ระบบแจ้งเตือนและสั่งการระยะไกลผ่านระบบบริการข่าวสารสั้น

นาย วิสุทธิ์ ศรีเมือง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547
ISBN 974-17-6120-1

REMOTE ALARM AND COMMAND SYSTEM VIA SHORT MESSAGE SERVICE SYSTEM

Mr. Wisut Srimuang

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2004

ISBN 974-17-6120-1

หัวข้อวิทยานิพนธ์	ระบบแจ้งเตือนและสั่งการระยะไกลผ่านระบบบริการข่าวสารสั้น
โดย	นาย วิสุทธิ์ ศรีเมือง
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ กฤษดา วิศวธีรานนธ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร. เอกชัย ลีลารัศมี)

..... อาจารย์ที่ปรึกษา
(รองศาสตราจารย์ กฤษดา วิศวธีรานนธ์)

..... กรรมการ
(รองศาสตราจารย์ ดร. วาทิต เบญจพลกุล)

วิสุทธิ์ ศรีเมือง ระบบแจ้งเตือนและสั่งการระยะไกลผ่านระบบบริการข่าวสารสั้น (REMOTE ALARM AND COMMAND SYSTEM VIA SHORT MESSAGE SERVICE SYSTEM) อ. ที่ปรึกษา ร.ศ. กฤษดา วิศวธีรานนท์ ,143 หน้า ISBN 974-17-6120-1

วิทยานิพนธ์ฉบับนี้นำเสนอระบบการแจ้งเตือนและสั่งการทางไกลผ่านระบบบริการข่าวสารสั้น ซึ่งระบบถูกพัฒนาให้อยู่ในรูปบอร์ดควบคุมที่สามารถรับสัญญาณเข้าได้ทั้งแบบดิจิตอลและแอนาลอก และสามารถให้สัญญาณออกแบบดิจิตอลได้ บอร์ดควบคุมนี้จะทำหน้าที่ในการตรวจสอบสัญญาณเพื่อหาความผิดปรกติและจะทำการแจ้งไปยังผู้ใช้งาน ทั้งยังทำหน้าที่ในการรับคำสั่งจากผู้ใช้งานในรูปแบบข่าวสารสั้นเพื่อนำมาประมวลผลในการดำเนินการต่างๆ บอร์ดควบคุมนี้ไม่ได้ถูกออกแบบมาเพื่อใช้ในงานเฉพาะด้าน แต่ถูกออกแบบมาเพื่อเป็นต้นแบบเพื่อให้สามารถพัฒนาใช้กับงานเฉพาะด้านอื่นๆ บอร์ดควบคุมนี้สามารถปรับเปลี่ยนค่าการทำงานต่างๆได้โดยการโหลดค่าจากคอมพิวเตอร์โดยใช้โปรแกรมที่ได้พัฒนาขึ้น หรือปรับเปลี่ยนจากการสั่งการผ่านทางข่าวสารสั้นโดยอาศัยคำสั่งเฉพาะได้

ผลการทดสอบปรากฏว่าระบบสามารถทำงานในการตรวจจับสัญญาณความผิดปรกติ และสามารถแจ้งเตือนรวมทั้งรับการสั่งการมาดำเนินการได้เป็นที่น่าพอใจ สามารถนำไปพัฒนาใช้ในงานด้านอุตสาหกรรมหรืองานด้านอื่นๆได้

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2547

4570549321 : MAJOR ELECTRICAL ENGINEERING

KEY WORD: ALARM UNIT / SHORT MESSAGE SERVICES / REMOTE CONTROL

WISUT SRIMUANG : REMOTE ALARM AND COMMAND SYSTEM VIA SHORT MESSAGE SERVICE SYSTEM. THESIS ADVISOR : ASSOC. PROC. KRISADA VISAVATEERANON, 143 pp. ISBN 974-17-6120-1.

This thesis presents remote alarm and command system via short message services developed in form of controller board which is able to input digital and analog signals and output digital signals. The controller board is used to detect errors from input signals ,alarm to users and input commands from users in short message form to compute for operations. The controller board is designed not only for a specific use but a prototype for general use in many applications. The configuration of the controller board can be programmed by computer and by user via short message service with specific command.

The testing results indicate that the controller board can properly function in error signal detection and send alarm to users and operate properly in response to the command. This system can be further developed for industrial use.

Department Electrical Engineering. Student's signature .....

Field of study Electrical Engineering. Advisor's signature .....

Academic year 2004.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความช่วยเหลืออย่างดียิ่งของ รศ.กฤษดา วิศวธีรนนท์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งได้ให้คำแนะนำ และข้อคิดเห็นต่างๆ พร้อมทั้งจัดหาอุปกรณ์ที่จำเป็นในการวิจัยด้วยดีตลอดมา จึงใคร่ขอกราบขอบพระคุณมา ณ ที่นี้

ข้าพเจ้าขอขอบคุณรองศาสตราจารย์ ดร. เอกชัย ลีลารัศมี และรองศาสตราจารย์ ดร. วาทิต เบญจพลกุล ที่กรุณาสละเวลาอันมีค่าในการเป็นกรรมการในการสอบวิทยานิพนธ์

ข้าพเจ้าขอขอบคุณห้องปฏิบัติการวิจัยวัดคุมทางอุตสาหกรรม ซึ่งเป็นสถานที่ทำการวิจัย รวมถึงเพื่อนพี่น้องนิสิตห้องปฏิบัติการวิจัยวัดคุมทางอุตสาหกรรมทุกท่านที่มีส่วนช่วยเหลือในการให้ข้อคิดเห็น คำแนะนำ และกำลังใจแก่ข้าพเจ้าตลอดระยะเวลาการศึกษาอย่างดียิ่ง

ท้ายนี้ ข้าพเจ้าขอกราบขอบพระคุณคุณพ่อ คุณแม่ของข้าพเจ้าที่ได้เลี้ยงดูและสนับสนุนด้านการศึกษาด้วยดีตลอดมา

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ หน้า

บทที่ หน้า

บทที่ 1

บทนำ

1.1 ความเบื้องต้น

ในปัจจุบันระบบการตรวจสอบความผิดพลาดของระบบการทำงาน(Alarm) และการสั่งการควบคุมกลับทางไกลมีอยู่หลายแบบทั้งแบบไร้สาย และแบบที่ต้องอาศัยสายสัญญาณเชื่อมต่อ ซึ่งทั้งสองแบบต่างก็มีข้อเสีย โดยแบบไร้สายจะใช้การติดต่อผ่านสัญญาณวิทยุซึ่งจำเป็นจะต้องสร้างตัวส่งสัญญาณและรับสัญญาณ เพื่อใช้ในการติดต่อสื่อสารสำหรับการเตือนและสั่งการกลับไปที่ตัวระบบที่ทำการวัดและควบคุมอยู่ และแบบอาศัยสายสัญญาณก็จำเป็นจะต้องติดตั้งสายสัญญาณไปที่ตัวระบบที่ทำการวัดและควบคุมอยู่ หากตัวระบบมีระยะทางไกลจากผู้รับมากก็จะต้องใช้งบประมาณในการติดตั้งสายสัญญาณมาก และยังมีความยุ่งยากในการติดตั้งสายสัญญาณในการใช้งานด้วย

ดั้งนั้นวิทยานิพนธ์ฉบับนี้จึงได้นำเสนอระบบการตรวจสอบความผิดพลาด และการสั่งการกลับทางไกล โดยอาศัยการสื่อสารของตัวตรวจสอบกับผู้รับผ่านทางการสื่อสารในระบบข่าวสารสั้นของโทรศัพท์มือถือ หรือที่เรียกว่า Short Message Services(SMS) โดยการสื่อสารในระบบนี้มีข้อดีคือ เป็นการสื่อสารแบบไร้สายและไม่จำเป็นต้องติดตั้งอุปกรณ์ในการรับและส่งสัญญาณเนื่องจากมีผู้ให้บริการอยู่แล้ว และยังสามารถติดต่อสื่อสารได้ในระยะทางไกล ทำให้ประหยัดค่าใช้จ่ายในการสร้างระบบสื่อสารและยังมีความสะดวกในการติดตั้งและการใช้งาน อีกทั้งยังมีความสะดวกในการติดต่อสื่อสารกับผู้รับที่เป็นตัวบุคคล เนื่องจากในปัจจุบันโทรศัพท์มือถือเป็นสิ่งที่คนส่วนใหญ่มีไว้ใช้งานในชีวิตประจำวัน

ระบบที่จะได้ทำการนำเสนอนี้จะแบ่งออกเป็น 2 ส่วนด้วยกันคือ ส่วนที่เป็นผู้รับโดยมีทั้งแบบที่เป็นบุคคลซึ่งมีโทรศัพท์มือถือใช้งาน หรือเป็นคอมพิวเตอร์ที่เชื่อมต่อข้อมูลกับโทรศัพท์มือถืออยู่ และส่วนที่เป็นตัวตรวจจับและควบคุม โดยส่วนตรวจจับและควบคุมนี้สร้างโมดูลต้นแบบจากไมโครคอนโทรลเลอร์ตระกูล MCS-51 ที่เชื่อมต่อข้อมูลกับโทรศัพท์มือถือเพื่อใช้โทรศัพท์มือถือเป็นสื่อกลางในการติดต่อสื่อสารในระบบข่าวสารสั้น โดยตัวโมดูลนี้จะทำหน้าที่ในการตรวจจับความผิดพลาดจากระบบที่ทำการวัดอยู่ และส่งข่าวสารสั้นไปเตือนแก่ผู้ใช้งานในระบบ อีกทั้งยังสามารถรับคำสั่งจากผู้ใช้งานในรูปข่าวสารสั้นเพื่อนำมาประมวลผลในการทำงานหรือการควบคุมอุปกรณ์ต่างๆที่ตัวโมดูลทำการควบคุมอยู่ โมดูลต้นแบบนี้สามารถตรวจวัด

สัญญาณเข้าที่เป็นแบบแอนาลอกและดิจิตอลได้ และสามารถสั่งการควบคุมออกด้วยสัญญาณแบบดิจิตอลได้

การนำเสนอวิทยานิพนธ์นี้จะนำเสนอโดยแบ่งเป็นหัวข้อดังนี้คือ การติดต่อสื่อสารในระบบข่าวสารสั้น การเชื่อมต่อข้อมูลกับโทรศัพท์มือถือ การออกแบบระบบในด้านฮาร์ดแวร์ การออกแบบระบบในด้านซอฟท์แวร์ การทดสอบระบบและการสรุปผล

1.2 วัตถุประสงค์ของการวิจัย

1. พัฒนาระบบการแจ้งเตือน และควบคุมทางไกล เพื่อสามารถนำมาใช้ในงานอุตสาหกรรมหรืองานควบคุมอื่นๆ โดย อาศัยการสื่อสารแบบข่าวสารสั้นของระบบโทรศัพท์มือถือ
2. พัฒนาให้อยู่ในรูปบอร์ดควบคุม โดย อาศัยโทรศัพท์มือถือเป็นตัวกลางในการสื่อสารสามารถแจ้งเตือนเมื่อมีความผิดพลาดเกิดขึ้นและสามารถรับคำสั่งจากผู้ใช้เพื่อนำมาดำเนินการต่างๆ ได้
3. ระบบสามารถรับสัญญาณเข้าแบบดิจิตอลและแอนาลอก และสามารถให้สัญญาณออกแบบดิจิตอลได้

1.3 ขอบเขตของการวิจัย

1. พัฒนาระบบการแจ้งเตือนและควบคุมทางไกล โดย ใช้การสื่อสารแบบข่าวสารสั้นโดยการสร้างส่วนประกอบที่เป็นฮาร์ดแวร์และซอฟท์แวร์
2. สร้าง Controller Board และ Alarm Control Unit จำนวน 1 ชุดเพื่อการทดสอบการทำงานของระบบ
3. ทดสอบระบบให้สามารถแจ้งเตือนและควบคุมทางไกลได้

1.4 ขั้นตอนของการวิจัย

1. ศึกษาวิธีการสื่อสารแบบข่าวสารสั้นในระบบโทรศัพท์มือถือ
2. ศึกษาวิธีการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือ
3. ทดลองการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือและทดลองส่งและรับข่าวสารสั้นโดยการควบคุมจากคอมพิวเตอร์

4. ทำการทดลองการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือโดยการใช้ไมโครคอนโทรลเลอร์
5. ออกแบบบอร์ดควบคุมเพื่อใช้ในการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือเพื่อใช้เป็นตัวศูนย์กลางในการควบคุมและการแจ้งเตือน
6. พัฒนาซอฟแวร์เพื่อใช้ในการติดต่อระหว่างบอร์ดควบคุมที่ได้สร้างกับคอมพิวเตอร์เพื่อให้สามารถปรับเปลี่ยนค่าต่างๆได้
7. ทดสอบการใช้งานและพัฒนาให้มีประสิทธิภาพ
8. สรุปผลและเขียนวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

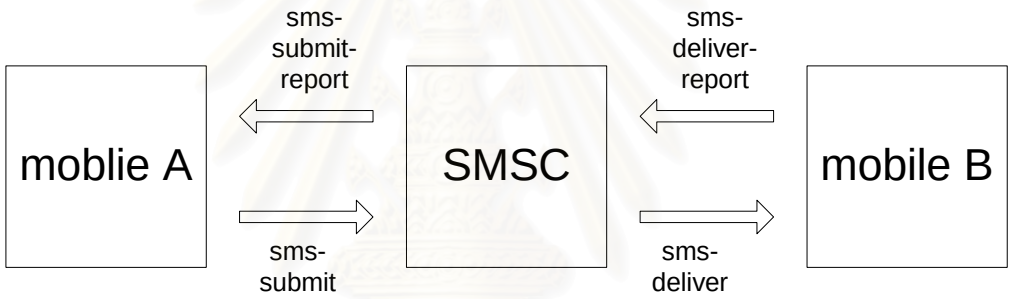
1. เข้าใจหลักการทำงานต่างๆของระบบการสื่อสารแบบข่าวสารสั้น
2. เข้าใจวิธีการติดต่อข้อมูลกับโทรศัพท์มือถือโดยผ่านฮาร์ดแวร์ภายนอก
3. สร้างระบบแจ้งเตือนและสั่งการทางไกลโดยผ่านระบบข่าวสารสั้น
4. ส่งเสริมให้มีการประยุกต์ใช้งานโทรศัพท์มือถือในการวัดคุมและแจ้งเตือนทางไกล

บทที่ 2

การสื่อสารในระบบข่าวสารสั้น

SMS หรือ Short Message Services เป็นระบบการสื่อสารในเครือข่ายของ GSM หรือ Global System of Mobile Communication โดยเป็นการสื่อสารในการรับและส่งข่าวสารที่เป็นตัวอักษรระหว่างโทรศัพท์มือถือซึ่งมีความยาวไม่เกิน 160 ตัวอักษร โดยสามารถครอบคลุมพื้นที่ให้บริการได้เท่ากับที่เครือข่าย GSM สามารถรองรับได้ การคิดอัตราค่าบริการจะคิดเป็นค่าคงที่ตามจำนวนครั้งในการส่งข่าวสาร

2.1 โครงสร้างของการสื่อสารในระบบข่าวสารสั้น



รูปที่ 2.1 องค์ประกอบของการสื่อสารแบบข่าวสารสั้น

ระบบการสื่อสารแบบข่าวสารสั้นนี้มีองค์ประกอบดังแสดงในรูปที่2.1 ซึ่งในระบบนี้จะมีองค์ประกอบไปด้วย

1. *ผู้ส่งและผู้รับข่าวสารสั้น* คือเลขหมายของผู้ใช้บริการที่ต้องการส่งข่าวสารสั้นและเลขหมายของผู้รับข่าวสารสั้น

2. *SMSC* หรือ Short Message Service Center ทำหน้าที่เป็นตัวกลางในการรับและส่งข่าวสารสั้นระหว่างผู้รับและผู้ส่งโดยข่าวสารสั้นที่ผู้ส่งส่งไปนั้นจะถูกเก็บไว้ที่ SMSC และ SMSC จะทำหน้าที่รับผิดชอบในการส่งข่าวสารสั้นต่อไปยังผู้รับ หากผู้รับยังไม่พร้อมที่จะรับหรือรับไม่สำเร็จ SMSC ก็จะทำหน้าที่ในการส่งข่าวสารสั้นไปยังผู้รับใหม่จนกว่าผู้รับจะรับข่าวสารสำเร็จ เมื่อทำการส่งข่าวสารสำเร็จ SMSC อาจจะส่งรายงานการส่งไปยังผู้ส่งหากผู้ส่งข่าวสารสั้นมีการขอให้ส่งรายงานในการส่งข่าวสารสั้น

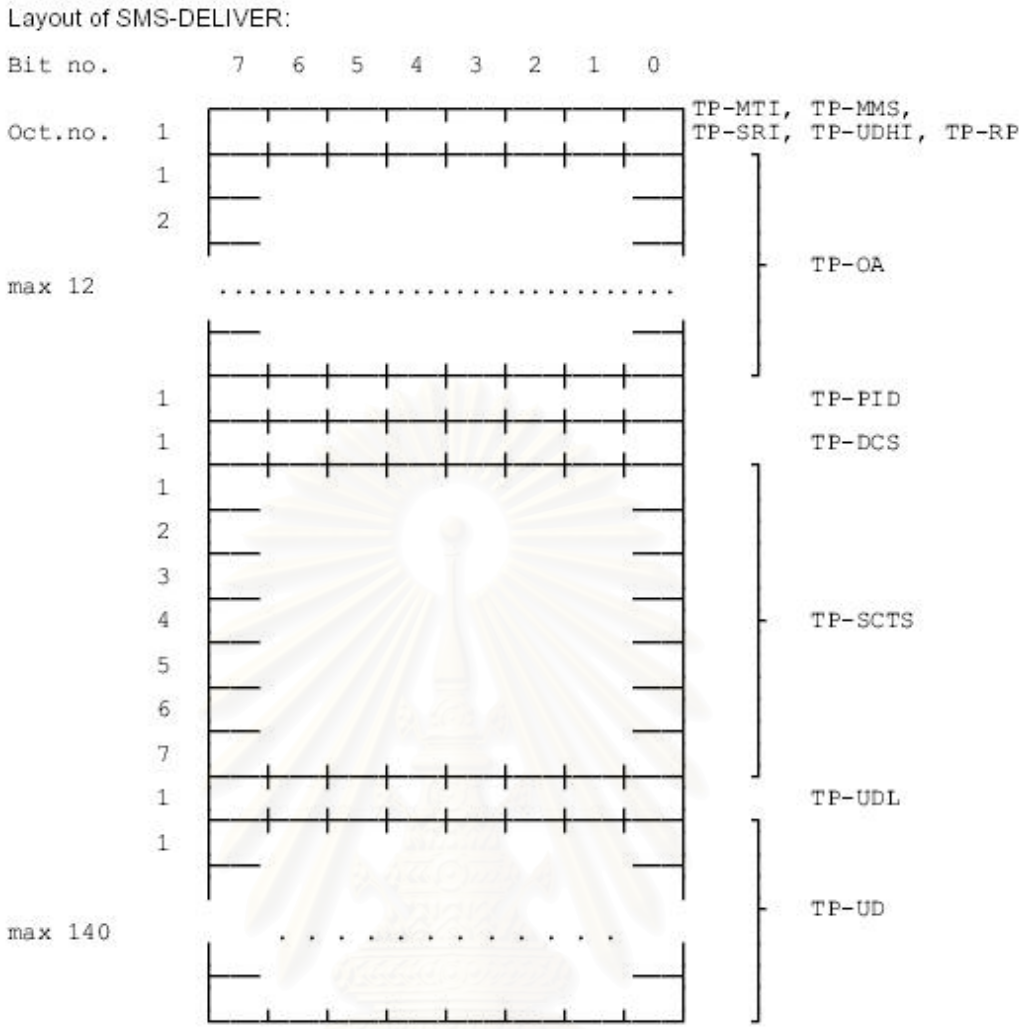
2.2 ประเภทของข่าวสารสั้น

ประเภทของข่าวสารสั้นที่มีการส่งในระบบการสื่อสารนี้มีดังต่อไปนี้

1. SMS-DELIVER เป็นข่าวสารที่ส่งจากศูนย์บริการ ไปยังโทรศัพท์ปลายทาง
2. SMS-DELIVER-REPORT เป็นข่าวสารที่รายงานการได้รับข่าวสารจากโทรศัพท์มือถือปลายทางไปยังศูนย์บริการ
3. SMS-SUBMIT เป็นข่าวสารที่ส่งจากโทรศัพท์มือถือไปยังศูนย์บริการเพื่อทำการส่งต่อไปยังปลายทาง
4. SMS-SUBMIT-REPORT เป็นข่าวสารที่รายงานการส่งข่าวสารไปยังโทรศัพท์มือถือปลายทาง จากศูนย์บริการส่งไปให้ต้นทาง
5. SMS-COMMAND เป็นข่าวสารที่เป็นคำสั่งจากโทรศัพท์ไปยังศูนย์บริการ
6. SMS-STATUS-REPORT เป็นข่าวสารรายงานสถานะภาพจากศูนย์ไปยังโทรศัพท์มือถือ

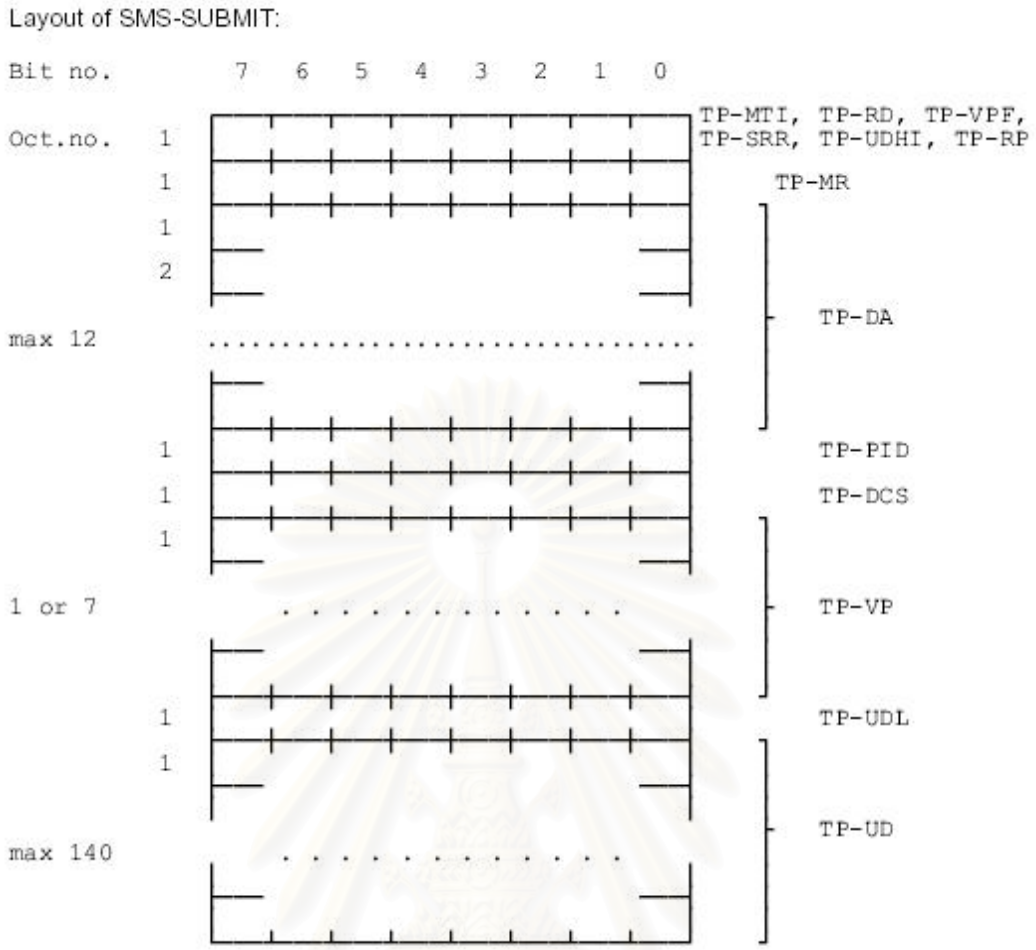
2.3 รายละเอียดองค์ประกอบของข่าวสารสั้น

ในการวิจัยนี้ได้ทำการศึกษาองค์ประกอบของข่าวสารสั้นเพียง 2 ประเภท คือ SMS-DELIVER และ SMS-SUBMIT เพื่อใช้ในการสร้างระบบแจ้งเตือนและสั่งการทางไกล โดยองค์ประกอบของข่าวสารสั้นนี้มีชื่อเรียกว่า TPDU(transfer protocol data unit) ซึ่งเป็นมาตรฐานของรายละเอียดข้อมูลต่างๆในข่าวสารสั้น ซึ่งรายละเอียดนี้สามารถอ้างอิงได้ตามมาตรฐาน GSM 03.40 ซึ่งข่าวสารสั้นแบบ SMS-DELIVER และ SMS-SUBMIT มีองค์ประกอบดังต่อไปนี้



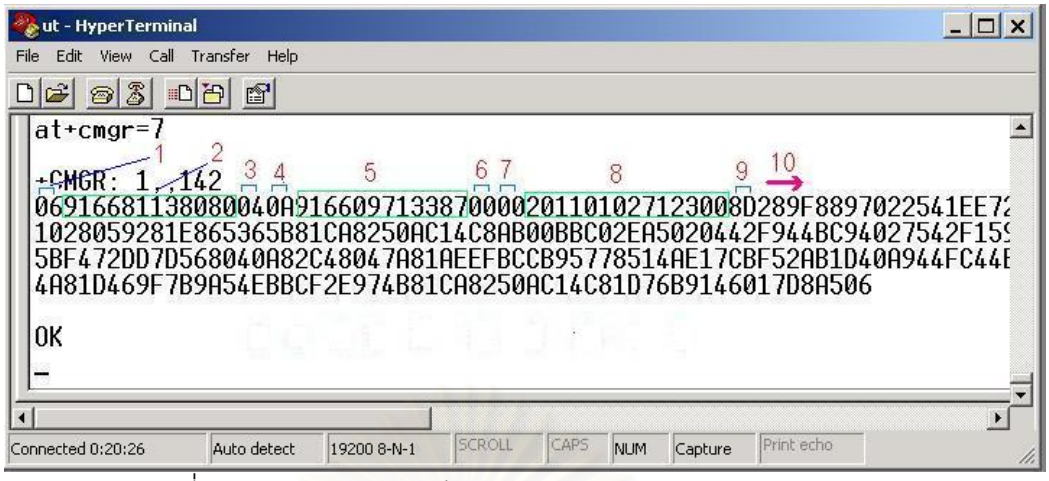
รูปที่ 2.2 องค์ประกอบของข่าวสารประเภท SMS-DELIVER

จากรูปที่ 2.2 แสดงองค์ประกอบมาตรฐานของข่าวสารสั้นประเภท SMS-DELIVER โดยการแสดงไล่ลำดับจากไบต์แรกสุดของข้อมูลในข่าวสารสั้นซึ่งแสดงในส่วนบนสุดของรูปไล่มาจนถึงไบต์สุดท้ายในข่าวสารสั้นซึ่งอยู่ในส่วนล่างสุดของรูป

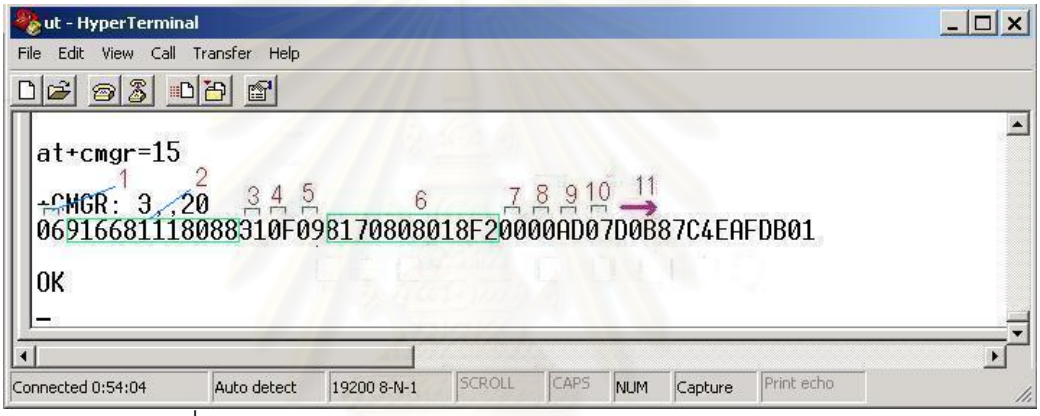


รูปที่ 2.3 องค์ประกอบของข่าวสารประเภท SMS-SUBMIT

จากรูปที่ 2.3 แสดงให้เห็นถึงองค์ประกอบมาตรฐานในข่าวสารสั้นประเภท SMS-SUBMIT ซึ่งมีลักษณะการแสดงในรูปแบบเดียวกับ SMS-DELIVERY โดยข้อมูลรายละเอียดขององค์ประกอบต่างๆแต่ละไบต์จะได้ทำการพูดถึงต่อไป ในขั้นต้นของการทำวิจัยนี้ได้ทำการทดลองทำการอ่านข้อมูลข่าวสารสั้นจากโทรศัพท์มือถือเพื่อนำมาเปรียบเทียบกับมาตรฐานที่ได้ทำการศึกษามา โดยการทดลองติดต่อข้อมูลระหว่างคอมพิวเตอร์กับโทรศัพท์มือถือ รุ่น Siemen C35 โดยการใช้โปรแกรม Hyper Terminal ของระบบปฏิบัติการ windows ซึ่งได้ตัวอย่างข้อมูลดังต่อไปนี้



รูปที่ 2.4 ตัวอย่างรายละเอียดของข่าวสารประเภท SMS-DELIVER



รูปที่ 2.5 ตัวอย่างรายละเอียดของข่าวสารประเภท SMS-SUBMIT

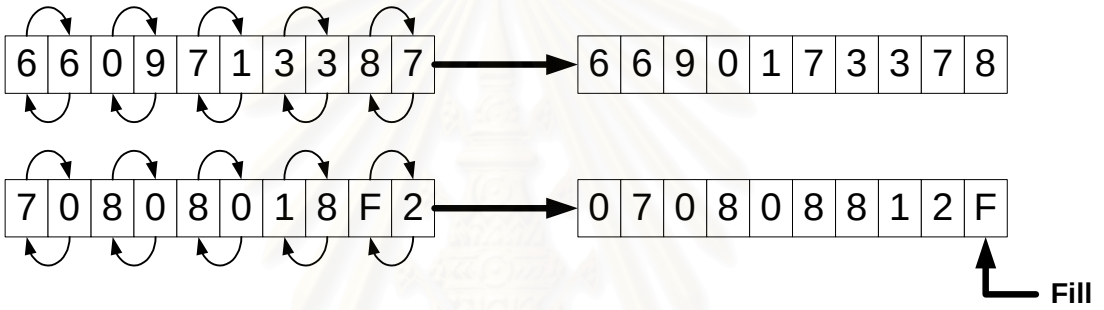
ซึ่งตัวอย่างข้อมูลข่าวสารสั้นในรูปที่ 2.4 และ 2.5 นั้นเป็นลักษณะของข้อมูลที่อยู่ในโทรศัพท์มือถือรุ่น siemen C35 เท่านั้นซึ่งอาจจะคล้ายคลึงหรือต่างกับรุ่นอื่นๆก็ได้ โดยก่อนที่จะได้พูดถึงรายละเอียดของข้อมูลในข่าวสารสั้นนี้ จะขอนำเสนอลักษณะของข้อมูลที่มีอยู่ในข่าวสารสั้นก่อน ซึ่งลักษณะของข้อมูลในข่าวสารสั้นสามารถจำแนกประเภทได้ดังต่อไปนี้

Integer representation เป็นข้อมูลที่ใช้แสดงตัวเลขจำนวนเต็มซึ่งอาจจะเกิดจากข้อมูลหนึ่งไบต์หรือจำนวนหลายไบต์รวมกันเพื่อแสดงเป็นตัวเลขจำนวนเต็มหนึ่งตัว ซึ่งสามารถแสดงได้ในตำแหน่งที่ 9 ในรูป 2.4 และ ในตำแหน่งที่ 10 ในรูป 2.5 ซึ่งเป็นส่วนข้อมูลที่ใช้แสดงความยาวของข่าวสารสั้นซึ่งมีข้อมูลอยู่ในรูปแบบ integer และถูกแสดงอยู่ในรูปเลขฐาน 16 โดยมีค่า 8D และ 07 ตามลำดับซึ่งแสดงว่าข่าวสารสั้นมีความยาว 141 และ 7 ตัวอักษรตามลำดับ

Octet representation เป็นข้อมูล 8 บิตที่ใช้แสดงเลขฐาน 10 จำนวน 1 ตัว

Semi-octet representation เป็นข้อมูลจำนวน 8 บิตที่ใช้แสดงเลขฐาน 10 จำนวน 2 ตัวโดยการแบ่งแสดงทีละ 4 บิต โดยที่บิตที่ 0 ถึง 3 จะแสดงเลขที่มีหลักมากกว่าบิตที่ 4

ถึง 7 และหากจำนวนตัวเลขที่แสดงไม่ครบเป็นจำนวนคู่ก็จะมีการเติม F ใส่เพื่อให้ครบเป็นจำนวนคู่ดังแสดงได้ในตำแหน่งที่ 5 ในรูป 2.4 และ ในตำแหน่งที่ 6 ในรูป 2.5 ซึ่งเป็นตำแหน่งที่แสดงเลขหมายของผู้ส่งข่าวสารสั้น และเลขหมายของผู้รับข่าวสารสั้นตามลำดับ โดยเลขหมายในรูป 2.4 จะขึ้นต้น 91 เพื่อแสดงว่าเป็นรูปแบบเลขหมายแบบสากล โดยในรูปที่ 2.4 มีเลขหมายคือ 6609713387 ซึ่งเป็นการแสดงข้อมูลในรูปเลขฐาน 16 และสามารถแปลงเป็นเลขหมายจริงคือ 6690173378 ตามการตีความหมายข้อมูลในรูปแบบ semi-octet และส่วนข้อมูลในรูปที่ 2.5 จะขึ้นต้นด้วย 81 เพื่อบอกให้ทราบว่าเป็นเลขหมายแบบท้องถิ่น โดยมีเลขหมายคือ 70808018F2 ซึ่งสามารถตีความหมายได้เป็น 070808812 โดยมีการเติม F(1111_2) ลงไปเพื่อให้ครบ 1 ไบต์ในไบต์สุดท้าย



รูปที่ 2.6 ตัวอย่างการอ่านข้อมูลแบบ semi-octet representation

Alphanumeric representation เป็นข้อมูลตัวอักษรในข่าวสารสั้นที่มีลักษณะเป็นแบบ 7 บิตตามมาตรฐานของ GSM 03.38[8] โดยจะอยู่ในตำแหน่งที่ 10 ของรูปที่ 2.4 และตำแหน่งที่ 11 ของรูปที่ 2.5 ซึ่งแสดงอยู่ในรูปเลขฐาน 16 การอ่านข้อมูลในส่วนนี้จำเป็นที่จะต้องมีการทำการถอดรหัสออกมา ซึ่งจะได้มีการกล่าวถึงในหัวข้อการบีบข้อมูลในข่าวสารสั้น

จากการศึกษาเปรียบเทียบข้อมูลในส่วนต่างๆ ของข่าวสารสั้นทั้งประเภท SMS-SUBMIT และ SMS-DELIVER เปรียบเทียบกับมาตรฐาน GSM 03.40[9] สามารถแสดงรายละเอียดได้ดังต่อไปนี้

MTI(message type indicator) เป็นส่วนที่ใช้แสดงประเภทของข่าวสารสั้นซึ่งจะอยู่ในไบต์แรกของ SMS-DELIVER และ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2และ2.3 โดยจะอยู่ในบิต 0 และ 1 ในไบต์แรก มีหน้าที่แสดงประเภทของข่าวสารสั้น แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.4 และ 2.5 โดยมีรายละเอียดในการดังต่อไปนี้

บิต1	บิต0	ประเภทข่าวสาร
0	0	SMS-DELIVER
0	0	SMS-DELIVER REPORT
1	0	SMS-STATUS REPORT
1	0	SMS-COMMAND
0	1	SMS-SUBMIT
0	1	SMS-SUBMIT REPORT
1	1	Reserved

MMS(more message to send) เป็นส่วนที่ใช้แสดงว่ามีข่าวสารจากศูนย์รอที่จะส่งมาอีกหรือไม่โดยจะเป็นข้อมูลบิตที่ 2 ในไบต์แรกของ SMS-DELIVER ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.4 โดยมีรายละเอียดในการดังต่อไปนี้

บิต 2	0	มีข่าวสารรอที่จะทำการส่งมาอีก
	1	ไม่มีข่าวสารที่จะทำการส่งมาอีก

VPF(validity period format) เป็นส่วนที่ใช้แสดงรูปแบบของข้อมูลระยะเวลาที่ศูนย์บริการจะทำการเก็บข่าวสารสั้นที่ส่งไว้ที่ศูนย์ โดยจะอยู่ในบิต 3 และ 4 ของไบต์แรกของ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.5 โดยมีความหมายดังนี้

บิต4	บิต3	ลักษณะของรูปแบบ
0	0	รูปแบบไม่ได้กำหนด
1	0	แสดงอยู่ในรูปแบบ integer represented
0	1	Reserved
1	1	รูปแบบแสดงในรูป semi-octet represented

SRI(status report indicator) เป็นส่วนที่แสดงว่าผู้ส่งข่าวสารต้องการให้มีการรายงานสถานะในการส่งข่าวสารกลับหรือไม่โดยจะอยู่ในบิตที่ 5 ของไบต์แรกของ SMS-DELIVER ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.4 โดยมีรายละเอียดดังต่อไปนี้

บิต 5 0 ไม่มีการแจ้งรายงานในการส่งข่าวสารไปยังผู้ส่ง

1 มีการแจ้งรายงานในการส่งข่าวสารไปยังผู้ส่ง

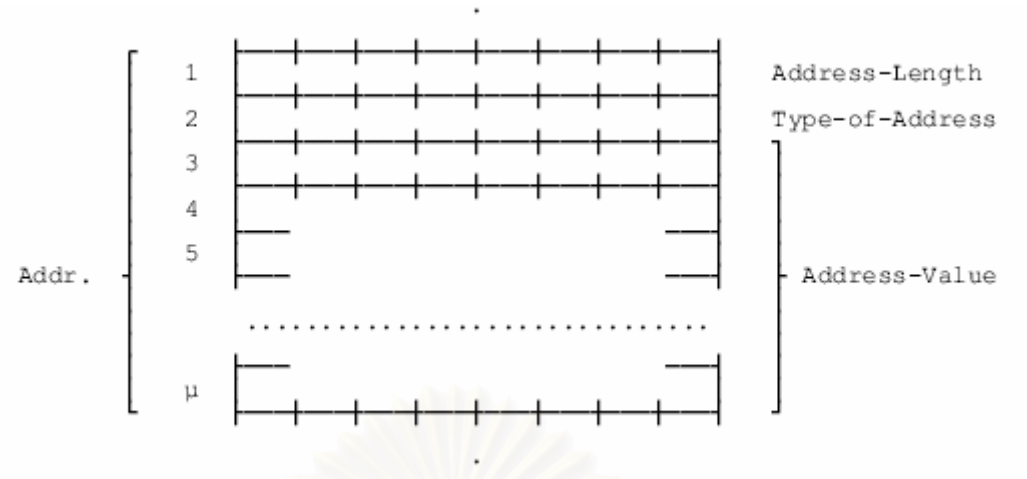
SRR(status report request) เป็นส่วนที่ใช้แจ้งให้ศูนย์บริการส่งรายงานกลับเมื่อทำการส่งข่าวสารสำเร็จแล้วโดยจะอยู่ในบิทที่ 5 ของไบต์แรกของ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.5 โดยมีรายละเอียดดังต่อไปนี้ โดยมีรายละเอียดดังนี้

