางที่ 2.2 แสดงลักษณะโมดูลของ CAN

ชนิดโมดูลของ CAN	รูปแบบ SFF (หมายเลข ID ขนาด 11 บิต)	รูปแบบ EFF (หมายเลข ID ขนาด 29 บิต)
โมดูลแบบ CAN 2.0B Active	สามารถทำการส่งและรับได้ (Tx / Rx OK)	สามารถทำการส่งและรับได้ (Tx / Rx OK)
โมดูลแบบ CAN 2.0B Passive	สามารถทำการส่งและรับได้ (Tx / Rx OK)	อยู่ในสภาวะอดทน (Tolerated)
โมดูลแบบ CAN 2.0A	สามารถทำการส่งและรับได้ (Tx / Rx OK)	เกิดความผิดพลาดขึ้นบนบัส (Bus ERROR)

โมดูลของ CAN ที่ถูกกำหนดขึ้นมาภายหลังจากที่กำหนดเป็นเวอร์ชัน CAN 2.0A สามารถทำการส่งและรับได้เฉพาะเฟรมที่เป็นรูปแบบ SFF เท่านั้น ซึ่งถ้าหากว่ามีเฟรมที่มีหมายเลข ID ขนาด 29 บิตถูกส่งให้กับโมดูลแบบ CAN 2.0A นี้ ก็อาจจะเป็นสาเหตุให้เกิดความผิดพลาดขึ้นได้ แต่ถ้าเป็นโมดูลของ CAN ที่ถูกกำหนดขึ้นมาภายหลังจากที่กำหนดเป็นเวอร์ชัน CAN 2.0B แล้วนั้น ซึ่งจะแบ่งออกเป็น 2 ชนิด คือ โมดูลแบบ CAN 2.0B Passive ที่สามารถทำการส่งและรับได้เฉพาะ

เฟรมที่เป็นรูปแบบ SFF เท่านั้น แต่สามารถอดทนต่อเฟรมที่เป็นรูปแบบ EFF ได้โดยไม่ทำให้เกิดความผิดพลาดขึ้น และโมดูลแบบ CAN 2.0B Active ที่สามารถทำการส่งและรับได้ทั้งเฟรมที่เป็นรูปแบบ SFF และรูปแบบ EFF

2.10 ชนิดของเฟรม

โปรโตคอลการส่งข้อมูลของ CAN นั้นนอกจากจะมีการกำหนดรูปแบบของเฟรมสื่อสารออกเป็น 2 เวอร์ชันแล้ว ยังมีการจัดแบ่งข้อมูลที่ส่งให้มีลักษณะเป็นแพ็กเก็ต (Packets) หรือเฟรม (Frames) ก่อนส่งออกไปที่บัส ซึ่งมีใช้อยู่ด้วยกัน 4 ชนิด ได้แก่

1. Data Frame ที่ใช้ในการส่งข้อมูลจากโหนดผู้ส่งไปยังโหนดผู้รับตามปกติ
2. Remote Frame ที่ใช้ในการร้องขอข้อมูลจากโหนดอื่น
3. Error Frame ที่ใช้แสดงว่ามีความผิดพลาดเกิดขึ้นบนบัส
4. Overload Frame ที่ใช้เพื่อบอกว่าต้องการเวลาในการประมวลผลข้อมูลที่ได้รับเพิ่มเติม

สำหรับ Data Frame และ Remote Frame สามารถใช้งานได้ทั้งในรูปแบบ SFF และในรูปแบบ EFF โดยเฟรมทั้งสองชนิดนี้จะถูกแยกออกจากเฟรมที่อยู่ก่อนหน้านี้ด้วย Interframe Space

2.10.1 Data Frame

เฟรมนี้ใช้สำหรับการส่งข้อมูลจากโหนดผู้ส่งไปยังโหนดผู้รับตามปกติ ซึ่งถูกส่งจากโหนดที่ต้องการจะส่งข้อมูลหรือถูกร้องขอข้อมูลจากโหนดอื่น โดยสามารถส่งข้อมูลได้ 8 ไบต์ภายในหนึ่งเฟรม ในเฟรม 1 เฟรมจะประกอบด้วยส่วนย่อย ๆ ที่เรียกว่าฟิลด์ (Fields) ซึ่งสามารถแบ่งออกได้เป็น 7 ฟิลด์ ดังภาพที่ 2.12

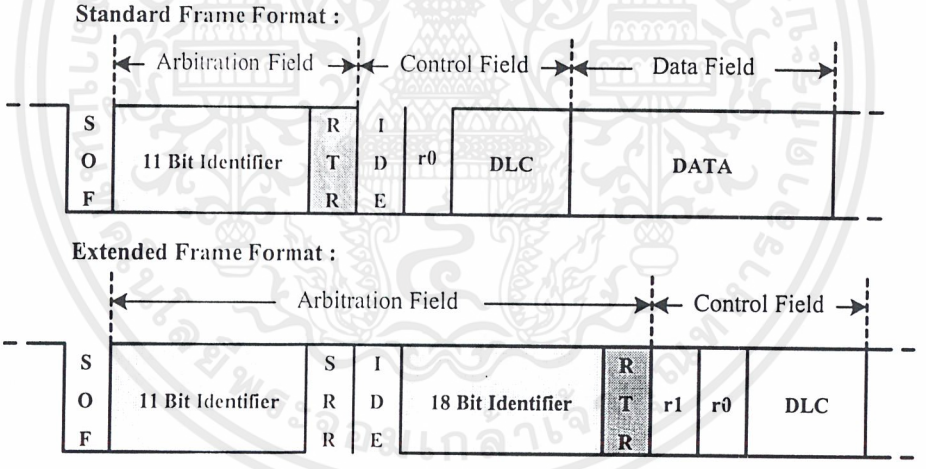
Bus Idle	Start of Frame (1 Bit)	Arbitation Field (12 or 32 Bit)	Control Field (6 Bit)	Data Field (0 to 8 Byte)	CRC Field (16 Bit)	ACK Field (2 Bit)	End of Frame (7 Bit)	Inter-Mission (3 Bit)

ภาพที่ 2.12 แสดงโครงสร้างภายในของ Data Frame

1. Start of Frame (SOF) ฟิลด์นี้จะเป็นบิตเริ่มต้นเฟรมของ Data Frame และ Remote Frame ภายในประกอบด้วยโดมิแนนต์บิตขนาด 1 บิต ซึ่งจะเป็นจุดที่ทุกโหนดบนบัสใช้ในการเริ่มต้นเข้าจังหวะสำหรับการรับและส่งข้อมูลระหว่างกัน โดยในข้อกำหนดของ CAN แล้วบิตนี้จะเป็นการ Hard Synchronization ให้กับทุกโหนดบนบัส ซึ่งกระทำทุกครั้งที่ขอบสัญญาณบนบัสเปลี่ยนสถานะจากรีเซสสีฟกลายเป็นโดมิแนนต์ในพื้นที่ระหว่างเฟรม (Interframe Space)

โดยในพื้นที่นี้สามารถที่จะเกิด SOF ได้มีทั้งหมด 3 ช่วง คือ ในช่วงที่บัสว่าง (Bus Idle) ในช่วงที่บัสพักการส่งชั่วคราว (Suspend Transmission) และในช่วงระยะหยุดพัก (Intermission) ที่ตำแหน่งบิตสุดท้าย

2. Arbitration Field ฟิลด์นี้ใช้สำหรับแสดงค่ารหัสประจำตัวของข้อมูล และใช้ในการตัดสินเข้าใช้บัส รวมถึงลำดับความสำคัญของข้อมูลอีกด้วย ภายในฟิลด์นี้มีลักษณะที่แตกต่างกันตามรูปแบบของเฟรม กล่าวคือ ในรูปแบบ SFF ของเวอร์ชัน CAN 2.0A ประกอบด้วยหมายเลข ID ขนาด 11 บิต (ID.10 - ID.0) ซึ่งจะสอดคล้องกับ Base ID ในรูปแบบ EFF และตามด้วยบิต RTR (Remote Transmission Request) ส่วนในรูปแบบ EFF ของเวอร์ชัน CAN 2.0B ประกอบด้วยหมายเลข ID ขนาด 29 บิต แบ่งออกเป็น 2 ส่วน คือ ส่วนแรกมีขนาด 11 บิต เรียกว่า Base ID (ID.28 - ID.18) และส่วนที่สองมีขนาด 18 บิต เรียกว่า Extended ID (ID.17 - ID.0) และตามด้วยบิต RTR แต่จะมีการเพิ่มเติมบิต SRR (Substitution Remote Request) และบิต IDE (Identifier Extension) เข้าไปด้วยสำหรับกั้นระหว่างหมายเลข ID ทั้งสองส่วน ดังภาพที่ 2.13



ภาพที่ 2.13 แสดงโครงสร้างภายในของ Arbitration Field

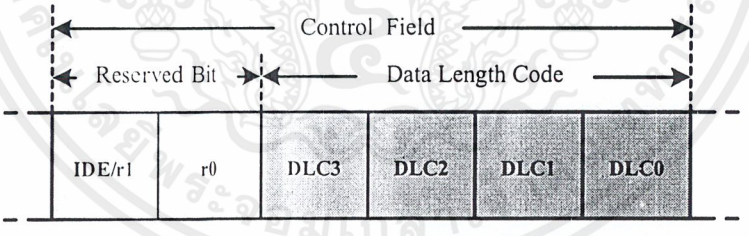
บิต RTR ซึ่งอยู่ถัดจากหมายเลข ID จะเป็นบิตที่ใช้สำหรับแสดงชนิดของเฟรม โดยใน Data Frame จะเป็นโดมิแนนต์บิต แต่ใน Remote Frame จะเป็นรีเซสสีฟบิต ดังนั้นในกรณีที่เฟรมทั้งสองที่มีหมายเลข ID เดียวกันถูกส่งออกมาพร้อมกันจะทำให้ Data Frame มีสิทธิได้เข้าใช้บัส เนื่องจากบิต RTR ที่เป็นโดมิแนนต์บิต ในขณะที่โหนดที่ส่ง Remote Frame มาก็จะได้รับข้อมูลจาก Data Frame นั้นทันทีเนื่องจากมีหมายเลข ID เดียวกัน

บิต SRR ที่เพิ่มเติมในรูปแบบ EFF จะอยู่ในตำแหน่งเดียวกันกับบิต RTR ที่อยู่ในรูปแบบ SFF เพื่อใช้แทนบิต RTR โดยเป็นรีเซสสีฟบิต ดังนั้นในการตัดสินเข้าใช้บัสถ้าเกิดว่ามี

Base ID ตรงกันแล้วจะทำให้รูปแบบ SFF ได้เปรียบมากกว่ารูปแบบ EFF ในกรณีที่บิต SRR นี้เกิดความผิดพลาดกลายเป็นโดมิแนนต์บิตแทนจะทำให้โหนดผู้รับจะมองข้ามบิตนี้ไป แต่ยังมีผลสำหรับวิธีการ Bit Stuffing ทำให้ไม่สามารถตัดสินได้ทันทีว่าเป็นบิต SRR หรือบิต RTR ด้วยเหตุนี้จึงต้องตัดสินกันที่บิต IDE แทนว่าเฟรมที่รับเข้ามาเป็นรูปแบบ SFF หรือรูปแบบ EFF

บิต IDE เป็นบิตที่ใช้สำหรับแสดงรูปแบบของเฟรม โดยในรูปแบบ SFF จะเป็นโดมิแนนต์บิตซึ่งอยู่ในตำแหน่งบิตแรกของ Control Field แต่ในรูปแบบ EFF จะเป็นรีเซสสีฟบิตซึ่งอยู่ถัดจากบิต SRR ใน Arbitration Field นี้

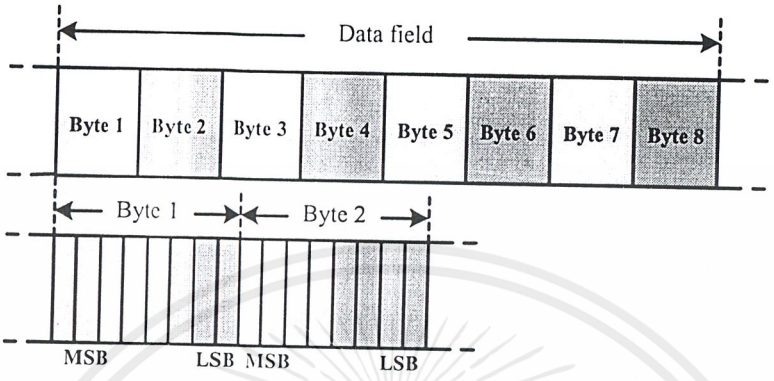
3. Control Field ฟิลด์นี้ใช้สำหรับควบคุมจำนวนข้อมูลในเฟรม มีขนาด 6 บิต ในรูปแบบ SFF และรูปแบบ EFF จะแตกต่างกันตรงตำแหน่งบิตแรกโดยในรูปแบบ SFF จะเป็นบิต IDE ที่เป็นโดมิแนนต์บิต แต่ในรูปแบบ EFF จะเป็นบิต r1 (Reserved Bit 1) แทน เนื่องจากบิต IDE ในรูปแบบ EFF ได้ถูกย้ายไปอยู่ในส่วนของ Arbitration Field แล้ว ดังนั้นที่ตำแหน่งนี้ใน Control Field ของรูปแบบ EFF จึงได้กำหนดให้เป็นบิต r1 ซึ่งสำรองไว้ใช้งานในอนาคต โดยจะกำหนดให้เป็นโดมิแนนต์บิต จากนั้นตามด้วยบิต r0 (Reserved Bit 0) ที่เป็นโดมิแนนต์บิต และส่วนของรหัส DLC (Data Length Code) จำนวน 4 บิต ได้แก่ DLC3 - DLC0 ใช้บอกขนาดของข้อมูลที่ส่งมาซึ่งเป็นจำนวนไบต์ข้อมูลที่อยู่ใน Data Field โดยข้อกำหนดของ CAN จะกำหนดให้ข้อมูลที่ส่งมาในแต่ละเฟรมมีขนาดได้ตั้งแต่ 0 – 8 ไบต์ เป็นดังภาพที่ 2.14



No. of Data Bytes	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

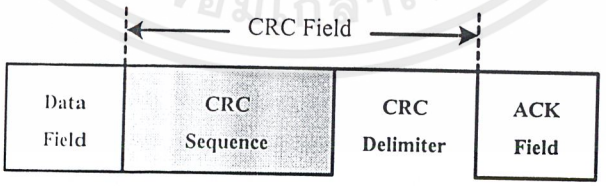
ภาพที่ 2.14 แสดงโครงสร้างภายในของ Control Field

4. Data Field ฟิลด์นี้ใช้สำหรับบรรจุไบต์ข้อมูลที่ต้องการส่ง มีขนาด 0-8 ไบต์ โดยไบต์ข้อมูลจะเริ่มทำการส่งทางด้าน MSB ของไบต์ต่ำ (Byte 1) ก่อน ซึ่งจำนวนไบต์ข้อมูลที่ส่งจะต้องตรงกับจำนวนที่กำหนดไว้ใน Control Field ดังภาพที่ 2.15



ภาพที่ 2.15 แสดงโครงสร้างภายในของ Data Field

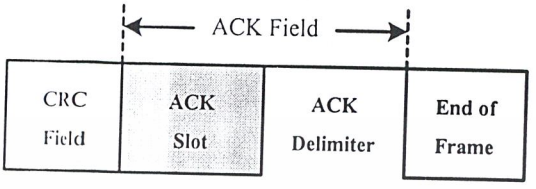
5. CRC Field ฟิลด์นี้ใช้สำหรับกระบวนการตรวจสอบความถูกต้องของข้อมูลที่ส่งในเฟรมนั้น มีขนาด 16 บิต ประกอบด้วยส่วนของ CRC Sequence ขนาด 15 บิต และตามด้วยบิต CRC Delimiter ที่เป็นรีเซสสีฟบิตเสมอ ดังภาพที่ 2.16 ในส่วนของ CRC Sequence จะได้จากการคำนวณค่า CRC (Cyclic Redundancy Check) ของการส่งข้อมูลในแต่ละเฟรมเริ่มจาก SOF ไปจนถึงบิตสุดท้ายของ Data Field โดยสามารถตรวจพบความผิดพลาดของข้อมูลที่เกิดขึ้นในช่วงดังกล่าวได้หากมีความผิดพลาดไม่เกิน 5 บิต เมื่อส่งค่า CRC ครบทั้ง 15 บิตแล้วจะปิดท้ายฟิลด์ด้วยบิต CRC Delimiter



ภาพที่ 2.16 แสดงโครงสร้างภายในของ CRC Field

6. Acknowledge Field ฟิลด์นี้ใช้สำหรับการตอบรับเฟรมที่ส่ง มีขนาด 2 บิต ประกอบด้วยบิต ACK Slot และบิต ACK Delimiter โดยโหนดผู้รับจะใช้ในการตอบกลับไปยังโหนดผู้ส่งว่าข้อมูลที่ส่งมานั้นได้รับถูกต้องเรียบร้อยแล้ว ซึ่งโหนดผู้ส่งจะทำการส่งบิต ACK Slot ที่เป็นรีเซสสีฟบิตออกไป ในขณะที่โหนดผู้รับตรวจสอบแล้วว่าข้อมูลที่ได้รับถูกต้องก็จะทำการ

ส่งบิต ACK Slot ที่เป็นโดมิแนนต์บิตตอบกลับมา ซึ่งจะทำให้สถานะบนบัสเป็นโดมิแนนต์บิต ดังนั้นเมื่อโหนดผู้ส่งอ่านค่าบิต ACK Slot กลับมาได้เป็นโดมิแนนต์บิตก็แสดงว่าจะต้องมีอย่างน้อยหนึ่งโหนดที่ได้รับข้อมูลอย่างถูกต้อง และปิดท้ายฟิลด์ด้วยบิต ACK Delimiter ที่เป็นรีเซสสีฟบิตเสมอ ดังภาพที่ 2.17



ภาพที่ 2.17 แสดงโครงสร้างภายในของ Acknowledge Field

7. End of Frame ฟิลด์นี้ใช้สำหรับแสดงว่าได้สิ้นสุดเฟรมนี้แล้วทั้ง Data Frame และ Remote Frame ประกอบด้วยรีเซสสีฟบิตจำนวน 7 บิตติดต่อกัน หรือมีค่าเท่ากับ " 1111111 "

2.10.2 Remote Frame

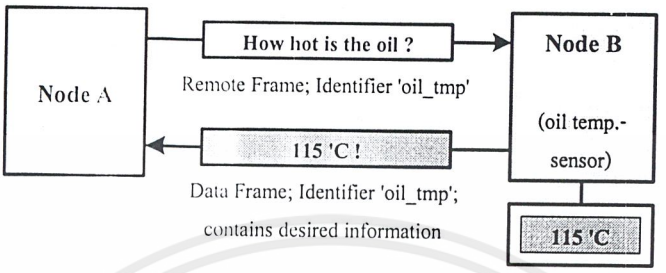
โหนดผู้รับสามารถร้องขอข้อมูลจากโหนดที่มีข้อมูลนั้นอยู่ เพื่อให้โหนดนั้นจัดส่งข้อมูลที่ต้องการนั้นมาให้ โดยการส่ง Remote Frame ที่มีหมายเลข ID ของข้อมูลที่ต้องการนั้นออกไป หากโหนดใดมีข้อมูลที่มีหมายเลข ID ตรงกับที่ถูกร้องขอมาก็จะทำการส่งข้อมูลนั้นตอบกลับไป โดยลักษณะของ Remote Frame จะคล้ายกับ Data Frame แต่จะมีส่วนที่แตกต่างกันอยู่สองประการ คือ บิต RTR จะเป็นรีเซสสีฟบิต และจะไม่มีส่วนของ Data field ประกอบอยู่ภายในเฟรมด้วย นอกจากนี้การกำหนดรหัส DLC ที่อยู่ใน Control Field นั้นต้องสอดคล้องกับจำนวนไบต์ข้อมูลที่ต้องการด้วย ดังภาพที่ 2.18

Bus Idle	Start of Frame (1 Bit)	Arbitation Field (12 or 32 Bit)	Control Field (6 Bit)	CRC Field (16 Bit)	ACK Field (2 Bit)	End of Frame (7 Bit)	Inter-Mission (3 Bit)

ภาพที่ 2.18 แสดงโครงสร้างภายในของ Remote Frame

การใช้งานของ Remote Frame นี้สามารถยกตัวอย่างได้ ดังภาพที่ 2.19 เป็นการสอบถามข้อมูลเกี่ยวกับอุณหภูมิของน้ำมัน ซึ่งวัดค่าได้จากเซนเซอร์ที่โหนด B ถ้าหากว่าโหนด A ต้องการข้อมูลนี้ก็จะทำการส่งด้วย Remote Frame ที่มีหมายเลข ID เกี่ยวกับ ' oil_temp ' จากนั้น

โหนดที่มีข้อมูลดังกล่าวจะบริการข้อมูลให้กับโหนดที่ร้องขอมา โดยที่โหนด B มีข้อมูลที่ต้องการ จึงส่งข้อมูลนั้นตอบกลับด้วย Data Frame ที่มีค่าหมายเลข ID เดียวกัน คือ ' oil_temp ' ซึ่งภายในเฟรมจะมีข้อมูลเกี่ยวกับอุณหภูมิของน้ำมันจากเซนเซอร์ที่โหนด B วัดได้ขณะนั้น



ภาพที่ 2.19 แสดงลักษณะการใช้งานของ Remote Frame

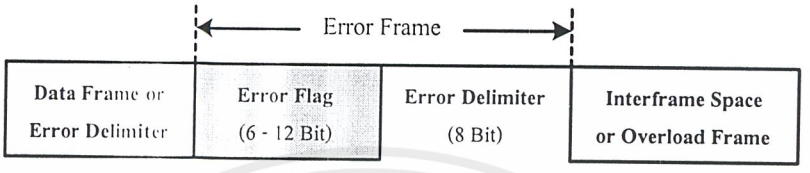
2.10.3 Error Frame

เฟรมนี้ใช้สำหรับการส่งเพื่อแสดงว่ามีความผิดพลาดเกิดขึ้นบนบัส ซึ่งถูกส่งจากโหนดใด ๆ ที่ตรวจพบความผิดพลาดบนบัส ภายใน 1 เฟรมจะประกอบด้วย 2 ฟิลด์ คือ ฟิลด์ของ Error Flag และตามด้วยฟิลด์ของ Error Delimiter ที่เป็นรีเซสสีฟบิตขนาด 8 บิตติดต่อกัน ซึ่งจะแสดงถึงการยินยอมให้โหนดต่าง ๆ เริ่มต้นการสื่อสารบนบัสได้อีกครั้งหนึ่งหลังจากความผิดพลาดที่หมดลงไปแล้ว ดังภาพที่ 2.20 นอกจากนี้ในฟิลด์ของ Error Flag นั้นจะแบ่งออกเป็น 2 รูปแบบ โดยขึ้นอยู่กับสถานะความผิดพลาด (Error Status) ได้แก่

1. Error Active โหนดที่อยู่ในสถานะนี้เมื่อตรวจพบความผิดพลาดบนบัส จะทำการขัดจังหวะการส่งข้อความในขณะนั้นด้วยการจัดส่งเป็น " Active Error Flag " ซึ่งจะประกอบด้วยโดมิแนนต์บิตขนาด 6 บิตติดต่อกัน โดยบิตพวกนี้จะทำให้วิธีการ Bit Stuffing เกิดความผิดพลาดขึ้นได้ ซึ่งโหนดอื่น ๆ ทั้งหมดสามารถรับรู้ผลของ Bit Stuffing ที่ผิดพลาดนี้และตามด้วยการส่ง Error Frame ของโหนดอื่น ๆ ดังนั้นภายใน Error Flag นี้จะประกอบด้วยโดมิแนนต์บิตจำนวน 6 – 12 บิตติดต่อกัน (เกิดจากโหนดเดียวหรือมากกว่านั้น) และปิดท้ายเฟรมนี้ด้วยฟิลด์ของ Error Delimiter ซึ่งเป็นผลให้บัสกลับสู่สภาวะปกติและโหนดที่ถูกขัดจังหวะดังกล่าวก็ได้พยายามจัดส่งข้อความที่เคยผิดพลาดนั้นอีกรอบ

2. Error Passive โหนดที่อยู่ในสถานะนี้เมื่อตรวจพบความผิดพลาดบนบัส ก็จะตรวจสอบสถานะบนบัสเพื่อรอคอยสัญญาณที่มีสถานะเดียวกันจำนวน 6 บิตติดต่อกัน จึงสามารถจัดส่งเป็น " Passive Error Flag " ได้ ซึ่งจะประกอบด้วยรีเซสสีฟบิตจำนวน 6 บิตติดต่อกัน ดังนั้น Error Frame สำหรับโหนดที่อยู่ในสถานะนี้จึงประกอบด้วยรีเซสสีฟบิตจำนวน 14 บิตติดต่อกัน กล่าวคือ ไม่มีโดมิแนนต์บิตอยู่เลย ด้วยเหตุนี้ถ้าไม่มีการตรวจพบความผิดพลาดบนบัสโดยโหนดที่

