การควบคุมอุปกรณ์ส่องสว่างผ่านเทคโนโลยีแบบไร้สาย
LIGHT BULB CONTROL VIA WIRELESS TECHNOLOGY

กฤติกา ชินรัตน์
ณัฐพงษ์ สุขประเสริฐ
สถิตย์พงศ์ ชนะโรจน์เจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอัตโนมัติ
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2555

การควบคุมอุปกรณ์ส่องสว่างผ่านเทคโนโลยีแบบไร้สาย
LIGHT BULB CONTROL VIA WIRELESS TECHNOLOGY

กฤติกา ชินรัตน์
ณัฐพงษ์ สุขประเสริฐ
สถิตย์พงศ์ ชนะโรจน์เจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัญฑิต
สาขาวิศวกรรมอัตโนมัติ
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2555

LIGHT BULB CONTROL VIA WIRELESS TECHNOLOGY

KRITTIKA CHINRUT

NATTAPONG SUKPRASERT

SATIDPONG CHANAROTCHAROEN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BECHELOR OF ENGINEERING IN AUTOMATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2012

ปริญญานิพนธ์ปีการศึกษา 2555
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองปริญญานิพนธ์

.....

หัวข้อปริญญานิพนธ์ การควบคุมอุปกรณ์ส่องสว่างผ่านเทคโนโลยีแบบไร้สาย
LIGHT BULB CONTROL VIA WIRELESS TECHNOLOGY

นักศึกษาผู้จัดทำ นางสาวกฤติกา ชินรัตน์ **รหัสนักศึกษา** 52010024
นายณัฐพงษ์ สุขประเสริฐ **รหัสนักศึกษา** 52010336
นายสถิตพงศ์ ชนะโรจน์เจริญ **รหัสนักศึกษา** 52011236

ปริญญา วิศวกรรมศาตรบัณฑิต
สาขาวิชา วิศวกรรมอัตโนมัติ
ปีการศึกษา 2555

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
รศ.ประภาษ อุคคกิมาพันธุ์	
อาจารย์กฤษณ์ เสมอพิทักษ์	

หัวข้อปริญญานิพนธ์	การควบคุมอุปกรณ์ส่องสว่างผ่านเทคโนโลยีแบบไร้สาย LIGHT BULB CONTROL VIA WIRELESS TECHNOLOGY
นักศึกษาผู้จัดทำ	นางสาวกฤติกา ชินรัตน์ **รหัสนักศึกษา** 52010024
	นายณัฐพงษ์ สุขประเสริฐ **รหัสนักศึกษา** 52010336
	นายสถิตย์พงศ์ ชนะโรจน์เจริญ **รหัสนักศึกษา** 52011236
อาจารย์ที่ปรึกษา	รศ.ประภาษ อุคคกิมาพันธุ์
	อาจารย์กฤษณ์ เสมอพิทักษ์
ปีการศึกษา	2555

บทคัดย่อ

ในโครงงานนี้เป็นการออกแบบและสร้างอุปกรณ์ซึ่งอำนวยความสะดวกในการจัดการระบบแสงสว่างสำหรับที่พักอาศัยนอกจากนั้นยังสามารถประยุกต์ใช้ในโรงงานอุตสาหกรรม รวมไปถึงสถานที่ต้องการจัดการระบบแสงสว่างผ่านระบบสารสนเทศ โดยใช้เทคโนโลยีแบบไร้สายทำงานร่วมกับไมโครคอนโทรลเลอร์ซึ่งอยู่ภายในชิปเดียวกัน ทำให้สามารถสร้างเครือข่ายของระบบประมวลผล ซึ่งอุปกรณ์ได้ออกแบบมาให้ติดตั้งเข้ากับระบบแสงสว่างแบบเดิม และลดค่าใช้จ่ายในส่วนฮาร์ดแวร์ที่มีความเป็นไปได้เมื่อนำมาประยุกต์ใช้จริง และยังอาจเป็นแนวทางสำหรับผู้ที่สนใจ

Thesis Title	LIGHT BULB CONTROL VIA WIRELESS TECHNOLOGY	
Authors	Ms. Krittika	Chinrut
	Mr. Nattapong	Sukprasert
	Mr. Satidpong	Chanarotcharoen
Thesis Advisor	Assoc.Prof. Prapart	Ukakipaparn
	Mr. Krit	Smerpitak
Year	2012	

ABSTRACT

This project concerns about the education of equipment decoration. It is able to be convenience tools and lighting management system for the users. Although the equipment are able to apply with the indoor lighting system such as home, warehouse, industrial factory along with the place where would like to use tool effectively. According to the users needs and achieve the user goals too. Using wireless technology to work. This series can communicate via wireless technology. Besides the equipment designed for work together with normally lighting system. Final this project could be approach for interest to interpolating people.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี คณะผู้จัดทำขอกราบขอบพระคุณอาจารย์ที่ปรึกษา รศ.ประภาษ อุคคกิมาพันธุ์ และ อาจารย์กฤษณ์ เสมอพิทักษ์ ที่ได้ให้คำปรึกษา ชี้แนะแนวทางในการแก้ปัญหาและให้ความดูแลเอาใจใส่ตลอดการทำงานทั้งหมด

ขอขอบพระคุณ รศ.ประภาษ อุคคกิมาพันธุ์ ที่ให้คำปรึกษา วิธีแก้ไขปัญหา ถ่ายทอดประสบการณ์ในการทำงาน ให้แนวทางความคิดริเริ่มสร้างสรรค์ และทฤษฎีต่างๆที่คณะผู้จัดทำไม่เคยได้เรียนรู้และไม่เคยปฏิบัติจริงมาก่อน ตลอดจนให้การดูแลเอาใจใส่แก่คณะผู้จัดทำและยังสอบถามความก้าวหน้าของโครงงานอย่างสม่ำเสมอ

ขอขอบพระคุณ อาจารย์กฤษณ์ เสมอพิทักษ์ ที่ได้เอื้อเฟื้ออุปกรณ์และสถานที่ตลอดการทำโครงงานนี้ที่เอื้อเฟื้ออุปกรณ์ในการทดลอง ให้คำแนะนำและช่วยแก้ไขปัญหาต่าง ๆ ที่เกิดขึ้นระหว่างการทำโครงงานรวมถึงให้คำแนะนำในการทำปริญญานิพนธ์ฉบับนี้

ขอขอบพระคุณคณาอาจารย์สาขาวิชาวิศวกรรมอัตโนมัติและวิศวกรรมการวัดและควบคุมทุกท่าน ที่ได้ให้คำแนะนำและการช่วยเหลืออันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

ขอขอบคุณเพื่อน ๆ ทุกคนที่ให้กำลังใจ สอบถามความคืบหน้าของโครงงานรวมถึงปริญญานิพนธ์ฉบับนี้ และให้คำปรึกษาต่าง ๆ ซึ่งเปรียบเสมือนแรงกระตุ้นเป็นอย่างดี จนสามารถทำให้โครงงานและปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี

ขอขอบคุณบุคคลที่คอยให้คำปรึกษาและคำแนะนำถึงสิ่งที่ไม่ถนัดและยังไม่เข้าใจ ที่เกี่ยวข้องกับโครงงาน และช่วยแก้ไขปัญหาต่าง ๆ ที่เกิดขึ้นระหว่างการทำโครงงานด้วย

และสุดท้าย ขอกราบขอบพระคุณ คุณพ่อและคุณแม่อันเป็นที่รักยิ่ง ซึ่งคอยเป็นแรงสนับสนุนเป็นแรงบันดาลใจรวมไปถึงแรงกระตุ้นในความสำเร็จและความก้าวหน้าของบุตร ทั้งคุณค่าและประโยชน์ของโครงงานในครั้งนี้ คณะผู้จัดทำขอมอบให้แด่ผู้มีพระคุณทุกท่านเป็นอย่างยิ่งที่ทำให้โครงงานนี้สำเร็จ

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

ถ้าพูดถึงอุปกรณ์ Electronics คงไม่มีใครที่จะปฏิเสธว่าไม่รู้จักอุปกรณ์ Electronics เพราะในทุกวันนี้เทคโนโลยีได้เจริญก้าวไกลและเป็นที่แพร่หลายมาก ซึ่งในชีวิตประจำวันนั้นเต็มไปด้วยเครื่องใช้ Electronics มากมาย ที่เป็นตัวช่วยในการใช้ชีวิตของผู้คนนั้นให้มีความคล่องตัวหรือลดความยุ่งยากให้ และเพราะสิ่งเหล่านี้ที่อุปกรณ์ Electronics มอบให้ จึงทำให้ผู้ใช้ส่วนใหญ่เกิดความเคยชินจนกลายเป็นความละเลยการใช้อุปกรณ์อย่างพอดี และไม่เล็งเห็นถึงคุณค่าของพลังงาน E-house หรือ Electronics House มีวัตถุประสงค์ในการจัดทำ โดยเป็นการนำอุปกรณ์ Electronics มาประยุกต์ใช้งานเพิ่มเติม เพื่อการจัดการด้านพลังงานแสงสว่าง การลดภาระของผู้ใช้งาน และการใช้พลังงานอย่างรู้คุณค่า

ในทุกวันของการใช้ชีวิต เราทุกคนจำเป็นต้องพึ่งแสงสว่าง ไม่ว่าจะเป็นจากธรรมชาติหรือจากการสร้างขึ้นโดยมนุษย์ จึงเป็นเหตุผลที่ไม่สามารถปฏิเสธได้ว่า ระบบแสงสว่างมีความสำคัญต่อการดำรงชีวิตเป็นอย่างมาก และระบบแสงสว่างที่สามารถตอบสนองความต้องการของตัวบุคคลได้นั้น ก็คงเป็นระบบแสงสว่างที่มนุษย์สร้างขึ้นนั่นเอง จึงได้พิจารณาและเห็นว่าถ้าพัฒนาหรือปรับปรุงระบบแสงสว่างให้มีความก้าวหน้ามากกว่าแค่การที่เป็นอยู่ในปัจจุบันนั้น จะเป็นการพัฒนาชีวิตและสิ่งแวดล้อมได้เป็นอย่างดี โดยการเพิ่มคุณสมบัติในการสั่งการ ซึ่งได้นำเทคโนโลยีแบบไร้สายเข้ามาช่วยในการทำงาน ซึ่งทำให้เกิดเป็นระบบเครือข่ายที่มีความสะดวกในการตรวจสอบการทำงาน คุณสมบัติในด้านการแสดงผลการใช้งาน คือมีการตรวจวัดค่าทางไฟฟ้าต่างๆ เช่น ค่าแรงดันไฟฟ้า ค่ากระแสไฟฟ้า เพื่อนำมาเป็นข้อมูลในการตรวจสอบปริมาณการใช้ไฟฟ้าหรือการใช้งานของหลอดไฟ ในด้านคุณสมบัติของระบบสำรองเมื่อเกิดการขัดข้อง จะมีการใช้ในส่วนของแม็คคานิคเข้ามาช่วยเหลือผู้ใช้งานในกรณีที่โมดูลเกิดการขัดข้องหรือเสียหายทำให้ระบบแสงสว่างยังคงทำงานได้ตามปกติ หรือคุณสมบัติของด้านระบบความปลอดภัยต่อผู้ใช้งาน ก็จะช่วยเพิ่มความปลอดภัยของชีวิตของตัวผู้ใช้งานหากเกิดการรั่วไหลของกระแสไฟฟ้า โดยในที่สุดแล้วผลที่คาดว่าจะได้รับหลังจากการติดตั้งโมดูลที่มีคุณสมบัติดังกล่าวนั้น จะช่วยให้การใช้ชีวิตในแต่ละวันที่ดีขึ้นในด้านใดด้านหนึ่ง ทั้งการจัดการพลังงานและค่าใช้จ่ายภายในบ้าน ซึ่งจะส่งผลต่อสภาพสิ่งแวดล้อมในทางที่ดีขึ้นได้

1.2 วัตถุประสงค์ของปริญญานิพนธ์

เพื่อออกแบบตัวควบคุมการเปิด-ปิด อุปกรณ์ส่องสว่างภายในอาคารสำหรับ E-house ซึ่งตามแนวคิด จะประกอบด้วย 2 ส่วนหลักคือประกอบไปด้วยวงจรที่เรียกว่า วงจรโหลด และวงจรควบคุม ซึ่งวงโหลดนั้นหมายถึง วงจรควบคุมการจ่ายกำลังงานไฟฟ้าเปรียบเสมือนสวิตช์ที่ต่ออนุกรมกับโหลดที่ต่ออนุกรมกับโหลดที่เชื่อมต่อกับแหล่งจ่ายกำลังไฟฟ้า ส่วนวงจรควบคุมนั้นเปรียบเสมือนตัวกำหนดสภาวะของสวิตช์ของวงจรโหลดว่าอยู่ในสภาวะปิดวงจรหรือเปิดวงจร และการนำเทคโนโลยีสื่อสารแบบไร้สาย (wireless) มาใช้งานร่วมกับอุปกรณ์ส่องสว่าง และใช้ไมโครคอนโทรลเลอร์ในการควบคุมการทำงาน

1.3 ขอบเขตของโครงงาน

1. สร้างโมดูลสำหรับควบคุมอุปกรณ์ส่องสว่างแบบไร้สาย
2. สร้างโมดูลสำหรับการสั่งการ (Remote)
3 .แสดงผลผ่านเว็บบราวเซอร์

1.4 ขั้นตอนการดำเนินงาน

1. ศึกษาการทำงานของเทคโนโลยีแบบไร้สาย
2. ศึกษารูปแบบการทำงานของไมโครคอนโทรลเลอร์
3. ศึกษาการออกแบบอุปกรณ์อิเล็กทรอนิกส์
4. รวบรวมข้อมูลทั้งหมด พร้อมศึกษาแนวทางความเป็นไปได้
5. ออกแบบและสร้างโมดูล
6. เขียนโปรแกรม
7. ทดสอบการทำงานร่วมกับโปรแกรม
8. สรุป วิเคราะห์ผลและเสนอแนะ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ประยุกต์ใช้ไมโครคอนโทรลเลอร์ให้สั่งงานอุปกรณ์ส่องสว่างและสามารถลดภาระของผู้ใช้งานในส่วนของการควบคุมระบบไฟฟ้าและเป็นสิ่งอำนวยความสะดวกในชีวิตประจำวัน

1.6 รายละเอียดของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ แบ่งออกเป็น 5 บท ซึ่งครอบคลุมเนื้อหาดังนี้

บทที่ 1 บทนำ กล่าวนำถึงที่มาและความสำคัญ วัตถุประสงค์ ขั้นตอนการศึกษา ประโยชน์ที่ได้รับ และรายละเอียดโดยย่อของปริญญานิพนธ์แต่ละบท

บทที่ 2 หลักการที่เกี่ยวข้องและอุปกรณ์ที่เลือกใช้ กล่าวถึงไมโครคอนโทรลเลอร์ nRF24le1 แลชชิ่งรีเลย์ ทฤษฎีกำลังไฟฟ้า หลักการวัดพลังงาน หลักการที่ใช้ในแหล่งจ่ายพลังงาน การสื่อสาร RS-232 ตลอดจนหลักการที่เกี่ยวข้องในออกแบบเว็บไซต์ ที่นำมาใช้ในการทำโครงงาน

บทที่ 3 การออกแบบการควบคุมอุปกรณ์ส่องสว่าง กล่าวถึงการออกแบบโครงสร้างของการควบคุมอุปกรณ์ส่องสว่าง การออกแบบวงจร การออกแบบการควบคุม และการออกแบบส่วนเชื่อมต่อเว็บบราวเซอร์

บทที่ 4 การทดสอบการทำงาน กล่าวถึงการทดสอบการทำงานของการควบคุมอุปกรณ์ส่องสว่างแบบไร้สายในส่วนต่าง ๆ เช่น การทดลองการควบคุมผ่านการสั่งงานจากคอมพิวเตอร์ การทดลองการควบคุมผ่านการสั่งงานจากบอร์ดสวิตซ์ไมโครคอนโทรลเลอร์ เป็นต้น

บทที่ 5 บทสรุปและข้อเสนอแนะ กล่าวถึง การสรุปผลของการทำงานปัญหาและอุปสรรค และข้อเสนอแนะของปริญญานิพนธ์

บทที่ 2
หลักการที่เกี่ยวข้องและอุปกรณ์ที่เลือกใช้

2.1 กล่าวนำ

ทฤษฎีและหลักการที่นำมาใช้เพื่อไปประยุกต์ใช้ในการออกแบบตัวควบคุมการเปิด-ปิด อุปกรณ์ส่องสว่างภายในอาคารแบบไร้สาย ซึ่งเป็นแนวทางในการออกแบบอุปกรณ์ทั้งที่ใช้ในวงจรโหลดและวงจรควบคุม โดยการประยุกต์ใช้ไมโครคอนโทรลเลอร์ nRF24le1 ที่รวมเทคโนโลยีรับส่งสัญญาณคลื่นวิทยุที่รวมอยู่กับไมโครคอนโทรลเลอร์ ในการควบคุมการเปิด-ปิดอุปกรณ์ส่องสว่างผ่านรีเลย์ที่เป็นแบบแลชชิ่งเพื่อประหยัดพลังงาน นอกจากนั้นยังสามารถประมาณการใช้พลังงานส่องสว่างผ่านพาวเวอร์ไอซี โดยข้อมูลการใช้พลังงานสามารถเชื่อมต่อผ่านทาง RS-232 เพื่อใช้ในการจัดเก็บข้อมูล รวมถึงการวิเคราะห์การใช้พลังงานด้วยโปรแกรมฐานข้อมูลแสดงผลผ่านทางเว็บไซต์

2.2 ไมโครคอนโทรลเลอร์ nRF24le1

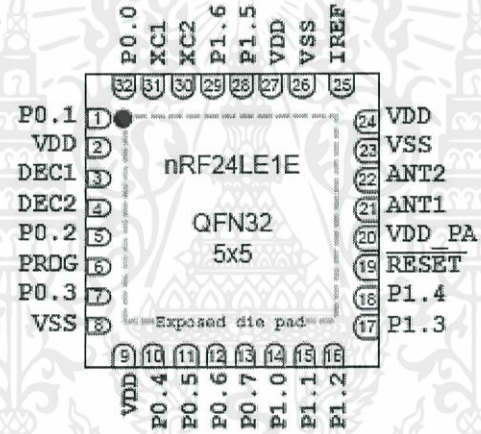
ไมโครคอนโทรลเลอร์ nRF24le1 เป็นผลิตภัณฑ์ของทางบริษัท Nordic semiconductor โดยตัวของผลิตภัณฑ์นั้นจะมีราคาค่อนข้างต่ำแต่มีประสิทธิภาพสูงในตระกูลที่รองรับเทคโนโลยีรับส่งสัญญาณคลื่นวิทยุช่วงความถี่ 2.4GHz (2.4GHz RF transceiver) ร่วมกับไมโครคอนโทรลเลอร์แบบฝังอยู่ภายในตัว บอร์ด nRF24le1 นั้นเหมาะสำหรับผู้ที่ต้องการใช้งานในลักษณะ Ultra Low Power แบบไร้สายโดยใช้ชิปเพียงตัวเดียว รวมถึงการใช้งานในลักษณะอื่น ๆ เช่น การประมวลผลพลังงาน, หน่วยความจำ, Oscillators พลังงานต่ำ, Real time, ตัวจับเวลา และอื่น ๆ สำหรับรูปแบบการใช้งาน nRF24le1 สามารถใช้งานร่วมกับชุดอุปกรณ์ต่าง ๆ ได้แก่ SPI (Serial Peripheral Interface), 2 wire UART, 6-12 bit ADC และการทำงานแบบ PWM (Pulse width modulation) ได้

2.2.1 คุณสมบัติไมโครคอนโทรลเลอร์

- Fast 8-bit microcontroller:
- หน่วยประมวลผล MCS 51
- หน่วยความจำแฟลช สำหรับเขียนโปรแกรม 16 kB และ RAM 1 kB ในตัวชิป
- สามารถรองรับ Input/ Output ได้หลากหลาย
- GPIO (General Purpose Input/ Output)
- SPI master/SPI slave
- 2-Wire master/ slave
- Full duplex serial port
- PWM
- ADC
- Analog comparator
- External interrupts
- Timer inputs

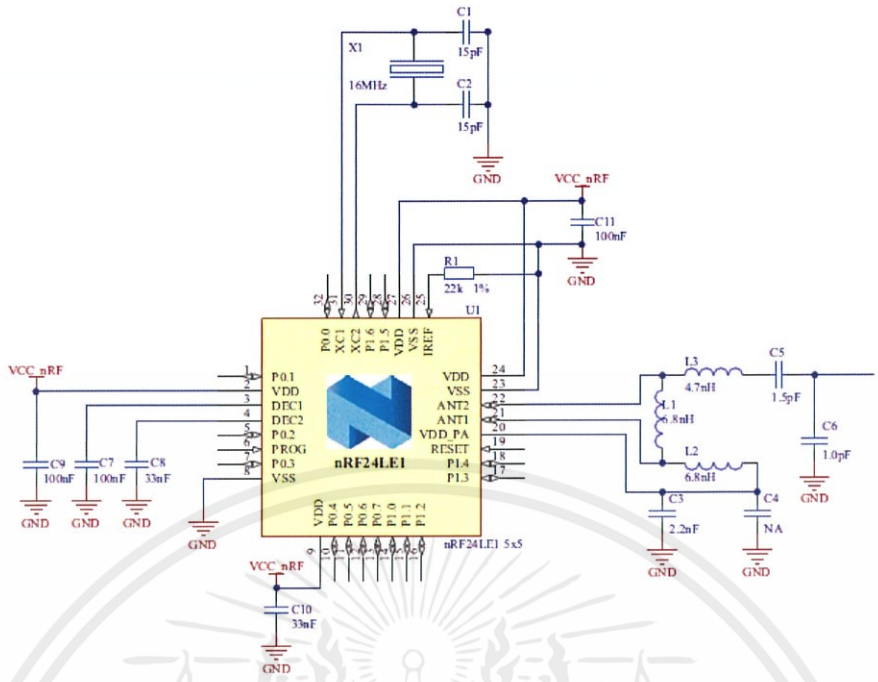
- 32.768 kHz crystal oscillator
- Debug interface
- High performance 2.4 GHz RF-transceiver
- GFSK transceiver
- Auto ACK and retransmit
- Data rate 250 kbps, 1 Mbps or 2 Mbps บนช่องทางอากาศ
- Digital interface (SPI) speed 0-8 Mbps
- ตอบสนองความถี่เปลี่ยนแปลงในระยะเวลาสั้น
- A/D converter: 6, 8, 10 or 12 bit resolution
- ตั้งค่าได้เต็มพิสัยโดย internal reference, external reference or VDD
- ใช้กระแสต่ำเพียงแค่ 0.1mA ที่ 2 ksps

2.2.2 โครงร่างของพอร์ต nRF24le1

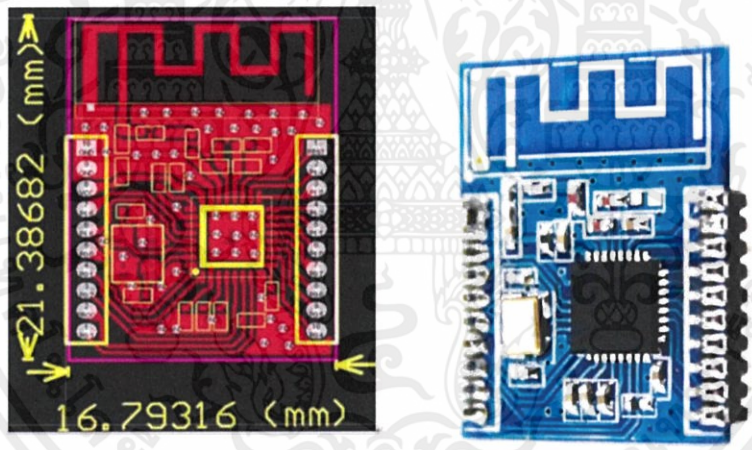


รูปที่ 2.1 โครงร่างของ nRF24le1 ชนิด 32 Pin

ขา VDD หมายถึง Power Supply (+1.9V to +3.6V DC)
ขา VSS หมายถึง กราวด์(0V)
ขา DEC1,DEC2 หมายถึง Power Supply เอาต์พุต
ขา P0.0-P3.6 หมายถึง ขาที่สามารถใช้งานได้
ขา PROG หมายถึง ขาที่ใช้สำหรับโปรแกรม
ขา RESET หมายถึง ขารีเซตของไมโครคอนโทรลเลอร์ ซึ่งเป็นแบบ Active Low
ขา IREF หมายถึง กระแสขาออกที่เป็นตัวอ้างอิง
ขา VDD_PA หมายถึง Power Supply ขาออก (+1.8V)
ขา ANT1,ANT2 หมายถึง เชื่อมต่อเสาอากาศอื่นๆ
ขา XC1,XC2 หมายถึง ขาสำหรับเชื่อมต่อคริสตรัล 16M



รูปที่ 2.2 วงจรการเชื่อมต่ออุปกรณ์ของไมโครคอนโทรลเลอร์ nRF24le1



รูปที่ 2.3 บอร์ด nRF24LE1

2.2.3 ระบบเครือข่ายไร้สาย (Wireless LAN)

ระบบเครือข่ายไร้สาย (WLAN = Wireless Local Area Network) คือ ระบบการสื่อสารข้อมูลที่มีรูปแบบในการสื่อสารแบบไม่ใช้สาย โดยใช้การส่งคลื่นความถี่วิทยุในย่านวิทยุ RF และ คลื่นอินฟราเรด ในการรับและส่งข้อมูลระหว่างคอมพิวเตอร์แต่ละเครื่อง ผ่านอากาศ, ทะลุกำแพง, เพดานหรือสิ่งก่อสร้างอื่นๆ โดยปราศจากความต้องการของการเดินสาย นอกจากนั้นระบบเครือข่ายไร้สายก็ยังมีคุณสมบัติครอบคลุมทุกอย่างเหมือนกับระบบ LAN แบบใช้สาย

ที่สำคัญของระบบเครือข่ายไร้สายก็คือ การที่ไม่ต้องใช้สายทำให้การเคลื่อนย้ายการใช้งานทำได้โดยสะดวก ไม่เหมือนระบบ LAN แบบใช้สายที่ต้องใช้เวลาและการลงทุนในการปรับเปลี่ยนตำแหน่งการใช้งานเครื่องคอมพิวเตอร์