บิต 5 0 ไม่ต้องการให้ศูนย์ส่งรายงานการส่งข่าวสาร

1 ต้องการให้ศูนย์ส่งรายงานการส่งข่าวสาร

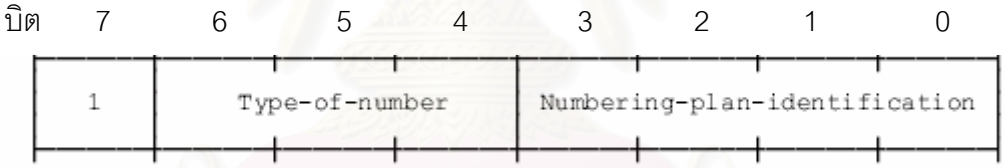
MR(message reference) เป็นส่วนที่แสดงเลขจำนวนเต็มที่มีการกำหนดให้กับข่าวสารสั้นที่ส่งในแต่ละครั้งโดยทุกครั้งที่มีการส่งข่าวสารสั้นจากโทรศัพท์มือถือจะมีการกำหนดตัวเลขให้ข่าวสารนั้นๆโดยมีค่าอยู่ระหว่าง 0 ถึง 255 ซึ่งค่าที่ใช้ในการกำหนดนี้จะเป็นค่าที่ได้มีการบันทึกไว้ใน SIM ในส่วน Last-Used-TP-MR บวกอีก 1 และจะทำการบันทึกค่าใหม่นี้กลับไปในส่วน Last-Used-TP-MR เพื่อใช้ในการส่งครั้งต่อไป ข้อมูลในส่วนนี้มีประโยชน์ในการส่งข่าวสารสั้นในกรณีที่โทรศัพท์มือถือและศูนย์บริการทำการติดต่อกันล้มเหลว และต้องมีการส่งข้อมูลไปให้ศูนย์บริการใหม่ โทรศัพท์มือถือจะต้องส่งข่าวสารนั้นโดยใช้ค่า MR เดิมจนกว่าการส่งข้อมูลจะเรียบร้อย หากใช้ค่า MR คนละค่าจะถือเป็นการส่งข่าวสารใหม่แม้ว่าข้อมูลจะเหมือนข่าวสารเดิมทุกประการ ข้อมูลในส่วนนี้จะอยู่ในไบต์ที่สอง SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 4 ของรูปที่ 2.5 โดยแสดงในรูปเลขฐาน 16(integer represented)

OA(originating address) เป็นส่วนที่ใช้แสดงเลขหมายของผู้ส่งข่าวสารสั้น โดยจะอยู่ในไบต์ที่ 2 เป็นต้นไปใน SMS-DELIVER มีจำนวนความยาวมากสุด 12 ไบต์ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2 โดยมีรูปแบบข้อมูลดังรูปที่ 2.7



รูปที่ 2.7 โครงสร้างข้อมูลของเลขหมาย

จากรูปที่ 2.7 โครงสร้างของข้อมูลเลขหมายจะประกอบไปด้วย ความยาวของเลขหมายในไบต์แรก ตามด้วยประเภทของเลขหมายในไบต์ถัดมา ซึ่งทั้งสองค่าเป็นข้อมูลแบบ integer แล้วจึงตามด้วยเลขหมายที่มีลักษณะข้อมูลแบบ semi-octet represented โดยรายละเอียดของประเภทเลขหมายสามารถแสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 รายละเอียดส่วนแสดงประเภทของเลขหมาย

โดยจากรูปที่ 2.7 ในแต่ละส่วนมีรายละเอียดดังต่อไปนี้

บิต 7 มีค่าเป็น 1 ตลอด

บิต 6 5 4

0 0 0	Unknown
0 0 1	International number
0 1 0	National number
0 1 1	Network specific number
1 0 0	Subscriber number
1 0 1	Alphanumeric (มีลักษณะเป็นตัวอักษรแบบ 7 บิตเหมือนข่าวสารสั้น)

1 1 0 Abbreviate number

1 1 1 Reserved for extension

บิต 3 2 1 0

0 0 0 0 Unknown

0 0 0 1 ISDN(E.164./E.163)

0 0 1 1 Data numbering plan (X.121)

1 0 0 0 Nation numbering plan

1 0 0 1 Private numbering plan

1 0 1 0 ERMES(European Radio Messaging System) numbering plan

1 1 1 1 Reserved for extension

โดยจากการได้ศึกษาข่าวสารสั้นพบการกำหนดเป็น 2 รูปแบบ คือ 91H หมายถึง รูปแบบสากลโดยขึ้นต้นเลขหมายด้วย 66 แล้วตามด้วยเลขหมายที่ตัด 0 ออก และ 81H หมายถึง เลขหมายในประเทศโดยขึ้นต้นด้วย 0 แล้วตามด้วยหมายเลข

จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้ เลขหมายผู้ส่งจะอยู่ในตำแหน่งที่ 5 ของรูปที่ 2.4

DA(destination address) แสดงเลขหมายของผู้รับข่าวสารสั้น โดยจะอยู่ในไบต์ที่3 เป็นต้นไปใน SMS-SUBMIT มีจำนวนความยาวมากสุด 12 ไบต์ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 6 ของรูปที่ 2.5 โดยมีลักษณะข้อมูลเหมือนกับ OA

PID(protocal identifier) เป็นส่วนที่ใช้ในการแสดงลักษณะโปรโตคอลการติดต่อสื่อสารของโทรศัพท์มือถือกับศูนย์บริการ โดยจะอยู่ในไบต์ที่ 14 และ 15 ของ SMS-DELIVER และ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2 และ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 6 ของรูปที่ 2.4 และตำแหน่งที่ 7 ของรูปที่ 2.5 โดยมีรายละเอียดดังต่อไปนี้

รายละเอียดสามารถแบ่งเป็นกรณีได้ดังต่อไปนี้

*กรณี บิต7,บิต6 เป็น 0,0 และ

-บิต5 เป็น 0 ไม่มี interworking

-บิต5เป็น 1 มี interworking โดยในบิต4 ถึงบิต0จะมีความหมายดังต่อไปนี้

บิต 4..0

00000	implicit
00001	telex
00010	group 3 telefax
00011	group 4 telefax
00100	voice telephone
00101	ERMES(Europian Radio Messaging System)
00110	Nation Paging System
00111	Videotex(T.100/T.101)
01000	teletex,carrier unspecified
01001	teletex,in PSPDN
01010	teletex,in CSPCDN
01011	teletex,in analog PSTN
01100	teletex,in digital ISDN
01101	UCI(Universal Computer Interface)
01110..01111	reserved
10000	a message handling facility
10001	any public X.400-based message handling system
10010	Intertnet Electrical Mail
10011..10111	reserved
10011..10111	ค่าที่กำหนดแล้วแต่ข้อตกลงของศูนย์บริการกับผู้ใช้
11111	ศูนย์บริการจะเปลี่ยนข่าวสารสั้นจาก TP-Data-Coding-Scheme หนึ่งไปยัง TP-Data-Coding-Scheme ซึ่งผู้ใช้บริการสามารถรองรับได้

ถ้าบิต5 ของ PID ใน SMS-SUBMIT TPDU ถูกตั้งเป็น 1 แสดงว่าผู้รับมีลักษณะเป็น telematic device ซึ่งมีประเภทดังที่ได้ระบุไปในบิตที่ 4 ถึง 0 ตามที่ได้แสดงไว้ด้านบน และศูนย์บริการมีหน้าที่จะต้องแปลงข่าวสารสั้นให้สามารถเหมาะสมกับประเภทของอุปกรณ์ตามที่ได้กำหนดในบิตที่ 4 ถึง 0

*กรณีบิต7,บิต6 เป็น 0,1 บิต5 ถึง บิต0 จะมีความหมายดังต่อไปนี้

บิต 5..0

000000	Short Message Type 0
000001	Replace Short Message Type 1
000010	Replace Short Message Type 2
000011	Replace Short Message Type 3
000100	Replace Short Message Type 4
000101	Replace Short Message Type 5
000110	Replace Short Message Type 6
000111	Replace Short Message Type 7
001000..011110	Reserved
011111	Return Call Message
100000..111111	Reserved

แต่ในการติดต่อข้อมูลประเภทข่าวสารสั้นธรรมดาระหว่างโทรศัพท์มือถือกับศูนย์บริการนั้น PID จะถูกตั้งค่าให้เป็น 0 ดังแสดงในรูปที่ 2.4 และ 2.5 และในงานวิจัยนี้มีการใช้การบริการข่าวสารสั้นแบบธรรมดาเท่านั้น ดังนั้นจึงขอนำเสนอรายละเอียดในส่วน PID นี้แต่เพียงสังเขป

DCS(data coding scheme) เป็นส่วนที่ใช้แสดงลักษณะการเข้ารหัสของข่าวสารในส่วนข้อมูลของข่าวสารสั้น โดยจะอยู่ในไบต์ที่ 15 และ 16 ของ SMS-DELIVER และ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2 และ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 7 ของรูปที่ 2.4 และตำแหน่งที่ 8 ของรูปที่ 2.5 โดยมีรายละเอียดดังต่อไปนี้

บิต 7..4 Coding Group Bits	บิต 3..0
0 0 0 0	Alphabet Indication ข้อมูลในกลุ่มนี้ใช้ในการแสดงประเภทตัวอักษรในข่าวสารสั้น โดยบิต 3..0 จะมีรายละเอียดดังนี้

0 0 0 0 Default Alphabet

0 0 0 1..1 1 1 1 Reserved

0 0 0 1..1 0 1 1 Reserved coding group

1 1 0 0 Message Waiting Indication Group:
Discard Message
ข้อมูลในกลุ่มนี้ใช้แสดงว่ามีข่าวสารรออยู่ที่ศูนย์บริการ โดยโทรศัพท์จะแจ้งผู้ใช้ถึงข่าวสารแต่จะละทิ้งข้อมูลส่วนเนื้อความโดยความหมายในบิต 3..0 จะเหมือนกับกลุ่ม 1101

1 1 0 1 Message Waiting Indication Group:
Store Message
ใช้ในการแสดงประเภทข่าวสารที่ทำการรอที่ศูนย์ โดยในกลุ่มนี้โทรศัพท์จะทำการเก็บรายละเอียดของข่าวสารที่แจ้งไว้ด้วย โดยบิต 3..0 มีรายละเอียดดังนี้
บิต 3
0 การแจ้งเตือนไม่ทำงาน
1 การแจ้งเตือนทำงาน
บิต 2 ไม่มีการใช้งานและถูกตั้งเป็น 0
บิต 1 บิต 0
0 0 Voicemail Message Waiting
0 1 Fax Message Waiting
1 0 Electronic Mail Message Waiting
1 1 Other Message Waiting

1 1 1 0 Reseved Coding Group

1 1 1 1 Data coding/message class
ข้อมูลในกลุ่มนี้ใช้ในการแสดงประเภทของข่าวสารสั้น
บิต 3 ไม่ถูกใช้งานและมีค่าเป็น 0
บิต 2 message coding

0 Default alphabet

1 8-bit data

บิต 1 บิต 0 message class

0 0 Class 0

0 1 Class 1

1 0 Class 2

1 1 Class 3

จากการศึกษาพบว่าลักษณะตัวอักษรมี 2 แบบ คือ 7 บิต และ 8 บิต โดยแบบ 7 บิตจะเป็นตัวอักษรภาษาอังกฤษมาตรฐาน แต่แบบ 8 บิตจะเป็นตัวอักษรอื่นที่มีการกำหนดใช้งาน โดยจากการศึกษาตัวอย่างข่าวสารสั้นพบลักษณะของ DCS สองแบบคือ 00H ซึ่งเป็นภาษาอังกฤษโดยมีลักษณะเป็นตัวอักษรแบบ 7 บิต และ 08H เป็นตัวอักษรภาษาไทยแบบ 8 บิต โดยในหนึ่งไบต์แสดงตัวอักษรหนึ่งตัวซึ่งต่างกับข้อมูลแบบ 7 บิต รายละเอียดของตัวอักษรแบบ 7 บิตจะได้กล่าวต่อไปในหัวข้อการบีบข้อมูลในข่าวสารสั้น

SCTS(service center time stamp) เป็นส่วนที่ใช้แสดงเวลาที่ข่าวสารสั้นถูกส่งมาถึงศูนย์บริการโดยจะอยู่ในไบต์ที่ 16 ถึง 22 ของ SMS-DELIVER ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.2 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 8 ของรูปที่ 2.4 มีขนาด 7 ไบต์มีลักษณะเป็น semi-octet represent และมีรายละเอียดดังนี้โดยค่า 2 semi-octet เท่ากับข้อมูล 1 ไบต์

	Year	Month	Day	Hour	Minute	Second	Time Zone
Semi-octet	2	2	2	2	2	2	2

VP(validity period) เป็นส่วนที่ใช้แสดงให้ศูนย์บริการทราบว่าต้องการให้เก็บข่าวสารสั้นที่ทำการส่งไว้ที่ศูนย์เป็นระยะเวลาเท่าไร โดยจะอยู่ในไบต์ที่ 17 เป็นต้นไปของ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] ดังรูปที่ 2.3 แต่จากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 9 ของรูปที่ 2.5 โดยรูปแบบข้อมูลจะมีทั้ง integer represented(1 byte) หรือ semi-octet represented(7 bytes) โดยการกำหนดรูปแบบนี้สามารถทำโดยการตั้งค่าที่ VPF(validity period format) ใน SMS-SUBMIT หากมีการตั้งค่าเป็นแบบ integer represented จะมีวิธีการคำนวณค่าระยะเวลาดังนี้

ค่า VP	ค่าระยะเวลา
0 ถึง 143	(VP+1)×5 นาที
144 ถึง 167	12 ช.ม.+((VP-143)×30 นาที)
168 ถึง 196	(VP-166)×1 วัน
197 ถึง 255	(VP-192)×1 สัปดาห์

หากมีการตั้งค่าเป็นแบบ semi-octet represented จะมีลักษณะรูปแบบเหมือนกับการแสดงเวลาในส่วน SCTS

UDL(user data length) เป็นส่วนที่ใช้แสดงความยาวของข่าวสารสั้นที่จะส่งหรือได้รับ โดยมีรูปแบบเป็น integer represented และจะอยู่ก่อนข่าวสารสั้นทั้งใน SMS-DELIVER และ SMS-SUBMIT ดังแสดงในรูป 2.2 และ 2.3 ตามมาตรฐาน GSM 03.40[9] และจากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 9 ของรูปที่ 2.4 และตำแหน่งที่ 10 ของรูปที่ 2.5 โดยแสดงเป็นเลขฐาน 16

RP(reply path) เป็นส่วนที่ใช้ในการแสดงว่ามีการตั้งค่าเส้นทางการตอบรับกลับหรือไม่โดยจะอยู่ในบิตที่ 7 ในไบต์แรกของทั้ง SMS-DELIVER และ SMS-SUBMIT ตามมาตรฐาน GSM 03.40[9] และจากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.4 และ 2.5 โดยหากมีการตั้งค่าเป็น 1 จะหมายถึงต้องการให้เลขหมายปลายทางตอบรับการได้รับข่าวสารสั้นกลับมาที่ศูนย์บริการที่ผู้ส่งใช้ในการส่งข่าวสารโดยตรง หากมีค่าเป็น 0 จะหมายถึงให้ปลายทางตอบรับกลับมาโดยใช้ศูนย์บริการที่ปลายทางใช้บริการอยู่ การตั้งค่าเป็น 1 นี้อาจจะทำให้เกิดปัญหาได้ เนื่องจากหมายเลขปลายทางอาจจะไม่ได้ใช้บริการอยู่ในเครือข่ายเดียวกับต้นทางทำให้ไม่สามารถตอบรับโดยใช้ศูนย์บริการของต้นทางได้

RD (reject duplicate) เป็นส่วนที่ใช้ในการแจ้งศูนย์บริการเกี่ยวกับการจัดการกับข่าวสารสั้นซ้ำ(มี MR และ DA เดียวกันและถูกส่งมาจาก OA เดียวกัน) ของ SMS-SUBMIT โดยจะอยู่ในบิตที่ 2 ในไบต์แรกของ SMS-SUBMIT ดังแสดงในรูปที่ 2.3 ตามมาตรฐาน GSM 03.40[9] และจากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.5 โดยมีรายละเอียดดังต่อไปนี้

บิต 2	0	สั่งให้ศูนย์บริการรับข่าวสารสั้นซ้ำ
	1	สั่งให้ศูนย์บริการละเลยข่าวสารสั้นซ้ำ

UDHI (user data header indicator) เป็นส่วนที่ใช้ในการแสดงว่าข่าวสารสั้นบริเวณที่เป็นข่าวสารนั้นประกอบด้วยข่าวสารเพียงอย่างเดียว หรือมีส่วนรายละเอียดต้นขั้ว (header) อย่างอื่นด้วย โดย UDHI จะอยู่ในบิตที่ 6 ของไบต์แรกใน SMS-SUBMIT และ SMS-DELIVER ดังแสดงในรูป 2.2 และ 2.3 ตามมาตรฐาน GSM 03.40 และจากการศึกษาและเปรียบเทียบจากข้อมูลที่อ่านได้จะอยู่ในตำแหน่งที่ 3 ของรูปที่ 2.4 และ 2.5 โดยมีรายละเอียดดังต่อไปนี้

บิต 6 0 มีเฉพาะข่าวสาร

1 ในส่วนต้นของข่าวสารมีรายละเอียดต้นขั้ว

UD(user data) เป็นส่วนที่เป็นเนื้อข่าวสารสั้นโดยอาจจะมีเฉพาะข่าวสารสั้นหรืออาจจะมีส่วนที่เป็นรายละเอียดส่วนอื่นก่อนข่าวสารสั้นดังที่ได้กล่าวมาแล้วใน UDHI โดยหากมีรายละเอียดอื่นเพิ่มเติมจะอยู่ในรูปแบบดังนี้

Length of User Data Header 1 ไบต์

Information-Element-Identifier 'A' 1 ไบต์

Length of Information-Element 'A' 1 ไบต์

Information-Element 'A' Data 1 to n ไบต์

⋮

Information-Element-Identifier 'n' 1 ไบต์

Length of Information-Element 'n' 1 ไบต์

Information-Element 'n' Data 1 to n ไบต์

โดย Length of User Data Header คือ ความยาวของส่วนต้นขั้วทั้งหมดเป็นไบต์โดยที่ไม่รวมตัวเอง

Information-Element-Identifier คือ ประเภทของรายละเอียดย่อย

Length of Information-Element คือ ความยาวของรายละเอียดย่อยโดยที่ไม่รวมตัวเอง

Information-Element Data คือ รายละเอียดย่อย

โดยในกรณีนี้ค่า UDL จะเป็นค่าความยาวของข่าวสารสั้นรวมกับความยาวของส่วนต้นขั้วทั้งหมด

ประเภทของรายละเอียดย่อย(Information-Element-Identifier)ในส่วนต้นขั้วมีดังต่อไปนี้

VALUE(hex) ความหมาย

00 เป็นข่าวสารสั้นที่นำมาประกอบกัน(concatenated short message)

01 เป็นการแจ้งข่าวสารรออยู่ที่ศูนย์บริการ(special SMS message indication)

Other value สำรองไว้ใช้ในอนาคต

Concatenated Short Message

ในการสื่อสารแบบข่าวสารสั้นนั้นสามารถส่งข้อมูลได้คราวละ 140 ตัวอักษรหากใช้ข้อมูลแบบ 8 บิต หรือ 160 ตัวอักษรในกรณีข้อมูลแบบ 7 บิต ดังนั้นหากต้องการส่งข้อมูลที่มีความยาวมากกว่านั้นสามารถทำได้โดยการใช้ข่าวสารสั้นหลายๆอันมาต่อกัน โดยสามารถเชื่อมต่อกันได้มากสุด 255 ข่าวสาร และอุปกรณ์ที่รับข่าวสารจะทำหน้าที่ในการรวมข่าวสารสั้นเพื่อแสดงแก่ผู้ใช้

ในรายละเอียดของข่าวสารสั้นประเภทนี้จะมีรายละเอียดในส่วน Information-Element Data ดังนี้

ไบต์ 1 เป็นเลขแสดงค่า message reference ซึ่งคล้ายกับค่า MR ที่ได้กล่าวไปแล้วด้านบนแต่ต่างตรงที่ทุกข่าวสารสั้นแบบนี้จะต้องมีค่าเดียวกันเพื่อใช้ในการรวมข่าวสารย่อยแต่ละอันเป็นข่าวสารรวมได้อย่างถูกต้อง

ไบต์ 2 เป็นเลขแสดงจำนวนข่าวสารย่อยทั้งหมดที่มีการนำมารวมกันทั้งหมด

ไบต์ 3 เป็นเลขแสดงค่าลำดับในข่าวสารย่อยนั้นๆ

Special Message Indication

ในส่วนนี้จะทำหน้าที่คล้ายกับส่วน DCS ในการทำการแจ้งว่ามีข่าวสารรออยู่ที่ศูนย์บริการแต่จะมีรายละเอียดมากกว่าใน DCS เช่นจำนวนของข่าวสารแต่ละแบบที่รออยู่ ซึ่งข้อมูลส่วนนี้เป็นส่วนที่ได้มีการพัฒนาขึ้นในภายหลังจากส่วน DCS ซึ่งรายละเอียดในส่วน information-element data มีรายละเอียดดังต่อไปนี้

ไบต์ 1 เก็บค่าการบันทึกและประเภทของข่าวสารที่แจ้ง

บิต 7

0 ไม่มีการบันทึกข่าวสารหลังจากการแจ้ง

1 มีการบันทึกข่าวสารหลังจากการแจ้ง

บิต 6..0

000 0000 Voice Message Waiting

000 0001 Fax Message Waiting

000 0010 Electronic Mail Message Waiting

000 0011 Other Message Waiting

ไบต์ 2 เก็บค่าจำนวนข่าวสารที่ทำการรออยู่โดยเป็นจำนวน 0 ถึง 255

ในการแจ้งข่าวสารรอในแต่ละประเภทนั้น จะใช้ข่าวสารย่อยคนละอันในการแจ้ง เช่นมีข่าวสารรอทั้งแฟกซ์และเมล ซึ่งหากมีข่าวสารทำการรออยู่หลายประเภทก็จะมีจำนวนข่าวสารย่อยอยู่เป็นจำนวนเท่ากับประเภทที่ทำการรออยู่ ดังแสดงในตอนต้นของหัวข้อ UD(user data)

จากที่ได้แสดงรายละเอียดของ TPDU มาแล้วข้างต้นจะสังเกตเห็นว่าข้อมูลที่อ่านออกมาได้จะมีลักษณะต่างไปจากรูปแบบข้อมูลตามมาตรฐาน GSM 03.40[9] ในรูปที่ 2.2 และ 2.3 เนื่องจากการอ่านค่าข้อมูลจากโทรศัพท์ในรุ่นนี้จะเป็นการอ่านค่าข้อมูล ซึ่งประกอบไปด้วยหมายเลขของศูนย์บริการในส่วนแรก(ส่วนที่ 2 ในรูปที่ 2.4 และ 2.5) ซึ่งมีลักษณะเป็นข้อมูลแบบ semi-octet แล้วจึงตามด้วยข้อมูลของข่าวสารสั้นหรือ TPDU ซึ่ง TPDU นี้จะมีลักษณะเช่นเดียวกับลักษณะของ TPDU ตามมาตรฐาน GSM 03.40[9]

ดังนั้นจึงสามารถสรุปข้อมูลในส่วนต่างๆของ SMS-SUBMIT และ SMS-DELIVER ที่ได้ทำการศึกษาเป็นตารางดังต่อไปนี้

ตารางที่ 2.1 ส่วนประกอบต่างๆของ SMS-DELIVER

ตำแหน่ง	รายละเอียด
1	Unknown
2	Service Center Number
3	MTI,MMS,SRI,UDHI,RP
4	Length of Number
5	Sender Number
6	PID
7	DCS
8	Time Stamp
9	Data Length
10	Data

ตารางที่ 2.2 ส่วนประกอบต่างๆของ SMS-SUBMIT

ตำแหน่ง	รายละเอียด
1	Unknown
2	Service Center Number
3	MTI,RD,VPF,SRR,UDHI,RP
4	Message Reference
5	Length of Number
6	Receiver Number
7	PID
8	DCS
9	Validity Period
10	Data Length
11	Data

2.3 การบีบข้อมูลในข่าวสารสั้น

จากตัวอย่างข้อมูลข่าวสารสั้นที่ได้ยกตัวอย่างในหัวข้อที่ผ่านมา จะเห็นได้ว่าข้อมูลส่วนที่เป็นตัวอักษรไม่สามารถอ่านได้เนื่องจากในการสื่อสารแบบข่าวสารสั้นนั้น มีการบีบข้อมูลให้เล็กลงเพื่อให้สามารถส่งข้อมูลได้มากขึ้น ซึ่งในอัตราส่วนปกติข่าวสารสั้น 1 ข่าวสารจะมีข้อมูลได้ 140 ไบต์ แต่ในการส่งตัวอักษรที่เป็นรหัส ASCII นั้นจะมีการบีบเพื่อทำให้สามารถส่งได้เพิ่มเป็น 160 ตัวอักษรซึ่งวิธีการในการบีบอัดข้อมูลสามารถแสดงได้ดังต่อไปนี้

เนื่องจากรหัส ASCII(ที่เป็นภาษาอังกฤษ) จะเริ่มต้นที่ 00H ไปถึง 7FH จึงทำให้บิตใหญ่สุด หรือบิตที่ 7 มีค่าเป็น 0 เสมอ ดังนั้นในการส่งข้อมูลในระบบข่าวสารสั้นนั้น ใน 1 ไบต์จะเกิดจากการรวมบิตข้อมูลในตัวอักษร ASCII ซึ่งใช้เพียง 7 บิต โดยตัดบิต 7 ที่มีค่าเป็น 0 ออกไปกับบิตล่างของตัวอักษรตัวถัดไปเพื่อให้ครบ 8 บิต และทำต่อเนื่องไปเรื่อยๆจนครบทุกตัวอักษร จึงทำให้สามารถบีบข้อมูลได้ในอัตรา 8 ต่อ 7 ซึ่งวิธีการสามารถแสดงได้ดังนี้

ตัวอักษร 1 ตัวใน 1 ไบต์

ตำแหน่ง	7	6	5	4	3	2	1	0
ข้อมูล	0	1a	1b	1c	1d	1e	1f	1g

ตัวอักษร 2 ตัวใน 2 ไบต์

7	6	5	4	3	2	1	0
2g	1a	1b	1c	1d	1e	1f	1g
0	0	2a	2b	2c	2d	2e	2f

ตัวอักษร 3 ตัว 3 ไบต์

7	6	5	4	3	2	1	0
2g	1a	1b	1c	1d	1e	1f	1g
3f	3g	2a	2b	2c	2d	2e	2f
0	0	0	3a	3b	3c	3d	3e

ตัวอักษร 8 ตัวใน 7 ไบต์

7	6	5	4	3	2	1	0
2g	1a	1b	1c	1d	1e	1f	1g
3f	3g	2a	2b	2c	2d	2e	2f
4e	4f	4g	3a	3b	3c	3d	3e
5d	5e	5f	5g	4a	4b	4c	4d
6c	6d	6e	6f	6g	5a	5b	5c
7b	7c	7d	7e	7f	7g	6a	6b
8a	8b	8c	8d	8e	8f	8g	7a

ตัวอย่างในการอ่านข้อมูลในข่าวสารสั้น

จากรูปที่ 2.4 ซึ่งเป็นข่าวสารสั้นแบบ SMS-SUBMIT นั้นเราจะได้นำมาเป็นตัวอย่างในการอ่านข้อมูลในข่าวสารสั้น

- เริ่มจากค่า UDL หรือ User Data Length ซึ่งมีค่า 07 ในฐาน16 แสดงว่ามีข้อมูลที่เป็นตัวอักษร 7 ตัว
- ข้อมูลมีดังนี้ D0B87C4EAFDB01 ซึ่งแสดงอยู่ในรูปเลขฐาน 16

ไบต์แรก D0 = 11010000 = 01010000(1) + 00000001 shl7(2)

ไบต์ที่สอง B8 = 10111000 = 01110000 shr1(2) + 00000010 shl6(3)

ไบต์ที่สาม 7C = 01111100 = 01110000 shr2(3) + 00000011 shl5(4)

ไบต์ที่สี่ 4E = 01001110 = 01110000 shr3(4) + 00000100 shl4(5)

ไบต์ที่ห้า AF = 10101111 = 01110000 shr4(5) + 00010101 shl3(6)

ไบต์ที่หก DB = 11011011 = 01100000 shr5(6) + 00110110 shl2(7)

ไบต์ที่เจ็ด 01 = 00000001 = 01000000 shr6(7) + 0

โดย shr คือ การเลื่อนบิตไปทางขวา

shl คือ การเลื่อนบิตไปทางซ้าย

ดังนั้นจะได้ตัวอักษรตัวที่ (1) เป็น 01010000 = 50H = P

ตัวอักษรตัวที่ (2) เป็น 01110000 + 00000001 = 71H = q

ตัวอักษรตัวที่ (3) เป็น 01110000 + 00000010 = 72H = r

ตัวอักษรตัวที่ (4) เป็น 01110000 + 00000011 = 73H = s

ตัวอักษรตัวที่ (5) เป็น 01110000 + 00000100 = 74H = t

ตัวอักษรตัวที่ (6) เป็น 01100000 + 00010101 = 75H = u

ตัวอักษรตัวที่ (7) เป็น 01000000 + 00110110 = 76H = v

จะได้ข้อมูลที่เป็นตัวอักษรเป็น Pqrstuv

และจากการที่ได้กล่าวถึงข้อมูลแบบ 8 บิตมาแล้วในข้างต้น ข้อมูลแบบ 8 บิตนั้นจะใช้ข้อมูล 8 บิตในการแสดงตัวอักษร 1 ตัวซึ่งใช้ในการแสดงตัวอักษรอื่นที่ไม่ใช่ตัวอักษรภาษาอังกฤษ และจะสามารถทำการส่งข่าวสารสั้นได้เพียง 140 ตัวอักษรเท่านั้นในการส่งหนึ่งครั้ง

บทที่ 3

การเชื่อมต่อข้อมูลกับโทรศัพท์มือถือ

ในงานวิจัยนี้ได้ทำการติดต่อข้อมูลกับโทรศัพท์มือถือ รุ่น Siemen C35 โดยอาศัยสายสัญญาณที่เรียกว่า data cable ซึ่งเป็นสายสัญญาณที่มีการติดต่อตามมาตรฐาน RS232 โดยในการติดต่อนั้นจะมีชุดคำสั่งที่ใช้ในการสั่งการต่างๆกับโทรศัพท์ โดยชุดคำสั่งนี้มีชื่อเรียกว่า AT-Command ซึ่งเป็นชุดคำสั่งที่มีใช้กับอุปกรณ์พวกโมเด็มแต่ก็มีส่วนคำสั่งที่ใช้สำหรับโทรศัพท์มือถือเช่นกัน แต่จะได้นำเสนอเฉพาะในส่วนที่เกี่ยวข้องกับการสื่อสารในระบบข่าวสารสั้นของโทรศัพท์มือถือเท่านั้น

ก่อนอื่นจะได้กล่าวถึงลักษณะของชุดคำสั่ง AT-Command โดยชุดคำสั่งที่ใช้ในงานวิจัยนี้อ้างอิงมาจาก siemen specific command ซึ่งใช้กับรุ่น S35,C35 และ M35 ซึ่งเป็นชุดคำสั่งที่มีการอ้างอิงมาจาก GSM 07.05[10] อีกทีซึ่งจากการศึกษาเปรียบเทียบพบว่ามีลักษณะคล้ายคลึงกับกลุ่มคำสั่ง AT ที่ในโทรศัพท์แบบอื่น ต่างเพียงรายละเอียดปลีกย่อยเท่านั้น โดยที่ชุดคำสั่งนี้จะมีลักษณะดังต่อไปนี้

AT+___= n เป็นคำสั่งในการตั้งค่าตัวแปรต่างๆ

AT+___? เป็นคำสั่งในการแสดงค่าของตัวแปรที่ได้ตั้งไว้

AT+___=? เป็นคำสั่งในการแสดงค่าตัวเลือกของตัวแปรที่สามารถตั้งได้

AT+___ เป็นคำสั่งที่ใช้ในการดำเนินการที่ไม่ต้องมีการตั้งค่าตัวแปร

ตัวอย่างคำสั่งของชุดคำสั่ง AT-Command ที่เกี่ยวข้องกับการสื่อสารแบบข่าวสารสั้นมีดังต่อไปนี้

3.1 AT+CSMS เป็นคำสั่งที่ใช้ในการเลือกระบบให้บริการมีรูปแบบดังนี้

Test command

AT+CSMS=? Response +CSMS (service)

โดย service มี 2 ค่าคือ 0 GSM 3.40 และ 3.41

1 GSM 3.40 และ 3.41 และเข้ากันได้กับชุด AT phase 2+

Read command

AT+CSMS? Response +CSMS (service),mt,mo,bm

โดย แต่ละตัวมีค่าดังนี้(ในรุ่นที่เลือกใช้ทดลอง)

Service 0 GSM 3.40 และ 3.41

mt 1 Mobile terminated type support(มือถือเป็นฝ่ายรับ)

m0 1 Mobile originated type support (มือถือเป็นฝ่ายส่ง)

bm 0 Mobile terminated type not support(การประกาศ)

Write command

AT+CSMS=service Response +CSMS: mt,mo,bm OK/ERROR/

+CSMS ERROR

3.2 AT+CPMS เป็นคำสั่งที่ใช้ในการเลือกหน่วยความจำของข่าวสารสั้น

Test command

AT+CPMS=? Response +CPMS: (list of mem1), (list of mem2),

(list of mem3)

โดยค่าทั้ง 3 นี้มีเพียงค่าเดียวคือ “SM” (sim card) ในรุ่นที่ใช้ทดลองโดยที่

mem1 คือ หน่วยความจำที่ใช้ในการอ่านและลบข่าวสารสั้น

mem2 คือ หน่วยความจำที่ใช้ในการเขียนและส่งข่าวสารสั้น

mem3 คือ หน่วยความจำที่ใช้ในการเก็บข่าวสารสั้นที่ได้รับมา

Read command

AT+CPMS? Response +CPMS: mem1,used1,total1, mem2,used2,total2

mem3,used3,total3

Write command

AT+CPMS=mem1,mem2,mem3

Response +CPMS: used1,total1, used2,total2, used3,total3

3.3 AT+CMGF เป็นคำสั่งที่ใช้ในการเลือกรูปแบบของข้อมูลในข่าวสารสั้น

Test command

AT+CMGF=? Response +CMGF: (list of mode)

โดยในรุ่นที่ใช้ทดลองมีเพียงแบบเดียวคือ 0 (PDU mode)

Read command

AT+CMGF? Response +CMGF: mode

Write command

AT+CMGF=mode Response OK/ERROR

3.4 AT+CSCA เป็นคำสั่งที่ใช้ในการตั้งค่าหมายเลขของศูนย์บริการ

Test command

AT+CSCA=? Response OK(สามารถตั้งได้แต่ไม่มีตัวเลือกให้ต้องทราบเอง)

Read command

AT+CSCA? Response +CSCA: sca,tosca

โดยที่ sca คือ เลขหมายของศูนย์บริการ

tosca คือ รูปแบบของหมายเลข

Write commnad

AT+CSCA=sca,tosca Response OK/ERROR

3.5 AT+CMNI เป็นคำสั่งที่ใช้ในการกำหนดรูปแบบการแจ้งเมื่อได้รับข่าวสารสั้นเข้ามาใหม่

Test command

AT+CNMI=? Response +CNMI: mode,mt,bm,ds,bfr

โดยที่มีรายละเอียดดังต่อไปนี้

mode 0 ไม่มีการแจ้ง

1 ไม่มีการแจ้งหากไม่ได้เชื่อมต่ออยู่ หากเชื่อมต่อให้มีการแจ้ง

2 ทำการบัฟเฟอร์และแสดงข่าวสาร(ในเครื่องที่ใช้ทดลองไม่สามารถทำได้)

mt 0 ไม่มีการแสดง

1 มีการแสดงข่าวสารเข้าในรูป +CMTI: mem,index

2 มีการแสดงข่าวสารเข้า(ยกเว้นข่าวสาร Class2)ในรูป +CMT: alpha,length<CR><LF><pdu> ส่วนข่าวสารใน Class2 จะมีการแสดงเหมือนกับแบบ mt=1

3 ข่าวสาร Class3 จะมีการแสดงเหมือนกับ mt=2 และข่าวสารที่มี data coding scheme อื่นจะแสดงเหมือน mt=1

ในรุ่นที่ใช้สามารถเลือกได้เพียง 0 และ 1 เท่านั้น

bm 0 ไม่มีการแจ้งเมื่อได้รับข่าวสารสั้นแบบ broadcast

1 มีการแจ้งเมื่อได้รับข่าวสารสั้นแบบ broadcast ในรูปแบบ +CBM: length<CR><LF><pdu>

ds 0 ไม่มีการแจ้งเมื่อได้รับข่าวสารสั้นแบบรายงาน

1 มีการแจ้งเมื่อได้รับข่าวสารสั้นแบบรายงานในรูปแบบ +CDS:

length<CR><LF><pdu>

bfr 1 มีการ buffer

โดย pdu คือ protocal data unit ซึ่งเป็นลักษณะของข้อมูลในระบบข่าวสารสั้น

Read command

AT+CNMI? Response +CNMI: mode,mt,bm,ds,bfr

Write command

AT+CNMI=mode,mt,bm,ds,bfr Response OK/ERROR/+CMS ERROR

3.6 AT+CNMA เป็นคำสั่งที่ใช้ในการเลือกการแสดงข่าวสารสั้นเข้าโดยไม่ต้องมีการบันทึกลงในหน่วยความจำ

Test command

AT+CNMA=? Response +CNMA:(list of mode) มีเพียง 0 คือไม่สามารถเลือกได้)

Write command

AT+CNMI=mode Response OK/ERROR/+CMS ERROR

3.7 AT+CMGL เป็นคำสั่งที่ใช้ในการดูรายการข่าวสารสั้นที่ได้ทำการบันทึกไว้

Test command

AT+CMGL=? Response +CMGL: (list of stat) โดยมีรายละเอียดดังนี้

stat 0 ข่าวสารสั้นเข้าที่ถูกบันทึกไว้และยังไม่ได้อ่าน

1 ข่าวสารสั้นเข้าที่ถูกบันทึกไว้และอ่านแล้ว

2 ข่าวสารสั้นที่ถูกบันทึกไว้และยังไม่ได้ส่ง

3 ข่าวสารสั้นที่ถูกบันทึกไว้และส่งแล้ว

4 ทั้งหมด

Write command

AT+CMGL=stat Response +CMGL:index,stat,alpha,

length<CR><LF><pdu>

3.8 AT+CMGR เป็นคำสั่งที่ใช้ในการอ่านข่าวสารสั้นที่ได้ถูกบันทึกไว้ทีละอัน

Test command

AT+CMGR=? Response OK หรือ +CMS ERROR

Write command

AT+CMGR=index Response +CMGR: stat,alpha,length

<CR><LF><pdu> หรือ +CMS ERROR

โดยที่ index คือเลขลำดับของข่าวสารสั้นที่ได้ทำการบันทึกไว้

3.9 AT+CMGS เป็นคำสั่งใช้ในการส่งข่าวสารสั้นโดยจะต้องใส่เนื้อหา pdu ด้วย

Test command

AT+CMGS=? Response OK

Write command

AT+CMGS=length<CR><pdu><ctrl-Z/ESC>

Response ถ้าสำเร็จ +CMGS: mr

ถ้าไม่สำเร็จ +CMS ERROR

โดยที่ length คือ ความยาวของ pdu

mr คือ message reference(ดังที่ได้กล่าวแล้วในบทที่ 2)

3.10 AT+CMSS เป็นคำสั่งที่ใช้ในการส่งข่าวสารสั้นที่มีบันทึกไว้

Test command

AT+CMSS=? Response OK

Write command

AT+CMSS=index,da,toda Response ถ้าสำเร็จ +CMSS: mr

ถ้าไม่สำเร็จ +CMS ERROR

โดยที่ da คือ เลขหมายปลายทางที่จะส่ง

toda คือ รูปแบบของเลขหมาย

3.11 AT+CMGW เป็นคำสั่งที่ใช้ในการเขียนข่าวสารสั้นลงไปในหน่วยความจำ

Test command

AT+CMGW=? Response OK

Write command

AT+CMGW=length,stat<CR><pdu><ctrl-Z/ESC> Response +CMGW: index

+CMGW ERROR

3.12 AT+CMGD เป็นคำสั่งที่ใช้ในการลบข่าวสารสั้นที่ถูกบันทึกไว้

Test command

AT+CMGD=? Response OK

Write command

AT+CMGD=index Response OK/ERROR/+CMS ERROR

3.13 AT+CSCB เป็นคำสั่งที่ใช้ในการรับข่าวสารสั้นแบบ broadcast

Test command

AT+CSCB=? Response +CSCB: (list of mode)