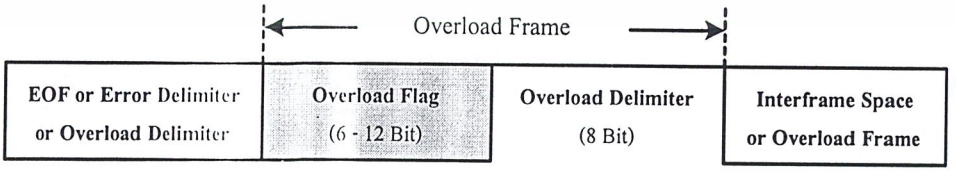
กำลังส่งจริงซึ่งเป็นตัวหลักบนบัส (Bus Master) แล้ว การส่ง Error Frame นี้จะไม่ส่งผลกระทบต่อโหนดอื่น ๆ เลย แต่ถ้าโหนดที่เป็นตัวหลักบนบัสทำการจัดส่งเป็น " Passive Error Flag " แล้วอาจเป็นเหตุให้โหนดอื่น ๆ ทำการส่ง Error Frame ออกมาได้ เนื่องจากวิธีการ Bit Stuffing ที่เกิดความผิดพลาดขึ้น



ภาพที่ 2.20 แสดงโครงสร้างภายในของ Error Frame

2.10.4 Overload Frame

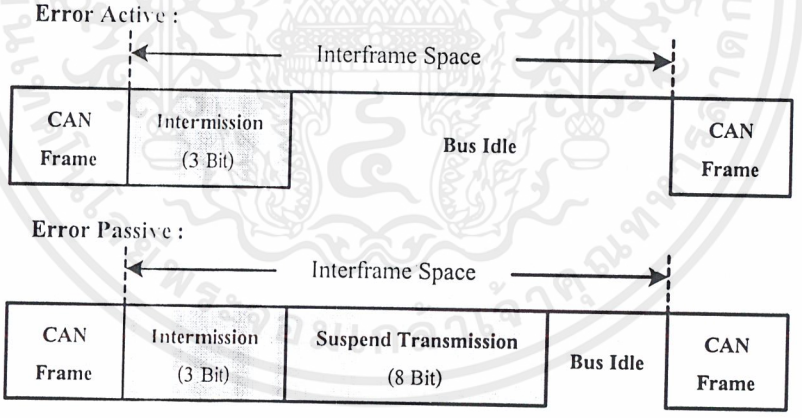
เฟรมนี้ใช้สำหรับการหน่วงเวลาบนบัส เพื่อต้องการเวลาในการประมวลผลข้อมูลที่ได้รับนั้นเพิ่มเติม โดยจะมีลักษณะคล้ายกับ Error Frame ที่อยู่ในสถานะ "Error Active" อย่างไรก็ตาม Overload Frame จะสามารถเกิดขึ้นได้ในช่วงเวลา Interframe Space เท่านั้น ซึ่งแตกต่างกับ Error Frame ที่เกิดขึ้นในช่วงเวลาที่มีการส่งข้อมูลอยู่ ภายใน 1 เฟรมจะประกอบด้วย 2 ฟิลด์ คือ ฟิลด์ของ Overload Flag ที่เป็นโดมิแนนต์บิตขนาด 6 – 12 บิตติดต่อกัน (เกิดจากโหนดเดียวหรือมากกว่านั้น) ซึ่งมีลักษณะเช่นเดียวกับ " Active Error Flag " และปิดท้ายเฟรมด้วยฟิลด์ของ Overload Delimiter ที่เป็นรีเซสสีฟบิตขนาด 8 บิตติดต่อกัน ดังภาพที่ 2.21 โดยเฟรมนี้สามารถเกิดขึ้นได้จากโหนดใด ๆ ด้วยเงื่อนไขภายในที่ทำให้โหนดนั้นไม่สามารถเริ่มรับข้อความถัดไปได้ และบางครั้งในหนึ่งโหนดอาจส่ง Overload Frame ได้สูงสุดจำนวน 2 เฟรมติดต่อกัน เพื่อทำการหน่วงเวลาในการเริ่มต้นเฟรมถัดไปที่เป็น Data Frame หรือ Remote Frame



ภาพที่ 2.21 แสดงโครงสร้างภายในของ Overload Frame

2.10.5 Interframe Space

ใช้สำหรับแยกเฟรมใด ๆ ที่อยู่ก่อนหน้านี้จาก Data Frame หรือ Remote Frame ที่ตามมา ในที่นี้ Overload Frame และ Error Frame จะไม่มี Interframe Space อยู่ก่อนหน้าเฟรมทั้งสอง และช่วงระหว่างเฟรมของ Overload Frame ด้วยกันก็จะไม่ถูกแยกด้วย Interframe Space เช่นกัน โดยเฟรมนี้จะประกอบด้วยรีเซสสีฟบิตอย่างน้อยที่สุดจำนวน 3 บิต โดยจัดอยู่ในเทอมของ Intermission ซึ่งเป็นช่วงระยะหยุดพักการสื่อสารต่าง ๆ ที่ถูกจัดเตรียมไว้ให้โหนดต่าง ๆ ได้มีเวลาสำหรับกระบวนการภายในก่อนที่จะเริ่มต้นเฟรมถัดไป โดยหลังจากเทอมของ Intermission แล้ว สำหรับโหนดต่าง ๆ ที่อยู่ในสถานะ " Error Active " นั้นสถานะของสัญญาณข้อมูลจะเป็นแบบรีเซสสีฟ กล่าวคือ เป็นสภาวะของบัสว่าง (Bus Idle) จนกว่าการส่งในครั้งต่อไปจะเริ่มขึ้น และสำหรับโหนดต่าง ๆ ที่อยู่ในสถานะ " Error Passive " ซึ่งเป็นโหนดผู้ส่งที่ส่งข้อมูลก่อนหน้านี้ หลังจากเทอมของ Intermission แล้วยังจะต้องรอคอยรีเซสสีฟบิตจำนวน 8 บิตก่อน โดยเรียกว่า Suspend Transmission ซึ่งเป็นช่วงที่บัสพักการส่งชั่วคราว ก่อนจะกลับเข้าสู่สภาวะของบัสว่างจึงจะยอมให้มีการส่งในครั้งต่อไปได้ ทำให้โหนดต่าง ๆ ที่อยู่ในสถานะ " Error Active " มีโอกาสส่งข้อมูลของตัวเองก่อนโหนดอื่นที่อยู่ในสถานะ " Error Passive " เมื่อบนบัสยอมให้มีการส่งข้อมูลได้ใหม่อีกครั้ง ดังภาพที่ 2.22

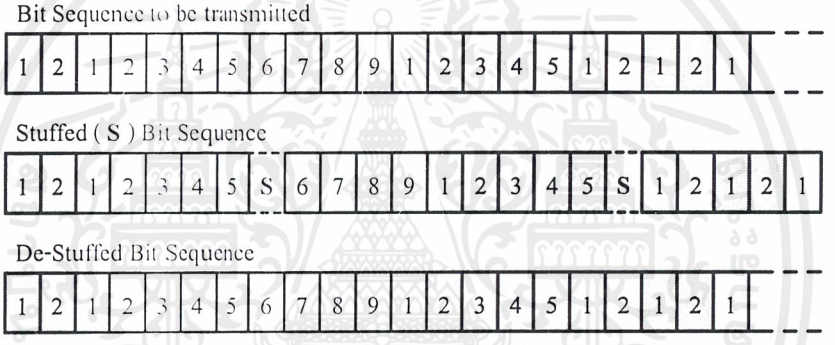


ภาพที่ 2.22 แสดงโครงสร้างภายในของ Interframe Space

2.11 การเติมสตัฟฟ์บิต

โปรโตคอลการส่งข้อมูลของ CAN มีการกำหนดให้ใช้วิธีการ Bit Stuffing เนื่องจากการส่งสัญญาณข้อมูลของ CAN จะใช้วิธีการส่งสัญญาณแบบ NRZ (Non-Return-to-Zero) ที่ระดับของสัญญาณในแต่ละบิตมีค่าคงที่ตลอดช่วงเวลาของบิตนั้นและเป็นการส่งข้อมูลแบบอะซิงโครนัส

(Asyncronous) ที่ต้องอาศัยการเข้าจังหวะสัญญาณนาฬิกาจากสัญญาณข้อมูลที่ส่งมา การส่งข้อมูลด้วยวิธีนี้จะเกิดปัญหาขึ้นในกรณีที่สัญญาณข้อมูลมีค่าเดียวกันติด ๆ กันหลายบิต ซึ่งจะทำให้การเข้าจังหวะกับสัญญาณข้อมูลนั้นมีความผิดพลาดเกิดขึ้นได้มาก ดังนั้นเพื่อป้องกันไม่ให้เหตุการณ์เช่นนี้เกิดขึ้นใน CAN จึงได้กำหนดให้ใช้วิธีการ Bit Stuffing ที่ทำการเติมสตัฟฟ์บิต (Stuff Bit) ถ้ามีการส่งบิตข้อมูลที่มีค่าเดียวกันติดกัน 5 บิตจะต้องคั่นด้วยบิตที่มีค่าตรงข้ามกับ 5 บิตนั้น 1 บิต และเมื่อโหนดผู้รับได้รับข้อมูลที่มีค่าเดียวกันติดกัน 5 บิต ก็จะไม่นับบิตที่ตามมา 1 บิตว่าเป็นบิตข้อมูลแต่จะเริ่มรับบิตข้อมูลต่อในบิตถัดไป ซึ่งการเติมสตัฟฟ์บิตนี้จะเกิดขึ้นใน Data Frame และ Remote Frame ในช่วงระหว่าง SOF กับบิต CRC Delimiter เท่านั้น นอกจากการเติมสตัฟฟ์บิตจะช่วยในการเข้าจังหวะสัญญาณข้อมูลแล้ว ยังสามารถใช้ตรวจสอบความผิดพลาดของข้อมูลได้อีกด้วย ซึ่งการเติมสตัฟฟ์บิตมีรายละเอียดดังภาพที่ 2.23



ภาพที่ 2.23 แสดงการเติมสตัฟฟ์บิตในวิธีการ Bit Stuffing

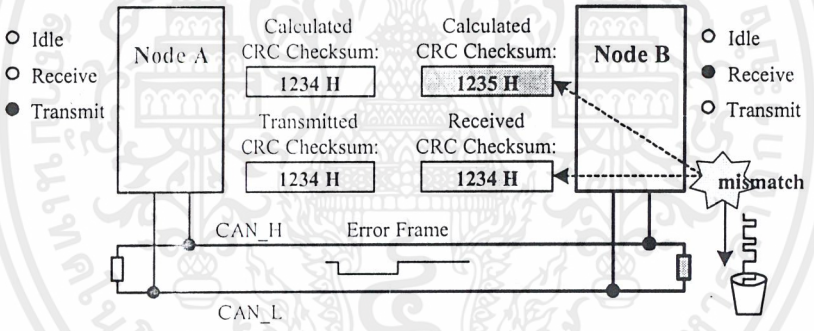
2.12 การจัดการความผิดพลาด

นับว่าเป็นจุดเด่นที่สำคัญอย่างหนึ่งของ CAN ก็คือการตรวจสอบความผิดพลาดของข้อมูลที่เกิดขึ้น เพื่อให้เห็นภาพว่า CAN มีการตรวจสอบความผิดพลาดของข้อมูลที่มีประสิทธิภาพมากเพียงใด จะขอยกตัวอย่างดังนี้ สมมติว่ามีการส่งข้อมูลด้วยอัตรา 500 กิโลบิตต่อวินาที โดยที่ทุก 0.7 วินาทีจะมีความผิดพลาดเกิดขึ้น 1 บิตและมีการใช้งาน 8 ชั่วโมงต่อหนึ่งวัน ด้วยระบบการตรวจสอบความผิดพลาดของ CAN จะรับประกันได้ว่าในระยะเวลา 1,000 ปีจะมีความผิดพลาดที่ไม่สามารถตรวจสอบได้เกิดขึ้นเพียง 1 ครั้งเท่านั้น ซึ่งจะเห็นได้ว่าโอกาสที่ข้อมูลที่ส่งจะเกิดความผิดพลาดขึ้นโดยที่ตรวจไม่พบนั้นมีน้อยมาก ด้วยเหตุนี้จึงทำให้การสื่อสารข้อมูลในระบบ CAN มีความน่าเชื่อถือและมีความปลอดภัยสูง

2.12.1 การตรวจสอบความผิดพลาด

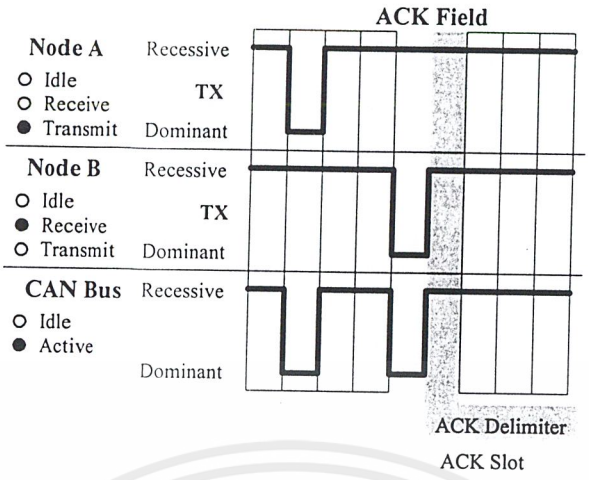
โปรโตคอลของ CAN นั้นได้จัดเตรียมวิธีการตรวจสอบความผิดพลาดของข้อมูลที่สามารถเกิดขึ้นได้ ซึ่งอาจส่งผลให้การตีความข้อมูลที่ได้รับนั้นผิดพลาดไป โดยความผิดพลาดที่ตรวจสอบได้มีอยู่ 5 ชนิด คือ CRC Error, ACK Error, From Error, Bit Error และ Stuff Error สามารถอธิบายได้ดังนี้

1. CRC Error เป็นความผิดพลาดที่เกิดการตรวจสอบความถูกต้องของข้อมูลที่ส่งในเฟรม ในขณะที่โหนดผู้ส่งทำการส่งข้อมูลจะคำนวณค่า CRC ของข้อมูลที่ส่งด้วยและส่งไปกับเฟรมใน CRC Sequence ในขณะเดียวกันโหนดผู้รับจะคำนวณค่า CRC ของข้อมูลที่รับมาด้วยวิธีเดียวกันกับโหนดผู้ส่งแล้วนำไปเปรียบเทียบกับค่า CRC ที่อยู่ใน CRC Sequence ของเฟรมที่ส่งมา ถ้าหากไม่ตรงกันแสดงว่าข้อมูลที่ได้รับมีความผิดพลาดเกิดขึ้นและจะทำการส่ง Error Frame แจ้งกลับไปเพื่อบอกว่าข้อมูลที่ได้รับไม่ถูกต้อง เมื่อโหนดที่ส่งข้อมูลได้รับการแจ้งว่ามีความผิดพลาดเกิดขึ้นก็จะทำการส่งข้อมูลนั้นออกไปใหม่อีกครั้งหนึ่ง ดังภาพที่ 2.24



ภาพที่ 2.24 แสดงวิธีการตรวจสอบความผิดพลาดชนิด CRC Error

2. ACK Error เป็นความผิดพลาดที่เกิดจากสัญญาณตอบรับเฟรมที่ส่ง โดยที่โหนดผู้ส่งจะทำการตรวจสอบดูว่าบิต ACK Slot ที่อยู่ใน Acknowledge Field เป็นโดมิแนนต์บิตหรือไม่ ในขณะที่โหนดผู้ส่งเองจะส่งค่าเป็นรีเซสสีฟบิตออกไป ถ้าหากมีค่าเป็นโดมิแนนต์บิตแสดงว่ามีโหนดผู้รับอย่างน้อยหนึ่งโหนดที่ได้รับข้อมูลแล้วอย่างถูกต้อง แต่ในทางตรงกันข้ามถ้าหากว่าบิต ACK Slot นี้มีค่าเป็นรีเซสสีฟบิตก็จะแสดงว่าไม่มีโหนดใดได้รับข้อมูลที่ถูกต้องเลย ซึ่งจะมีการส่ง Error Frame แจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง ดังภาพที่ 2.25



ภาพที่ 2.25 แสดงวิธีการตรวจสอบความผิดพลาดชนิด ACK Error

3. Frome Error เป็นความผิดพลาดที่เกิดจากรูปแบบของบิตที่กำหนดแน่นอนเกิดเปลี่ยนแปลงไป กล่าวคือ ถ้าหากพบว่ามีสถานะโดมิแนนต์เกิดขึ้นในส่วนต่าง ๆ ต่อไปนี้ ได้แก่ บิต CRC Delimiter, บิต ACK Delimiter, EOF และ Interframe Space เนื่องจากส่วนต่าง ๆ เหล่านี้ควรจะมีสถานะรีเซสสีฟแสดงว่ามีความผิดพลาดเกิดขึ้นในการส่งข้อมูลดังกล่าว ซึ่งจะมีการส่ง Error Frame แจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง ในที่นี้จะยกเว้นให้สำหรับโหนดผู้รับที่ตรวจพบโดมิแนนต์บิตอยู่ในบิตสุดท้ายของ EOF โดยจะไม่ตีความว่าเป็น Frome Error

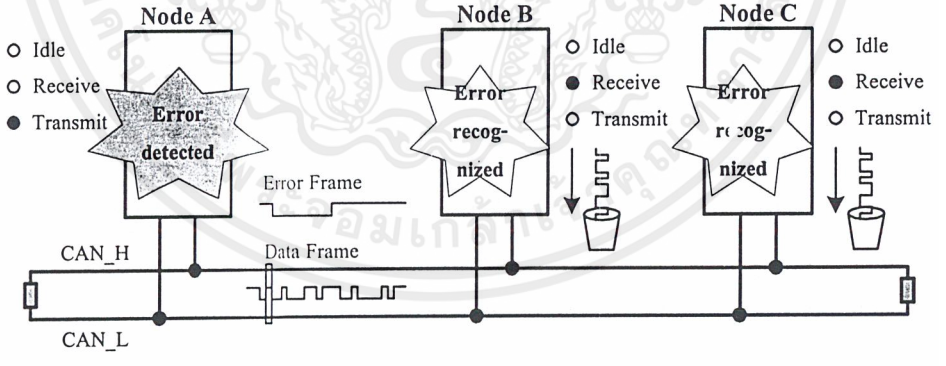
4. Bit Error เป็นความผิดพลาดที่เกิดจากสถานะของบิตที่ส่งเกิดเปลี่ยนแปลง ซึ่งจะเกิดขึ้นเมื่อโหนดผู้ส่งอ่านค่าจากบัสกลับมาได้ค่าไม่ตรงกับค่าที่ส่งออกไป คือ ส่งค่าโดมิแนนต์แต่อ่านค่าได้เป็นรีเซสสีฟแทนหรือส่งค่ารีเซสสีฟแต่อ่านค่าได้เป็นโดมิแนนต์แทน แสดงว่ามีความผิดพลาดเกิดขึ้นในการส่งข้อมูลดังกล่าว ซึ่งจะมีการส่ง Error Frame แจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง โดยความผิดพลาดชนิดนี้จะถูกยกเว้นในฟิลด์ของ Arbitration Field และบิต ACK Slot ในกรณีที่ส่งค่ารีเซสสีฟแต่อ่านค่าได้เป็นโดมิแนนต์แทน ก็จะไม่ถือว่าเป็นความผิดพลาดเนื่องจากสามารถเกิดขึ้นได้ เพราะส่วนนี้สามารถเขียนทับได้ (overwrite) ด้วยโดมิแนนต์เพื่อใช้ในการตัดสินระดับบิตและการตอบรับเฟรม นอกจากนี้แล้วสำหรับโหนดผู้ส่งที่ทำการส่งเป็น Passive Error Flag และอ่านค่าได้เป็นโดมิแนนต์ก็จะไม่ตีความว่าเป็น Bit Error

5. Stuff Error เป็นความผิดพลาดที่เกิดจากวิธีการ Bit Stuffing เกิดผิดพลาด จากที่กล่าวไปแล้วว่าการส่งข้อมูลของ CAN จะมีการสตัฟฟ์บิตเมื่อข้อมูลที่ส่งมีค่าเดียวกันติดกัน 5 บิต ดังนั้นเมื่อข้อมูลที่ได้รับในช่วงระหว่าง SOF กับบิต CRC Delimiter มีค่าเดียวกันติดกันเกิน 5 บิต แสดงว่ามีความผิดพลาดเกิดขึ้นในการส่งข้อมูลดังกล่าว ซึ่งจะมีการส่ง Error Frame แจ้งกลับไปเพื่อให้มีการส่งข้อมูลดังกล่าวมาอีกครั้ง

2.12.2 การส่งสัญญาณแสดงความผิดพลาด

จากวิธีการตรวจสอบความผิดพลาดของข้อมูล เมื่อมีโหนดใดที่ตรวจพบเงื่อนไขการเกิดความผิดพลาด ก็จะส่งสัญญาณแสดงความผิดพลาดเพื่อแจ้งให้กับระบบได้รับทราบด้วยการส่ง Error Flag สำหรับโหนดที่มีสถานะเป็น " Error Active " จะทำการส่งเป็น Active Error Flag และสำหรับโหนดที่มีสถานะเป็น " Error Passive " จะทำการส่งเป็น Passive Error Flag โดยเมื่อใดก็ตามที่โหนดใด ๆ ตรวจพบความผิดพลาดของข้อมูลในชนิด Bit Error, Stuff Error, Form Error หรือ ACK Error การส่ง Error Flag ของแต่ละโหนดจะเริ่มต้นในบิตข้อมูลถัดไปทันที แต่เมื่อใดก็ตามที่โหนดใด ๆ ตรวจพบความผิดพลาดของข้อมูลในชนิด CRC Error การส่ง Error Flag ของแต่ละโหนดจะเริ่มต้นในบิตข้อมูลที่อยู่ถัดไปจาก CRC Delimiter

ความผิดพลาดของข้อมูลที่เกิดขึ้น เมื่อถูกตรวจพบจะทำให้โหนดอื่น ๆ ทั้งหมดสามารถรับทราบได้ผ่านทาง Error Frame โดยข้อมูลที่ส่งออกไปแล้วเกิดผิดพลาดนี้จะถูกยกเลิกไปดังภาพที่ 2.26 และหลังจากที่การส่ง Error Frame ผ่านไปแล้วก็แสดงว่าไม่มีความผิดพลาดบนบัสอีกจึงสามารถเริ่มการสื่อสารได้ใหม่อีกครั้ง โดยโหนดผู้ส่งจะพยายามจัดส่งข้อมูลที่เคยผิดพลาดนั้นอีกรอบ ถ้าหากไม่มีข้อความใด ๆ ที่มีลำดับความสำคัญสูงกว่าต้องการเข้าใช้บัสพร้อมกันแล้วทำให้การส่งข้อมูลที่เคยผิดพลาดจะใช้เวลาอย่างน้อยที่สุด 23 บิตในการจัดส่งข้อมูลดังกล่าวอีกครั้ง และในกรณีของโหนดที่มีสถานะเป็น " Error Passive " จะใช้เวลาสูงสุดถึง 31 บิตเนื่องจากมีการเพิ่มช่วง Suspend Transmission ขนาด 8 บิตอีกด้วย



ภาพที่ 2.26 แสดงการจัดการกับความผิดพลาดที่เกิดขึ้นบนบัส

2.13 สถานะความผิดพลาด

นอกจากความสามารถในการตรวจสอบความผิดพลาดที่เกิดขึ้นแล้วนั้น CAN ยังได้มีการกำหนดสถานะความผิดพลาดของแต่ละโหนดบนบัสขึ้น เพื่อจัดการควบคุมให้การใช้งานบนบัสนั้นมีประสิทธิภาพมากขึ้น กล่าวคือ ในกรณีที่โหนดใดมีความผิดพลาดเกิดขึ้นมากเกินกว่าเกณฑ์ที่กำหนดแล้วก็จะไม่อนุญาตให้เข้าใช้บัสได้ เนื่องจากหากปล่อยให้ดังกล่าวใช้บัสได้ตามปกติจะทำ

ให้การใช้งานบัสโดยรวมมีประสิทธิภาพต่ำลงได้ เพราะจะทำให้เกิดความผิดพลาดขึ้นบ่อยครั้งได้ ซึ่งทุกครั้งจะต้องมีการส่งข้อมูลกันใหม่ ทำให้ปริมาณการใช้งานบนบัสมีมากและไปแย่งเวลากับโหนดอื่นโดยไม่จำเป็นอีกด้วย

ในแต่ละโหนดของ CAN นั้นอาจจะอยู่หนึ่งในสถานะความผิดพลาดทั้งหมด 3 แบบ ได้แก่ " Error Active ", " Error Passive " หรือ " Bus Off " ซึ่งจะขึ้นอยู่กับค่าของตัวนับความผิดพลาดภายใน (Internal Error Counters) ของแต่ละโหนดว่ามีค่าอยู่ในเกณฑ์ของสถานะความผิดพลาดแบบใด ดังภาพที่ 2.26 โดยการทำงานในแต่ละสถานะจะมีลักษณะ ดังนี้

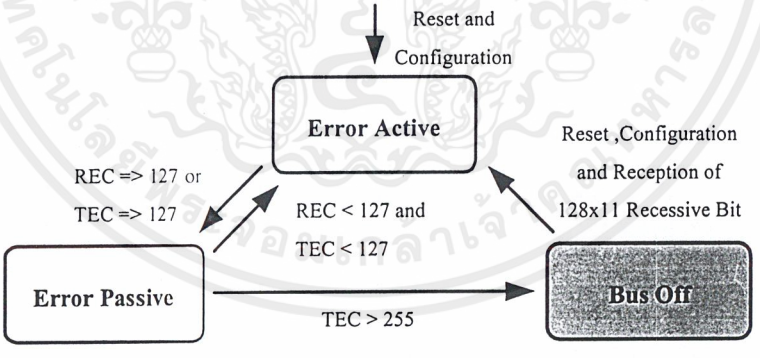
1. Error Active โหนดที่อยู่ในสถานะนี้สามารถทำการสื่อสารข้อมูลบนบัสได้ปกติ และทำการส่งเป็น Active Error Flag เมื่อตรวจพบความผิดพลาดบนบัส