ประโยชน์ของระบบเครือข่ายไร้สาย

1. mobility improves productivity & service มีความคล่องตัวสูง ดังนั้นไม่ว่าจะเคลื่อนที่ไปที่ไหน หรือเคลื่อนย้ายคอมพิวเตอร์ไปตำแหน่งใด ก็ยังมีการเชื่อมต่อกับเครือข่ายตลอดเวลาตราบใดที่ยังอยู่ในระยะการส่งข้อมูล
2. installation speed and simplicity สามารถติดตั้งได้ง่ายและรวดเร็ว เพราะไม่ต้องเสียเวลาติดตั้งสายเคเบิล และไม่รกรุงรัง
3. installation flexibility สามารถขยายระบบเครือข่ายได้ง่าย เพราะเพียงแค่มีพีซีการ์ดมาต่อเข้ากับโน้ตบุ๊ก หรือพีซี ก็เข้าสู่เครือข่ายได้ทันที
4. reduced cost- of-ownership ลดค่าใช้จ่ายโดยรวมที่ผู้ลงทุนต้องลงทุน ซึ่งมีราคาสูง เพราะในระยะยาวแล้วระบบเครือข่ายไร้สายไม่จำเป็นต้องเสียค่าบำรุงรักษาและการขยายเครือข่ายก็ลงทุนน้อยกว่าเดิมหลายเท่า เนื่องด้วยความง่ายในการติดตั้ง
5. scalability เครือข่ายไร้สายทำให้องค์กรสามารถปรับขนาดและความเหมาะสมได้ง่ายไม่ยุ่งยาก เพราะสามารถโยกย้ายตำแหน่งการใช้งานโดยเฉพาะระบบที่มีการเชื่อมระหว่างจุดต่อจุด เช่น ระหว่างตึก

2.2.4 มาตาฐาน Shock Burst Protocol

Shock burst Protocol เป็นแพ็คเก็ตข้อมูลในชั้นเชื่อมโยงข้อมูล โดยคุณสมบัติประกอบด้วย แพ็คเก็ตแบบอัตโนมัติ, เวลา, การรับรู้โดยอัตโนมัติ, การส่งสัญญาณใหม่ของแพ็คเก็ต Shock burst Protocol ใช้พลังงานในการใช้งานน้อยแต่มีประสิทธิภาพสูงในการทำงานเชื่อมต่อกับไมโครคอนโทรลเลอร์ และมีคุณสมบัติที่สำคัญช่วยในการปรับปรุงการใช้พลังงานของระบบที่มีสองทิศทางและทิศทางเดียว โดยไม่ต้องเพิ่มความซับซ้อนให้กับด้านตัวควบคุม

คุณสมบัติหลัก

- ขนาดข้อมูลมีขนาดตั้งแต่ 1 – 32 ไบต์ซึ่งสามารถเลือกกำหนดตามความต้องการได้
- สามารถตั้งค่าการจัดการในส่วนของแพ็คเกจข้อมูลได้โดยอัตโนมัติ
- ตอบสนองการรับ-ส่งข้อมูลแบบอัตโนมัติ และจัดส่งข้อมูลใหม่อีกครั้งโดยอัตโนมัติเมื่อมีการสูญหายของข้อมูล
- ติดต่อกันในลักษณะโทโปโลยีแบบสตาร์ (Star Topology) แบบ 1 : 6

หลักการจัดการรับ-ส่งข้อมูล

- การส่งแพ็คเกจข้อมูลจาก PTX เมื่อ PRX ไดรับข้อมูล จะกำหนด PTX ให้อยู่ในโหมดรับเพื่อรอให้แพ็คเกจ ACK สงกลับ
- เมื่อไดรับแพ็คเกจข้อมูลแล้ว PRX จะประกอบแพ็คเกจและส่ง แพ็คเกจที่รับมา (ACK แพ็คเกจ) กลับไปที่ PTX ในโหมดรับ
- ถา PTX ไมไดรับแพ็คเกจ ACK ในทันทีหรือภายในระยะเวลาที่กำหนด จะ มีการจัดการส่งข้อมูลชุดเดิมส่งออกไปใหม่ เพื่อเป็นการตรวจสอบการสูญ หายของข้อมูล
- รูปแบบแพ็คเกจของ Shock Burst Protocol (ตามรูปที่ 2.4) ประกอบไปด้วย

รูปที่ 2.4 ตัวอย่างรูปแบบแพ็คเกจของ Shock Burst Protocol

จากรูปที่ 2.15 เป็นตัวอย่างรูปแบบแพ็คเกจของ Shock Burst Protocol จะประกอบด้วยส่วนต่าง ๆ ดังนี้

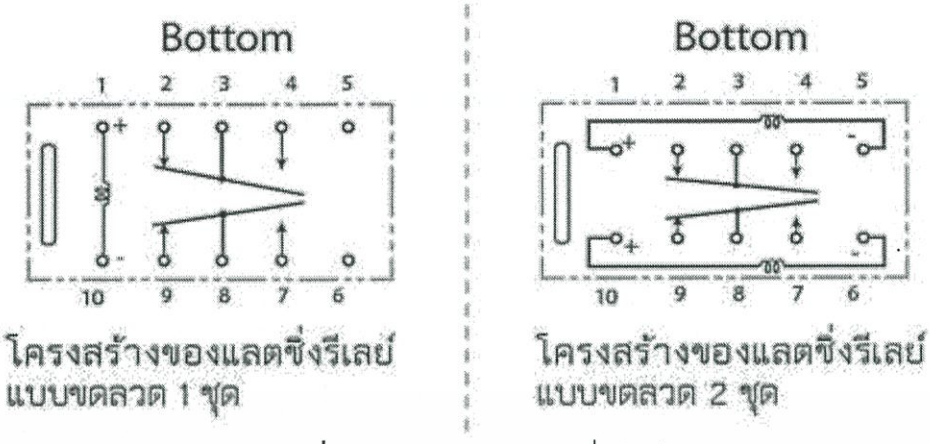
- Preamble มีขนาด 1 ไบต์ จะมีการเปลี่ยนแปลงอยู่ตลอดเวลา โดย ขึ้นอยู่กับบิตแรก ถ้าเป็น 1 จะมีการตั้งค่า เริ่มต้นอัตโนมัติเป็น 10101010 แต่ถ้าบิตแรกเป็น 0 จะมีการตั้งค่าเริ่มต้นเป็น 01010101
- Address เปรียบเสมือนขอบเขตที่อยู่ของสัญญาณ เพื่อให้สามารถ รับทราบได้ว่าจะมีการตรวจพบข้อมูลที่ถูกส่งออกมาจะมีการกำหนด Address ให้สามารถตอบสนองกันได้ โดยสามารถกำหนดได้เป็น 3 ,4 และ 5 ไบต์ ซึ่งถ้าตัวอุปกรณ์มี Address ที่ไม่เหมือนกันก็ไม่สามารถที่ จะรับ – ส่งข้อมูลกันได
- Packet Control Field ดังรูปที่ 2.16 จะแบ่งออกเป็น 3 สวน ดังนี้
  1. Payload length มีขนาด 6 บิต ทำหน้าที่กำหนดขนาดของข้อมูลได้ 0-32 ไบต์
  2. PID (Packet Identification) มีขนาด 2 บิต ทำหน้าที่ เป็นรีจิสเตอร์ที่กำกับในโหมด PTX และ PRX ในการรับ –ส่งข้อมูล และตรวจสอบข้อมูลสูญหาย
  3. No Acknowledgment flag (NO_ACK) มีขนาด 1 บิต ทำหน้าที่เป็นรีจิสเตอร์ที่จะบอกถึงการตอบสนองของข้อมูล

รูปที่ 2.5 รูปแบบ Packet Control Field

2.3 แลตชิ่งรีเลย์

แลตชิ่งรีเลย์ (Latching relay) ประกอบด้วยหน้าสัมผัส 2 ชุด และมีขดลวดเหนี่ยวนำหรือโซลินอยด์ (solenoid) 2 แบบคือ แบบขดลวดเหนี่ยวนำ 1 ชุด และขดลวดเหนี่ยวนำ 2 ชุด สำหรับควบคุมหน้าสัมผัสให้ต่อหรือจากกัน

แลตชิ่งรีเลย์แบบขดลวดเหนี่ยวนำ 2 ชุด ควบคุมโดยการจ่ายกระแสไฟฟ้ากระตุ้นที่ขั้วของขดลวดเหมือนกับแบบขดลวด 1 ชุด แต่จะต่างกันตรงที่หากต้องการกลับสถานะของหน้าสัมผัสจะไม่ใช้การสลับขั้วไฟ แต่จะต้องจ่ายกระแสไฟฟ้ากระตุ้นให้กับขดลวดอีกหนึ่งชุดดังรูปที่ 2.6



รูปที่ 2.6 โครงร่างของแลตชิ่งรีเลย์

2.4 ทฤษฎีกำลังไฟฟ้า

โดยทั่วไปอุปกรณ์ไฟฟ้า ในอาคารหรือโรงงานนั้นต้องอาศัยทั้งกำลังไฟฟ้าจริง (real power) และกำลังไฟฟ้ารีแอคทีฟ (reactive power) เพื่อใช้ในการทำงาน ค่าสัดส่วนของกำลังไฟฟ้าทั้งสองชนิดดังกล่าวบ่งบอกถึงค่าตัวประกอบกำลังไฟฟ้า (Power Factor :PF) ของอุปกรณ์ไฟฟ้าแต่ละชนิดหรือของอาคารหรือโรงงานโดยรวม

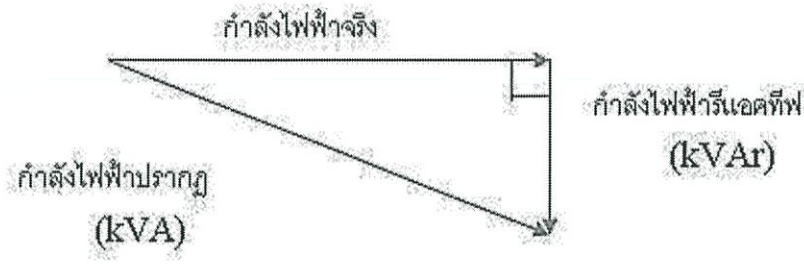
การจ่ายกำลังไฟฟ้าตามปกติประกอบด้วย 2 ส่วนหลัก คือ

1. กำลังไฟฟ้าจริง (real power) มีหน่วยเป็น (กิโลวัตต์) หรืออาจเรียกเป็นกำลังไฟฟ้าหรือพลังไฟฟ้าใช้งานก็ได้ เป็นกำลังไฟฟ้าที่ใช้ประโยชน์ในการทำงานของเครื่องจักร เช่น งานที่ได้จากมอเตอร์หรือจากแสงสว่าง เป็นต้น
2. กำลังไฟฟ้ารีแอคทีฟ (reactive power) มีหน่วยเป็น กิโลวา (kVAr) เป็นกำลังไฟฟ้าที่จ่ายเพื่อสร้างสนามแม่เหล็กให้แก่อุปกรณ์และเครื่องใช้ทางไฟฟ้า เช่น สนามแม่เหล็กในมอเตอร์ บัลลาสต์ของหลอดไฟแสงสว่าง เป็นต้น

ทั้งนี้ค่ากำลังไฟฟ้าทั้งสองส่วนดังกล่าวจะมีทิศทางตั้งฉากซึ่งกันและกัน สามารถนำมาเขียนแสดงความสัมพันธ์ได้เป็นรูปสามเหลี่ยมมุมฉาก หรือ phasor diagram ดังแสดงในรูปที่ 2.7 ส่วนความสัมพันธ์ระหว่างกำลังไฟฟ้า กระแสไฟฟ้าและแรงดันไฟฟ้าสามารถแสดงได้ ดังรูปที่ 2.8

∅ คือ มุมระหว่างแรงดันและกระแสไฟฟ้า ส่วน I คือ ขนาดกระแสรวม

รูปที่ 2.7 ความสัมพันธ์ระหว่าง กำลังไฟฟ้า กระแสไฟฟ้า และแรงดันไฟฟ้า



รูปที่ 2.8 ความสัมพันธ์ระหว่างกำลังไฟฟ้าจริง และกำลังไฟฟ้ารีแอคทีฟ

2.5 หลักการวัดพลังงาน

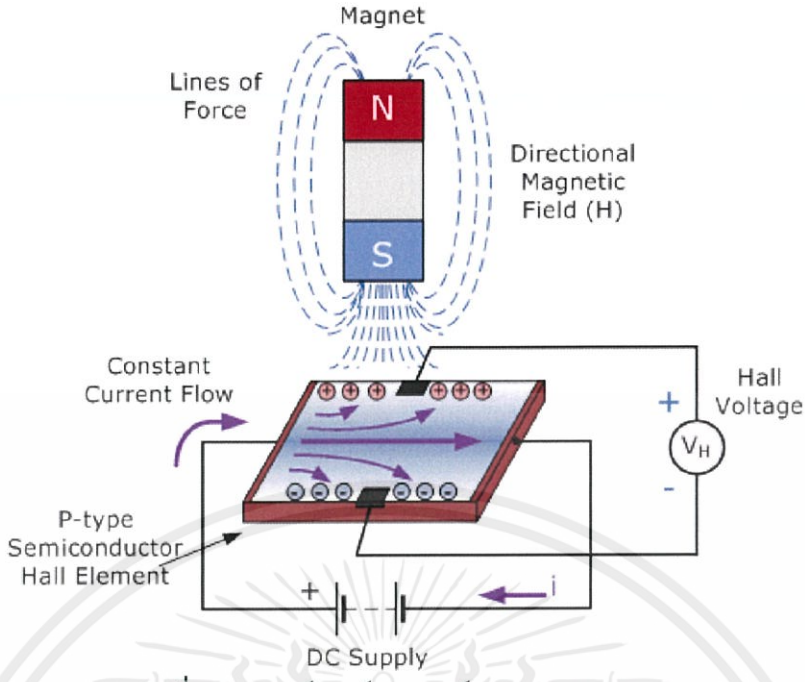
2.5.1 Hall effect current sensor

ฮอลล์เอฟเฟคเคอร์เรนท์เซนเซอร์ (Hall effect current sensor) สามารถวัดค่ากระแสโดยการวัดคล้องสายของกระแสที่ไหล และให้เอาท์พุตออกมาเป็นแรงดัน ฮอลล์เอฟเฟคเคอร์เรนท์เซนเซอร์ มี 2 รุ่นคือรุ่นที่ให้แรงดันเอาต์พุตเป็นแบบดิจิตอลคือ 0 กับ 1 รุ่นที่ให้แรงดันเอาต์พุตเป็นแบบอนาล็อก แต่ ฮอลล์เอฟเฟคเคอร์เรนท์เซนเซอร์ ส่วนใหญ่จะใช้เป็นเครื่องมือวัดสำหรับความแม่นยำสูง สามารถวัดสัญญาณที่มี DC และ Harmonics ปะปนมากได้ หรือกระแสมีความซับซ้อนของสัญญาณปะปนสูง เหมาะสำหรับการต้องวัดเพื่อวิเคราะห์หาHarmonics ต่าง ๆ

ปรากฏการณ์ฮอลล์ (Hall Effect) หรือฮอลล์เอฟเฟค

ปรากฏการณ์ฮอลล์ (Hall Effect) หรือฮอลล์เอฟเฟค เป็นปรากฏการณ์ทางไฟฟ้าที่ค้นพบโดย เอ็ดวิน ฮอลล์ (Edwin Hall) ในปีพ.ศ. 2422 สิ่งที่เค้าค้นพบมีหลักการโดยสรุปดังนี้

แผ่นตัวนำที่มีกระแสไหลผ่านเมื่อมีฟลักซ์แม่เหล็ก (Magnetic Flux) มากระทำในทิศทางตั้งฉากกับแผ่นตัวนำ จะทำให้เกิดสนามไฟฟ้าหรือแรงดันเรียกว่าแรงดันฮอลล์ (Hall Voltage) ขึ้นที่ตัวนำในทิศทางตั้งฉากกับกระแสและฟลักซ์แม่เหล็ก เมื่อจ่ายกระแสคงที่ให้แผ่นตัวนำจะทำให้กระแสไหลผ่านแผ่นตัวนำอย่างคงที่ โดยอิเล็กตรอนจะเคลื่อนที่จากขั้วลบไปขั้วบวก

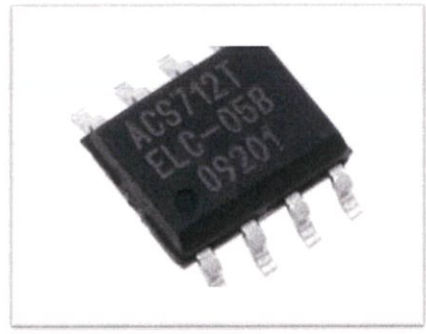


รูปที่ 2.9 เคอร์เรนท์เซนเซอร์แสดง Hall Effect

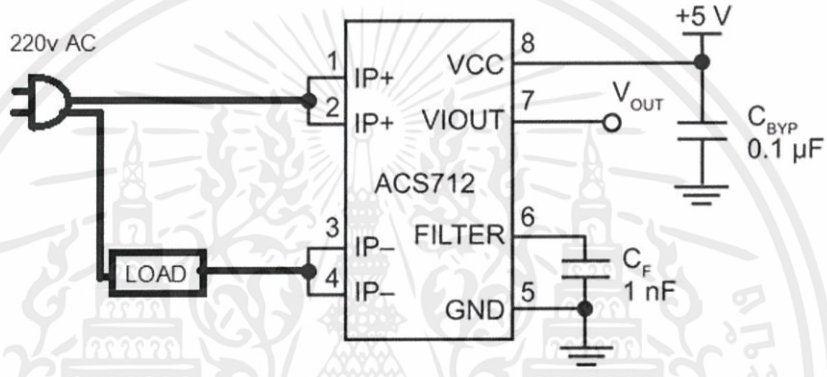
เมื่อมีฟลักซ์แม่เหล็กมากระทำกับแผ่นตัวนำในทิศทางตั้งฉากจะทำให้ประจุพาหะ (Charge Carrier) ของตัวนำเบี่ยงเบนไปด้านบนของตัวนำ จากรูปที่ 2.9 ประจุพาหะเป็นอิเล็กตรอนมีประจุเป็นประจุลบทำให้ด้านบนของแผ่นตัวนำมีขั้วไฟฟ้าเป็นลบ ส่วนด้านล่างของแผ่นตัวนำจะมีขั้วตรงข้ามกับด้านบนนั่นคือมีประจุบวก เมื่อวัดความต่างศักย์ระหว่างด้านบนกับด้านล่างทำให้ได้แรงดันไฟฟ้าออกมาเป็นแรงดันลบ โดยขนาดของแรงดันที่วัดได้จะขึ้นอยู่กับความหนาแน่นของฟลักซ์แม่เหล็กที่มากระทำ หากความเข้มสนามแม่เหล็กมากก็จะทำให้เกิดแรงดันมาก และถ้าความเข้มสนามแม่เหล็กน้อย แรงดันก็จะน้อยตามไปด้วย

ส่วนกรณีที่มีการกลับขั้วแม่เหล็กจะทำให้แรงดันเอาท์พุตกลับขั้วกับกรณีที่กล่าวมา ตัวนำที่มีประจุพาหะเป็นอิเล็กตรอนได้แก่ ตัวนำไฟฟ้าทั่วไป สารกึ่งตัวนำชนิดเอ็น (N-Type) ส่วนตัวนำที่มีประจุพาหะเป็นประจุบวกได้แก่ สารกึ่งตัวนำชนิดพี (P-Type) ปัจจุบันฮอลล์เอฟเฟคจะอยู่ในรูปของวงจรรวมหรือ IC (Integrated Circuit) ที่ทำมาจากสารกึ่งตัวนำ เนื่องจากสารกึ่งตัวนำจะให้แรงดันเอาท์พุตสูงกว่าตัวนำไฟฟ้าทั่วไป

โดยในปริญญานิพนธ์ฉบับนี้เลือกใช้ ไอซี ACS712 ดังรูปที่ 2.10 โดยมีตัวอย่างต่อการใช้งานดัง รูปที่ 2.11 โดยที่ค่าแรงดันเอาต์พุต Vout อยู่ภายในช่วง 0.5 V - 4.5 V สามารถต่อใช้งานไอซีโดยอ้างอิงการเชื่อมต่อพอร์ตใช้งานจากรูปที่ 2.9



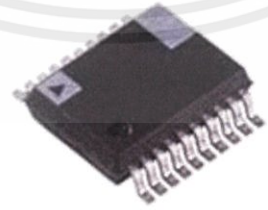
รูปที่ 2.10 ACS712



รูปที่ 2.11 ตัวอย่างการต่อใช้งานไอซี ACS712

2.5.2 Power IC

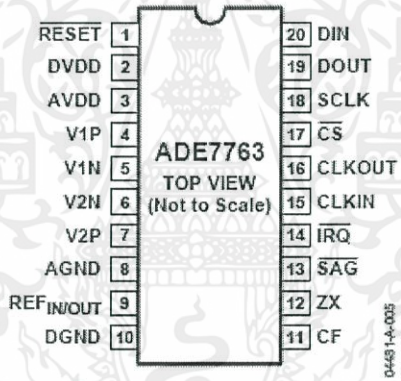
ในปริญญานิพนธ์ฉบับนี้เลือกใช้ ไอซี ADE7763 รูปที่ 2.12 โดยไอซี ADE7763 เป็นไอซีวัดพลังงาน หรือ Energy Metering IC ซึ่ง IC ตัวนี้มีความสามารถในการวัดพลัง ที่มีความถูกต้องและความแม่นยำสูง สามารถวัดแรงดันทางไฟฟ้า กระแสทางไฟฟ้า และค่ากำลังทางไฟฟ้าต่างๆ สามารถวัดค่าทางไฟฟ้าออกมาได้หลายค่า เช่น ค่ากำลังไฟฟ้าจริง ค่ากำลังไฟฟ้าที่ปรากฏ



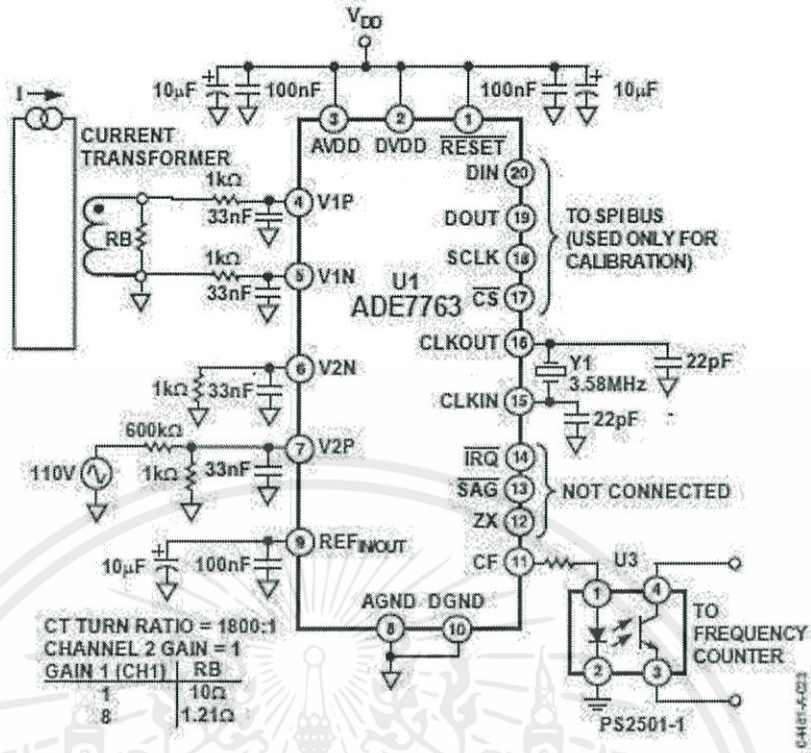
รูปที่ 2.12 ไอซี ADE7763

ADE7763 มีส่วนที่ใช้ติดต่อกับอุปกรณ์ภายนอก แบบอนุกรม (SPI: Serial Peripheral Interface) เพื่อใช้ในการติดต่อระหว่าง Power IC กับไมโครคอนโทรลเลอร์ได้อีกด้วย

ADE7763 มีอินพุตออกสองทาง คือ ทางที่หนึ่งเป็นอินพุตสำหรับวัดค่ากระแสไฟฟ้า (Current Channel) และอีกทาง คือ อินพุตของแรงดันไฟฟ้า (Voltage Channel) ซึ่งทั้งสอง Channel นี้ รับค่าแรงดันไฟฟ้าได้มากที่สุดไม่เกิน ±0.5 Vrms หลังจากนั้นสัญญาณจะผ่านเข้ามาที่ตัวขยายสัญญาณ (Amplifier) ต่อมา สัญญาณที่วัดได้ ก็จะผ่านตัวแปลงสัญญาณจากอนาล็อกเป็นดิจิตอล (A to D) จากนั้นผ่านเข้าสู่ตัวกรองความถี่สูง (High Pass Filter) เพื่อตัดสัญญาณรบกวนต่าง ๆ ออก และเมื่อสัญญาณผ่านวงจรอินทิเกรเตอร์ (Integrator) ทั้งสอง Channel สัญญาณจะแบ่งออกเป็นสองส่วน โดยส่วนที่หนึ่ง จะนำไปคูณกับแรงดันไฟฟ้า ผลที่ได้จะเป็นกาลังไฟฟ้าจริง (Active Power) มีหน่วยเป็นวัตต์ (Watt) และนำค่าที่ได้เก็บไว้ในรีจิสเตอร์ ซึ่งจะสะสมค่าเพิ่มขึ้นเรื่อยๆ ดังนั้นค่าในรีจิสเตอร์ จึงเป็นค่าพลังงานไฟฟ้าจริง (Active Energy) ส่วนสัญญาณที่สอง นั้นจะนามาคำนวณ เป็นค่ารากของกำลังสองเฉลี่ย (rms) ก่อนที่จะนำมาคูณกับแรงดันไฟฟ้า (rms) เพื่อให้ได้กำลังไฟฟ้าปรากฏ (Apparent Power) มีหน่วยเป็น VA (Volt-Amp) และนำค่าที่ได้เก็บไว้ในรีจิสเตอร์ ซึ่งจะสะสมค่าเพิ่มขึ้นเรื่อยๆ ดังนั้นค่าในรีจิสเตอร์ จึงเป็นค่าพลังงานไฟฟ้าปรากฏ (Apparent Energy)



รูปที่ 2.13 โครงร่าง ADE7763



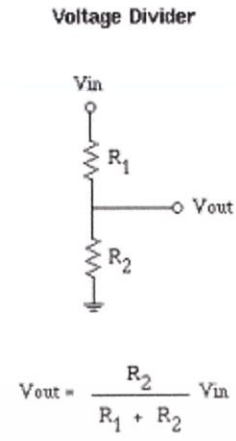
รูปที่ 2.14 ไดอะแกรมของไอซี ADE7763

Voltage Channel เป็นอีก Input ซึ่งสัญญาณแรงดันไฟฟ้าผ่านเข้ามาแล้ว จะต้องผ่านตัวแปลงสัญญาณจากอนาล็อกเป็นดิจิตอล จากนั้นต้องผ่านส่วน การคำนวณ Phase ซึ่งเมื่อสัญญาณผ่านออกมาแล้ว จะแบ่งออกเป็นสามส่วน ส่วนที่หนึ่งจะไปคูณกับสัญญาณกระแส ซึ่งจะได้ผลลัพธ์ออกมาเป็นกำลังไฟฟ้าจริง ดังที่กล่าวไว้ ส่วนที่สอง จะนำไปคำนวณเป็นค่ารากที่สองของกำลังสองเฉลี่ย (rms) เพื่อคูณกับกระแส จากช่องทางที่หนึ่ง ที่เป็นค่ารากที่สองของกำลังสองเฉลี่ย (rms) เช่นกัน เพื่อให้ได้เป็นกำลังไฟฟ้าปรากฏ (Apparent Power) และส่วนที่สาม จะถูกนำค่าไปเก็บไว้ในรีจิสเตอร์ เพื่อส่งข้อมูลออกต่อไป