0 รับข่าวสารตามที่ระบุใน mids และ dcss

1 ไม่รับข่าวสารตามที่ระบุใน mids และ dcss

Read command

AT+CSCB? Response +CSCB:mode,mids,dcss

โดยที่ mids คือ ข้อความแบบ combination of CBM message Ids

dcss คือ ข้อความแบบ combination of CBM data coding scheme

Write command

AT+CSCB=mode,mids,dcss Response +CSCB:mode,mids,dcss

3.14 AT+CMGC เป็นคำสั่งที่ใช้ในการส่งข่าวสารสั้นประเภทคำสั่ง

Test command

AT+CMGC=? Response OK

Write command

AT+CMGC=length<CR><pdu><ctrl-Z/ESC>

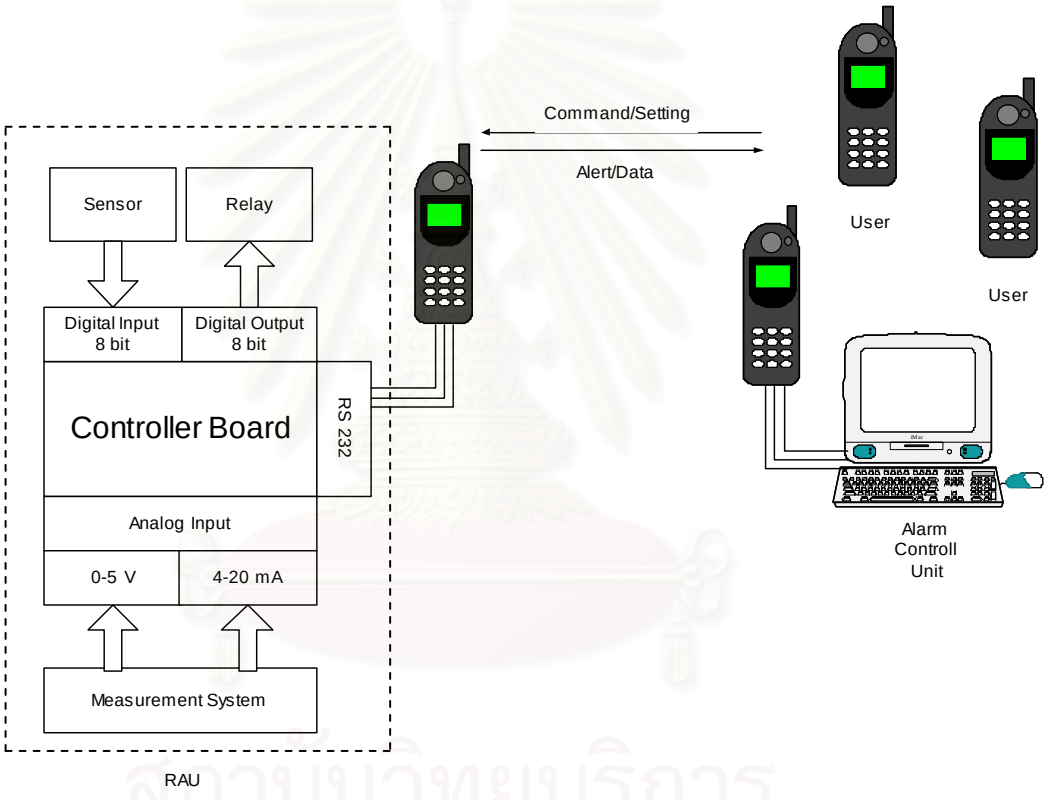
Response ถ้าสำเร็จ +CMGC: mr

ถ้าไม่สำเร็จ +CMS ERROR

บทที่ 4

การออกแบบและการสร้างระบบ

ในงานวิจัยนี้มีจุดประสงค์เพื่อสร้างระบบแจ้งเตือนและสั่งการทางไกลผ่านระบบข่าวสารสั้นที่ประกอบไปด้วยบอร์ดควบคุมที่สามารถรับสัญญาณขาเข้าจากอุปกรณ์ตรวจวัดต่างๆ เพื่อนำมาประมวลผลหาความผิดพลาด และทำการแจ้งเตือนไปยังผู้ใช้งานโดยมีทั้งแบบบุคลหรือแบบคอมพิวเตอร์ โดยโครงสร้างของระบบสามารถแสดงได้ดังรูปที่ 4.1



รูปที่ 4.1 โครงสร้างของระบบแจ้งเตือนและสั่งการทางไกลผ่านระบบข่าวสารสั้น

4.1 องค์ประกอบของระบบ

จากรูปที่ 4.1 องค์ประกอบของระบบแบ่งออกเป็น 2 ส่วนดังนี้คือ

4.1.1 Remote Alarm Unit(RAU) เป็นส่วนที่ทำหน้าที่ในการรับสัญญาณภายนอกที่แจ้งความผิดพลาดเข้ามา หรือรับสัญญาณวัดมาเพื่อประมวลผลหาความผิดพลาดและทำการส่งข่าวสารสั้นไปยังผู้ใช้โดยการสั่งการโทรศัพท์มือถือ ผ่านสายสัญญาณที่เชื่อมต่อข้อมูลอยู่

และทำการรับข่าวสารสั้นที่ส่งมาจากผู้ใช้เพื่อนำไปประมวลผลเพื่อดำเนินการต่างๆ เช่น การควบคุมอุปกรณ์ที่เชื่อมต่อกับตัว RAU อยู่

4.1.2 User เป็นผู้ใช้งานระบบที่จะได้รับข่าวสารสั้นแจ้งในกรณีที่เกิดเหตุการณ์ผิดพลาดขึ้นและสามารถที่จะสั่งการไปยัง RAU ได้ถ้าหากอยู่ในกลุ่มผู้ใช้ที่มีสิทธิในการสั่งการได้ โดย user แบ่งออกเป็น 2 ประเภทคือ

- บุคคล คือ บุคคลทั่วไปที่มีโทรศัพท์มือถือใช้งาน
- Alarm Control Unit เป็นส่วนอุปกรณ์คอมพิวเตอร์ที่เชื่อมต่อข้อมูลกับโทรศัพท์มือถือโดยผ่านโปรแกรมที่ได้พัฒนาขึ้นในงานวิจัยเพื่อทำหน้าที่ในการเก็บข้อมูลในการเตือนไว้ เพื่อสามารถใช้ในการวิเคราะห์ได้ในภายหลัง(data logging)

4.2 คุณสมบัติของระบบ

คุณสมบัติของระบบสามารแบ่งออกได้ดังนี้

4.2.1 Digital I/O

- สามารถรับข้อมูลเข้าแบบดิจิตอลได้(ON/OFF) และสามารถให้ข้อมูลออกเป็นแบบดิจิตอลได้เพื่อนำไปใช้ในการควบคุมอุปกรณ์ภายนอก โดยรับสัญญาณเข้าได้ 8 ช่อง และให้สัญญาณออกได้ 8 ช่อง

4.2.2 Analog Input

- สามารถรับสัญญาณแอนาลอกมาประมวลผลได้โดยมีความละเอียด 12 บิต โดยสามารถรับสัญญาณได้ทั้งแบบ 0-5V และ 4-20mA อย่างละช่องสัญญาณ
- สามารถตั้งระดับการเตือนได้ 2 ค่า คือกรณีสัญญาณมีค่าสูงกว่า high limit และสัญญาณมีค่าต่ำกว่า low limit ของแต่ละช่องสัญญาณ

4.2.3 สามารถตั้งค่าจากคอมพิวเตอร์ผ่านพอร์ต RS232

- ตั้งค่าการเลือก enable/disable ช่องสัญญาณขาเข้าต่างๆ
- ตั้งค่าข่าวสารสั้นที่จะส่งเมื่อเกิดเหตุการณ์ต่างๆโดยแบ่งเป็น

*ข่าวสารสั้นเมื่อมีสัญญาณเข้ามาที่ช่องสัญญาณดิจิตอลขาเข้า(ON/OFF) 8 ช่อง ช่องละ 1 ข่าวสาร

*ข่าวสารสั้นเมื่อสัญญาณที่วัดต่ำว่า low limit หรือสูงกว่า high limit ของช่องสัญญาณแอนาลอก ทั้งสองช่องสัญญาณ

*ข่าวสารสั้นเมื่อสัญญาณที่วัดสอดคล้องกับเงื่อนไขที่ได้ตั้งเอาไว้(condition)

โดยข่าวสารสั้นที่ตั้งได้ในแต่ละกรณีมีความยาวไม่เกิน 25 ตัวอักษร

- ตั้งชื่อพอร์ตต่างๆเพื่อความสะดวกในการใช้งานโดยแบ่งเป็นพอร์ตดิจิตอลขาเข้า 8 ชื่อ พอร์ตดิจิตอลขาออก 8 ชื่อ และพอร์ตแอนาลอก 2 ชื่อ โดยมีความยาวไม่เกิน 8 ตัวอักษร
- ตั้งเลขหมายของผู้ใช้งานได้ โดยสามารถตั้งเลขหมายของผู้ที่สามารถใช้งานในระบบนี้ได้มากสุด 10 เลขหมายโดยแบ่งระดับความสำคัญเป็น 2 ระดับคือ ระดับที่สามารถอ่านค่าจากการวัดได้เพียงอย่างเดียว กับระดับที่สามารถสั่งการ RAU ได้ด้วย(การแจ้งเตือนจะทำการเตือนทุกเลขหมายไม่มีการแบ่งระดับความสำคัญ)
- สามารถตั้งเงื่อนไข(condition) ในการแจ้งเตือนได้ โดยการกำหนดให้ตรวจสัญญาณเข้าหลายอันพร้อมกันหากเกิดความผิดพลาดพร้อมกันให้ทำการแจ้งเตือน โดยสามารถตั้งได้ 8 เงื่อนไขในรูปของการ AND ของแต่ละสัญญาณเข้า
- ตั้งเลขหมายที่ต้องการแจ้งเตือนด้วยข่าวสารสั้น เมื่อเกิดเหตุการณ์ผิดพลาดต่างๆได้ โดยสามารถตั้งได้มากกว่า 1 เลขหมายต่อ 1 เหตุการณ์
- ตั้งความสำคัญให้เลขหมายว่ามีสิทธิในการสั่งการ RAU หรือไม่

4.2.4 แจ้งเตือนเมื่อเกิดความผิดพลาด

- เมื่อมีสัญญาณเข้ามาจากSensor (ON/OFF)
- เมื่อระดับสัญญาณที่วัดเกินกว่าระดับหรือต่ำกว่าระดับที่กำหนด
- เมื่อสัญญาณวัดสอดคล้องกับเงื่อนไขที่ได้ตั้งไว้

4.2.5 สามารถรับการสั่งการจากผู้ใช้

- สั่งปิดหรือเปิดพอร์ตดิจิตอลขาออกได้(ON/OFF)
- สั่งอ่านค่าจากพอร์ตขาเข้า หรือขาออกได้
- สั่งเปลี่ยนข่าวสารสั้นที่จะเตือน เมื่อเกิดความผิดพลาดที่ช่องสัญญาณดิจิตอลขาเข้าได้(สงวนการเปลี่ยนข่าวสารสั้นของช่องแอนาลอก และการตรวจสอบแบบเงื่อนไขไว้ เนื่องจากต้องมีการตั้งค่าอย่างอื่นร่วมด้วยในการเตือน โดยให้สามารถตั้งค่าได้จากการโหลดจากคอมพิวเตอร์เท่านั้น)
- สั่งเปลี่ยนชื่อพอร์ตต่างๆได้(ดิจิตอลขาออก 8 ช่อง ดิจิตอลขาเข้า 8 ช่อง แอนาลอก 2 ช่อง) เพื่อความสะดวกในการใช้งานโดยชื่อที่ตั้งขึ้นนี้สามารถใช้ในการอ้างอิงในการสั่งการได้
- สั่งการ enable หรือ disable การตรวจสอบความผิดพลาดจากช่องดิจิตอลขาเข้าทั้ง 8 ช่องได้ (ไม่สามารถสั่งการกับพอร์ตแอนาลอกกับการตรวจสอบเงื่อนไขได้)
- สามารถสั่ง reset ได้(ใช้ในกรณีที่ต้องการให้ระบบเริ่มต้นทำงานใหม่ หรือหลังจากที่

มีการแจ้งเตือนแล้วเนื่องจากเมื่อทำการแจ้งเตือนระบบจะยกเลิกการตรวจสอบในกรณี ที่ได้มีการแจ้งไปแล้วเพื่อป้องกันการส่งข่าวสารสั้นไม่รู้จบ)

4.3 คำสั่งที่มีใช้ในระบบ

จากที่ได้กล่าวมาแล้วข้างต้น ว่าระบบนี้ต้องการสามารถรับการสั่งการจากผู้ใช้ผ่านข่าวสารสั้นได้ ดังนั้นจึงต้องมีการกำหนดชุดคำสั่งในการดำเนินการซึ่งจะมีรายละเอียดดังแสดงต่อไป

เนื่องจากในระบบนี้สามารถมีการตั้งชื่อพอร์ตต่างๆได้อาจทำให้มีความสับสนในการใช้งานหลายคน จึงได้มีการกำหนดชื่อมาตรฐานของแต่พอร์ตไว้ด้วยเพื่อให้ผู้ใช้สามารถสั่งการได้โดยไม่จำเป็นที่ต้องรู้ชื่อที่ได้ตั้งไว้แต่หากผู้ใช้งานต้องการที่จะสั่งการโดยใช้ชื่อที่ตั้งไว้ก็ยังคงทำได้โดยรายชื่อมาตรฐานที่ได้กำหนดไว้มีดังนี้

	ภาค INPUT		ภาค OUTPUT
IN1	ดิจิตอลอินพุตช่องที่1	OUT1	ดิจิตอลเอาท์พุตช่องที่1
IN2	ดิจิตอลอินพุตช่องที่2	OUT2	ดิจิตอลเอาท์พุตช่องที่2
IN3	ดิจิตอลอินพุตช่องที่3	OUT3	ดิจิตอลเอาท์พุตช่องที่3
IN4	ดิจิตอลอินพุตช่องที่4	OUT4	ดิจิตอลเอาท์พุตช่องที่4
IN5	ดิจิตอลอินพุตช่องที่5	OUT5	ดิจิตอลเอาท์พุตช่องที่5
IN6	ดิจิตอลอินพุตช่องที่6	OUT6	ดิจิตอลเอาท์พุตช่องที่6
IN7	ดิจิตอลอินพุตช่องที่7	OUT7	ดิจิตอลเอาท์พุตช่องที่7
IN8	ดิจิตอลอินพุตช่องที่8	OUT8	ดิจิตอลเอาท์พุตช่องที่8

ANA1 แอนาลอกอินพุตช่องที่1

ANA2 แอนาลอกอินพุตช่องที่2

รายการคำสั่งที่ได้กำหนดไว้มีดังนี้

1. ON X ใช้ในการเปิดพอร์ตดิจิตอลขาออกโดย X คือชื่อพอร์ต(OUT1-OUT8) หรือจะใช้ชื่อที่ตั้งไว้ก็ได้

2. OFF X ใช้ในการปิดพอร์ตดิจิตอลขาออกโดย X คือชื่อพอร์ต(OUT1-OUT8) หรือจะใช้ชื่อที่ตั้งไว้ก็ได้

3. EN X ใช้ในการ enable การตรวจสอบจากพอร์ตดิจิตอลขาเข้าโดย X คือชื่อพอร์ต(IN1-IN8) หรือจะใช้ชื่อที่ตั้งไว้ก็ได้

4. DIS X ใช้ในการ disable การตรวจสอบจากพอร์ตดิจิตอลขาเข้าโดย X คือชื่อพอร์ต(IN1-IN8) หรือจะใช้ชื่อที่ตั้งไว้ก็ได้

5. NAME X Y ใช้ในการตั้งชื่อพอร์ตต่างโดย X คือชื่อพอร์ตเดิม(อ้างอิงจากชื่อมาตรฐานหรือใช้ชื่อที่ได้มีการตั้งไว้แล้วก็ได้) และ Y คือชื่อที่จะทำการตั้งใหม่

6. MESSAGE X Y ใช้ในการเปลี่ยนข่าวสารสั้นที่จะทำการส่งเมื่อเกิดความผิดพลาดที่พอร์ตดิจิตอลขาเข้า โดย X คือชื่อพอร์ต(IN1-IN8) หรือจะใช้ชื่อที่ตั้งไว้ก็ได้ และ Y คือข่าวสารสั้นที่ต้องการเปลี่ยนใหม่

7. RESET ใช้ในการสั่งให้ระบบทำการโหลดค่าต่างๆใหม่ จากหน่วยความจำ EEPROM และเริ่มทำการตรวจวัดใหม่(ใช้ในกรณีที่ RAU ได้มีการแจ้งเตือนไปแล้วและผู้ใช้ต้องการให้ RAU กลับมาตรวจวัดพอร์ตเดิมอีกครั้ง)

8. READ X ใช้ในการอ่านค่าที่กำลังทำการวัดอยู่โดย X มีค่าเป็น 'INPUT' เมื่อต้องการอ่านค่าสัญญาณขาเข้า หรือ 'OUTPUT' เมื่อต้องการอ่านค่าสัญญาณขาออก หรืออาจจะไม่มีก็ได้ซึ่งจะเป็นการอ่านค่าสัญญาณทั้งหมดแบบย่อ

ในการสั่งการนั้นจะแบ่งระดับของผู้ใช้เป็น 2 ระดับคือ ระดับที่สามารถอ่านค่าจาก RAU ได้เพียงอย่างเดียว และระดับที่สามารถสั่งการและปรับเปลี่ยนค่าต่างๆใน RAU ได้ด้วย โดยผู้ใช้ในระดับแรกจะสามารถใช้คำสั่งได้เพียงคำสั่งที่ 8 เท่านั้นแต่ผู้ใช้ในระดับหลังจะสามารถใช้คำสั่งได้ทุกคำสั่ง แต่ในกรณีการแจ้งเตือน RAU จะทำการแจ้งเตือนแก่ผู้ใช้งานทุกระดับ

4.4 การแสดงผล

การแสดงผล คือลักษณะของข่าวสารสั้นที่จะปรากฏในหน้าจอของโทรศัพท์มือถือในกรณีที่ RAU ได้มีการส่งข่าวสารสั้นไปให้ การแสดงผลมี 2 แบบคือ การแสดงผลในกรณีการเตือนและการแสดงผลในกรณีมีการสอบถามข้อมูลจากผู้ใช้งาน โดยการแสดงผลในกรณีการเตือนจะมีลักษณะเป็นข่าวสารสั้นธรรมดาความยาวไม่เกิน 25 ตัวอักษร 1 ประโยค แต่การแสดงผลในกรณีมีการสอบถามข้อมูลจะมีลักษณะดังต่อไปนี้

การแสดงผลบนหน้าจอเมื่อมีการใช้คำสั่ง READ

1. READ

การแสดงผล

INPUT xxxxxxxx

OUTPUT xxxxxxxx

A1 yyy.yy%

A2 yyy.yy%

โดย x คือ ค่าของแต่ละพอร์ตดิจิตอลโดยแสดงเป็นตัวเลขตัวเดียวคือ 1(ON) หรือ 0(OFF)
โดยแสดงค่าจากพอร์ต 1 ถึง 8 ไล่จากซ้ายไปขวา

y คือ ค่าแอนาลอกที่วัดได้แต่ละช่องแสดงเป็นเปอร์เซ็นต์

2. READ INPUT

การแสดงผล

1 xxxxxxxx ON/OFF
⋮
8 xxxxxxxx ON/OFF
A1 xxxxxxxx yyy.yy%
A2 xxxxxxxx yyy.yy%

โดย x คือ ชื่อของแต่ละพอร์ตที่ได้ทำการตั้งไว้(มากสุด 8 ตัวอักษร)โดยมี
ค่าสถานะเป็น ON หรือ OFF

y คือ ค่าแอนาลอกที่วัดได้เป็นเปอร์เซ็นต์

3. READ OUTPUT

การแสดงผล

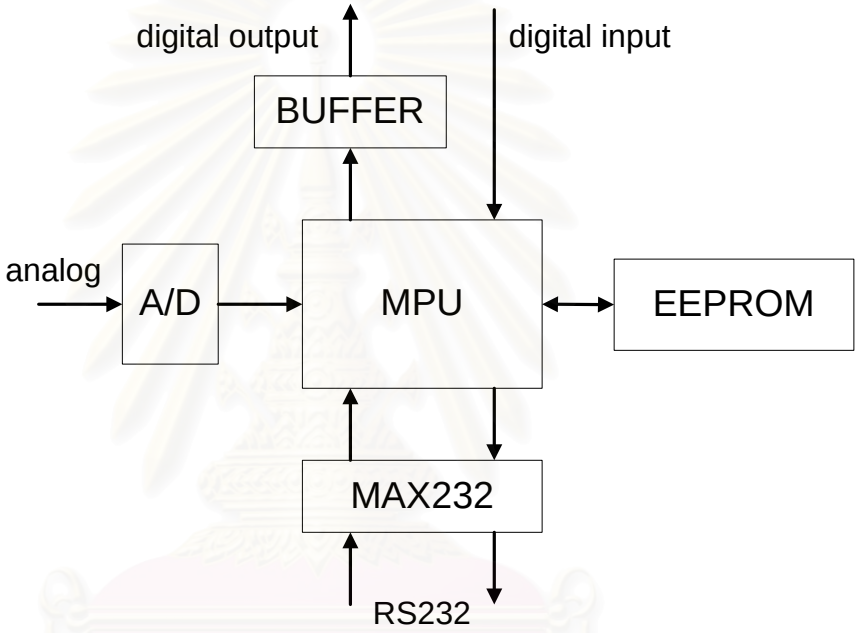
1 xxxxxxxx ON/OFF
⋮
8 xxxxxxxx ON/OFF

โดย x คือชื่อของแต่ละพอร์ตดิจิตอลขาออกที่ได้ทำการตั้งไว้(มากสุด 8 ตัวอักษร)โดยมี
ค่าสถานะเป็น ON หรือ OFF

บทที่ 5

โครงสร้างทางด้านฮาร์ดแวร์

ดังที่ได้กล่าวไปแล้วข้างต้นว่าระบบแจ้งเตือนและสั่งการทางไกลนี้ ประกอบไปด้วย 2 ส่วนคือ ส่วน RAU และส่วน user แต่โครงสร้างส่วนที่เป็นฮาร์ดแวร์นั้นจะมีอยู่แค่ในส่วน RAU เท่านั้นโดยจะมีโครงสร้างดังแสดงในรูป



รูปที่ 5.1 โครงสร้างทางฮาร์ดแวร์ของ RAU

จากรูปที่ 5.1 โครงสร้างของ RAU นี้จะประกอบไปด้วยส่วนประกอบดังนี้

- MPU หรือ microprocessor unit เป็นส่วนที่ใช้ในการควบคุมการทำงานของ RAU ซึ่งใช้ไมโครคอนโทรลเลอร์ในตระกูล MCS-51 เบอร์ P89C51RD2
- MAX 232 เป็นชิพที่ทำหน้าที่เป็น line driver ในการติดต่อสื่อสารแบบอนุกรม
- A/D เป็นชิพที่ทำหน้าที่ในการรับสัญญาณแอนาลอกเพื่อนำมาแปลงเป็นข้อมูลแบบดิจิตอล
- BUFFER เป็นส่วนที่ใช้ในการป้องกันสัญญาณขาออกในการนำสัญญาณไปควบคุมอุปกรณ์ภายนอก
- EEPROM เป็นหน่วยความจำที่สามารถลบและเขียนได้ด้วยไฟฟ้า ทำหน้าที่ในการเก็บข้อมูลที่ได้มีการตั้งจากผู้ใช้ ทั้งจากทางคอมพิวเตอร์และทางข่าวสารสั้น

5.1 รายละเอียดโครงสร้างในแต่ละส่วน

5.1.1 MPU ใช้ชิพไมโครคอนโทรลเลอร์ P89C51RD2 ซึ่งมีคุณสมบัติดังต่อไปนี้

- เป็นไมโครคอนโทรลเลอร์แบบ 8 บิต ที่เข้ากันได้กับไมโครคอนโทรลเลอร์ตระกูล MCS-51
- หน่วยความจำโปรแกรมภายในตัวเป็นแบบแฟลช ทำให้สามารถลบและเขียนใหม่ได้ถึง 10000 ครั้ง และมีขนาดของหน่วยความจำสูงถึง 64 KB
- มีหน่วยความจำแรมภายในขนาด 1 KB
- สามารถโปรแกรมข้อมูลลงในหน่วยความจำโปรแกรมแบบในวงจร หรือ แบบ In-System Programming(ISP)
- ทำงานโดยใช้สัญญาณนาฬิกา 6 ลูกต่อ 1 machine cycle
- ความเร็วของสัญญาณนาฬิกาสูงสุด 20 MHz
- มีพอร์ต 8 บิตจำนวน4 พอร์ต แบบกึ่ง 2 ทิศทาง
- มีวงจรสื่อสาร UART แบบ full-duplex ภายในจำนวน 1 ตัว
- สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ได้ 7 ประเภท
- กำหนดระดับความสำคัญของอินเตอร์รัปต์ได้ 4 ระดับ
- สามารถติดต่อหน่วยความจำภายนอกได้สูงสุด 64 KB
- มีวอตช์ด็อกไทเมอร์
- มีโมดูลวงจรนับแบบโปรแกรมได้(Programmable Counter Array: PCA) สำหรับการ capture/compare และการสร้างสัญญาณ PWM

5.1.2 MAX232 เป็นชิพ line driver สำหรับการติดต่อสื่อสารอนุกรมแบบอะซิงโครนัสสามารถให้บริการได้ 2 ช่องสัญญาณ โดยชิพจะรับสัญญาณจาก MPU ที่แรงดันอยู่ในช่วง 0-5 V แล้วทำการแปลงให้อยู่ในรูปสัญญาณช่วง –12 ถึง 12 V เพื่อทำการส่งให้อุปกรณ์ภายนอก และรับสัญญาณจากอุปกรณ์ภายนอกเพื่อทำการแปลงแรงดันและส่งให้ MPU

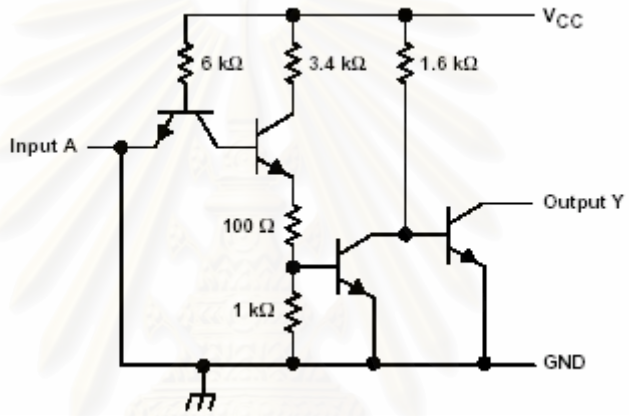
5.1.3 EEPROM เป็นชิพหน่วยความจำที่สามารถลบ และเขียนได้ด้วยไฟฟ้า โดยในงานวิจัยได้ใช้ชิพเบอร์ 24C256 ซึ่งรายละเอียดดังต่อไปนี้

- มีขนาดความจำ 32 KB
- ใช้การสื่อสารโดยใช้สายสัญญาณ 2 เส้นแบบ I^2C
- สามารถเขียนและอ่านข้อมูลแบบ page ได้มากสุด 64 ไบต์ต่อ 1 page
- มีการรับสัญญาณขาเข้าแบบชมิตทริกเกอร์เพื่อลดผลสัญญาณรบกวน
- สามารถเขียนและลบได้ 100,000 ครั้ง
- รองรับสัญญาณนาฬิกาได้สูงสุด 400KHz

5.1.4 A/D ใช้ชิพเบอร์ ADS7841 ซึ่งมีรายละเอียดดังต่อไปนี้

- มีความละเอียดในการแปลงข้อมูล 12 บิต
- ใช้วิธีการประมาณค่าสัญญาณแบบ successive approximation
- สามารถรองรับสัญญาณได้ 4 ช่องแบบ single ended หรือ 2 ช่องในแบบ differential โดยสามารถเลือกอ่านค่าในแต่ละช่องสัญญาณได้
- ใช้การสื่อสารในแบบอนุกรม 3 สาย(รายละเอียดจะได้กล่าวต่อไป)

5.1.5 Buffer ใช้ชิพเบอร์ 7407 ซึ่งเป็นบัฟเฟอร์แบบ open collector และไม่มีการกลับสัญญาณ ซึ่งโครงสร้างสามารถแสดงได้ดังรูปที่ 5.2



รูปที่ 5.2 โครงสร้างของชิพ 7407

5.2 การสื่อสารอนุกรมแบบอะซิงโครนัส

เนื่องจากในงานวิจัยนี้มีการทำงานที่เกี่ยวข้องกับการสื่อสารแบบอนุกรมอะซิงโครนัส โดยใช้ในการติดต่อข้อมูลกับโทรศัพท์มือถือเพื่อนำมาใช้งานเป็นอุปกรณ์ในการรับและส่งข่าวสารสั้น ดังนั้นจึงจะได้อธิบายถึงการสื่อสารในระบบนี้เพื่อความเข้าใจในการทำงานของระบบ

การสื่อสารแบบอนุกรมแบบอะซิงโครนัสคือการสื่อสารระหว่างอุปกรณ์ โดยไม่มีการใช้สัญญาณนาฬิการ่วมกันในการส่งและรับข้อมูล แต่จะมีการกำหนดอัตราการเปลี่ยนแปลงของข้อมูล(buad rate)ให้มีค่าเท่ากัน และจะมีข้อกำหนดในการติดต่อสื่อสารกันดังรูป 5.3

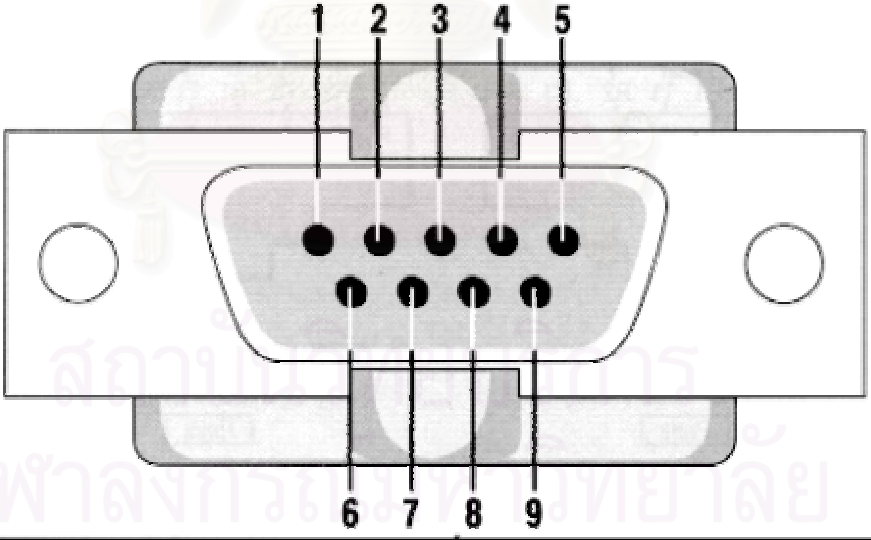
รูปที่ 5.3 การสื่อสารแบบอนุกรมอะซิงโครนัส

จากรูปที่ 5.3 ในการสื่อสารจะประกอบไปด้วยส่วนต่างๆดังนี้

- Idle State เป็นสภาวะในการหยุดรอเพื่อให้มีการเริ่มส่งข้อมูล
- Start Bit เป็นบิตที่ใช้ในการแสดงการเริ่มต้นการส่งข้อมูล
- Data Bit เป็นข้อมูลที่จะทำการส่งโดยการส่งนั้นจะทำการส่งจากบิตที่มีนัยสำคัญต่ำสุดก่อน(LSB) ไล่ไปจนถึงบิตที่มีนัยสูงสุด (MSB) โดยจำนวนบิตที่จะทำการส่งนี้สามารถกำหนดได้ และอยู่ในช่วง 5 ถึง 8 บิต
- Parity Bit เป็นบิตที่ใช้ในการตรวจสอบความถูกต้องของข้อมูลโดยตรวจสอบจากจำนวนบิตที่เป็น 1 ว่ามีจำนวนคู่(even) หรือคี่(odd) หรือจะเลือกไม่ทำการตรวจสอบก็ได้ โดยหากไม่ทำการตรวจสอบจะไม่มีการส่งค่าบิตนี้ในการสื่อสาร
- Stop Bit เป็นบิตที่ใช้ในการแสดงการจบการส่งข้อมูลโดยสามารถเลือกความกว้างได้เป็น 1,1.5 หรือ 2 บิต

ในการสื่อสารในระบบนี้ทั้งต้นทางและปลายทางจะต้องมีการกำหนดลักษณะการสื่อสารให้เหมือนกันดังนี้คือ Buad Rate,Data Bit,Pariy Bit และ Stop Bit

ในงานวิจัยนี้ได้ทำการติดต่อสื่อสารแบบอนุกรมอะซิงโครนัสโดยการใช้จุดเชื่อมต่อแบบ DB9 ซึ่งมีลักษณะดังแสดงในรูป 5.4



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

รูปที่ 5.4 รายละเอียดของพอร์ตแบบ DB9

รายละเอียดของแต่ละขาสามารถแสดงได้ดังนี้

1. Data Carrier Detect(DCD) เป็นขาที่ใช้ในการรับสัญญาณเชื่อมต่อจากโมเด็ม
2. Received Data(RXD) เป็นขาที่ใช้ในการรับข้อมูลเข้า
3. Transmitted Data(TXD) เป็นขาที่ใช้ในการส่งข้อมูลออก
4. Data Terminal Ready(DTR) เป็นขาที่ใช้ในการติดต่อกับอุปกรณ์ปลายทาง เพื่อแจ้งว่าต้องการจะติดต่อด้วย
5. Signal Ground(GND) เป็นสัญญาณ ground ของการติดต่อสื่อสาร
6. Data Set Ready(DSR) เป็นขาที่ใช้ในการรับการแจ้งว่าอุปกรณ์ปลายทาง ว่าต้องการติดต่อด้วย
7. Request to Send(RTS) เป็นขาที่ใช้ในการแจ้งให้อุปกรณ์ปลายทางทราบว่าต้องการส่งข้อมูลไปให้
8. Clear to Send(CTS) เป็นขาที่ใช้ในการรับแจ้งให้อุปกรณ์ที่ต้องการติดต่อทราบว่าพร้อมที่จะรับข้อมูลแล้ว
9. Ring Indicator(RI) เป็นขาที่ใช้ในการรับสัญญาณโทรศัพท์เข้า(ใช้งานในโมเด็ม)

แต่ในการใช้งานโดยทั่วไปแล้วจะมีเพียง 3 ขาที่มีการระบุการใช้งานอย่างชัดเจน คือขา TXD,RXD และ GND ซึ่งเป็นการกำหนดโดยฮาร์ดแวร์และไม่สามารถเปลี่ยนแปลงได้ แต่ขาอื่นในการใช้งานสามารถที่จะกำหนดความหมายและวิธีในการใช้งานได้ตามต้องการ

5.1.1 การใช้งานการสื่อสารอนุกรมอะซิงโครนัสใน MCS-51

ในการใช้งานการสื่อสารอนุกรมอะซิงโครนัสในไมโครคอนโทรลเลอร์ ตระกูล MCS-51 นั้นจะใช้การทำงานของส่วน UART(Universal Asynchronous Receiver) ในการควบคุม โดยการควบคุมผ่านหน่วยความจำพิเศษ(special function register) SCON โดยมีรายละเอียดดังต่อไปนี้

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

รูปที่ 5.5 รายละเอียดของ SCON

จากรูปที่ 5.5 รายละเอียดในบิตต่างๆสามารถแสดงได้ดังนี้

SM0 บิตเลือกโหมดการทำงาน

SM1 บิตเลือกโหมดการทำงาน

SM2	เป็นแฟลกกำหนดการทำงานแบบมัลติโพรเซสเซอร์
REN	แฟลกยอมให้มีการรับข้อมูลเข้า
TB8	ค่าของบิตที่ 9 ในการส่งข้อมูลออก
RB8	ค่าของบิตที่ 9 ในการรับข้อมูลเข้า
TI	แฟลกแสดงการอินเตอร์รัปต์ภายหลังการส่งข้อมูลเสร็จ 1 ชุด
RI	แฟลกแสดงการอินเตอร์รัปต์เมื่อรับข้อมูลเสร็จ 1 ชุด

การทำงานของ UART ในไมโครคอนโทรลเลอร์ตระกูล MCS-51 จะทำหน้าที่ในการส่งและรับข้อมูลอนุกรมแบบอะซิงโครนัสซึ่งเป็นแบบสองทิศทางสมบูรณ์(full duplex)ผ่านทางหน่วยความจำพิเศษ(special function register) SBUF ซึ่งเป็นหน่วยความจำแบบ 8 บิต โดยที่หน่วยความจำนี้จะมีสองส่วน คือส่วนที่ใช้ในการรับข้อมูลเข้าและส่งข้อมูลออกแยกจากกัน ในการทำงานนั้น UART จะเป็นตัวควบคุมในการรับข้อมูลเข้าและส่งข้อมูลออกที่ SBUF ทีละบิต โดยเริ่มจากบิตใหญ่สุดก่อนและสามารถทำการรับและส่งข้อมูลได้ในเวลาเดียวกัน เมื่อมีการรับข้อมูลครบ 1 ไบต์ก็จะมีการเซ็ตค่า RI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าการรับข้อมูลสำเร็จและเพื่อใช้ในการอินเทอร์รัปต์เพื่อการเขียนโปรแกรมในการรองรับการสื่อสารแบบอนุกรม และในการส่งข้อมูลนั้นเมื่อทำการส่งเสร็จก็จะมีการเซ็ตค่า TI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าการส่งข้อมูลสำเร็จและเพื่อใช้ในการอินเทอร์รัปต์เพื่อการเขียนโปรแกรมเช่นเดียวกัน การกำหนดโหมดการทำงานต่างๆในการสื่อสารแบบอนุกรมทำได้โดยการกำหนดค่าใน SM0 และ SM1 โดยมีรายละเอียดดังต่อไปนี้

SM0	SM1	
0	0	มีการทำงานเป็น shift register
0	1	8 บิต UART โดยสามารถกำหนดอัตราการเปลี่ยนแปลงข้อมูล(buad rate) ได้ การเลือกโหมดนี้จะไม่มีการใช้งานในส่วน parity bit ของการสื่อสารอนุกรมอะซิงโครนัส
1	0	9 บิต UART โดยไม่สามารถกำหนดอัตราการเปลี่ยนแปลงข้อมูล(buad rate) ได้ โดยอัตราการเปลี่ยนแปลงข้อมูลจะมีค่าคงที่เป็น $F_{osc}/64$ หรือ $F_{osc}/32$ โดยที่ F_{osc} คือความถี่ของสัญญาณนาฬิกาที่ใช้ในไมโครคอนโทรลเลอร์ การเลือกโหมดนี้จะมีการใช้งานในส่วน parity bit ของ การสื่อสารอนุกรมอะซิงโครนัส โดยในการรับข้อมูล pariy bit จะถูกเก็บไว้ใน RB8 และในการส่งข้อมูลจะต้องใส่ค่า pariy bit ลงใน

		TB8	
1	1		9 บิต UART โดยสามารถกำหนดอัตราการเปลี่ยนแปลงข้อมูล(buad rate)ได้ การทำงานในโหมดนี้จะเหมือนกับโหมด 2 แต่ต่างตรงที่ สามารถกำหนด buad rate ได้

5.1.2 การกำหนด buad rate ให้กับการสื่อสารอนุกรมอะซิงโครนัส

การกำหนด buad rate ให้กับการสื่อสารแบบอนุกรมอะซิงโครนัสสามารถทำได้โดยการกำหนดการทำงานให้กับ timer1 ในไมโครคอนโทรลเลอร์ โดยการควบคุมการทำงานของ timer1 ต้องทำการตั้งค่าผ่านรีจิสเตอร์ TCON ซึ่งสามารถแสดงได้ดังนี้

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

รูปที่ 5.6 รายละเอียดของ TCON

จากรูปที่ 5.6 มีรายละเอียดดังต่อไปนี้

TF1 แฟลกแสดงการนับเกิน(overflow) ของ Timer1

TR1 บิตควบคุมการทำงาน/หยุดทำงานของ Timer1

TF0 แฟลกแสดงการนับเกิน(overflow) ของ Timer0

TR0 บิตควบคุมการทำงาน/หยุดทำงานของ Timer0

IE1 แฟลกแสดงการอินเตอร์รัปต์ Timer1 จากภายนอก

IT1 บิตเลือกประเภทสัญญาณในการอินเตอร์รัปต์ Timer1

IE0 แฟลกแสดงการอินเตอร์รัปต์ Timer0 จากภายนอก

IT0 บิตเลือกประเภทสัญญาณในการอินเตอร์รัปต์ Timer0

โดยในการใช้งานสร้าง buad rate นั้นจะใช้แค่บิต TR1 เพื่อใช้ในการเริ่มและหยุด timer1 ในการสร้างสัญญาณนาฬิกาเท่านั้นโดยการกำหนดค่าเป็น 1 เพื่อเริ่มการทำงานและ 0 เพื่อหยุดการทำงาน นอกจากนี้ยังต้องมีการกำหนดโหมดการทำงานให้กับ timer1 ด้วยโดยการกำหนดค่าที่ register TMOD ซึ่งมีรายละเอียดดังนี้