2. Error Passive โหนดที่อยู่ในสถานะนี้สามารถทำการสื่อสารข้อมูลบนบัสได้เช่นกัน แต่ทำการส่งเป็น Passive Error Flag เมื่อตรวจพบความผิดพลาดบนบัส และหลังจากที่มีการส่งข้อมูลแล้วจะต้องหยุดพักการส่งชั่วคราว (Suspend Transmission) ก่อนที่จะทำการส่งข้อมูลได้ใหม่

3. Bus Off โหนดที่อยู่ในสถานะนี้จะไม่สามารถทำการสื่อสารข้อมูลบนบัสได้

ในแต่ละโหนดจะมีตัวนับความผิดพลาดภายในเป็นส่วนประกอบสำคัญ โดยสามารถแบ่งออกได้เป็น 2 ชนิด คือ

1. Transmit Error Counter (TEC) เป็นตัวนับความผิดพลาดจากการส่งข้อมูล
2. Receive Error Counter (REC) เป็นตัวนับความผิดพลาดจากการรับข้อมูล



ภาพที่ 2.27 แสดงเกณฑ์ของสถานะความผิดพลาด

ในระหว่างที่ CAN ทำการสื่อสารข้อมูล ตัวนับความผิดพลาดภายในของแต่ละโหนดจะต้องถูกปรับค่าให้ทันสมัยเสมอ ความผิดพลาดในแต่ละครั้งที่เกิดจากการรับหรือการส่งจะเป็นการเพิ่มค่าขึ้นด้วยค่าที่กำหนด และสำหรับการสื่อสารข้อมูลที่เป็นผลสำเร็จในแต่ละครั้งก็จะเป็นการลดค่าลงด้วยค่าที่กำหนดเช่นกัน โดยการเปลี่ยนแปลงสถานะของโหนดและค่าภายในตัวนับความผิดพลาดทั้งสองชนิดนั้นจะมีวิธีการดังนี้ คือ

1. เมื่อโหนดผู้รับตรวจพบความผิดพลาดจะทำให้ REC เพิ่มค่าขึ้นจากเดิมอีก 1 โดยยกเว้นสำหรับการตรวจพบ Bit Error ในระหว่างการส่งของ Active Error Flag หรือ Overload Flag

2. เมื่อโหนดผู้รับตรวจพบโดมิแนนต์บิตที่บิตแรกหลังจากการส่งของ Active Error Flag จะทำให้ REC เพิ่มค่าขึ้นจากเดิมอีก 8

3. เมื่อโหนดผู้ส่งทำการส่ง Error Flag จะทำให้ TEC เพิ่มค่าขึ้นจากเดิมอีก 8 โดยยกเว้นสำหรับโหนดผู้ส่งที่มีสถานะเป็น " Error Passive " ที่ตรวจพบ ACK Error และต้องตรวจไม่พบโดมิแนนต์บิตขณะที่ทำการส่ง Passive Error Flag ของตัวเอง และยกเว้นสำหรับโหนดผู้ส่งที่ทำการส่ง Error Flag เพราะว่าเกิด Stuff Error ในระหว่าง Arbitration Field สำหรับข้อยกเว้นทั้งสองกรณีจะไม่มีการเปลี่ยนแปลงค่าใน TEC

4. เมื่อโหนดผู้ส่งตรวจพบ Bit Error ในระหว่างการส่งของ Active Error Flag หรือ Overload Flag จะทำให้ TEC เพิ่มค่าขึ้นจากเดิมอีก 8

5. เมื่อโหนดผู้รับตรวจพบ Bit Error ในระหว่างการส่งของ Active Error Flag หรือ Overload Flag จะทำให้ REC เพิ่มค่าขึ้นจากเดิมอีก 8

6. โหนดใด ๆ จะยอมอดทนต่อโดมิแนนต์บิตได้ถึง 7 บิตติดต่อกัน ภายหลังจากการส่งของ Active Error Flag , Passive Error Flag หรือ Overload Flag กล่าวคือ หลังจากตรวจพบในบิตที่ 14 ของโดมิแนนต์บิตที่ติดต่อกันในกรณีของ Active Error Flag หรือ Overload Flag หรือหลังจากตรวจพบในบิตที่ 8 ของโดมิแนนต์บิตที่ติดต่อกันที่อยู่ถัดจาก Passive Error Flag จะทำให้ TEC ของโหนดผู้ส่งทุกโหนดเพิ่มค่าขึ้นจากเดิมอีก 8 และจะทำให้ REC ของโหนดผู้รับทุกโหนดเพิ่มค่าขึ้นจากเดิมอีก 8

7. หลังจากการส่งข้อมูลเป็นผลสำเร็จ กล่าวคือ ต้องได้รับ ACK และไม่มีความผิดพลาดเกิดขึ้นจนจบ EOF จะทำให้ TEC ลดค่าลงจากเดิมอีก 1 ถ้าไม่เช่นนั้นแสดงว่ามีค่าเป็น 0 แล้ว

8. หลังจากการรับข้อมูลเป็นผลสำเร็จ กล่าวคือ ต้องไม่มีความผิดพลาดเกิดขึ้นจนถึง ACK และทำการส่งบิต ACK Slot เรียบร้อย จะทำให้ REC ลดค่าลงจากเดิมอีก 1 ถ้าไม่เช่นนั้นแสดงว่ามีค่าเป็น 0 แล้ว และถ้าหากว่ามีค่ามากกว่า 127 จะถูกปรับค่าให้อยู่ในช่วงระหว่าง 119-127

9. โหนดจะมีสถานะเป็น " Error Passive " เมื่อค่าใน TEC และ REC มีค่ามากกว่าหรือเท่ากับ 128

10. โหนดจะมีสถานะเป็น " Bus Off " เมื่อค่าใน TEC มีค่ามากกว่าหรือเท่ากับ 256

11. โหนดจะมีสถานะเป็น " Error Passive " จะกลับมาเป็น " Error Active " ได้อีกครั้งเมื่อค่าใน TEC และ REC มีค่าน้อยกว่าหรือเท่ากับ 127

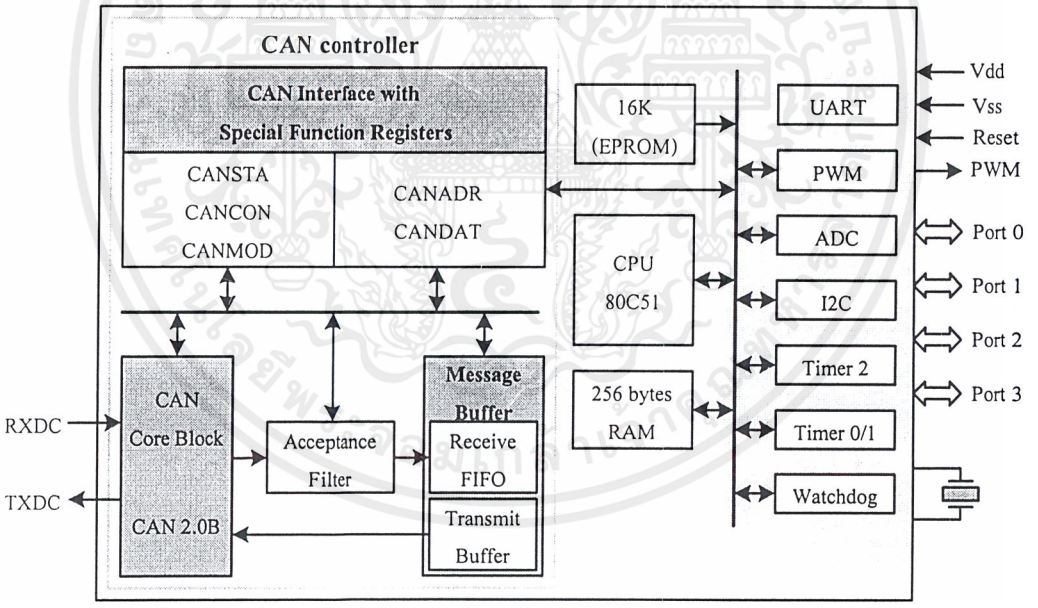
12. โหนดจะมีสถานะเป็น " Bus Off " จะยอมให้กลับมาเป็น " Error Active " พร้อมเซ็ทค่าใน TEC และ REC ให้มีค่าเท่ากับ 0 ภายหลังจากที่มีการตรวจพบรีเซสสีฟบิตจำนวน 11 บิตติดต่อกันเกิดขึ้นเป็นจำนวน 128 ครั้งบนบัส

บทที่ 3
รายละเอียดของไอซี CAN คอนโทรลเลอร์

3.1 บทกล่าวนำ

ไอซีเบอร์ P8xC591 เป็นไมโครคอนโทรลเลอร์ ขนาด 8 บิต ที่ประกอบด้วยส่วนของ 8051 และ CAN คอนโทรลเลอร์ เวอร์ชัน CAN 2.0B Active ซึ่งได้ถูกพัฒนาขึ้นสำหรับการใช้งานในด้านการสื่อสารข้อมูลระหว่างอุปกรณ์อิเล็กทรอนิกส์ที่ใช้ในงานอุตสาหกรรม โดยเฉพาะในระบบอิเล็กทรอนิกส์สำหรับยานยนต์ โดยระบบปฏิบัติการของ CAN คอนโทรลเลอร์ จะสนับสนุนและรองรับมาตรฐาน ISO11898 ในส่วนของ Data Link Layer

โครงสร้างภายในของไมโครคอนโทรลเลอร์ P8xC591 ที่เป็นเวอร์ชัน P83C591 (ROM) หรือเวอร์ชัน P87C591 (OTP EPROM) ประกอบด้วยส่วนการทำงานที่สำคัญต่าง ๆ โดยแสดงไว้ในบล็อกไดอะแกรมดังภาพที่ 3.1



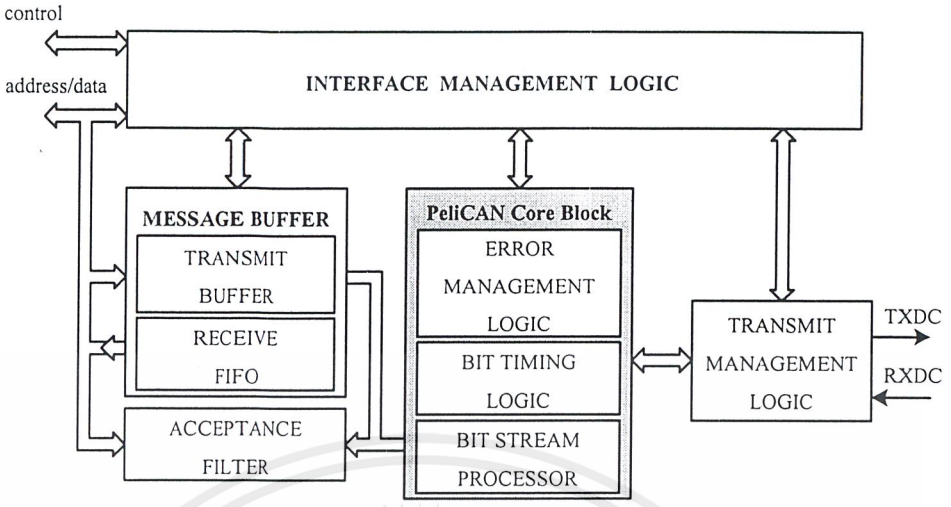
ภาพที่ 3.1 แสดงบล็อกไดอะแกรมของ P8xC591

P8xC591 ประกอบด้วยส่วนของ CAN คอนโทรลเลอร์ ที่มีสมรรถภาพในการทำงานสูง ซึ่งมีการเพิ่มเติมจากลักษณะทั่วไปของ CAN โดยเป็นที่รู้จักกันในนามของ PeliCAN ได้จัดเตรียมฟังก์ชันการทำงานที่มีการปรับปรุงเพิ่มเติม จากไอซีเบอร์ SJA1000 ที่เป็น CAN คอนโทรลเลอร์แบบ Stand-alone โดยมีลักษณะการทำงานดังนี้

1. รองรับหมายเลข ID ในรูปแบบ SFF ขนาด 11 บิต และรูปแบบ EFF ขนาด 29 บิต
2. อัตราเร็วข้อมูลสูงสุด 1 เมกะบิตต่อวินาที
3. ตัวนับความผิดพลาด (Error Counter) สามารถเข้าถึงได้ด้วยการอ่านและเขียน
4. ขอบเขตการเตือนความผิดพลาด (Error Warning Limit) สามารถโปรแกรมได้
5. จัดเก็บข้อมูลแบบ Receive FIFO ขนาด 64 ไบต์ และ Transmit Buffer ขนาด 13 ไบต์
6. รูปแบบของ Acceptance Filter Bank สามารถแยกกำหนดได้อิสระจำนวน 4 Bank
7. ในแต่ละ Bank ของ Acceptance Filter Bank สามารถกำหนดได้ 4 รูปแบบ
8. ในแต่ละฟิลเตอร์มีการกำหนด Code ขนาด 32 บิต และ Mask ขนาด 32 บิต
9. ฟิลเตอร์ทั้งหมดสามารถเปลี่ยนแปลงได้ขณะทำงาน เรียกว่า " change on the fly "
10. ในส่วนของ Acceptance Filter มีการรองรับกับ HLP (Higher Layer Protocol)
11. อินเทอร์รัพท์การรับ (Receive Interrupt) เมื่อ RX Interrupt Level ถึงระดับที่กำหนด
12. อินเทอร์รัพท์การรับ (Receive Interrupt) เมื่อรับ Data Frame ที่มีระดับความสำคัญสูง
13. โหมดการรับฟังอย่างเดียว (Listening Only Mode) ที่ทำงานโดยไม่ต้องการสัญญาณตอบรับ (Acknowledge) และจะไม่เกิด Active Error Flag
14. โหมดการทดสอบตัวเอง (Self Test Mode) ที่ทำงานโดยทำการรับข้อมูลของตัวเอง แบ่งเป็นแบบ Global Self Test ที่ต้องการ ACK และแบบ Local Self Test ที่ไม่ต้องการ ACK
15. มีการตรวจจับรหัสความผิดพลาด (Error Code Capture) โดยจะแจ้งให้ทราบเกี่ยวกับชนิดความผิดพลาด และรายละเอียดของตำแหน่งบิต
16. มีการตรวจจับการแพ้ในการตัดสินเข้าใช้บัส (Arbitration Lost Capture) โดยจะแจ้งให้ทราบเกี่ยวกับรายละเอียดของตำแหน่งบิต

3.2 โครงสร้างภายในของ PeliCAN

โครงสร้างภายในมีส่วนอินเทอร์เฟสของ CPU 8051 ที่ถูกใช้สำหรับการเชื่อมต่อ PeliCAN เข้ากับบัสภายในของไมโครคอนโทรลเลอร์ P8xC591 โดยผ่านทางรีจิสเตอร์ฟังก์ชันพิเศษ (SFR) จำนวน 5 ตัว ได้แก่ CANADR, CANDAT, CANMOD, CANSTA และ CANCON ด้วยวิธีการนี้ทำให้ CPU สามารถเข้าถึง PeliCAN ได้อย่างสะดวกและรวดเร็วยิ่งขึ้น นอกจากส่วนอินเทอร์เฟสดังกล่าวแล้วโครงสร้างภายในของ PeliCAN ยังประกอบด้วยส่วนการทำงานที่สำคัญต่าง ๆ โดยแสดงไว้ในบล็อกไดอะแกรมดังภาพที่ 3.2



ภาพที่ 3.2 แสดงบล็อกไดอะแกรมของ PeliCAN

1. Interface Management Logic (IML) ส่วนนี้มีหน้าที่ในการแปลคำสั่งที่ได้รับจาก CPU เพื่อจัดการและควบคุมการอ้างตำแหน่งแอดเดรสรีจิสเตอร์ของ CAN รวมถึงจัดเตรียมในเรื่องของการอินเทอร์รัพท์และข้อมูลข่าวสารด้านสถานะให้กับ CPU ในส่วนนี้จัดว่าเป็นส่วนอินเทอร์เฟสหลักของ PeliCAN

2. Transmit Buffer (TXB) ส่วนนี้เป็นอินเทอร์เฟสระหว่าง Bit Steam Processor (BSP) และ CPU โดยมีหน้าที่ในการจัดเก็บข้อมูลของ CAN ที่สมบูรณ์ ซึ่งพร้อมจะถูกส่งออกไปที่บัส บัฟเฟอร์นี้มีความยาวขนาด 13 ไบต์ ทำการเขียนโดย CPU และทำการอ่านค่าออกไปโดย BSP หรือจาก CPU เอง

3. Receive Buffer (RXB, RXFIFO) ส่วนนี้เป็นอินเทอร์เฟสระหว่าง Acceptance Filter (ACF) และ CPU โดยมีหน้าที่ในการจัดเก็บข้อมูลของ CAN ที่รับเข้ามาจากบัส และเป็นข้อมูลที่ผ่านการฟิลเตอร์แล้ว บัฟเฟอร์นี้เป็นส่วนแสดงผลให้กับ CPU สามารถเข้าถึงข้อมูลดังกล่าวได้จากวินโดว์ของ RXB ที่มีความยาวขนาด 13 ไบต์ ซึ่งถูกจัดวางอยู่ใน Receive FIFO (RXFIFO) ที่มีความยาวทั้งหมดขนาด 64 ไบต์ ด้วยความช่วยเหลือของ FIFO นี้ส่งผลให้ CPU สามารถทำการประมวลผลข้อมูลหนึ่งได้ในขณะที่ข้อมูลอื่น ๆ กำลังถูกรับเข้ามา

4. Acceptance Filter (ACF) ส่วนนี้มีหน้าที่ในการฟิลเตอร์ข้อมูลที่รับเข้ามา โดยการเปรียบเทียบค่าหมายเลข ID ของข้อมูลด้วยรายละเอียดที่กำหนดใน Acceptance Filter และทำการตัดสินใจว่าควรจะรับข้อมูลนี้หรือไม่ ในกรณีที่ผ่านการฟิลเตอร์เรียบร้อยแล้วจะส่งผลให้ข้อมูลดังกล่าวถูกจัดเก็บใน RXFIFO โดยที่ ACF ประกอบด้วย Acceptance Filter Bank ที่สามารถแยกกำหนดได้อิสระจำนวน 4 Bank ซึ่งรองรับเฟรมได้ทั้งรูปแบบ SFF และรูปแบบ EFF นอกจากนี้ยังมีลักษณะการทำงานแบบ " change on the fly "

5. Bit Steam Processor (BSP) ส่วนนี้มีหน้าที่ในการควบคุมกระแสข้อมูลระหว่าง TXB, RXFIFO และบัสของ CAN โดยในส่วนนี้จะมีการทำงานในเรื่องของการตรวจสอบความผิดพลาด การตัดสินเข้าใช้บัส วิธีการ Bit Stuffing และการตรวจจับความผิดพลาดบนบัสของ CAN เช่นกัน

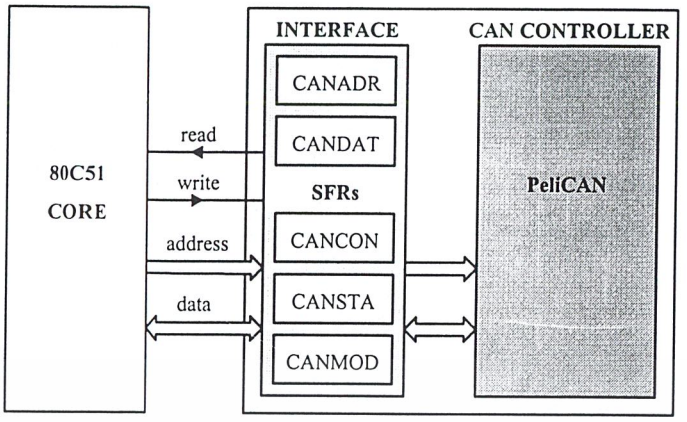
6. Error Management Logic (EML) ส่วนนี้จะตอบสนองกับการจำกัดความผิดพลาดของโมดูลในชั้นส่งผ่านข้อมูล กล่าวคือ เมื่อได้รับการแจ้งความผิดพลาดจาก BSP แล้วจะทำการแจ้งกลับให้ BSP และ IML ได้รับทราบเกี่ยวกับสถิติของความผิดพลาดในขณะนั้น

7. Bit Timing Logic (BTL) ส่วนนี้มีหน้าที่ในการตรวจสอบสัญญาณบนบัสของ CAN และจัดการควบคุมอัตราเร็วข้อมูลบนบัสให้สัมพันธ์กัน นอกจากนี้ยังได้จัดเตรียมการโปรแกรมเวลาเพื่อชดเชยสำหรับการส่งผ่านที่ถูกหน่วงเวลาและเฟสที่ถูกชิปไป รวมถึงการกำหนดเวลาของการสุ่มและจำนวนการสุ่มที่กระทำในหนึ่งเวลาบิต

8. Transmit Management Logic (TML) ส่วนนี้มีหน้าที่ในการจัดเตรียมสัญญาณของไดรเวอร์สำหรับการ push-pull สถานะทรานซิสเตอร์ของ CAN TX ซึ่งจะขึ้นอยู่กับการโปรแกรมไดรเวอร์ทางเอาท์พุท

3.3 การสื่อสารระหว่าง CPU และ PeliCAN คอนโทรลเลอร์

ส่วนอินเทอร์เฟสของ CPU เชื่อมต่อ PeliCAN เข้ากับบัสภายในของไมโครคอนโทรลเลอร์โดยผ่านทางรีจิสเตอร์ฟังก์ชันพิเศษ (SFR) ซึ่งยอมให้มีการเข้าถึงรีจิสเตอร์ใน PeliCAN และพื้นที่แอดเดรสใน RAM ได้อย่างสะดวกและรวดเร็วยิ่งขึ้น เนื่องจากการรองรับย่านแอดเดรสขนาดใหญ่จึงใช้การอ้างแอดเดรสทางอ้อมผ่านทางพอยน์เตอร์ ซึ่งมีความรวดเร็วในการเข้าถึงรีจิสเตอร์ด้วยโหมดการเพิ่มแอดเดรสแบบอัตโนมัติ ทำให้สามารถลดจำนวนของ SFR ที่ใช้เหลือเพียง 5 ตัว คือ CANADR, CANDAT, CANMOD, CANSTA และ CANCON ในส่วนของ CANSTA และ CANCON จะมีการจัดวางรีจิสเตอร์ที่แตกต่างกันโดยขึ้นอยู่กับทิศทางของการเข้าถึง รีจิสเตอร์ใน PeliCAN อาจถูกเข้าถึงได้ด้วยวิธีที่แตกต่างกัน 2 วิธี กล่าวคือ ถ้าหากเป็นรีจิสเตอร์ที่มีความสำคัญมากและควรรองรับการโพลลิ่งทางซอฟท์แวร์ หรือเป็นรีจิสเตอร์ที่ควบคุมฟังก์ชันหลักของ CAN จะสามารถเข้าถึงได้ทางตรงโดยผ่านทาง SFR และถ้าหากเป็นรีจิสเตอร์ในส่วนอื่น ๆ จะสามารถเข้าถึงได้ทางอ้อมผ่านทางพอยน์เตอร์ของ SFR ในกรณีที่ทำการรับข้อมูลในระดับสูง กล่าวคือ จำนวนมากกว่า 1 ไบต์ และใช้การอ้างแอดเดรสทางอ้อมแล้วจะมีการทำงานในลักษณะการเพิ่มแอดเดรสแบบอัตโนมัติ



ภาพที่ 3.3 แสดงการอินเทอร์เฟสของ CPU กับ PeliCAN

ตารางที่ 3.1 แสดงรีจิสเตอร์ฟังก์ชันพิเศษ (SFR) ของ CAN

SFR	Access	PeliCAN REG.	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
CANADR	Read/ Write	-	CANA7	CANA6	CANA5	CANA4	CANA3	CANA2	CANA1	CANA0
CANDAT	Read/ Write	-	CAND7	CAND6	CAND5	CAND4	CAND3	CAND2	CAND1	CAND0
CANMOD	Read/ Write	Mode	TM	RIPM	RPM	SM	-	STM	LOM	RM
CANSTA	Read Write	Status Interrupt Enable	BS BEIE	ES ALIE	TS EPIE	RS WUIE	TCS DOIE	TBS EIE	DOS TIE	RBS RIE
CANCON	Read Write	Interrupt Command	BEI -	ALI -	EPI -	WUI SRR	DOI CDO	EI RRB	TI AT	RI TR

1. CANADR เป็นรีจิสเตอร์ที่ใช้ในการกำหนดแอดเดรสของรีจิสเตอร์ภายใน PeliCAN ที่จะถูกเข้าถึงผ่านทาง CANDAT ซึ่งเปรียบได้กับพอยน์เตอร์ที่ชี้ไปยัง PeliCAN และด้วยการสนับสนุนในโหมดการเพิ่มแอดเดรสอัตโนมัติที่เป็นส่วนประกอบอยู่นี้ ทำให้การอ่านและเขียนของรีจิสเตอร์ใน PeliCAN สามารถทำได้รวดเร็วยิ่งขึ้น ถ้าหากในขณะนั้นมีการกำหนดแอดเดรสภายใน CANADR ให้มีค่ามากกว่าหรือเท่ากับ 32 จะส่งผลให้ค่าของ CANADR เพิ่มขึ้นเองโดยอัตโนมัติหลังจากที่มีการอ่านหรือเขียนเข้าถึง CANDAT ยกตัวอย่างเช่น การโหลดข้อมูลไปที่ Transmit Buffer สามารถทำได้ด้วยการเขียนแอดเดรสเริ่มต้นของ Transmit Buffer คือ 112 ให้กับ CANADR ซึ่งจะทำให้เกิดการส่งผ่านของข้อมูลแบบไบต์ต่อไบต์ไปยัง CANDAT โดยการเพิ่มขึ้นของ CANADR ที่เกินกว่า FFh ก็จะเป็นการรีเซ็ทค่าแอดเดรสให้เป็น 00h ส่วนในกรณีของ