2.5.3 โวลท์เตจ ดีไวเดอร์

เป็นวงจรสำหรับแบ่งโวลท์เตจ ให้กับวงจรต่าง ๆ ตามต้องการ ซึ่งจำเป็นมากสำหรับแหล่งจ่ายพลังงานของเครื่องอิเล็กทรอนิกส์ ต่าง ๆ แทนที่จะสร้างแหล่งจ่ายพลังงานหลาย ๆ ชุด สำหรับค่าโวลท์เตจต่าง ๆ กัน ก็สามารถสร้างแหล่งจ่ายพลังงานที่มีโวลท์เตจ สูง เพียงชุดเดียวแล้วจึงใช้โวลท์เตจ ดีไวเดอร์ช่วยแบ่งโวลท์เตจ ลงมาตามค่าที่ต้องการ

คุณสมบัติในการหารแรงดันลงมานี้ ทำให้เป็นวิธีที่ง่ายที่สุดที่จำทำให้วัดค่าแรงดันออกมาได้ และแรงดันทางด้านเอาต์พุตจะเป็นไปตามดังรูปที่ 2.15



รูปที่ 2.15 โวลท์เตจดีไวเดอร์

2.6 หลักการที่ใช้ในแหล่งจ่ายพลังงาน

2.6.1 หม้อแปลง

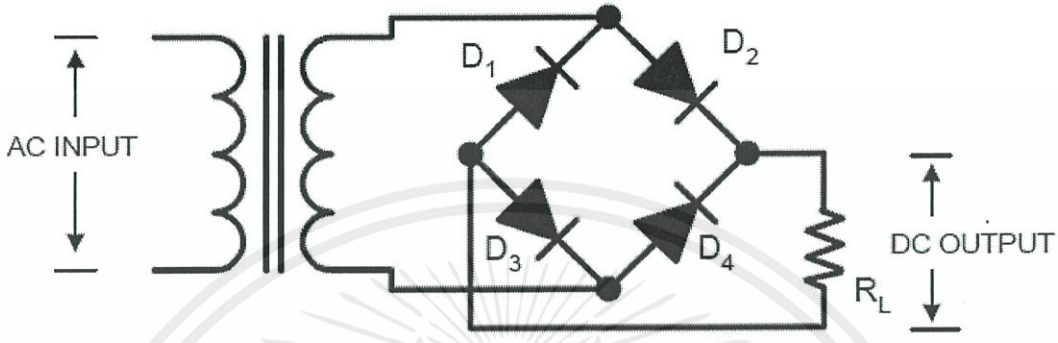
หม้อแปลงไฟฟ้า คืออุปกรณ์ที่ใช้แปลงแรงดันไฟฟ้าสลับ ให้มีขนาดแรงดันตามที่ต้องการ มีการนำหม้อแปลงไฟฟ้าไปใช้ในงานหลายด้าน ทั้งในระบบการจ่ายไฟฟ้า หรือเป็นอุปกรณ์ประกอบในเครื่องใช้ไฟฟ้าที่ใช้กันตามบ้านเรือน ไม่ว่าจะเป็น โทรทัศน์ เครื่องขยายเสียง วิทยุเทป หรือ อะแดปเตอร์แปลงไฟเพื่อใช้ในงาน จึงนับว่ามีความสำคัญและเกี่ยวข้องกับงานทางไฟฟ้าและอิเล็กทรอนิกส์อย่างมาก

รูปที่ 2.16 หม้อแปลงแบบ 2 ขดลวด

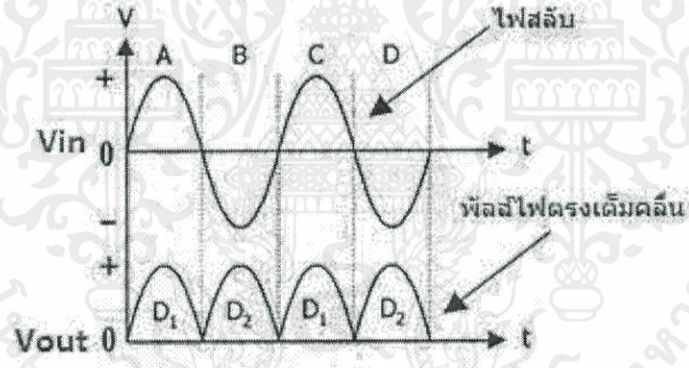
หลักการทำงานหม้อแปลง คือ ในระบบจ่ายไฟฟ้าจะมีการแปลงแรงดันไฟฟ้าสลับให้มีขนาดสูงมากๆ เช่นให้มีขนาดเป็น 48 kV หรือ 24 kV เพื่อลดขนาดของลวดตัวนำ ที่ต้องใช้ในการจ่ายไฟฟ้าเป็นระยะทางไกลๆ เมื่อถึงปลายทางก่อนที่จะจ่ายไฟฟ้าไปให้แก่บ้านเรือนต่างๆ ก็จะแปลงระดับแรงดันไฟฟ้าให้ลดลงเป็น 220 Vac เพื่อลดอันตรายที่จะเกิดแก่ผู้ใช้ไฟฟ้า และเมื่อต้องการใช้กับอุปกรณ์ไฟฟ้าที่ใช้ระดับแรงดันต่ำๆ เช่น 6 V หรือ 9 V ก็จะต้องมีการแปลงดันไฟฟ้า ตามบ้านจาก 220 Vac เป็นระดับแรงดันไฟฟ้าตามที่ต้องการ อุปกรณ์ที่ทำหน้าที่ดังกล่าว เราเรียกว่า หม้อแปลงไฟฟ้า (Transformer)

2.6.2 วงจรเรียงกระแสเต็มคลื่นแบบบริดจ์ (Bridge Rectifier)

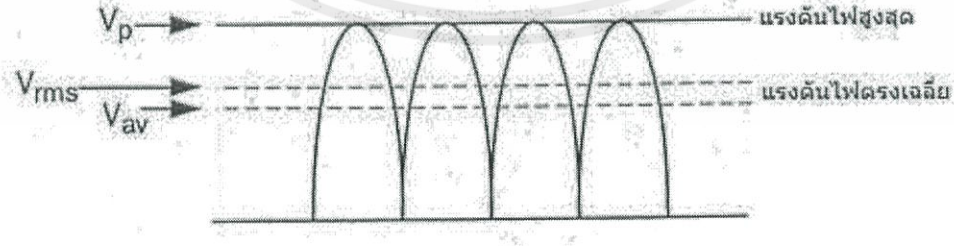
วงจรเรียงกระแสเต็มคลื่นแบบบริดจ์จะใช้ไดโอด 4 ตัวในการทำงานและสามารถใช้กับหม้อแปลงแบบ 2 หรือ 3 ขั้วก็ได้ โดยไดโอดจะผลัดกันนำกระแสครั้งละ 2 ตัว ซึ่งคลื่นเอาต์พุต ที่ออกมาจะมีลักษณะดังรูปที่ 2.17 ซึ่งค่าแรงดันเอาต์พุตที่ออกมานั้นจะมีค่าเป็น 0.636 เท่า ของแรงดันไฟสูงสุด



รูปที่ 2.17 วงจรเรียงกระแสเต็มคลื่นแบบบริดจ์



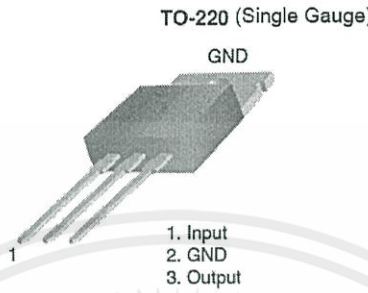
รูปที่ 2.18 รูปคลื่นแรงดันขาออกเปรียบเทียบกับแรงดันขาเข้า



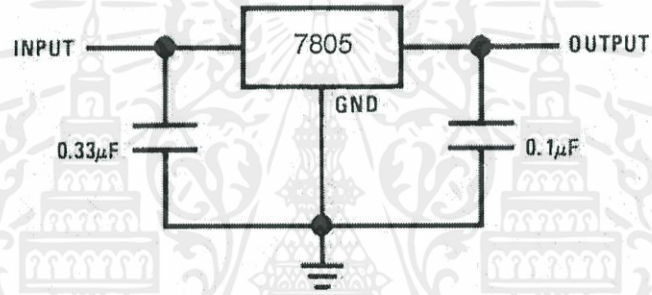
รูปที่ 2.19 แสดงค่าแรงดันไฟตรงเฉลี่ยกับค่าแรงดันไฟสูงสุด Vp

2.6.3 โวลท์เตจเรคกูเลเตอร์ไอซี

ในวงจรจ่ายไฟที่ต้องมีการควบคุมระดับแรงดันและจ่ายกระแสให้คงที่ นิยมใช้ไอซีสำเร็จรูปเป็นอย่างมาก อย่างเบอร์ที่ใช้งานในโครงงานอิเล็กทรอนิกส์ทั่วไปมักอยู่ในตระกูล 78XX และ 79XX เป็นไอซีเรคกูเลเตอร์แรงดันคงที่ใช้งานสำหรับแรงดันไฟบวกและลบ จ่ายกระแสสูงสุด 1 แอมแปร์ ใช้งานได้ง่าย ต่ออุปกรณ์ภายนอกน้อยชิ้น



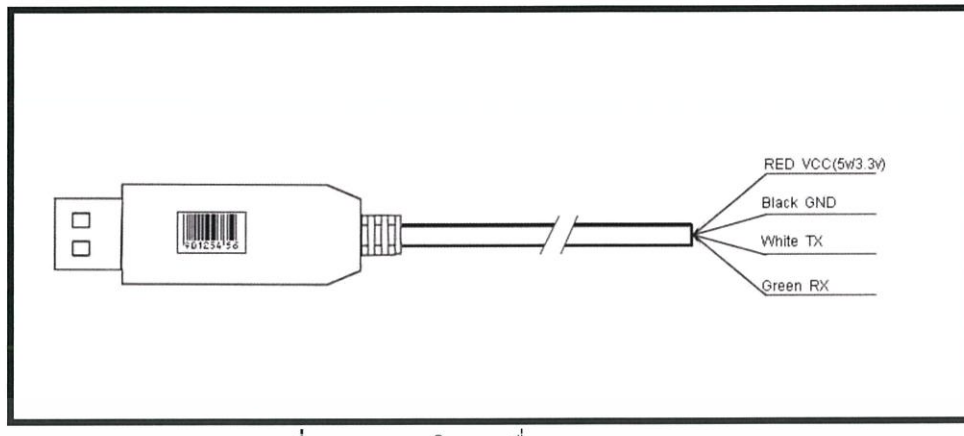
รูปที่ 2.20 โวลท์เตจเรคกูเลเตอร์ไอซีเบอร์ LM7805



รูปที่ 2.21 วงจรโวลท์เตจเรคกูเลเตอร์ไอซีเบอร์ LM7805

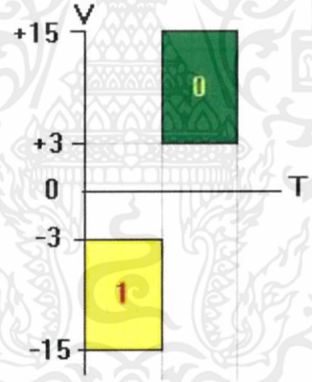
2.7 การสื่อสาร RS-232

การสื่อสาร RS-232 มาจากคำว่า Recommended Standard-232 (มาตรฐานแนะนำรุ่น 232) เป็นมาตรฐานการเชื่อม ต่อข้อมูล แบบอนุกรม (Serial Port) การสื่อสารแบบอนุกรม นับว่ามีความสำคัญ ต่อการใช้งาน ไมโครคอนโทรลเลอร์มาก เพราะสามารถใช้แป้นพิมพ์ และจอภาพของคอมพิวเตอร์ส่วนบุคคลเป็น อินพุต และ เอาต์พุต ในการติดต่อ หรือ ควบคุมไมโครคอนโทรลเลอร์ด้วยสัญญาณอย่างน้อย เพียง 3 เส้นเท่านั้น คือ สายส่งสัญญาณ TX, สายรับสัญญาณ RX, และสายกราวด์ GND สำหรับในโครงงานนี้ใช้สายในการสื่อสารแบบอนุกรมทั้งหมด 4 เส้นคือ สายส่งสัญญาณ TX, สายรับสัญญาณ RX, สายกราวด์ GND, และสาย VCC ที่ใช้เป็นแหล่งจ่ายพลังงานให้กับไมโครคอนโทรลเลอร์ ดังรูปที่ 2.22



รูปที่ 2.22 สายในการสื่อสารแบบอนุกรม

สัญญาณรบกวนที่เกิดขึ้น ในสายนำสัญญาณ มักจะมีแรงดันเป็นบวก เมื่อเทียบกับกราวด์ เพื่อป้องกันสัญญาณรบกวนนี้ จึงออกแบบแรงดัน ของโลจิก 1 เป็นลบ คืออยู่ในช่วง -3 V ถึง -15 V ส่วนแรงดัน ของโลจิก 0 อยู่ในช่วง +3 V ถึง +15 V และเหตุที่ ระดับสัญญาณ ของ RS232 อยู่ในช่วง +15 V ถึง -15 V ก็เพื่อให้ต่อสายสัญญาณไปได้ไกลขึ้นและมีอัตราการส่งข้อมูล (Baud rate) หรือความเร็วของการรับ-ส่งข้อมูล เป็นจำนวนบิตต่อวินาทีเช่น 300, 1,200, 2,400, 4,800, 9,600 ,14,400, 19,200, 38,400, 56,000 เป็นต้น



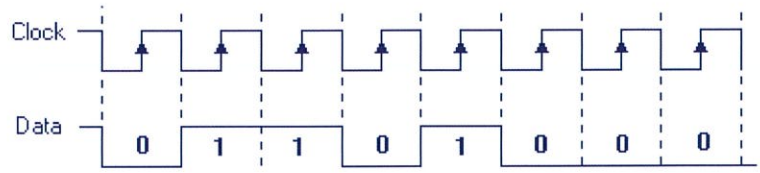
รูปที่ 2.23 ระดับสัญญาณ RS-232

รูปแบบการสื่อสารแบบอนุกรม

มีด้วยกันอยู่ 2 แบบ คือแบบซิงโครนัส (Synchronous) และแบบอะซิงโครนัส(Asynchronous)

1. การสื่อสารแบบซิงโครนัส (Synchronous)

การรับส่งข้อมูล จะมีสัญญาณนาฬิกา ซึ่งเป็นตัวกำหนด จังหวะเวลา การส่งข้อมูล ร่วมอยู่ด้วยอีกเส้นหนึ่ง ใช้คู่กับสัญญาณข้อมูล ตัวอย่างเช่น การส่งสัญญาณจากคีย์บอร์ด

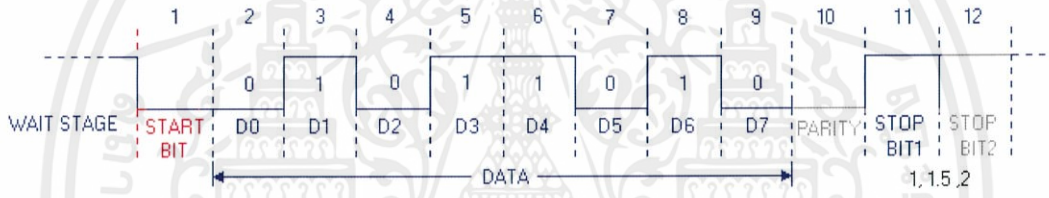


รูปที่ 2.24 สัญญาณการสื่อสารแบบซิงโครนัส

2. การสื่อสารแบบอะซิงโครนัส (Asynchronous)

การรับส่งข้อมูล โดยที่ไม่จำเป็นต้อง มีสัญญาณนาฬิกา ร่วมด้วย แต่จะใช้ให้ตัวส่ง และตัวรับมีอัตราส่งข้อมูล ที่เท่ากันรูปแบบข้อมูลแบบอะซิงโครนัส ประกอบด้วย 4 ส่วนคือ

1. บิตเริ่มต้น (Start bit) มีขนาด1 บิต
2. บิตข้อมูล (Data) มีขนาด 5, 6, 7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity bit) มีขนาด 1 บิตหรือไม่มี
4. บิตหยุด (Stop bit) มีขนาด 1, 1.5, 2 บิต



รูปที่ 2.25 สัญญาณการสื่อสารแบบอะซิงโครนัส

- เมื่อไม่มีการส่งข้อมูล ขา data จะมีสถานะเป็นโลจิก 1 หรือ สถานะหยุดรอ (Waiting stage)
- เมื่อเริ่มต้นส่งข้อมูลจะให้ขา data เป็นโลจิก 0 เป็นจำนวน 1 บิต เรียกว่าบิตเริ่มต้น (Start bit)
- จากนั้นก็จะเริ่มต้นส่งข้อมูล โดยส่งบิตต่ำไปก่อน (LSB)
- แล้วตามด้วยพาริตี้บิต (จะมีหรือไม่ก็ได้ ขึ้นอยู่กับการติดตั้งค่า ของทั้งสองฝ่าย)
- สุดท้ายตามด้วยโลจิก 1 อย่างน้อย 1 บิต (มีขนาด 1, 1.5, หรือ 2 บิต) เพื่อแสดงว่าสิ้นสุดข้อมูล

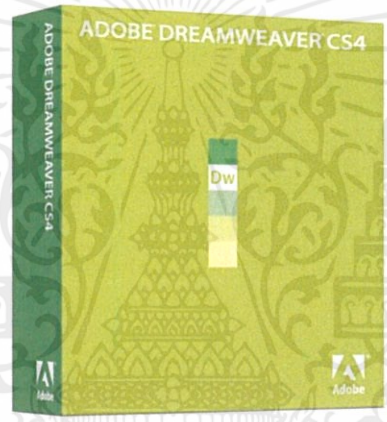
2.8 หลักออกแบบเว็บไซต์

การสร้างเว็บไซต์ต้องมีการจัดเตรียมข้อมูล ออกแบบหน้าเว็บเพจ ให้มีรูปแบบที่สวยงาม สามารถใช้งานได้ง่าย และมีความสะดวกต่อการเข้าถึงข้อมูล การออกแบบเว็บไซต์จึงมีความสำคัญในการสร้างความประทับใจให้กับผู้ที่เข้ามาใช้บริการ ทำให้ผู้ใช้งานเข้าใจและเข้าถึงสิ่งที่ต้องการนำเสนอและต้องการสื่อออกมาด้วย

2.8.1 การออกแบบ

1. ควรมีจุดเด่นหรือเอกลักษณ์ เนื่องจากเว็บไซต์แต่ละเว็บไซต์ก็มีเป้าหมายที่แตกต่าง ต้องหาจุดเด่นหรือเอกลักษณ์ เพื่อให้ผู้ใช้งานจดจำและเข้าใจง่าย
2. ควรมีการจัดวางข้อมูลที่เรียบง่าย เพื่อให้ผู้ที่เข้ามาใช้บริการสามารถอ่านเนื้อหาที่อยู่ภายในแต่ละเว็บเพจได้ง่าย สบายตา การจัดวางข้อมูลภายในเว็บไซต์ต้องไม่ซับซ้อน มีการจัดการที่ดีเป็นระบบ หมวดหมู่ให้กับข้อมูล ภาพกราฟิก ตัวอักษร อย่างเหมาะสม เพื่อให้ผู้เข้ามาใช้บริการสามารถเข้าถึงเนื้อหาที่ใช้งานได้อย่างสะดวกและรวดเร็ว

โปรแกรมในการสร้างเว็บไซต์มีด้วยกันหลายโปรแกรม ยกตัวอย่างเช่น Joomla , Drupal Opensouce , NetBeans , Dreamweaver เป็นต้น โดยในที่นี้ ได้เลือกโปรแกรม Dreamweaver มาในการสร้างตัวเว็บไซต์นี้ โดยมีขั้นตอนในการดาวน์โหลดและติดตั้งโปรแกรม Dreamweaver ดังนี้ โดยในที่นี้ได้เลือก Version CS4



รูปที่ 2.22 โปรแกรม Dreamweaver CS4

2.8.2 ระบบฐานข้อมูล

ฐานข้อมูล (database) หมายถึง กลุ่มของข้อมูลที่ถูกเก็บรวบรวมไว้ โดยมีความสัมพันธ์ซึ่งกันและกัน โดยไม่ได้บังคับว่าข้อมูลทั้งหมดนี้จะต้องเก็บไว้ในแฟ้มข้อมูลเดียวกันหรือแยกเก็บหลาย ๆ แฟ้มข้อมูล นั่นก็คือการเก็บข้อมูลในฐานข้อมูลนั้นเราอาจจะเก็บทั้งฐานข้อมูล โดยใช้แฟ้มข้อมูลเพียงแฟ้มข้อมูลเดียวกันได้ หรือจะเก็บไว้ในหลาย ๆ แฟ้มข้อมูล ที่สำคัญคือจะต้องสร้างความสัมพันธ์ระหว่างระเบียนและเรียกใช้ความสัมพันธ์นั้นได้ มีการกำจัดความซ้ำซ้อนของข้อมูลออกและเก็บแฟ้มข้อมูลเหล่านี้ไว้ที่ศูนย์กลาง เพื่อที่จะนำข้อมูลเหล่านี้มาใช้ร่วมกัน ควบคุมดูแลรักษาเมื่อผู้ต้องการใช้งานและผู้มีสิทธิ์จะใช้ข้อมูลนั้นสามารถดึงข้อมูลที่ต้องการออกไปใช้ได้ ข้อมูลบางส่วนอาจใช้ร่วมกับผู้อื่นได้ แต่บางส่วนผู้มีสิทธิ์เท่านั้นจึงจะสามารถใช้ได้ โดยทั่วไปองค์กรต่าง ๆ จะสร้างฐานข้อมูลไว้ เพื่อเก็บข้อมูลต่าง ๆ ของตัวองค์กร โดยเฉพาะอย่างยิ่งข้อมูลในเชิงธุรกิจ เช่น ข้อมูลของลูกค้า ข้อมูลของสินค้า ข้อมูลของลูกจ้าง และการจ้างงาน เป็นต้น การควบคุมดูแลการใช้ฐานข้อมูลนั้น เป็นเรื่องที่ยุ่งยากกว่าการใช้แฟ้มข้อมูลมาก เพราะเราจะต้องตัดสินใจว่าโครงสร้างในการจัดเก็บข้อมูลควรจะเป็นเช่นไร การเขียนโปรแกรมเพื่อสร้างและเรียกใช้ข้อมูลจากโครงสร้างเหล่านี้ ถ้า

โปรแกรมเหล่านี้เกิดทำงานผิดพลาดขึ้นมา ก็จะเกิดความเสียหายต่อโครงสร้างของข้อมูลทั้งหมดได้ เพื่อเป็นการลดภาระการทำงานของผู้ใช้ จึงได้มีส่วนของฮาร์ดแวร์และโปรแกรมต่าง ๆ ที่สามารถเข้าถึงและจัดการข้อมูลในฐานข้อมูลนั้น เรียกว่า ระบบจัดการฐานข้อมูล หรือDBMS (data base management system) ระบบจัดการฐานข้อมูล คือ ซอฟต์แวร์ที่เปรียบเสมือนสื่อกลางระหว่างผู้ใช้และโปรแกรมต่าง ๆ ที่เกี่ยวข้องกับการใช้ฐานข้อมูล ซึ่งมีหน้าที่ช่วยให้ผู้ใช้เข้าถึงข้อมูลได้ง่ายสะดวกและมีประสิทธิภาพ การเข้าถึงข้อมูลของผู้ใช้อาจเป็นการสร้างฐานข้อมูล การแก้ไขฐานข้อมูล หรือการตั้งคำถามเพื่อให้ข้อมูลมา โดยผู้ใช้ไม่จำเป็นต้องรับรู้เกี่ยวกับรายละเอียดภายในโครงสร้างของฐานข้อมูล เปรียบเสมือนเป็นสื่อกลางระหว่างผู้ใช้และโปรแกรมต่าง ๆ ที่เกี่ยวข้องกับการใช้ฐานข้อมูล

ความสำคัญของระบบฐานข้อมูล

การจัดข้อมูลให้เป็นระบบฐานข้อมูลทำให้ข้อมูลมีส่วนดีกว่าการเก็บข้อมูลในรูปของแฟ้มข้อมูล เพราะการจัดเก็บข้อมูลในระบบฐานข้อมูล จะมีส่วนที่สำคัญกว่าการจัดเก็บข้อมูลในรูปของแฟ้มข้อมูลดังนี้

1. ลดการเก็บข้อมูลที่ซ้ำซ้อน ข้อมูลบางชุดที่อยู่ในรูปของแฟ้มข้อมูลอาจมีปรากฏอยู่หลาย ๆ แห่ง เพราะมีผู้ใช้ข้อมูลชุดนี้หลายคน เมื่อใช้ระบบฐานข้อมูลแล้วจะช่วยให้ความซ้ำซ้อนของข้อมูลลดน้อยลง เช่น ข้อมูลอยู่ในแฟ้มข้อมูลของผู้ใช้หลายคน ผู้ใช้แต่ละคนจะมีแฟ้มข้อมูลเป็นของตนเอง ระบบฐานข้อมูลจะลดการซ้ำซ้อนของข้อมูลเหล่านี้ให้มากที่สุด โดยจัดเก็บในฐานข้อมูลไว้ที่เดียวกัน ผู้ใช้ทุกคนที่ต้องการใช้ข้อมูลชุดนี้จะใช้โดยผ่านระบบฐานข้อมูล ทำให้ไม่เปลืองเนื้อที่ในการเก็บข้อมูลและลดความซ้ำซ้อนลงได้

2. รักษาความถูกต้องของข้อมูล เนื่องจากฐานข้อมูลมีเพียงฐานข้อมูลเดียว ในกรณีที่มีข้อมูลชุดเดียวกันปรากฏอยู่หลายแห่งในฐานข้อมูล ข้อมูลเหล่านี้จะต้องตรงกัน ถ้ามีการแก้ไขข้อมูลนี้ทุก ๆ แห่งที่ข้อมูลปรากฏอยู่จะแก้ไขให้ถูกต้องตามกันหมดโดยอัตโนมัติด้วยระบบจัดการฐานข้อมูล