GATE	C/T	M1	M0	GATE	C/T	M1	M0
		Timer 1				Timer 0	

รูปที่ 5.7 รายละเอียดของ TMOD

จากรูปที่ 5.7 มีรายละเอียดดังต่อไปนี้

GATE ใช้เลือกลักษณะการควบคุมการทำงานของ Timer โดยหากมีค่าเป็น 0 การเริ่ม และหยุดการทำงานจะทำด้วยบิต TR แต่ถ้ามีค่าเป็น 1 การเริ่มและหยุดการทำงาน ถูกควบคุมด้วยบิต TR และ การอินเตอร์รัปต์จากภายนอกที่ขา INT0 และ INT1 ของชิพ

C/T บิตเลือกการทำงานของ Timer โดยหากมีค่าเป็น 0 จะเป็นการทำงานแบบ timer หากมีค่าเป็น 1 จะเป็นการทำงานแบบ counter

MO1 บิตเลือกโหมดการทำงานของ Timer

MO0 บิตเลือกโหมดการทำงานของ Timer

โดยในรายละเอียดการเลือกโหมดการทำงานนี้ จะไม่ขอกล่าวถึงเพราะมีรายละเอียดค่อนข้างมาก และในการสร้าง buad rate นั้นจะใช้การทำงานในโหมดการนับ 8 บิต แบบโหลดค่าอัตโนมัติเพียงอย่างเดียว โดยการกำหนดค่า MO1 และ MO0 เป็น 1 และ 0 ตามลำดับ และการคำนวณหาอัตรา buad rate สามารถทำได้ดังต่อไปนี้

$$\text{Buad rate} = 2^{SMOD}/32 \times F_{osc}/(12\times[256-TH1])$$

โดยที่ SMOD เป็นค่าบิตภายในรีจิสเตอร์ PCON

TH1 เป็นค่าภายในรีจิสเตอร์ TH1 ซึ่งใช้สำหรับการโหลดค่าอัตโนมัติ

F_{osc} เป็นความถี่ของสัญญาณนาฬิกาที่ใช้

โดยในงานวิจัยนี้ได้เลือกใช้การสื่อสารอนุกรมอะซิงโครนัสที่มี buad rate 19200 ขนาดข้อมูล 8 บิต ไม่มีการตรวจสอบ parity และ stop bit ขนาด 1 บิต เนื่องจากโทรศัพท์ที่ใช้ในงานวิจัยมีลักษณะข้อกำหนดดังกล่าว

5.3 การสื่อสารกับหน่วยความจำ EEPROM

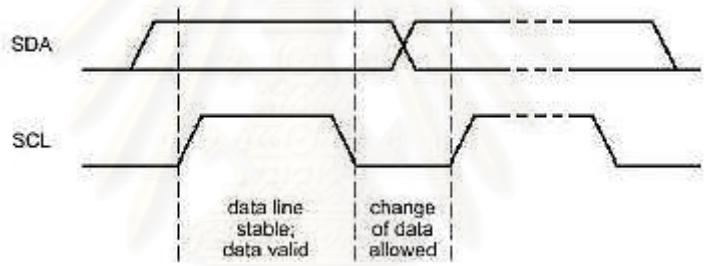
ในงานวิจัยนี้ได้มีการใช้การติดต่อสื่อสารแบบ I^2C เพื่อใช้ในการติดต่อระหว่าง MPU กับหน่วยความจำแบบ EEPROM ภายนอกจึงจะได้กล่าวถึงรายละเอียดของการสื่อสารแบบ I^2C ดังนี้

การสื่อสารแบบ I^2C เป็นการสื่อสารแบบกึ่ง 2 ทิศทางแบบอนุกรมสองทิศทางโดยอาศัยสายสัญญาณ 2 สายโดยมีสายหนึ่งทำหน้าที่เป็นสายสำหรับส่งข้อมูล(SDA)ทั้งไปและกลับ

และอีกสายทำหน้าที่ในการส่งสัญญาณนาฬิกา(SCL) การสื่อสารในระบบนี้มีข้อดีคือสามารถต่อพ่วงอุปกรณ์หลายๆตัวที่สามารถติดต่อแบบ I^2C โดยการใช้สายสัญญาณเดียวกันได้ โดยแต่ละตัวจะมีค่าตำแหน่ง(address)ประจำตัวที่ไม่เหมือนกัน เพื่อการจำแนกการอ่านและเขียนข้อมูลไปยังอุปกรณ์ต่างๆ ทำให้มีความสะดวกต่อการออกแบบและลดขนาดของลายวงจรได้ การสื่อสารแบบนี้มีข้อกำหนดดังต่อไปนี้

5.3.1 Bit Transfer

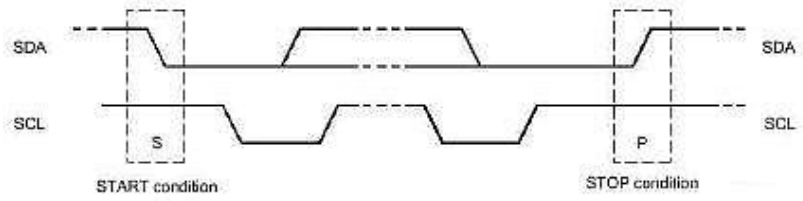
ในการสื่อสารแบบ I^2C นี้ข้อมูล 1 ไบต์(8 บิต) จะถูกส่งเรียงกันไปทีละบิตในสายสัญญาณโดยการเริ่มส่งจากบิตที่มีนัยสูงสุด(MSB)ก่อน จากนั้นจึงตามด้วยบิตที่มีนัยต่ำกว่าจนครบ 1 ไบต์ การส่งข้อมูลแต่ละบิตจะทำโดยอาศัยสายสัญญาณ SDA โดยใช้สัญญาณนาฬิกา 1 ลูก จาก SCL ในการอ้างอิง และในขณะที่ทำการส่งข้อมูลบิตนั้นอยู่สายสัญญาณ SCL จะต้องมีค่าคงที่ไม่มีการเปลี่ยนแปลงซึ่งสามารถแสดงได้ดังรูป 5.8



รูปที่ 5.8 การส่งข้อมูลแบบ I^2C

5.3.2 Start and Stop Condition

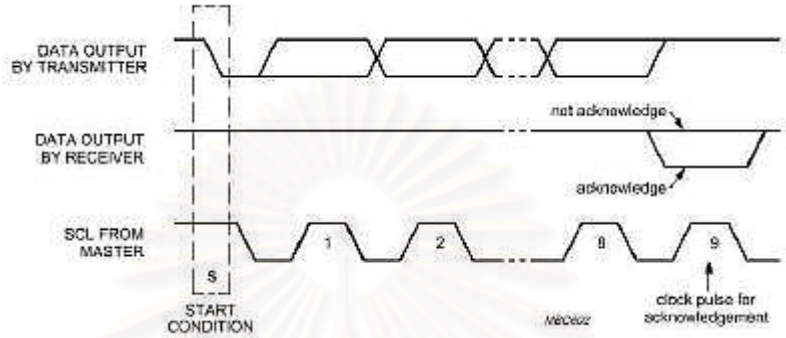
ทั้งสายสัญญาณ SDA และสายสัญญาณ SCL ถ้าอยู่ในสภาวะไม่มีการติดต่อสื่อสารจะมีค่าเป็น 1 การเปลี่ยนแปลงค่าจาก 1 เป็น 0 ของสายสัญญาณ SDA ในขณะที่สายสัญญาณ SCL มีค่าเป็น 1 เป็นการกำหนดเงื่อนไขการเริ่มส่งข้อมูล และการเปลี่ยนแปลงค่าจาก 0 เป็น 1 ของสายสัญญาณ SDA ในขณะที่สายสัญญาณ SCL เป็น 1 เป็นการกำหนดเงื่อนไขการสิ้นสุดการส่งข้อมูลดังแสดงในรูป 5.9



รูปที่ 5.9 การกำหนดเงื่อนไขเริ่มและหยุดส่งข้อมูลแบบ I^2C

5.3.3 Acknowledge

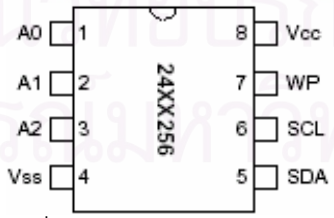
ในการติดต่อแบบ I^2C นั้น เมื่อทำการเขียนหรือข้อมูลเสร็จ 1 ไบต์จะต้องมีการตอบรับโดยทำการส่งบิต ACK ซึ่งมีค่าเป็น 0 กลับไปดังแสดงในรูป 5.10



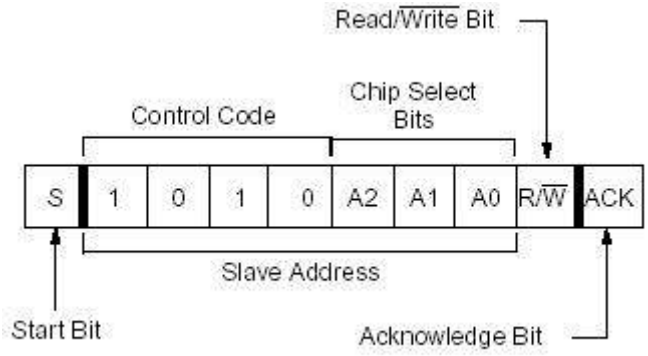
รูปที่ 5.10 การตอบรับในการสื่อสารแบบ I^2C

5.3.4 การเขียนและอ่าน EEPROM

ในการเขียนและอ่านอุปกรณ์ที่มีการสื่อสารแบบ I^2C นั้นจะต้องมีการส่งข้อมูลส่วนควบคุม(control byte) ออกไปก่อนเพื่อใช้สั่งการว่าต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์ตัวใด(ในcontrol byte จะระบุ addressของอุปกรณ์ไว้) แล้วจึงทำการส่ง address ของส่วนที่ต้องการอ่านหรือเขียนของอุปกรณ์นั้น จากนั้นจึงทำการส่งหรือรับข้อมูลจากอุปกรณ์นั้น โดยการรับข้อมูลนี้อาจทำทีละไบต์ หรือทำต่อเนื่องเป็นหน้า(page)ได้โดยต้องไม่เกินจำนวนไบต์ที่อุปกรณ์นั้นจะรองรับได้ การใช้งาน EEPROM ที่ใช้ในงานวิจัยนี้ใช้เบอร์ 24C256 (รูปที่ 5.11)โดยมีส่วนควบคุมอยู่ 1 ไบต์ซึ่งมีรายละเอียดดังรูปที่ 5.12



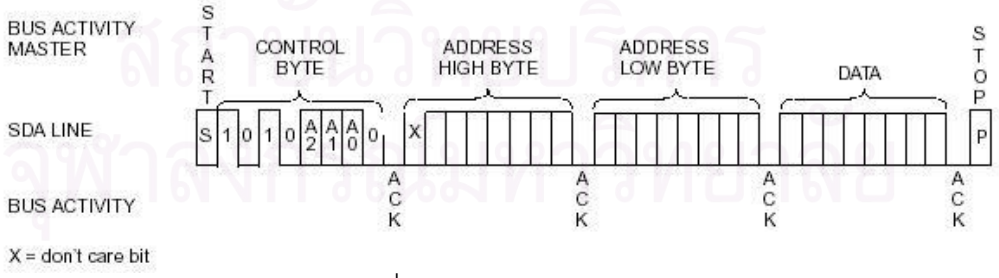
รูปที่ 5.11 EEPROM 24C256



รูปที่ 5.12 ส่วน Control Byte ของ 24C256

โดยอุปกรณ์ตัวนี้มีตำแหน่ง address เป็น 1010 ใน 4 บิตบน และ 3 บิตถัดมาเป็นตำแหน่งแบบเลือกได้จากการกำหนดค่าให้ขา A2 ถึง A0 ของตัวชิพ 3 บิตนี้มีประโยชน์ในการต่อขยายหน่วยความจำโดยสามารถขยายชิพหน่วยความจำเพิ่มได้อีก 8 ตัวโดยอาศัยสายสัญญาณเดียวกันและใช้การกำหนดค่าที่ A2 ถึง A0 เพื่อใช้เป็นตำแหน่ง address ของชิพแต่ละตัว(รวมกับ 4 บิตบนที่มีค่า 1010) และบิตต่ำสุดจะเป็นตัวระบุการเขียนหรืออ่าน โดยหากมีค่าเป็น 0 หมายถึงทำการเขียน หากเป็น 1 หมายถึงการอ่าน ในการอ่านและเขียน 24C256 นี้แบ่งเป็น 5 วิธีดังนี้

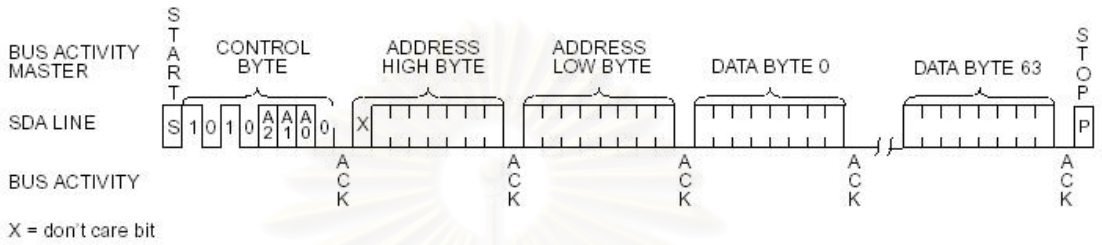
- *การเขียนแบบไบต์* เป็นการเขียนข้อมูลทีละไบต์โดยการส่งเงื่อนไขการเริ่มและ control byte ให้กับชิพและรอการ ACK กลับมาแล้วจึงทำการส่ง address ของข้อมูลไปซึ่งมีขนาด 2 ไบต์(สำหรับ 24C256) โดยชิพจะมีการ ACK กลับมาทีละไบต์ จากนั้นจึงทำการส่งข้อมูลให้ชิพและรอการ ACK กลับเพื่อส่งเงื่อนไขในการหยุดให้ชิพซึ่งสามารถแสดงได้ดังรูปที่ 5.13



รูปที่ 5.13 การเขียน 24C256 แบบไบต์

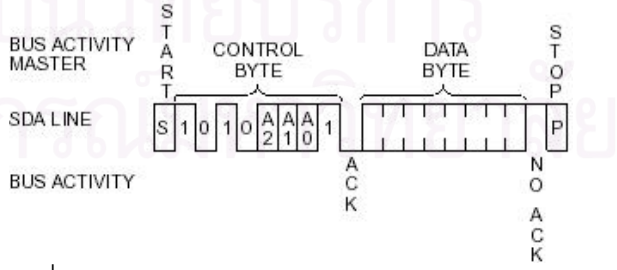
- *การเขียนแบบเพจ* เป็นการเขียนข้อมูลทีละหลายๆไบต์โดยเงื่อนไขในการเริ่มทำการเขียนจะเหมือนกับการเขียนแบบไบต์แต่หลังจากที่ได้ทำการเขียนข้อมูลไปแล้วและชิพทำการ ACK กลับมาแล้วไม่ต้องส่งเงื่อนไขในการหยุดแต่ส่งข้อมูลไบต์

ถัดไปให้กับชิพ ชิพจะทำการบันทึกโดยเพิ่มค่า addrees ให้กับข้อมูลโดยอัตโนมัติ และเมื่อชิพทำการ ACK กลับมาอีกก็สามารถที่จะส่งข้อมูลที่จะเขียนไบต์ต่อไปอีกได้ แต่ห้ามเกินความสามารถในการเขียนแบบเพจของชิพ สำหรับ 24C256 นั้นสามารถทำการเขียนแบบเพจได้มากสุด 64 ไบต์ ซึ่งรายละเอียดในการทำงานสามารถแสดงได้ดังรูปที่ 5.14



รูปที่ 5.14 การเขียน 24C256 แบบเพจ

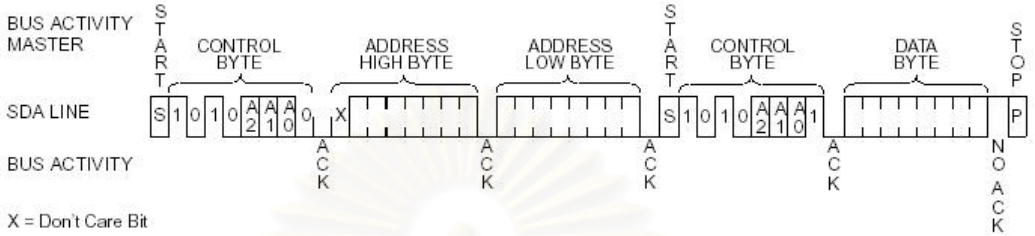
- *การอ่านแบบตำแหน่งปัจจุบัน* ใน 24C256 จะมีตัวชี้ตำแหน่งของหน่วยความจำโดยทุกครั้งที่เราทำการส่ง address ให้กับชิพ ตัวชี้ตำแหน่งนี้จะเปลี่ยนแปลงตามค่าที่ส่งไปและหากมีการดำเนินการแบบเพจตัวชี้ตำแหน่งก็จะเพิ่มค่าขึ้นทีละ 1 ตามจำนวนข้อมูลที่ได้เขียนหรืออ่าน การอ่านแบบตำแหน่งปัจจุบันนี้เป็นการอ่านข้อมูลในตำแหน่งที่ตัวชี้ตำแหน่งทำการชี้อยู่ ซึ่งการอ่านต้องเริ่มด้วยการส่งเงื่อนไขในการเริ่มและส่ง control byte ในการอ่าน และรอการ ACK จากชิพจากนั้นจึงทำการอ่านข้อมูล เมื่ออ่านข้อมูลเสร็จทำการส่งเงื่อนไข NO ACK(ลอจิก 1) กลับไปที่ชิพและส่งเงื่อนไขในการหยุดให้ชิพเพื่อเป็นการจบการอ่าน โดยสามารถแสดงได้ดังรูปที่ 5.15



รูปที่ 5.15 การอ่าน 24C256 แบบตำแหน่งปัจจุบัน

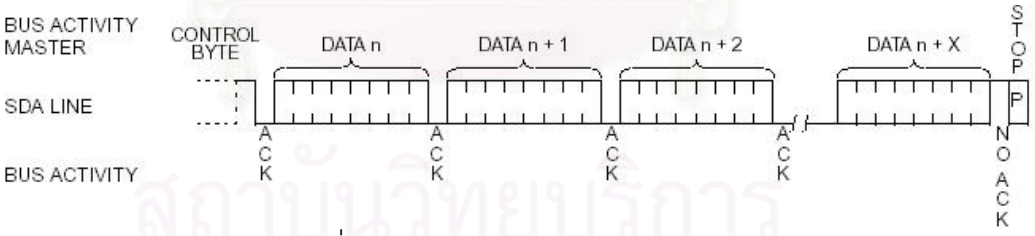
- *การอ่านแบบสุ่ม* การอ่านแบบสุ่มเป็นการอ่านข้อมูลในตำแหน่งใดก็ได้ของชิพ โดยจะมีลักษณะคล้ายกับการอ่านแบบตำแหน่งปัจจุบัน แต่ต้องมีการกำหนด

address ที่จะอ่านให้กับตัวชิพก่อน โดยการใช้คำสั่งในการเขียนเขียนค่า address ลงในตัวชี้ตำแหน่งของชิพก่อนแล้วจึงค่อยส่งเงื่อนไขในการเริ่มเพื่อทำการอ่านและเมื่ออ่านข้อมูลครบแล้วจึงทำการส่งเงื่อนไข NO ACK และเงื่อนไขหยุด โดยมีรายละเอียดดังแสดงในรูปที่ 5.16



รูปที่ 5.16 การอ่าน 24C256 แบบสุ่ม

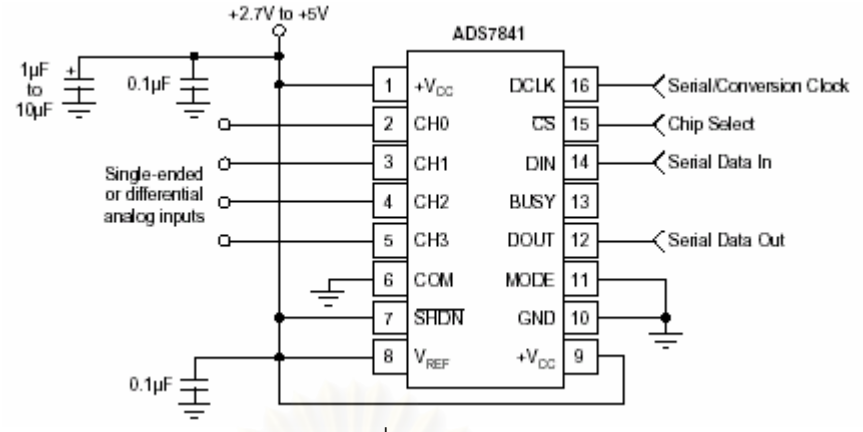
- *การอ่านแบบเป็นลำดับ* เป็นการอ่านที่มีลักษณะในการอ่านเหมือนกับแบบสุ่มแต่ต่างตรงที่ไม่มีการส่งเงื่อนไข NO ACK เมื่ออ่านข้อมูลจบ 1 ไบต์แต่จะทำการส่งเงื่อนไข ACK เพื่อทำการอ่านข้อมูลในไบต์ถัดไป โดยตัวชี้ตำแหน่งจะทำการเพิ่มค่าโดยอัตโนมัติทีละ 1 โดยการอ่านแบบลำดับนี้จะไม่สามารถอ่านข้อมูลได้เกินความสามารถของชิพที่จะรองรับได้ โดย 24C256 สามารถอ่านข้อมูลแบบลำดับได้มากสุด 64 ไบต์ เมื่อต้องการที่จะสิ้นสุดการอ่านให้ส่งเงื่อนไข NO ACK และเงื่อนไขหยุดเพื่อจบการทำงาน โดยรายละเอียดสามารถแสดงได้ดังรูป 5.17



รูปที่ 5.17 การอ่าน 24C256 แบบลำดับ

5.4 การสื่อสารกับอุปกรณ์ A/D

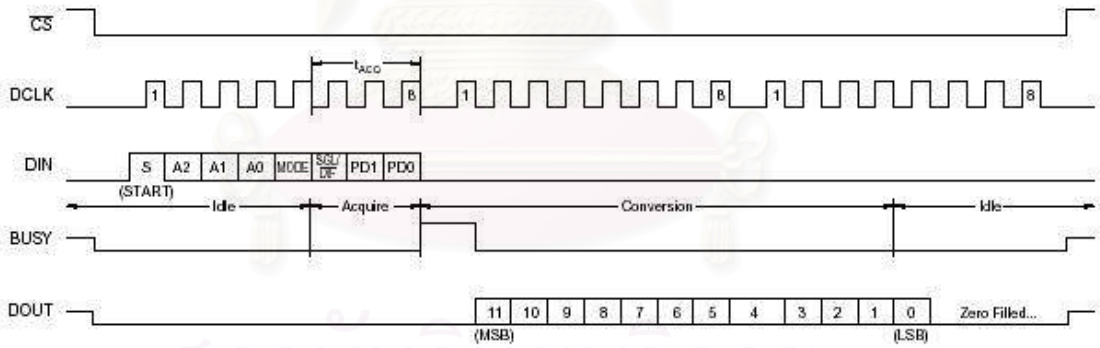
ในงานวิจัยนี้ได้ใช้ชิพ ADS7841 ในการแปลงสัญญาณแอนาลอกเป็นข้อมูลแบบดิจิตอล โดยใช้วิธีการประมาณแบบ Successive Approximation Resister (SAR) ซึ่งตัวชิพสามารถแสดงได้ดังรูปที่ 5.18



รูปที่ 5.18 ชิพ ADS7841

โดยแรงดันอ้างอิงที่ป้อนให้กับ ADS7841 สามารถเลือกได้ตั้งแต่ 100mV จนถึง +Vcc ชิพสามารถรับสัญญาณในการวัดได้ 4 ช่องและสามารถทำได้ทั้งแบบ Single-Ended หรือแบบ differential โดยการใช้ช่องสัญญาณ 2 ช่องร่วมกัน โดยการเลือกการอ่านสัญญาณช่องใดสามารถทำได้โดยการเลือกใน control byte ที่ส่งให้กับชิพ

การเชื่อมต่อข้อมูลกับตัวชิพใช้วิธีการสื่อสารแบบอนุกรมสองทิศทางซึ่งสามารถแสดงได้ดังรูปที่ 5.19

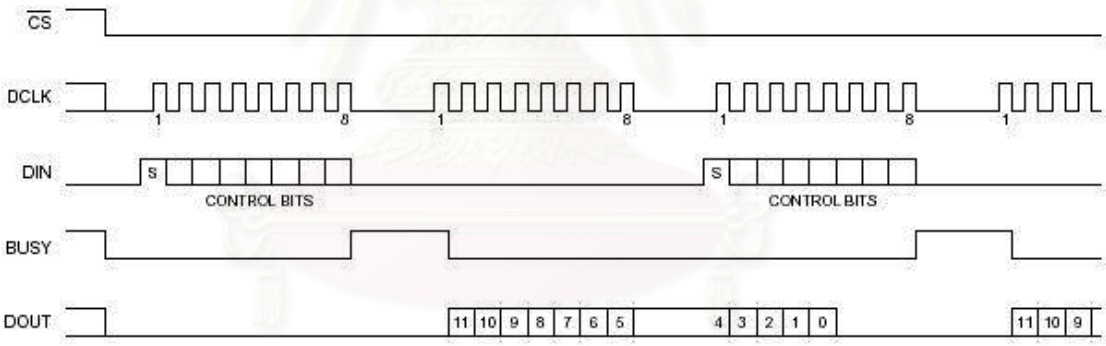


รูปที่ 5.19 การสื่อสารแบบอนุกรมของ ADS7841

จากรูปที่ 5.19 การสื่อสารกับ ADS7841 นั้นจะต้องทำการสั่งตัวชิพให้ทำงานก่อนโดยการเลือกผ่านขาสัญญาณ CS แล้วจึงทำการติดต่อโดยการเริ่มส่ง control byte ไปก่อนทางขา DIN เพื่อเลือกการทำงานและเลือกช่องสัญญาณที่ต้องการ จากนั้นชิพจะส่งข้อมูลกลับมาทางขา DOUT โดยจะเริ่มส่งจากบิตที่มีนัยสำคัญมากสุดก่อน(MSB) โดยข้อมูลมีจำนวน 12 บิต แล้วตามด้วย zero field ที่มีขนาด 4 บิต ซึ่งรวมแล้วในการติดต่อ 1 ครั้งจะต้องใช้ข้อมูลทั้งหมด 24 บิตหรือ 3 ไบต์(ทั้งไปและกลับ) โดยในส่วน control byte นั้นมีรายละเอียดดังนี้

- S เป็นส่วนที่ใช้เริ่มการติดต่อ(start) โดยเมื่อต้องการติดต่อ S จะมีค่าเป็น 1
- A2 A1 A0 เป็นส่วนที่ใช้เลือกช่องสัญญาณในการวัดทั้งแบบ Single-Ended และ differential โดยจะมีรายละเอียดต่างกันไปในการใช้งานแบบ Single-Ended และแบบ differential
- Mode เป็นส่วนที่ใช้ในการเลือกการแปลงข้อมูลว่าต้องการแบบ 12 บิตหรือ 8 บิต
- SGL/DIF เป็นส่วนที่ใช้ในการเลือกว่าเป็นสัญญาณแบบ Single-Ended หรือแบบ Differential และมีผลต่อการกำหนดค่า A2 A1 A0
- PD1 PD0 เป็นส่วนที่ใช้ในการเลือกการทำงาน power down ของตัวชิพ

นอกจากนี้การอ่านค่าจากชิพสามารถทำได้โดยไม่ต้องรอให้ครบ 24 สัญญาณนาฬิกา โดยเริ่มจากการส่ง control byte ให้กับชิพและหลังจากที่ทำการอ่านข้อมูล 8 บิตแรกเสร็จให้ส่ง control byte ให้กับตัวชิพต่อเพื่อให้ชิพทำการแปลงสัญญาณครั้งต่อไปในขณะที่ยังทำการส่งข้อมูลอีก 4 บิตที่เหลืออยู่ ซึ่งจะทำให้มีการแปลงข้อมูลทุก 16 สัญญาณนาฬิกา และสามารถแสดงได้ดังรูปที่ 5.20



รูปที่ 5.20 การอ่านค่าจาก ADS7841 ทุก 16 สัญญาณนาฬิกา

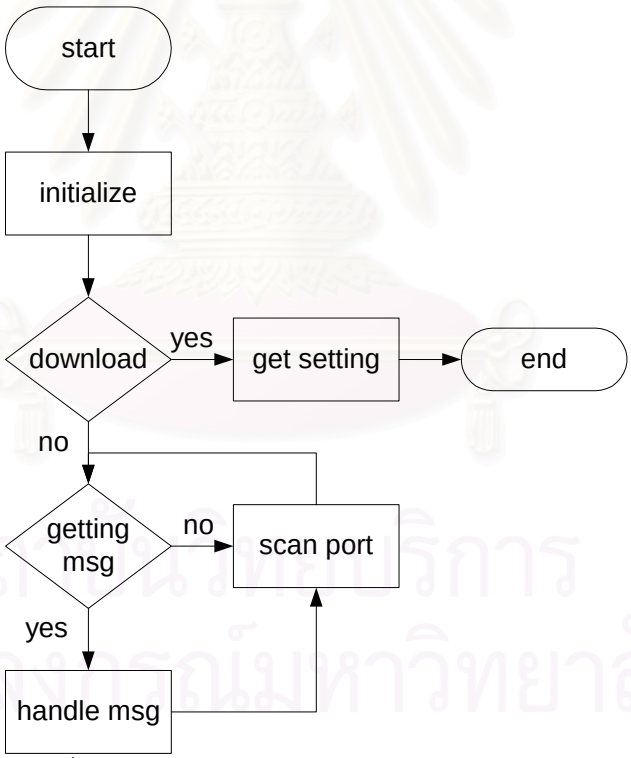
บทที่ 6

โครงสร้างทางซอฟท์แวร์ของระบบ

ในงานวิจัยนี้โครงสร้างทางซอฟท์แวร์จะประกอบไปด้วย 2 ส่วนคือส่วนที่เป็นส่วนควบคุมใน RAU และส่วนที่เป็นโปรแกรมซึ่งทำงานในคอมพิวเตอร์

6.1 โครงสร้างทางซอฟท์แวร์ใน RAU

ในระบบแจ้งเตือนและสั่งการทางไกลผ่านข่าวสารสั้นนี้ตัว RAU จะทำหน้าที่หลักอยู่ 3 อย่างคือการรับการโหลดค่าจากคอมพิวเตอร์เพื่อการตั้งค่าต่างๆใน RAU การตรวจจับความผิดพลาดจากสัญญาณที่ได้มีการป้อนเข้ามาที่ RAU และรับคำสั่งจากผู้ใช้งานเพื่อนำมาดำเนินการต่างๆ ซึ่งโครงสร้างทางโปรแกรมของ RAU สามารถแสดงได้ดังรูปที่ 6.1



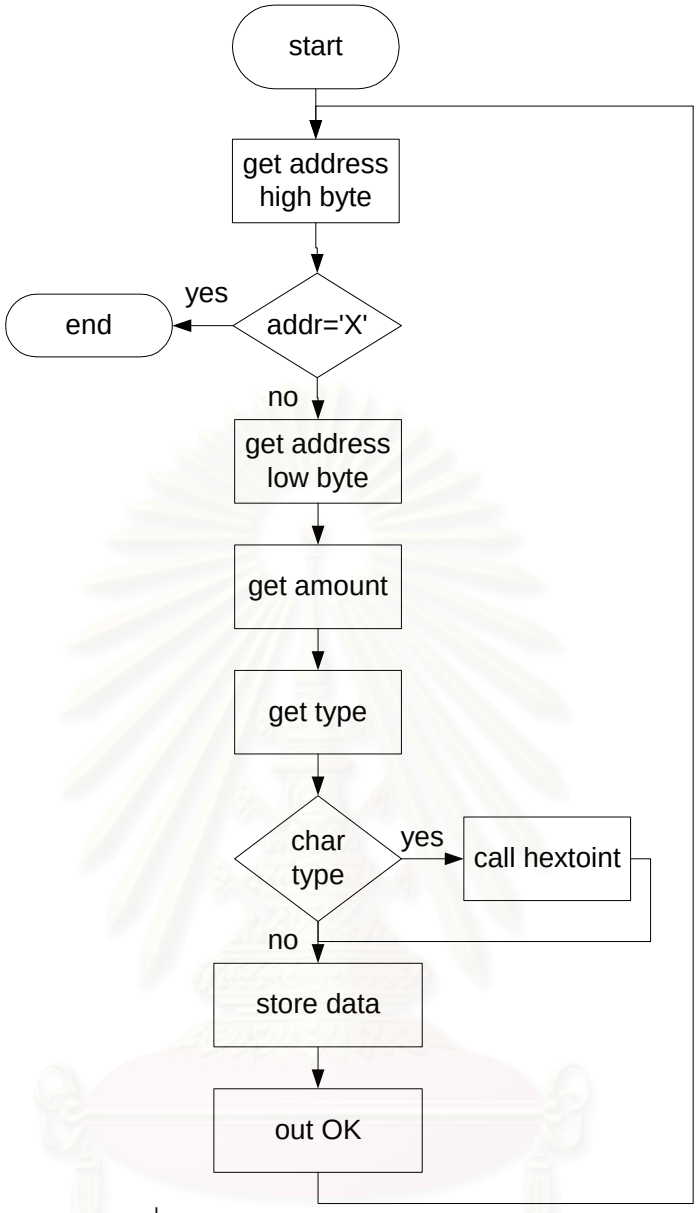
รูปที่ 6.1 โครงสร้างทางโปรแกรมของ RAU โดยรวม

จากรูปที่ 6.1 จะเห็นได้ว่าเมื่อ RAU ทำงานจะผ่านขั้นตอนการ initialize เพื่อเตรียมพร้อมในการทำงานแล้ว จะทำการตรวจสอบการโหลดค่าจากคอมพิวเตอร์ หากไม่มีการโหลดค่าก็จะเริ่มการทำงาน โดยการตรวจสอบว่ามีข่าวสารสั้นเข้ามาหรือไม่หากมีก็จะทำการ

จัดการเกี่ยวกับข่าวสารสั้นนั้นแล้วจึงกลับไปตรวจสอบความผิดพลาดของสัญญาณที่ป้อนเข้ามา หากไม่มีข่าวสารสั้นเข้าก็จะทำการตรวจสอบความผิดพลาดเลย และการทำงานทั้งสองอย่างนี้จะมีการทำสลับกันไปเรื่อยๆ ซึ่งรายละเอียดการทำงานในแต่ละส่วนของ RAU จะได้กล่าวถึงต่อไป โดยจะแบ่งเป็นส่วน get setting, handle message และ scan port

6.1.1 โครงสร้างการรับข้อมูลจากคอมพิวเตอร์ของ RAU

เมื่อทำการ initialize เสร็จ RAU จะเริ่มการสื่อสารโดยการให้ข้อความออกทางพอร์ต RS232 เป็น ATE0 เพื่อใช้ในการเริ่มการติดต่อกับโทรศัพท์ในกรณีนำ RAU ไปใช้งาน โดยคำสั่งนี้เป็นคำสั่งที่ใช้ในการยกเลิกการทวนคำสั่งของโทรศัพท์ (การส่งตัวอักษรที่ได้รับกลับมาที่ RAU เพื่อเป็นการยืนยัน) หาก RAU ทำการเชื่อมต่อข้อมูลอยู่กับโทรศัพท์อยู่ ทางโทรศัพท์ก็จะทำการตอบรับกลับมาว่า OK เมื่อ RAU ได้รับการตอบกลับก็จะเริ่มการทำงานโดยการตรวจสอบข่าวสารสั้นเข้าและตรวจความผิดพลาดที่เกิดขึ้นจากสัญญาณที่ถูกป้อนเข้า แต่หาก RAU ทำการเชื่อมต่อข้อมูลกับคอมพิวเตอร์ที่มีการเปิดโปรแกรมสำหรับโหลดข้อมูลอยู่ ทางโปรแกรมจะทำการส่งค่าตัวอักษร X มาให้ RAU เพื่อเป็นการแจ้งให้ทราบว่าจะทำการโหลดข้อมูลลง RAU และ RAU จะทำการจัดการในการรับข้อมูลที่จะบันทึก โดยการทำงานในส่วนการจัดการการรับข้อมูลของ RAU สามารถแสดงได้ดังรูปที่ 6.2



รูปที่ 6.2 โครงสร้างส่วน get setting ของ RAU

จากรูปที่ 6.2 ขั้นตอนต่างๆสามารถอธิบายได้ดังนี้

- get address high byte หลังจากที่เริ่มทำการโหลดค่า RAU จะทำการรับข้อมูลตำแหน่งเพื่อใช้ในการอ้างอิงการบันทึกข้อมูลลงในหน่วยความจำ EEPROM แต่เนื่องจาก EEPROM ที่ใช้ในงานวิจัยมีขนาดความจุ 32 KB ดังนั้นในการอ้างอิงตำแหน่งจึงต้องมีการใช้ค่าตำแหน่งด้วยข้อมูล 2 ไบต์ และการรับค่าตำแหน่ง address high byte นี้ยังเป็นการตรวจสอบการสิ้นสุดการรับข้อมูลด้วย โดยหากค่าที่รับได้เป็นตัวอักษร X ก็จะทำการสิ้นสุดการทำงานในส่วนการรับข้อมูลเพื่อบันทึก
- get address low byte เป็นการรับข้อมูลตำแหน่งอีก 1 ไบต์ที่เหลือ

- get amount เป็นการรับจำนวนข้อมูลที่ต้องการให้ RAU รับไปบันทึก
- get type เป็นการรับประเภทของข้อมูลที่ถูกส่งมาให้ RAU โดยประเภทของข้อมูลที่ถูกส่งให้ RAU นี้จะมี 2 ประเภทคือ ตัวอักษร(ASCII) ซึ่งใช้ในส่วนข่าวสารสั้นที่จะทำการแจ้งเมื่อเกิดความผิดพลาด ชื่อของพอร์ตต่างๆที่ต้องการกำหนด และเลขหมายของผู้ที่ใช้งานในระบบและอีกส่วนจะเป็นข้อมูลที่ไม่ใช่ตัวอักษร(integer) ซึ่งเป็นค่าที่ใช้ในการตั้งการทำงานในส่วนต่างๆของ RAU เช่น การตั้ง enable/disable พอร์ตต่างๆ การตั้งเลขหมายที่ต้องการแจ้งข่าวสารสั้นไปเมื่อเกิดความผิดพลาดขึ้น หรือการตั้งค่าระดับสัญญาณแอนาลอกที่จะทำการตรวจสอบและอื่นๆ ซึ่งรายละเอียดของข้อมูลที่จะทำการบันทึกนี้จะได้กล่าวถึงในส่วนต่อไป โดยข้อมูลแบบ integer นี้ในการส่งให้กับ RAU คอมพิวเตอร์จะทำการแปลงให้อยู่ในรูปตัวอักษรที่ใช้แสดงเลขฐาน 16 หาก RAU ได้รับการแจ้งว่าข้อมูลเป็นแบบ integer ก็จะทำการบันทึกค่าใน buffer แล้วทำการเรียกฟังชัน hextoint ขึ้นมาเพื่อทำการแปลงข้อมูลกลับเป็นแบบ integer เพื่อทำการบันทึกใน EEPROM ต่อไป
- store data เป็นการทำงานในการบันทึกค่าของข้อมูลที่ได้รับมาลงใน EEPROM โดยใช้ตำแหน่งเริ่มจาก address high byte และ address low byte รวมกัน และจะทำการเขียนข้อมูลแบบหน้า(page) ลงใน EEPROM โดยมีจำนวนข้อมูลใน page เท่ากับจำนวนที่ระบุมาใน amount
- out OK เป็นการแจ้งให้ คอมพิวเตอร์ทราบว่าข้อมูลที่ได้รับมาถูกบันทึกเรียบร้อยแล้วและพร้อมที่จะรับข้อมูลชุดใหม่ หากข้อมูลที่จะทำการส่งมาให้ RAU ยังไม่หมดคอมพิวเตอร์ก็จะเริ่มทำการส่งข้อมูลชุดถัดไปโดยมีวิธีการดังที่ได้กล่าวมาแล้วด้านบน แต่หากข้อมูลถูกส่งมาหมดแล้วก็จะทำการส่งอักษร X มาแทน address high byte เพื่อเป็นการแจ้งให้ RAU ทราบว่าสิ้นสุดการบันทึก RAU จะเข้าสู่สภาวะไม่ทำงานใดๆ หากต้องการให้ RAU เริ่มทำงานอีกครั้งสามารถทำได้โดยการกดปุ่ม reset บนตัว RAU เพื่อให้เริ่มทำงานใหม่อีกครั้ง โดย RAU จะเริ่มทำงานตามรูปที่ 6.1