CANADR ที่มีค่าต่ำกว่า 32 จะไม่มีการทำงานในโหมดการเพิ่มแอดเดรสแบบอัตโนมัติ โดยที่ CANADR จะทำการเก็บค่าแอดเดรสของตัวเองไว้เสมอ ถ้าหากในขณะนั้น CANDAT ถูกเข้าถึงสำหรับการอ่านหรือเขียนอยู่ ซึ่งจะเป็นการยอมให้มีการโพลลิ่งของรีจิสเตอร์ในพื้นที่แอดเดรสระดับล่างของ PeliCAN

2. CANDAT เป็นรีจิสเตอร์ที่เปรียบได้กับพอร์ทสื่อสารที่เชื่อมต่อตำแหน่งหน่วยความจำของรีจิสเตอร์ภายใน PeliCAN ซึ่งกำหนดแอดเดรสไว้แล้วใน CANADR การเข้าถึงรีจิสเตอร์นี้โดยให้ CANADR มีการเพิ่มแอดเดรสแบบอัตโนมัติได้ ถ้าในขณะนั้นแอดเดรสใน CANADR มีค่ามากกว่าหรือเท่ากับ 32

3. CANMOD ด้วยการอ่านหรือเขียนเข้าถึง CANMOD นี้จะเป็นการเข้าถึงทางตรงไปยัง Mode Register ซึ่งเป็นรีจิสเตอร์ภายใน PeliCAN โดยถูกจัดวางไว้ที่แอดเดรส 00h

4. CANSTA มีการจัดเตรียมการเข้าถึงทางตรงไปยังรีจิสเตอร์ภายใน PeliCAN ได้แก่ Status Register และ Interrupt Enable Register โดยจะขึ้นอยู่กับทิศทางของการเข้าถึง กล่าวคือ การอ่านของ CANSTA จะเป็นการเข้าถึง Status Register ในแอดเดรสที่ 2 และการเขียนของ CANSTA จะเป็นการเข้าถึง Interrupt Enable Register ในแอดเดรสที่ 4

5. CANCON มีการจัดเตรียมการเข้าถึงทางตรงไปยังรีจิสเตอร์ภายใน PeliCAN ได้แก่ Interrupt Register และ Command Register โดยจะขึ้นอยู่กับทิศทางของการเข้าถึง กล่าวคือ การอ่านของ CANCON จะเป็นการเข้าถึง Interrupt ในแอดเดรสที่ 3 และการเขียนของ CANCON จะเป็นการเข้าถึง Command Register ในแอดเดรสที่ 1

3.4 การกำหนดแอดเดรสใน PeliCAN

รีจิสเตอร์ภายใน PeliCAN จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำบนชิปที่จัดวางไว้สำหรับรีจิสเตอร์ภายนอกในส่วนของ 80C51 และเนื่องจาก PeliCAN สามารถที่จะทำงานได้ในโหมดที่แตกต่างกัน ดังนั้นจึงมีการกำหนดลักษณะการทำงานในแต่ละแอดเดรสที่แตกต่างกันด้วย นอกจากนี้ในส่วนของพื้นที่แอดเดรสที่เริ่มต้นจากแอดเดรสที่ 128 จะถูกจัดวางไว้เป็นพื้นที่ของหน่วยความจำภายในแบบ FIFO (internal FIFO RAM) ซึ่งถูกจัดวางไปที่อินเทอร์เฟสของ CPU ดูรายละเอียดได้ในตารางที่ 3.2

ตารางที่ 3.2 แสดงการกำหนดแอดเดรสภายใน PeliCAN

CAN ADDR.	OPERATING MODE		RESET MODE	
	READ	WRITE	READ	WRITE
0	Mode	Mode	Mode	Mode
1	(00)	Command	(00)	Command
2	Status	-	Status	-

3	Interrupt	-	Interrupt	-
4	Interrupt Enable	Interrupt Enable	Interrupt Enable	Interrupt Enable
5	Rx Interrupt Level	Rx Interrupt Level	Rx Interrupt Level	Rx Interrupt Level
6	Bus Timing 0	-	Bus Timing 0	Bus Timing 0
7	Bus Timing 1	-	Bus Timing 1	Bus Timing 1
8	-	-	-	-
9	Rx Message Counter	-	Rx Message Counter	-
10	Rx Buffer Start Address	-	Rx Buffer Start Address	-
11	Arbitration Lost Capture	-	Arbitration Lost Captu e	-
12	Error Code Capture	-	Error Code Capture	-
13	Error Warning Limit	Error Warning Limit	Error Warning Limit	Error Warning Limit
14	Rx Error Counter	-	Rx Error Counter	Rx Error Counter
15	TX Error Counter	-	TX Error Counter	TX Error Counter
16-28	reserved (00)	-	reserved (00)	-
29	ACF Mode	-	ACF Mode	ACF Mode
30	ACF Enable	ACF Enable	ACF Enable	ACF Enable
31	ACF Priority	ACF Priority	ACF Priority	ACF Priority
32	BANK1 Acceptance Code 0	Acceptance Code 0	Acceptance Code 0	Acceptance Code 0
33	BANK1 Acceptance Code 1	Acceptance Code 1	Acceptance Code 1	Acceptance Code 1
34	BANK1 Acceptance Code 2	Acceptance Code 2	Acceptance Code 2	Acceptance Code 2
35	BANK1 Acceptance Code 3	Acceptance Code 3	Acceptance Code 3	Acceptance Code 3
36	BANK1 Acceptance Mask 0	Acceptance Mask 0	Acceptance Mask 0	Acceptance Mask 0
37	BANK1 Acceptance Mask 1	Acceptance Mask 1	Acceptance Mask 1	Acceptance Mask 1
38	BANK1 Acceptance Mask 2	Acceptance Mask 2	Acceptance Mask 2	Acceptance Mask 2
39	BANK1 Acceptance Mask 3	Acceptance Mask 3	Acceptance Mask 3	Acceptance Mask 3
40	BANK2 Acceptance Code 0	Acceptance Code 0	Acceptance Code 0	Acceptance Code 0
41	BANK2 Acceptance Code 1	Acceptance Code 1	Acceptance Code 1	Acceptance Code 1
42	BANK2 Acceptance Code 2	Acceptance Code 2	Acceptance Code 2	Acceptance Code 2
43	BANK2 Acceptance Code 3	Acceptance Code 3	Acceptance Code 3	Acceptance Code 3
44	BANK2 Acceptance Mask 0	Acceptance Mask 0	Acceptance Mask 0	Acceptance Mask 0
45	BANK2 Acceptance Mask 1	Acceptance Mask 1	Acceptance Mask 1	Acceptance Mask 1
46	BANK2 Acceptance Mask 2	Acceptance Mask 2	Acceptance Mask 2	Acceptance Mask 2
47	BANK2 Acceptance Mask 3	Acceptance Mask 3	Acceptance Mask 3	Acceptance Mask 3
48	BANK3 Acceptance Code 0	Acceptance Code 0	Acceptance Code 0	Acceptance Code 0
49	BANK3 Acceptance Code 1	Acceptance Code 1	Acceptance Code 1	Acceptance Code 1
50	BANK3 Acceptance Code 2	Acceptance Code 2	Acceptance Code 2	Acceptance Code 2
51	BANK3 Acceptance Code 3	Acceptance Code 3	Acceptance Code 3	Acceptance Code 3
52	BANK3 Acceptance Mask 0	Acceptance Mask 0	Acceptance Mask 0	Acceptance Mask 0
53	BANK3 Acceptance Mask 1	Acceptance Mask 1	Acceptance Mask 1	Acceptance Mask 1
54	BANK3 Acceptance Mask 2	Acceptance Mask 2	Acceptance Mask 2	Acceptance Mask 2
55	BANK3 Acceptance Mask 3	Acceptance Mask 3	Acceptance Mask 3	Acceptance Mask 3

56	BANK 4	Acceptance Code 0	Acceptance Code 0		Acceptance Code 0		Acceptance Code 0	
57		Acceptance Code 1	Acceptance Code 1		Acceptance Code 1		Acceptance Code 1	
58		Acceptance Code 2	Acceptance Code 2		Acceptance Code 2		Acceptance Code 2	
59		Acceptance Code 3	Acceptance Code 3		Acceptance Code 3		Acceptance Code 3	
60		Acceptance Mask 0	Acceptance Mask 0		Acceptance Mask 0		Acceptance Mask 0	
61		Acceptance Mask 1	Acceptance Mask 1		Acceptance Mask 1		Acceptance Mask 1	
62		Acceptance Mask 2	Acceptance Mask 2		Acceptance Mask 2		Acceptance Mask 2	
63		Acceptance Mask 3	Acceptance Mask 3		Acceptance Mask 3		Acceptance Mask 3	
64-95	reserved (00)		-		reserved (00)		-	
96	(SFF) Rx Fr. Info	(EFF) Rx Fr. Info	-		(SFF) Rx Fr. Info	(EFF) Rx Fr. Info	(SFF) Rx Fr. Info	(EFF) Rx Fr. Info
97	Rx ID. 1	Rx ID. 1	-		Rx ID. 1	Rx ID. 1	Rx ID. 1	Rx ID. 1
98	Rx ID. 2	Rx ID. 2	-		Rx ID. 2	Rx ID. 2	Rx ID. 2	Rx ID. 2
99	Rx Data 1	Rx ID. 3	-		Rx Data 1	Rx ID. 3	Rx Data 1	Rx ID. 3
100	Rx Data 2	Rx ID. 4	-		Rx Data 2	Rx ID. 4	Rx Data 2	Rx ID. 4
101	Rx Data 3	Rx Data 1	-		Rx Data 3	Rx Data 1	Rx Data 3	Rx Data 1
102	Rx Data 4	Rx Data 2	-		Rx Data 4	Rx Data 2	Rx Data 4	Rx Data 2
103	Rx Data 5	Rx Data 3	-		Rx Data 5	Rx Data 3	Rx Data 5	Rx Data 3
103	Rx Data 6	Rx Data 4	-		Rx Data 6	Rx Data 4	Rx Data 6	Rx Data 4
105	Rx Data 7	Rx Data 5	-		Rx Data 7	Rx Data 5	Rx Data 7	Rx Data 5
106	Rx Data 8	Rx Data 6	-		Rx Data 8	Rx Data 6	Rx Data 8	Rx Data 6
107	(FIFO RAM)	Rx Data 7	-		(FIFO RAM)	Rx Data 7	(FIFO RAM)	Rx Data 7
108	(FIFO RAM)	Rx Data 8	-		(FIFO RAM)	Rx Data 8	(FIFO RAM)	Rx Data 8
109-111	reserved (00)		-		reserved (00)		-	
112	(SFF) Tx Fr. Info	(EFF) Tx Fr. Info	(SFF) Tx Fr. Info	(EFF) Tx Fr. Info	(SFF) Tx Fr. Info	(EFF) Tx Fr. Info	(SFF) Tx Fr. Info	(EFF) Tx Fr. Info
112	Tx ID. 1	Tx ID. 1	Tx ID. 1	Tx ID. 1	Tx ID. 1	Tx ID. 1	Tx ID. 1	Tx ID. 1
114	Tx ID. 2	Tx ID. 2	Tx ID. 2	Tx ID. 2	Tx ID. 2	Tx ID. 2	Tx ID. 2	Tx ID. 2
115	Tx Data 1	Tx ID. 3	Tx Data 1	Tx ID. 3	Tx Data 1	Tx ID. 3	Tx Data 1	Tx ID. 3
116	Tx Data 2	Tx ID. 4	Tx Data 2	Tx ID. 4	Tx Data 2	Tx ID. 4	Tx Data 2	Tx ID. 4
117	Tx Data 3	Tx Data 1	Tx Data 3	Tx Data 1	Tx Data 3	Tx Data 1	Tx Data 3	Tx Data 1
118	Tx Data 4	Tx Data 2	Tx Data 4	Tx Data 2	Tx Data 4	Tx Data 2	Tx Data 4	Tx Data 2
119	Tx Data 5	Tx Data 3	Tx Data 5	Tx Data 3	Tx Data 5	Tx Data 3	Tx Data 5	Tx Data 3
120	Tx Data 6	Tx Data 4	Tx Data 6	Tx Data 4	Tx Data 6	Tx Data 4	Tx Data 6	Tx Data 4
121	Tx Data 7	Tx Data 5	Tx Data 7	Tx Data 5	Tx Data 7	Tx Data 5	Tx Data 7	Tx Data 5
122	Tx Data 8	Tx Data 6	Tx Data 8	Tx Data 6	Tx Data 8	Tx Data 6	Tx Data 8	Tx Data 6
123	(TXB mem.)	Tx Data 7	(TXB mem.)	Tx Data 7	(TXB mem.)	Tx Data 7	(TXB mem.)	Tx Data 7
124	(TXB mem.)	Tx Data 8	(TXB mem.)	Tx Data 8	(TXB mem.)	Tx Data 8	(TXB mem.)	Tx Data 8
125-127	General purpose RAM		General purpose RAM		General purpose RAM		General purpose RAM	
128-191	Internal RAM Address 0-63 (FIFO)		-		Internal RAM Address 0-63 (FIFO)		Internal RAM Address 0-63 (FIFO)	

3.5 การรีเซ็ทค่าของรีจิสเตอร์ใน PeliCAN

มีการตรวจสอบการเซ็ทของบิต Reset Mode (RM) ที่เป็นผลให้ยกเลิกการส่งผ่านและการรองรับของข้อมูลในขณะนั้น และปรับโหมดการทำงานเข้าสู่ Reset Mode ด้วยการเปลี่ยนแปลงจาก '1' เป็น '0' ของบิต RM จะทำให้ CAN คอนโทรลเลอร์กลับเข้าสู่โหมดการทำงานที่กำหนดไว้ภายใน Mode Register ในตารางที่ 3.3 จะเป็นการแสดงค่าของรีจิสเตอร์ใน PeliCAN หลังจากการรีเซ็ททางฮาร์ดแวร์ และการเซ็ทบิต RM (MOD.0) ทางซอฟท์แวร์ หรือเข้าสู่สถานะ Bus Off

ตารางที่ 3.3 แสดงโครงสร้างของ Reset Mode

ADDR.	REGISTER	BIT	NAME	RESET BY HARDWARE	RESET BY SOFTWARE OR BUS-OFF
0	Mode	MOD.7	Test Mode	0 (disabled)	0 (disabled)
		MOD.6	-	X (reserved)	X (reserved)
		MOD.5	Receive Polarity Mode	0 (active low)	0 (active high)
		MOD.4	Sleep Mode	0 (wake-up)	0 (wake-up)
		MOD.3	-	0 (reserved)	0 (reserved)
		MOD.2	Self Test Mode	0 (normal)	X no change
		MOD.1	Listen Only Mode	0 (normal)	X no change
		MOD.0	Reset Mode	1 (present)	1 (present)
1	Command	CMR.7-5	-	0 (reserved)	0 (reserved)
		CMR.4	Self Reception Request	0 (absent)	0 (absent)
		CMR.3	Clear Data Overrun	0 (no action)	0 (no action)
		CMR.2	Release Receive Buffer	0 (no action)	0 (no action)
		CMR.1	Abort Transmission	0 (absent)	0 (absent)
		CMR.0	Transmission Request	0 (absent)	0 (absent)
2	Status	SR.7	Bus Status	0 (Bus-On)	0 (reset)
		SR.6	Error Status	0 (ok)	0 (reset)
		SR.5	Transmit Status	1 (wait idle)	0 (reset)
		SR.4	Receive Status	1 (wait idle)	0 (reset)
		SR.3	Transmission Complete Status	1 (complete)	0 (reset)
		SR.2	Transmit Buffer Status	1 (released)	X no change
		SR.1	Data Overrun Status	0 (absent)	0 (reset)
		SR.0	Receive Buffer Status	0 (empty)	0 (reset)
3	Interrupt	IR.7	Bus Error Interrupt	0 (reset)	X no change
		IR.6	Arbitration Lost Interrupt	0 (reset)	0 (reset)
		IR.5	Error Passive Interrupt	0 (reset)	0 (reset)
		IR.4	Wake-Up Interrupt	0 (reset)	0 (reset)
		IR.3	Data Overrun Interrupt	0 (reset)	0 (reset)
		IR.2	Error Warning Interrupt	0 (reset)	X no change

		IR.1	Transmit Interrupt	0 (reset)	0 (reset)
		IR.0	Receive Interrupt	0 (reset)	0 (reset)
4	Interrupt Enable	IER.7	Bus Error Interrupt Enable	X no change	X no change
		IER.6	Arbitr. Lost Interrupt Enable	X no change	X no change
		IER.5	Error Passive Interrupt	X no change	X no change
		IER.4	Wake-Up Interrupt Enable	X no change	X no change
		IER.3	Data Overrun Interrupt Enable	X no change	X no change
		IER.2	Error Warning Interrupt Enable	X no change	X no change
		IER.1	Transmit Interrupt Enable	X no change	X no change
		IER.0	Receive Interrupt Enable	X no change	X no change
5	Rx Interrupt Level	-	Rx Interrupt Level	00000000b	X no change
6	Bus Timing 0	BTR0.7	Synchronization Jump Width 1	X no change	X no change
		BTR0.6	Synchronization Jump Width 0	X no change	X no change
		BTR0.5	Baud Rate Prescaler 5	X no change	X no change
		BTR0.4	Baud Rate Prescaler 4	X no change	X no change
		BTR0.3	Baud Rate Prescaler 3	X no change	X no change
		BTR0.2	Baud Rate Prescaler 2	X no change	X no change
		BTR0.1	Baud Rate Prescaler 1	X no change	X no change
		BTR0.0	Baud Rate Prescaler 0	X no change	X no change
7	Bus Timing 1	BTR1.7	Sampling	X no change	X no change
		BTR1.6	Time Segment 2.2	X no change	X no change
		BTR1.5	Time Segment 2.1	X no change	X no change
		BTR1.4	Time Segment 2.0	X no change	X no change
		BTR1.3	Time Segment 1.3	X no change	X no change
		BTR1.2	Time Segment 1.2	X no change	X no change
		BTR1.1	Time Segment 1.1	X no change	X no change
		BTR1.0	Time Segment 1.0	X no change	X no change
9	Rx Message Counter	-	Rx Message Counter	0	0
10	Rx Buffer Start Address	-	Rx Buffer Start Address	00000000b	X no change
11	Arbitration Lost Capture	-	Arbitration Lost Capture	0	X no change
12	Error Code Capture	-	Error Code Capture	0	X no change
13	Error Warning Limit	-	Error Warning Limit Register	96d	X no change
14	Rx Error Counter	-	Receive Error Counter	0 (reset)	X no change
15	TX Error Counter	-	Transmit Error Counter	0 (reset)	X no change
29	ACF Mode	ACFMOD.7	Message Format Bank4	0 (SFF)	X no change
		ACFMOD.6	Accept. Filt. Mode Bank4	0 (dual)	X no change
		ACFMOD.5	Message Format Bank3	0 (SFF)	X no change
		ACFMOD.4	Accept. Filt. Mode Bank3	0 (dual)	X no change
		ACFMOD.3	Message Format Bank2	0 (SFF)	X no change
		ACFMOD.2	Accept. Filt. Mode Bank2	0 (dual)	X no change
		ACFMOD.1	Message Format Bank1	0 (SFF)	X no change
		ACFMOD.0	Accept. Filt. Mode Bank1	0 (dual)	X no change

30	ACF Enable		ACFEN.7	Bank 4 Filter 2 Enable	X no change	X no change
			ACFEN.6	Bank 4 Filter 1 Enable	X no change	X no change
			ACFEN.5	Bank 3 Filter 2 Enable	X no change	X no change
			ACFEN.4	Bank 3 Filter 1 Enable	X no change	X no change
			ACFEN.3	Bank 2 Filter 2 Enable	X no change	X no change
			ACFEN.2	Bank 2 Filter 1 Enable	X no change	X no change
			ACFEN.1	Bank 1 Filter 2 Enable	X no change	X no change
			ACFEN.0	Bank 1 Filter 1 Enable	X no change	X no change
31	ACF Priority		ACFPRIO.7	Bank 4 Filter 2 Priority	X no change	X no change
			ACFPRIO.6	Bank 4 Filter 1 Priority	X no change	X no change
			ACFPRIO.5	Bank 3 Filter 2 Priority	X no change	X no change
			ACFPRIO.4	Bank 3 Filter 1 Priority	X no change	X no change
			ACFPRIO.3	Bank 2 Filter 2 Priority	X no change	X no change
			ACFPRIO.2	Bank 2 Filter 1 Priority	X no change	X no change
			ACFPRIO.1	Bank 1 Filter 2 Priority	X no change	X no change
			ACFPRIO.0	Bank 1 Filter 1 Priority	X no change	X no change
32-35	Bank 1	ACR0 - ACR3	-	Acceptance Code Register	X no change	X no change
36-39		AMR0 - AMR3	-	Acceptance Mask Register	X no change	X no change
40-43	Bank 2	ACR0 - ACR3	-	Acceptance Code Register	X no change	X no change
44-47		AMR0 - AMR3	-	Acceptance Mask Register	X no change	X no change
48-51	Bank 3	ACR0 - ACR3	-	Acceptance Code Register	X no change	X no change
52-55		AMR0 - AMR3	-	Acceptance Mask Register	X no change	X no change
56-59	Bank 4	ACR0 - ACR3	-	Acceptance Code Register	X no change	X no change
60-63		AMR0 - AMR3	-	Acceptance Mask Register	X no change	X no change
96-108	Rx Buffer			Receive Buffer	X empty	X empty
112-124	Tx Buffer			Transmit Buffer	X no change	X no change
125-127	General Purpose RAM			General Purpose RAM	X no change	X no change

3.6 รายละเอียดของรีจิสเตอร์ใน PeliCAN

ในหัวข้อนี้จะกล่าวถึงรายละเอียดของบิตและหน้าที่ของรีจิสเตอร์ใน PeliCAN เพื่อให้เกิดความเข้าใจในรูปแบบของฟังก์ชันการทำงานและการโปรแกรมมากขึ้น

3.6.1 Mode Register (MOD)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 0 ใช้สำหรับการเปลี่ยนแปลงโหมดการทำงานของ CAN คอนโทรลเลอร์ ซึ่งการเปลี่ยนแปลงในแต่ละครั้งจำเป็นจะต้องเข้าสู่ Reset Mode ก่อนหน้านี้ จึงสามารถเปลี่ยนเป็นโหมดการทำงานที่กำหนดได้ โดยบิตที่อยู่ภายในรีจิสเตอร์นี้สามารถถูกเซ็ทหรือรีเซ็ทได้จาก CPU ซึ่งเป็นการใช้ Mode Register เช่นเดียวกับหน่วยความจำสำหรับการอ่านและเขียน และบิตที่จองไว้จะถูกอ่านเป็น '0'

ตารางที่ 3.4 แสดงรายละเอียดภายในของ Mode Register

BIT	NAME	VALUE	FUNCTION
MOD.7	Test Mode (TM)	1 (activated) 0 (disabled)	ขา TXDC จะสะท้อนบิตเพื่อการตรวจสอบให้กับขา RXDC และบิต RPM ต้องไม่มีผลกระทบภายในโหมดนี้
MOD.6	Reserved.	-	-
MOD.5	Receive Polarity Mode (RPM)	1 (high active) 0 (low active)	อินพุทของ RXDC ทำงานที่สัญญาณศักย์สูง (dominant = 1) อินพุทของ RXDC ทำงานที่สัญญาณศักย์ต่ำ (dominant = 0)
MOD.4	Sleep Mode (SM)	1 (high active) 0 (low active)	CAN คอนโทรลเลอร์ จะปรับเข้าสู่ Sleep Mode ถ้าไม่อยู่ในช่วงการบริการอินเทอร์รัพท์ของ CAN และไม่มีการใช้งานบนบัส
MOD.3	Reserved.	-	-
MOD.2	Self Test Mode (STM)	1 (self test) 0 (normal)	CAN คอนโทรลเลอร์ ทำการทดสอบตัวเองได้โดยไม่มีโหนดอื่นต่อร่วมบนบัสด้วยการใช้คำสั่ง Self Reception Request (SRR) ซึ่งการส่งผ่านข้อมูลเป็นผลสำเร็จได้ถึงแม้จะไม่ได้รับ ACK ต้องการรับ ACK เพื่อให้การส่งผ่านข้อมูลที่เป็นผลสำเร็จได้
MOD.1	Listen Only Mode (LOM)	1 (reset) 0 (normal)	CAN คอนโทรลเลอร์ จะไม่ส่ง ACK ไปที่บัส ถึงแม้ข้อมูลที่รับเข้ามาจะเป็นผลสำเร็จก็ตาม โดยจะไม่เกิด Active Error Flag และตัวนับความผิดพลาดจะถูกหยุดไว้ที่ค่าเดิม การสื่อสารข้อมูลบนบัสเป็นปกติ
MOD.0	Reset Mode (RM)	1 (reset) 0 (normal)	CAN คอนโทรลเลอร์ จะยกเลิกการส่งผ่านและการรองรับข้อมูลในขณะนั้น และปรับเข้าสู่ Reset Mode การเปลี่ยนแปลงจาก '1' เป็น '0' ทำให้ CAN คอนโทรลเลอร์ กลับเข้าสู่ Operating Mode