3. การป้องกันและรักษาความปลอดภัยให้กับข้อมูลทำได้อย่างสะดวก การป้องกันและรักษาความปลอดภัยกับข้อมูลระบบฐานข้อมูลจะให้เฉพาะผู้ที่เกี่ยวข้องเท่านั้นจึงจะมีสิทธิ์เข้าไปใช้ฐานข้อมูลได้เรียกว่ามีสิทธิส่วนบุคคล (privacy) ซึ่งก่อให้เกิดความปลอดภัย (security) ของข้อมูลด้วย ฉะนั้นผู้ใดจะมีสิทธิ์ที่จะเข้าถึงข้อมูลได้จะต้องมีการกำหนดสิทธิ์กันไว้ก่อนและเมื่อเข้าไปใช้ข้อมูลนั้น ๆ ผู้ใช้จะเห็นข้อมูลที่ถูกเก็บไว้ในฐานข้อมูลในรูปแบบที่ผู้ใช้ออกแบบไว้
ตัวอย่างเช่น ผู้ใช้สร้างตารางข้อมูลขึ้นมาและเก็บลงในระบบฐานข้อมูล ระบบจัดการฐานข้อมูลจะเก็บข้อมูลเหล่านี้ลงในอุปกรณ์เก็บข้อมูลในรูปแบบของระบบจัดการฐานข้อมูลซึ่งอาจเก็บข้อมูลเหล่านี้ลงในแผ่นจานบันทึกแม่เหล็กเป็นระเบียน บล็อกหรืออื่น ๆ ผู้ใช้ไม่จำเป็นต้องรับรู้ว่าโครงสร้างของแฟ้มข้อมูลนั้นเป็นอย่างไร ปล่อยให้เป็นหน้าที่ของระบบจัดการฐานข้อมูล

4. สามารถใช้ข้อมูลร่วมกันได้ เนื่องจากในระบบฐานข้อมูลจะเป็นที่เก็บรวบรวมข้อมูลทุกอย่างไว้ ผู้ใช้แต่ละคนจึงสามารถที่จะใช้ข้อมูลในระบบได้ทุกข้อมูล ซึ่งถ้าข้อมูลไม่ได้ถูกจัดให้

เป็นระบบฐานข้อมูลแล้ว ผู้ใช้ก็จะใช้ได้เพียงข้อมูลของตนเองเท่านั้น เช่น ดังภาพที่ 4.9 ข้อมูลของระบบเงินเดือน ข้อมูลของระบบงานบุคคลถูกจัดไว้ในระบบแฟ้มข้อมูลผู้ใช้ที่ใช้ข้อมูลระบบเงินเดือนจะใช้ข้อมูลได้ระบบเดียว แต่ถ้าข้อมูลทั้ง 2 ถูกเก็บไว้เป็นฐานข้อมูลซึ่งถูกเก็บไว้ในที่ที่เดียวกัน ผู้ใช้ทั้ง 2 ระบบก็จะสามารถเรียกใช้ฐานข้อมูลเดียวกันได้ ไม่เพียงแต่ข้อมูลเท่านั้นสำหรับโปรแกรมต่าง ๆ ถ้าเก็บไว้ในฐานข้อมูลก็จะสามารถใช้ร่วมกันได้

5. มีความเป็นอิสระของข้อมูล เมื่อผู้ใช้ต้องการเปลี่ยนแปลงข้อมูลหรือนำข้อมูลมาประยุกต์ใช้ให้เหมาะสมกับโปรแกรมที่เขียนขึ้นมา จะสามารถสร้างข้อมูลนั้นขึ้นมาใช้ใหม่ได้ โดยไม่มีผลกระทบต่อระบบฐานข้อมูล เพราะข้อมูลที่ผู้ใช้นำมาประยุกต์ใช้ใหม่นั้นจะไม่กระทบต่อโครงสร้างที่แท้จริงของการจัดเก็บข้อมูล นั่นคือ การใช้ระบบฐานข้อมูลจะทำให้เกิดความเป็นอิสระระหว่างการจัดเก็บข้อมูลและการประยุกต์ใช้

6. สามารถขยายงานได้ง่าย เมื่อต้องการจัดเพิ่มเติมข้อมูลที่เกี่ยวข้องจะสามารถเพิ่มได้อย่างง่ายไม่ซับซ้อน เนื่องจากมีความเป็นอิสระของข้อมูล จึงไม่มีผลกระทบต่อข้อมูลเดิมที่มีอยู่

7. ทำให้ข้อมูลบูรณะกลับสู่สภาพปกติได้เร็วและมีมาตรฐาน เนื่องจากการจัดพิมพ์ข้อมูลในระบบที่ไม่ได้ใช้ฐานข้อมูล ผู้เขียนโปรแกรมแต่ละคนมีแฟ้มข้อมูลของตนเองเฉพาะ ฉะนั้นแต่ละคนจึงต่างก็สร้างระบบการบูรณะข้อมูลให้กลับสู่สภาพปกติในกรณีที่ข้อมูลเสียหายด้วยตนเองและด้วยวิธีการของตนเอง จึงขาดประสิทธิภาพและมาตรฐาน แต่เมื่อมาเป็นระบบฐานข้อมูลแล้ว การบูรณะข้อมูลให้กลับคืนสู่สภาพปกติจะมีโปรแกรมชุดเดียวและมีผู้ดูแลเพียงคนเดียวที่ดูแลทั้งระบบ ซึ่งย่อมต้องมีประสิทธิภาพและเป็นมาตรฐานเดียวกันแน่นอน

2.8.3 ระบบการจัดการฐานข้อมูล

หน้าที่ของระบบการจัดการฐานข้อมูล

1. ระบบจัดการฐานข้อมูลเป็นซอฟต์แวร์ที่ทำหน้าที่ดังต่อไปนี้ ดูแลการใช้งานให้กับผู้ใช้ ในการติดต่อกับตัวจัดการระบบแฟ้มข้อมูลได้ ในระบบฐานข้อมูลนี้ข้อมูลจะมีขนาดใหญ่ ซึ่งจะถูกจัดเก็บไว้ในหน่วยความจำสำรองเมื่อผู้ใช้ต้องการจะใช้ฐานข้อมูล ระบบการจัดการฐานข้อมูลจะทำหน้าที่ติดต่อกับระบบแฟ้มข้อมูลซึ่งเสมือนเป็นผู้จัดการแฟ้มข้อมูล (file manager) นำข้อมูลจากหน่วยความจำสำรองเข้าสู่หน่วยความจำหลักเฉพาะส่วนที่ต้องการใช้งาน และทำหน้าที่ประสานกับตัวจัดการระบบแฟ้มข้อมูลในการจัดเก็บ เรียกใช้ และแก้ไขข้อมูล

2. ควบคุมระบบความปลอดภัยของข้อมูลโดยป้องกันไม่ให้ผู้ที่ไม่ได้รับอนุญาตเข้ามาเรียกใช้หรือแก้ไขข้อมูลในส่วนป้องกันเอาไว้ พร้อมทั้งสร้างฟังก์ชันในการจัดทำข้อมูลสำรอง โดยเมื่อเกิดมีความขัดข้องของระบบแฟ้มข้อมูลหรือของเครื่องคอมพิวเตอร์เกิดการเสียหายนั้น ฟังก์ชันนี้จะสามารถทำการฟื้นสภาพของระบบข้อมูลกลับเข้าสู่สภาพที่ถูกต้องสมบูรณ์ได้

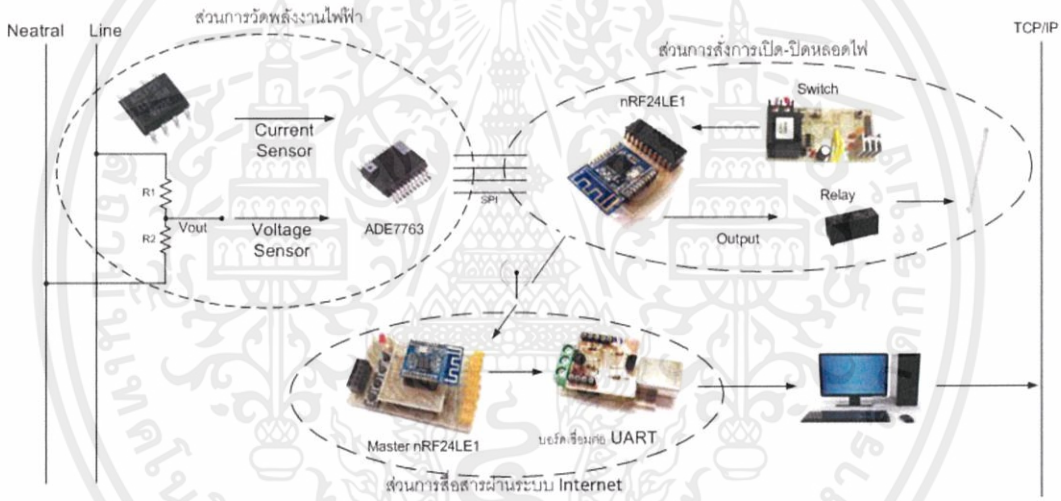
3. ควบคุมการใช้ข้อมูลในสภาพที่มีผู้ใช้พร้อม ๆ กันหลายคน โดยจัดการเมื่อมีข้อผิดพลาดของข้อมูลเกิดขึ้น

บทที่ 3
การออกแบบการควบคุมอุปกรณ์ส่องสว่าง

3.1 กล่าวนำ

ในปัจจุบันระบบไฟฟ้ายังคงต้องใช้สายไฟส่งแรงดันไฟฟ้ายังตัวอุปกรณ์ส่องสว่างและใช้สวิตซ์ในการเปิด- ปิด เพื่อให้อุปกรณ์ส่องสว่างสามารถประยุกต์ใช้งานร่วมกับเทคโนโลยีแบบไร้สายได้นั้น ได้ออกแบบโมดูลควบคุมอุปกรณ์เพื่อให้ผู้ใช้สามารถนำไปติดตั้งเข้ากับระบบในปัจจุบันได้ง่าย และยังสามารถวัดปริมาณทางไฟฟ้า โดยโมดูลควบคุมมีช่องทางการสื่อสารผ่าน RS-232 ใช้เชื่อมต่อกับคอมพิวเตอร์ส่วนบุคคลที่มีโปรแกรมควบคุมและแสดงผลผ่านทางเว็บเบราว์เซอร์

3.2 แผนภาพการวางแผนโครงสร้างของโครงงาน



รูปที่ 3.1 แผนภาพรวมโครงงาน

3.3 การออกแบบโมดูลควบคุมอุปกรณ์ส่องสว่าง

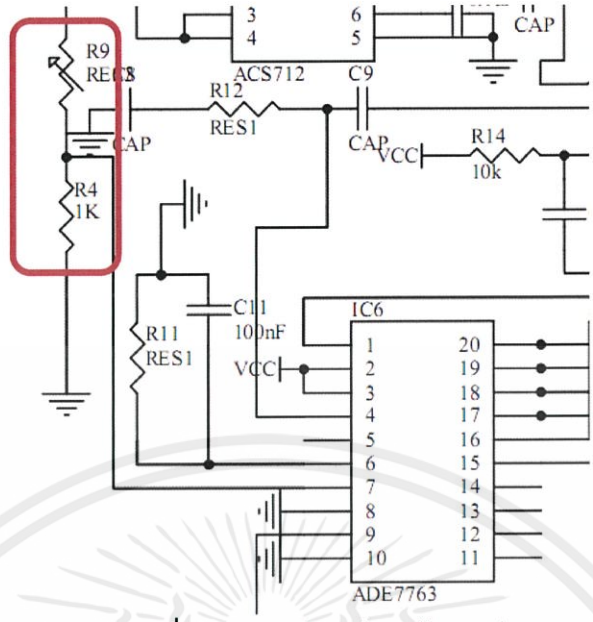
โมดูลควบคุมอุปกรณ์ส่องสว่างนั้นแบ่งออกเป็น 2 ส่วนคือ โมดูลส่วนสั่งการเปิด- ปิดหลอดไฟ และโมดูลส่วนการสื่อสารผ่านทางระบบอินเทอร์เน็ต

3.3.1 โมดูลส่วนสั่งการเปิด- ปิดหลอดไฟ

โดยโมดูลส่วนสั่งการเปิด- ปิดหลอดไฟนั้นประกอบด้วยวงจรดังนี้

3.3.1.1 โวลท์เตจดีไวเดอร์

พาวเวอร์ไอซีชิป ADE7763 ที่ได้เลือกมานั้นสามารถรับโวลท์เตจอินพุต ได้ไม่เกิน ± 0.5 V จึงได้ใช้วงจรโวลท์เตจดีไวเดอร์ (Voltage Divider) ทำหน้าที่ในส่วนการหารแรงดันลงมา เนื่องจากแรงดันที่ทำการวัดนั้นเป็นแรงดันไฟฟ้าในบ้าน หรือ 220 Vrms และจากสมการการคำนวณวงจรโวลท์เตจดีไวเดอร์ เมื่อกำหนดค่า R1 = 1 MΩ และ R2 = 1 kΩ จะทำให้ได้ค่า Vout = 0.28 Vrms ซึ่งจะทำให้ พาวเวอร์ไอซีสามารถทำงานได้ปกติ

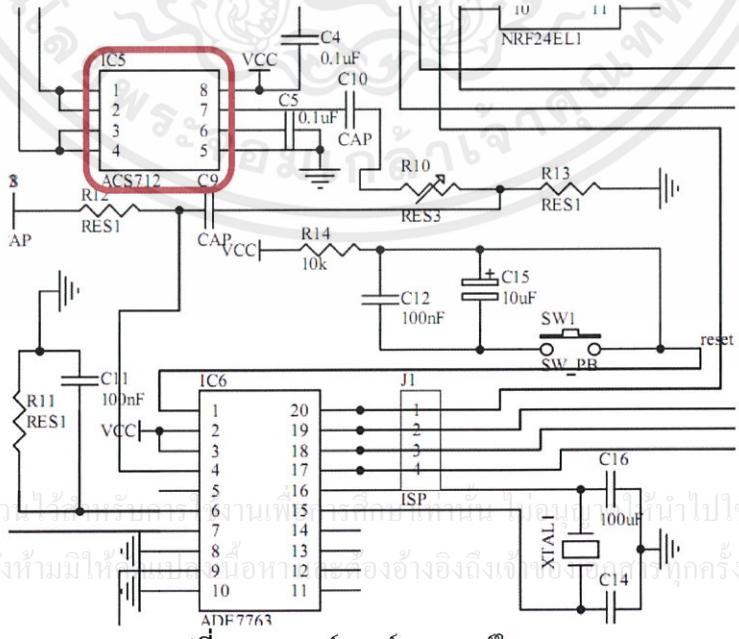


รูปที่ 3.2 วงจรโวลท์เตจดีไวเดอร์

3.3.1.2 เคอร์เรนท์เซนเซอร์

ในส่วนของ เคอร์เรนท์เซนเซอร์ (Current Sensor) นั้นจะทำการวัดค่ากระแสที่โหลดใช้ และเนื่องจาก พาวเวอร์ไอซีชิ ADE7763 ที่ได้เลือกมานั้นไม่สามารถรับค่ากระแสเข้ามาที่ตัวของพาวเวอร์ไอซีได้เลย ดังนั้นจึงต้องทำการแปลงกระแสที่ต้องการวัดมาเป็นแรงดันทางไฟฟ้า และแรงดันทางไฟฟ้าที่แปลงมานั้นก็ต้องไม่เกิน ± 0.5 V จึงต้องมีวงจรหารแรงดันทางไฟฟ้าจากตัว ACS712 ซึ่งเป็นเคอร์เรนท์เซนเซอร์ลงมาอีกครั้ง

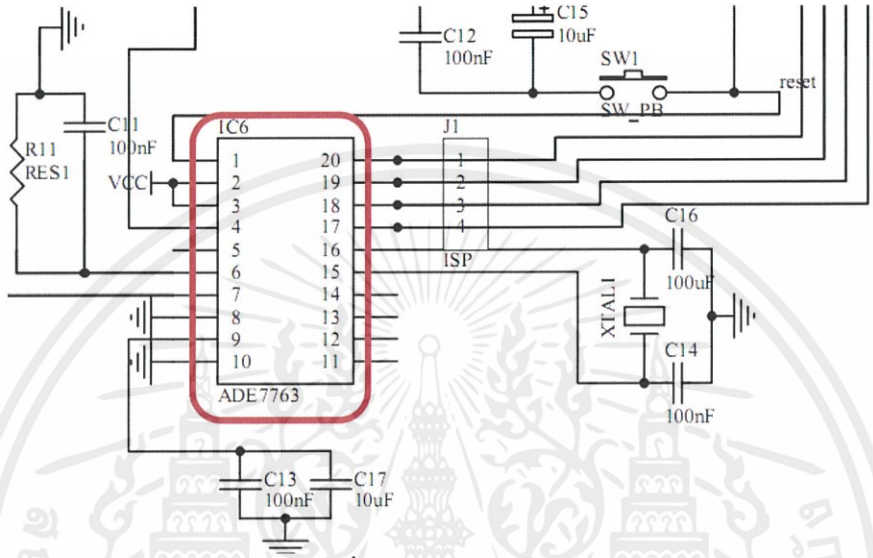
เนื่องจาก เคอร์เรนท์เซนเซอร์ ACS712 มีค่า Sensitivity = 66 mV/A ดังนั้นสามารถส่งค่ากระแสไปให้พาวเวอร์ไอซี มากที่สุดและยังสามารถทำงานได้ปกติ คือ 7.57 A



รูปที่ 3.3 เคอร์เรนท์เซนเซอร์ในวงจร

3.3.1.3 พาวเวอร์ไอซี

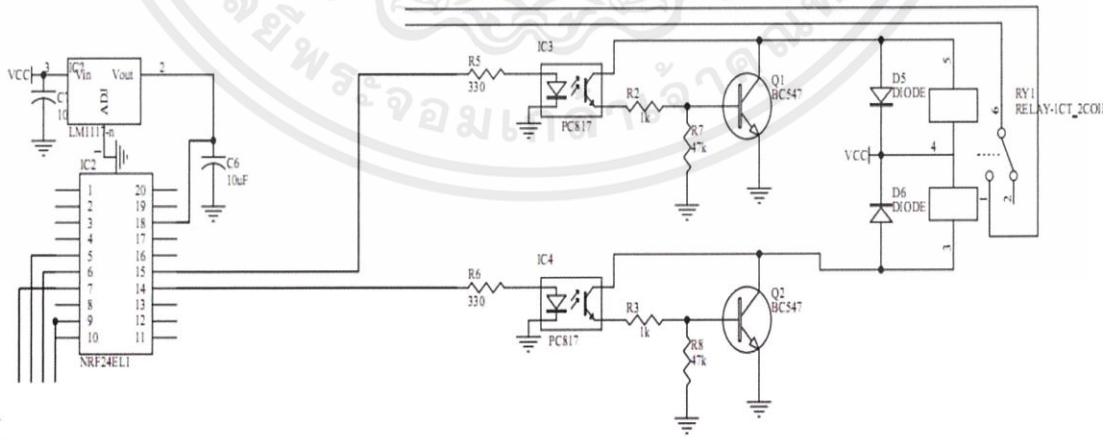
พาวเวอร์ไอซี (Power IC) ถือเป็นอุปกรณ์สำคัญในการรับค่าจากการวัดและนำมาคำนวณค่าพลังงานทางไฟฟ้าเพื่อให้ได้ค่าพลังงานทางไฟฟ้าในรูปแบบต่างๆ เช่น รับค่าจากเคอร์เรนท์เซนเซอร์ และคำนวณค่ากระแส หรือ Irms รับค่าจากวงจรโวลท์เตจดีไวเดอร์ และคำนวณค่าแรงดัน หรือ Vrms คำนวณค่ากำลังไฟฟ้าได้ทั้งในหน่วยของ Watt และ VA สามารถหาค่าพลังงานไฟฟ้าในหน่วยของ kWhr เป็นต้น



รูปที่ 3.4 พาวเวอร์ไอซีในวงจร

3.3.1.4 วงจรขับรีเลย์

วงจรขับรีเลย์มีหน้าที่ตัดต่อไฟฟ้าในส่วนที่ต้องการควบคุม โดยรับคำสั่งจากไมโครคอนโทรลเลอร์ โดยรีเลยที่ใช้นั้นเป็นแบบแลตชิ่งรีเลย์ ซึ่งหน้าสัมผัสจะเปลี่ยนแปลงเมื่อมีไฟฟ้ามากระตุ้น

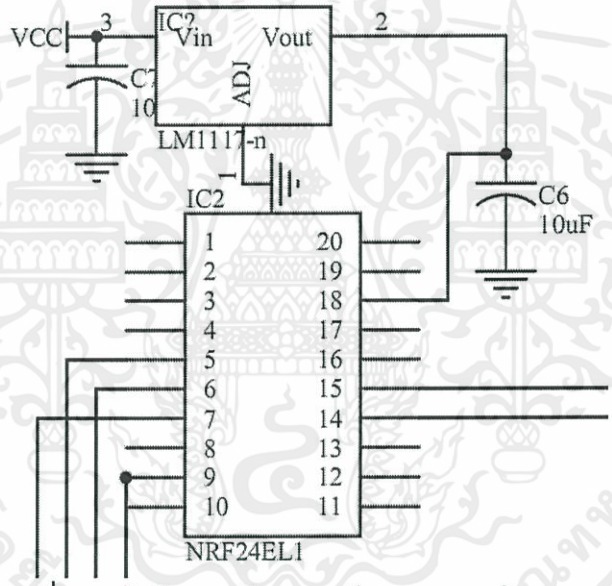


รูปที่ 3.5 วงจรขับรีเลย์ที่รับคำสั่งจากไมโครคอนโทรลเลอร์

3.3.1.5 ไมโครคอนโทรลเลอร์ (nRF24le1)

หน้าที่ของไมโครคอนโทรลเลอร์ในส่วนการสั่งการเปิด-ปิดหลอดไฟนั้นจะทำหน้าที่ควบคุมการทำงานของรีเลย์ การควบคุมการทำงานนั้นไมโครคอนโทรลเลอร์จะรับคำสั่งมาจากอุปกรณ์สั่งการอื่น ๆ ได้แก่ สวิตซ์ไวร์เลส หรือ รีโมทสั่งการ เมื่อได้รับคำสั่งมาแล้วนั้นจะทำการประมวลผลว่าคำสั่งที่ได้รับนั้นคือคำสั่งเปิดหลอดไฟหรือปิดหลอดไฟ เมื่อได้ผลลัพธ์จากการประมวลผลแล้วนั้นก็จะส่งคำสั่งไปสั่งการยังแลตชิ่งรีเลย์ต่อไป

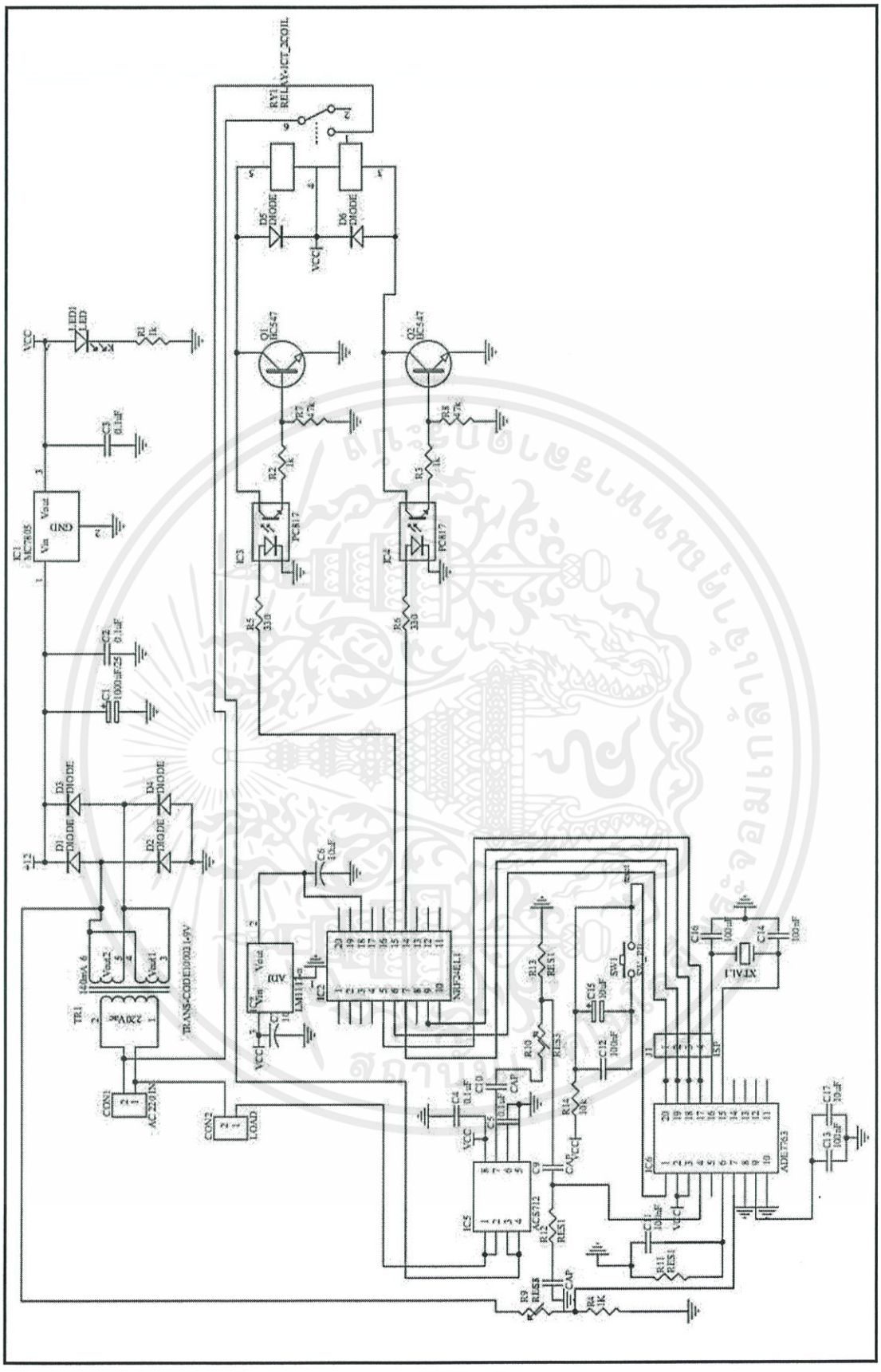
ไมโครคอนโทรลเลอร์ (nRF24le1) ยังมีหน้าที่ส่วนของการวัดค่าพลังงานทางไฟฟ้า จะทำหน้าที่รับค่าพลังงานทางไฟฟ้าที่สามารถวัดมาได้โดยค่าพลังงานทางไฟฟ้าที่วัดได้นั้นจะเข้ามายัง พาวเวอร์ไอซี ADE7763 เพื่อทำการคำนวณค่าในรูปแบบอื่นๆ และค่าที่ได้จากพาวเวอร์ไอซี ADE7763 จะส่งต่อเข้ามายังไมโครคอนโทลเลอร์ (nRF24le1) ซึ่งการส่งค่าจากพาวเวอร์ไอซีมายังไมโครคอนโทรลเลอร์นั้นจะใช้การส่งในรูปแบบ Serial Peripheral Interface หรือ SPI นั่นเอง นอกจากนั้นยังต้องทำการคำนวณค่าพลังงานทางไฟฟ้าที่รับมา เพื่อทำการส่งต่อไปยังส่วนของการแสดงผลต่อไป