6.1.2 รายละเอียดของข้อมูลที่มีการบันทึกใน RAU

จากหัวข้อที่ 6.1.1 นั้นได้กล่าวถึงการบันทึกค่าลงใน RAU ซึ่งค่าต่างๆที่มีการบันทึกและใช้ในการทำงานของ RAU นั้นสามารถแสดงได้ดังนี้

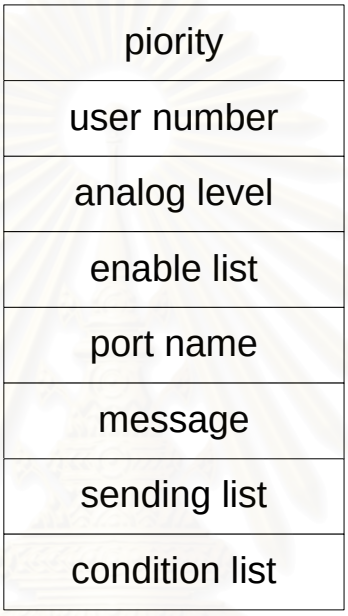
- user number เป็นชุดตัวอักษรที่แสดงเลขหมายของผู้ใช้งานในระบบ โดยมีขนาด 8 ตัวอักษร และมีทั้งหมด 10 ชุด(มีผู้ใช้งานระบบได้ 10 เลขหมาย)

- piority เป็นไบต์ที่ทำการบันทึกความสำคัญของเลขหมายผู้ใช้งานในระบบว่าเลขหมายใดมีสิทธิในการสั่งการ RAU ได้บ้าง โดยจะมีทั้งหมด 2 ไบต์ ในการเก็บค่าแต่ละไบต์จะแบ่งเก็บค่าความสำคัญไบต์ละ 5 เลขหมาย โดยใช้บิต 0 ถึง 4 แสดงค่าความสำคัญของเลขหมาย 1-5 และ 6 -10 โดยหากมีค่าเป็น 1 แสดงว่าสามารถสั่งการ RAU ได้ หากเป็น 0 แสดงว่าไม่สามารถสั่งการ RAU ได้
- enable list เป็นไบต์ที่เก็บค่าว่าต้องการให้มีการตรวจสอบช่องสัญญาณในช่องใดบ้าง โดยจะแบ่งเป็นสามไบต์คือ การตรวจสอบสัญญาณดิจิตอล การตรวจสอบสัญญาณแอนาลอก และการตรวจสอบแบบเงื่อนไข โดยบิต 0-7 จะแสดงการตรวจช่องสัญญาณ 1-8 โดยหากมีค่าเป็น 1 ให้ทำการตรวจ แต่หากเป็น 0 ไม่ต้องทำการตรวจสอบในกรณีของสัญญาณดิจิตอลและเงื่อนไข และในกรณีของสัญญาณแอนาลอกบิต 0-3 จะแสดงการตรวจสอบสัญญาณต่ำกว่า low limit ในช่องที่ 1 สัญญาณสูงกว่า high limit ในช่องที่ 1 สัญญาณต่ำกว่า low limit ในช่องที่ 2 และ สัญญาณสูงกว่า high limit ในช่องที่ 2 ตามลำดับ หากมีค่าเป็น 1 ให้ทำการตรวจ หากมีค่าเป็น 0 ไม่ต้องทำการตรวจ
- port name เป็นชื่อของพอร์ตต่างๆของ RAU ที่มีการกำหนดจากผู้ใช้งานโดยเป็นตัวอักษรไม่เกิน 8 ตัว สามารถตั้งได้ 18 ชื่อดังนี้ ดิจิตอลอินพุต 8 พอร์ต ดิจิตอลเอาท์พุต 8 พอร์ต และแอนาลอก 2 พอร์ต
- analog level เป็นค่าของระดับ low limit และ high limit ของแต่ละช่องสัญญาณแอนาลอกโดยมีขนาดค่าละ 2 ไบต์(ช่องแอนาลอกมีความละเอียดในการวัด 12 บิต)
- condition list เป็นค่าที่ใช้เก็บเงื่อนไขในการตรวจสอบซึ่งสามารถตั้งได้ 8 แบบโดยมีขนาดข้อมูลแบบละ 2 ไบต์ โดยในไบต์แรกบิต 0-7 จะแทนการตรวจสอบพอร์ตดิจิตอล 1-8 ในไบต์ที่สองบิต 0-1 จะแทนการตรวจสอบ low limit และ high limit ในช่องแอนาลอกที่ 1 และ บิต 2-3 จะแทนการตรวจสอบ low limit และ high limit ในช่องแอนาลอกที่ 2 โดยหากมีค่าเป็น 1 จะทำการตรวจสอบหากมีค่าเป็น 0 จะไม่ทำการตรวจสอบ ถ้าทุกพอร์ตที่มีการเลือกตรวจสอบเกิดความผิดพลาดพร้อมกันก็จะทำการเตือนไปยังผู้ใช้
- sending list เป็นค่าเงื่อนไขในการแจ้งข่าวสารสั้นว่าเหตุการณ์ผิดพลาดหนึ่งๆต้องการส่งให้เลขหมายใดทราบบ้าง โดยเป็นข้อมูลขนาด 2 ไบต์โดยบิต 0-4 ในไบต์แรกแทนการส่งข่าวสารให้ผู้ใช้งานหมายเลขที่ 1 ถึง 5 บิต 0-4 ในไบต์ที่ 2 แทนการส่งข่าวสารให้ผู้ใช้งานหมายเลขที่ 6-10 โดยหากมีค่าเป็น 1 แสดงว่ามี

การส่ง หากเป็น 0 แสดงว่าไม่ต้องส่ง โดยข้อมูลประเภทนี้จะมีด้วยกันทั้งหมด 20 ชุด(RAU สามารถมีการแจ้งความผิดพลาดได้ทั้งหมด 20 แบบ)

- message เป็นข่าวสารสั้นที่จะทำการแจ้งไปให้กับผู้ใช้ โดยมีขนาดไม่เกิน 25 ตัวอักษร มีด้วยกันทั้งหมด 20 ข่าวสาร

จากที่ได้กล่าวมาแล้วด้านบนสามารถสรุปเป็นแผนผังหน่วยความจำใน EEPROM ที่ใช้ในการบันทึกข้อมูลต่างๆ ได้ดังรูปที่ 6.3



รูปที่ 6.3 แผนผังหน่วยความจำใน EEPROM

จากรูปที่ 6.3 จะเห็นได้ว่าประเภทของข้อมูลที่ทำการบันทึกใน EEPROM นั้นมีอยู่ด้วยกัน 8 ประเภท โดยในรูปที่ 6.3 แสดงการเรียงลำดับของข้อมูลแต่ละประเภทใน EEPROM โดยในการทำงานของ RAU นั้น RAU จะทำการอ่านค่าข้อมูลประเภท piority, user number,enable list และ port name ไว้ในหน่วยความจำ ram ภายในแต่ข้อมูลในส่วนอื่นจะทำการอ่านค่าเข้ามาเมื่อต้องการใช้ เช่นข้อมูลส่วน analog level และ condition list ที่จะทำการอ่านค่าเมื่อ RAU จะทำงานในการตรวจสอบความผิดพลาดของสัญญาณ ส่วนข้อมูลในส่วน message และ sending list นั้นจะทำการอ่านเมื่อต้องการแจ้งข่าวสารสั้นให้กับผู้ใช้งาน

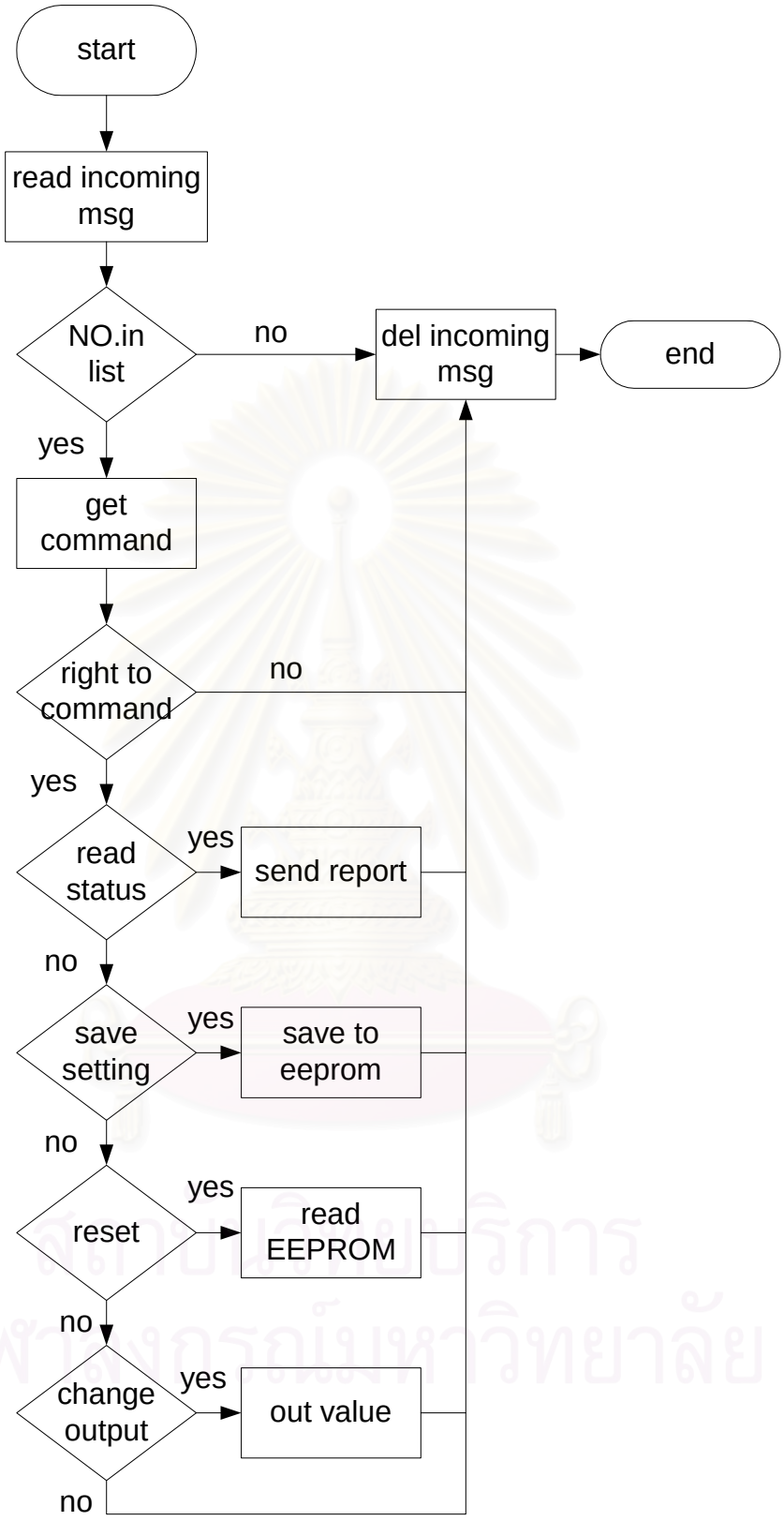
ดังนั้นสามารถสรุปปริมาณการใช้งานหน่วยความจำ EEPROM ได้ดังนี้

piority(2)+[user number(10)×8]+[analog lever(4)×2]+enable list(3)+[port name(18)×8]
+[message(20)×25]+[sending list(20)×2]+[condition list(8)×2] = 793 Bytes

6.1.3 การจัดการข่าวสารสั้นเข้าใน RAU

ในกรณีที่มีข่าวสารสั้นเข้ามานั้น โทรศัพท์มือถือจะทำการแจ้งมาที่ RAU ว่ามีข่าวสารสั้นและถูกบันทึกไว้ใน SIM card พร้อมทั้งแจ้งหมายเลขลำดับของข่าวสารสั้นที่ได้บันทึกไว้ด้วย เพื่อให้ RAU สามารถอ่านข่าวสารสั้นตามเลขลำดับนั้นได้ เมื่อ RAU ทำการอ่านข่าวสารสั้นซึ่งอยู่ในรูป PDU(protocal data unit) มาไว้ใน buffer ก็จะทำการดึงข้อมูลส่วนที่ต้องการใช้งานออกมา ซึ่งก็คือเลขหมายของผู้ส่ง และส่วนข่าวสาร โดยส่วนข่าวสารจะต้องนำมาทำการแปลงข้อมูลจากเพื่อให้เป็นตัวอักษรก่อน แล้วจึงทำการวิเคราะห์เนื้อหาในข่าวสารว่ามีการสั่งการใดกับ RAU บ้าง โดยคำสั่งที่ใช้ในการสั่งการ RAU สามารถดูได้ในบทที่ 4

เมื่อ RAU จัดการกับข่าวสารสั้นเข้าเสร็จสิ้น ก็จะทำการลบข่าวสารสั้นที่ได้ถูกบันทึกไว้ออกเพื่อป้องกันข้อมูลเต็ม เนื่องจาก SIM card สามารถบันทึกข่าวสารสั้นได้จำนวนจำกัด แล้วจึงกลับไปทำงานในการตรวจสอบความผิดพลาดจากสัญญาณวัดที่เข้ามาที่ RAU ต่อไป ซึ่งการทำงานในส่วนนี้สามารถแสดงได้ดังรูปที่ 6.4



รูปที่ 6.4 โครงสร้างส่วน handle msg ของ RAU

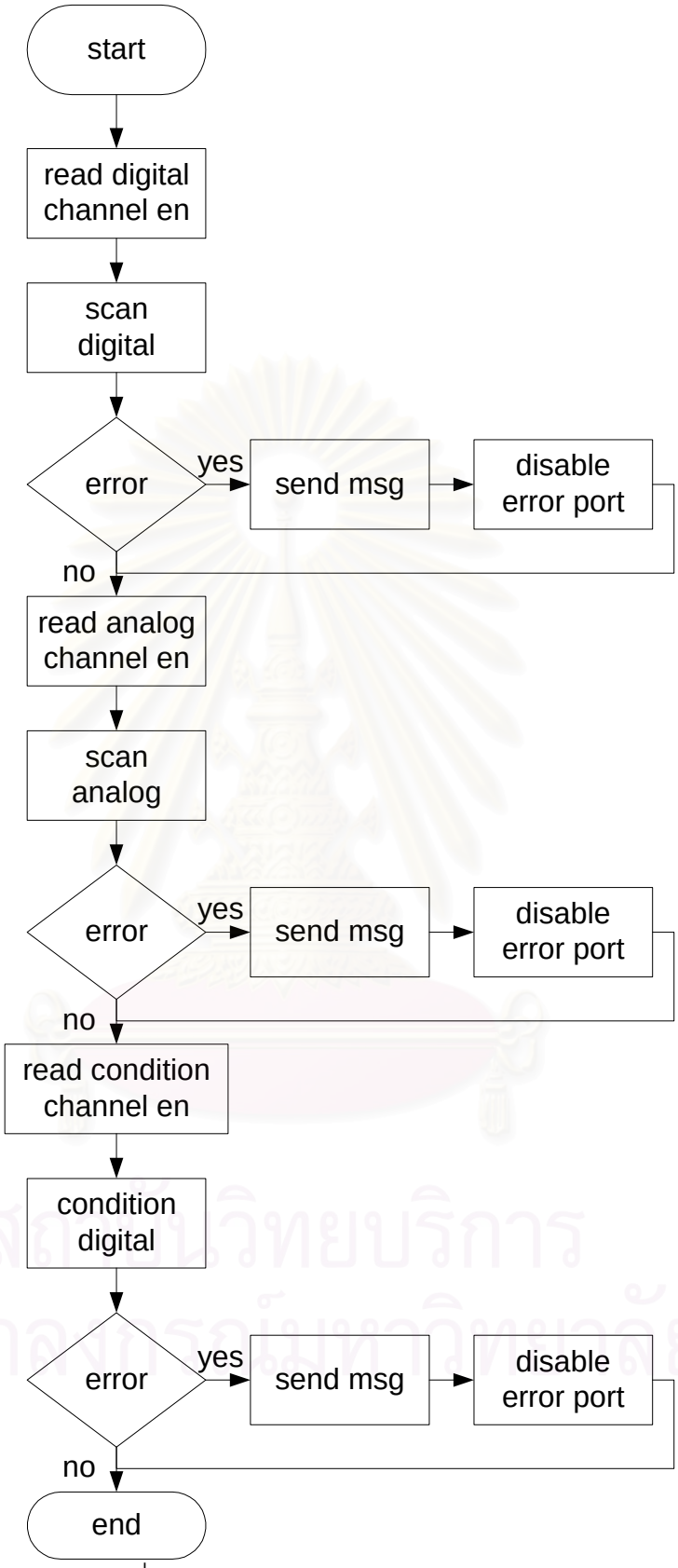
โดยการทำงานในส่วนต่างๆ สามารถแยกพิจารณาได้ดังต่อไปนี้

- read incoming msg เป็นการอ่านข่าวสารสั้น โดยการอ่านจากค่าเลขลำดับที่โทรศัพท์มือถือส่งมาให้ ในการอ่านนี้ขั้นแรกจะทำการเช็คเลขหมายที่ส่งเข้ามาก่อนเพื่อตรวจสอบว่าอยู่ในรายชื่อผู้ใช้งานหรือไม่ หากไม่อยู่ก็จะทำการลบข่าวสารสั้นและจบการทำงานในส่วนนี้
- get command เป็นการอ่านข่าวสารสั้นเพื่อหาคำสั่งที่ผู้ใช้งานได้สั่งการมาที่ RAU โดยในการหาคำสั่งนี้จะมีการตรวจสอบด้วยว่าเลขหมายนั้นมีสิทธิในการใช้คำสั่งนั้นหรือไม่ หากไม่ก็จะไม่ดำเนินการคำสั่งและออกจากการทำงานในส่วนนี้
- read status เป็นคำสั่งในการขอดูค่าต่างๆที่ RAU ทำการวัดอยู่ โดยการแสดงผลจะทำโดยการส่งข่าวสารสั้นไปยังเลขหมายที่ใช้คำสั่งเข้ามา
- save setting เป็นกลุ่มคำสั่งในการเปลี่ยนค่าต่างๆใน RAU ซึ่งค่าที่สามารถเปลี่ยนได้โดยการใช้กลุ่มคำสั่งนี้คือ ชื่อพอร์ต ข่าวสารสั้น การสั่งตรวจสอบหรือเลิกตรวจสอบสัญญาณเข้า
- reset เป็นคำสั่งที่ใช้ในการให้ RAU เริ่มทำงานใหม่ โดย RAU จะทำการโหลดค่าตัวแปรต่างๆใน EEPROM ใหม่ เพื่อใช้ในการเริ่มการทำงานใหม่
- change output เป็นคำสั่งที่ใช้ในการสั่งให้ RAU ให้สัญญาณออกที่พอร์ตขาออกเพื่อทำการควบคุมอุปกรณ์ภายนอกอื่นๆ

6.1.4 การทำงานตรวจหาความผิดพลาดของ RAU

ในการตรวจสอบหาความผิดพลาดนั้น RAU จะทำการตรวจสอบเรียงไปตามลำดับดังนี้ พอร์ตดิจิตอล พอร์ตแอนาลอก และการตรวจสอบแบบเงื่อนไข โดยก่อนการตรวจสอบค่าสัญญาณ RAU จะทำการตรวจสอบไบต์ควบคุมที่ทำหน้าที่ในการบันทึกการเลือกหรือไม่เลือกให้ทำการตรวจสอบสัญญาณนั้นๆก่อนที่จะทำการตรวจสอบสัญญาณ หากไม่มีการเลือกก็จะไม่ทำการตรวจสอบสัญญาณนั้น

ในการตรวจสอบความผิดพลาดนั้น หากพบความผิดพลาด RAU ก็จะทำการส่งข่าวสารสั้นไปเตือนยังเลขหมายที่ได้มีการเลือกให้ส่งไปในกรณีนั้นๆ และจะทำการยกเลิกการตรวจสอบสัญญาณนั้นเพื่อป้องกันการส่งข่าวสารสั้นแบบไม่รู้จบ การที่จะให้ RAU กลับมาทำการตรวจสอบสัญญาณนั้นอีกครั้งทำได้โดยการสั่งด้วยคำสั่ง RESET มาที่ RAU จะทำให้ RAU โหลดค่าไบต์ควบคุมในการตรวจสอบสัญญาณที่ได้มีการบันทึกไว้ขึ้นมาใหม่และจะทำให้ RAU กลับมาตรวจสอบช่องสัญญาณนั้นตามเดิม ซึ่งการทำงานในส่วนนี้สามารถแสดงได้ดังรูปที่ 6.5



รูปที่ 6.5 โครงสร้างส่วน scan port ของ RAU

6.15 รายละเอียดฟังชันการทำงานของ RAU

ในงานวิจัยนี้ได้มีการออกแบบโครงสร้างทางโปรแกรมของ RAU ให้อยู่ในลักษณะการทำงานแบบการเรียกใช้งานโปรแกรมย่อยจากโปรแกรมหลัก โดยรายละเอียดฟังชันต่างๆที่โปรแกรมหลักใน RAU ได้มีการเรียกใช้งานมีดังต่อไปนี้

Interrupt เป็นโปรแกรมส่วนที่ใช้ในการติดต่อสื่อสารแบบอนุกรมโดยจะทำหน้าที่หลักในการรับข้อมูลเข้าของการสื่อสารอนุกรมโดยวิธีการอินเตอร์รัปต์ แล้วนำมาเก็บไว้ในหน่วยความจำ buffer เพื่อรอการประมวลผล

Read เป็นส่วนโปรแกรมที่ใช้ในการอ่านค่าต่างๆ ที่ใช้ในการเริ่มการทำงานของ RAU และใช้ในการ reset RAU

Scan เป็นส่วนโปรแกรมที่ทำหน้าที่ในการตรวจสอบความผิดพลาดของสัญญาณที่ต่อเข้ามาที่ RAU โดยอ้างอิงการตรวจจากข้อมูลส่วน enable/disable การตรวจสอบ

Download เป็นส่วนโปรแกรมที่ทำหน้าที่ในการจัดการข้อมูลที่ถูกส่งมาจากคอมพิวเตอร์ในการตั้งค่าการทำงานของ RAU เพื่อนำไปบันทึกลงใน EEPROM

Encode เป็นส่วนโปรแกรมที่ใช้ในการแปลงข้อมูลตัวอักษรแบบ 8 บิต ให้เป็นแบบ 7บิตตามมาตรฐานการสื่อสารในระบบข่าวสารสั้นดังที่ได้กล่าวไว้ในบทที่ 2

Decode เป็นส่วนโปรแกรมที่ใช้ในการแปลงข้อมูลตัวอักษรแบบ 7 บิต ตามมาตรฐานการสื่อสารในระบบข่าวสารสั้นให้กลับเป็นแบบ 8 บิตดังที่ได้กล่าวไว้ในบทที่ 2

Find_no เป็นส่วนโปรแกรมที่ใช้ในการค้นหาเลขหมายผู้ส่งในข่าวสารสั้นที่ได้รับ

I2C เป็นส่วนโปรแกรมที่ใช้ในการสื่อสารแบบ I^2C

Find_com เป็นส่วนโปรแกรมที่ใช้ในการค้นหาคำสั่งจากข่าวสารสั้นที่ได้รับการแปลจาก Decode แล้วให้อยู่ในรูปตัวเลขเพื่อพร้อมที่นำไปดำเนินการ

Find_port เป็นส่วนโปรแกรมที่ใช้ในการค้นหาเป้าหมายในการทำงานของคำสั่งในข่าวสารสั้นที่ได้รับการแปลจาก Decode ให้อยู่ในรูปตัวเลขพร้อมที่จะนำไปดำเนินการ

Find_data เป็นส่วนโปรแกรมที่ใช้ในการค้นหาข้อมูลที่ต้องการเปลี่ยนในข่าวสารสั้นที่ได้รับการแปลจาก Decode เพื่อใช้ในกรณีที่ต้องการเปลี่ยนชื่อพอร์ต หรือเปลี่ยนข่าวสารสั้น

Exe เป็นส่วนโปรแกรมที่นำค่าตัวเลขระบุคำสั่งและเป้าหมายจาก Find_com และ Find_port มาดำเนินการตามคำสั่ง

Inttohex เป็นส่วนโปรแกรมที่ใช้ในการแปลงข้อมูลแบบ integer เพื่อให้อยู่ในรูปตัวอักษรที่แสดงเป็นเลขฐาน 16

Inttostr เป็นส่วนโปรแกรมที่ใช้ในการแปลงข้อมูลแบบ integer เพื่อให้อยู่ในรูปตัวอักษรที่แสดงเป็นเลขฐาน 10

Inttofloat เป็นส่วนโปรแกรมที่ใช้ในการแปลงข้อมูลแบบ integer เพื่อให้อยู่ในรูปตัวอักษรที่แสดงเป็นเลขฐาน 10 แบบมีทศนิยม

Hextoint เป็นส่วนโปรแกรมที่ใช้ในการแปลงข้อมูลตัวอักษรที่แสดงเลขฐาน 16 ให้อยู่ในรูปข้อมูลแบบ integer

Save_name เป็นส่วนโปรแกรมที่ใช้ในการบันทึกชื่อของพอร์ตต่างๆ ที่ได้มีการตั้งจากทางข่าวสารสั้น ซึ่งจะถูกเรียกโดยโปรแกรม Exe

Save_msg เป็นส่วนโปรแกรมที่ใช้ในการบันทึกข่าวสารสั้นที่ใช้ในการแจ้งเตือนที่ได้มีการตั้งจากทางข่าวสารสั้น ซึ่งจะถูกเรียกโดยโปรแกรม Exe

ADC เป็นส่วนโปรแกรมที่ทำหน้าที่ติดต่อเพื่ออ่านข้อมูลกับ A/D

Send_mess เป็นส่วนโปรแกรมที่ใช้ในการส่งข่าวสารสั้นของ RAU

6.15 การคำนวณเกี่ยวกับสัญญาณแอนาลอกของ RAU

ในการทำงานการตรวจสอบความผิดพลาดนั้น RAU จะต้องทำการอ่านค่าจากสัญญาณแอนาลอก 0-5V และ 4-20mA เพื่อนำมาเปรียบเทียบกับระดับที่ได้มีการตั้งค่าไว้ โดยระดับที่ตั้งค่าไว้นี้จะมีรูปแบบเป็นเปอร์เซ็นต์ และมีทศนิยมอีก 2 ตำแหน่ง แต่จากการที่ไมโครคอนโทรลเลอร์ไม่สามารถจะคำนวณข้อมูลแบบ floating point ได้ดังนั้นจึงต้องมีการออกแบบการคำนวณเพื่อช่วยในการทำงานของไมโครคอนโทรเลอร์ดังนี้

จากการที่ข้อมูลจาก A/D มีความละเอียด 12 บิตจึงสามารถแบ่งเป็นระดับได้

$$2^{12} = 4096 \text{ ระดับ}$$

ในการแปลงข้อมูล 12 บิตให้อยู่ในรูปเปอร์เซ็นต์นั้นสามารถทำได้ดังนี้โดยให้ข้อมูลจาก A/D แทนด้วย A และข้อมูลที่ทำการแปลงเป็นเปอร์เซ็นต์แล้วแทนด้วย B

$$B = (A \times 10000) \text{ DIV } 4096$$

โดยข้อมูล B ที่ได้จะมีการสูญเสียข้อมูลไปเนื่องจากคำสั่ง DIV เป็นการหารแบบเอาค่าจำนวนเต็มไม่เอาเศษ และค่า B ที่ได้นั้นต้องนำมาควณเพื่อแสดงเป็นตัวเลขรูปแบบเปอร์เซนต์ดังต่อไปนี้

หลักร้อย = B DIV 10000

หลักสิบ = (B MOD 10000) DIV 1000

หลักหน่วย = (B MOD 1000) DIV 100

ทศนิยมตำแหน่งแรก = (B MOD 100) DIV 10

ทศนิยมตำแหน่งที่สอง = B MOD 10

โดยการ MOD เป็นการหารแบบเอาเศษ ซึ่งการคำนวณข้างบนสามารถใช้ได้เลยกับสัญญาณแบบ 0-5 V แต่ในกรณีสัญญาณ 4-20 mA นั้นต้องมีการคำนวณเพิ่มเนื่องจากการรับสัญญาณ 4-20 mA นั้นไม่ได้มีการออกแบบวงจรขยายในการแปลงสัญญาณแบบกระแสให้เป็นแรงดันเพื่อป้อนให้กับ A/D แต่ใช้วิธีการรับสัญญาณกระแสผ่านความต้านทาน 250 Ω เพื่อแปลงเป็นสัญญาณแรงดันแบบ differential ป้อนให้กับ A/D ดังนั้นจึงมีช่วงแรงดันอยู่ในช่วง 1-5V จึงต้องมีการนำมาคำนวณแปลงให้อยู่ในช่วง 0-5V ก่อนที่จะได้นำเป็นคำนวณตามที่ได้แสดงไว้ด้านบนโดยการใช้สมการดังต่อไปนี้

out = in × 5/4 –5/4

โดยที่ out คือสัญญาณที่แปลงได้อยู่ในช่วง 0-5V

in คือสัญญาณที่ก่อนแปลงอยู่ในช่วง 1-5V

แต่ในความเป็นจริงแล้วการคำนวณไม่สามารถทำการหาร(/) ได้เนื่องจากไม่สามารถคำนวณแบบ floating point ได้จึงใช้การ DIV แทนทำให้มีข้อมูลหายไปบ้างบางส่วน และการลบด้วย 5/4 นั้นก็จะถูกแปลงให้อยู่ในรูปค่าอื่น เนื่องจากสัญญาณจาก A/D มีระดับอยู่ในช่วง 0-4096 ซึ่งเป็นช่วงที่ใช้ในการเปรียบเทียบกับสัญญาณในช่วง 0-5 V ดังนั้น –5/4 จะมีค่าเป็น –400H และการคำนวณเพื่อเตรียมข้อมูลสำหรับแสดงในรูปแบบเปอร์เซนต์สามารถแสดงได้ดังนี้

data_out = (data_in × 5) DIV 4 - 400H

โดยที่ data_out คือข้อมูลที่พร้อมจะนำไปคำนวณเป็นเปอร์เซนต์

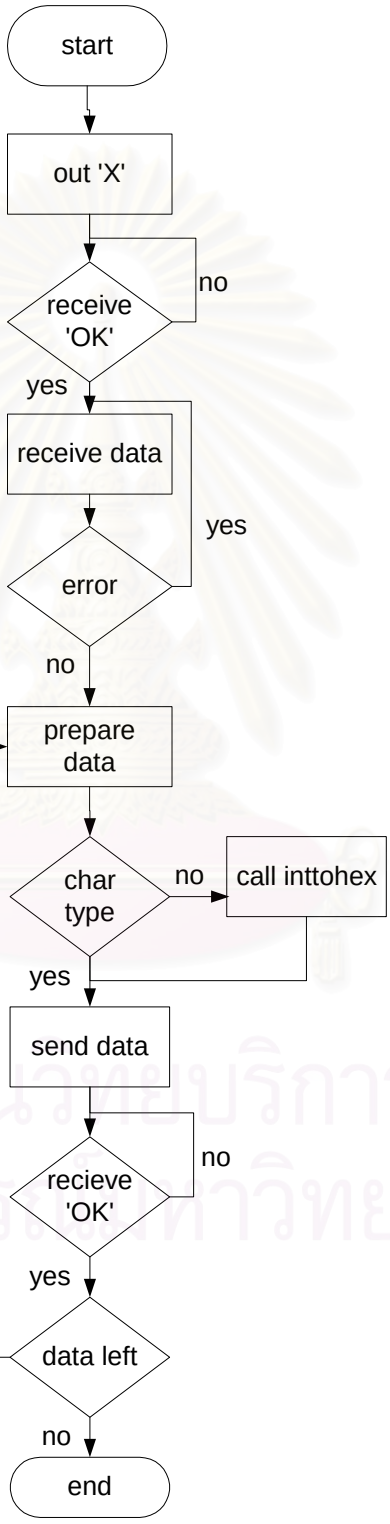
data_in คือข้อมูลที่ได้รับมาจาก A/D

6.2 โครงสร้างทางโปรแกรมที่ใช้ในคอมพิวเตอร์

โครงสร้างทางโปรแกรมที่ใช้ในคอมพิวเตอร์มีอยู่ด้วยกันสองส่วนคือ ส่วนโหลดข้อมูลลง RAU และส่วนที่ใช้เป็น alarm control unit เพื่อใช้ในการเก็บข่าวสารสั้นที่ RAU ส่งมาให้เพื่อทำการบันทึกค่าเพื่อใช้ในการสืบหาสาเหตุย้อนหลังได้(history) โดยในแต่ละส่วนจะมีการทำงานดังนี้

6.2.1 โครงสร้างของโปรแกรมโหลดข้อมูลลง RAU

ในการตั้งค่าการทำงานต่างๆใน RAU สามารถทำได้โดยการโหลดค่าจากคอมพิวเตอร์โดยผ่านโปรแกรมที่ได้พัฒนาขึ้น เพื่อใช้ในการแปลงข้อมูลและส่งให้กับ RAU โดยโปรแกรมที่ได้พัฒนาขึ้นนี้มีโครงสร้างดังต่อไปนี้



รูปที่ 6.6 โครงสร้างทางโปรแกรมของโปรแกรมโหลดข้อมูล

จากรูปที่ 6.6 ในการเริ่มทำงานโปรแกรมจะทำการส่งค่าตัวอักษร X ไปให้ยัง RAU เพื่อเป็นการแจ้งการเริ่มทำการติดต่อ หากได้รับการตอบกลับมาว่า OK ก็จะเริ่มทำการโหลดข้อมูล โดยเริ่มจากการรับข้อมูลที่ถูกป้อนเข้ามาเพื่อเตรียมส่งไปให้แก่ RAU โดยในการรับข้อมูลนี้จะมีการตรวจสอบความผิดพลาดจากการป้อนข้อมูลด้วยโดยแบ่งเป็นดังต่อไปนี้

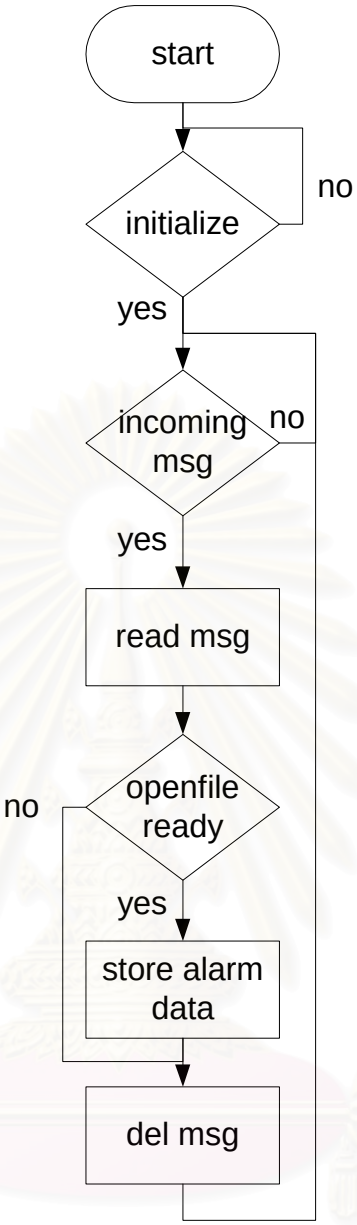
- เลขหมายโทรศัพท์มีจำนวนตัวอักษรน้อยกว่า 9 ตัวอักษร
- ข่าวสารสั้นมีความยาวเกิน 25 ตัวอักษร
- ค่าขอบเขตแอนาลอกมีค่าสูงกว่า 100 หรือ ต่ำกว่า 0
- ชื่อของพอร์ตต่างๆ มีความยาวเกิน 8 ตัวอักษร

หากพบความผิดพลาดโปรแกรมก็จะทำการรับข้อมูลใหม่ จนกว่าจะได้ค่าที่ถูกต้อง เมื่อทำการรับข้อมูลเสร็จก็จะเข้าสู่ขั้นตอนการเตรียมข้อมูล โดยในการเตรียมข้อมูลนี้จะมีการตรวจสอบว่าข้อมูลที่จะส่งให้ RAU เป็นแบบตัวอักษรหรือไม่ หากไม่ใช่ก็จะทำการแปลงข้อมูลโดยการเรียกโปรแกรม hextoint เพื่อแปลงข้อมูลที่เป็น integer ให้กลายเป็นตัวอักษรที่อยู่ในรูปเลขฐาน 16 แล้วจึงทำการส่งให้กับ RAU เมื่อโปรแกรมทำการส่งข้อมูลให้กับ RAU แล้วก็จะทำการรอการตอบรับกลับจาก RAU หากมีการตอบกลับมาว่า OK แสดงว่าการบันทึกข้อมูลเรียบร้อย โปรแกรมจะทำการส่งข้อมูลชุดถัดไปให้กับ RAU หากข้อมูลที่จะโหลดลง RAU ยังไม่หมด หากข้อมูลหมดแล้วก็จะจบการทำงานในส่วนนี้

6.2.2 โครงสร้างทางโปรแกรมของ alarm control unit

โปรแกรม alarm control unit นี้ใช้ในการบันทึกข่าวสารสั้นที่ RAU ส่งมา เพื่อทำการทำการเก็บบันทึกไว้เป็นประวัติการเตือน ซึ่งสามารถนำมาวิเคราะห์ในภายหลังเพื่อหาสาเหตุของความผิดปกติในระบบที่ RAU ทำการวัดอยู่ โดยในการเก็บค่านั้น RAU จะทำการดึงข้อมูลจากข่าวสารสั้นออกมาสามส่วนคือ เลขหมายของผู้ส่ง วันเวลาที่มีการเตือนซึ่งจะเป็นการบันทึกข้อมูลในส่วน time stamp ในข่าวสารสั้นไว้โดยอยู่ในรูปแบบ ปี เดือน วัน ชั่วโมง นาที และวินาที และอีกส่วนคือส่วนข่าวสารสั้นที่ RAU ส่งมาให้ โดยในการบันทึกข้อมูลจะมีการตรวจสอบว่ามีการกำหนดชื่อไฟล์ข้อมูลที่ต้องการจะทำการบันทึกหรือยัง หากยังก็จะไม่ทำการบันทึกแต่จะแสดงรายละเอียดข่าวสารสั้นขึ้นทางหน้าจอเพียงอย่างเดียว

เมื่อทำการจัดการกับข่าวสารสั้นเข้าเสร็จโปรแกรมก็จะทำการลบข่าวสารสั้นที่ถูกบันทึกไว้ใน SIM card ของโทรศัพท์มือถือ เพื่อทำการเตรียมการรับข่าวสารสั้นเข้าใหม่ต่อไป ซึ่งรายละเอียดโปรแกรมสามารถแสดงดังต่อไปนี้



รูปที่ 6.7 โครงสร้างของโปรแกรมเก็บประวัติการเตือน

บทที่ 7

การทดสอบระบบ

7.1 การเตรียมระบบในการทดสอบ

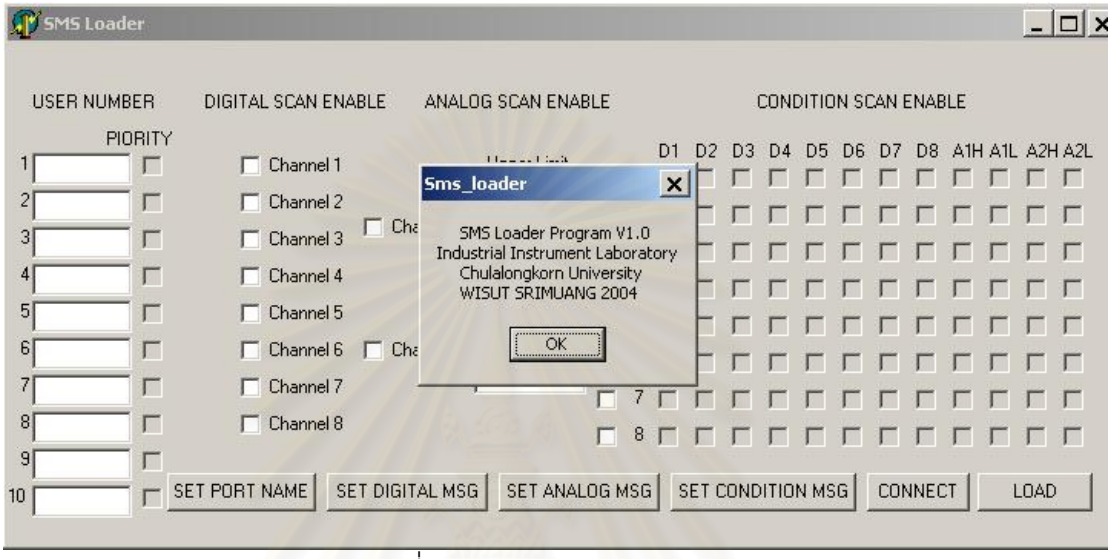
ในบทนี้จะได้กล่าวถึงการทดสอบระบบที่ได้ทำการออกแบบขึ้น ทั้งทางฮาร์ดแวร์และซอฟต์แวร์ โดยจากที่ได้กล่าวไปแล้วข้างต้นว่าระบบนี้ประกอบไปด้วยส่วนแจ้งเตือนและสั่งการระยะไกล(RAU) และส่วนที่เป็นผู้ใช้งานโดยแบ่งเป็นผู้ใช้งานที่เป็นบุคคลและผู้ใช้งานที่เป็นคอมพิวเตอร์(Alarm Control Unit) ซึ่งเชื่อมต่อข้อมูลกับโทรศัพท์มือถือ โดยขั้นตอนการทดสอบสามารถแสดงได้ดังนี้

7.1.1 การเตรียม RAU

ในงานวิจัยนี้ได้ใช้บอร์ดสำเร็จรูปที่มีไมโครคอนโทรลเลอร์ P89C51RD2 ของบริษัท ETT ซึ่งรูปตัวบอร์ดสามารถแสดงได้ดังรูปที่ 7.1

รูปที่ 7.1 บอร์ดคอนโทรลเลอร์ P89C51RD2

โดยก่อนนำไปใช้งานนั้น ต้องมีการตั้งค่ารายละเอียดในการทำงานต่างๆให้กับตัวบอร์ด ซึ่งสามารถทำได้โดยการโหลดผ่านคอมพิวเตอร์ด้วยโปรแกรมที่ได้พัฒนาขึ้นในงานวิจัยชื่อว่า SMS_loader โดยการโหลดผ่านพอร์ต COM1 ซึ่งรายละเอียดของโปรแกรมสามารถดูได้ในภาคผนวก ก.