3.6.2 Command Register (CMR)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 1 ใช้สำหรับเป็นคำสั่งในการจัดการและควบคุมคุณสมบัติของ CAN คอนโทรลเลอร์ โดยบิตภายในสามารถถูกเซ็ทหรือรีเซ็ทได้จาก CPU ซึ่งเป็นการใช้ Command Register เช่นเดียวกับหน่วยความจำสำหรับการเขียนเท่านั้น

ตารางที่ 3.5 แสดงรายละเอียดภายในของ Command Register

BIT	NAME	VALUE	FUNCTION
CMR.7 - 5	Reserved	-	
CMR.4	Self Reception Request (SRR)	1 (present) 0 (absent)	ข้อมูลจะถูกส่งและรับในเวลาเดียวกัน

CMR.3	Clear Data Overrun (CDO)	1 (clear) 0 (no action)	ทำการเคลียร์บิต Data Overrun Status
CMR.2	Release Receive Buffer (RRB)	1 (released) 0 (no action)	ยกเลิกการใช้งาน Receive Buffer
CMR.1	Abort Transmission (AT)	1 (present) 0 (absent)	ยกเลิกบิต Transmission Request ถ้าการส่งข้อมูลยังไม่มีความคืบหน้า
CMR.0	Transmission Request (TR)	1 (present) 0 (absent)	ข้อมูลจะถูกส่งออกไปบนบัส

3.6.3 Status Register (SR)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 2 ใช้สำหรับการแสดงสภาวะที่กระทำในขณะนั้นของ CAN คอนโทรลเลอร์ โดยที่ Status Register จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านเท่านั้น

ตารางที่ 3.6 แสดงรายละเอียดภายในของ Status Register

BIT	NAME	VALUE	FUNCTION
SR.7	Bus Status (BS)	1 (Bus-Off) 0 (Bus-On)	CAN คอนโทรลเลอร์ ไม่สามารถใช้งานบนบัสได้ CAN คอนโทรลเลอร์ สามารถใช้งานบนบัสได้
SR.6	Error Status (ES)	1 (error) 0 (ok)	ตัวนับความผิดพลาดอย่างน้อยหนึ่งชนิดมีค่ามากกว่าหรือเท่ากับขอบเขตการเตือนของ CPU ตัวนับความผิดพลาดทั้งสองชนิดมีค่าต่ำกว่าขอบเขตการเตือนของ CPU
SR.5	Transmit Status (TS)	1 (transmit) 0 (idle)	CAN คอนโทรลเลอร์ กำลังส่งข้อมูลอยู่
SR.4	Receive Status (RS)	1 (receive) 0 (idle)	CAN คอนโทรลเลอร์ กำลังรับข้อมูลอยู่
SR.3	Transmission Complete Status (TCS)	1 (complete) 0 (incomplete)	การส่งผ่านข้อมูลครั้งล่าสุดเป็นผลสำเร็จเรียบร้อยแล้ว การส่งผ่านข้อมูลครั้งก่อนยังไม่เป็นผลสำเร็จ
SR.2	Transmit Buffer Status (TBS)	1 (released) 0 (locked)	CPU สามารถคัดลอกข้อมูลไปยัง Transmit Buffer ได้ CPU ไม่สามารถคัดลอกข้อมูลไปยังTransmit Buffer ได้
SR.1	Data Overrun Status (DOS)	1 (overrun) 0 (absent)	ข้อมูลเกิดสูญเสีย เนื่องจากมีพื้นที่ใน RXFIFO ไม่เพียงพอ ไม่เกิดขึ้นอีกตั้งแต่ใช้คำสั่ง Clear Data Overrun คราวก่อน
SR.0	Receive Buffer Status (RBS)	1 (full) 0 (empty)	มีข้อมูลที่สมบูรณ์พร้อมใช้งานอยู่ใน RXFIFO ไม่มีข้อมูลสำหรับใช้งานอยู่เลยใน RXFIFO

3.6.4 Interrupt Register (IR)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 3 ใช้สำหรับการแสดงให้ CPU ได้รับทราบเกี่ยวกับอินเทอร์รัพท์จากแหล่งต่าง ๆ หลังจากการอ่านค่าโดย CPU แล้วทุกบิตจะถูกรีเซ็ททั้งหมดยกเว้นบิต Receive Interrupt ที่ถูกเคลียร์เมื่อใช้คำสั่ง Relesease Receive Buffer โดยที่ Interrupt Register จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านเท่านั้น

ตารางที่ 3.7 แสดงรายละเอียดภายในของ Interrupt Register

BIT	NAME	VALUE	FUNCTION
IR.7	Bus Error Interrupt (BEI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่อตรวจพบความผิดพลาด และบิต BEIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้ โดยจะล็อกอินเทอร์รัพท์นี้จนกว่ามีการอ่านค่าจาก Error Code Capture Register เสียก่อน
IR.6	Arbitration Lost Interrupt (ALI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่อแพ้ในการตัดสินเข้าใช้บัส และบิต ALIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้ โดยจะทำการล็อกอินเทอร์รัพท์นี้ไว้จนกว่ามีการอ่านค่าจาก Arbitration Lost Capture Register เสียก่อน
IR.5	Error Passive Interrupt (EPI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ททุกครั้งที่มีการปรับเข้าสู่สถานะ " Error Passive " หรือกลับกัน และบิต EPIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้
IR.4	Wake-Up Interrupt (WUI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่ออยู่ในช่วงหยุดพักและตรวจพบการใช้งานบนบัสขึ้น และบิต WUIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้
IR.3	Data Overrun Interrupt (DOI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่อบิต Data Overrun Status เปลี่ยนแปลงจาก '0' เป็น '1' และบิต DOIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้
IR.2	Error Interrupt (EI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่อบิต Bus Status หรือบิต Error Status เปลี่ยนแปลง และบิต EIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้
IR.1	Transmit Interrupt (TI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่อบิต Transmit Buffer Status เปลี่ยนแปลงจาก '0' เป็น '1' และบิต TIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้
IR.0	Receive Interrupt (RI)	1 (set) 0 (reset)	บิตนี้ถูกเซ็ทเมื่อ RXFIFO มีจำนวนไบต์ข้อมูลเกินกว่าที่กำหนดไว้ใน RX Interrupt Level Register หรือมีข้อมูลที่เป็น " High Priority " และบิต RIE ต้องถูกเซ็ทเพื่อยอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้

3.6.5 Interrupt Enable Register (IER)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 4 ใช้สำหรับการจัดการและควบคุมการอินเทอร์รัพท์จากแหล่งต่าง ๆ ว่าต้องการให้ทำอินเทอร์รัพท์หรือไม่ จากการกำหนดบิตภายในรีจิสเตอร์นี้ โดยที่ Interrupt Enable Register จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านและเขียน

ตารางที่ 3.8 แสดงรายละเอียดภายในของ Interrupt Enable Register

BIT	NAME	VALUE	FUNCTION
IER.7	Bus Error Interrupt Enable (BEIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์ถ้าตรวจพบความผิดพลาดบนบัส
IER.6	Arbitration Lost Interrupt Enable (ALIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์ถ้าเกิดแพ้ในการตัดสินเข้าใช้บัส
IER.5	Error Passive Interrupt Enable (EPIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์ถ้ามีการเปลี่ยนแปลงจากสถานะ “ Error Active ” เป็นสถานะ “ Error Passive ” หรือกลับกัน
IER.4	Wake-Up Interrupt Enable (WUIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์ถ้าเกิดมีการทำงานในช่วงหยุดพัก
IER.3	Data Overrun Interrupt Enable (DOIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์ถ้าบิต Data Overrun Status ถูกเซ็ท
IER.2	Error Interrupt Enable (EIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์ถ้าบิต Bus Status หรือ Error Status มีการเปลี่ยนแปลง
IER.1	Transmit Interrupt Enable (TIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์เมื่อทำการส่งข้อมูลเป็นผลสำเร็จ หรือ Transmit Buffer สามารถเข้าถึงได้อีกครั้ง ยกตัวอย่าง เช่น หลังจากการใช้คำสั่ง Abort Transmission
IER.0	Receive Interrupt Enable (RIE)	1 (enabled) 0 (disabled)	ยอมให้ทำอินเทอร์รัพท์เมื่อบิต Receive Buffer Status ถูกเซ็ท

3.6.6 Rx Interrupt Level (RIL)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 5 ใช้สำหรับกำหนดขอบเขตของ Receive Interrupt สำหรับ RXFIFO ซึ่งจะเกิดขึ้นถ้าจำนวนไบต์ข้อมูลของ CAN ที่อยู่ใน RXFIFO เกินกว่าขอบเขตที่กำหนดไว้ในรีจิสเตอร์นี้ และถ้าหากเซ็ทบิตภายในรีจิสเตอร์นี้ให้เป็น 00h จะทำให้ PeliCAN มีฟังก์ชันการทำงานเหมือนกับ SJA1000

ตารางที่ 3.9 แสดงรายละเอียดภายในของ Rx Interrupt Level

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
RIL.7	RIL.6	RIL.5	RIL.4	RIL.3	RIL.2	RIL.1	RIL.0

3.6.7 BUS Timing Register 0 (BTR 0)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 6 ใช้สำหรับการควบคุมความเร็วบนบัส โดยการกำหนด Baud Rate Prescaler (BRP) และ Synchronization Jump Width (SJW) โดยรีจิสเตอร์นี้สามารถเข้าถึงสำหรับการอ่านและเขียนถ้าอยู่ใน Reset Mode แต่ถ้าอยู่ใน Operating Mode จะทำได้เพียงการอ่านเท่านั้น

ตารางที่ 3.10 แสดงรายละเอียดภายในของ BUS Timing Register 0

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

1. Baud Rate Prescaler (BRP) ใช้สำหรับโปรแกรมคาบเวลาสัญญาณนาฬิกาของระบบ CAN และการกำหนดความเร็วบนบัส โดยที่สัญญาณนาฬิกาของระบบ CAN (t_{scl}) สามารถคำนวณได้จากสมการต่อไปนี้

$$t_{scl} = t_{CLK} \times (32 \times BRP.5 + 16 \times BRP.4 + 8 \times BRP.3 + 4 \times BRP.2 + 2 \times BRP.1 + BRP.0 + 1)$$

โดยที่ t_{CLK} เป็นคาบเวลาสัญญาณนาฬิกาของระบบไมโครคอนโทรลเลอร์

2. Synchronization Jump Width (SJW) ใช้สำหรับการชดเชยสำหรับเฟสที่คลาดเคลื่อนไประหว่างสัญญาณออสซิลเลเตอร์จากบัสคอนโทรลเลอร์ที่แตกต่างกัน ซึ่งจะต้องรีซิงโครไนซ์บนขอบสัญญาณของการส่งผ่านในขณะนั้น โดย SJW เป็นการกำหนดจำนวนรอบสูงสุดของสัญญาณนาฬิกาในการรีซิงโครไนซ์ สามารถคำนวณได้จากสมการต่อไปนี้

$$T_{SJW} = t_{scl} \times (2 \times SJW.1 + SJW.0 + 1)$$

3.6.8 Bus Timing Register 1 (BTR 1)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 7 ใช้สำหรับการกำหนดความยาวของคาบเวลาบิต ตำแหน่งของจุด Sampling และจำนวนการ Sampling สัญญาณในแต่ละเวลาบิต โดยรีจิสเตอร์นี้สามารถเข้าถึงสำหรับการอ่านและเขียนถ้าอยู่ใน Reset Mode แต่ถ้าอยู่ใน Operating Mode จะทำได้เพียงการอ่านเท่านั้น

ตารางที่ 3.11 แสดงรายละเอียดภายในของ Bus Timing Register 1

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0

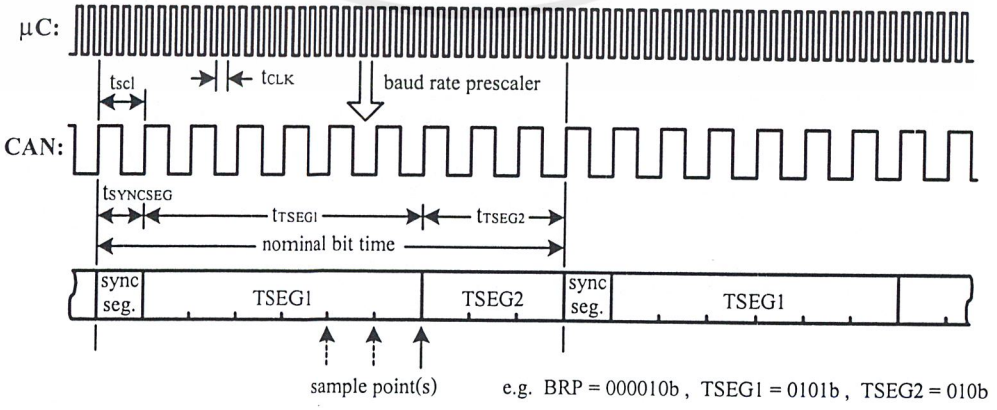
1. Sampling (SAM) ใช้สำหรับการกำหนดจำนวนการ Sampling สัญญาณในแต่ละเวลาบิต ถ้าเซ็ทเป็น '1' จะเป็นการกำหนดให้บัสถูก Sampling จำนวน 3 ครั้ง ใช้สำหรับบัสที่มีความเร็วระดับต่ำและปานกลางซึ่งอยู่ใน SAE คลาส A และ B แต่ถ้าเคลียร์เป็น '0' จะเป็นการกำหนดให้บัสถูก Sampling เพียงครั้งเดียวใช้สำหรับบัสที่มีความเร็วสูง ซึ่งอยู่ใน SAE คลาส C

2. Time Segment 1 (TSEG 1) และ Time Segment 2 (TSEG 2) ใช้สำหรับการกำหนดจำนวนรอบของสัญญาณนาฬิกาต่อคาบเวลาบิต และตำแหน่งของจุด Sampling บนบัส ซึ่งสามารถคำนวณได้จากสมการต่อไปนี้

$$t_{SYNCEG} = 1 \times t_{scl}$$

$$t_{TSEG1} = t_{scl} \times (8 \times TSEG1.3 + 4 \times TSEG1.2 + 2 \times TSEG1.1 + TSEG1.0 + 1)$$

$$t_{TSEG2} = t_{scl} \times (4 \times TSEG2.2 + 2 \times TSEG2.1 + TSEG2.0 + 1)$$



ภาพที่ 3.4 แสดงโครงสร้างทั่วไปของคาบเวลาบิต

3.6.9 RX Message Counter (RMC)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 9 ใช้สำหรับการแสดงจำนวนข้อมูลที่มีอยู่ทั้งหมดใน RXFIFO โดยจะมีค่าเพิ่มขึ้นเมื่อทำการรับข้อมูลในแต่ละครั้ง และจะมีค่าลดลงเมื่อมีการใช้คำสั่ง Release Receive Buffer หลังจากที่มีการรีเซ็ทเกิดขึ้นค่าในรีจิสเตอร์นี้จะถูกเคลียร์ทั้งหมด

ตารางที่ 3.12 แสดงรายละเอียดภายในของ RX Message Counter

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
RMC.7	RMC.6	RMC.5	RMC.4	RMC.3	RMC.2	RMC.1	RMC.0

3.6.10 RX Buffer Start Address (RBSA)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 10 ใช้สำหรับการแสดงตำแหน่งแอดเดรสปัจจุบันของ RAM ภายใน โดยเป็นไบต์เริ่มต้นของข้อมูลที่จะถูกจัดวางเป็น Receive Buffer Window เพื่อแสดงให้ CPU สามารถเข้าถึงได้ ใน CAN คอนโทรลเลอร์ จะมีพื้นที่แอดเดรสของ RAM ภายในโดยจะเริ่มต้นในแอดเดรสที่ 32 และจะถูกเข้าถึงสำหรับการอ่านและเขียนได้จาก CPU ซึ่งการเขียนจะต้องอยู่ใน Reset Mode เท่านั้น

ตารางที่ 3.13 แสดงรายละเอียดภายในของ RX Buffer Start Address

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
RBSA.7	RBSA.6	RBSA.5	RBSA.4	RBSA.3	RBSA.2	RBSA.1	RBSA.0

ยกตัวอย่างเช่น ถ้าหากรีจิสเตอร์ RBSA ถูกเซ็ทให้มีค่าเท่ากับ 24 จะทำให้ข้อมูลปัจจุบันใน Receive Buffer Window (CAN แอดเดรสที่ 96-108) ถูกจัดเก็บไว้ที่ RAM ภายในโดยเริ่มต้นใน RAM แอดเดรสที่ 24 เนื่องจาก RAM ถูกจัดวางสำหรับพื้นที่แอดเดรสของ CAN โดยตรงด้วย เริ่มต้นใน CAN แอดเดรสที่ 128 ซึ่งจะเท่ากับใน RAM แอดเดรสที่ 0 จึงทำให้ข้อมูลนี้อาจถูกเข้าถึงได้เช่นเดียวกับใน CAN แอดเดรสที่ 152 เป็นไบต์เริ่มต้น ซึ่งสามารถคำนวณได้จากสมการต่อไปนี้

$$\text{CAN Address} = \text{RBSA} + 128 = 24 + 128 = 152$$

ทุกครั้งที่มีการใช้คำสั่ง Release Receive Buffer ในขณะที่มีข้อมูลอย่างน้อยที่สุดหนึ่งหรือมากกว่าอยู่ใน FIFO จะทำให้รีจิสเตอร์ RBSA ถูกปรับค่าใหม่เป็นแอดเดรสเริ่มต้นของข้อมูลที่อยู่ถัดไป สำหรับการรีเซ็ททางฮาร์ดแวร์จะปรับเป็นค่าเริ่มต้นที่ 00h ส่วนการรีเซ็ททาง

ซอฟท์แวร์จะยังคงเก็บรักษาค่าเดิมของตัวเองไว้ แต่ภายใน FIFO จะถูกเคลียร์ทั้งหมด ซึ่งหมายถึงข้อมูลใน RAM ไม่ถูกเปลี่ยนแปลง แต่ข้อมูลถัดไปที่รับเข้ามานั้นจะแทนที่ข้อมูลปัจจุบันภายใน Receive Buffer Window โดยรีจิสเตอร์นี้จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านและเขียนใน Reset Mode และหน่วยความจำสำหรับการอ่านเท่านั้นใน Operating Mode

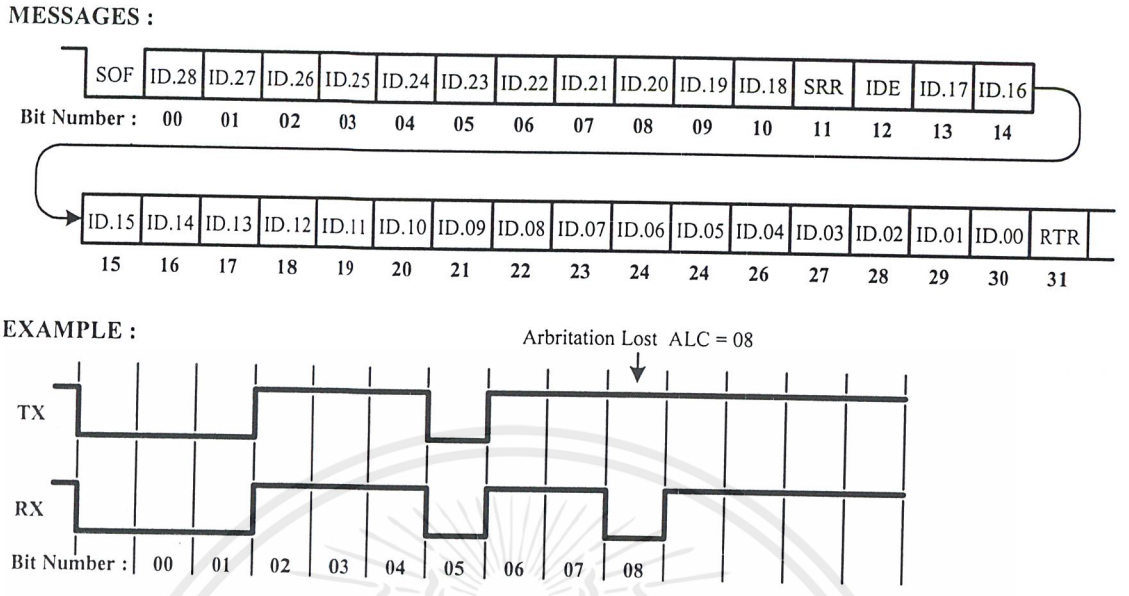
3.6.11 Arbitration Lost Capture (ALC)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 11 ใช้สำหรับการแสดงตำแหน่งบิตข้อมูลที่แพ้ในการตัดสินเข้าใช้บัสแล้วทำให้ต้องเปลี่ยนมาเป็นโหนดผู้รับแทน โดยรีจิสเตอร์นี้จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านเท่านั้น

ตารางที่ 3.14 แสดงรายละเอียดภายในของ Arbitration Lost Capture

BIT	NAME	VALUE	FUNCTION
7 - 5	-	-	Reserved.
4	Bit Number 4 (BITNO4)	รหัสไบนารีแสดงหมายเลขบิตภายในเฟรมที่แพ้ในการตัดสินเข้าใช้บัส 00 -> เกิดขึ้นในตำแหน่งบิตแรกของหมายเลข ID ... 10 -> เกิดขึ้นในตำแหน่งบิตที่ 10 ของหมายเลข ID 11 -> เกิดขึ้นในตำแหน่งบิต SRR (บิต RTR ในรูปแบบ SFF) 12 -> เกิดขึ้นในตำแหน่งบิต IDE 13 -> เกิดขึ้นในตำแหน่งบิตที่ 12 ของหมายเลข ID (รูปแบบ EFF เท่านั้น) ... 30 -> เกิดขึ้นในตำแหน่งบิตสุดท้ายของหมายเลข ID (รูปแบบ EFF เท่านั้น) 31 -> เกิดขึ้นในตำแหน่งบิต RTR (รูปแบบ EFF เท่านั้น)
3	Bit Number 3 (BITNO3)		
2	Bit Number 2 (BITNO2)		
1	Bit Number 1 (BITNO1)		
0	Bit Number 0 (BITNO0)		

ในขณะที่แพ้ในการตัดสินเข้าใช้บัสจะทำให้เกิด Arbitration Lost Interrupt (ALI) ถ้ายอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้ และตำแหน่งบิตปัจจุบันใน Bit Stream Processor (BSP) จะถูกตรวจจับค่าเก็บไว้ในรีจิสเตอร์นี้ โดยถูกเก็บรักษาไว้จนกว่าจะมีการอ่านค่าออกไปครั้งหนึ่งก่อนจึงจะสามารถเริ่มวิธีการตรวจจับบิตได้ใหม่ ซึ่งบิต ALI จะถูกเคลียร์ในระหว่างการอ่านเข้าถึง Interrupt Register นั่นหมายความว่า อินเทอร์รัพท์ครั้งใหม่จะไม่สามารถเกิดขึ้นได้ถ้ารีจิสเตอร์นี้ยังไม่ถูกอ่านค่าออกไปครั้งหนึ่งก่อน



ภาพที่ 3.5 แสดงหมายเลขบิตภายในเฟรมที่แพ้ในการตัดสินเข้าใช้บัส

3.6.12 Error Code Capture (ECC)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 12 ใช้สำหรับการแสดงชนิดของความผิดพลาด และตำแหน่งของบิตข้อมูลที่เกิดความผิดพลาดขึ้น โดยรีจิสเตอร์นี้จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านเท่านั้น

ตารางที่ 3.15 แสดงรายละเอียดภายในของ Error Code Capture

BIT	NAME	VALUE		FUNCTION
7	Error Code 1 (ERRC1)	ERRC1	ERRC0	
6	Error Code 0 (ERRC0)	0	0	Bit Error
		0	1	Form Error
		1	0	Stuff Error
		1	1	Other Error
5	Direction (DIR)	1 (RX)		เกิดความผิดพลาดในระหว่างการรับข้อมูล
		0 (TX)		เกิดความผิดพลาดในระหว่างการส่งข้อมูล
4	Segment 4 (SEG4)	แสดงส่วนการทำงานของเฟรมที่เกิดความผิดพลาด		
3	Segment 3 (SEG3)	00011 Start Of Frame		
2	Segment 2 (SEG2)	00010 ID28...ID21		
1	Segment 1 (SEG1)	00110 ID20...ID18		
0	Segment 0 (SEG0)	00100 SRTR Bit		

		00101 IDE Bit 00111 ID17...ID13 01111 ID12...ID5 01110 ID4...ID0 01100 RTR Bit 01101 Reserved Bit 1 01001 Reserved Bit 0 01011 Data Length Code 01010 Data Field 01000 CRC Sequence 11000 CRC Delimiter 11001 Acknowledge Slot 11011 Acknowledge Delimiter 11010 End Of Frame 10010 Intermission 10001 Activc Error Flag 10110 Passive Error Flag 10011 Tolerate Dom. Bits 10111 Error Delimiter 11100 Overload Flag