รูปที่ 3.6 ไมโครคอนโทรลเลอร์ nRF24LE1 ในวงจร

3.3.1.6 วงจรโดยรวม

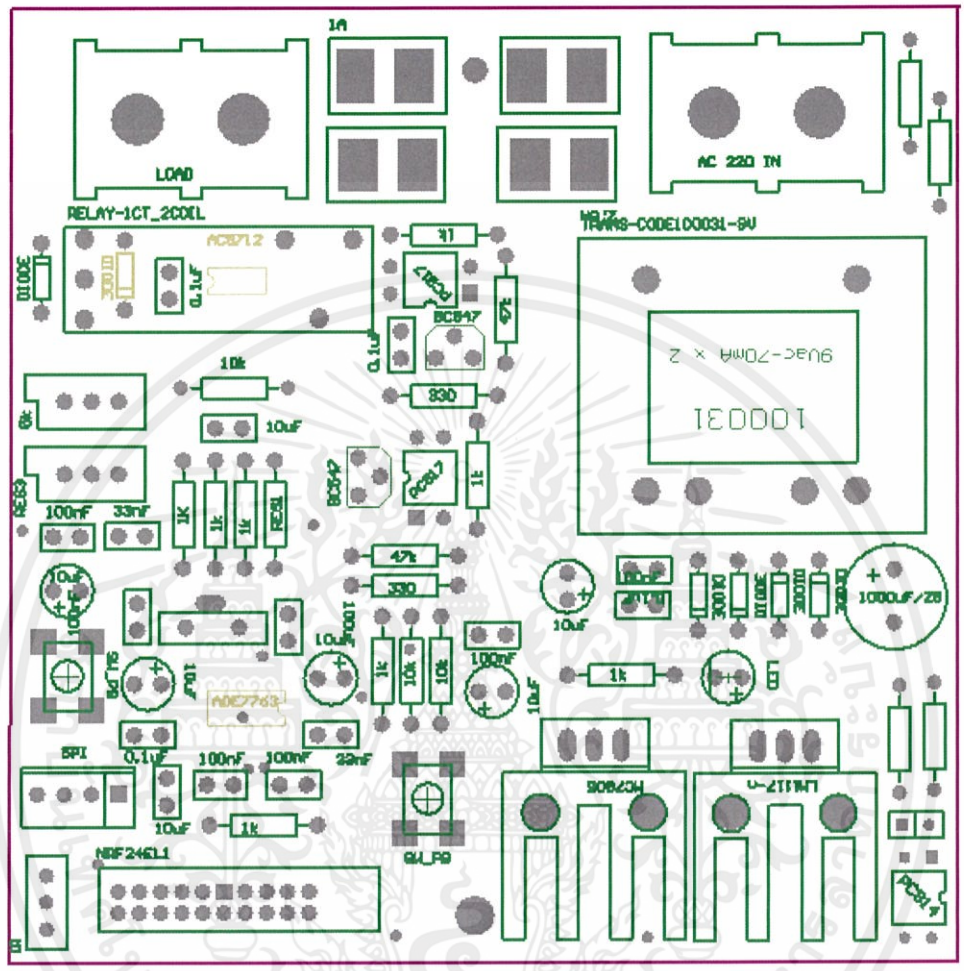
เมื่อออกแบบวงจรการใช้งานในโมดูลสั่งการเปิด- ปิดในแต่ละส่วนวนครบแล้ว จึงนำวงจรทั้งหมดมาเขียนแบบวงจรการทำงานของทุกส่วนร่วมกัน ดังรูปที่ 3.6



รูปที่ 3.7 วงจรโดยรวม

3.3.1.7 การออกแบบแผ่นลายวงจร

จากวงจรโดยรวมตามรูปที่ 3.6 ได้นำมาออกแบบลายวงจรที่ใช้งานด้วยโปรแกรม Protel

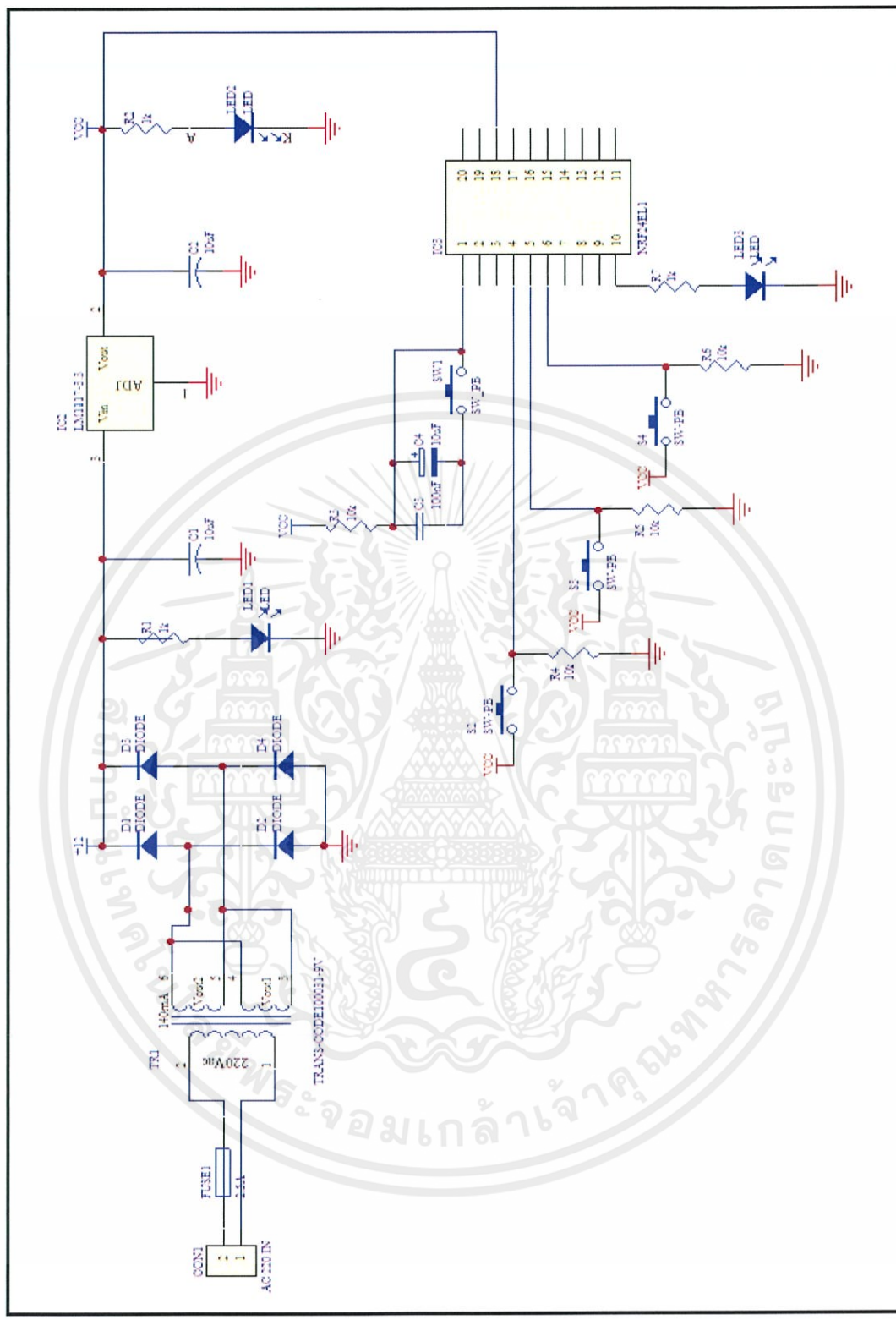


รูปที่ 3.8 ลายวงจรที่ออกแบบ

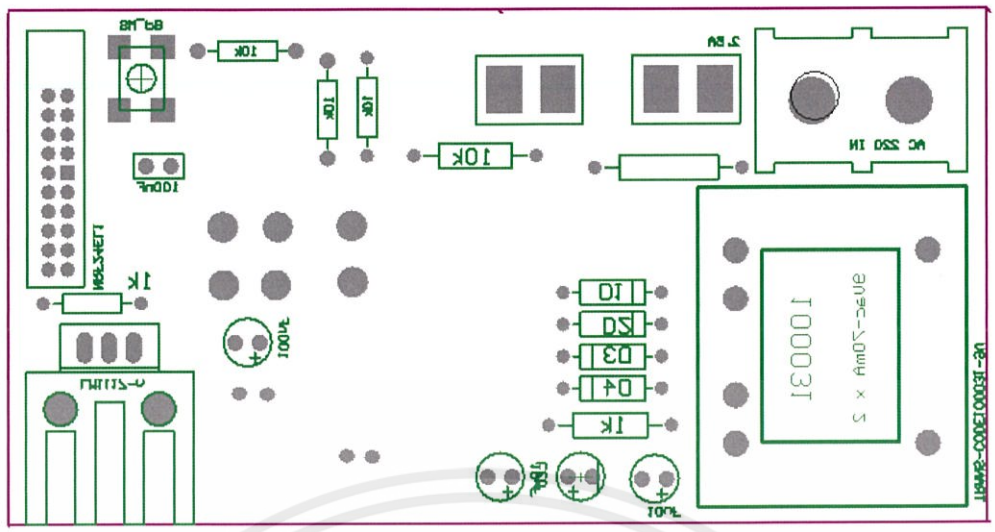
รูปที่ 3.9 วงจรที่ใช้งาน

3.3.1.8 สวิตซ์ไมโครคอนโทรลเลอร์ (nRF24le1)

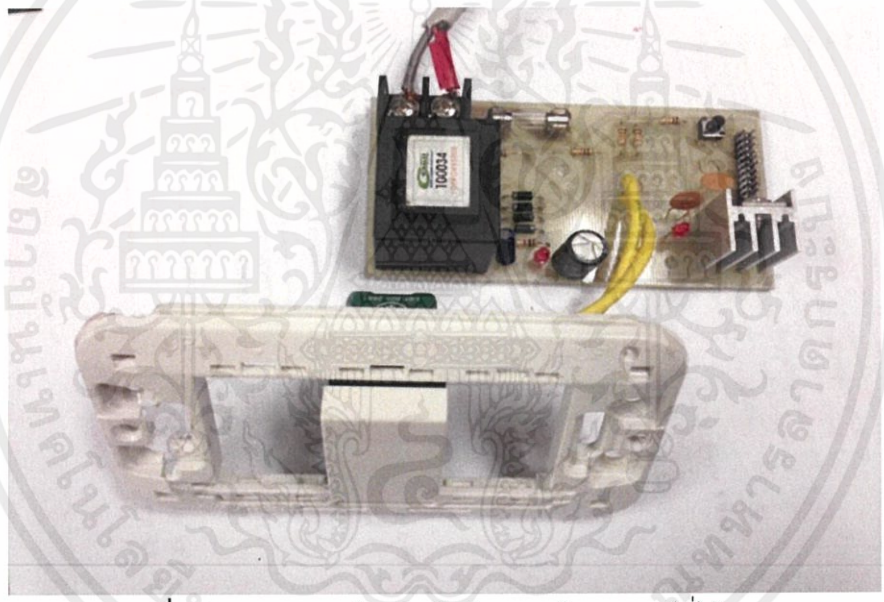
บอร์ดสวิตซ์ไมโครคอนโทรลเลอร์จะมีลักษณะเหมือนบอร์ดสวิตซ์ไฟบ้านทั่วไป และก็ยังทำหน้าที่เหมือนสวิตซ์ไฟบ้านทั่วไปคือสามารถทำการสั่งการเปิดหรือปิดหลอดไฟภายในบ้านได้ปกติด้วย แต่ส่วนที่แตกต่างกันคือบอร์ดสวิตซ์ไมโครคอนโทรลเลอร์นี้จะใช้ระบบการส่งคำสั่งที่ได้จากผู้ใช้นั่นก็คือเมื่อผู้ใช้กดเปิดหรือปิดสวิตซ์สัญญาณจะเข้าไปที่ไมโครคอนโทรลเลอร์ เมื่อไมโครคอนโทรลเลอร์ประมวลผลเสร็จก็จะส่งคำสั่งไปยังโมดูลหรือไมโครคอนโทรลเลอร์อีกตัวที่ติดอยู่กับหลอดไฟแบบไร้สาย ด้วยความแตกต่างจากการส่งคำสั่งนั้นทำให้บอร์ดสวิตซ์ไมโครคอนโทรลเลอร์สามารถเปลี่ยนตำแหน่งการติดตั้งได้อีกด้วย โดยในลายวงจรของบอร์ดสวิตซ์ไมโครคอนโทรลเลอร์นั้นจะประกอบไปด้วยวงจรเรียงกระแสเต็มคลื่นแบบบริดจ์ จากนั้นจะใช้ไอซีเรคกูเลเตอร์ในการควบคุมระดับแรงดันและจ่ายกระแสให้คงที่ดังรูปที่ 3.10



รูปที่ 3.10 วงจรโดยรวมของสวิตซ์ไมโครคอนโทรลเลอร์



รูปที่ 3.11 ลายวงจรบอร์ดสวิตซ์ไมโครคอนโทรลเลอร์

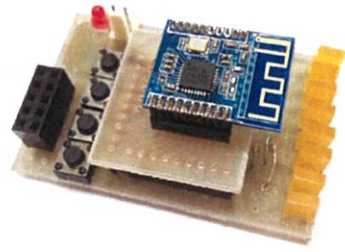


รูปที่ 3.12 วงจรบอร์ดสวิตซ์ไมโครคอนโทรลเลอร์ที่ใช้งาน

3.3.2 ส่วนการสื่อสารผ่านระบบอินเทอร์เน็ต

3.3.2.1 มาสเตอร์ไมโครคอนโทรลเลอร์ (nRF24le1)

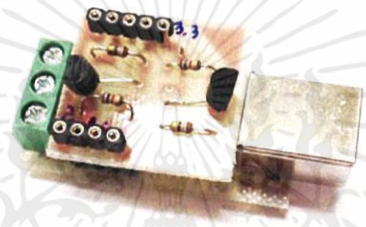
บอร์ดมาสเตอร์ไมโครคอนโทรลเลอร์ก็ได้ใช้ไมโครคอนโทรลเลอร์ nRF24LE1 เหมือนกับในส่วนของการสั่งเปิด-ปิดหลอดไฟ แต่ไมโครคอนโทรลเลอร์ในบอร์ดนี้จะทำหน้าที่เป็นตัวมาสเตอร์เพื่อรับค่าจากไมโครคอนโทรลเลอร์ตัวอื่นๆ แล้วทำการส่งข้อมูลต่อไปยังบอร์ดเชื่อมต่อ UART



รูปที่ 3.13 มาสเตอร์ ไมโครคอนโทรลเลอร์ nRF24LE1

3.3.2.2 **บอร์ดเชื่อมต่อ UART**

อุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสนั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารแบบอนุกรม



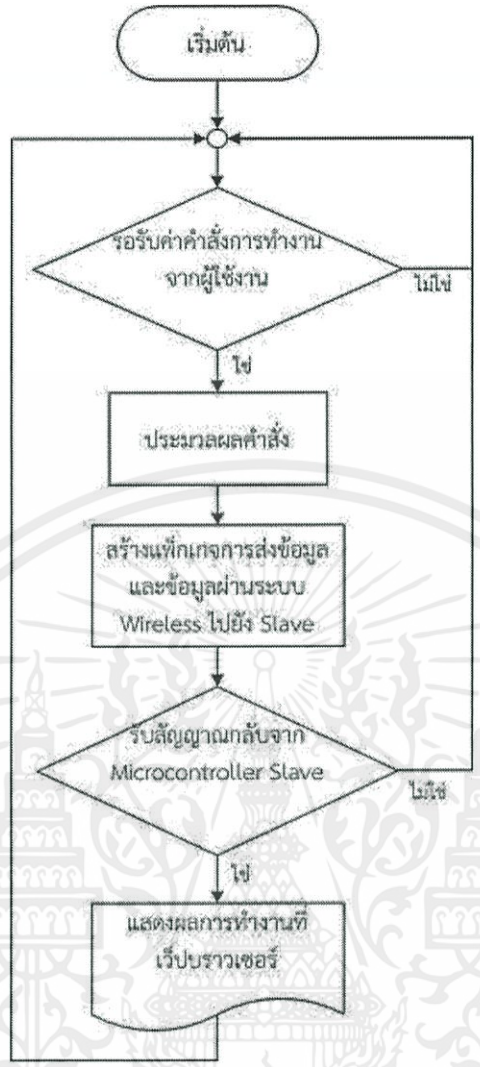
รูปที่ 3.14 บอร์ด UART

3.4 การออกแบบการทำงานของไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์จะแบ่งการทำงานออกเป็น 2 ส่วน คือ มาสเตอร์ และ เสลฟ ส่วนของมาสเตอร์ไมโครคอนโทรลเลอร์ จะเชื่อมต่ออยู่กับคอมพิวเตอร์ที่ติดต่อกับเซิร์พเวอร์อินเตอร์เน็ต และส่วนของเสลฟไมโครคอนโทรลเลอร์ นั้นจะอยู่ในกล่องอุปกรณ์ส่งสัญญาณอินฟราเรด ควบคุมเครื่องปรับอากาศ

3.4.1 การทำงานของมาสเตอร์ไมโครคอนโทรลเลอร์

มาสเตอร์ไมโครคอนโทรลเลอร์ เปรียบเสมือนอุปกรณ์สื่อกลางในการสื่อสารระหว่างผู้ใช้งานและอุปกรณ์โดยมีขั้นตอนการทำงานดังรูปที่ 3.14



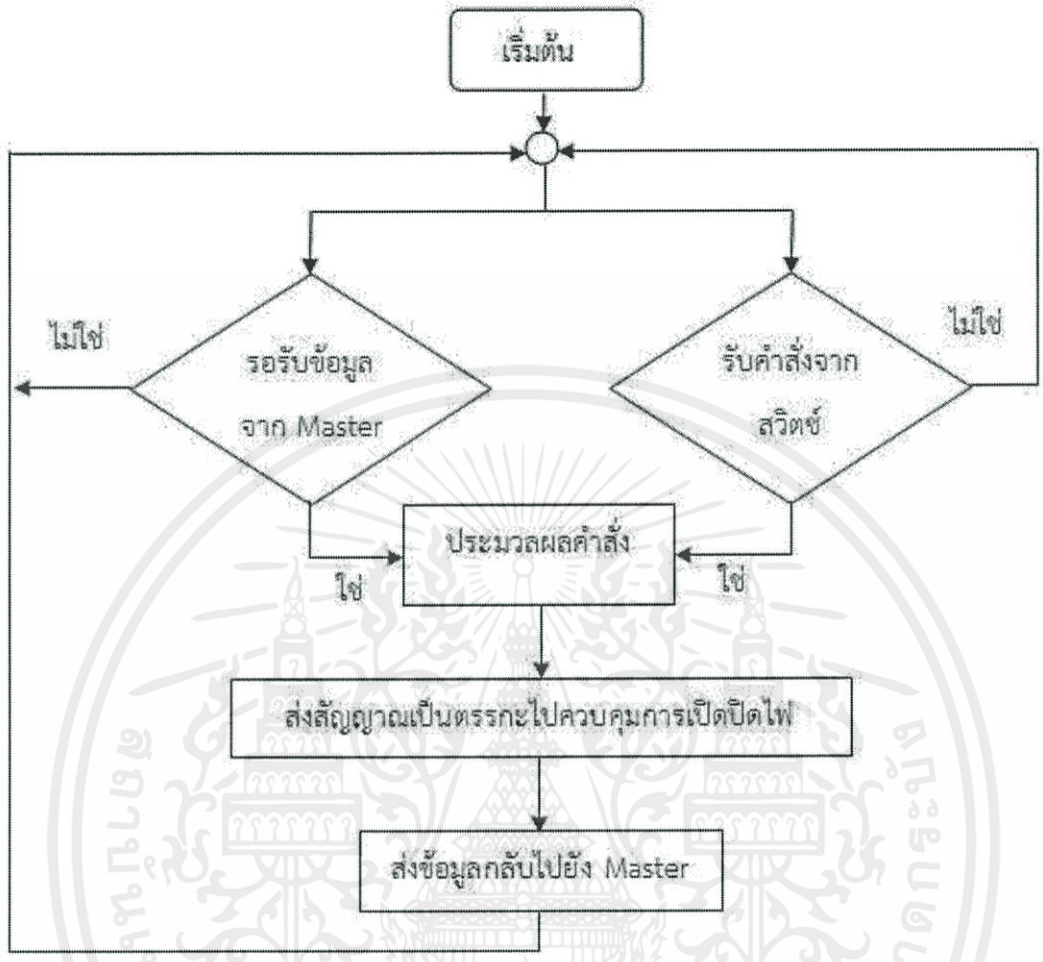
รูปที่ 3.15 ขั้นตอนการทำงานของมาสเตอร์ไมโครคอนโทรลเลอร์

มาสเตอร์ไมโครคอนโทรลเลอร์เชื่อมต่ออยู่กับคอมพิวเตอร์ด้วยพอร์ตอนุกรม ผ่านทางบอร์ด Uart เมื่อมีการสั่งงานมาจากเว็บบราวเซอร์ ข้อมูลจะถูกส่งมายังไมโครคอนโทรลเลอร์จะมีการประมวลผลคำสั่ง พร้อมกับการสร้างแพ็กเกจข้อมูลเพื่อส่งข้อมูลนั้นไปยังเสลฟไมโครคอนโทรลเลอร์ โดยส่งผ่านระบบไร้สายจากนั้นเสลฟไมโครคอนโทรลเลอร์จะส่งค่ากลับมายังมาสเตอร์ไมโครคอนโทรลเลอร์ เพื่อนำไปแสดงสภาวะทำงานของอูปกรณ์ส่องสว่างบนหน้าเว็บบราวเซอร์

3.4.2 การทำงานของเสลฟไมโครคอนโทรลเลอร์

เสลฟไมโครคอนโทรลเลอร์ จะเชื่อมต่อกับวงจรเพื่อส่งสัญญาณไปควบคุมหลอดไฟ โดยรับคำสั่งมาจากมาสเตอร์ไมโครคอนโทรลเลอร์ในรูปของแพ็กเกจข้อมูล จากนั้นหน่อยประมวลกลางจะทำการประมวลผล และถอดคำสั่งการทำงานจากนั้นจะส่งสัญญาณไปสั่งงานรีเลย์เพื่อเปิด- ปิดหลอดไฟ ตามคำสั่งของผู้ใช้งานพร้อมกับการส่งข้อมูลกลับไปยังมาสเตอร์ไมโครคอนโทรลเลอร์ เพื่อ

แสดงสถานะของคำสั่งที่ได้สั่งงานไปแล้ว เพื่อส่งคำสั่งนั้นไปแสดงสถานะการทำงานที่หน้าเว็บบราวเซอร์ได้เช่นเดียวกัน



รูปที่ 3.16 ขั้นตอนการทำงานของเสลฟ์ไมโครคอนโทรลเลอร์

3.5 การออกแบบและการทำงานของโปรแกรม

การออกแบบหน้าเว็บ (Application) และการออกแบบโปรแกรมสำหรับรับข้อมูลเว็บไซต์ (Application) เป็นสื่อที่ได้รับความนิยมอย่างมากบนอินเตอร์เน็ต ซึ่งเว็บไซต์หรือแอพพลิเคชั่นเป็นสื่อที่อยู่ในความควบคุมของผู้ใช้โดยสมบูรณ์ กล่าวคือ ผู้ใช้สามารถตัดสินใจเลือกได้ว่าจะดูเว็บไซต์ใดไม่ดูเว็บไซต์ใด โดยเว็บไซต์ที่ได้รับการออกแบบอย่างสวยงาม มีการใช้งานที่สะดวก ย่อมได้รับความสนใจจากผู้ใช้มาก ดังนั้น การออกแบบเว็บไซต์จึงเป็นกระบวนการที่สำคัญในการสร้างเว็บไซต์ ให้ประทับใจแก่ผู้ใช้งานและมีประโยชน์อีกด้วย

การออกแบบเว็บไซต์ที่มีประสิทธิภาพนั้นต้องคำนึงถึง องค์ประกอบสำคัญดังต่อไปนี้

- ความเรียบง่าย (Simplicity)
- ความสม่ำเสมอ (Consistency)
- ความเป็นเอกลักษณ์ (Identity)

- คุณภาพของสิ่งที่ปรากฏให้เห็นในเว็บไซต์ (Visual Appeal)
- ความสะดวกของการใช้ในสภาพต่าง ๆ (Compatibility)

3.5.1 การออกแบบหน้าเว็บเพจ

การออกแบบเว็บไซค์/แอพพลิเคชั่น ในการแสดงผลของโครงงานนี้มีดังนี้

โดยภาพรวมทั้งหมดจะประกอบไปด้วย

- สัญลักษณ์ของสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
- ชื่อโครงงานโดยเป็นการออกแบบรวมกันของสมาชิกภายในกลุ่ม
- กราฟฟิคแสดงสถานะของหลอกไฟ
- วัน / เดือน / ปี
- ตารางแสดงผลของข้อมูลที่วัดได้