รูปที่ 7.2 โปรแกรม SMS_loader

ในการนำบอร์ดนี้ไปใช้งานเป็น RAU นั้นจะต้องมีการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือก่อนโดยอาศัยสายสัญญาณซึ่งสามารถแสดงได้ดังรูปที่ 7.3 ซึ่งเป็นการเชื่อมต่อแบบอนุกรมอะซิโครนัส

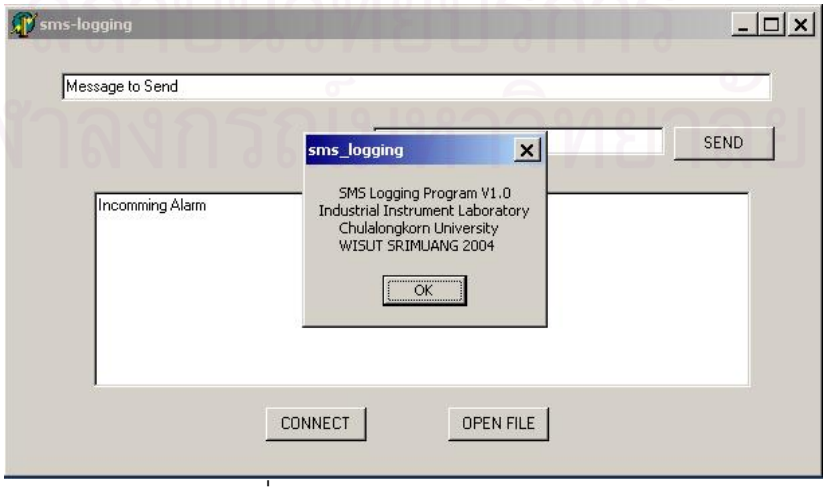
รูปที่ 7.3 สายสัญญาณสำหรับการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือ

รูปที่ 7.4 RAU ของระบบแจ้งเตือนและสั่งการทางไกล

7.1.2 การเตรียมการส่วนผู้ใช้งาน

ในส่วนของผู้ใช้งานในกรณีที่เป็นบุคคล(ผู้ที่มีโทรศัพท์มือถือใช้งาน) ไม่ต้องมีการเตรียมการในการใช้งานใดๆ แต่ส่วนคอมพิวเตอร์(Alarm Control Unit) ต้องทำการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือก่อนด้วยสายสัญญาณดังรูปที่ 7.3 โดยต่อเข้าที่พอร์ต COM1 ของคอมพิวเตอร์

และในการใช้งานเป็นตัว Alarm Control Unit นั้นจะต้องมีการเปิดโปรแกรมในการทำงานชื่อว่า SMS_logging เพื่อทำงานในการเก็บและบันทึกการแจ้งเตือนจากตัว RAU



รูปที่ 7.5 โปรแกรม SMS_logging

รูปที่ 7.6 Alarm Control Unit ของระบบ

รูปที่ 7.7 ระบบแจ้งเตือนและสั่งการทางไกลผ่านระบบบริการข่าวสารสั้น

7.2 การทดสอบระบบ

ในการทดลองระบบนั้น ได้มีการทดสอบโดยผ่านชุดทดสอบซึ่งประกอบไปด้วยชุดทดสอบดังรูปที่ 7.8 เพื่อทดสอบการทำงานในส่วนดิจิตอล และตัว DC generator เพื่อใช้ทดสอบการทำงานในส่วนแอนาลอก

รูปที่ 7.8 ชุดทดสอบการทำงานของระบบแจ้งเตือนและสั่งการระยะไกล

โดยในการทดลองระบบนี้ได้มีการตั้งค่าการทำงานให้แก่ RAU ดังต่อไปนี้

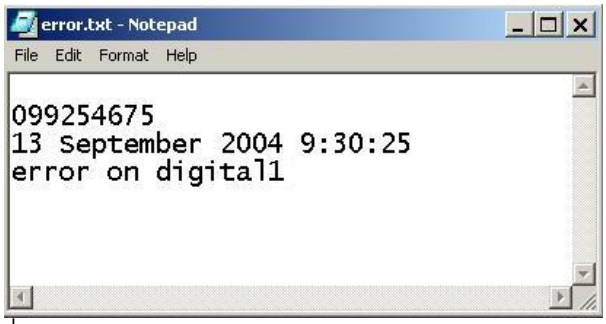
- เลขหมายที่ใช้งานระบบ 2 เลขหมาย คือ 070808812 ซึ่งทำหน้าเป็น Alarm Control Unit มีความสามารถในการสั่งการ RAU ได้ และ 096717527 ซึ่งเป็นผู้ใช้งานทั่วไป และไม่มีสิทธิสั่งการ RAU
- เลือกการตรวจความผิดพลาดของสัญญาณดิจิตอลทั้งหมด
- เลือกการตรวจสอบสัญญาณแอนาลอกประเภท 0-5V และ 4-20mA โดยสัญญาณ 0-5V เลือกระดับขอบบนที่ 80%(4V) และขอบล่างที่ 20%(1V) ส่วนสัญญาณ 4-20mA เลือกขอบบนที่ 75%(16mA) และขอบล่างที่ 30%(4.8mA)
- การตรวจสอบแบบเงื่อนไขเลือกเป็นช่องสัญญาณดิจิตอลช่องที่ 1 กับสัญญาณ 0-5V เกินระดับขอบบน

- เลือกการแจ้งเตือนไปที่เลขหมายผู้ใช้งานทั้งสองเลขหมายด้านบน
- กำหนดชื่อพอร์ตในการทำงานโดยกำหนดชื่อพอร์ตดิจิตอลขาออก3 ชื่อ fan

ในการเริ่มการทดสอบนั้น เมื่อเชื่อมต่อระบบที่จะทดสอบดังรูปที่ 7.8 แล้วจึงโดยเริ่มทำการทดสอบโดยการใช้สวิตช์บนตัวบอร์ดทดลองเพื่อทดสอบการตรวจจับความผิดพลาดและใช้ LED บนตัวบอร์ดเพื่อแสดงผลของสัญญาณขาออกของ RAU

รูปที่ 7.9 แผงทดสอบระบบแจ้งเตือนและสั่งการทางไกล

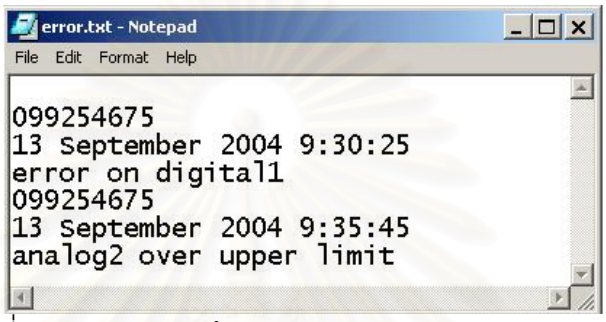
1. ทดสอบการตรวจความผิดพลาดแบบดิจิตอลโดยการสับสวิตช์ดิจิตอลขาเข้า 1(IN1) ซึ่งได้ผลการทดสอบแสดงอยู่ในรูปแฟ้มข้อมูลที่ถูกบันทึกโดย Alarm Control Unit ดังรูปที่ 7.10



รูปที่ 7.10 ผลการบันทึกการแจ้งความผิดพลาดแบบดิจิตอล

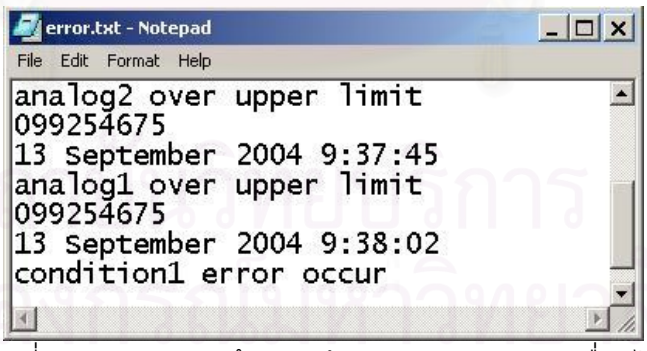
โดยในการทดลองนี้เลขหมาย 096717527 ก็ได้รับการแจ้งเตือนเป็นข้อความในลักษณะเดียวกันกับ Alarm Control Unit

2. ทดสอบการตรวจความผิดพลาดแบบแอนาลอกโดยการปรับสัญญาณ 4-20mA ให้มีค่าสูงกว่า 16mA ซึ่งได้ผลการทดสอบแสดงอยู่ในรูปแฟ้มข้อมูลที่ถูกบันทึกโดย Alarm Control Unit ดังรูปที่ 7.11



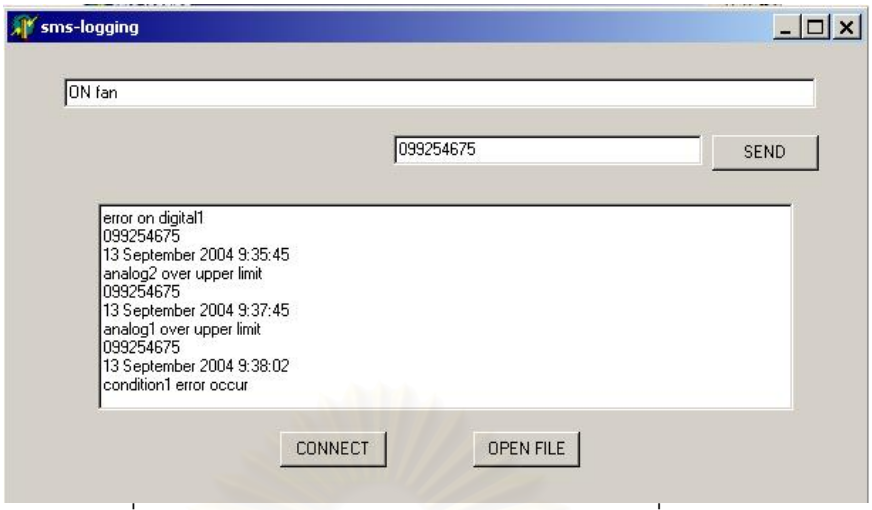
รูปที่ 7.11 ผลการบันทึกการแจ้งความผิดพลาดแบบดิจิตอล

3. ทดสอบการตรวจสอบแบบเงื่อนไขโดยการปรับสัญญาณ 0-5V ให้เกิน 4V ซึ่งจากเหตุการณ์ร่วมในข้อ 1 ทำให้สอดคล้องเงื่อนไขที่ได้ตั้ง RAU จะทำการเตือนทั้งแบบแอนาลอกและเงื่อนไข ซึ่งได้ผลการทดสอบแสดงอยู่ในรูปแฟ้มข้อมูลที่ถูกบันทึกโดย Alarm Control Unit ดังรูปที่ 7.12

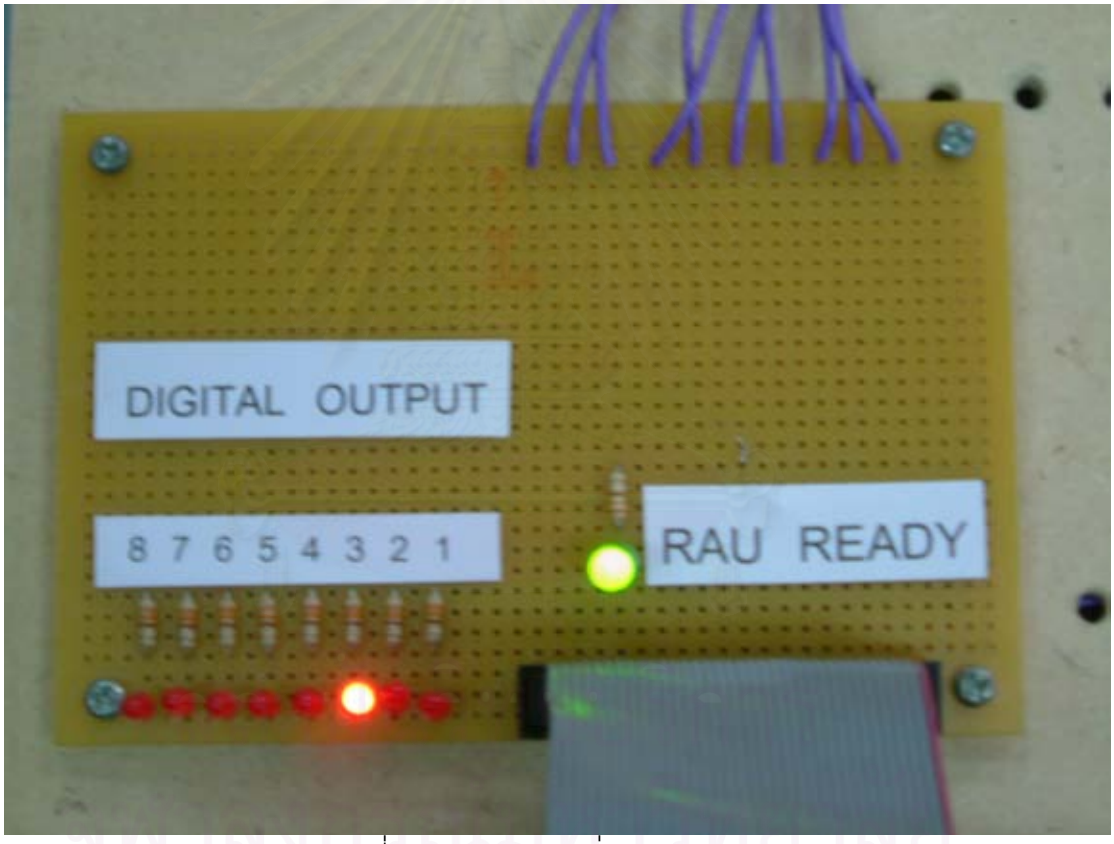


รูปที่ 7.12 ผลการบันทึกการแจ้งความผิดพลาดแบบเงื่อนไข

4. ทดสอบการสั่งการกลับไปที่ RAU โดยใช้ Alarm Control Unit ส่งข่าวสารสั้นไปที่ RAU และเป็นการทดสอบชื่อที่ได้มีการตั้งให้แก่พอร์ตของ RAU ด้วย โดยสั่งการไปที่ fan (พอร์ตดิจิตอลขาออกช่องที่3) ซึ่งสามารถแสดงได้ดังรูปที่ 7.13



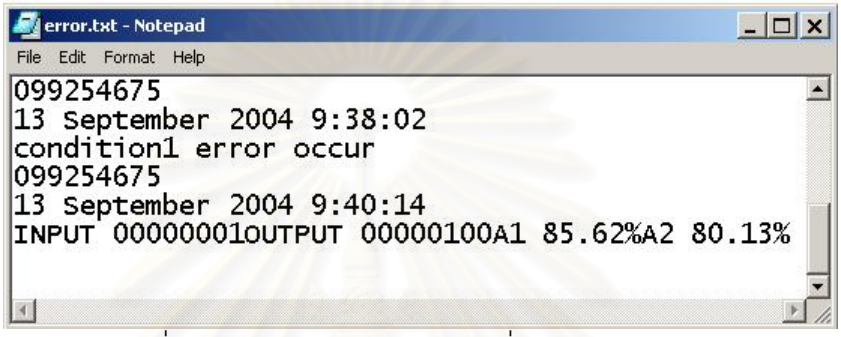
รูปที่ 7.13 การใช้งาน Alarm Control Unit ในการสั่งการ RAU



รูปที่ 7.14 ผลจากการสั่งงานของ RAU

ผลปรากฏว่า LED ที่ช่องดิจิตอลขาออก3 ติดสว่างแสดงว่าสั่งการโดยอ้างอิงจากชื่อที่ตั้งสำเร็จ และในการทดสอบการสั่งการนี้ได้ทดลองใช้โทรศัพท์เบอร์ 096717527 ในการสั่งด้วยคำสั่ง OFF fan ผลปรากฏว่า LED ไม่ดับเพราะเป็นเลขหมายที่ไม่ได้ถูกตั้งสิทธิในการสั่งการ RAU ไว้

5. ทดสอบการอ่านค่าจาก RAU ด้วยคำสั่ง READ ซึ่งได้ผลการทดสอบแสดงอยู่ในรูปแฟ้มข้อมูลที่ถูกบันทึกโดย Alarm Control Unit ดังรูปที่ 7.15



รูปที่ 7.15 ผลการบันทึกการใช้คำสั่งอ่านค่าของ RAU

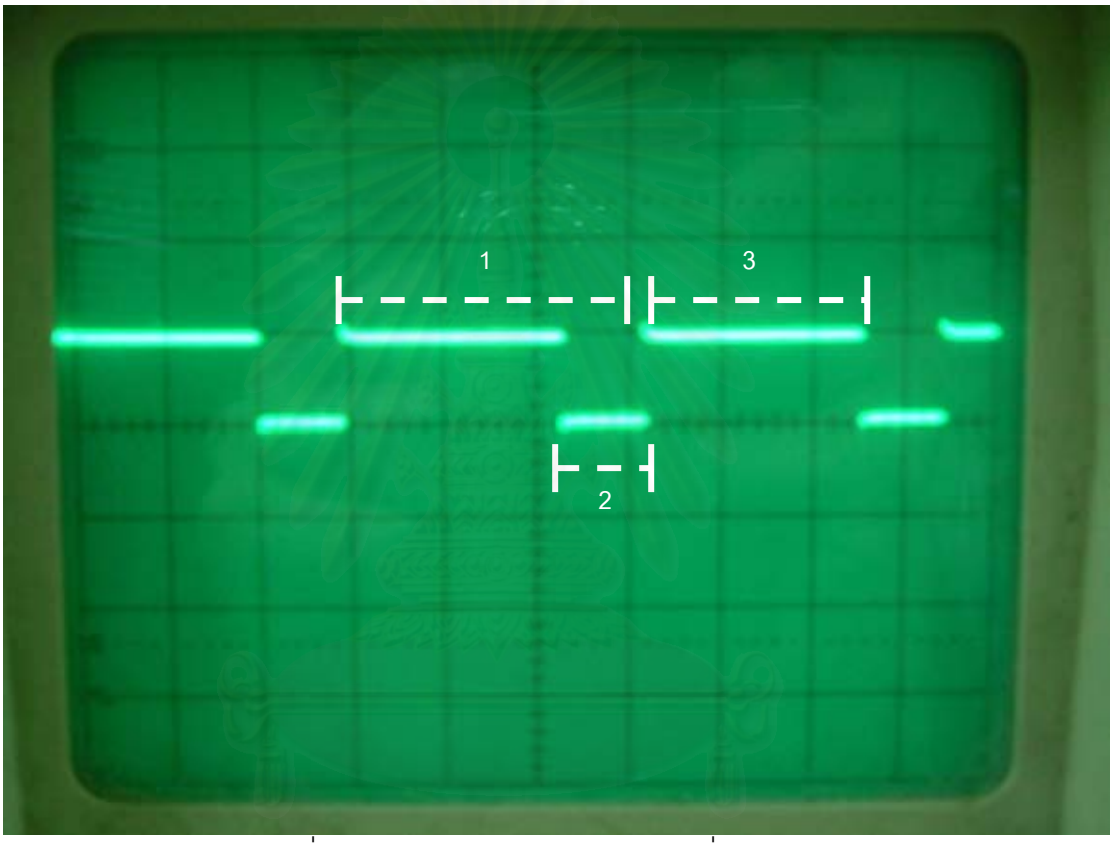
จากรูปที่ 7.15 จะพบว่ารูปแบบการบันทึกข้อมูลต่างจากข้อกำหนดที่ได้ตั้งไว้ในบทที่ 4 ซึ่งต้องมีการเว้นบรรทัดในสัญญาณแต่ละประเภท แต่เมื่อทดลองสั่งการ READ ด้วยเครื่องเลขหมาย 096717527(คำสั่ง READ ไม่มีการแบ่งแยกความสำคัญของเลขหมาย) พบว่าสามารถแสดงผลได้ตามที่ได้กล่าวไว้ในบทที่ 4

จากการที่ได้ทำการทดสอบระบบปรากฏว่า RAU สามารถทำงานในการตรวจจับความผิดพลาดของสัญญาณเข้าได้ทั้งแบบดิจิตอลและแอนาลอก แจ้งเตือนไปยังผู้ใช้งาน และสามารถรับคำสั่งจากผู้ใช้งานในรูปข่าวสารสั้นไปประมวลผลเพื่อดำเนินต่างๆโดยสามารถจำแนกผู้ใช้ที่มีสิทธิและไม่มีสิทธิในการสั่งการออกจากกันได้ ในการแจ้งเตือนเหตุการณ์ไปยัง Alarm Control Unit นั้น Alarm Control Unit สามารถทำการบันทึกผลไว้ในรูปไฟล์ข้อมูลตัวอักษรได้

7.2 การคำนวณรอบเวลาในการทำงานของ RAU

ในการคำนวณหารอบในการทำงานของ RAU นั้นไม่สามารถทำได้โดยตรงจากการคำนวณเนื่องจากในการพัฒนา RAU นั้นได้ใช้ภาษา C ในการพัฒนาจึงทำให้ไม่สามารถทำการคำนวณรอบการทำงานได้เหมือนในกรณีพัฒนาด้วยภาษา assembly ดังนั้นจึงได้ใช้วิธีการทางอ้อมคือการให้ RAU ส่งสัญญาณออกมาทางพอร์ตเมื่อมีการทำงานเสร็จ 1 รอบ โดยการทำงานที่ได้ทำการวัดรอบการทำงานนี้เป็นส่วนการทำงานของ RAU ในการตรวจจับความผิดพลาด 1 รอบและเป็นกรณีที่ไม่มีข่าวสารสั้นเข้ามาที่ RAU เนื่องจาก RAU จะทำงานในลักษณะ

การตรวจสอบความผิดพลาดและการจัดการกับข่าวสารสั้นเข้าสลับกันไปเรื่อยๆ ดังนั้นหากมีข่าวสารสั้นเข้า RAU จะต้องทำการจัดการกับข่าวสารก่อนที่จะกลับมาทำการตรวจสอบสัญญาณต่อไป โดยในการทดลองนี้ได้มีการตั้งให้ RAU ทำการส่งสัญญาณออกมาที่พอร์ต P2.3 ให้มีค่าเป็น 0 เมื่อเริ่มทำงานการตรวจสอบสัญญาณเข้าแบบดิจิตอล เมื่อทำการตรวจสัญญาณเข้าดิจิตอลเสร็จให้ส่งสัญญาณออกมาที่พอร์ต P2.3 ให้มีค่าเป็น 1 และเริ่มทำการตรวจสอบสัญญาณแอนาลอกและการตรวจสอบแบบเงื่อนไขต่อไป เมื่อเสร็จแล้วจึงให้ส่งสัญญาณออกมาที่พอร์ต P2.3 ให้มีค่าเป็น 0 ตามเดิม โดยรูปกราฟสัญญาณที่ออกจากพอร์ต P2.3 สามารถแสดงได้ดังรูป



รูปที่ 7.16 รูปสัญญาณออกของ RAU ที่พอร์ต P2.3

จากรูปที่ 7.5 เป็นรูปสัญญาณที่วัดได้จาก oscilloscope โดยมีการตั้งค่าดังนี้ แกน X ใช้เสกล 50 μs ต่อ 1 ช่อง แกน Y ใช้เสกล 5 V ต่อ 1 ช่อง จากรูปส่วนที่ 1 คือเวลาที่ RAU ใช้ในการตรวจสอบความผิดพลาดเสร็จ 1 รอบโดยมีความกว้างประมาณ 3.2 ช่องคิดเป็นเวลาประมาณ 160 μs และส่วนที่ 2 เป็นเวลาที่ RAU ใช้ในการตรวจสอบสัญญาณดิจิตอลเสร็จ และส่วนที่ 3 เป็นส่วนที่ RAU ใช้ในการตรวจสอบสัญญาณแอนาลอกและการตรวจสอบแบบเงื่อนไข จะสังเกตเห็นว่าส่วนที่ 3 ใช้เวลานานกว่าส่วนที่ 2 มากเนื่องจากต้องมีการติดต่อกับ A/D และต้องทำการอ่านค่าเงื่อนไขในการตรวจสอบจาก EEPROM เพราะไม่สามารถทำการอ่านค่าเงื่อนไขมาเก็บไว้ในหน่วยความจำภายในได้เนื่องจากมีหน่วยความจำจำกัด

7.3 การหาเวลาเฉลี่ยในการส่งและรับข่าวสารสั้น

ในงานวิจัยได้ทำการทดลองหาเวลาเฉลี่ยในการส่งข่าวสารสั้นไปยังปลายทาง โดยในการทดลองนี้ใช้โทรศัพท์มือถือสองเครื่องในการทดสอบ เนื่องจากความสะดวกในการทดสอบ และผลการส่งข่าวสารสั้นนี้สามารถใช้ในการเปรียบเทียบการทำงานในการส่งข่าวสารสั้นในการแจ้งเตือนของ RAU ได้เนื่องจากในการส่งข่าวสารสั้นของ RAU นั้นใช้โทรศัพท์มือถือเป็นสื่อกลางในการส่ง และผลการทดลองสามารถแสดงเป็นตารางได้ดังตารางที่ 7.1 โดยมีการส่งทดสอบเป็นจำนวน 10 ครั้งโดยใช้ข่าวสารสั้นว่า test

ตารางที่ 7.1 ผลการทดสอบระยะเวลาในการส่งข่าวสารสั้น test

ครั้งที่	เวลา(วินาที)
1	6.51
2	7.13
3	7.1
4	6.59
5	7.7
6	7.66
7	6.92
8	7.6
9	6.82
10	6.59
เฉลี่ย	7.06

และนอกจากนี้ยังได้ทดลองเปรียบเทียบโดยการส่งข่าวสารสั้นที่ยาวขึ้นโดยใช้โทรศัพท์มือถือคู่เดิมในการทดสอบ โดยทำการส่งข่าวสารสั้นว่า test for sending short message โดยได้ผลการทดลองดังตารางที่ 7.2

ตารางที่ 7.2 ผลการทดสอบระยะเวลาในการส่งข่าวสารสั้น test for sending short message

ครั้งที่	เวลา(วินาที)
1	8.1
2	7.74
3	7.96
4	7.64
5	7.73
6	8.22
7	7.42
8	8.28
9	8.42
10	7.53
เฉลี่ย	7.9

จากผลการทดลองจะพบว่าระยะเวลาเฉลี่ยในการส่งข่าวสารสั้นจนถึงปลายทางขึ้นอยู่กับความยาวของข่าวสารสั้นที่ส่งด้วย โดยในการส่งใช้ระยะเวลาประมาณ 7 ถึง 8 วินาที ดังนั้นในการประยุกต์ใช้งานการแจ้งเตือนและสั่งการทางไกลผ่านระบบบริการข่าวสารสั้นจึงต้องมีการคำนึงถึงเวลาในส่วนนี้ เพื่อสามารถนำระบบไปใช้งานให้เหมาะสม

บทที่ 8

สรุปผลและข้อเสนอแนะ

8.1 สรุปผล

วิทยานิพนธ์ฉบับนี้ได้แสดงการออกแบบและพัฒนาระบบแจ้งเตือนและสั่งการระยะไกลผ่านระบบบริการข่าวสารสั้น โดยระบบประกอบไปด้วยส่วนที่เป็นตัวตรวจจับความผิดพลาดของสัญญาณและทำการแจ้งเตือนไปยังผู้ใช้และสามารถรับคำสั่งจากผู้ใช้งานมาดำเนินการได้ ซึ่งอุปกรณ์นี้สามารถติดต่อสื่อสารในระบบข่าวสารสั้นได้โดยการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือเพื่อใช้เป็นสื่อกลางในการสื่อสาร โดยเรียกอุปกรณ์ส่วนนี้ว่า Remote Alarm Unit(RAU) และอีกส่วนคือผู้ที่ใช้งานระบบโดยมี 2 ลักษณะคือผู้ใช้งานที่เป็นบุคคลซึ่งมีโทรศัพท์มือถือไว้ใช้งาน และแบบที่เป็นคอมพิวเตอร์โดยทำการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือในลักษณะเดียวกับ RAU โดยตัวคอมพิวเตอร์นี้เรียกว่า Alarm Control Unit ทำหน้าที่ในการบันทึกเหตุการณ์ที่ RAU ทำการแจ้งเตือนไว้ในรูปแบบของไฟล์ตัวอักษรเพื่อใช้เป็นข้อมูลในการวิเคราะห์หาสาเหตุความผิดพลาด และยังสามารถใช้ในการสั่งการกลับไปที่ RAU ได้อีกด้วย ข้อดีของระบบนี้คือเป็นระบบที่มีความสะดวกในการใช้งานเนื่องจากไม่ต้องสร้างระบบสื่อสารระหว่างผู้ใช้กับตัว RAU ใหม่เนื่องจากระบบการสื่อสารแบบข่าวสารสั้นนั้นมีผู้ให้บริการอยู่แล้ว และยังสามารถติดต่อได้ในระยะทางไกลและเป็นการติดต่อสื่อสารแบบไร้สาย

จากการพัฒนาระบบอุปกรณ์ RAU นั้นสามารถรับสัญญาณขาเข้าได้ 2 ลักษณะ คือ ดิจิตอลและแอนาลอก และให้สัญญาณออกแบบดิจิตอลได้ โดยในการรับสัญญาณเข้าแบบดิจิตอลทำได้ 8 บิตโดยแต่ละบิตแทนสัญญาณความผิดพลาดจากตัวตรวจจับในรูป ON/OFF 1 สัญญาณ และสามารถให้สัญญาณออกแบบดิจิตอล 8 บิตเพื่อใช้ควบคุมอุปกรณ์ภายนอกได้โดย 1 บิตแทนการใช้งานกับอุปกรณ์ภายนอก 1 ตัว ส่วนการรับสัญญาณเข้าแบบแอนาลอกสามารถรับได้ 2 แบบคือแบบ 0-5V 1 ช่องสัญญาณ และแบบ 4-20mA 1 ช่องสัญญาณโดยมีความละเอียดในการประมาณค่า 12 บิต

RAU นี้ได้มีการออกแบบให้สามารถทำการปรับค่าได้โดยการโหลดจากคอมพิวเตอร์ผ่านทางโปรแกรมที่ได้มีการพัฒนาขึ้น และยังสามารถปรับค่าได้จากทางการสั่งการโดยข่าวสารสั้นอีกด้วย โดยในการออกแบบ RAU นี้ได้มีการกำหนดคำสั่งมาตรฐานในการสั่งการทำงานต่างๆของ RAU ไว้ทั้งหมด 8 คำสั่ง สามารถมีผู้ใช้งานในระบบได้ 10 เลขหมาย สามารถกำหนดชื่อให้พอร์ตต่างๆได้โดยมีความยาวไม่เกิน 8 ตัวอักษร และความยาวของข่าวสารสั้นที่ใช้ในการแจ้งเตือนต้องไม่เกิน 25 ตัวอักษร

จากการทดสอบการทำงานของระบบพบว่าตัว RAU สามารถทำงานในการตรวจสอบความผิดพลาดจากสัญญาณดิจิตอลขาเข้าและแอนาลอกขาเข้าได้ โดยมีรอบการทำงานในการตรวจสอบ 1 ครั้งประมาณ 160 μs และยังสามารถแจ้งเตือนให้กับผู้ใช้งานระบบได้ รวมทั้งยังสามารถรายงานสถานะภาพของสัญญาณต่างๆในระบบให้แก่ผู้ใช้งานเมื่อมีผู้ใช้งานร้องขอได้อีกด้วย แต่ในการประมวลผลในด้านแอนาลอกยังมีความคลาดเคลื่อนอยู่บ้าง เนื่องจากไม่ได้ใช้การคำนวณแบบ floating point ส่วนในด้านการรับคำสั่งมาดำเนินการนั้น RAU สามารถปฏิบัติตามคำสั่งที่ได้รับเป็นอย่างดี

ในส่วน Alarm Control Unit นั้นทำงานโดยการควบคุมจากโปรแกรมที่ได้พัฒนาขึ้นมาในงานวิจัย และจากการทดสอบพบว่าสามารถทำงานในการบันทึกเหตุการณ์แจ้งเตือนจาก RAU ได้เป็นอย่างดีรวมถึงสามารถใช้ส่งข่าวสารสั้นไปที่ RAU ได้

8.2 ปัญหาและข้อเสนอแนะ

ในการพัฒนาระบบแจ้งเตือนและสั่งการระยะไกลผ่านระบบบริการข่าวสารสั้นนี้มีปัญหาและข้อเสนอแนะในการพัฒนาระบบต่อไปดังนี้

1. ขนาดของอุปกรณ์ RAU นี้ยังมีขนาดใหญ่หากต้องการนำไปประยุกต์ใช้งานจริงอาจจะต้องพัฒนาให้มีขนาดเล็กลง โดยอาจจะใช้วิธีการของ FPGA ในการออกแบบเพื่อให้สามารถรวมอุปกรณ์หลายๆอย่างลงในชิฟเดียวเพื่อลดขนาดของตัววงจรได้
2. ระบบนี้สามารถนำไปประยุกต์ใช้งานระบบ SCADA ได้โดยอาจต้องมีการปรับเปลี่ยนลักษณะการทำงานใหม่ เช่น การส่งข้อมูลระหว่างอุปกรณ์วัดและหน่วยประมวลผลเป็นรอบ โดยไม่จำเป็นต้องส่งตลอดเวลา หรือมีการบีบอัดข้อมูลที่ทำการวัดได้เพื่อให้สามารถส่งข้อมูลได้จำนวนมากด้วยข่าวสารสั้นเพียงครั้งเดียว
3. ในการพัฒนาระบบนี้ในอนาคตอาจจะเพิ่มความสามารถในการส่งข่าวสารที่เป็นเสียงหรือภาพด้วยในการแจ้งเตือน โดยอาศัยการทำงานในส่วน MMS(multi media message services) ของระบบโทรศัพท์มือถือ เพื่อเพิ่มรายละเอียดของข้อมูลที่ต้องการเตือนแก่ผู้ใช้งานระบบ
4. ในการประมวลผลข้อความในข่าวสารสั้นจำเป็นต้องใช้หน่วยความจำจำนวนมาก แต่ไมโครคอนโทรลเลอร์ที่ใช้งานมีหน่วยความจำภายในเพิ่มจากตระกูล 8051 มาตรฐานเพียง 768 ไบต์ทำให้ไม่สามารถประมวลผลข้อความที่มีความยาวมากได้โดยสามารถทำได้ที่ 25 ตัวอักษร แต่เนื่องจากอุปกรณ์ที่

ได้พัฒนานั้นใช้งานในการแจ้งเตือนจึงมีความยาวของข้อความพอเพียง ดังนั้นในการพัฒนาใช้งานที่ต้องการข่าวสารสั้นที่มีความยาวมากกว่านี้จึงควรมีการเพิ่มหน่วยความจำในการประมวลผลให้แก่ RAU

5. อุปกรณ์ RAU ที่ได้ทำการพัฒนานี้ใช้ได้กับโทรศัพท์ที่สามารถรองรับคำสั่งแบบ AT-command ได้เท่านั้น แต่ในการนำไปใช้งานจริงอาจไม่ต้องใช้โทรศัพท์มือถือในการสื่อสารก็ได้ เนื่องจากในขณะมีอุปกรณ์พวก SMS-modem ซึ่งสามารถทำหน้าที่ในการส่งข่าวสารสั้น และสามารถรองรับคำสั่งแบบ AT-command ได้ในการทำงาน ซึ่งสามารถนำมาใช้งานแทนโทรศัพท์มือถือในระบบแจ้งเตือนและสั่งการทางไกลผ่านระบบบริการข่าวสารสั้นได้
6. การทำงานเกี่ยวกับข่าวสารสั้นของ RAU ที่ได้พัฒนาขึ้นนี้สามารถทำงานได้เพียงภาษาอังกฤษเท่านั้นเนื่องจากเป็นภาษามาตรฐานในการส่งข่าวสารสั้น หากต้องการพัฒนาเป็นภาษาไทยต้องมีการปรับเปลี่ยนในส่วน data coding scheme(DCS) ของข่าวสารสั้นเพื่อให้รองรับภาษาไทยได้ โดยจากการศึกษาพบว่าต้องปรับเป็น 08H จากเดิมที่เป็น 00H ในแบบภาษาอังกฤษ และจะต้องทำการเปลี่ยนลักษณะของข้อมูลใหม่ เนื่องจากการใช้งานภาษาไทยนั้นจะใช้ตัวอักษรแบบ 8 บิต ซึ่งต่างจากภาษาอังกฤษที่ใช้เพียง 7 บิต(ดังที่ได้กล่าวไปแล้วในบทที่2) และจะทำให้สามารถส่งข้อมูลได้น้อยกว่าแบบภาษาอังกฤษ

รายการอ้างอิง

1. ประภาพร ช่างไม้.คู่มือการเขียนโปรแกรมภาษาซีฉบับผู้เริ่มต้น. พิมพ์ครั้งที่ 1. บริษัทคอมฟอร์ม จำกัด 212 ม.13 ถ.กรุงเทพกรีฑา สะพานสูง กรุงเทพฯ, 2545
2. ธีรบูลย์ หล่อวิเชียร, นคร ภักดีชาติ, ชัยวัฒน์ ลิ้มพรจิตวิไล.ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษาซี. บริษัทอินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด 3133/53 ซ. สุขุมวิท 101/2 ถ.สุขุมวิท แขวงบางนา เขตบางนา กรุงเทพฯ.
3. ผศ.นุกูล กระจาย.การเขียนโปรแกรมและประมวลผลข้อมูลด้วยเทอร์โบปาสคาล.บริษัทซีเอ็ดยูเคชั่น 800/43-45 ซอยตระกูลสุข ถนนอโศก-ดินแดง เขตดินแดง กรุงเทพฯ,2539.
4. สัจจะ จรัสรุ่งรวีวร, จักรพงษ์ สุขประเสริฐ.เริ่มต้นอย่างมืออาชีพด้วย Delphi7 ฉบับสมบูรณ์. บริษัท เอซ เอ็นกรุ๊ป จำกัด 496 ซ.โชคชัยจงจำเริญ ถ.สาธุประดิษฐ์ 49 บางโพงพาง ยานนาวา กรุงเทพฯ,2546.
5. G.peersman S.R. Cvetkovic, C.smythe, Spear and P. Griffiths. The Integration of SMS with Voice Based Technology. Advances in Interactive Voice Technologies for Telecommunication Services (Digest No: 1997/147), IEE Colloquium (June 1997): 9/1 - 9/7
6. S.Collesei,P. di Tria , G. Morena. Short Message Service Based Application in the GSM Network. Indoor and Mobile Radio Communications. 1994. Wireless Networks - Catching the Mobile Future. 5th IEEE International Symposium Vol.3 (Sep 1994): 939 - 943
7. Dermot Friel , Liam Kilmartin. An Automated Stock Price Delivery Based on the GSM Short Message. Communications, 1998. ICC 98. Conference Record.1998 IEEE International Conference Vol. 3 (June 1998): 1591 – 1595.
8. ETSI GSM 03.38. Digital Cellular Telecommunication System(phase 2+) Alphabets and Language-specification Information, 1996.
9. ETSI GSM 03.40. Digital Cellular Telecommunication System(phase 2+) Technical Realization of The Short Message Service(SMS) Point-to- Point (PP),1995.
10. ETSI GSM 07.05. Digital Cellular Telecommunication System(phase 2+) Use of Data Terminal Equipment-Data Circuit terminating; Equipment (DTE-DCE) interface for Short Message Service(SMS) and Cell Broadcast Service(CBS), 1998