ในขณะที่มีความผิดพลาดบนบัสเกิดขึ้น จะทำให้เกิด Bus Error Interrupt (BEI) ถ้ายอมให้ทำอินเทอร์รัพท์จากแหล่งนี้ได้ และตำแหน่งบิตปัจจุบันใน Bit Stream Processor (BSP) จะถูกตรวจจับค่าเก็บไว้ในรีจิสเตอร์นี้ โดยถูกเก็บรักษาไว้จนกว่าจะมีการอ่านค่าออกไปครั้งหนึ่งก่อนจึงจะสามารถเริ่มวิธีการตรวจจับบิตได้ใหม่ ซึ่งบิต BEI จะถูกเคลียร์ในระหว่างการอ่านเข้าถึง Interrupt Register นั่นหมายความว่า อินเทอร์รัพท์ครั้งใหม่จะไม่สามารถเกิดขึ้นได้ถ้ารีจิสเตอร์นี้ยังไม่ถูกอ่านค่าออกไปครั้งหนึ่งก่อน

3.6.13 Error Warning Limit Register (EWLR)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 13 ใช้สำหรับการกำหนดขอบเขตของการเตือนความผิดพลาดเพื่อแจ้งให้กับ CPU ได้รับทราบก่อนที่จะปรับสถานะความผิดพลาด โดยหลังจากการรีเซ็ททางฮาร์ดแวร์จะมีค่าเป็น 96 โดยรีจิสเตอร์นี้จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านและเขียนใน Reset Mode

ตารางที่ 3.16 แสดงรายละเอียดภายในของ Error Warning Limit Register

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
EWL.7	EWL.6	EWL.5	EWL.4	EWL.3	EWL.2	EWL.1	EWL.0

การเปลี่ยนแปลงค่าของรีจิสเตอร์นี้สามารถทำได้ ถ้ามีการปรับเข้าสู่ Reset Mode แล้วก่อนหน้านี้ Error Status และ Error Warning Interrupt สำหรับค่าที่เปลี่ยนแปลงใหม่นี้จะไม่เกิดขึ้นจนกว่า Reset Mode ถูกยกเลิกอีกครั้ง

3.6.14 RX Error Counter Register (RXERR)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 14 ใช้สำหรับการแสดงค่าปัจจุบันของตัวนับความผิดพลาดสำหรับการรับ (Receive Error Counter) โดยหลังจากการรีเซ็ททางฮาร์ดแวร์จะมีค่าเริ่มต้นที่ 0 แต่ถ้าเข้าสู่สถานะ Bus off รีจิสเตอร์นี้จะถูกปรับเป็นค่าเริ่มต้นที่ 0 ในสถานะนี้จะทำให้การเขียนเข้าถึงรีจิสเตอร์ไม่เป็นผล โดยรีจิสเตอร์นี้จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านเท่านั้นใน Operating Mode และสามารถเขียนเข้าถึงได้ใน Reset Mode เท่านั้น

ตารางที่ 3.17 แสดงรายละเอียดภายในของ RX Error Counter Register

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
RXERR.7	RXERR.6	RXERR.5	RXERR.4	RXERR.3	RXERR.2	RXERR.1	RXERR.0

การเปลี่ยนแปลงค่าของรีจิสเตอร์นี้สามารถทำได้ ถ้ามีการปรับเข้าสู่ Reset Mode แล้วก่อนหน้านี้ Error Status และ Error Warning Interrupt หรือ Error Passive Interrupt สำหรับค่าที่เปลี่ยนแปลงใหม่นี้จะไม่เกิดขึ้นจนกว่า Reset Mode ถูกยกเลิกอีกครั้ง

3.6.15 TX Error Counter Register (TXERR)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 15 ใช้สำหรับการแสดงค่าปัจจุบันของตัวนับความผิดพลาดสำหรับการส่ง (Transmit Error Counter) โดยหลังจากการรีเซ็ททางฮาร์ดแวร์จะมีค่าเริ่มต้นที่ 0 แต่ถ้าเข้าสู่สถานะ Bus Off รีจิสเตอร์นี้จะถูกปรับเป็นค่าเริ่มต้นที่ 127 ตามข้อกำหนดในโปรโตคอล ซึ่งเป็นค่านับขั้นต่ำของสัญญาณ Bus Free ที่เป็นรีเซสสีพบิตจำนวน 11 บิตที่ติดต่อกันจำนวน 128 ครั้ง สำหรับการเขียนเข้าถึงในย่าน 0-254 จะทำให้เคลียร์ Bus Off Flag และรอคอยสัญญาณ Bus Free หนึ่งครั้งหลังจากยกเลิก Reset Mode ส่วนการเขียนเข้าถึงด้วยค่า 255 จะเป็นการเริ่มต้นเข้าสู่สถานะ Bus Off โดยรีจิสเตอร์นี้จะปรากฏที่ CPU เช่นเดียวกับหน่วยความจำสำหรับการอ่านเท่านั้นใน Operating Mode และสามารถเขียนเข้าถึงได้ใน Reset Mode เท่านั้น

ตารางที่ 3.18 แสดงรายละเอียดภายในของ TX Error Counter Register

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
TXERR.7	TXERR.6	TXERR.5	TXERR.4	TXERR.3	TXERR.2	TXERR.1	TXERR.0

การเปลี่ยนแปลงค่าของรีจิสเตอร์นี้สามารถทำได้ ถ้ามีการปรับเข้าสู่ Reset Mode แล้วก่อนหน้านี้ Error Status และ Error Warning Interrupt หรือ Error Passive Interrupt สำหรับค่าที่เปลี่ยนแปลงใหม่นี้จะไม่เกิดขึ้นจนกว่า Reset Mode ถูกยกเลิกอีกครั้ง

3.6.16 Acceptance Filter Mode Register (ACF Mode)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 29 ใช้สำหรับการกำหนดโหมดการทำงานของ Acceptance Filter Banks ในแต่ละ Bank ว่าต้องการให้มีการรองรับเฟรมในรูปแบบ SFF หรือรูปแบบ EFF และให้ใช้การฟิลเตอร์แบบ Single Filter ที่ใช้ฟิลเตอร์ขนาดยาวเพียงตัวเดียว หรือแบบ Dual Filter ที่ใช้ฟิลเตอร์ขนาดสั้นจำนวน 2 ตัว การเขียนเข้าถึงรีจิสเตอร์นี้สามารถทำได้ใน Reset Mode เท่านั้น

ตารางที่ 3.19 แสดงรายละเอียดภายในของ Acceptance Filter Mode Register

BIT	NAME	VALUE	FUNCTION
ACFMOD.7	Acceptance Filter Format Bank 4 (MFORMATB4)	1 (EFF) 0 (SFF)	ACF Bank 4 ถูกใช้สำหรับข้อมูลในรูปแบบ EFF ACF Bank 4 ถูกใช้สำหรับข้อมูลในรูปแบบ SFF
ACFMOD.6	Acceptance Filter Mode Bank 4 (AMODEB4)	1 (single) 0 (dual)	Single Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 4 Dual Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 4
ACFMOD.5	Acceptance Filter Format Bank 3 (MFORMATB3)	1 (EFF) 0 (SFF)	ACF Bank 3 ถูกใช้สำหรับข้อมูลในรูปแบบ EFF ACF Bank 3 ถูกใช้สำหรับข้อมูลในรูปแบบ SFF
ACFMOD.4	Acceptance Filter Mode Bank 3 (AMODEB3)	1 (single) 0 (dual)	Single Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 3 Dual Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 3
ACFMOD.3	Acceptance Filter Format Bank 2 (MFORMATB2)	1 (EFF) 0 (SFF)	ACF Bank 2 ถูกใช้สำหรับข้อมูลในรูปแบบ EFF ACF Bank 2 ถูกใช้สำหรับข้อมูลในรูปแบบ SFF
ACFMOD.2	Acceptance Filter Mode Bank 2 (AMODEB2)	1 (single) 0 (dual)	Single Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 2 Dual Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 2
ACFMOD.1	Acceptance Filter Format Bank 1 (MFORMATB1)	1 (EFF) 0 (SFF)	ACF Bank 1 ถูกใช้สำหรับข้อมูลในรูปแบบ EFF ACF Bank 1 ถูกใช้สำหรับข้อมูลในรูปแบบ SFF
ACFMOD.0	Acceptance Filter Mode Bank 1 (AMODEB1)	1 (single) 0 (dual)	Single Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 1 Dual Filter ถูกใช้สำหรับการฟิลเตอร์ใน Bank 1

3.6.17 Acceptance Filter Enable Register (ACF Enable)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 30 ใช้สำหรับการจัดการและควบคุมการฟิลเตอร์ของ Acceptance Filter Banks ในแต่ละ Bank ว่าต้องการให้ใช้ Filter 1 และ Filter 2 หรือไม่ สำหรับโหมดการทำงานที่ใช้การฟิลเตอร์แบบ Single Filter จะใช้เพียง Filter 1 เท่านั้น นอกจากนี้ยังมีลักษณะการทำงานในฟังก์ชัน " change on the fly " โดยยอมให้เปลี่ยนแปลงค่าใน ACRn และ AMRn ได้เมื่ออยู่ใน Operating Mode ถ้าหากฟิลเตอร์ดังกล่าวถูกยกเลิก (disabled) ไปแล้วก่อนหน้านี้ ส่วนฟิลเตอร์ที่ถูกยกเลิกนั้นจะไม่ยอมให้มีการส่งผ่านข้อมูลไปยัง RXFIFO ดังนั้น ถ้าฟิลเตอร์ทั้งหมดถูกยกเลิกก็จะไม่มีข้อมูลใด ๆ สามารถส่งผ่านไปยัง RXFIFO ได้เลย

ตารางที่ 3.20 แสดงรายละเอียดภายในของ Acceptance Filter Enable Register

BIT	NAME	VALUE	FUNCTION
ACFEN.7	Bank 4 Filter 2 Enable (B4F2EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 2 ของ Bank 4 ได้ ไม่ยอมให้ใช้ Filter 2 ของ Bank 4 ได้
ACFEN.6	Bank 4 Filter 1 Enable (B4F1EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 1 ของ Bank 4 ได้ ไม่ยอมให้ใช้ Filter 1 ของ Bank 4 ได้
ACFEN.5	Bank 3 Filter 2 Enable (B3F2EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 2 ของ Bank 3 ได้ ไม่ยอมให้ใช้ Filter 2 ของ Bank 3 ได้
ACFEN.4	Bank 3 Filter 1 Enable (B3F1EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 1 ของ Bank 3 ได้ ไม่ยอมให้ใช้ Filter 1 ของ Bank 3 ได้
ACFEN.3	Bank 2 Filter 2 Enable (B2F2EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 2 ของ Bank 2 ได้ ไม่ยอมให้ใช้ Filter 2 ของ Bank 2 ได้
ACFEN.2	Bank 2 Filter 1 Enable (B2F1EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 1 ของ Bank 2 ได้ ไม่ยอมให้ใช้ Filter 1 ของ Bank 2 ได้
ACFEN.1	Bank 1 Filter 2 Enable (B1F2EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 2 ของ Bank 1 ได้ ไม่ยอมให้ใช้ Filter 2 ของ Bank 1 ได้
ACFEN.0	Bank 1 Filter 1 Enable (B1F1EN)	1 (enabled) 0 (disabled)	ยอมให้ใช้ Filter 1 ของ Bank 1 ได้ ไม่ยอมให้ใช้ Filter 1 ของ Bank 1 ได้

3.6.18 Acceptance Filter Priority Register (ACF Priority)

รีจิสเตอร์นี้อยู่ในแอดเดรสที่ 31 ใช้สำหรับการกำหนดระดับความสำคัญของการฟิลเตอร์ เพื่อควบคุมการเกิด Receive Interrupt ว่าต้องการให้เกิดอินเทอร์รัพท์ขึ้นทันทีถ้ามีข้อมูลผ่านการฟิลเตอร์ที่กำหนดไว้ หรือจะกำหนดให้ใช้ Receive Interrupt Level สำหรับควบคุมการเกิดอินเทอร์รัพท์แทน

ตารางที่ 3.21 แสดงรายละเอียดภายในของ Acceptance Filter Priority

BIT	NAME	VALUE	FUNCTION
ACFPRIO.7	Bank 4 Filter 2 Priority (B4F2PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 2 ของ Bank 4 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.6	Bank 4 Filter 1 Priority (B4F1PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 1 ของ Bank 4 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.5	Bank 3 Filter 2 Priority (B3F2PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 2 ของ Bank 3 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.4	Bank 3 Filter 1 Priority (B3F1PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 1 ของ Bank 3 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.3	Bank 2 Filter 2 Priority (B2F2PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 2 ของ Bank 2 Receive Interrupt เกิดขึ้นทันทีถ้า ะดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.2	Bank 2 Filter 1 Priority (B2F1PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 1 ของ Bank 2 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.1	Bank 1 Filter 2 Priority (B1F2PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 2 ของ Bank 1 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register
ACFPRIO.0	Bank 1 Filter 1 Priority (B1F1PRIO)	1 (high) 0 (low)	Receive Interrupt เกิดขึ้นทันทีถ้ามีข้อมูลผ่าน Filter 1 ของ Bank 1 Receive Interrupt เกิดขึ้นทันทีถ้าระดับของ FIFO เกินกว่าขอบเขตที่กำหนดไว้ใน Receive Interrupt Level Register

3.6.19 Transmit Buffer

บัฟเฟอร์นี้อยู่ในแอดเดรสที่ 112-124 ใช้สำหรับการพักข้อมูลที่ต้องการส่งจำนวนหนึ่งเฟรม โครงสร้างของบัฟเฟอร์จะแบ่งออกเป็นแผนผัง สำหรับรูปแบบ SFF และรูปแบบ EFF ภายในแผนผังจะประกอบด้วยส่วนของ Descriptor Field และ Data Field โดยที่ไบต์แรกของ Descriptor Field จะเป็น Frame Information ที่บอกให้ทราบเกี่ยวกับข้อมูลที่ส่งว่าเป็นรูปแบบ SFF หรือรูปแบบ EFF, Data Frame หรือ Remote Frame และ Data Length Code ส่วนที่เหลือเป็นไบต์หมายเลข ID สำหรับรูปแบบ SFF จะมีขนาด 2 ไบต์ และรูปแบบ EFF จะมีขนาด 4 ไบต์ ส่วนของ Data Field จะประกอบด้วยไบต์ข้อมูลสูงสุด 8 ไบต์ ด้วยเหตุนี้ Transmit Buffer จึงต้องมี

ขนาดความยาวเท่ากับ 13 ไบต์ เพื่อให้สามารถรองรับข้อมูลที่ต้องการส่งได้ทั้งในรูปแบบ SFF และรูปแบบ EFF

Standard Frame Format (SFF)

CAN Addr.		
	112	TX Frame information
	113	TX Identifier 1
	114	TX Identifier 2
	115	TX Data byte 1
	116	TX Data byte 2
	117	TX Data byte 3
	118	TX Data byte 4
	119	TX Data byte 5
	120	TX Data byte 6
	121	TX Data byte 7
	122	TX Data byte 8
	123	unused
	124	unused

Extended Frame Format (EFF)

CAN Addr.		
	112	TX Frame information
	113	TX Identifier 1
	114	TX Identifier 2
	115	TX Identifier 3
	116	TX Identifier 4
	117	TX Data byte 1
	118	TX Data byte 2
	119	TX Data byte 3
	120	TX Data byte 4
	121	TX Data byte 5
	122	TX Data byte 6
	123	TX Data byte 7
	124	TX Data byte 8

ภาพที่ 3.6 แสดงแผนผังของ Transmit Buffer

Standard Frame Format (SFF)

Addr.112	TX Frame information						
7	6	5	4	3	2	1	0
FF	RTR	(0)	(0)	DLC.3	DLC.2	DLC.1	DLC.0

Addr.113	TX Identifier 1						
7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Addr.114	TX Identifier 2						
7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Extended Frame Format (EFF)

Addr.112	TX Frame information						
7	6	5	4	3	2	1	0
FF	RTR	(0)	(0)	DLC.3	DLC.2	DLC.1	DLC.0

Addr.113	TX Identifier 1						
7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Addr.114	TX Identifier 2						
7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Addr.115	TX Iden ifier 3						
7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Addr.116	TX Identifier 4						
7	6	5	4	3	2	1	0
ID.4	ID.3	ID.2	ID.1	ID.0	(RTR)	(0)	(0)

Meaning of the Transmit Buffer Bits:

ID.x	Identifier bit x
FF	Frame Format
RTR	Remote Transmission Request
DLC.x	Data Length Code bit x
X	don't care
(0)	don't care, but recommended to becompatible to Receive Buffer

ภาพที่ 3.7 แสดงรายละเอียดในส่วน Descriptor Field ของ Transmit Buffer

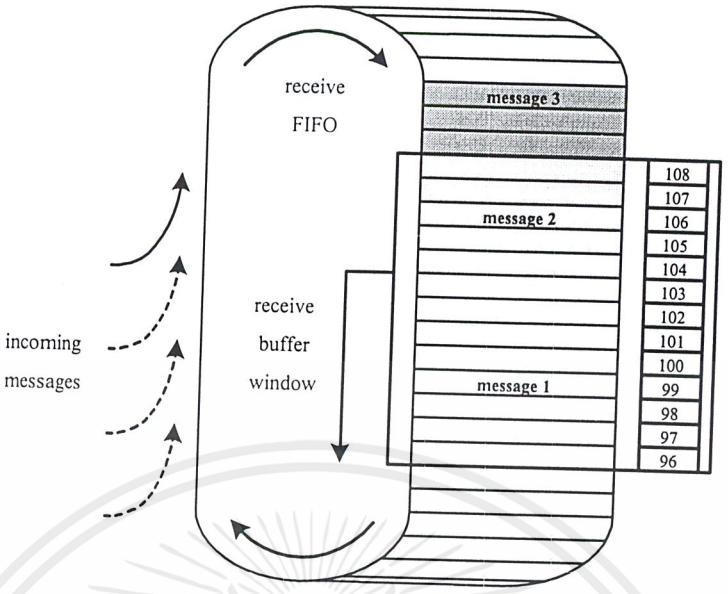
ตารางที่ 3.22 แสดงรายละเอียดของบิต FF และบิต RTR

BIT	VALUE	FUNCTION
FF	1 (EFF)	CAN คอนโทรลเลอร์จะส่งข้อมูลในรูปแบบ EFF
	0 (SFF)	CAN คอนโทรลเลอร์จะส่งข้อมูลในรูปแบบ FF
RTR	1 (remote)	CAN คอนโทรลเลอร์จะส่งข้อมูลเป็น Remote Frame
	0 (data)	CAN คอนโทรลเลอร์จะส่งข้อมูลเป็น Data Frame

เนื่องจากจำนวนไบต์ข้อมูลใน Data Field จะถูกเข้ารหัสโดย Data Length Code (DLC) ดังนั้นในการเริ่มต้นการส่งของ Remote Frame ส่วนของ DLC นี้จะไม่ถูกนำมาตัดสินใจ เพราะว่าบิต RTR ที่เป็น '1' จะปรับให้จำนวนไบต์ข้อมูลที่ส่งและรับมีค่าเท่ากับ '0' อย่างไรก็ตาม DLC ต้องมีการกำหนดให้ถูกต้องเพื่อหลีกเลี่ยงความผิดพลาดบนบัส ถ้ามีจำนวน 2 โหนดเริ่มต้นการส่ง Remote Frame พร้อมกันด้วยหมายเลข ID เดียวกัน สำหรับค่าของ DLC ที่มากกว่า 8 จะไม่ถูกนำมาใช้ ดังนั้นถึงแม้จะกำหนดค่าให้มากกว่า 8 ก็ตาม แต่ภายใน Data Frame จะทำการส่งเพียง 8 ไบต์เท่านั้น ในรูปแบบ SFF จะประกอบด้วยหมายเลข ID ขนาด 11 บิต (ID.28-ID.18) และในรูปแบบ EFF จะประกอบด้วยหมายเลข ID ขนาด 29 บิต (ID.28-ID.0) โดยบิตที่มีความสำคัญมากที่สุดคือ ID.28 ซึ่งจะถูกส่งไปที่บัสก่อนในช่วงการตัดสินเข้าใช้บัส ค่าหมายเลข ID เป็นส่วนที่แสดงถึงชื่อของข้อมูล รวมถึงถูกใช้ในโหนดผู้รับสำหรับการฟิลเตอร์ข้อมูล และใช้ในการกำหนดระดับความสำคัญในการเข้าใช้บัสอีกด้วย ซึ่งค่าหมายเลข ID ที่เป็นไบนารีต่ำจะมีระดับความสำคัญสูงเนื่องจากว่าจะมีโดมิแนนท์บิตจำนวนมากอยู่ในส่วนแรกของหมายเลข ID

3.6.20 Receive Buffer

บัฟเฟอร์สำหรับการรับนี้อยู่ในแอดเดรสที่ 96-108 ถูกใช้ในการแสดงข้อมูลที่รับเข้ามาจำนวน 1 เฟรมใน RXFIFO ให้กับ CPU สามารถเข้าถึงข้อมูลดังกล่าวได้ โครงสร้างของบัฟเฟอร์นี้จะคล้ายกับโครงสร้างของ Transmit Buffer ที่แบ่งออกเป็นแผนผังสำหรับรูปแบบ SFF และรูปแบบ EFF ภายในแผนผังจะประกอบด้วยส่วนของ Descriptor Field และ Data Field โดยที่ไบต์แรกของ Descriptor Field จะเป็น Frame Information ที่บอกให้ทราบเกี่ยวกับข้อมูลที่รับว่าเป็นรูปแบบ SFF หรือรูปแบบ EFF, Data Frame หรือ Remote Frame และ Data Length Code ส่วนที่เหลือเป็นไบต์หมายเลข ID สำหรับรูปแบบ SFF จะมีขนาด 2 ไบต์ และรูปแบบ EFF จะมีขนาด 4 ไบต์ ส่วนของ Data Field จะประกอบด้วยไบต์ข้อมูลสูงสุด 8 ไบต์ ดังนั้นจึงต้องมีขนาดความยาวเท่ากับ 13 ไบต์เช่นเดียวกัน เพื่อให้สามารถรองรับข้อมูลที่รับเข้ามาได้ทั้งในรูปแบบ SFF และรูปแบบ EFF



ภาพที่ 3.8 แสดงแผนผังของ Receive Buffer

Standard Frame Format (SFF)

Addr.112	RX Frame information						
7	6	5	4	3	2	1	0
FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0

Addr.113	RX Identifier 1						
7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Addr.114	RX Identifier 2						
7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	RTR	0	0	0	0

Meaning of the Receive Buffer Bits:

ID.x	Identifier bit x
FF	Frame Format
RTR	Remote Transmission Request
DLC.x	Data Length Code bit x

Extended Frame Format (EFF)

Addr.112	RX Frame information						
7	6	5	4	3	2	1	0
FF	RTR	(0)	(0)	DLC.3	DLC.2	DLC.1	DLC.0

Addr.113	RX Identifier 1						
7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Addr.114	RX Identifier 2						
7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Addr.115	RX Identifier 3						
7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Addr.116	RX Identifier 4						
7	6	5	4	3	2	1	0
ID.4	ID.3	ID.2	ID.1	ID.0	(RTR)	(0)	(0)

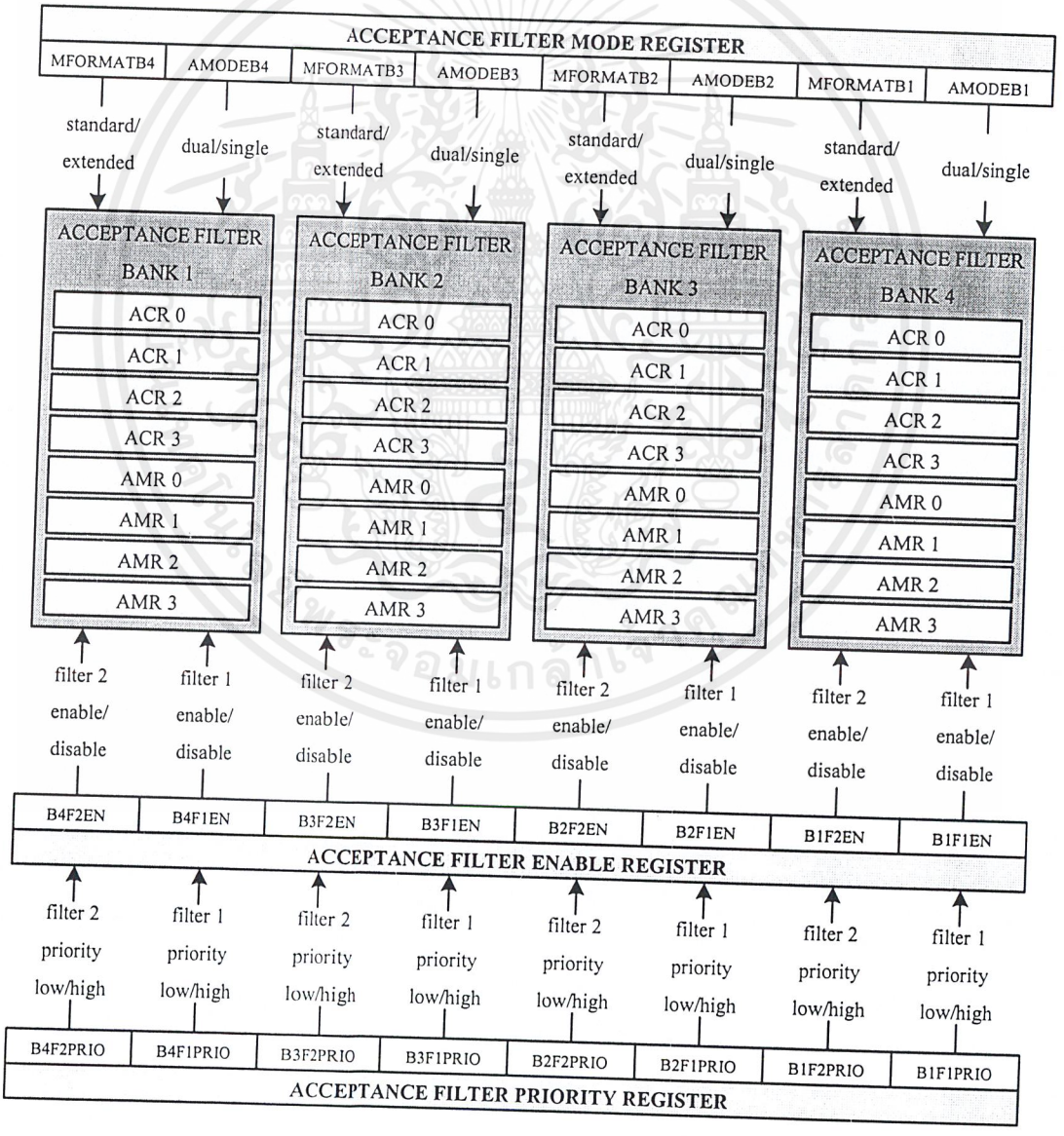
ภาพที่ 3.9 แสดงรายละเอียดในส่วน Descriptor Field ของ Receive Buffer