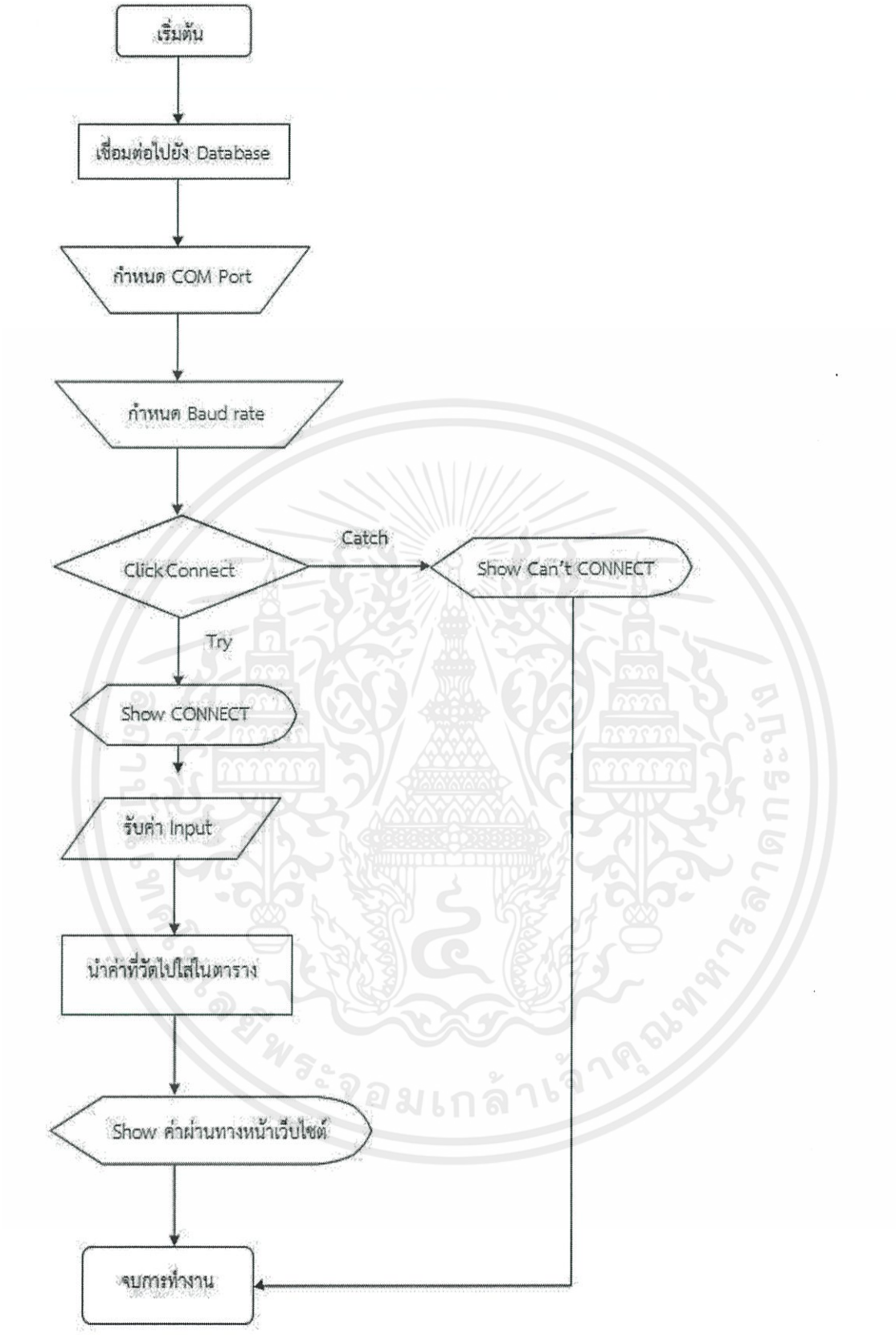
โดยพื้นฐานในการออกแบบหน้าจอในการแสดงผลหรือแอพพลิเคชั่นในครั้งนี้ เป็นการปรึกษาและรวบรวมความคิดเห็นจากสมาชิกภายในกลุ่ม โดยเป็นการลงความเห็นว่าควรจะออกแบบหน้าจอแสดงผลให้ออกมา ง่ายต่อการเข้าใจของผู้ใช้งาน ,ง่ายต่อการอ่านค่าที่แสดงผลออกมา , สามารถตรวจสอบผลของการวัดย้อนหลังได้ , สามารถรับรู้สถานะของไฟที่ทำงานได้ ซึ่งจากที่กล่าวมาทั้งหมดนี้ เป็นจุดประสงค์หลักของงาน ส่วนจุดประสงค์รองลงมา คุณจะสามารถรับรู้ถึง ความสวยงามของหน้าเว็บเพจ ,ความทันสมัยและอื่น ๆ อีกมากมาย

โดยพื้นที่ ที่ไว้จัดเก็บหน้าเว็บไซต์นี้ อยู่ภายในหน่วยความจำของคอมพิวเตอร์ โดยได้ทำการจำลองว่า คอมพิวเตอร์เป็นตัวเซิร์ฟเวอร์จัดเก็บทั้งหน้าเว็บไซต์และข้อมูลที่วัดค่า ซึ่งผลที่ได้ออกมาจะปรากฏดังรูปที่ 3.16

รูปที่ 3.17 แสดงภาพรวมของหน้าเว็บไซต์และข้อมูลที่วัดค่าออกมา

3.5.2 การออกแบบโปรแกรม

โปรแกรมจะทำหน้าที่ในการรับค่าจากตัวไมโครคอนโทรลเลอร์ที่อยู่ในบอร์ด โดยเป็นการเชื่อมต่อโดยตรงผ่าน Port USB(COM Port 9) และ อัตราความเร็วในการรับส่งข้อมูล (Baudrate) เท่ากับ 38400 โดยค่าที่ส่งมามีลักษณะเป็นเฮ็กซ์ไฟล์ นำค่าที่ส่งมาเชื่อมต่อไปยังตารางภายในฐานข้อมูลที่ถูกสร้างเพื่อรองรับค่าที่ถูกส่งเข้ามา โดยหน้าเว็บไซต์ได้ทำการเขียนโปรแกรมให้แสดงข้อมูลที่อยู่ภายในตารางฐานข้อมูล เรียบร้อย



รูปที่ 3.18 แผนภาพขั้นตอนการทำงานของโปรแกรม

จากรูปแสดงถึงแผนภาพของขั้นตอนการทำงานของโปรแกรมเพื่อการรับค่าและส่งค่าไปแสดงผลหน้าเว็บไซต์

บทที่ 4
การทดสอบการทำงาน

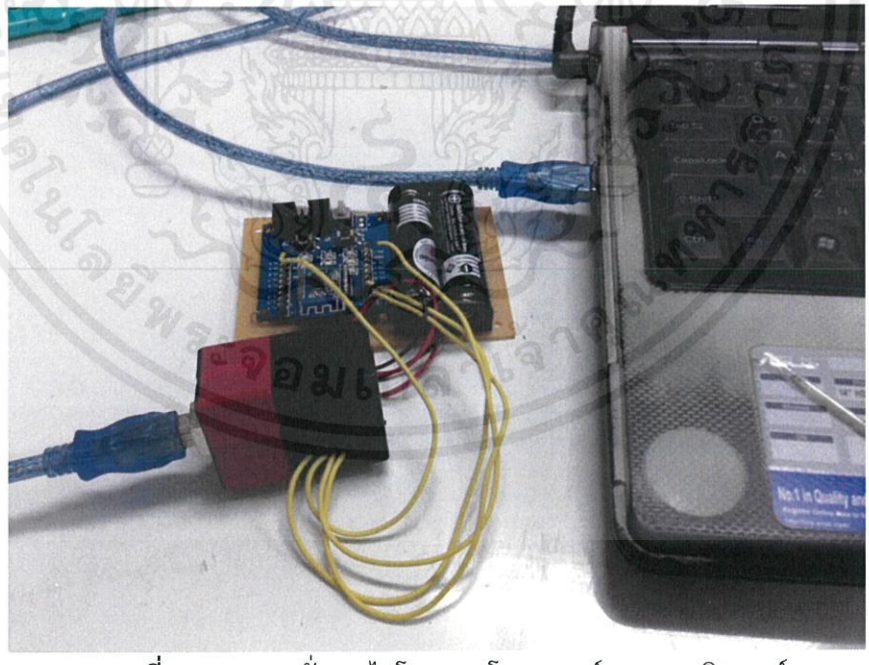
4.1 กล่าวนำ

การทดลอง ซึ่งทดสอบการควบคุมหลอดไฟผ่านไมโครคอนโทรลเลอร์ในลักษณะต่าง ๆ ตามขอบเขตของโครงงานที่ได้กำหนดไว้ซึ่งก่อนทำการทดลองจะต้องตรวจเช็คบอร์ดวงจร ทุก ๆ บอร์ดเสียก่อน เพื่อให้สามารถทดลองและได้ผลการทดลองตามที่คาดคิดไว้

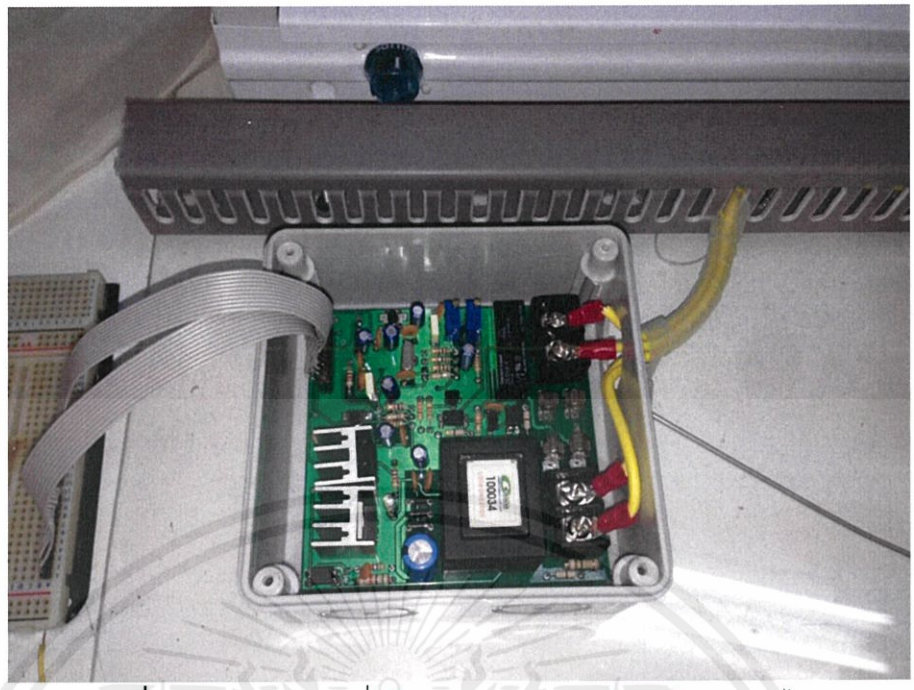
4.2 การทดลองการควบคุมหลอดไฟผ่านการสั่งงานจากคอมพิวเตอร์

ทำการทดลองการควบคุมหลอดไฟโดยการสั่งงานจากคอมพิวเตอร์ อุปกรณ์จะทำการส่งสัญญาณไปควบคุมหลอดไฟตามที่ต้องการ โดยการสั่งงานควบคุมการเปิด - ปิด ของหลอดไฟ

โดยการสั่งงานจะเริ่มจากผู้ใช้งาน จากนั้นคำสั่งจะถูกส่งมายังไมโครคอนโทรลเลอร์ที่เป็นตัวมาสเตอร์ ซึ่งเชื่อมตัวผ่านช่องทางของ USB ของคอมพิวเตอร์ไปยังไมโครคอนโทรลเลอร์ โดยใช้ฟังก์ชั่นคำสั่ง UARTในการรับคำสั่งจากคอมพิวเตอร์ จากนั้นจะส่งคำสั่งต่อไปยังตัวเสลฟ โดยผ่านการส่งข้อมูลด้วยระบบไร้สาย จากนั้นไมโครคอนโทรลเลอร์ที่เป็นตัวเสลฟจะทำการประมวลผลและส่งสัญญาณไปสั่งงานรีเลย์และเปิด - ปิดหลอดไฟตามคำสั่งที่ผู้ใช้งานต้องการ

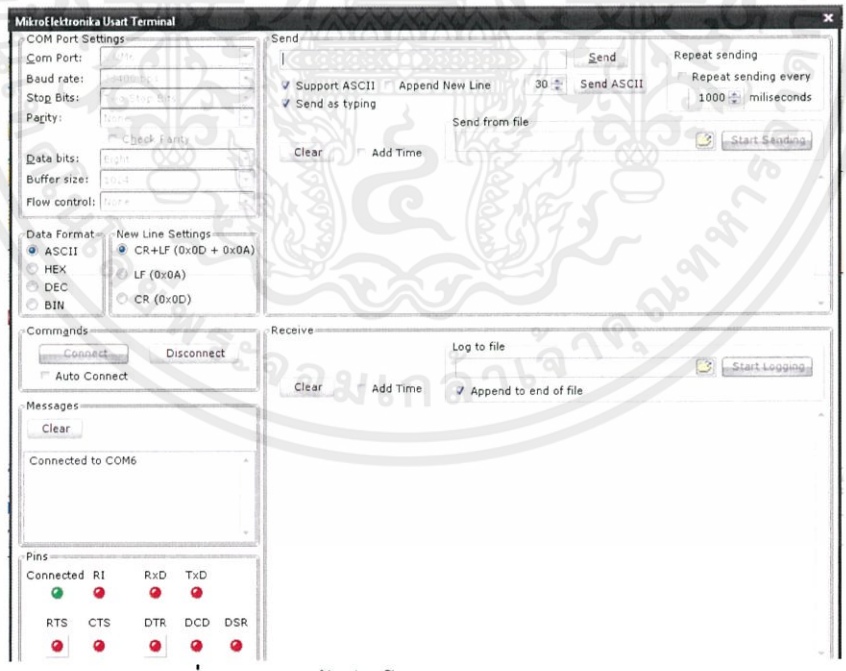


รูปที่ 4.1.1 การสั่งงานไมโครคอนโทรลเลอร์จากคอมพิวเตอร์



รูปที่ 4.1.2 บอร์ดวงจรที่ใช้ในการควบคุมการเปิด – ปิด หลอดไฟ

โดยการสั่งงานผ่านคอมพิวเตอร์โดยตรงนั้นจะต้องอาศัยการทำงานของโปรแกรมที่เรียกว่า Hyper terminal ซึ่งก่อนที่จะสั่งงานผ่าน Hyper Terminal ได้นั้นจะต้องทำการเชื่อมต่อกันก่อนจึงจะสามารถทำงานร่วมกันได้

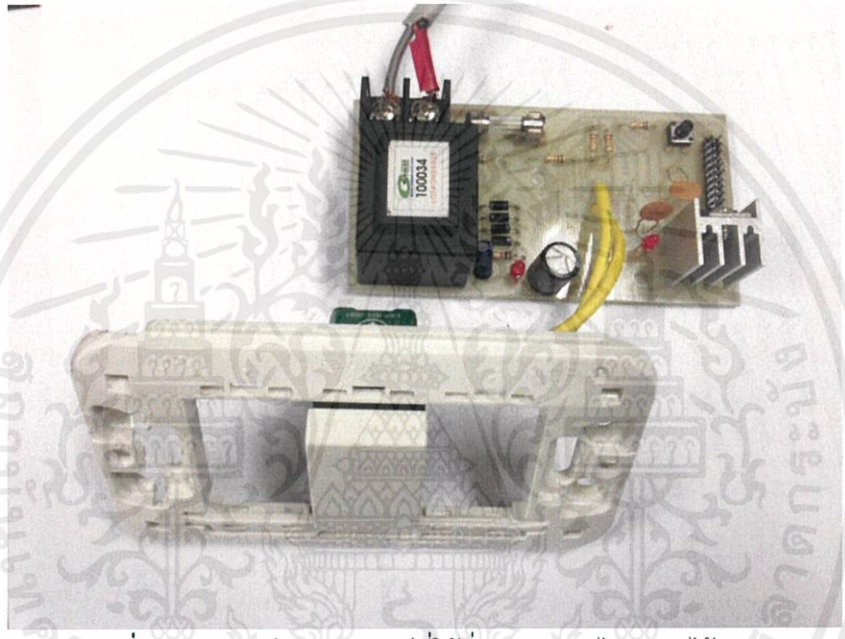


รูปที่ 4.1.3 หน้าต่างโปรแกรม Hyper Terminal

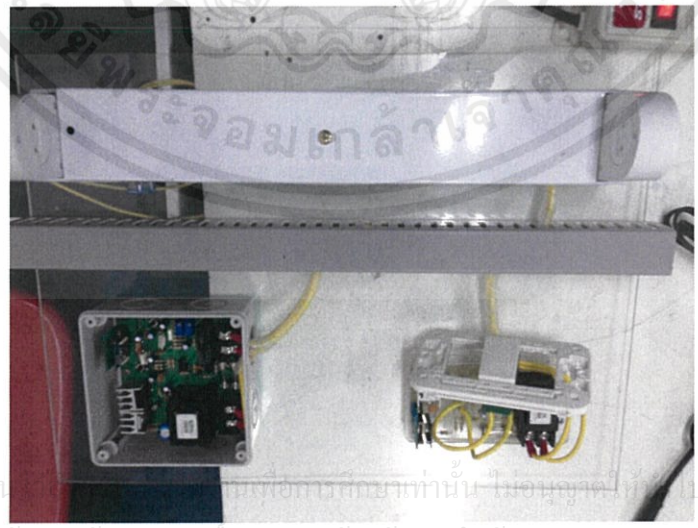
4.3 การทดลองการควบคุมหลอดไฟผ่านการสั่งงานบอร์ดสวิตช์

ทำการทดลองการควบคุมหลอดไฟโดยการสั่งงานจากบอร์ดสวิตช์ อุปกรณ์จะทำการส่งสัญญาณไปควบคุมหลอดไฟตามที่ต้องการ โดยการสั่งงานควบคุมการเปิด - ปิด ของหลอดไฟ

ในการทดลองนี้ อุปกรณ์ที่ใช้สั่งงานหลักคือตัวบอร์ดสวิตช์ โดยบอร์ดสวิตช์จะมีไมโครคอนโทรลเลอร์ติดอยู่ ซึ่งจะคอยรอรับสถานะจากสวิตช์เมื่อทำการเปิด – ปิด ที่สวิตช์ เมื่อมีกดปุ่มที่บอร์ดสวิตช์นั้นจะมีการส่งสัญญาณไปที่ไมโครคอนโทรลเลอร์ตามที่ได้เขียนโปรแกรมลงในตัวไมโครคอนโทรลเลอร์ จากนั้นไมโครคอนโทรลเลอร์จะส่งสัญญาณผ่านเทคโนโลยีแบบไร้สาย เพื่อไปสั่งงานไมโครคอนโทรลเลอร์อีกตัวหนึ่งที่ติดตั้งไว้กับโมดูลที่เชื่อมต่อกับหลอดไฟ เพื่อทำการประมวลผลและส่งสัญญาณไปสั่งงานรีเลย์และเปิด – ปิดหลอดไฟตามคำสั่งที่ผู้ใช้งานต้องการ



รูปที่ 4.2.1 บอร์ดวงจรสวิตช์ที่ใช้สั่งการหลอดไฟ แบบไร้สาย



รูปที่ 4.2.2 อุปกรณ์ที่รับคำสั่งจากคอมพิวเตอร์และเชื่อมต่อกะหลอดไฟ

4.4 การทดลองการควบคุมหลอดไฟผ่านการสั่งงานจากเว็บเบราว์เซอร์

ทำการทดลองการควบคุมหลอดไฟโดยการสั่งงานจากเว็บเบราว์เซอร์ อุปกรณ์จะทำการส่งสัญญาณไปควบคุมหลอดไฟตามที่ต้องการ โดยการสั่งงานควบคุมการเปิด - ปิด ของหลอดไฟ

ในการทดลองนี้จะมีหลักการคล้ายกับการทดลองการควบคุมหลอดไฟผ่านการสั่งงานจากคอมพิวเตอร์ คือ การสั่งงานจะเริ่มจากผู้ใช้งาน จากนั้นคำสั่งจะถูกส่งมายังไมโครคอนโทรลเลอร์ที่เป็นตัวมาสเตอร์ ซึ่งเชื่อมตัวผ่านช่องทางของ USB ของคอมพิวเตอร์ไปยังไมโครคอนโทรลเลอร์ โดยใช้ฟังก์ชั่นคำสั่ง UARTในการรับคำสั่งจากคอมพิวเตอร์ จากนั้นจะส่งคำสั่งต่อไปยังตัวเสลฟ โดยผ่านการส่งข้อมูลด้วยระบบไร้สาย โดยการสั่งงานนั้นจะเป็นดารสั่งงานจากเว็บเบราว์เซอร์แทน โดยชุดข้อมูลคำสั่งที่สั่งลงมาจากเว็บเบราว์เซอร์นั้นจะถูกเขียนโดยใช้ภาษาซีชาร์ป (C#) และใช้โปรแกรม Dreamweaver มาในการสร้างตัวเว็บไซต์นี้

รูปที่ 4.3 แสดงภาพรวมของหน้าเว็บไซต์และข้อมูลที่วัดค่าออกมา

บทที่ 5
สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

ปริญญานิพนธ์ฉบับนี้ได้นำเสนอแนวคิดในการจัดการและการควบคุมการเปิด - ปิดหลอดไฟแบบไร้สายภายในบ้าน โดยใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมกระบวนการทั้งหมดและใช้งานร่วมกับเทคโนโลยีแบบไร้สาย ให้มีการรับส่งข้อมูล โดยการรับส่งข้อมูลนั้นเป็นรูปแบบมาสเตอร์-สเลฟ อีกทั้งยังสามารถดูค่าสถานการณ์ทำงาน ควบคุมการทำงานผ่านหน้าเว็บบราวเซอร์ได้

5.2 ปัญหาและอุปสรรค

ไมโครคอนโทรลเลอร์ที่ใช้ทำงานของระบบควบคุมนี้คือ nRF24le1 ซึ่งไมโครคอนโทรลเลอร์ตัวนี้ได้จัดอยู่ในตระกูล C-51 และตัวที่ได้เลือกมานั้นมีขนาดเล็ก และตัวอุปกรณ์มีตัวรับสัญญาณในการรับ-ส่งไวร์เลสที่ค่อนข้างเล็ก จึงทำให้เกิดปัญหาและอุปสรรคต่างๆดังนี้

1) เสาสัญญาณนั้นได้มีการออกแบบมาตั้งแต่แรกแล้ว ซึ่งได้มีขนาดเล็ก จึงทำให้เกิดปัญหาและอุปสรรคในการรับ – ส่งสัญญาณ ทำให้มีระยะในการรับ – ส่งสัญญาณที่ค่อนข้างใกล้ประมาณ 3 -5 เมตร จึงทำให้เกิดข้อจำกัดในด้านการรับ– ส่งข้อมูลในระยะไกล

2) ในด้านการประมวลผลของข้อมูล ซึ่งไมโครคอนโทรลเลอร์ที่ได้ทำการเลือกมาใช้นี้ เป็นเบอร์ที่มีขนาดเล็ก และการประมวลผลของซีพียูที่ช้า จึงทำให้เกิดการประมวลผลที่ช้าตามไปด้วย และส่งผลในการรับ – ส่งข้อมูลที่ช้าลงอีกด้วย ทั้งสองส่วนนี้จึงทำให้เกิดความล่าช้าในการรับ – ส่งข้อมูลทั้งในตัวอุปกรณ์ไมโครคอนโทรลเลอร์ด้วยกัน และยังส่งผลไปถึงการรับ – ส่งข้อมูลขึ้นไปยังเซิร์ฟเวอร์อีกด้วย

5.3 ข้อเสนอแนะ

1) การรับส่งข้อมูลระหว่างตัวอุปกรณ์มาสเตอร์กับอุปกรณ์เสลฟ ไม่สามารถรับส่งได้ไกลเท่าที่ควร ควรจัดการเรื่องเสาสัญญาณไวร์เลสให้มีการรับ – ส่งที่ไกลขึ้น เพื่อความสามารถในการทำงานที่มีประสิทธิภาพมากกว่าเดิม

2) ควรจะทำการเปลี่ยนขนาดเบอร์ของไมโครคอนโทรลเลอร์ให้มีขนาดใหญ่ขึ้น เพื่อให้มีการประมวลผลที่เร็วขึ้น พร้อมทั้งมีพอร์ตให้ใช้ได้มากขึ้น เพื่อความสะดวกและมีความยืดหยุ่นในการทำงานที่มากขึ้น

บรรณานุกรม

[1] สุธี พงศาสกุลชัย.**คัมภีร์ Visual C# 2005.** กรุงเทพ : เคทีพี คอมพ์ แอนด์ คอนซัลท์, 2549.

[2] สุรพรรษ์ เพ็ญจำรัส.**เรียนลัด C# และการเขียนโปรแกรม.NET.** กรุงเทพ : โปรวิชั่น, 2546.

[3] ธีรวัฒน์ ประกอบผล.**คู่มือการเขียนโปรแกรม ภาษา C.** กรุงเทพ : ซิมพลิฟาย, 2553.

[4] ศุภชัย สมพานิช.**คู่มือการเขียนโปรแกรม Visual C#.NET.** นนทบุรี : อินโฟเพรส, 2546.