ภาคผนวก

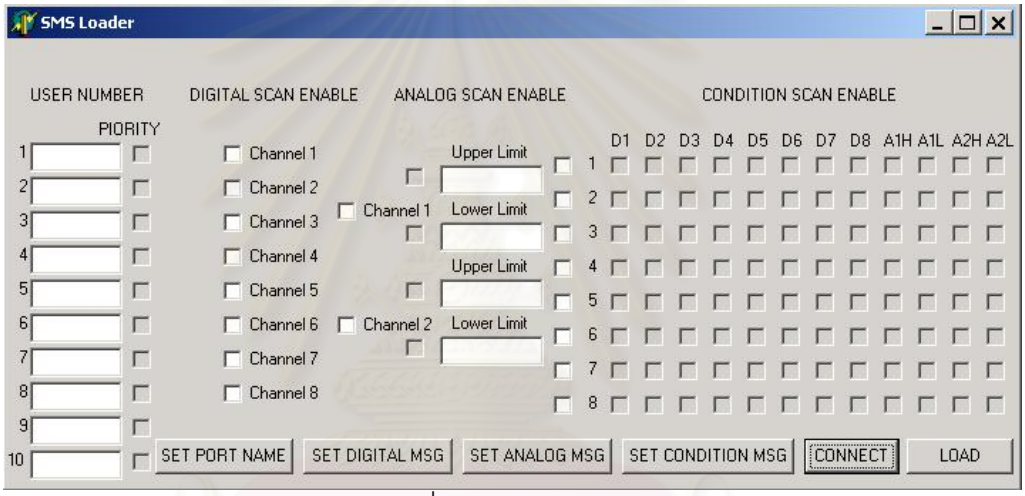
ภาคผนวก ก
คู่มือการใช้งานของระบบ

คู่มือการใช้งานระบบ

จากที่ได้กล่าวไปแล้วว่าระบบการแจ้งเตือนและสั่งการทางไกลผ่านระบบข่าวสารสั้นนี้ประกอบไปด้วยส่วนฮาร์ดแวร์ที่เป็น RAU และส่วนซอฟต์แวร์ที่ใช้งานในคอมพิวเตอร์ จึงจะได้นำเสนอวิธีการใช้งานระบบนี้ดังนี้

ก.1 การเตรียมการใช้งาน RAU

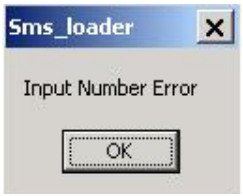
ในการใช้งาน RAU นั้นในขั้นต้นจะต้องมีการตั้งค่าการใช้งานให้กับ RAU ก่อน โดยใช้โปรแกรมที่ได้พัฒนาขึ้นในงานวิจัยนี้ชื่อว่า sms_loader ซึ่งใช้ในการโหลดข้อมูลในการทำงานให้กับ RAU โดยตัวโปรแกรมนี้มีรูปร่างดังต่อไปนี้



รูป ก.1 โปรแกรมที่ใช้ในการโหลดข้อมูลให้แก่ RAU

โดยในหน้าจอแรกจะประกอบไปด้วยการตั้งการทำงานให้กับ RAU ดังต่อไปนี้

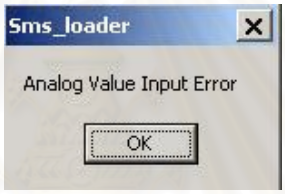
- USER NUMBER เป็นเลขหมายของผู้ที่ต้องการใช้งานในระบบนี้โดยการใส่เลขหมายแบบทั่วไป เช่น 019254675 โดยจะต้องใส่เลขหมายให้ครบ 9 ตัว มิฉะนั้นโปรแกรมจะทำการฟ้องว่ามีการป้อนเลขหมายผิดและไม่ทำการโหลดค่าต่างๆให้ โดยการใส่เลขหมายนี้ไม่จำเป็นที่จะต้องใส่ให้ครบ 10 เลขหมายก็ได้ และในส่วน PIORITY ใช้ในการเลือกว่าเลขหมายนั้นมีสิทธิในการสั่งการกับ RAU หรือไม่โดยหากทำเครื่องหมายแสดงว่ามีสิทธิสั่งการได้



รูป ก.2 ผลจากการป้อนเลขหมายผิด

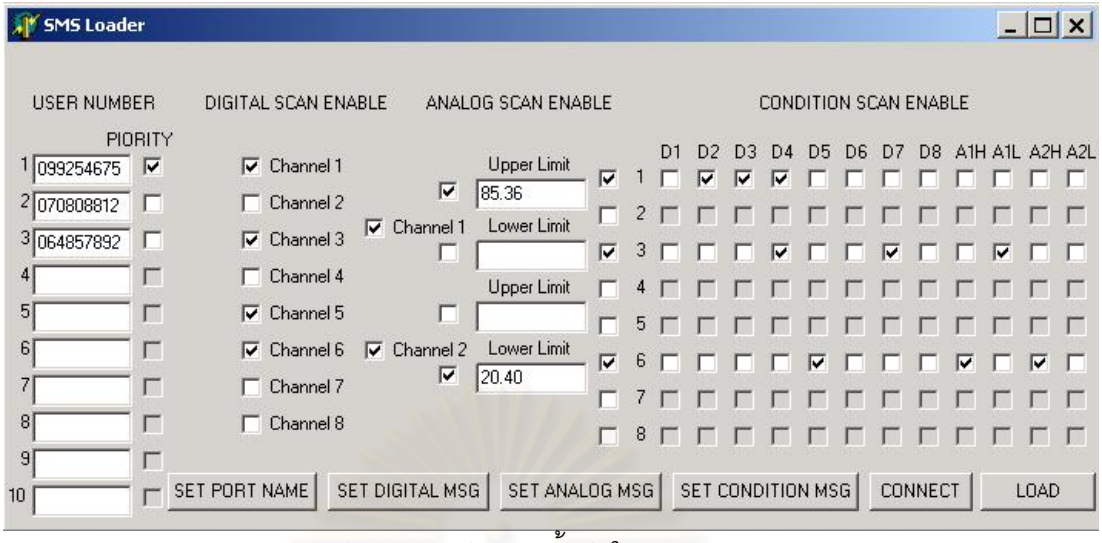
- DIGITAL SCAN ENABLE เป็นส่วนที่ใช้ในการเลือกการตรวจสอบความผิดพลาดจากช่องสัญญาณดิจิตอลขาเข้า โดยการทำเครื่องหมายหมายถึงการเลือกการตรวจสอบ

- ANALOG SCAN ENABLE เป็นส่วนที่ใช้ในการเลือกการตรวจสอบความผิดพลาดจากช่องสัญญาณแอนาลอก โดยการทำเครื่องเลือกในช่องสัญญาณที่ 1 จะเป็นการเลือกการตรวจสอบสัญญาณแบบ 0-5 V ส่วนการเลือกทำเครื่องหมายในช่องสัญญาณที่ 2 จะเป็นการเลือกการตรวจสอบสัญญาณแบบ 4-20mA เมื่อทำการเลือกช่องสัญญาณแล้วจะทำให้สามารถเลือกการตรวจสอบว่าจะตรวจสอบแบบขอบเขตบน(Upper Limit) หรือขอบเขตล่าง (Lower Limit)
ได้และต้องทำการใส่ค่าระดับที่ต้องการด้วยโดยใส่ในรูปแบบเปอร์เซนต์ โดยสามารถมีทศนิยมได้ 2 ตำแหน่ง และไม่สามารถใส่ค่าได้เกิน 100 และต่ำกว่า 0 หากมีการใส่ค่าระดับผิดทางโปรแกรมจะทำการเตือนและไม่ทำการโหลดค่าต่างๆลงใน RAU



รูป ก.3 ผลจากการป้อนค่าระดับแอนาลอกไม่ถูกต้อง

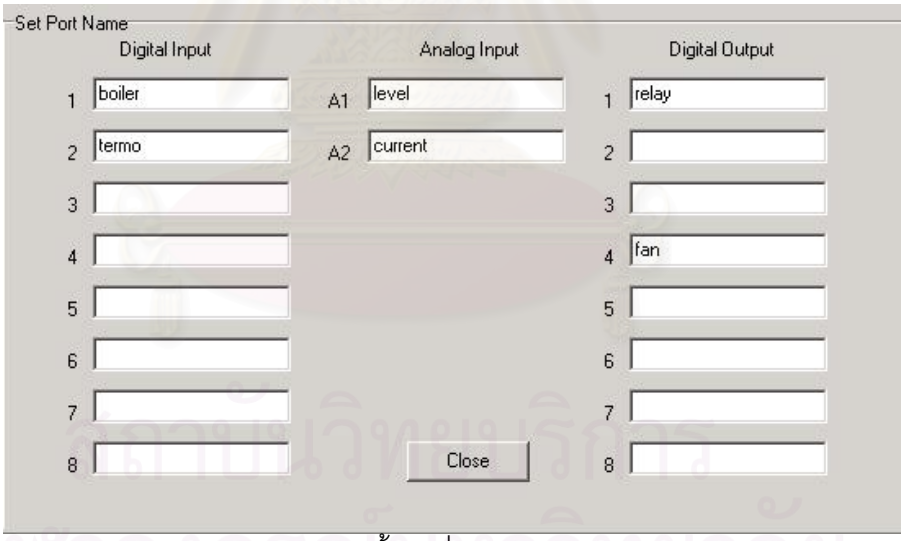
- CONDITION SCAN ENABLE เป็นส่วนที่ใช้ในการเลือกการทำงานแบบเงื่อนไขเพื่อใช้ในการแจ้งเตือน ในกรณีที่ต้องการให้มีการตรวจสอบความผิดพลาดหลายอันพร้อมกัน โดยในการเลือกต้องมีการเลือกช่องสัญญาณก่อน จากนั้นจึงทำการเลือกการตรวจสอบว่าต้องการให้มีการตรวจสอบสัญญาณใดร่วมกันบ้าง โดย D1 ถึง D8 หมายถึงช่องสัญญาณดิจิตอลขาเข้าช่องที่ 1 ถึง 8 และ A1H หมายถึงการตรวจสอบสัญญาณเกิน Upper Limit ในช่องแอนาลอกที่ 1 A1L หมายถึงการตรวจสอบสัญญาณต่ำกว่า Low Limit ในช่องสัญญาณแอนาลอกที่ 1 ส่วน A2H และ A2L นั้นจะมีความหมายเช่นเดียวกันแต่ใช้กับช่องสัญญาณแอนาลอกที่ 2



รูป ก.4 ตัวอย่างการตั้งค่าใน sms_loader

นอกจากการตั้งค่าในหน้าจอนี้แล้วยังมีส่วนการตั้งค่าในหน้าจออื่นดังนี้

- SET PORT NAME เป็นการตั้งค่าชื่อให้กับพอร์ตต่างๆของ RAU โดยเมื่อทำการกดปุ่มนี้จะมีหน้าจอใหม่ขึ้นมาดังนี้

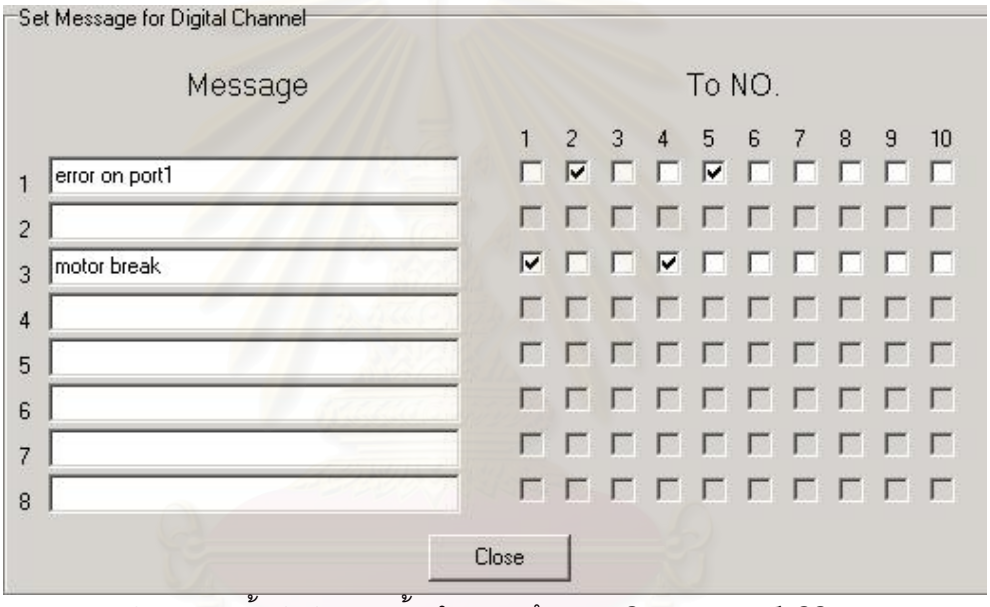


รูป ก.5 การตั้งค่าชื่อพอร์ตให้แก่ RAU

โดยในการตั้งชื่อพอร์ตนี้จะไม่สามารถตั้งได้เกิน 8 ตัวอักษร หากมีการตั้งค่ามากกว่า 8 ตัวอักษร โปรแกรมจะทำการเตือนและจะไม่ทำการโหลดค่าต่างๆ ลงใน RAU

รูป ก.6 ผลจากการป้อนชื่อพอร์ตไม่ถูกต้อง

- SET DIGITAL MSG เป็นส่วนที่ใช้ในการตั้งข่าวสารสั้นที่จะทำการแจ้งในกรณีที่เกิดตรวจพบความผิดพลาดในช่องสัญญาณดิจิตอล โดยเมื่อทำการกดปุ่มนี้จะมีหน้าจอใหม่ขึ้นมาดังนี้

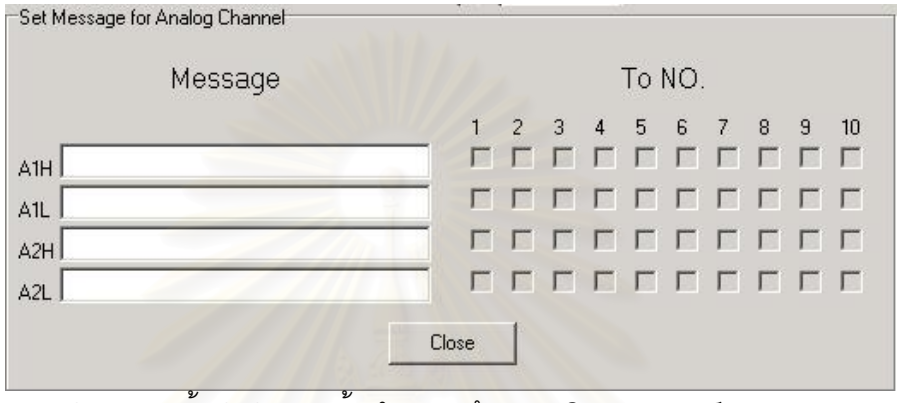


รูป ก.7 การตั้งค่าข่าวสารสั้นสำหรับแจ้งความผิดพลาดพอร์ตดิจิตอล

โดยข่าวสารสั้นจะสามารถตั้งได้เฉพาะช่องสัญญาณที่ได้มีการเลือกให้มีการตรวจสอบจาก DIGITAL SCAN ENABLE เท่านั้น และในการแจ้งเตือนยังสามารถเลือกเลขหมายของผู้ใช้งานที่ต้องการแจ้งได้ด้วยโดยการเลือกใน To NO. โดยมีลำดับ 1-10 ซึ่งอ้างอิงจากลำดับเลขหมายที่ได้มีการตั้งไว้ใน USER NUMBER และข่าวสารสั้นที่ตั้งต้องมีความยาวไม่เกิน 25 ตัวอักษรหากมีการตั้งเกินทางโปรแกรมจะทำการเตือนและไม่ทำการโหลดค่าต่างๆลงใน RAU

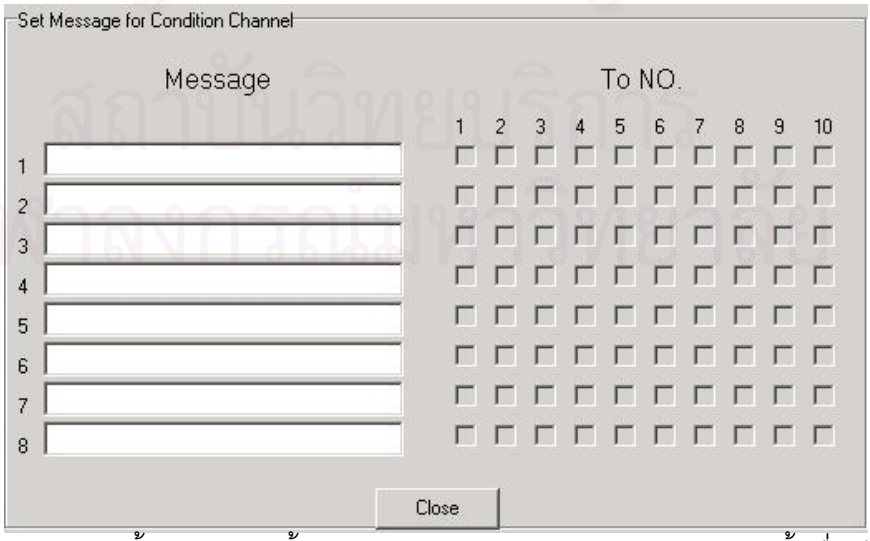
รูป ก.8 ผลจากการป้อนข่าวสารสั้นผิด

- SET ANALOG MSG เป็นส่วนที่ใช้ในการตั้งข่าวสารสั้นที่ต้องการแจ้งในกรณีที่มีตรวจพบความผิดพลาดของช่องสัญญาณแอนาลอก โดยเมื่อกดปุ่มนี้จะมีหน้าจอใหม่ออกมาดังแสดงในรูป โดยการตั้งค่าต่างๆจะทำเหมือนกับกรณี SET DIGITAL MSG และมีการตรวจสอบข่าวสารสั้นความยาวเกิน 25 ตัวอักษรเช่นเดียวกัน โดยโปรแกรมจะมีการจัดการเกี่ยวกับความผิดพลาดนี้เช่นเดียวกับกรณี SET DIGITAL MSG



รูป ก.9 การตั้งค่าข่าวสารสั้นสำหรับแจ้งความผิดพลาดพอร์ตแอนาลอก

- SET CONDITION MSG เป็นส่วนที่ใช้ในการตั้งข่าวสารสั้นที่ต้องการแจ้งในกรณีที่มีตรวจพบความผิดพลาดของการตั้งเงื่อนไข โดยเมื่อกดปุ่มนี้จะมีหน้าจอใหม่ออกมาดังแสดงในรูป โดยการตั้งค่าต่างๆจะทำเหมือนกับกรณี SET DIGITAL MSG และมีการตรวจสอบข่าวสารสั้นความยาวเกิน 25 ตัวอักษรเช่นเดียวกัน โดยโปรแกรมจะมีการจัดการเกี่ยวกับความผิดพลาดนี้เช่นเดียวกับกรณี SET DIGITAL MSG



รูป ก.10 การตั้งค่าข่าวสารสั้นสำหรับแจ้งความผิดพลาดของตรวจแบบตั้งเงื่อนไข

โดยในการเริ่มโหลดค่าให้แก่ RAU นั้นจะต้องเริ่มทำการเชื่อมต่อกับ RAU ก่อน โดยการนำ RAU มาเชื่อมต่อกับคอมพิวเตอร์ทางพอร์ต RS232 ก่อนแล้วจึงกดปุ่ม reset บนตัว RAU เพื่อเริ่มในการเชื่อมต่อ จากนั้นจึงกดปุ่ม CONNECT ในตัวโปรแกรมเพื่อทำการติดต่อ และจะมีการแสดงผลดังต่อไปนี้

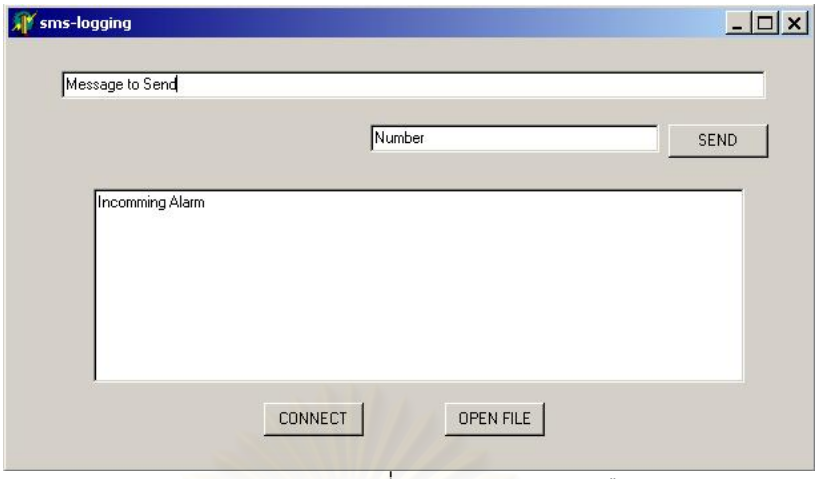
รูป ก.11 ผลการเชื่อมต่อข้อมูลกับ RAU สำเร็จ

เมื่อได้ทำการเลือกข้อมูลที่ต้องการบันทึกไว้เรียบร้อยแล้วให้ทำการกดปุ่ม LOAD เพื่อทำการโหลดค่าต่างๆลงใน RAU เมื่อโปรแกรมโหลดข้อมูลสำเร็จก็จะทำการแสดงผลดังรูป

รูป ก.12 ผลการโหลดข้อมูลลง RAU สำเร็จ

ก.2 การใช้โปรแกรมในส่วนบันทึกเหตุการณ์เตือน

โปรแกรมสำหรับการบันทึกเหตุการณ์เตือนนี้ มีไว้ใช้สำหรับการทำเป็น Alarm Control Unit ซึ่งทำหน้าที่ในการบันทึกเหตุการณ์ที่ RAU ได้ทำการเตือนไว้เป็นประวัติ และยังสามารถใช้โปรแกรมนี้ในการสั่งการไปยัง RAU ได้อีกด้วย การใช้งานโปรแกรมนี้จำเป็นต้องทำการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือเพื่อใช้งานเป็นตัวกลางในการสื่อสารโดยการต่อโทรศัพท์เข้าที่พอร์ต RS232 โปรแกรมนี้มีรูปร่างดังแสดง



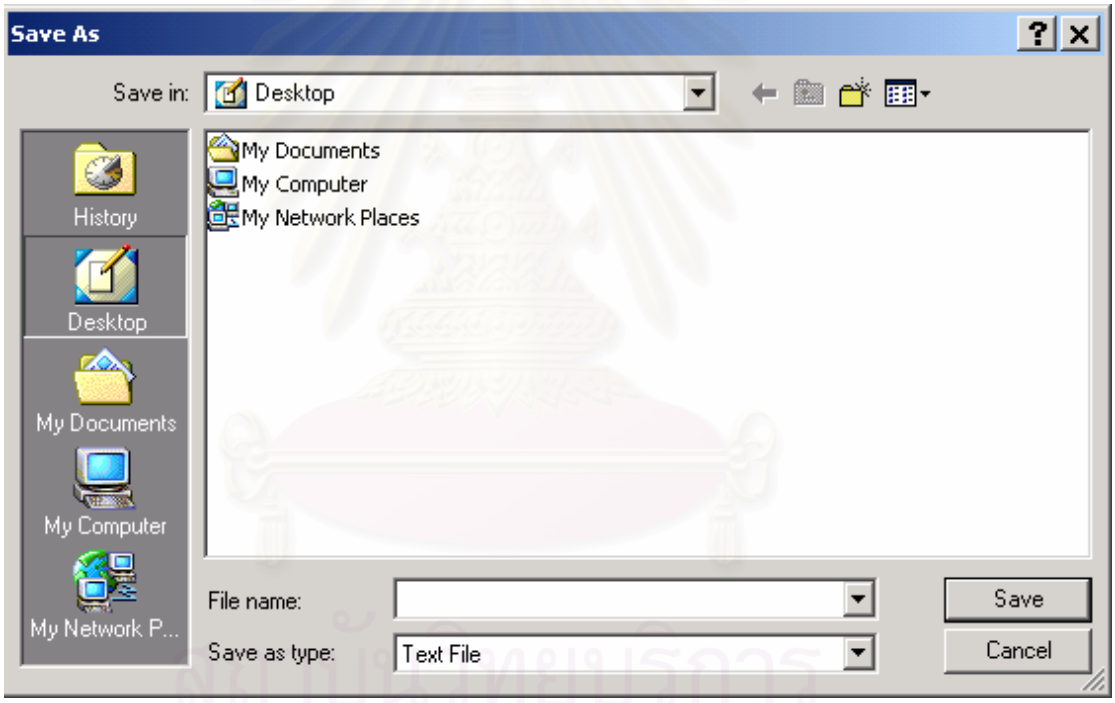
รูป ก.13 โปรแกรมที่ใช้เก็บประวัติการเตือน

จากรูปที่ ก.13 สามารถอธิบายส่วนต่างๆได้ดังต่อไปนี้

- Message to Send เป็นข่าวสารสั้นที่ต้องการส่งให้ RAU โดยต้องมีรูปแบบตามคำสั่งที่ได้มีการกล่าวถึงไปแล้วในบทที่ 4
- Send เป็นปุ่มที่ใช้ในการส่งข่าวสารสั้นไปยัง RAU โดยมีเนื้อความตาม Message to Send และส่งไปที่ RAU เลขหมายตาม Number
- Incoming Alarm เป็นส่วนที่ใช้แสดงข่าวสารสั้นที่ RAU ได้ส่งเข้ามาโดยจะแสดงข้อมูลในรูปแบบ เลขหมายผู้ส่ง เวลา และข่าวสารสั้นในแต่ละบรรทัด โดยแต่ละข่าวสารสั้นที่ RAU ส่งเข้ามาจะทำการต่อท้ายข้อมูลเดิมโดยขึ้นบรรทัดใหม่ไปเรื่อยๆ
- Connect เป็นปุ่มที่ใช้ในการเชื่อมต่อข้อมูลกับโทรศัพท์มือถือเพื่อเริ่มการทำงานของโปรแกรมโดยเมื่อเชื่อมต่อสำเร็จจะมีการแสดงผลดังรูปที่ ก.14 และหากทำการเชื่อมต่อโดยไม่มีการเลือกชื่อไฟล์ข้อมูลที่ต้องการบันทึกก็จะมีการเตือนด้วยดังรูปที่ ก.15
- Open File เป็นปุ่มที่ใช้ในการเปิดไฟล์เพื่อใช้ในการบันทึกข้อมูลประวัติการเตือน โดยในการบันทึกนั้นจะมีการบันทึกต่อกันไปเรื่อยๆในไฟล์เดิม จนกว่าจะมีการเลือกไฟล์ใหม่ จากปุ่ม Open File โดยไฟล์ที่ทำการบันทึกจะมีลักษณะเป็นไฟล์ตัวอักษร(*.txt) เท่านั้น โดยสามารถดูรูปแบบข้อมูลที่ถูกบันทึกได้จากบทที่ 7 และลักษณะการเลือกไฟล์สามารถดูได้ในรูปที่ ก.16