Receive Buffer สามารถบรรจุข้อมูลได้เพียง 1 เฟรมเท่านั้น จากภาพที่ 3.6 พบว่ามีข้อมูลอยู่ 2 เฟรมภายใน RXB โดยข้อมูลที่ 1 จะเป็นข้อมูลที่สามารถใช้งานได้ในบัฟเฟอร์นี้ ส่วนข้อมูลที่ 2 จะไม่ถูกอ่านจนกว่ามีการเลื่อนถึงแอดเดรสที่ 96 ด้วยคำสั่ง Release Receive Buffer

3.7 โครงสร้างของ Acceptance Filter

การทำงานของฟิลเตอร์นี้ช่วยให้ CAN คอนโทรลเลอร์ สามารถส่งผ่านข้อมูลที่รับเข้ามาไปยัง RXFIFO ได้ เมื่อค่าหมายเลข ID และชนิดของเฟรมตรงกับค่าที่กำหนดไว้ก่อนแล้วเท่านั้นใน Acceptance Filter Register กล่าวคือ ถ้ามีอย่างน้อยที่สุดหนึ่งฟิลเตอร์ที่ตรงกันก็จะส่งผลให้ข้อมูลถูกคัดลอกไปยัง RXFIFO ได้

รูปแบบของ Acceptance Filter ถูกกำหนดโดย Acceptance Code Register (ACRn) และ Acceptance Mask Register (AMRn) ซึ่งภายใน ACRn เป็นการกำหนดรูปแบบบิตของข้อมูลที่ต้องการรับเข้ามา และภายใน AMRn เป็นการกำหนดตำแหน่งบิตที่ต้องการให้ " don't care " เพื่อช่วยในการจัดแบ่งกลุ่มของข้อมูล



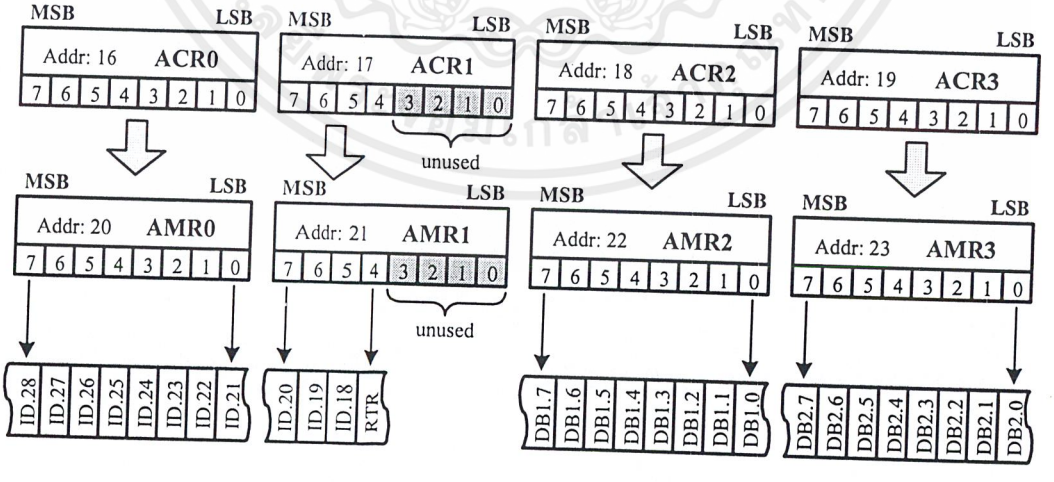
ภาพที่ 3.10 แสดงรูปแบบการกำหนด Acceptance Filter Banks

PeliCAN ถูกออกแบบให้สนับสนุนกับ Acceptance Filter Banks จำนวน 4 Bank โดยในแต่ละ Bank จะมีฟังก์ชันการทำงานเหมือนกับ SJA1000 แต่มีการเพิ่มเติมการทำงานในฟังก์ชัน " change on the fly " ที่สามารถเปลี่ยนแปลงค่าฟิลเตอร์ได้ใน Operating Mode โดยไม่ต้องยกเลิกการทำงานในขณะนั้นเพื่อเข้าสู่ Reset Mode เพื่อตอบสนองการทำงานของ CAN คอนโทรลเลอร์ให้มีประสิทธิภาพมากขึ้น สำหรับโครงสร้างการฟิลเตอร์ข้อมูลสามารถกำหนดได้ 2 แบบ ได้แก่ Single Filter และ Dual Filter โดยแต่ละแบบจะขึ้นอยู่กับรูปแบบของเฟรมที่รับอีกด้วย ด้วยเหตุนี้จึงทำให้ในแต่ละโครงสร้างมีการกำหนดรูปแบบบิตที่ใช้ในการฟิลเตอร์แตกต่างกัน

3.7.1 โครงสร้างการฟิลเตอร์แบบ Single Filter

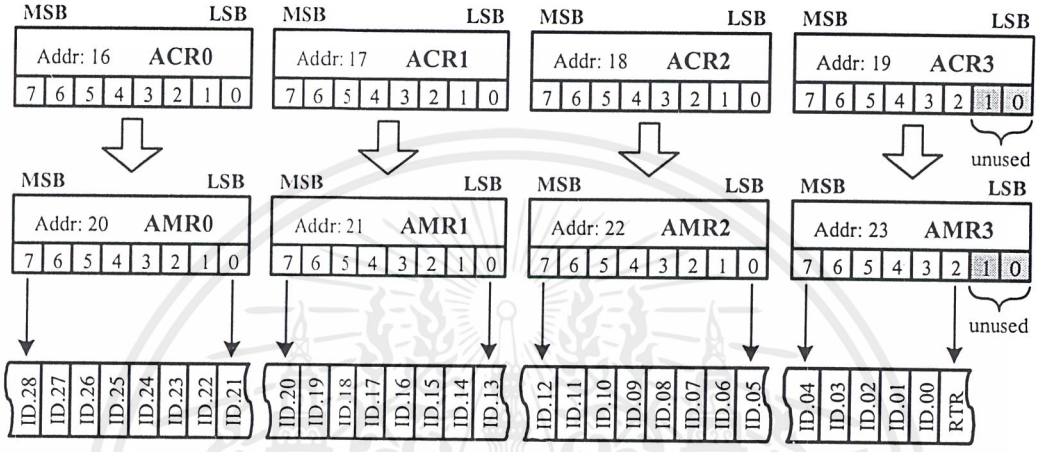
โครงสร้างการฟิลเตอร์แบบนี้จะใช้ฟิลเตอร์ขนาดยาวเพียงตัวเดียว ด้วยความยาวขนาด 4 ไบต์ โดยตำแหน่งบิตที่ตรงกันระหว่างไบต์ของฟิลเตอร์และไบต์ของข้อมูลจะขึ้นอยู่กับรูปแบบเฟรมที่กำหนดด้วย แบ่งออกเป็น 2 ชนิด คือ

1. Single Filter Standard Frame ในโครงสร้างนี้จะใช้ส่วนของ Base ID, บิต RTR และไบต์ข้อมูลขนาด 2 ไบต์แรกสำหรับการฟิลเตอร์ข้อมูล ในบางครั้งก็อาจยอมรับข้อมูลได้ถ้าไม่มีไบต์ข้อมูลเลย เนื่องจากการเซ็ทที่บิต RTR ให้เป็น Remote Frame หรือมีเพียงไบต์เดียวเนื่องจาก Data Length Code ที่กำหนดไว้ การรับรองข้อมูลจะเป็นผลสำเร็จได้ถ้าทุกบิตที่ทำการเปรียบเทียบมีการส่งสัญญาณยอมรับ ในตำแหน่งบิตทาง LSB จำนวน 4 บิต ของ ACR1 และ AMR1 จะไม่ถูกนำมาใช้และควรโปรแกรมให้เป็น " don't care " ด้วยการเซ็ทบิต AMR1.3, AMR1.2, AMR1.1 และ AMR1.0 ให้เป็น '1'



ภาพที่ 3.11 แสดงโครงสร้างของ Single Filter Standard Frame

2. Single Filter Extended Frame ในโครงสร้างนี้จะใช้ส่วนของ Base ID, Extended ID และบิต RTR สำหรับการฟิลเตอร์ข้อมูล การรับรองข้อมูลจะเป็นผลสำเร็จได้ถ้าทุกบิตที่ทำการเปรียบเทียบมีการส่งสัญญาณยอมรับ ในตำแหน่งบิตทาง LSB จำนวน 2 บิต ของ ACR3 และ AMR3 จะไม่ถูกนำมาใช้และควรโปรแกรมให้เป็น " don't care " ด้วยการเซ็ทบิต AMR3.1 และ AMR3.0 ให้เป็น '1'



ภาพที่ 3.12 แสดงโครงสร้างของ Single Filter Extended Frame

ตารางที่ 3.23 แสดงตัวอย่างการฟิลเตอร์แบบ Single Filter Standard Frame

n	0	1 (upper 4 bits)	2	3
ACRn	01XX X010	XXXX	XXXX XXXX	XXXX XXXX
AMRn	0011 1000	1111	1111 1111	1111 1111
Accepted message (ID.28-ID.18, RTR)	01xx x010	xxxx		

(" X " = irrelevant, " x " = don't care)

ตารางที่ 3.24 แสดงตัวอย่างการฟิลเตอร์แบบ Single Filter Extended Frame

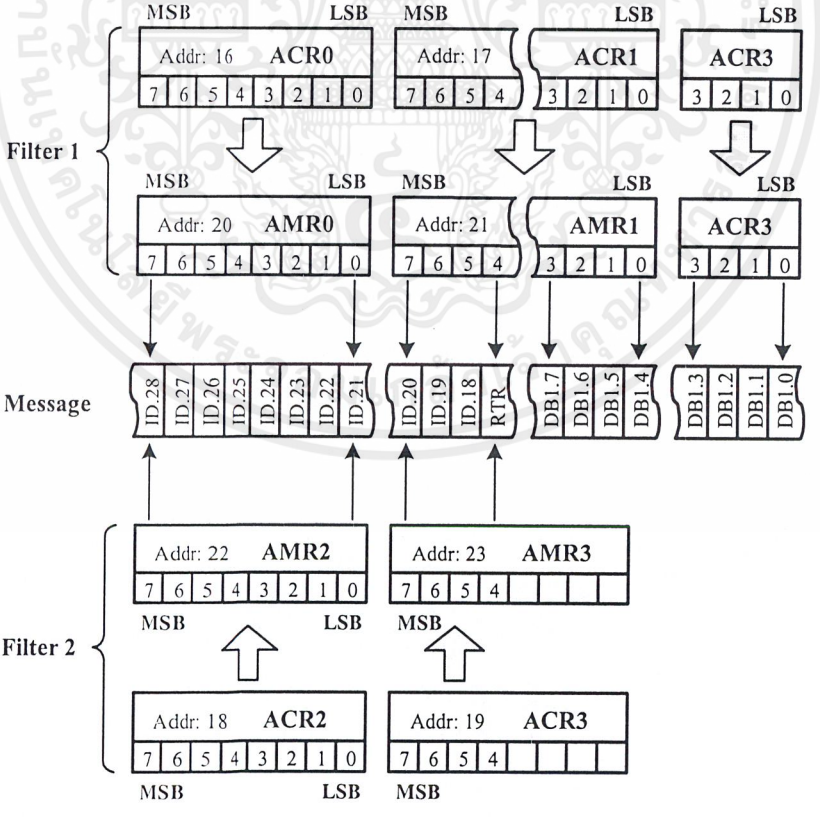
n	0	1	2	3 (upper 6 bits)
ACRn	1011 0100	1011 000X	1100 XXXX	0011 0XXX
AMRn	0000 0000	0000 0001	0000 1111	0000 0111
Accepted message (ID.28-ID.00, RTR)	1011 0100	1011 000x	1100 xxxx	0011 0x

(" X " = irrelevant, " x " = don't care)

3.7.2 โครงสร้างการฟิลเตอร์แบบ Dual Filter

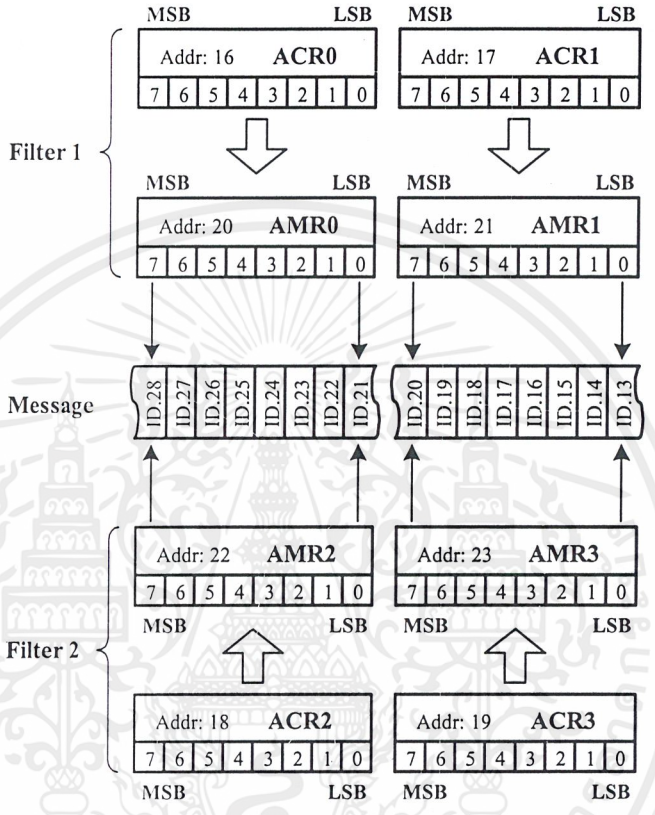
โครงสร้างการฟิลเตอร์แบบนี้ จะใช้ฟิลเตอร์ขนาดสั้นจำนวน 2 ตัว คือ Filter1 และ Filter2 เพื่อตัดสินใจว่าข้อมูลนี้ควรถูกคัดลอกไปที่ RXFIFO หรือไม่ ข้อมูลอาจถูกยอมรับได้ถ้ามีฟิลเตอร์เพียงตัวเดียวที่ส่งสัญญาณยอมรับ โดยตำแหน่งบิตที่ตรงกันระหว่างไบต์ของฟิลเตอร์และไบต์ของข้อมูล จะขึ้นอยู่กับรูปแบบเฟรมที่กำหนดด้วย แบ่งออกเป็น 2 ชนิด คือ

1. Dual Filter Standard Frame ในโครงสร้างนี้จะมีการกำหนดฟิลเตอร์ทั้งสองแตกต่างกัน โดยใน Filter1 จะใช้ส่วนของ Base ID, บิต RTR และไบต์ข้อมูลแรก สำหรับการฟิลเตอร์ข้อมูล ส่วนใน Filter2 จะใช้ส่วนของ Base ID และบิต RTR สำหรับการฟิลเตอร์ข้อมูล การรองรับข้อมูลจะเป็นผลสำเร็จได้ถ้าทุกบิตที่ทำการเปรียบเทียบของฟิลเตอร์เพียงตัวเดียวมีการส่งสัญญาณยอมรับ ในกรณีที่ไม่มีไบต์ข้อมูลเลยเนื่องจากการเซ็ทที่บิต RTR ให้เป็น Remote Frame หรือใน Data Length Code มีค่าเท่ากับ 0 ข้อมูลก็อาจผ่าน Filter1 ไปได้ ถ้าในส่วนแรกจนถึงบิต RTR มีการส่งสัญญาณยอมรับ และควรโปรแกรมให้ตำแหน่งบิตทาง LSB จำนวน 4 บิตของ AMR1 และ AMR3 ถูกเซ็ทเป็น '1' ดังนั้นจึงทำให้ฟิลเตอร์ทั้งสองทำงานในส่วนของ Base ID และบิต RTR เหมือนกัน



ภาพที่ 3.13 แสดงโครงสร้างของ Dual Filter Standard Frame

2. Dual Filter Extended Frame ในโครงสร้างนี้จะมีการกำหนดฟิลเตอร์ทั้งสองเหมือนกัน โดยใช้ส่วนของหมายเลข ID ในรูปแบบ EFF จำนวน 2 ไบต์แรก สำหรับการฟิลเตอร์ข้อมูล การรองรับข้อมูลจะเป็นผลสำเร็จได้ถ้าทุกบิตที่ทำการเปรียบเทียบของฟิลเตอร์เพียงตัวเดียว มีการส่งสัญญาณยอมรับ



ภาพที่ 3.14 แสดงโครงสร้างของ Dual Filter Extended Frame

ตารางที่ 3.25 แสดงตัวอย่างการฟิลเตอร์แบบ Dual Filter Standard Frame

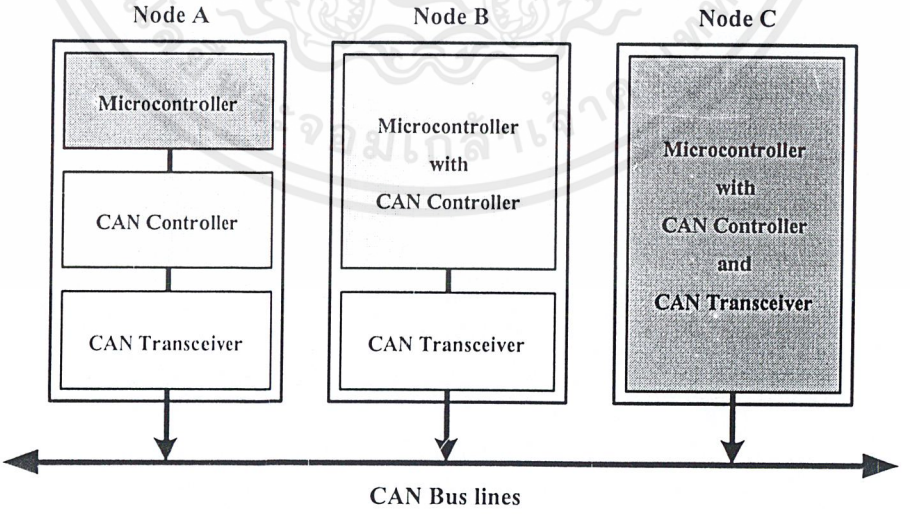
	Filter 1						Filter 2			
n	**0**		**1**		**3**	**lower 4 bits**	**2**		**3**	**upper 4 bits)**
ACRn	1110	1011	0010	1111	...	1001	1111	0100	XXX0	...
AMRn	0000	0000	0000	0000	...	0000	0000	0000	1110	...
Accepted message	1110	1011	0010	1111	...	1001	1111	0100	xxx0	
	ID + RTR		| first data byte				ID		| RTR	

(" X " = irrelevant, " x " = don't care)

บทที่ 4
ชุดทดลองสำหรับระบบ CAN

4.1 บทกล่าวนำ

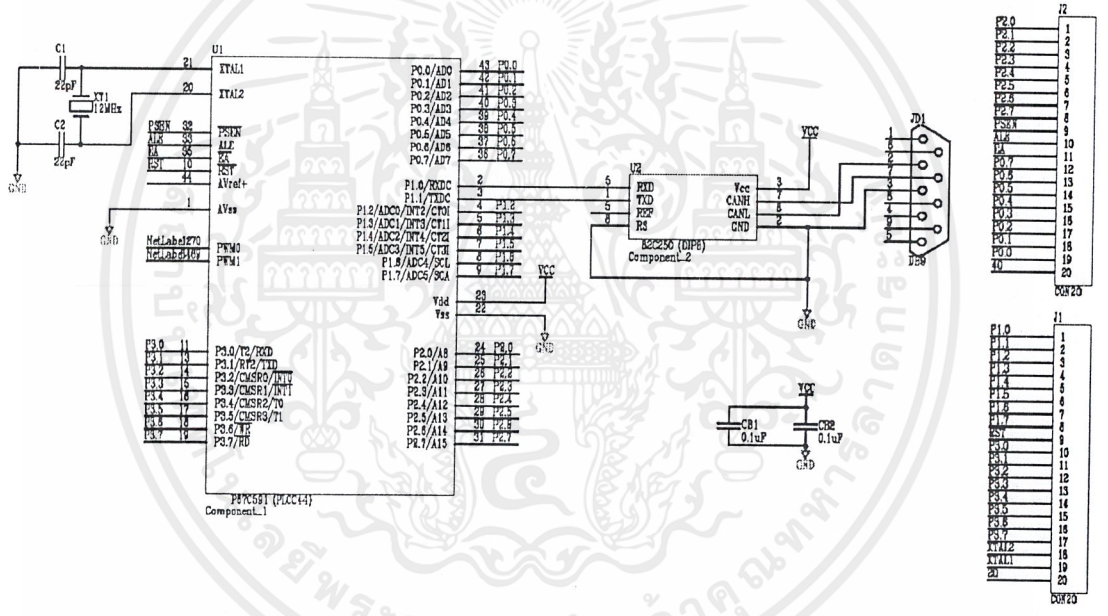
ในแต่ละโหนดที่ต่อร่วมอยู่บนบัสในระบบ CAN จำเป็นจะต้องมีอุปกรณ์ที่ช่วยจัดการและควบคุมการทำงานในด้านโปรโตคอลของ CAN ให้ตรงตามมาตรฐานและข้อกำหนด ซึ่งจะต้องประกอบด้วยอุปกรณ์ควบคุมจำนวน 3 ส่วน ได้แก่ CAN ทรานซีฟเวอร์ ที่สนับสนุนโครงสร้างการทำงานในชั้นกายภาพ (Physical Layer) CAN คอนโทรลเลอร์ ที่สนับสนุนโครงสร้างการทำงานในชั้นเชื่อมโยงข้อมูล (Data Link Layer) และไมโครคอนโทรลเลอร์ ที่สนับสนุนโครงสร้างการทำงานในชั้นประยุกต์ใช้งาน (Application Layer) นอกจากนี้ในแต่ละโหนดนั้นมีลักษณะของโครงสร้างการต่อใช้งานที่แตกต่างกันขึ้นอยู่กับรูปแบบของ CAN คอนโทรลเลอร์ ที่บริษัทผู้ผลิตจะออกแบบ ซึ่งอาจเป็นแบบชิปเดี่ยวที่มีเพียง CAN คอนโทรลเลอร์ ที่ต้องการความสะดวกในการใช้งานบนบอร์ด หรือเป็นไอซีที่รวม CAN คอนโทรลเลอร์ และไมโครคอนโทรลเลอร์ไว้ด้วยกันในชิปเดียว ที่ต้องการความเร็วในติดต่อระหว่าง CAN คอนโทรลเลอร์ กับ CPU มากขึ้น และช่วยให้ง่ายต่อการโปรแกรมและยังลดพื้นที่ใช้งานบนบอร์ดได้อีกด้วย หรือจะเป็นไอซีที่รวมทั้งสามส่วนไว้ด้วยกัน ซึ่งเป็นไอซีที่ออกแบบในยุคแรกสำหรับใช้งานในยานยนต์และมีปัญหาหลักในด้านของการรวมเทคโนโลยีที่แตกต่างกันไว้ในชิปเดียว มีรายละเอียดดังภาพที่ 4.1