[5] ประจิน พลังสันติกุล, และชัยวัฒน์ ลิ้มพรจิตรวิไล. **ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 กับ Keil C51 คอมไพเลอร์.** กรุงเทพฯ : อินโนเวตีฟ เอ็กเพอริเมนต์, 2521

[6] **การใช้โปรแกรม Keil uVision 3 เบื้องต้น.** 10 กุมภาพันธ์ 2556, จาก http://www.mcuthailand.com/articles/mcs/Keil.html

[7] General-purpose Latching Relay MYK, 2002 : 1-4

[8] ACS712 Fully Integrated, Hall Effect-Based Linear Current Sensor with 2.1 kVRMS

[9] ADE7763, 2004 : 13, 48-55

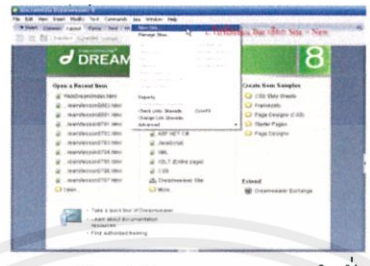
[10] nRF24LE1, 2010 : 1-196

ภาคผนวก

ขั้นตอนการสร้างไซต์

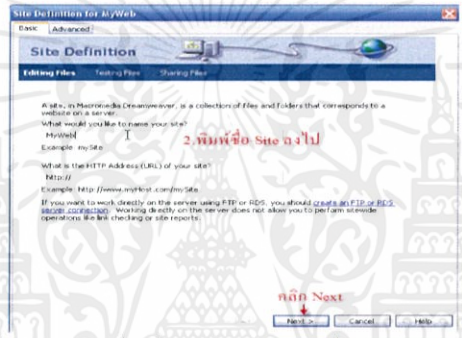
การสร้างไซต์ ก็เพื่อที่เวลาสร้างเว็บไซต์ ข้อมูลจะได้ถูกจัดอย่างเป็นระเบียบ ไม่กระจัดกระจาย ง่ายต่อการตรวจสอบ โดยมีขั้นตอนดังนี้

1. Click ที่ Tool Bar



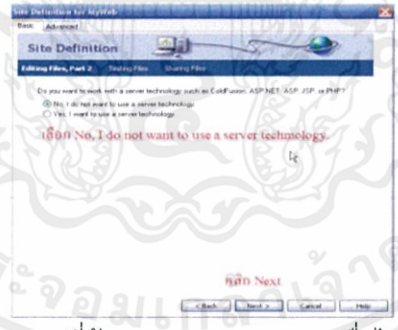
รูปที่ 1 กดแทบ Tool Bar และจะพบคำสั่ง New Site

2. พิมพ์ชื่อ Site ที่ต้องการ โดยเป็นชื่อของ file เว็บไซต์



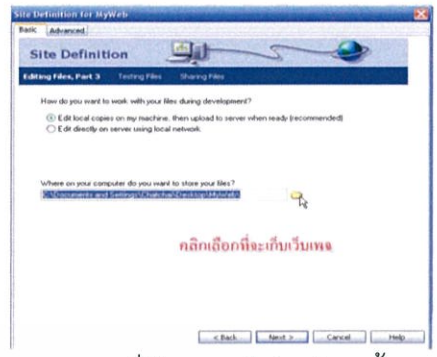
รูปที่ 2 กรอกชื่อที่ต้องการให้เป็นชื่อของ Site งาน

3. เลือกช่องที่ต้องการ“ต้องการใช้ Technology รวมกับพวก ASP NET,ASP,PHP”



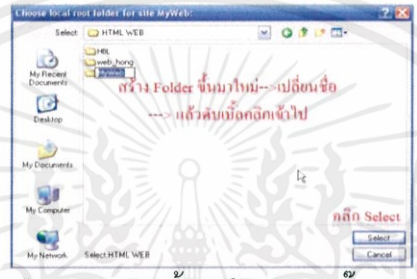
รูปที่ 3 เลือกช่องทางที่ต้องการและกด Next เพื่อไปยังขั้นตอนต่อไป

4. เลือก Folder ที่ต้องการเก็บ File ที่ได้สร้างขึ้น



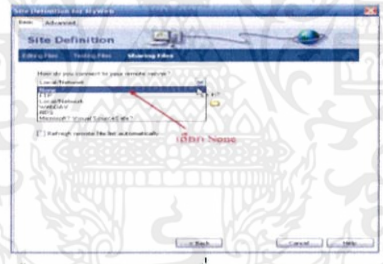
รูปที่ 4 สร้างหรือเลือก Folder ที่ต้องการเก็บไฟล์งานทั้งหมดและทำขั้นตอนต่อไป

5. สร้าง Folder ใหม่เพื่อความสะดวกและลดความสับสน



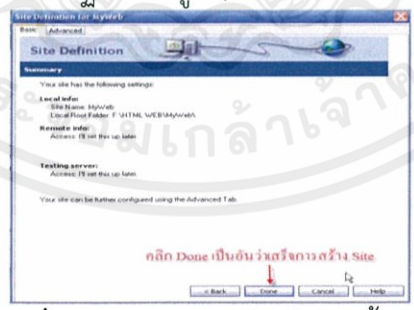
รูปที่ 5 กรณีสร้างหรือเลือก Folder ขึ้นมาใหม่และคลิ๊ก Select เพื่อดำเนินการต่อ

6. หัวข้อ “ ต้องการเชื่อมต่อกับ Remote Server ” ตอบ None ก่อน



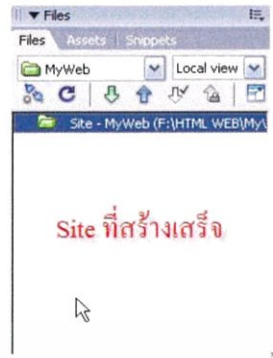
รูปที่ 6 ตัวเลือกสำหรับการเชื่อมต่อกับ Remote Server

7. เมื่อตั้งค่าเรียบร้อยจะปรากฏภาพดังรูป (แสดงผลตามที่ตั้งค่า)



รูปที่ 7 แสดงรายละเอียดในการตั้งค่า

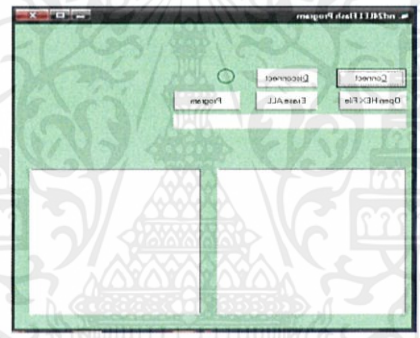
8. เมื่อติดตั้งโปรแกรมเสร็จสมบูรณ์ จะปรากฏดังรูปภาพ



รูปที่ 8 โฟร์เดอร์ (Folder) จัดเก็บหน้าเว็ปไซต์ที่ได้สร้างไว้

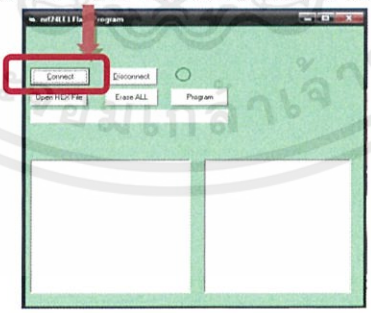
การดาวน์โหลด Hex File ให้กับ MCU

โปรแกรม nRF24LE1 เป็นโปรแกรมที่ใช้ Download ข้อมูลแบบ Hex File ให้กับ CPU MCS-51 โดยใช้วิธีการแบบ Serial Programming ซึ่งวิธีการใช้งานโปรแกรมนั้นมีขั้นตอนที่ไม่ยากซับซ้อนมากนัก สามารถทำความเข้าใจได้ง่าย โดยในที่นี้จะขอแนะนำให้ทราบถึงวิธีการใช้โปรแกรม nRF24LE1 Flash Program เพื่อใช้งานกบบอร์ด nRF24LE1



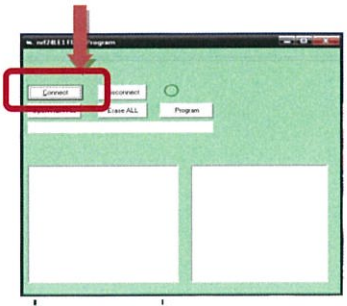
รูปที่ 1 โปรแกรม nRF24LE1 Flash Program

1. เมื่อทำการเชื่อมต่อบอร์ด nRF24LE1 เข้ากับเครื่องคอมพิวเตอร์ จากนั้นเมื่อเปิดโปรแกรม nRF24LE1 เพื่อดาวน์โหลดโปรแกรมให้กับบอร์ด nRF24LE1 จำเป็นที่จะต้องเชื่อมต่อโปรแกรมเข้าบอร์ด nRF24LE1 ทุกครั้ง โดยการกดที่ปุ่ม Connect ที่โปรแกรม nRF24lE1



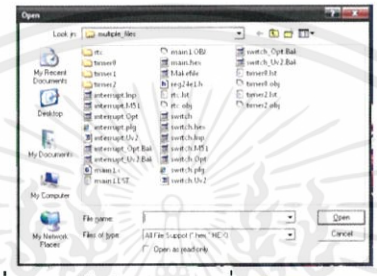
รูปที่ 2 การเชื่อมต่อโปรแกรม nRF24LE1 Flash Program เข้ากับบอร์ด

2. เลือก HEX File ที่ต้องการดาวน์โหลดลงในบอร์ด nRF24LE1 โดยไปที่คำสั่ง Open HEX File



รูปที่ 3 ปุ่มคำสั่ง Open HEX File

3. เมื่อกดปุ่ม Open HEX File แล้ว จะขึ้นหน้าต่างให้เลือกที่อยู่ที่ใช้บันทึก HEX File ไว้ จากนั้นเลือก HEX File ที่ต้องการ



รูปที่ 4 หน้าต่างแสดง ที่อยู่ของ HEX File

4. เมื่อเลือก HEX File ที่ต้องการดาวน์โหลดลงในบอร์ดแล้ว จากนั้นสามารถดาวน์โหลด HEX File ลงในบอร์ด โดยกดที่ปุ่ม Program

รูปที่ 5 คำสั่งโปรแกรมเพื่อใช้ในการดาวน์โหลดโปรแกรม

5. โปรแกรม จะแสดงว่า ดาวน์โหลด HEX File ลงในบอร์ดเรียบร้อยแล้ว

รูปที่ 6 ดาวน์โหลด HEX File ลงในบอร์ด

ซึ่งหลังจากการโปรแกรมเสร็จเรียบร้อยแล้ว CPU จะเริ่มต้นทำงานตามข้อมูลในโปรแกรมที่ส่ง Download ให้ทันที

โปรแกรมมาสเตอร์ไมโครคอนโทรลเลอร์

```
#include<reg24le1.H>
#include<stdio.H>
#include "main.h"
#include "nrf_lab.h"
#include "nRF_definmm.h"

#define  nRF_CE    RFCE
#define  nRF_CS    RFCSN
#define  SSP1BUF   SPIRDAT

#define  Address_Route  0xEE

unsigned char LOOP, inputdata, Check, Reset_Loop ,Test ,Checksum_Tx = 0;
unsigned int Datanet,p;
unsigned char temp,Keep;
unsigned int Rx_Flag ,Uart_flag , Decode_in, Decode_out = 0x0000, Send_Decode = 0,x;


void Delay_1us(volatile unsigned int n){
   unsigned char i;
   while(n--)
    for(i=0;i<3;i++);
 }
 void Delay_100us(volatile unsigned int n){
   unsigned char i;
   while(n--)
    for(i=0;i<33;i++);
 }

 void Delay_ms(int i)
{
    int j;
    for(; i>0; i--)
    for(j=0; j<100; j++);
}

void UART_Mode() interrupt 4 using 3
{
    if(RIO)
    {

        Keep = (unsigned int)SOBUF;
        Datanet = Keep;
        RIO = 0;
        LOOP = 1;
    }
    else if(TIO)
    {
        TIO=0;
    }
}
void CE_Pin_Active(unsigned char action)    // CE pin high, low or pulse..
{
  switch(action)
  {
    case CE_LOW:                    // action == 0, CE low
      nRF_CE = 0;
    break;

    case CE_HIGH:                   // action == 1, CE high
      nRF_CE = 1;
    break;

    case CE_PULSE:                  // action == 2, CE pulse (10?s)
     // TH02  = 0;
      nRF_CE = 1;                   // Set CE pin high
      Delay_1us(100);
      nRF_CE = 0;
     //TH02ON_bit = 1;                // Start Timer2, CE pulse timer
    break;
  }
}
unsigned char LO1_Clear_IRQ(unsigned char irq_flag)
{
  return nRF_SPI_RW_Reg(WRITE_REG + STATUS, irq_flag);
}

unsigned char nRF_SPI_RW(unsigned char byte1)
{
  unsigned char status;
  //Delay_1us(10);
  SSP1BUF = byte1;
  Delay_1us(1);          // Delay 300s ok
  status  = SSP1BUF;
  return(status);
}
unsigned char nRF_SPI_RW_Reg(unsigned char reg, unsigned char value)
{
  unsigned char status;

  nRF_CS = 0;                                   // CSN low, init SPI transaction
  status = nRF_SPI_RW(reg);                     // select register
  nRF_SPI_RW(value);                            // ..and write value to it..
  nRF_CS = 1;                                   // CSN high again

  return(status);                               // return nRF24L01 status byte
}
unsigned char nRF_SPI_Write_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
  unsigned char status, byte_ctr;

  nRF_CS = 0;                                   // Set CSN low, init SPI transaction
  status = nRF_SPI_RW(reg);                     // Select register to write to and read status byte

  for(byte_ctr=0; byte_ctr<bytes; byte_ctr++)   // then write all byte in buffer(*pBuf)
  {
    nRF_SPI_RW(*pBuf++);
  }


  nRF_CS = 1;                                   // Set CSN high again

  return(status);                               // return nRF24L01 status byte
}

unsigned char LO1_RD_RX_PW_n(unsigned char pipe)
{
  return nRF_SPI_Read(RX_PW_P0 + pipe);
}
void LO1_Write_TX_Pload(unsigned char *pBuf, unsigned char plWidth)
{
  nRF_SPI_Write_Buf(WR_TX_PLOAD, pBuf, plWidth);
}

unsigned char Send_Packet()                     // Send one data packet, controlled by Button1(SW1)
{
  LO1_Write_TX_Pload(tTX_pload, TX_PLOAD_WIDTH);  // Write new TX payload
  CE_Pin_Active(CE_PULSE);                          // Eneble CE pulse, 12?s
  //ucTry_Ctr = 0;                                // Reset Try_Ctr before nex transmitt..
  return 1;
}
```

```
void LO1_Flush_RX(void)
{
  nRF_SPI_RW_Reg(FLUSH_RX,0);
}
void LO1_Flush_TX(void)
{
  nRF_SPI_RW_Reg(FLUSH_TX,0);
}
unsigned char LO1_Get_FIFO(void)
{
  return nRF_SPI_Read(FIFO_STATUS);
}
unsigned char nRF_SPI_Read_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
  unsigned char status, byte_ctr;

  nRF_CS = 0;                                   // Set CSN low, init SPI tranaction
  status = nRF_SPI_RW(reg);                     // Select register to write to and read status byte

  for(byte_ctr=0;byte_ctr<bytes;byte_ctr++)     // Perform SPI_RW to read byte from nRF24LO1
  {
    pBuf[byte_ctr] = nRF_SPI_RW(0);
  }

  nRF_CS = 1;                                   // Set CSN high again

  return(status);                               // return nRF24LO1 status byte
}
unsigned char nRF_SPI_Read(unsigned char reg)
{
  unsigned char reg_val;

  nRF_CS = 0;                                   // CSN low, initialize SPI communication...
  nRF_SPI_RW(reg);                              // Select register to read from..
  reg_val = nRF_SPI_RW(0);                      // ..then read registervalue
  nRF_CS = 1;                                   // CSN high, terminate SPI communication

  return(reg_val);                              // return register value
}
unsigned char LO1_Get_Status(void)
{
  return nRF_SPI_Read(STATUS);
}
unsigned char LO1_Get_Current_Pipenum(void)
{
  return ((LO1_Get_Status() & RX_P_NO) >> 1);
}
unsigned int LO1_Read_RX_Pload(unsigned char *pBuf)
{
  unsigned char plWidth, pipe;
  plWidth = LO1_RD_RX_PW_n(pipe = LO1_Get_Current_Pipenum());   // Read current pipe's payload width
  nRF_SPI_Read_Buf(RD_RX_PLOAD, pBuf, plWidth);                 // Then get RX data

  return ((pipe << 8) + plWidth);                               // return pipe# & pipe#.plWidth
}
 /*unsigned char nRF_SPI_RW_Reg_IRQ(unsigned char reg, unsigned char value)
{
  unsigned char status;

  nRF_CS = 0;                                   // CSN low, init SPI transaction

  SSP1BUF = reg;
  //while (SSP1STAT.BF == 0)
  Delay_1us(10);
  status = SSP1BUF;  // free the register

  SSP1BUF = value;
  //while (SSP1STAT.BF == 0)
    Delay_1us(10);
  reg = SSP1BUF;  // free the register

  nRF_CS = 1;                                   // CSN high again

  return(status);                               // return nRF24LO1 status byte
}*/
void Tx_Mode ()
{
      CE_Pin_Active(CE_LOW);                        // Set CE pin low to enable stanby mode
      Delay_ms(200);
      LO1_Flush_TX();
      LO1_Flush_RX();
      //LO1_Clear_IRQ(MASK_IRQ_FLAGS);              // Clear interrupts
      nRF_SPI_RW_Reg(WRITE_REG + STATUS, MASK_IRQ_FLAGS);
      //ucIRQ_Source = CLEAR;
      //ucLastStat = ucLinkStat = LINK_ESTABLISH;   // LINK_ESTABLISH = 0x02

      nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);      // Enable Auto.Ack:Pipes 0-5
      nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);  // Enable Pipes 0-5
      nRF_SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x2a); // 500?s + 86?s, 10 retrans...
      nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);       // Select RF channel 40
      nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);     // Select Address
      nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x08);   // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
      nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);     // Set PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

      nRF_SPI_Write_Buf(WRITE_REG + TX_ADDR, &ADDRESS_P0, sizeof(ADDRESS_P0));    // Writes TX_Address to nRF24LO1
      nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // RX_Addr0 same as TX_Adr for Auto.Ack
      CE_Pin_Active(CE_HIGH);
}
/*void Tx_Mode_runtime()
{
      CE_Pin_Active(CE_LOW);                        // Set CE pin low to enable stanby mode
      //Delay_ms(200);
      LO1_Flush_TX();
      LO1_Flush_RX();
      LO1_Clear_IRQ(MASK_IRQ_FLAGS);              // Clear interrupts
      //nRF_SPI_RW_Reg(WRITE_REG + STATUS, MASK_IRQ_FLAGS);
      //ucIRQ_Source = CLEAR;
      //ucLastStat = ucLinkStat = LINK_ESTABLISH;   // LINK_ESTABLISH = 0x02

      //nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);      // Enable Auto.Ack:Pipes 0-5
      //nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);  // Enable Pipes 0-5
      //nRF_SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x2a); // 500?s + 86?s, 10 retrans...
      //nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);       // Select RF channel 40

      //nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x08);   // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
      //nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);     // Set PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

      //nRF_SPI_Write_Buf(WRITE_REG + TX_ADDR, &ADDRESS_P0, sizeof(ADDRESS_P0));    // Writes TX_Address to nRF24LO1
      //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // RX_Addr0 same as TX_Adr for Auto.Ack
      CE_Pin_Active(CE_HIGH);
      Delay_ms(50);                                 //max 1 ms
      //CE_Pin_Active(CE_LOW);
}*/
void Rx_Mode ()
{

      CE_Pin_Active(CE_LOW);                        // Set CE pin low to enable stanby mode
      Delay_ms(1);                                  // 200ms
      LO1_Flush_TX();
      LO1_Flush_RX();

      LO1_Clear_IRQ(MASK_IRQ_FLAGS);                // Clear interrupts
```

```
    nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);              // Enable Auto.Ack:Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);          // Enable Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);                // Select RF channel 40
    nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);           // Select Addres
    nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x0f);           // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);             // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // Use the same address on the RX device as the TX devi
    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P1, &ADDRESS_P1, sizeof(ADDRESS_P1));

    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P2, 0x03);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P3, 0x04);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P4, 0x05);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P5, 0x06);

    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); // Select same RX payload width as TX Payload width
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P1, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P2, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P3, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P4, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P5, RX_PLOAD_WIDTH);

    CE_Pin_Active(CE_HIGH);
}
void Rx_Mode_Runtime()
{
    //CE_Pin_Active(CE_LOW);                              // Set CE pin low to enable stanby mode
    //Delay_ms(1);                                          // 200ms
    //LO1_Flush_TX();
    //LO1_Flush_RX();

    //LO1_Clear_IRQ(MASK_IRQ_FLAGS);                      // Clear interrupts

    //nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);            // Enable Auto.Ack:Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);        // Enable Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);              // Select RF channel 40
    //nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);         // Select Addres
    //nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x0f);         // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    //nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);           // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // Use the same address on the RX device as the TX de
    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P1, &ADDRESS_P1, sizeof(ADDRESS_P1));

    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P2, 0x03);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P3, 0x04);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P4, 0x05);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P5, 0x06);

    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); // Select same RX payload width as TX Payload width
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P1, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P2, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P3, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P4, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P5, RX_PLOAD_WIDTH);

    CE_Pin_Active(CE_HIGH);
    Delay_ms(30);
}
void Rx_Real_Mode()
{
    Rx_Mode_Runtime();
        do
        {
          //P00 = ~P00;
          uiRX_info = LO1_Read_RX_Pload(RX_pload);        // Read current payload
          // temp  = LO1_Get_FIFO() & 0x02;    temp  >> 1;

        }
        while(!(LO1_Get_FIFO() & RX_EMPTY));              // ..until FIFO empty
         //while( temp == 0 );
        temp  = LO1_Get_FIFO() & 0x01;    // temp  >> 1;

        //P14 = temp;

        //uiRX_info = LO1_Read_RX_Pload(RX_pload);
        LO1_Clear_IRQ(MASK_RX_DR_FLAG);

        /*if (RX_pload[0] == 0x31){ P00 = ~P00; RX_pload[0] = 0; }
        if (RX_pload[1] == 0x32){ P01 = ~P01; RX_pload[1] = 0; }
        if (RX_pload[2] == 0x33){ P02 = ~P02; RX_pload[2] = 0; }
        if (RX_pload[3] == 0x34){ P03 = ~P03; RX_pload[3] = 0; }
        if (RX_pload[4] == 0x35){ P04 = ~P04; RX_pload[4] = 0; }*/
        CE_Pin_Active(CE_LOW);
}
void Tx_Real_Mode()
{
    Send_Packet();
    P13 = ~P13;
    Delay_ms(1);
    LO1_Clear_IRQ(MASK_TX_DS_FLAG);
}
/*void Decode_output()
{
    int DO;
    unsigned char Data_All;
    DO = (3*(((Address_Route>>4) % 9))+1);
    if ( Send_Decode > Data_All )
    {
        Send_Decode = 0;
    }
    switch ( Send_Decode )
    {
        case 0:
        break;
    }
    Send_Decode++;
}*/
void Decode_input()
{
    int DI;
    DI = (3*(((Address_Route>>4) % 9))+1);

    switch ( (RX_pload[DI] & 0xf0) >> 4 )
    {
        case 0:
        /*SOBUF = RX_pload[2];
        while(TIO)
        {
            TIO = 0;
        }*/
        break;
        case 1:
        break;
    }
}
```

```
void Check_Device()
{
   /*   int CD,set,pload;
    CD = ((Address_Route>>4) % 9);
    if(CD == 0)
    {
        Rx_Real_Mode();

        for(set=3;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[0] = Address_Route;
    }
    else if(CD == 1)
    {
        Rx_Real_Mode();

        for(set=0;set<=2;set++)
        TX_pload[set] = RX_pload[set];

        for(set=6;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[3] = Address_Route;
    }
    else if(CD == 2)
    {
        Rx_Real_Mode();

        for(set=0;set<5;set++)
        TX_pload[set] = RX_pload[set];

        for(set=9;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[6] = Address_Route;
    }
    else if(CD == 3)
    {
        Rx_Real_Mode();

        for(set=0;set<=8;set++)
        TX_pload[set] = RX_pload[set];

        for(set=12;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[9] = Address_Route;
    }
    else if(CD == 4)
    {
        Rx_Real_Mode();

        for(set=0;set<=11;set++)
        TX_pload[set] = RX_pload[set];

        for(set=15;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[12] = Address_Route;
    }
    else if(CD == 5)
    {
        Rx_Real_Mode();

        for(set=0;set<=14;set++)
        TX_pload[set] = RX_pload[set];

        for(set=18;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[15] = Address_Route;
    }
    else if(CD == 6)
    {
        Rx_Real_Mode();

        for(set=0;set<=17;set++)
        TX_pload[set] = RX_pload[set];

        for(set=21;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[18] = Address_Route;
    }
    else if(CD == 7)
    {
        Rx_Real_Mode();

        for(set=0;set<=20;set++)
        TX_pload[set] = RX_pload[set];

        for(set=24;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[21] = Address_Route;
    }
```

```
    else if(CD == 8)
    {
        Rx_Real_Mode();

        for(set=0;set<=23;set++)
        TX_pload[set] = RX_pload[set];

        for(set=27;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[24] = Address_Route;
    }*/
    Rx_Real_Mode();
    CE_Pin_Active(CE_LOW);
}

void Read_Data ()
{
    int set;
    /*int SD;
    SD = ((Address_Route>>4) % 9);
    if(SD == 0)
    {
    if(((Address_Route & 0xf0) == (RX_pload[0] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[6] & 0x0f)) || (((Address_Route & 0x0f)+15) == (RX_pload[30] & 0x0f)))
        {
          Decode_input();
          /*if(RX_pload[2] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
          //P00 = ~P00;
          Check = 1;
        }
    if(RX_pload[6] == 0x00)
        {
        if(((Address_Route & 0xf0) == (RX_pload[0] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[3] & 0x0f)) || (((Address_Route & 0x0f)+15) == (RX_pload[30] & C
          //P02 = ~P02;
          Decode_input();
          Check = 1;
        }
    }
    else if(SD == 1)
    {
        if(((Address_Route & 0xf0) == (RX_pload[3] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[0] & 0x0f)))
        //if(((Address_Route>>4) == (RX_pload[3]>>4)) && (((Address_Route&0x0f)-1] && (RX_pload[0]&0x0f)))
        {
          Decode_input();
          /*if(RX_pload[5] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
          Check = 1;
        }
    }
    else if(SD == 2)
    {
        if(((Address_Route & 0xf0) && (RX_pload[6] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[3] & 0x0f)))
        {
          Decode_input();
          /*if(RX_pload[8] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
          if (TX_pload[7] == 0x00) P00 = ~P00;
          if (TX_pload[8] == 0x01) P01 = ~P01;
          Check = 1;
        }
    }
    else if(SD == 3)
    {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[15] & 0x0f)) || (((Address_Route & 0x0f)-2) == (RX_pload[3] & 0x0f
        {
          if(RX_pload[11] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
        if(RX_pload[15] == 0x00)
        {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[12] & 0x0f)) || (((Address_Route & 0x0f)-2) == (RX_pload[3] & 0x
          P02 = ~P02;
        }
    }
    else if(SD == 4)
    {
        if(((Address_Route & 0xf0) && (RX_pload[12] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[6] & 0x0f)))
        {
          if(RX_pload[14] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
    else if(SD == 5)
    {
        if(((Address_Route & 0xf0) && (RX_pload[15] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[12] & 0x0f)))
        {
          if(RX_pload[17] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
    else if(SD == 6)
    {
        if(((Address_Route & 0xf0) && (RX_pload[18] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[24] & 0x0f)) || (((Address_Route & 0x0f)-3) == (RX_pload[6] & 0xC
        {
          if(RX_pload[20] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
        if(RX_pload[24] == 0x00)
        {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[21] & 0x0f)) || (((Address_Route & 0x0f)-3) == (RX_pload[2] & 0x
          P02 = ~P02;
        }
    }
    else if(SD == 7)
    {
        if(((Address_Route & 0xf0) && (RX_pload[21] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[18] & 0x0f)))
        {
          if(RX_pload[23] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
    else if(SD == 8)
    {
        if(((Address_Route & 0xf0) && (RX_pload[24] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[21] & 0x0f)))
        {
          if(RX_pload[26] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }*/
    //P02 = ~P02;
    if(((Address_Route&0xf0)&&(RX_pload[30]&0xf0))&&(((Address_Route&0xf0)-15)==(RX_pload[0]&0x0f)))
    {
    for(set=0;set<=31;set++)
    {
        x = RX_pload[set];
    }
    }
}
```

```
void RF_Mode() interrupt 9 using 1
{
    RF = 0;
    P07 = ~P07;
    Rx_flag = 1;
    RF = 1;
}
void main()
{
 P0DIR = 0x10;
 P1DIR = 0x00;

 SOCON = 0x50;            //Mode 1 8-bit UART.
 SOBUF = 0x00;
 SORELH = 0x03;
 SORELL = 0xF3;
 ADCON = 0x80;

 PCON = PCON| 0x80;


 nRF_CE = 0;
 nRF_CS = 1;

 RFCKEN = 1;
 Delay_ms(500);

    CE_Pin_Active(CE_LOW);                                  // Set CE pin low to enable stanby mode
    Delay_ms(200);
    LO1_Flush_TX();
    LO1_Flush_RX();

    /*for(p=0;p<32;p++)
                {
                SOBUF = 'A';
                Delay_ms(1);
                while(TI0);
                TI0 = 0 ;

                )*/
    //printf("Heeello");
    Delay_ms(500);

    Rx_Mode();
    RF = 1;
 ES0 = 0;
 RI0 = 0;
 TI0 = 1;
 EA = 1;



 while(1)
 {
    if(RI0 == 1)
    {
    P12 = ~P12;
    keep = (unsigned int)SOBUF;
        Datanet = keep;
        RI0 = 0;
        LOOP = 1;
    //  printf("%c\r\n",keep);

    }

    if(LOOP == 1)
        {
            P06 = ~P06;


            //Decode_output();
            TX_pload[0] = 0x00;
            TX_pload[1] = 0x00;
            TX_pload[2] = Datanet;
            TX_pload[3] = 0x10;
            for (p=4;p<32;p++)
                {
                TX_pload[p] = 0x00;
                }

            TX_pload[30] = 0xff;

            Tx_Mode();
            Tx_Real_Mode();
            P01 = ~P01;

            //P06 = ~P06;
        /*  for (p=0;p<32;p++)
                {
                printf("%c",TX_pload[p]);
                )*/
            LOOP = 0;
            Delay_ms(100);

        }
            if(Rx_flag == 1)
            {
                P02 = ~P02;
                Check_Device();
                Read_Data();
                //CE_Pin_Active(CE_HIGH);
                Rx_flag = 0;


                //for(p=0;p<32;p++)
                //{
                printf("%c",RX_pload[2]);
                //}


            }
        Rx_Mode();

        Delay_ms(10);
 }
}
```