รูป ก.14 การเชื่อมต่อข้อมูลกับโทรศัพท์สำเร็จ

รูป ก.15 การเตือนว่าไม่มีการเลือกไฟล์ในการบันทึกการเตือน



รูป ก.16 การเลือกไฟล์ในการบันทึกการเตือน

ภาคผนวก ข
รายละเอียดภาษา C ที่ใช้ในการออกแบบ RAU

```
#include <reg51f.h>
#include <stdio.h>
#include <string.h>
bdata unsigned char word;
sbit in1 = P1^0;
sbit in2 = P1^1;
sbit in3 = P1^2;
sbit in4 = P1^3;
sbit in5 = P1^4;
sbit in6 = P1^5;
sbit in7 = P1^6;
sbit in8 = P1^7;
sbit out1 = P0^0;
sbit out2 = P0^1;
sbit out3 = P0^2;
sbit out4 = P0^3;
sbit out5 = P0^4;
sbit out6 = P0^5;
sbit out7 = P0^6;
sbit out8 = P0^7;
code char CH0=0x97,CH1=0xA3;
code unsigned char at1[]="069166";
code unsigned char
at2[]="110F0A9166";
code unsigned char at3[]="0000AD";
code char EEP_W=0xA0,EEP_R=0xA1;
bit inbit=0,outbit=0,OK=0;
code unsigned char c1[]="ON";
code unsigned char c2[]="OFF";
code unsigned char c3[]="EN";
code unsigned char c4[]="DIS";
code unsigned char c5[]="NAME";
code unsigned char c6[]="MESSAGE";
code unsigned char c7[]="READ";
code unsigned char c8[]="RESET";
code unsigned char d1[]="IN1";
code unsigned char d2[]="IN2";
code unsigned char d3[]="IN3";
code unsigned char d4[]="IN4";
code unsigned char d5[]="IN5";
code unsigned char d6[]="IN6";
code unsigned char d7[]="IN7";
code unsigned char d8[]="IN8";
code unsigned char d9[]="OUT1";
code unsigned char d10[]="OUT2";
code unsigned char d11[]="OUT3";
code unsigned char d12[]="OUT4";
code unsigned char d13[]="OUT5";
code unsigned char d14[]="OUT6";
code unsigned char d15[]="OUT7";
code unsigned char d16[]="OUT8";
code unsigned char d17[]="ANA1";
code unsigned char d18[]="ANA2";
code unsigned char d19[]="INPUT";
code unsigned char d20[]="OUTPUT";
code unsigned char s0[]="0";
code unsigned char s1[]="1";
code unsigned char s2[]="2";
code unsigned char s3[]="3";
code unsigned char s4[]="4";
code unsigned char s5[]="5";
code unsigned char s6[]="6";
```

```
code unsigned char s7[]="7";
code unsigned char s8[]="8";
code unsigned char s9[]="A1";
code unsigned char s10[]="A2";
code unsigned char s11[]=" ";
code unsigned char
s12[3]={0x0A,0x0D,0};
code unsigned char s13[]="%";
code unsigned char LOAD[]="FF";
unsigned char comm=0,port=0;
xdata unsigned char
en_D,en_A,en_B,pior1,pior2;
xdata unsigned char
buffer1[200],buffer2[200],buffer3[30];
xdata unsigned char
number1[9],number2[9],number3[9],nu
mber4[9],number5[9];
xdata unsigned char
number6[9],number7[9],number8[9],nu
mber9[9],number10[9];
xdata unsigned char
port1[9],port2[9],port3[9],port4[9],port5[
9],port6[9],port7[9],port8[9],port9[9];
xdata unsigned char
port10[9],port11[9],port12[9],port13[9],
port14[9],port15[9],port16[9],port17[9],
port18[9];
xdata unsigned char
str[15],enc[7],dec[8],num[4],hex[3],dat[
3],sms_index[4];
xdata unsigned char
csca[9],number[9];
unsigned char output[7];
unsigned char
index=0,check=0,data_in=0,proc=2,ind
=0,inc=0,check2=0,index2=0,out=0;
unsigned char addr_H,addr_L;
unsigned long
low1,high1,low2,high2,analog;
sbit SCL = P2^1;
sbit SDA = P2^0;
sbit CS = P3^3;
sbit DIN = P3^4;
sbit DOUT = P3^5;
sbit CLK = P3^2;
unsigned char find_comm(void);
unsigned char find_port(void);
void find_data(void);
void I2C_start(void);
void I2C_delay(void);
void I2C_stop(void);
void I2C_out(unsigned char dat);
void I2C_ack(unsigned char ack);
unsigned char I2C_in(void);
unsigned char hextoint (unsigned char
in1,unsigned char in2);
void decode(void);
void inttohex(unsigned char in);
void encode(void);
void inttostr(unsigned char in);
void inttofloat(unsigned long input);
```

```
void ADC (unsigned char cbyte);
void clrbuff1(void);
void clrbuff2(void);
void clrbuff3(void);
void exe(char x,char y);
void save_name(char n);
void save_mess(char m);
void read(void);
char find_no(void);
void scan(void);
void send_msg(unsigned char mo);
void download(void);
void int_routine(void) interrupt 4
{char temp;
if (proc==1){
if (TI == 1) TI = 0 ;
if (RI == 1){temp=SBUF;
switch(temp){
case 0x0A: break;
case 0x20: break;
case 0x0D: inc++;
          break;
default: buffer3[ind]=SBUF;
        ind++;}
if (ind==29) ind=0;
RI =0;}
if(inc==2) {data_in=1;
           inc=0;}}
if (proc==2){
if (TI == 1) TI = 0 ;
if (RI == 1){temp=SBUF;
switch(temp){
case 0x0A: break;
case 0x20: break;
case 0x0D: check=1;
           break;
default: buffer1[index]=SBUF;
         index++;}
if (index==199) index=0;
RI =0;}}
if(proc==3){
if (RI == 1){temp=SBUF;
switch(temp){
case 0x0A: break;
case 0x0D: check2=1;
           break;
default: buffer2[index2]=SBUF;
           index2++;}
if (index2==199) index2=0;
RI =0;}}}
void main()
{ char l,k;
  unsigned char length,count;
     proc=2;
     read();
     CS = 1;    //initial condition
                  CLK = 0;
                 TMOD = 0x20;
                 SCON = 0x50;
                  PCON = 0x00;
                  TH1 = 0xFB;
                  RI = 0;
```

```
                TI = 0;
                IE = 0x90;
                TR1 = 1;
                clrbuff1();
                clrbuff2();
                clrbuff3();
                EA=0;
                TI=1;
                printf("ate0\n");
                TI=0;
                EA=1;
w1:     while(check==0);
                check=0;
if(buffer1[strlen(buffer1)-1]==0x58)
download();
if(buffer1[strlen(buffer1)-1]!=0x4B) goto
w1;
                index=0;
                clrbuff1();
                EA=0;
                TI=1;
        printf("at+cnmi=1,1,0,0,1\n");
                TI=0;
                EA=1;
w2:     while(check==0);
                check=0;
if(buffer1[strlen(buffer1)-1]!=0x4B) goto
w2;
                index=0;
                clrbuff1();
                EA=0;
                TI=1;
                printf("at+csca?\n");
                TI=0;
                EA=1;
w3:     while(check==0);
                check=0;
                if(buffer1[strlen(buffer1)-
1]!=0x4B) goto w3;
                index=0;
                for(l=0;l<8;l++){
                k=l+10;
                csca[l]=buffer1[k];}
                csca[8]=0;
                for(l=0;l<4;l++){
                buffer2[2*l]=csca[2*l+1];
                buffer2[2*l+1]=csca[2*l];
                }
                for(l=0;l<8;l++)
                csca[l]=buffer2[l];
                clrbuff1();
                clrbuff2();
loop:           while(1){
                proc=1;
                scan();
                if(data_in==1) goto data_input;}
data_input:             data_in=0;
                for(k=0;k<4;k++)
                sms_index[k]=0;
                l=strpos(buffer3,0x2C);
            for(k=(l+1);k<strlen(buffer3);k++)
              sms_index[k-(l+1)]=buffer3[k];
```

```
                clrbuff3();
                ind=0;
                proc=2;
                EA=0;
                TI=1;
printf("at+cmgr=%s\n",sms_index);
                TI=0;
                EA=1;
                 l=0;
w5:     while(check==0);
                check=0;
                l++;
                if(l!=2) goto w5;
                clrbuff1();
                index=0;
w6:  while(check==0);
    check=0;
if(buffer1[strlen(buffer1)-1]!=0x4B) goto
w6;
                index=0;
                length=strlen(buffer1);
                for(l=22;l<30;l++)
                number[l-22]=buffer1[l];
                number[8]=0;
                k=find_no();
                if(k==0)goto pass;
                clrbuff1();
                l=0;
               EA=0;
                TI=1;
    printf("at+cmgd=%s\n",sms_index);
    TI=0;
    EA=1;
w7:     while(check==0);
                check=0;
if(buffer1[strlen(buffer1)-1]!=0x4B) goto
w7;
                clrbuff1();
                index=0;
                goto loop;
pass:   for(l=50;l<length;l++)
                buffer2[l-50]=buffer1[l];
                clrbuff1();
                length=strlen(buffer2);
                buffer2[length-1]=0;
                buffer2[length-2]=0;
                count = (length/2)/7;
    if ((length/2)%7>0) count=count+1;
                for (l=0;l<count;l++){
    for (k=0;k<7;k++)
enc[k] =
hextoint(buffer2[14*l+2*k],buffer2[14*l+
2*k+1]);
    decode();
    for (k=0;k<8;k++)
    buffer1[8*l+k] = dec[k];}
     clrbuff2();
    comm=find_comm();
    port=find_port();
     find_data();
     exe(comm,port);
```

```
            clrbuff1();
            clrbuff2();
    proc=2;
    EA=0;
    TI=1;
    printf("at+cmgd=%s\n",sms_index);
    TI=0;
    EA=1;
w8:     while(check==0);
                check=0;
if(buffer1[strlen(buffer1)-1]!=0x4B) goto
w8;
                index=0;
                clrbuff1();
                I=0;
          goto loop;}
void I2C_delay(void)
{ char temp;
for (temp=0;temp<=10;temp++);}
void I2C_start(void)
{       SDA = 1;
        SCL = 1;
        I2C_delay();
        SDA = 0;
        I2C_delay();
        SCL = 0;
        SDA = 1;
        I2C_delay();}
void I2C_stop(void)
{       I2C_delay();
        SDA = 0;
        SCL = 1;
        I2C_delay();
        SDA = 1;}
void I2C_out(unsigned char dat)
{ char z;
  for (z=1;z<=8;z++)
 {outbit = dat & 0x80;
  SDA = outbit;
  dat = dat<<1;
  SCL = 1;
  I2C_delay();
  SCL = 0;
  I2C_delay();}
  SDA = 1;
  I2C_delay();
  SCL = 1;
  outbit = SDA;
  SCL = 0;}
unsigned char I2C_in(void)
{   char y;
  word = 0;
  for (y=1;y<=8;y++)
  {I2C_delay();
  SCL = 1;
   inbit = SDA;
   word = word<<1;
   word = word | inbit;
   I2C_delay();
   SCL = 0;}
  SDA = 1;
  return(word);}
```

```
void I2C_ack(unsigned char ack)
{ I2C_delay();
  if (ack == 1) SDA = 0;
  if (ack == 0) SDA = 1;
  SCL = 1;
  I2C_delay();
  SCL = 0;
  SDA = 1;}
unsigned char hextoint (unsigned char
in1,unsigned char in2)
 {char i,temp1,temp2,temp3;
  hex[0] = in1;
  hex[1] = in2;
  for (i=0;i<=1;i++){
  switch (hex[i]){
  case 0x30: hex[i] = 0;
  break;
  case 0x31: hex[i] = 1;
  break;
  case 0x32: hex[i] = 2;
  break;
  case 0x33: hex[i] = 3;
  break;
  case 0x34: hex[i] = 4;
  break;
  case 0x35: hex[i] = 5;
  break;
  case 0x36: hex[i] = 6;
  break;
  case 0x37: hex[i] = 7;
  break;
  case 0x38: hex[i] = 8;
  break;
  case 0x39: hex[i] = 9;
  break;
  case 0x41: hex[i] = 10;
  break;
  case 0x42: hex[i] = 11;
  break;
  case 0x43: hex[i] = 12;
  break;
  case 0x44: hex[i] = 13;
  break;
  case 0x45: hex[i] = 14;
  break;
  case 0x46: hex[i] = 15;
  break;}}
  temp1 = hex[0];
  temp1 = temp1<<4;
  temp3 = 0;
  temp2 = hex[1];
  temp3 = temp2|temp1;
  return(temp3);}
  void decode(void)
  { unsigned char c,d;
dec[0] = enc[0]&0x7F;
  c = enc[0]&0x80;
  c = c>>7;
  d = enc[1]&0x3F;
  d = d<<1;
  d = d|c;
  dec[1] = d;
```

```
c = enc[1]&0xC0;
c = c>>6;
d = enc[2]&0x1F;
d = d<<2;
d = d|c;
dec[2] = d;
c = enc[2]&0xE0;
c =c>>5;
d =enc[3]&0x0F;
d = d<<3;
d = d|c;
dec[3] = d;
c = enc[3]&0xF0;
c =c>>4;
d = enc[4]&0x07;
d = d<<4;
d = d|c;
dec[4] = d;
c = enc[4]&0xF8;
c = c>>3;
d = enc[5]&0x03;
d = d<<5;
d = d|c;
dec[5] = d;
c = enc[5]&0xFC;
c = c>>2;
d = enc[6]&0x01;
d = d<<6;
d = d|c;
dec[6] = d;
dec[7] = enc[6]&0xFE;
dec[7] = dec[7]>>1;}
 void inttohex(unsigned char in)
{unsigned char i;
hex[0] = in & 0xF0;
hex[0] = hex[0]>>4;
hex[1] = in & 0x0F;
for (i=0;i<=1;i++){
switch (hex[i]){
case 0: hex[i] = 0x30;
break;
case 1: hex[i] = 0x31;
break;
case 2: hex[i] = 0x32;
break;
case 3: hex[i] = 0x33;
break;
case 4: hex[i] = 0x34;
break;
case 5: hex[i] = 0x35;
break;
case 6: hex[i] = 0x36;
break;
case 7: hex[i] = 0x37;
break;
case 8: hex[i] = 0x38;
break;
case 9: hex[i] = 0x39;
break;
case 10: hex[i] = 0x41;
break;
```

```
 case 11: hex[i] = 0x42;
break;
 case 12: hex[i] = 0x43;
 break;
 case 13: hex[i] = 0x44;
break;
 case 14: hex[i] = 0x45;
 break;
 case 15: hex[i] = 0x46;
break;}}}
void encode(void)
 {unsigned char c;
  enc[0] = dec[0];
  c = dec[1] & 0x01;
  c = c<<7;
  enc[0] = enc[0] | c;
  enc[1] = dec[1]>>1;
  c = dec[2] & 0x03;
  c = c<<6;
  enc[1] = enc[1] | c;
  enc[2] = dec[2]>>2;
  c = dec[3] & 0x07;
  c = c<<5;
  enc[2] = enc[2] | c;
  enc[3] = dec[3]>>3;
 c = dec[4] & 0x0F;
  c = c<<4;
  enc[3] = enc[3] | c;
  enc[4] = dec[4] >>4;
  c = dec[5] & 0x1F;
  c = c<<3;
  enc[4] = enc[4] | c;
  enc[5] = dec[5] >>5;
   c = dec[6] & 0x3F;
  c = c<<2;
  enc[5] = enc[5] | c;
  enc[6] = dec[6] >>6;
  c = dec[7] & 0x7F;
  c = c<<1;
  enc[6] = enc[6] | c;}
void inttostr(unsigned char in)
{ unsigned char a[3],i;
  char b;
          a[0] = in/100;
          b = in % 100;
          a[1] = b/10;
          a[2] = b % 10;
          for (i=0;i<=2;i++)
          {switch (a[i]){
          case 0: dat[i] = 0x30;
          break;
          case 1: dat[i] = 0x31;
          break;
         case 2: dat[i] = 0x32;
          break;
         case 3: dat[i] = 0x33;
          break;
         case 4: dat[i] = 0x34;
          break;
         case 5: dat[i] = 0x35;
          break;
         case 6: dat[i] = 0x36;
```

```
            break;
            case 7: dat[i] = 0x37;
            break;
            case 8: dat[i] = 0x38;
            break;
            case 9: dat[i] = 0x39;
            break; }}
            for(i=0;i<4;i++)
            num[i]=0;
            b=0;
            for(i=0;i<2;i++){
            if(dat[i]==0x30) b++;
            else break;}
            if(b==0) {for(i=0;i<3;i++)
            num[i]=dat[i];}
            if(b==1) {num[0]=dat[1];
            num[1]=dat[2];}
            if(b==2) num[0]=dat[2];

            }
void ADC (unsigned char cbyte)
{ unsigned char addr,i;
  bit inbit,outbit;
  unsigned long temp;
      addr = cbyte;
      word = 0;
      CS = 0;
      CLK = 0;
      for(i=0;i<=7;i++){
      outbit = addr & 0x80;
      DIN = outbit;
      CLK = 1;
      CLK = 0;
      addr = addr<<1;}
      CLK = 1;
      CLK = 0;
      for(i=0;i<=7;i++){
      inbit = DOUT;
      CLK = 1;
      CLK = 0;
      word = word<<1;
      word = word | inbit;}
      analog=word;
      analog=analog<<4;
      word = 0;
      for(i=0;i<=7;i++){
      inbit = DOUT;
      CLK = 1;
      CLK = 0;
      word = word<<1;
      word = word | inbit;}
      word=word>>4;
      analog=analog|word;
      CS = 1;
      if(cbyte==0xA3) {temp=analog*5;
      temp=temp/4;
      temp=temp-0x400;
      analog=temp&0xFFF;}}
void clrbuff1(void){
unsigned char i;
for(i=0;i<200;i++)
buffer1[i]=0;}
```

```
void clrbuff2(void){
unsigned char i;
for(i=0;i<200;i++)
buffer2[i]=0;}
void clrbuff3(void){
unsigned char i;
for(i=0;i<30;i++)
buffer3[i]=0;}
unsigned char find_comm(void)
{  char l,k,m;
     l=strpos(buffer1,0x20);
    if(l==-1) goto quit;
    for(k=0;k<8;k++)
    dec[k]=0;
    for(k=0;k<l;k++)
    dec[k]=buffer1[k];
    m=strcmp(dec,c1);
    if(m==0) return(1);
    m=strcmp(dec,c2);
    if(m==0) return(2);
    m=strcmp(dec,c3);
    if(m==0) return(3);
   m=strcmp(dec,c4);
    if(m==0) return(4);
   m=strcmp(dec,c5);
    if(m==0) return(5);
   m=strcmp(dec,c6);
    if(m==0) return(6);
   m=strcmp(dec,c7);
    if(m==0) return(7);
quit: m=strcmp(dec,c7);
     if(m==0) return(7);
     m=strcmp(dec,c8);
     if(m==0) return(8);
     return(0);}
unsigned char find_port(void)
{ char l,k,z;
    l=strpos(buffer1,0x20);
    if(l==-1) goto quit;
    buffer1[l]=0x3B;
    for(k=(l+1);k<strlen(buffer1);k++)
    buffer2[k-(l+1)]=buffer1[k];
    l=strpos(buffer2,0x20);
    if(l==-1) goto not;
    goto yes;
yes: for(k=l;k<strlen(buffer2);k++)
    buffer2[k]=0;
not: z=strcmp(buffer2,d1);
    if(z==0) return(1);
    z=strcmp(buffer2,d2);
    if(z==0) return(2);
    z=strcmp(buffer2,d3);
    if(z==0) return(3);
    z=strcmp(buffer2,d4);
    if(z==0) return(4);
    z=strcmp(buffer2,d5);
    if(z==0) return(5);
    z=strcmp(buffer2,d6);
    if(z==0) return(6);
    z=strcmp(buffer2,d7);
    if(z==0) return(7);
```

```
        z=strcmp(buffer2,d8);
if(z==0) return(8);
        z=strcmp(buffer2,d9);
if(z==0) return(9);
        z=strcmp(buffer2,d10);
if(z==0) return(10);
        z=strcmp(buffer2,d11);
if(z==0) return(11);
        z=strcmp(buffer2,d12);
if(z==0) return(12);
        z=strcmp(buffer2,d13);
if(z==0) return(13);
        z=strcmp(buffer2,d14);
if(z==0) return(14);
        z=strcmp(buffer2,d15);
if(z==0) return(15);
        z=strcmp(buffer2,d16);
if(z==0) return(16);
        z=strcmp(buffer2,d17);
if(z==0) return(17);
       z=strcmp(buffer2,d18);
if(z==0) return(18);
        z=strcmp(buffer2,d19);
if(z==0) return(19);
        z=strcmp(buffer2,d20);
if(z==0) return(20);
      z=strcmp(buffer2,port1);
if(z==0) return(1);
      z=strcmp(buffer2,port2);
if(z==0) return(2);
      z=strcmp(buffer2,port3);
if(z==0) return(3);
      z=strcmp(buffer2,port4);
if(z==0) return(4);
      z=strcmp(buffer2,port5);
if(z==0) return(5);
      z=strcmp(buffer2,port6);
if(z==0) return(6);
      z=strcmp(buffer2,port7);
if(z==0) return(7);
      z=strcmp(buffer2,port8);
if(z==0) return(8);
      z=strcmp(buffer2,port9);
if(z==0) return(9);
      z=strcmp(buffer2,port10);
if(z==0) return(10);
      z=strcmp(buffer2,port11);
if(z==0) return(11);
      z=strcmp(buffer2,port12);
if(z==0) return(12);
      z=strcmp(buffer2,port13);
if(z==0) return(13);
      z=strcmp(buffer2,port14);
if(z==0) return(14);
      z=strcmp(buffer2,port15);
if(z==0) return(15);
      z=strcmp(buffer2,port16);
if(z==0) return(16);
      z=strcmp(buffer2,port17);
if(z==0) return(17);
      z=strcmp(buffer2,port18);
if(z==0) return(18);
```

```
    quit: return(0);}
void find_data(void)
{ char a,b;
  clrbuff2();
  b=strpos(buffer1,0x20);
  if(b==-1) goto quit;
  for(a=(b+1);a<strlen(buffer1);a++)
  buffer2[a-(b+1)]=buffer1[a];
  quit:;}
void exe(char x,char y)
{  unsigned char i,j,c,PDU,done=0x1A;
   unsigned char div,mod;
   unsigned long temp;
   bit high;
   switch(x){
  case 1: if(!OK) break;
        switch(y){
        case 9: out=out | 0x01;
        out1=1;
        break;
        case 10: out=out | 0x02;
        out2=1;
        break;
        case 11: out=out | 0x04;
        out3=1;
        break;
        case 12: out=out | 0x08;
        out4=1;
        break;
        case 13: out=out | 0x10;
        out5=1;
        break;
        case 14: out=out | 0x20;
        out6=1;
        break;
        case 15: out=out | 0x40;
        out7=1;
        break;
        case 16: out=out | 0x80;
         out8=1;
        break;}
        break;
case 2: if(!OK) break;
        switch(y){
        case 9:  out=out & 0xFE;
        out1=0;
        break;
        case 10: out=out & 0xFD;
        out2=0;
        break;
        case 11: out=out & 0xFB;
        out3=0;
        break;
        case 12: out=out & 0xF7;
        out4=0;
        break;
        case 13: out=out & 0xEF;
        out5=0;
        break;
        case 14: out=out & 0xDF;
         out6=0;
```

```
        break;
        case 15: out=out & 0xBF;
        out7=0;
        break;
        case 16: out=out & 0x7F;
        out8=0;
        break;}
        break;
    case 3: if(!OK) break;
                    switch(y){
        case 1: en_D=en_D|0x01;
        break;
        case 2: en_D=en_D|0x02;
        break;
        case 3: en_D=en_D|0x04;
        break;
        case 4: en_D=en_D|0x08;
        break;
        case 5: en_D=en_D|0x10;
        break;
        case 6: en_D=en_D|0x20;
        break;
        case 7: en_D=en_D|0x40;
        break;
        case 8: en_D=en_D|0x80;
        break;}
en1:    I2C_start();
        I2C_out(EEP_W);
        if (outbit) goto en1;
        I2C_out(0x01);
        if (outbit) goto en1;
        I2C_out(0xA6);
        if (outbit) goto en1;
        I2C_out(en_D);
        if (outbit) goto en1;
        I2C_stop();
        break;
    case 4: if(!OK) break;
                    switch(y){
        case 1: en_D=en_D&0xFE;
        break;
        case 2: en_D=en_D&0xFD;
        break;
        case 3: en_D=en_D&0xFB;
        break;
        case 4: en_D=en_D&0xF7;
        break;
        case 5: en_D=en_D&0xEF;
        break;
        case 6: en_D=en_D&0xDF;
        break;
        case 7: en_D=en_D&0xBF;
        break;
        case 8: en_D=en_D&0x7F;
        break;}
en2:    I2C_start();
        I2C_out(EEP_W);
        if (outbit) goto en2;
        I2C_out(0x01);
        if (outbit) goto en2;
        I2C_out(0xA6);
        if (outbit) goto en2;
```

```
        I2C_out(en_D);
        if (outbit) goto en2;
        I2C_stop();
            break;
case 5: if(!OK) break;
      save_name(y);
      read();
      break;
case 6: if(!OK) break;
      save_mess(y);
      break;
case 7: clrbuff1();
       clrbuff2();
       if(y==0){
       strcat(buffer1,d19);
       strcat(buffer1,s11);
       if(in1==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in2==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in3==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in4==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in5==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in6==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in7==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       if(in8==1) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       strcat(buffer1,s12);
       strcat(buffer1,d20);
       strcat(buffer1,s11);
       high=out&0x01;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x02;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x04;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x08;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x10;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x20;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x40;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       high=out&0x80;
       if(high) strcat(buffer1,s1);
       else strcat(buffer1,s0);
       strcat(buffer1,s12);
       strcat(buffer1,s9);
       strcat(buffer1,s11);
```

```
ADC(CH0);
inttofloat(analog);
strcat(buffer1,output);
strcat(buffer1,s13);
strcat(buffer1,s12);
strcat(buffer1,s10);
strcat(buffer1,s11);
ADC(CH1);
inttofloat(analog);
strcat(buffer1,output);
strcat(buffer1,s13);
strcat(buffer1,s12);
EA=0;
TI=1;
printf("%s",buffer1);
TI=0;
EA=1;}
if(y==19){
strcat(buffer1,s1);
strcat(buffer1,s11);
strcat(buffer1,port1);
strcat(buffer1,s11);
if(in1==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s2);
strcat(buffer1,s11);
strcat(buffer1,port2);
strcat(buffer1,s11);
if(in2==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s3);
strcat(buffer1,s11);
strcat(buffer1,port3);
strcat(buffer1,s11);
if(in3==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s4);
strcat(buffer1,s11);
strcat(buffer1,port4);
strcat(buffer1,s11);
if(in4==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s5);
strcat(buffer1,s11);
strcat(buffer1,port5);
strcat(buffer1,s11);
if(in5==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s6);
strcat(buffer1,s11);
strcat(buffer1,port6);
strcat(buffer1,s11);
if(in6==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s7);
strcat(buffer1,s11);
```

```
strcat(buffer1,port7);
strcat(buffer1,s11);
if(in7==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s8);
strcat(buffer1,s11);
strcat(buffer1,port8);
strcat(buffer1,s11);
if(in8==1) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s9);
strcat(buffer1,s11);
strcat(buffer1,port17);
strcat(buffer1,s11);
ADC(CH0);
inttofloat(analog);
strcat(buffer1,output);
strcat(buffer1,s13);
strcat(buffer1,s12);
strcat(buffer1,s10);
strcat(buffer1,s11);
strcat(buffer1,port18);
strcat(buffer1,s11);
ADC(CH1);
inttofloat(analog);
strcat(buffer1,output);
strcat(buffer1,s13);
strcat(buffer1,s12);
EA=0;
TI=1;
printf("%s",buffer1);
TI=0;
EA=1;}
if(y==20){
strcat(buffer1,s1);
strcat(buffer1,s11);
strcat(buffer1,port9);
strcat(buffer1,s11);
high=out&0x01;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s2);
strcat(buffer1,s11);
strcat(buffer1,port10);
strcat(buffer1,s11);
high=out&0x02;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s3);
strcat(buffer1,s11);
strcat(buffer1,port11);
strcat(buffer1,s11);
high=out&0x04;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s4);
strcat(buffer1,s11);
```

```
strcat(buffer1,port12);
strcat(buffer1,s11);
high=out&0x08;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s5);
strcat(buffer1,s11);
strcat(buffer1,port13);
strcat(buffer1,s11);
high=out&0x10;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s6);
strcat(buffer1,s11);
strcat(buffer1,port14);
strcat(buffer1,s11);
high=out&0x20;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s7);
strcat(buffer1,s11);
strcat(buffer1,port15);
strcat(buffer1,s11);
high=out&0x40;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
strcat(buffer1,s8);
strcat(buffer1,s11);
strcat(buffer1,port16);
strcat(buffer1,s11);
high=out&0x80;
if(high) strcat(buffer1,c1);
else strcat(buffer1,c2);
strcat(buffer1,s12);
EA=0;
TI=1;
printf("%s",buffer1);
TI=0;
EA=1;}
c=strlen(buffer1);
inttohex(c);
hex[2]=0;
div=strlen(buffer1)/8;
mod=strlen(buffer1)%8;
temp=div*14;
temp=temp+40;
temp=temp+(mod*2);
PDU=temp/2;
inttostr(PDU);
clrbuff2();
proc=3;
EA=0;
TI=1;
printf("at+cmgs=%s\n",num);
TI=0;
EA=1;
while(check2==0);
check2=0;
```

```
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r);
 printf("%s%s",at3,hex);
    str[14]=0;
        for(j=0;j<div;j++){
        for(i=0;i<8;i++)
        dec[i] = buffer1[8*j+i];
    encode();
 for (i=0;i<7;i++){
 inttohex(enc[i]);
 str[2*i]=hex[0];
 str[2*i+1]=hex[1];}
 printf("%s",str);}
 if(mod>0) {
 c=8*div;
 for(i=c;i<c+8;i++)
 dec[i-c] = buffer1[i];
 encode();
 for (i=0;i<7;i++){
 inttohex(enc[i]);
 str[2*i]=hex[0];
 str[2*i+1]=hex[1];}
 c=mod*2;
 for(i=c;i<15;i++)
 str[i]=0;
 printf("%s",str);}
 printf("%c",done);
                TI=0;
                EA=1;
s1:     while(check2==0);
                check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s1;
                index2=0;
                clrbuff2();
                break;
 case 8: if(!OK) break;
read_en: I2C_start();
                I2C_out(EEP_W);
                if (outbit) goto read_en;
                I2C_out(0x01);
                if (outbit) goto read_en;
                I2C_out(0xA6);
                if (outbit) goto read_en;
                I2C_start();
                I2C_out(EEP_R);
                if (outbit) goto read_en;
                en_D=I2C_in();
                I2C_ack(1);
                en_A=I2C_in();
                I2C_ack(1);
                c=I2C_in();
                I2C_ack(1);
                en_B=I2C_in();
                I2C_ack(0);
                I2C_stop();
                break;}}
```

```
void save_name(char n)
{ unsigned char addr,i;
  if(n==8) addr=0x96;
  else if(n==16) addr=0x9E;
  else addr=6+8*(n-1);
write:  I2C_start();
            I2C_out(EEP_W);
            if (outbit) goto write;
                  I2C_out(0x01);
            if (outbit) goto write;
                  I2C_out(addr);
            if (outbit) goto write;
                  for(i=0;i<8;i++){
        I2C_out(buffer2[i]);
            if (outbit) goto write;}
                  I2C_stop();}
void save_mess(char m)
{  unsigned char i;
      switch(m){
      case 1: addr_H=0x02;
            addr_L=0x00;
            break;
      case 2: addr_H=0x02;
            addr_L=0x50;
            break;
      case 3: addr_H=0x02;
            addr_L=0xA0;
            break;
      case 4: addr_H=0x02;
            addr_L=0xC8;
            break;
      case 5: addr_H=0x03;
            addr_L=0x18;
            break;
      case 6: addr_H=0x03;
            addr_L=0x90;
            break;
      case 7: addr_H=0x03;
            addr_L=0xE0;
            break;
      case 8: addr_H=0x04;
            addr_L=0x08;
            break;}

write:  I2C_start();
            I2C_out(EEP_W);
            if (outbit) goto write;
            I2C_out(addr_H);
            if (outbit) goto write;
            I2C_out(addr_L);
            if (outbit) goto write;
            for(i=0;i<40;i++){
            I2C_out(buffer2[i]);
            if (outbit) goto write;}
                  I2C_stop();}
void read(void)
{   unsigned char b,c;
 read:  I2C_start();
            I2C_out(EEP_W);
            if (outbit) goto read;
            I2C_out(0x00);
            if (outbit) goto read;
```

```
                I2C_out(0x00);
                if (outbit) goto read;
                I2C_start();
                I2C_out(EEP_R);
                if (outbit) goto read;
                pior1=I2C_in();
                I2C_ack(1);
                if (outbit) goto read;
                pior2=I2C_in();
                I2C_ack(1);
                for(b=0;b<8;b++){
                number1[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                number2[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                number3[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                number4[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<7;b++){
                number5[b]=I2C_in();
                        I2C_ack(1);}
                number5[7]=I2C_in();
                        I2C_ack(0);
                        I2C_stop();
read2:  I2C_start();
                        I2C_out(EEP_W);
                if (outbit) goto read2;
                I2C_out(0x00);
                if (outbit) goto read2;
                I2C_out(0x2A);
                if (outbit) goto read2;
                I2C_start();
                I2C_out(EEP_R);
                if (outbit) goto read2;
                for(b=0;b<8;b++){
                number6[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                number7[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                        c=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                number9[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                number10[b]=I2C_in();
                        I2C_ack(1);}
                        c=I2C_in();
                        low1=c<<4;
                        I2C_ack(1);
                        c=I2C_in();
                        c=c>>4;
                        low1=low1|c;
                        I2C_ack(1);
                        c=I2C_in();
                        high1=c<<4;
```

```
                I2C_ack(1);
                c=I2C_in();
                c=c>>4;
                high1=high1|c;
                I2C_ack(1);
                c=I2C_in();
                low2=c<<4;
                I2C_ack(1);
                c=I2C_in();
                c=c>>4;
                low2=low2|c;
                I2C_ack(1);
                c=I2C_in();
                high2=c<<4;
                I2C_ack(1);
                c=I2C_in();
                c=c>>4;
                high2=high2|c;
                I2C_ack(1);
            for(b=0;b<7;b++){
            number8[b]=I2C_in();
                I2C_ack(1);}
            number8[7]=I2C_in();
                I2C_ack(0);
                I2C_stop();
read3:   I2C_start();
            I2C_out(EEP_W);
            if (outbit) goto read3;
            I2C_out(0x01);
            if (outbit) goto read3;
            I2C_out(0x06);
            if (outbit) goto read3;
            I2C_start();
            I2C_out(EEP_R);
            if (outbit) goto read3;
            for(b=0;b<8;b++){
            port1[b]=I2C_in();
                I2C_ack(1);}
            for(b=0;b<8;b++){
            port2[b]=I2C_in();
                I2C_ack(1);}
            for(b=0;b<8;b++){
            port3[b]=I2C_in();
                I2C_ack(1);}
            for(b=0;b<8;b++){
            port4[b]=I2C_in();
                I2C_ack(1);}
            for(b=0;b<8;b++){
            port5[b]=I2C_in();
                I2C_ack(1);}
            for(b=0;b<8;b++){
            port6[b]=I2C_in();
                I2C_ack(1);}
            for(b=0;b<7;b++){
            port7[b]=I2C_in();
                I2C_ack(1);}
            port7[7]=I2C_in();
                I2C_ack(0);
                I2C_stop();
read4:   I2C_start();
                I2C_out(EEP_W);
            if (outbit) goto read4;
```

```
                I2C_out(0x01);
                if (outbit) goto read4;
                I2C_out(0x46);
                if (outbit) goto read4;
                I2C_start();
                I2C_out(EEP_R);
                if (outbit) goto read4;
                for(b=0;b<8;b++){
                port9[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                port10[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                port11[b]=I2C_in();
                        I2C_ack(1);}
              for(b=0;b<8;b++){
                port12[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                port13[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<7;b++){
                port14[b]=I2C_in();
                        I2C_ack(1);}
                port14[7]=I2C_in();
                        I2C_ack(0);
                        I2C_stop();
read5:   I2C_start();
                I2C_out(EEP_W);
                if (outbit) goto read5;
                I2C_out(0x01);
                if (outbit) goto read5;
                I2C_out(0x76);
                if (outbit) goto read5;
                I2C_start();
                I2C_out(EEP_R);
                if (outbit) goto read5;
                for(b=0;b<8;b++){
                port15[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                        c=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                port17[b]=I2C_in();
                        I2C_ack(1);}
              for(b=0;b<8;b++){
                port18[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                port8[b]=I2C_in();
                        I2C_ack(1);}
                for(b=0;b<8;b++){
                port16[b]=I2C_in();
                        I2C_ack(1);}
                        en_D=I2C_in();
                        I2C_ack(1);
                        en_A=I2C_in();
                        I2C_ack(1);
                        c=I2C_in();
                        I2C_ack(1);
```

```
en_B=I2C_in();
I2C_ack(0);
I2C_stop();
number1[8]=0;
number2[8]=0;
number3[8]=0;
number4[8]=0;
number5[8]=0;
number6[8]=0;
number7[8]=0;
number8[8]=0;
number9[8]=0;
number10[8]=0;
port1[8]=0;
port2[8]=0;
port3[8]=0;
port4[8]=0;
port5[8]=0;
port6[8]=0;
port7[8]=0;
port8[8]=0;
port9[8]=0;
port10[8]=0;
port11[8]=0;
port12[8]=0;
port13[8]=0;
port14[8]=0;
port15[8]=0;
port16[8]=0;
port17[8]=0;
port18[8]=0;}
```

```
char find_no(void)
{  unsigned char result;
    result=strcmp(number,number1);
   if (result==0){
   OK=0x01&pior1;
   goto exit;}
   result=strcmp(number,number2);
   if(result==0){
   OK=0x02&pior1;
   goto exit;}
   result=strcmp(number,number3);
   if(result==0){
   OK=0x04&pior1;
   goto exit;}
result=strcmp(number,number4);
   if(result==0){
   OK=0x08&pior1;
   goto exit;}
result=strcmp(number,number5);
   if(result==0){
   OK=0x10&pior1;
   goto exit;}
result=strcmp(number,number6);
   if(result==0){
   OK=0x01&pior2;
   goto exit;}
result=strcmp(number,number7);
   if(result==0){
   OK=0x02&pior2;
   goto exit;}
result=strcmp(number,number8);
```

```
  if(result==0){
  OK=0x04&pior2;
  goto exit;}
result=strcmp(number,number9);
  if(result==0){
  OK=0x08&pior2;
  goto exit;}
result=strcmp(number,number10);
  if(result==0){
  OK=0x10&pior2;
  goto exit;}
exit: return(result);}
void scan(void)
{  bit
test,enable,b1,b2,b3,b4,b5,b6,b7,b8,ch
0_L,ch0_H,ch1_L,ch1_H,pass;
  unsigned char
mode,i,logic1,logic2,temp;
   enable=0x01&en_D;
        if(enable){
        test=in1;
        if(test==0){mode=1;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xFE;}}
        enable=0x02&en_D;
        if(enable){
  test=in2;
  if(test==0){mode=2;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xFD;}}
        enable=0x04&en_D;
        if(enable){
test=in3;
if(test==0){mode=3;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xFB;}}
        enable=0x08&en_D;
        if(enable){
test=in4;
if(test==0){mode=4;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xF7;}}
        enable=0x10&en_D;
        if(enable){
test=in5;
if(test==0){mode=5;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xEF;}}
        enable=0x20&en_D;
        if(enable){
test=in6;
if(test==0){mode=6;
```

```
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xDF;}}
        enable=0x40&en_D;
        if(enable){
test=in7;
if(test==0){mode=7;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0xBF;}}
        enable=0x80&en_D;
        if(enable){
test=in8;
if(test==0){mode=8;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_D=en_D&0x7F;}}
        enable=0x01&en_A;
        if(enable){
        ADC(CH0);
        if(analog<low1){mode=9;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_A=en_A&0xFE;}}
        enable=0x02&en_A;
        if(enable){
        ADC(CH0);
        if(analog>high1){mode=10;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_A=en_A&0xFD;}}

        enable=0x04&en_A;
        if(enable){
        ADC(CH1);
        if(analog<low2){mode=11;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_A=en_A&0xFB;}}
        enable=0x08&en_A;
        if(enable){
        ADC(CH1);
        if(analog>high2){mode=12;
        send_msg(mode);
        clrbuff1();
        clrbuff2();
        en_A=en_A&0xF7;}}
          temp=en_B;
          for(i=0;i<8;i++){
          test=temp&0x01;
          if(test){
          addr_L=i*2;
b:  I2C_start();
              I2C_out(EEP_W);
              if (outbit) goto b;
              I2C_out(0x0A);
```

```
if (outbit) goto b;
I2C_out(addr_L);
if (outbit) goto b;
I2C_start();
I2C_out(EEP_R);
if (outbit) goto b;
logic1 = I2C_in();
I2C_ack(1);
logic2 = I2C_in();
I2C_ack(0);
I2C_stop();
pass=logic1&0x01;
if(pass) b1=!in1;
else b1=1;
pass=logic1&0x02;
if(pass) b2=!in2;
else b2=1;
pass=logic1&0x04;
if(pass) b3=!in3;
else b3=1;
pass=logic1&0x08;
if(pass) b4=!in4;
else b4=1;
pass=logic1&0x10;
if(pass) b5=!in5;
else b5=1;
pass=logic1&0x20;
if(pass) b6=!in6;
else b6=1;
pass=logic1&0x40;
if(pass) b7=!in7;
else b7=1;
pass=logic1&0x80;
if(pass) b8=!in8;
else b8=1;
ch0_L=1;
pass=logic2&0x01;
if(pass){
ADC(CH0);
if (analog<low1) ch0_L=1;
else ch0_L=0;}
ch0_H=1;
pass=logic2&0x02;
if(pass){
ADC(CH0);
if (analog>high1) ch0_H=1;
else ch0_H=0;}
ch1_L=1;
pass=logic2&0x04;
if(pass){
ADC(CH1);
if(analog<low2) ch1_L=1;
else ch1_L=0;}
ch1_H=1;
pass=logic2&0x08;
if(pass){
ADC(CH1);
if(analog>high2) ch1_H=1;
else ch1_H=0;}
enable=b1&b2&b3&b4&b5&b6
&b7&b8&ch0_L&ch0_H&ch1_L,ch1_H;
if(enable) {mode=13+i;
```

```
                send_msg(mode);
                clrbuff1();
          clrbuff2();}
                switch(mode){
        case 13: en_B=en_B&0xFE;
                break;
        case 14: en_B=en_B&0xFD;
                break;
        case 15: en_B=en_B&0xFB;
                break;
        case 16: en_B=en_B&0xF7;
                break;
        case 17: en_B=en_B&0xEF;
                break;
        case 18: en_B=en_B&0xDF;
                break;
        case 19: en_B=en_B&0xBF;
                break;
        case 20: en_B=en_B&0x7F;
                break;}}
                temp=temp>>1;}]
void send_msg(unsigned char mo)
{unsigned char
no_1,no_2,i,j,c,addr,wide,PDU,done;
 bit send;
       done=0x1A;
       switch(mo){
      case 1: addr_H=0x02;
            addr_L=0x00;
            break;
      case 2: addr_H=0x02;
                addr_L=0x50;
                break;
case 3: addr_H=0x02;
                addr_L=0xA0;
                break;
case 4: addr_H=0x02;
                addr_L=0xC8;
                break;
case 5: addr_H=0x03;
                addr_L=0x18;
                break;
case 6: addr_H=0x03;
                addr_L=0x90;
                break;
case 7: addr_H=0x03;
                addr_L=0xE0;
                break;
case 8: addr_H=0x04;
                addr_L=0x08;
                break;
case 9: addr_H=0x04;
                addr_L=0x80;
                break;
case 10: addr_H=0x04;
                addr_L=0xD0;
                break;
case 11: addr_H=0x05;
                addr_L=0x20;
                break;
case 12: addr_H=0x05;
                addr_L=0x98;
```

```
            break;
case 13: addr_H=0x05;
            addr_L=0xC0;
            break;
case 14: addr_H=0x06;
            addr_L=0x10;
            break;
case 15: addr_H=0x06;
            addr_L=0x60;
            break;
case 16: addr_H=0x06;
            addr_L=0xD8;
            break;
case 17: addr_H=0x07;
            addr_L=0x00;
            break;
case 18: addr_H=0x07;
            addr_L=0x50;
            break;
case 19: addr_H=0x07;
            addr_L=0xA0;
            break;
case 20: addr_H=0x08;
            addr_L=0x18;
            break; }
s:          I2C_start();
            I2C_out(EEP_W);
            if (outbit) goto s;
            I2C_out(addr_H);
            if (outbit) goto s;
            I2C_out(addr_L);
            if (outbit) goto s;
            I2C_start();
            I2C_out(EEP_R);
            if (outbit) goto s;
            for(i=0;i<39;i++){
            buffer2[i] = I2C_in();
            I2C_ack(1);}
            buffer2[39]=I2C_in();
            I2C_ack(0);
            I2C_stop();
            wide=strlen(buffer2);
            c = strlen(buffer2)/8;
        if (strlen(buffer2)%8>0) c=c+1;
            for(j=0;j<c;j++){
            for(i=0;i<8;i++)
            dec[i] = buffer2[8*j+i];
          encode();
        for (i=0;i<7;i++){
        inttohex(enc[i]);
        str[2*i]=hex[0];
        str[2*i+1]=hex[1];}
        for (i=0;i<14;i++)
        buffer1[j*14+i] = str[i];}
      j=(strlen(buffer2)/8)*14+(strlen(buffer2)
      %8)*2;
        for(i=j;i<200;i++)
        buffer1[i]=0;
        addr=0+(mo-1)*2;
        b:  I2C_start();
            I2C_out(EEP_W);
            if (outbit) goto b;
```

```
                I2C_out(0x09);
                if (outbit) goto b;
                I2C_out(addr);
                if (outbit) goto b;
                I2C_start();
                I2C_out(EEP_R);
                if (outbit) goto b;
                no_1=I2C_in();
                I2C_ack(1);
                no_2=I2C_in();
                I2C_ack(0);
                I2C_stop();
                hex[2]=0;
inttohex(wide);
PDU=(40+strlen(buffer1))/2;
inttostr(PDU);
clrbuff2();
 send = no_1&0x01;
if(send) {
proc=3;
EA=0;
TI=1;
printf("at+cmgs=%s\n",num);
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r1);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                TI=0;
                EA=1;
s1:     while(check2==0);
                check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s1;
                index2=0;
                clrbuff2();}
 send=no_1&0x02;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r2);
```

```
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                    TI=0;
                    EA=1;
s2:       while(check2==0);
                    check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s2;
                    index2=0;
                    clrbuff2();}
 send=no_1&0x04;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                    TI=0;
                    EA=1;
                    while(check2==0);
                    check2=0;
                    while(buffer2[1]!=0x20);
                    index2=0;
                    clrbuff2();
                    EA=0;
                    TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r3);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                    TI=0;
                    EA=1;
s3:       while(check2==0);
                    check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s3;
                    index2=0;
                    clrbuff2();}
 send=no_1&0x08;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                    TI=0;
                    EA=1;
                    while(check2==0);
                    check2=0;
                    while(buffer2[1]!=0x20);
                    index2=0;
                    clrbuff2();
                    EA=0;
                    TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r4);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                    TI=0;
                    EA=1;
s4:       while(check2==0);
                    check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s4;
```

```
                index2=0;
                clrbuff2();}
 send=no_1&0x10;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r5);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                TI=0;
                EA=1;
s5:     while(check2==0);
                check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s5;
                index2=0;
                clrbuff2();}
 send=no_2&0x01;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r6);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                TI=0;
                EA=1;
s6:     while(check2==0);
                check2=0;
                if(buffer2[strlen(buffer2)-
1]!=0x4B) goto s6;
                index2=0;
                clrbuff2();}
 send=no_2&0x02;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
```

```
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r7);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                TI=0;
                EA=1;
s7:     while(check2==0);
                check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s7;
                index2=0;
                clrbuff2();}
 send=no_2&0x04;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
                TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r8);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                TI=0;
                EA=1;
s8:     while(check2==0);
                check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s8;
                index2=0;
                clrbuff2();}
 send=no_2&0x08;
 if(send) {
 proc=3;
 EA=0;
 TI=1;
 printf("at+cmgs=%s\n",num);
                TI=0;
                EA=1;
                while(check2==0);
                check2=0;
                while(buffer2[1]!=0x20);
                index2=0;
                clrbuff2();
                EA=0;
```

```
                    TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r9);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                     TI=0;
                    EA=1;
s9:       while(check2==0);
                    check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s9;
                    index2=0;
                    clrbuff2();}
  send=no_2&0x10;
  if(send) {
  proc=3;
  EA=0;
  TI=1;
  printf("at+cmgs=%s\n",num);
                     TI=0;
                    EA=1;
                    while(check2==0);
                    check2=0;
                    while(buffer2[1]!=0x20);
                    index2=0;
                    clrbuff2();
                    EA=0;
                    TI=1;
printf("%s%s%s%s",at1,csca,at2,numbe
r10);
printf("%s%s%s%c",at3,hex,buffer1,don
e);
                     TI=0;
                    EA=1;
s10:      while(check2==0);
                    check2=0;
if(buffer2[strlen(buffer2)-1]!=0x4B) goto
s10;
                    index2=0;
                    clrbuff2();}}
void inttofloat(unsigned long input)
{  unsigned long temp;
   unsigned char i;
   input=(input*10000)/4096;
   temp=input/10000;
   output[0]=temp;
   temp=input%10000;
   output[1]=temp/1000;
   temp=input%1000;
   output[2]=temp/100;
   temp=input%100;
   output[3]=temp/10;
   output[4]=input%10;
   for(i=0;i<5;i++){
   switch (output[i]){
          case 0: output[i] = 0x30;
          break;
          case 1: output[i] = 0x31;
          break;
                  case 2: output[i] = 0x32;
```

```
        break;
                case 3: output[i] = 0x33;
        break;
                case 4: output[i] = 0x34;
        break;
                 case 5: output[i] = 0x35;
        break;
                case 6: output[i] = 0x36;
        break;
                 case 7: output[i] = 0x37;
        break;
                 case 8: output[i] = 0x38;
        break;
                case 9: output[i] = 0x39;
        break; }}
      output[6]=0;
      output[5]=output[4];
      output[4]=output[3];
      output[3]='.';}
void download(void)
{  unsigned char
temp1,temp2,i,amount,type;
EA=0;
TI=1;
i=0;
printf("K");
start: temp1=getchar();
    if(temp1=='X') goto quit;
    temp2=getchar();
    addr_H=hextoint(temp1,temp2);
    temp1=getchar();
          temp2=getchar();
addr_L=hextoint(temp1,temp2);
                temp1=getchar();
                temp2=getchar();
          amount=hextoint(temp1,temp2);
                temp1=getchar();
                temp2=getchar();
          type=hextoint(temp1,temp2);
                if(type==1){
                for(i=0;i<amount;i++){
                hex[0]=getchar();
                hex[1]=getchar();
buffer1[i]=hextoint(hex[0],hex[1]);}
write:   I2C_start();
                      I2C_out(EEP_W);
                if (outbit) goto write;
                      I2C_out(addr_H);
                if (outbit) goto write;
                      I2C_out(addr_L);
                if (outbit) goto write;
                for(i=0;i<amount;i++){
                I2C_out(buffer1[i]);
                if (outbit) goto write;}
                      I2C_stop();
                      printf("K");
                      goto start;}
                      if(type==2){
                for(i=0;i<amount;i++){
                buffer1[i]=getchar();
          if(buffer1[i]=='.') buffer1[i]=0;  }
```

```
write2:  I2C_start();
                    I2C_out(EEP_W);
                if (outbit) goto write2;
                    I2C_out(addr_H);
                if (outbit) goto write2;
                    I2C_out(addr_L);
                if (outbit) goto write2;
                for(i=0;i<amount;i++){
                I2C_out(buffer1[i]);
                if (outbit) goto write2;}
                    I2C_stop();
                    printf("K");
                    goto start;}
quit: while(1);}
```

ประวัติผู้เขียนวิทยานิพนธ์

นายวิสุทธิ์ ศรีเมือง เกิดวันที่ 21 กันยายน พ.ศ. 2523 ที่จังหวัดสุพรรณบุรี สำเร็จการศึกษาปริญญาวิศวกรรมศาตรบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัยในปี พ.ศ. 2545 และได้เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้าที่จุฬาลงกรณ์มหาวิทยาลัยเมื่อปี พ.ศ. 2545