ภาพที่ 4.1 แสดงโครงสร้างการต่อใช้งานของ CAN

4.2 การออกแบบสร้างชุดทดลอง

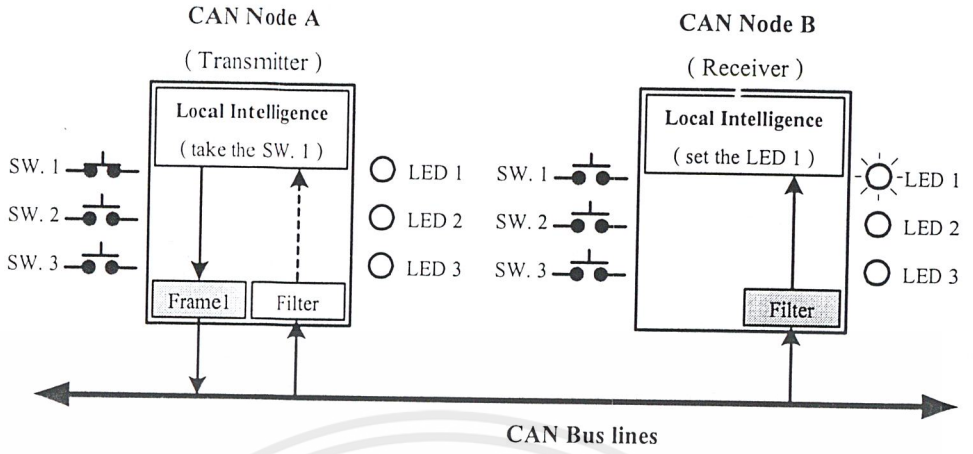
ในปัจจุบันมีบริษัทผู้ผลิตหลายรายผลิตไอซี CAN คอนโทรลเลอร์ ออกมาจำหน่ายมากมาย และพัฒนาให้สามารถจัดการในเรื่องโปรโตคอลได้อย่างมีประสิทธิภาพมากขึ้น รวมถึงเพิ่มฟังก์ชันการทำงานให้มีความยืดหยุ่นและสะดวกในการโปรแกรมอีกด้วย สำหรับชุดทดลองที่ออกแบบนี้จะใช้ไอซีเบอร์ P87C591 ที่มี CAN คอนโทรลเลอร์ เวอร์ชัน CAN 2.0B และไมโครคอนโทรลเลอร์ขนาด 8 บิตรวมอยู่ในชิปเดียวกัน และไอซีเบอร์ PCA82C250 ที่เป็น CAN ทรานซีฟเวอร์ ซึ่งทั้งสองเบอร์จะเป็นของ Philips Semiconductors โดยชุดทดลองนี้ถูกออกแบบให้มีลักษณะเป็นแบบโมดูลของ CAN ที่มีความยืดหยุ่นในการใช้งาน สามารถนำไปต่อลงบนบอร์ด MCS-51 ที่มีวางจำหน่ายทั่วไปได้ง่าย เพื่อเพิ่มความสะดวกในการเรียนรู้และการพัฒนา วงจรเบื้องต้นจะมีลักษณะดังภาพที่ 4.2



ภาพที่ 4.2 แสดงวงจรเบื้องต้นของชุดทดลอง

4.3 การทดลองการสื่อสารข้อมูลของ CAN

การทดลองนี้เป็นการทดสอบฟังก์ชันการทำงานของ CAN ที่เกี่ยวกับการรับและส่งข้อมูลบนบัสระหว่างโหนด A และโหนด B เพื่อพิสูจน์ว่าทั้งสองโหนดสามารถสื่อสารข้อมูลถึงกันได้ รวมถึงทดสอบว่ามีการทำงานเป็นไปตามข้อกำหนดในโปรโตคอลหรือไม่ โดยในที่นี้จะทดสอบด้วยการกดสวิทช์ที่โหนด A แล้วให้หลอดไฟติดที่โหนด B หรือกดสวิทช์ที่โหนด B แล้วให้หลอดไฟติดที่โหนด A ถ้าผลลัพธ์ที่ได้นั้นตรงตามที่กำหนดไว้แสดงว่าทั้งสองโหนดสามารถสื่อสารข้อมูลถึงกันได้จริง ดังภาพที่ 4.3



ภาพที่ 4.3 แสดงการทดสอบการสื่อสารข้อมูลระหว่างโหนด A และ B

ในการทดลองนี้มีการกำหนดรูปแบบการทำงานของโหนด A และโหนด B โดยรวมถึงผลลัพธ์จากการทดลองที่ควรจะเป็น คือ

1. กด SW.1 ของโหนด A จะทำให้ LED 1 ของโหนด B ติด
2. กด SW.2 ของโหนด A จะทำให้ LED 2 ของโหนด B ติด
3. กด SW.3 ของโหนด A จะทำให้ LED 3 ของโหนด B ติด
4. กด SW.1 ของโหนด B จะทำให้ LED 1 ของโหนด A ติด
5. กด SW.2 ของโหนด B จะทำให้ LED 2 ของโหนด A ติด
6. กด SW.3 ของโหนด B จะทำให้ LED 3 ของโหนด A ติด

ในการทดลองนี้มีการกำหนดรูปแบบการส่งข้อมูลของโหนด A และโหนด B โดยรวมถึงรายละเอียดของเฟรมสื่อสารและการฟิลเตอร์ข้อมูล คือ

1. เมื่อทำการกด SW.1 ของโหนด A จะเป็นการส่ง Data Frame ในรูปแบบ SFF ที่มีค่าหมายเลข ID เป็นแบบ A

2. เมื่อทำการกด SW.2 ของโหนด A จะเป็นการส่ง Data Frame ในรูปแบบ SFF ที่มีค่าหมายเลข ID เป็นแบบ B

3. เมื่อทำการกด SW.3 ของโหนด A จะเป็นการส่ง Data Frame ในรูปแบบ EFF ที่มีค่าหมายเลข ID เป็นแบบ C

4. เมื่อทำการกด SW.1 ของโหนด B จะเป็นการส่ง Remote Frame ในรูปแบบ SFF ที่มีค่าหมายเลข ID เป็นแบบ A

5. เมื่อทำการกด SW.2 ของโหนด B จะเป็นการส่ง Remote Frame ในรูปแบบ SFF ที่มีค่าหมายเลข ID เป็นแบบ B

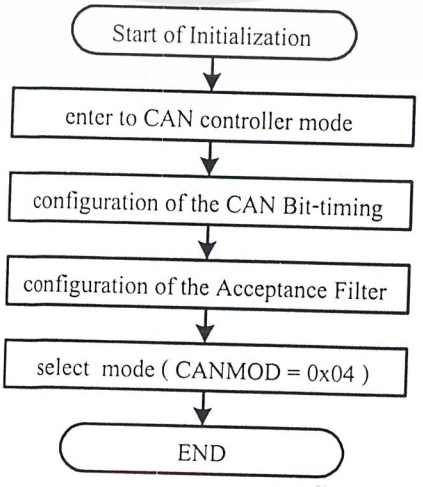
6. เมื่อทำการกด SW.3 ของโหนด B จะเป็นการส่ง Remote Frame ในรูปแบบ EFF ที่มีค่าหมายเลข ID เป็นแบบ C

ตารางที่ 4.1 แสดงรายละเอียดของเฟรมสื่อสารและโครงสร้างการฟิลเตอร์ข้อมูล

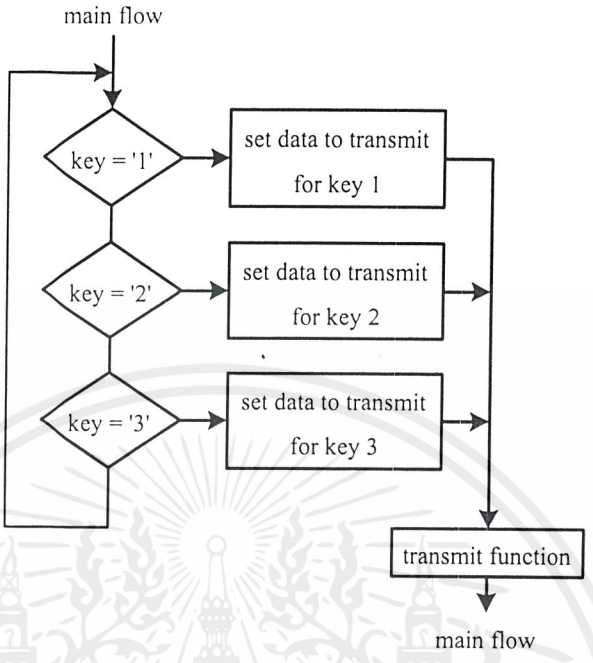
รูปแบบเฟรม	ค่าหมายเลข ID	ข้อมูลใน Data Field	โครงสร้างการฟิลเตอร์
SFF	แบบ A = 228 (0001.1100.100)	Byte 1 = 01h	ใช้ Filter 1 ใน Bank 1 แบบ Single Filter Standard Frame
SFF	แบบ B = 229 (0001.1100.101)	Byte 1 = 02h	ใช้ Filter 1 ใน Bank 1 แบบ Single Filter Standard Frame
EFF	แบบ C = 60,031,000 (0001.1100.1010.0000. 0000.0000.1100.0)	Byte 1 = 03h	ใช้ Filter 1 ใน Bank 2 แบบ Single Filter Extended Frame (Based ID จะกำหนดเหมือนกัน)

จากตารางที่ 4.1 ข้อมูลที่มีหมายเลข ID แบบ A จะมีค่าไบนารีต่ำที่สุด ดังนั้นจึงมีระดับความสำคัญของข้อมูลสูงที่สุด รองลงมาคือแบบ B และ C ตามลำดับ ข้อมูลที่ส่งนั้นจะใช้เพียงไบต์แรกใน Data Field เท่านั้น เพื่อความสะดวกในการทดสอบและการโปรแกรม ส่วนขั้นตอนการโปรแกรมสำหรับการทดลองนี้จะเป็นไปตามแผนผังการทำงาน ซึ่งจะแบ่งออกเป็นฟังก์ชันย่อยที่มีลักษณะดังนี้

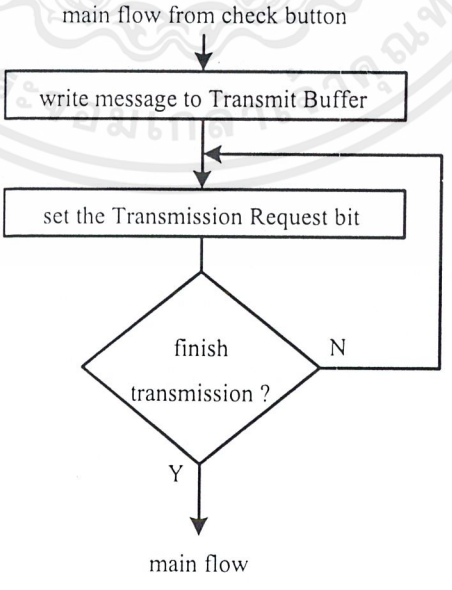
Initialization Function :



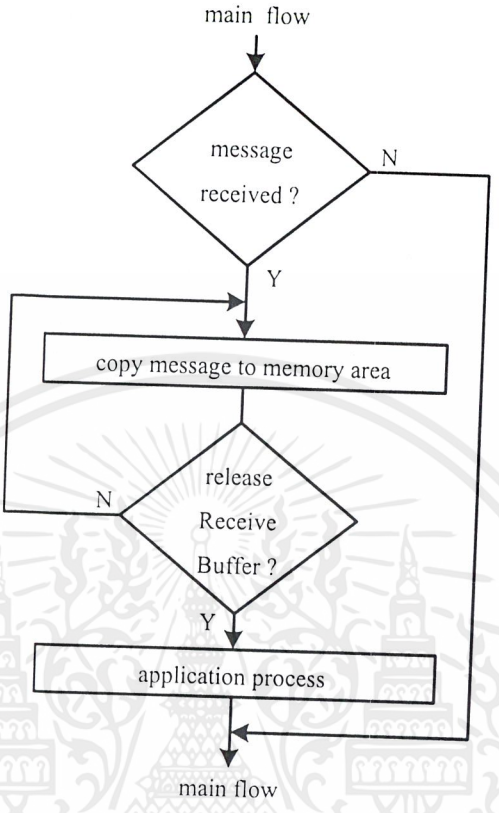
Check Button Function :



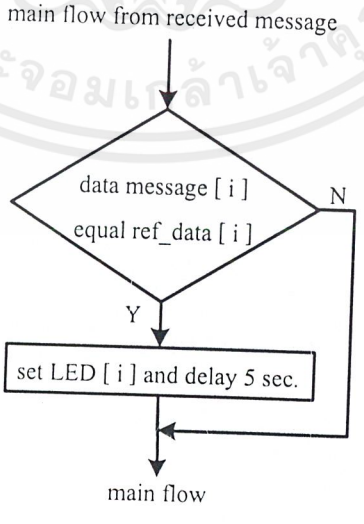
Transmit Service Function :



Receive Service Function :



Application Process Function :



จากแผนผังการทำงานเป็นขั้นตอนการทำงานภายใน CAN คอนโทรลเลอร์ ที่เกี่ยวกับการกำหนดการทำงานเริ่มแรกของ CAN การเตรียมข้อมูลสำหรับการส่ง การเตรียมการฟิลเตอร์และรองรับข้อมูล และการประมวลผลข้อมูลที่ได้รับ เพื่อตัดสินใจทำกระบวนการที่กำหนดไว้ โดยมีรายละเอียดของการทำงานดังนี้ คือ

1. ในการทำงานเริ่มแรกหลังจากการ Power-up หรือการรีเซ็ททางฮาร์ดแวร์ โดยจะต้องโปรแกรมให้ CAN คอนโทรลเลอร์ เข้าสู่ Reset Mode ด้วยการเซ็ทที่บิต RM ภายใน Mode Register ซึ่งจะเป็นการกำหนดโครงสร้างการทำงานต่าง ๆ ให้กับ CAN คอนโทรลเลอร์ ได้แก่ การกำหนดสภาวะของขา TXDC การกำหนดอินเทอร์รัพท์ การกำหนด CAN Bit Timming และการกำหนด Acceptance Filter หลังจากนั้นจะต้องโปรแกรมให้ CAN คอนโทรลเลอร์ เข้าสู่ Operating Mode ด้วยการเคลียร์ที่บิต RM

2. เมื่อทำการกำหนดโครงสร้างการทำงานต่าง ๆ ให้กับ CAN คอนโทรลเลอร์ เสร็จแล้ว หลังจากนั้น CPU จะทำการตรวจสอบสภาวะของ SW.1, SW.2 และ SW.3 ว่ามีการทำงานหรือไม่ โดยการกดสวิทช์แต่ละอันจะเป็นการเตรียมข้อมูลที่ส่งของตัวเอง ซึ่งข้อมูลดังกล่าวจะถูกคัดลอกไปยัง Transmit Buffer (TXB) เพื่อรอการส่งเมื่อบัสว่าง

3. สำหรับการส่งข้อมูลนั้นในตอนเริ่มต้นจะต้องยกเลิกการอินเทอร์รัพท์เสียก่อน และจากข้อ 2 ข้อมูลจะสามารถส่งผ่านไปยัง TXB ได้เมื่อมีการเซ็ทที่บิต TBS ภายใน Status Register โดย CPU จะคอยตรวจสอบสภาวะของบิตนี้ตลอดเวลา เพื่อตรวจสอบดูว่าภายใน TXB มีการใช้งานอยู่หรือไม่ ถ้าว่างแล้ว CPU จึงสามารถส่งผ่านข้อมูลใหม่ไปยัง TXB ได้และทำการเซ็ทบิต TR ภายใน Command Register เพื่อร้องขอให้เริ่มต้นการส่งข้อมูลดังกล่าว เมื่อมีการเซ็ทที่บิต TCS ภายใน Status Register แสดงว่าการส่งข้อมูลดังกล่าวเป็นผลสำเร็จแล้ว

4. ในการฟิลเตอร์ข้อมูลที่ได้รับนั้น CAN คอนโทรลเลอร์ จะทำให้เองโดยอัตโนมัติ โดยมีโครงสร้างตามที่กำหนดไว้ในข้อ 1 ในการทดลองนี้จะเลือกใช้ Filter 1 ภายใน ACF Bank 1 และ 2 ให้เป็นแบบ Single Filter Standard Frame และแบบ Single Filter Extended Frame ตามลำดับ ซึ่งมีรายละเอียดตามตารางที่ 4.2 และตารางที่ 4.3

ตารางที่ 4.2 แสดงรายละเอียดการฟิลเตอร์แบบ Single Filter Standard Frame

n	0	1 (upper 4 bits)	2	3
ACRn	0001 1100	10XX	XXXX XXXX	XXXX XXXX
AMRn	0000 0000	0011	1111 1111	1111 1111
Accepted message (ID.28-ID.18, RTR)	0001 1100 10xx (ยอมรับเฟรมที่มีหมายเลข ID แบบ A และ B)			

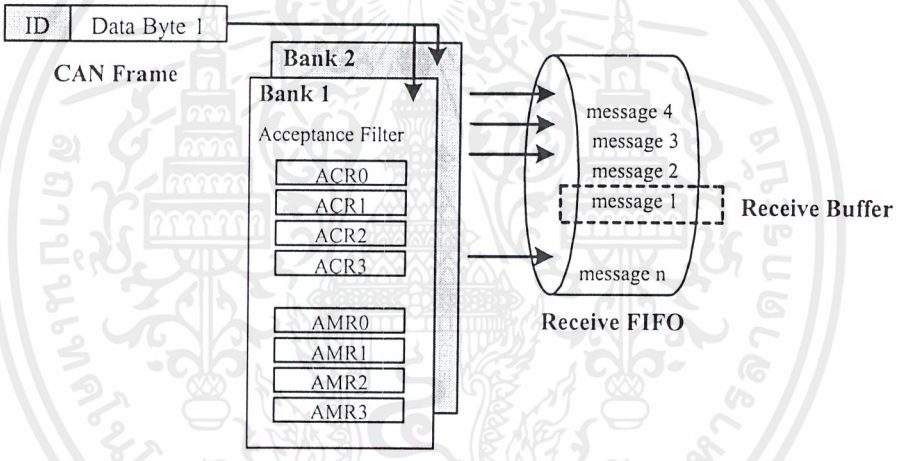
(" X " = irrelevant, " x " = don't care)

ตารางที่ 4.3 แสดงรายละเอียดการฟิลเตอร์แบบ Single Filter Extended Frame

n	0		1		2		3 (upper 6 bits)	
ACRn	0001	1100	10X0	0000	0000	0000	1100	0XXX
AMRn	0000	0000	00x0	0000	0000	0000	0000	0111
Accepted message (ID.28-ID.00, RTR)	0001	1100	10x0	0000	0000	0000	1100	0x
	(ยอมรับเฟรมที่มีหมายเลข ID แบบ C)							

(" X " = irrelevant, " x " = don't care)

เมื่อข้อมูลที่ผ่านการฟิลเตอร์แล้วจะถูกส่งผ่านไปยัง Receive FIFO ซึ่งเป็นพื้นที่สำหรับการจัดเก็บข้อมูลที่รับเข้ามา หลังจากนั้น CPU จะจัดการกับข้อมูลดังกล่าวผ่านทาง Receive Buffer โดยแสดงลักษณะการทำงานไว้ ดังภาพที่ 4.4



ภาพที่ 4.4 แสดงลักษณะการฟิลเตอร์และการจัดเก็บข้อมูล

5. สำหรับการรับข้อมูลนั้นในตอนเริ่มต้นจะต้องยกเลิกการอินเทอร์รัพท์เสียก่อน โดยที่ CPU จะคอยตรวจสอบสภาวะของบิต RBS ภายใน Status Register ตลอดเวลา เพื่อตรวจสอบดูว่ามีการรับข้อมูลเข้ามาภายใน Receive Buffer (RXB) หรือไม่ ถ้าไม่มี CPU จะไปทำกระบวนการอื่น ๆ ต่อไป เมื่อมีการเซ็ทที่บิตนี้จะทำให้ CPU ได้รับข้อมูลจาก CAN คอนโทรลเลอร์ และทำการเซ็ทบิต RRB ภายใน Command Register เพื่อยกเลิกการใช้ RXB หลังจากนั้น CPU จะเริ่มทำกระบวนการจากข้อมูลที่ได้รับ

6. ในการตัดสินใจเพื่อทำกระบวนการจากข้อมูลที่ได้รับนั้น จะใช้การเปรียบเทียบข้อมูลที่ได้รับว่าตรงกับข้อมูลที่กำหนดไว้ในส่วนใด หลังจากนั้นจึงสามารถเริ่มทำกระบวนการที่ได้ โดยจะเป็นการทำให้ LED 1, LED 2 หรือ LED 3 ติด

เมื่อเขียนโปรแกรมตามแผนผังการทำงานดังกล่าว จึงเริ่มขั้นตอนในการทดลองได้ดังนี้

1. ทำการกด SW.1, SW.2 และ SW.3 ของโหนด A ทีละครั้ง และบันทึกผลที่เกิดขึ้นกับโหนด B ลงในตารางที่ 4.4

2. ทำการกด SW.1, SW.2 และ SW.3 ของโหนด B ทีละครั้ง และบันทึกผลที่เกิดขึ้นกับโหนด A ลงในตารางที่ 4.4

3. ทำการกด SW.1 ของโหนด A กับ SW.1, SW.2 หรือ SW.3 ของโหนด B พร้อมกัน และบันทึกผลที่เกิดขึ้นบนบัสลงในตารางที่ 4.5

4. ทำการกด SW.2 ของโหนด A กับ SW.1, SW.2 หรือ SW.3 ของโหนด B พร้อมกัน และบันทึกผลที่เกิดขึ้นบนบัสลงในตารางที่ 4.5

5. ทำการกด SW.3 ของโหนด A กับ SW.1, SW.2 หรือ SW.3 ของโหนด B พร้อมกัน และบันทึกผลที่เกิดขึ้นบนบัสลงในตารางที่ 4.5

ตารางที่ 4.4 แสดงผลการทดลองการสื่อสารข้อมูลระหว่างโหนด A และ B

โหนดของ CAN	สวิทช์	ผลการทดลอง
โหนด A	SW. 1	LED 1 ที่โหนด B ติด
	SW. 2	LED 2 ที่โหนด B ติด
	SW. 3	LED 3 ที่โหนด B ติด
โหนด B	SW. 1	LED 1 ที่โหนด A ติด
	SW. 2	LED 2 ที่โหนด A ติด
	SW. 3	LED 3 ที่โหนด A ติด

ตารางที่ 4.5 แสดงผลการทดลองการตัดสินเข้าใช้บัสระหว่างโหนด A และ B

โหนด B / โหนด A	SW. 1	SW. 2	SW. 3
SW. 1	โหนด A	โหนด A	โหนด A
SW. 2	โหนด B	โหนด A	โหนด A
SW. 3	โหนด B	โหนด B	โหนด A

จากผลของการทดลองที่ได้รับในตารางที่ 4.4 และตารางที่ 4.5 พบว่ามีความถูกต้องตามข้อกำหนดในโปรโตคอลของ CAN กล่าวคือ การส่งและรับข้อมูลสามารถทำได้อย่างถูกต้อง การฟิลเตอร์ข้อมูลสามารถทำได้อย่างถูกต้องตามที่กำหนด และการตัดสินเข้าใช้บัสในกรณีที่ทั้งสอง

โหนดทำการส่งข้อมูลพร้อมกันก็มีความถูกต้อง ได้แก่ การส่งข้อมูลในรูปแบบ SFF จะทำให้ได้เปรียบมากกว่ารูปแบบ EFF และการส่งเป็น Data Frame จะทำให้ได้เปรียบมากกว่า Remote Frame ด้วยเหตุนี้จึงทำให้มีความมั่นใจในประสิทธิภาพของการสื่อสารข้อมูลใน CAN มากยิ่งขึ้น สำหรับการทดลองนี้เป็นการทดสอบการสื่อสารข้อมูลเพียง 2 โหนด ทำให้การมองเห็นภาพรวมของระบบยังไม่ชัดเจน ต่อไปถ้ามีอุปกรณ์เพียงพอจะเพิ่มโหนดในการทดสอบให้มากขึ้น

บทที่ 5
สรุปผลการทดลอง

การสื่อสารโดยใช้ระบบเครือข่าย CAN เหมาะสำหรับการควบคุมอุปกรณ์เล็กทรอนิกส์ที่ต่อร่วมบนบัส ซึ่งมีการทำงานในแบบเรียลไทม์และต้องการความปลอดภัยของข้อมูลสูง เพื่อให้ระบบมีความน่าเชื่อถือและมีประสิทธิภาพในการทำงานมากขึ้น ในขณะที่เป็นการลดความยุ่งยากซับซ้อนและจำนวนสายสัญญาณในการเชื่อมต่อระหว่างอุปกรณ์ในส่วนต่างๆ ลง ด้วยเหตุนี้ CAN จึงเป็นที่นิยมอย่างแพร่หลายในเวลาอันสั้น นอกจากการใช้งานในระบบอิเล็กทรอนิกส์ในรถยนต์แล้วยังมีการนำไปประยุกต์ใช้ในงานอื่นๆ โดยเฉพาะในงานควบคุมอัตโนมัติในอุตสาหกรรม

จากการทดลองเพื่อทดสอบสมรรถภาพในการสื่อสารข้อมูลในระบบ CAN ทำให้เข้าใจถึงหลักการทำงานชัดเจนขึ้น ทั้งในด้านคุณสมบัติและข้อกำหนดในโปรโตคอล จากการทดลองจะได้ข้อมูลที่สำคัญที่เกี่ยวกับการเขียนโปรแกรมสำหรับการสื่อสารข้อมูลในระบบ CAN ซึ่งจัดเป็นอัลกอริทึมเบื้องต้นในการโปรแกรมควบคุมโปรโตคอลของ CAN เพื่อเป็นข้อมูลและแนวทางแก่ผู้ที่สนใจที่จะนำ CAN ไปพัฒนาต่อได้

บรรณานุกรม

1. www.can.bosch.com : เป็นเว็บไซต์ที่ให้ข้อมูล CAN ของ Robert Bosch GmbH
2. Philips Semiconductors, " Data Sheet : P8xC591 Single-chip 8-bit microcontroller with CAN controller " , 160 หน้า
3. Philips Semiconductors, " Data Sheet : PCA82C250 CAN controller interface " , 20 หน้า
4. Philips Semiconductors, " Application Note : P8xC591 in CAN Applications " , 45 หน้า
5. www.keil.com : เป็นเว็บไซต์ที่ให้ข้อมูล CAN ในเรื่องของคอมไพเลอร์ และตัวอย่างการเขียนโปรแกรมภาษา C51
6. www.can-cai.org : เป็นเว็บไซต์ที่ให้ข้อมูล CAN ของกลุ่ม CiA (CAN in Automation)