โปรแกรมสวิตซ์ไมโครคอนโทรลเลอร์

```
#include<reg51e1.H>
#include<stdio.H>
#include "main.h"
#include "nrf_lab.h"
#include "nRF_defines.h"

#define  nRF_CE   RFCE
#define  nRF_CS   RFCSN
#define  SSP1BUF  SPIRDAT

#define  Address_Route  0x10

unsigned char LOOP, inputdata, Check, Reset_Loop ,Test ,Checksum_Tx = 0,Do,onetime;
unsigned int Datanec,p;
unsigned char temp;
unsigned int Rx_flag ,Uart_flag , Decode_in, Decode_out = 0x0000, Send_Decode = 0,SW;


void Delay_1us(volatile unsigned int n){
   unsigned char i;
   while(n--)
    for(i=0;i<3;i++);
 }
 void Delay_100us(volatile unsigned int n){
   unsigned char i;
   while(n--)
    for(i=0;i<35;i++);
 }

 void Delay_ms(int i)
{
    int j;
    for(; i>0; i--)
    for(j=0; j<100; j++);
}

void Int() interrupt 0 using 2
{
    if (P12 == 1 && SW == 0x62)
    {
        SW = 0x61;
        Do = 1;
        onetime = 1;
    }
    if (P12 == 0 && SW == 0x61)
    {
        SW = 0x62;
        Do = 1;
        onetime = 1;
    }
}
void CE_Pin_Active(unsigned char action)    // CE pin high, low or pulse..
{
  switch(action)
  {
    case CE_LOW:                    // action == 0, CE low
     nRF_CE = 0;
    break;

    case CE_HIGH:                   // action == 1, CE high
     nRF_CE = 1;
    break;

    case CE_PULSE:                  // action == 2, CE pulse (10?s)
     // TMR2   = 0;
     nRF_CE = 1;                    // Set CE pin high
      Delay_1us(100);
      nRF_CE = 0;
     //TMR2ON_bit = 1;                // Start Timer2, CE pulse timer
    break;
  }
}
unsigned char LO1_Clear_IRQ(unsigned char irq_flag)
{
  return nRF_SPI_RW_Reg(WRITE_REG + STATUS, irq_flag);
}

unsigned char nRF_SPI_RW(unsigned char byte1)
{
  unsigned char status;
  //Delay_1us(10);
  SSP1BUF = byte1;
  Delay_1us(1);        // Delay 300s ok
  status  = SSP1BUF;
  return(status);
}
unsigned char nRF_SPI_RW_Reg(unsigned char reg, unsigned char value)
{
  unsigned char status;

  nRF_CS = 0;                                // CSN low, init SPI transaction
  status = nRF_SPI_RW(reg);                  // select register
  nRF_SPI_RW(value);                         // ..and write value to it..
  nRF_CS = 1;                                // CSN high again

  return(status);                            // return nRF24LO1 status byte
}
unsigned char nRF_SPI_Write_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
  unsigned char status, byte_ctr;

  nRF_CS = 0;                                // Set CSN low, init SPI transaction
  status = nRF_SPI_RW(reg);                  // Select register to write to and read status byte

  for(byte_ctr=0; byte_ctr<bytes; byte_ctr++)     // then write all byte in buffer(*pBuf)
  {
    nRF_SPI_RW(*pBuf++);
  }


  nRF_CS = 1;                                // Set CSN high again

  return(status);                            // return nRF24LO1 status byte
}

unsigned char LO1_RD_RX_PW_n(unsigned char pipe)
{
  return nRF_SPI_Read(RX_PW_P0 + pipe);
}
void LO1_Write_TX_Pload(unsigned char *pBuf, unsigned char plWidth)
{
  nRF_SPI_Write_Buf(WR_TX_PLOAD, pBuf, plWidth);
}

unsigned char Send_Packet()                  // Send one data packet, controlled by Button1(SW1)
{
   LO1_Write_TX_Pload(&TX_pload, TX_PLOAD_WIDTH);     // Write new TX payload
   CE_Pin_Active(CE_PULSE);                           // Enable CE pulse, 12?s
  //ucTry_Ctr = 0;                              // Reset Try_Ctr before nex transmitt..
  return 1;
}
```

```
void LO1_Flush_RX(void)
{
  nRF_SPI_RW_Reg(FLUSH_RX,0);
}
void LO1_Flush_TX(void)
{
  nRF_SPI_RW_Reg(FLUSH_TX,0);
}
unsigned char LO1_Get_FIFO(void)
{
  return nRF_SPI_Read(FIFO_STATUS);
}
unsigned char nRF_SPI_Read_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
  unsigned char status, byte_ctr;

  nRF_CS = 0;                                   // Set CSN low, init SPI tranaction
  status = nRF_SPI_RW(reg);                     // Select register to write to and read status byte

  for(byte_ctr=0;byte_ctr<bytes;byte_ctr++)     // Perform SPI_RW to read byte from nRF24LO1
  {
    pBuf[byte_ctr] = nRF_SPI_RW(0);
  }

  nRF_CS = 1;                                   // Set CSN high again

  return(status);                               // return nRF24LO1 status byte
}
unsigned char nRF_SPI_Read(unsigned char reg)
{
  unsigned char reg_val;

  nRF_CS = 0;                                   // CSN low, initialize SPI communication...
  nRF_SPI_RW(reg);                              // Select register to read from..
  reg_val = nRF_SPI_RW(0);                      // ..then read registervalue
  nRF_CS = 1;                                   // CSN high, terminate SPI communication

  return(reg_val);                              // return register value
}
unsigned char LO1_Get_Status(void)
{
  return nRF_SPI_Read(STATUS);
}
unsigned char LO1_Get_Current_Pipenum(void)
{
  return ((LO1_Get_Status() & RX_P_NO) >> 1);
}
unsigned int LO1_Read_RX_Pload(unsigned char *pBuf)
{
  unsigned char plWidth, pipe;
  plWidth = LO1_RD_RX_PW_n(pipe = LO1_Get_Current_Pipenum());   // Read current pipe's payload width
  nRF_SPI_Read_Buf(RD_RX_PLOAD, pBuf, plWidth);                 // Then get RX data

  return ((pipe << 8) + plWidth);                               // return pipe# & pipe#.plWidth
}
/*unsigned char nRF_SPI_RW_Reg_IRQ(unsigned char reg, unsigned char value)
{
  unsigned char status;

  nRF_CS = 0;                                   // CSN low, init SPI transaction

  SSP1BUF = reg;
  //while (SSP1STAT.BF == 0)
  Delay_1us(10);
  status = SSP1BUF;  // free the register

  SSP1BUF = value;
  //while (SSP1STAT.BF == 0)
    Delay_1us(10);
  reg = SSP1BUF;  // free the register

  nRF_CS = 1;                                   // CSN high again

  return(status);                               // return nRF24LO1 status byte
}*/
void Tx_Mode ()
{
    CE_Pin_Active(CE_LOW);                          // Set CE pin low to enable stanby mode
    Delay_ms(200);
    LO1_Flush_TX();
    LO1_Flush_RX();
    //LO1_Clear_IRQ(MASK_IRQ_FLAGS);                // Clear interrupts
    nRF_SPI_RW_Reg(WRITE_REG + STATUS, MASK_IRQ_FLAGS);
    //ucIRQ_Source = CLEAR;
    //ucLastStat = ucLinkStat = LINK_ESTABLISH;     // LINK_ESTABLISH = 0x02

    nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);        // Enable Auto.Ack:Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);    // Enable Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x2a);   // 500?s + 86?s, 10 retrans...
    nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);         // Select RF channel 40
    nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);       // Select Addres
    nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x08);     // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);       // Set PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

    nRF_SPI_Write_Buf(WRITE_REG + TX_ADDR, &ADDRESS_P0, sizeof(ADDRESS_P0));    // Writes TX_Address to nRF24LO1
    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // RX_Addr0 same as TX_Adr for Auto.Ack
    CE_Pin_Active(CE_HIGH);
}
/*void Tx_Mode_runtime()
{
    CE_Pin_Active(CE_LOW);                          // Set CE pin low to enable stanby mode
    //Delay_ms(200);
    LO1_Flush_TX();
    LO1_Flush_RX();
    LO1_Clear_IRQ(MASK_IRQ_FLAGS);                  // Clear interrupts
    //nRF_SPI_RW_Reg(WRITE_REG + STATUS, MASK_IRQ_FLAGS);
    //ucIRQ_Source = CLEAR;
    //ucLastStat = ucLinkStat = LINK_ESTABLISH;     // LINK_ESTABLISH = 0x02

    //nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);        // Enable Auto.Ack:Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);    // Enable Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x2a);   // 500?s + 86?s, 10 retrans...
    //nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);         // Select RF channel 40

    //nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x08);     // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    //nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);       // Set PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

    //nRF_SPI_Write_Buf(WRITE_REG + TX_ADDR, &ADDRESS_P0, sizeof(ADDRESS_P0));    // Writes TX_Address to nRF24LO1
    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // RX_Addr0 same as TX_Adr for Auto.Ack
    CE_Pin_Active(CE_HIGH);
    Delay_ms(50);                                 //max 1 ms
    //CE_Pin_Active(CE_LOW);
}*/
void Rx_Mode ()
{
    CE_Pin_Active(CE_LOW);                          // Set CE pin low to enable stanby mode
    Delay_ms(1);                                    // 200ms
    LO1_Flush_TX();
    LO1_Flush_RX();

    LO1_Clear_IRQ(MASK_IRQ_FLAGS);                  // Clear interrupts
```

```
    nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);              // Enable Auto.Ack:Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);          // Enable Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);                // Select RF channel 40
    nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);           // Select Addres
    nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x0f);           // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);             // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // Use the same address on the RX device as th
    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P1, &ADDRESS_P1, sizeof(ADDRESS_P1));

    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P2, 0x03);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P3, 0x04);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P4, 0x05);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P5, 0x06);

    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); // Select same RX payload width as TX Payload width
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P1, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P2, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P3, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P4, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P5, RX_PLOAD_WIDTH);

    CE_Pin_Active(CE_HIGH);

}
void Rx_Mode_Runtime()
{
    //CE_Pin_Active(CE_LOW);                              // Set CE pin low to enable stanby mode
    //Delay_ms(1);                                          // 200ms
    //LO1_Flush_TX();
    //LO1_Flush_RX();

    //LO1_Clear_IRQ(MASK_IRQ_FLAGS);                      // Clear interrupts

    //nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);            // Enable Auto.Ack:Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);        // Enable Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);              // Select RF channel 40
    //nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);         // Select Addres
    //nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x0f);         // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    //nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);           // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // Use the same address on the RX device as
    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P1, &ADDRESS_P1, sizeof(ADDRESS_P1));

    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P2, 0x03);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P3, 0x04);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P4, 0x05);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P5, 0x06);

    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); // Select same RX payload width as TX Payload width
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P1, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P2, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P3, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P4, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P5, RX_PLOAD_WIDTH);

    CE_Pin_Active(CE_HIGH);
    Delay_ms(30);
}
void Rx_Real_Mode()
{
    Rx_Mode_Runtime();
        do
        {
          //P00 = ~P00;
          uiRX_info = LO1_Read_RX_Pload(RX_pload);      // Read current payload
          // temp  = LO1_Get_FIFO() & 0x02;    temp  >> 1;

        }
        while(!(LO1_Get_FIFO() & RX_EMPTY));            // ..until FIFO empty
         //while( temp == 0 );
        temp  = LO1_Get_FIFO() & 0x01;    // temp  >> 1;

        //P14 = temp;

        //uiRX_info = LO1_Read_RX_Pload(RX_pload);
        LO1_Clear_IRQ(MASK_RX_DR_FLAG);

        /*if (RX_pload[0] == 0x31){ P00 = ~P00; RX_pload[0] = 0; }
        if (RX_pload[1] == 0x32){ P01 = ~P01; RX_pload[1] = 0; }
        if (RX_pload[2] == 0x33){ P02 = ~P02; RX_pload[2] = 0; }
        if (RX_pload[3] == 0x34){ P03 = ~P03; RX_pload[3] = 0; }
        if (RX_pload[4] == 0x35){ P04 = ~P04; RX_pload[4] = 0; }*/
        CE_Pin_Active(CE_LOW);
}
void Tx_Real_Mode()
{
    Send_Packet();
    P13 = ~P13;
    Delay_ms(1);
    LO1_Clear_IRQ(MASK_TX_DS_FLAG);
}
/*void Decode_output()
{   for
    int DO;
    unsigned char Data_All;
    DO = (3*(((Address_Route>>4) % 9))+1);
    if ( Send_Decode > Data_All )
    {
        Send_Decode = 0;
    }
    switch ( Send_Decode )
    {
        case 0:
        break;
    }
    Send_Decode++;
}*/
void Decode_input()
{
    int DI;
    DI = (3*(((Address_Route>>4) % 9))+1);

    switch ( (RX_pload[DI] & 0xf0) >> 4 )
    {
        case 0:
        /*SOBUF = RX_pload[2];
        while(TI0)
        {
            TI0 = 0;
        }*/
        break;
        case 1:
        break;
    }
}
```

```
void Check_Device()
{
    int CD,set,pload;
    CD = ((Address_Route>>4) % 9);
    if(CD == 0)
    {
        Rx_Real_Mode();

        for(set=3;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[0] = Address_Route;
    }
    else if(CD == 1)
    {
        Rx_Real_Mode();

        for(set=0;set<=2;set++)
        TX_pload[set] = RX_pload[set];

        for(set=6;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[9] = Address_Route;
    }
    else if(CD == 2)
    {
        Rx_Real_Mode();

        for(set=0;set<5;set++)
        TX_pload[set] = RX_pload[set];

        for(set=9;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[6] = Address_Route;
    }
    else if(CD == 3)
    {
        Rx_Real_Mode();

        for(set=0;set<=8;set++)
        TX_pload[set] = RX_pload[set];

        for(set=12;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[9] = Address_Route;
    }
    else if(CD == 4)
    {
        Rx_Real_Mode();

        for(set=0;set<=11;set++)
        TX_pload[set] = RX_pload[set];

        for(set=15;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[12] = Address_Route;
    }
    else if(CD == 5)
    {
        Rx_Real_Mode();

        for(set=0;set<=14;set++)
        TX_pload[set] = RX_pload[set];

        for(set=18;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[15] = Address_Route;
    }
    else if(CD == 6)
    {
        Rx_Real_Mode();

        for(set=0;set<=17;set++)
        TX_pload[set] = RX_pload[set];

        for(set=21;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[18] = Address_Route;
    }
    else if(CD == 7)
    {
        Rx_Real_Mode();

        for(set=0;set<=20;set++)
        TX_pload[set] = RX_pload[set];

        for(set=24;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[21] = Address_Route;
    }
    else if(CD == 8)
    {
        Rx_Real_Mode();

        for(set=0;set<=23;set++)
        TX_pload[set] = RX_pload[set];

        for(set=27;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xE0;
        }
        TX_pload[24] = Address_Route;
    }
}
```

```
void Read_Data ()
{
    int SD;
    SD = ((Address_Route>>4) % 9);
    if(SD == 0)
    {
    if(((Address_Route & 0xf0) == (RX_pload[0] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[6] & 0x0f)) || (((Address_Route & 0x0f)+15) == (RX_pload[30] & 0x0f)))
        {
          Decode_input();
          /*if(RX_pload[2] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }*/
          //P00 = ~P00;
          Check = 1;
        }
    if(RX_pload[6] == 0x00)
        {
        if(((Address_Route & 0xf0) == (RX_pload[0] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[3] & 0x0f)) || (((Address_Route & 0x0f)+15) == (RX_pload[30] & 0
          //P02 = ~P02;
          Decode_input();
          Check = 1;
        }
    }
    else if(SD == 1)
    {
        if(((Address_Route & 0xf0) == (RX_pload[3] & 0xf0)) && ((Address_Route & 0x0f)-1) == (RX_pload[0] & 0x0f))
        //if(((Address_Route>>4) == (RX_pload[3]>>4)) && ((Address_Route&0x0f)-1) && (RX_pload[0]&0x0f)))
        {
          Decode_input();
          /*if(RX_pload[5] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }*/
          Check = 1;
        }
    }
    else if(SD == 2)
    {
        if(((Address_Route & 0xf0) && (RX_pload[6] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[3] & 0x0f)))
        {
          Decode_input();
          /*if(RX_pload[8] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }*/
          if (TX_pload[7] == 0x00) P00 = ~P00;
          if (TX_pload[8] == 0x01) P01 = ~P01;
          Check = 1;
        }
    }
    else if(SD == 3)
    {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[15] & 0x0f)) || (((Address_Route & 0x0f)-2) == (RX_pload[3] & 0x0f
        {
          if(RX_pload[11] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
        if(RX_pload[15] == 0x00)
        {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[12] & 0x0f)) || (((Address_Route & 0x0f)-2) == (RX_pload[3] & 0x
          P02 = ~P02;
        }
    }
    else if(SD == 4)
    {
        if(((Address_Route & 0xf0) && (RX_pload[12] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[6] & 0x0f)))
        {
          if(RX_pload[14] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
    else if(SD == 5)
    {
        if(((Address_Route & 0xf0) && (RX_pload[15] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[12] & 0x0f)))
        {
          if(RX_pload[17] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
    else if(SD == 6)
    {
        if(((Address_Route & 0xf0) && (RX_pload[18] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[24] & 0x0f)) || (((Address_Route & 0x0f)-3) == (RX_pload[6] & 0x0
        {
          if(RX_pload[20] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
        if(RX_pload[24] == 0x00)
        {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[21] & 0x0f)) || (((Address_Route & 0x0f)-3) == (RX_pload[2] & 0x
          P02 = ~P02;
        }
    }
    else if(SD == 7)
    {
        if(((Address_Route & 0xf0) && (RX_pload[21] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[18] & 0x0f)))
        {
          if(RX_pload[23] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
    else if(SD == 8)
    {
        if(((Address_Route & 0xf0) && (RX_pload[24] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[21] & 0x0f)))
        {
          if(RX_pload[26] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }
        }
    }
}
void RF_Mode() interrupt 9 using 1
{
    RF = 0;
    P07 = ~P07;
    Rx_flag = 1;
    RF = 1;
}
```

```
void main()
{
 P0DIR = 0x10;
 P1DIR = 0xff;

    if (P12 == 1)
    {
        SW = 0x61;
        Do = 1;
    }
    if (P12 == 0)
    {
        SW = 0x62;
        Do = 1;
    }

//INTEXP =0x18;
//IP1 = 1;
//IP0 = 1;

S0CON = 0x50;           //Mode 1 8-bit UART.
S0BUF = 0x00;
S0RELH = 0x03;
S0RELL = 0xF3;
ADCON = 0x80;

PCON = PCON| 0x80;

RF = 0;
//ES0 = 1;
RI0 = 0;
TI0 = 0;
//EX0 = 1;
//EX1 = 1;
//EA = 1;
nRF_CE = 0;
nRF_CS = 1;

RFCKEN = 1;
Delay_ms(500);

    CE_Pin_Active(CE_LOW);                                  // Set CE pin low to enable stanby mode
    Delay_ms(200);
    L01_Flush_TX();
    L01_Flush_RX();


    Delay_ms(500);
    Rx_Mode();

while(1)
{
    if (P12 == 1 && SW == 0x62)
    {
        SW = 0x61;
        Do = 1;
        onetime = 1;
    }
    if (P12 == 0 && SW == 0x61)
    {
        SW = 0x62;
        Do = 1;
        onetime = 1;
    }

    if(Do == 1)
        {
            //P06 = ~P06;
            //Decode_output();              TX_pload
            TX_pload[0] = Address_Route;
            TX_pload[1] = 0x00;
            if(onetime != 0x63)
            {

            TX_pload[2] = SW;
            }
            else (TX_pload[2] = onetime ; )



            /*for (p=3;p<32;p++)
                {
                TX_pload[p] = 0x00;
                }*/
            Tx_Mode();                          //Tx_mode
            Tx_Real_Mode();
            Delay_ms(1000);
            //P06 = ~P06;
            Do = 0;
            Delay_ms(500);
            onetime = 0x63 ;




        }
            /*if(Rx_flag == 1)
            {
                //P00 = ~P00;
                Check_Device();
                Read_Data();
                //CE_Pin_Active(CE_HIGH);
                Rx_flag = 0;
            }
        Rx_Mode();*/
        Delay_ms(10);
    }
}
```

โปรแกรมโมดูลสั่งการควบคุมอุปกรณ์ส่องสว่าง

```
#include <reg24le1.H>
#include <stdio.H>
#include <intrins.H>
#include "main.h"
#include "nrf_lab.h"
#include "nRF_defines.h"

#define  nRF_CE    RFCE
#define  nRF_CS    RFCSN
#define  SSP1BUF   SPIRDAT

#define  Address_Route  0x00
//#define  Data_All  4

#define cs_on1  P01 = 0;
#define cs_off1 P01 = 1;
#define cs_on2  P03 = 0;
#define cs_off2 P03 = 1;
#define cs_on3  P11 = 0;
#define cs_off3 P11 = 1;
#define delay_nop  _nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();_nop_();

unsigned char temp;
unsigned int Rx_flag, Check, Decode_in, Decode_out = 0x0000, Send_Decode = 0 ,Result = 0x00 ,Sum = 0x00;
unsigned int Relay1 = 0, Relay2 = 0, Relay3 = 0;

unsigned char Read[3] ,Write[3] ,uc;
unsigned int ui ,Vrms = 220 ,Power = 0 ,Ireal = 0;
unsigned long Irms1 ,Irms2 ,Irms3;
float fVrms ,fIrms;

 void Delay_1us(volatile unsigned int n){
   unsigned char i;
   while(n--)
    for(i=0;i<3;i++);
 }
 void Delay_100us(volatile unsigned int n){
   unsigned char i;
   while(n--)
    for(i=0;i<35;i++);
 }

 void Delay_ms(volatile unsigned int n){
   while(n--)
    Delay_100us(50);
 }

void CE_Pin_Active(unsigned char action)     // CE pin high, low or pulse..
{
  switch(action)
  {
    case CE_LOW:                    // action == 0, CE low
     nRF_CE = 0;
    break;

    case CE_HIGH:                   // action == 1, CE high
     nRF_CE = 1;
    break;

    case CE_PULSE:                  // action == 2, CE pulse (10?s)
     // TMR2   = 0;
      nRF_CE = 1;                   // Set CE pin high
       Delay_1us(100);
       nRF_CE = 0;
      //TMR2ON_bit = 1;               // Start Timer2, CE pulse timer
    break;
  }
}
unsigned char LO1_Clear_IRQ(unsigned char irq_flag)
{
  return nRF_SPI_RW_Reg(WRITE_REG + STATUS, irq_flag);
}

unsigned char nRF_SPI_RW(unsigned char byte1)
{
  unsigned char status;
  //Delay_1us(10);
  SSP1BUF = byte1;
  Delay_1us(1);         // Delay 300s ok
  status  = SSP1BUF;
  return(status);
}
unsigned char nRF_SPI_RW_Reg(unsigned char reg, unsigned char value)
{
  unsigned char status;

  nRF_CS = 0;                                // CSN low, init SPI transaction
  status = nRF_SPI_RW(reg);                  // select register
  nRF_SPI_RW(value);                         // ..and write value to it..
  nRF_CS = 1;                                // CSN high again

  return(status);                            // return nRF24L01 status byte
}
unsigned char nRF_SPI_Write_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
  unsigned char status, byte_ctr;

  nRF_CS = 0;                                // Set CSN low, init SPI transaction
  status = nRF_SPI_RW(reg);                  // Select register to write to and read status byte

  for(byte_ctr=0; byte_ctr<bytes; byte_ctr++)     // then write all byte in buffer(*pBuf)
  {
    nRF_SPI_RW(*pBuf++);
  }


  nRF_CS = 1;                                // Set CSN high again

  return(status);                            // return nRF24L01 status byte
}

unsigned char LO1_RD_RX_PW_n(unsigned char pipe)
{
  return nRF_SPI_Read(RX_PW_P0 + pipe);
}
void LO1_Write_TX_Pload(unsigned char *pBuf, unsigned char plWidth)
{
  nRF_SPI_Write_Buf(WR_TX_PLOAD, pBuf, plWidth);
}

unsigned char Send_Packet()                       // Send one data packet, controlled by Button1(SW1)
{
    LO1_Write_TX_Pload(&TX_pload, TX_PLOAD_WIDTH);     // Write new TX payload
    CE_Pin_Active(CE_PULSE);                           // Enable CE pulse, 12?s
  //ucTry_Ctr = 0;                               // Reset Try_Ctr before new transmitt..
  return 1;
}
void LO1_Flush_RX(void)
{
  nRF_SPI_RW_Reg(FLUSH_RX,0);
}
```

```
void LO1_Flush_TX(void)
{
  nRF_SPI_RW_Reg(FLUSH_TX,0);
}
unsigned char LO1_Get_FIFO(void)
{
  return nRF_SPI_Read(FIFO_STATUS);
}
unsigned char nRF_SPI_Read_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
  unsigned char status, byte_ctr;

  nRF_CS = 0;                                   // Set CSN low, init SPI tranaction
  status = nRF_SPI_RW(reg);                     // Select register to write to and read status byte

  for(byte_ctr=0;byte_ctr<bytes;byte_ctr++)     // Perform SPI_RW to read byte from nRF24LO1
  {
    pBuf[byte_ctr] = nRF_SPI_RW(0);
  }

  nRF_CS = 1;                                   // Set CSN high again

  return(status);                               // return nRF24LO1 status byte
}
unsigned char nRF_SPI_Read(unsigned char reg)
{
  unsigned char reg_val;

  nRF_CS = 0;                                   // CSN low, initialize SPI communication...
  nRF_SPI_RW(reg);                              // Select register to read from..
  reg_val = nRF_SPI_RW(0);                      // ..then read registervalue
  nRF_CS = 1;                                   // CSN high, terminate SPI communication

  return(reg_val);                              // return register value
}
unsigned char LO1_Get_Status(void)
{
  return nRF_SPI_Read(STATUS);
}
unsigned char LO1_Get_Current_Pipenum(void)
{
  return ((LO1_Get_Status() & RX_P_NO) >> 1);
}
unsigned int LO1_Read_RX_Pload(unsigned char *pBuf)
{
  unsigned char plWidth, pipe;
  plWidth = LO1_RD_RX_PW_n(pipe = LO1_Get_Current_Pipenum());   // Read current pipe's payload width
  nRF_SPI_Read_Buf(RD_RX_PLOAD, pBuf, plWidth);                 // Then get RX data

  return ((pipe << 8) + plWidth);                               // return pipe# & pipe#.plWidth
}
 /*unsigned char nRF_SPI_RW_Reg_IRQ(unsigned char reg, unsigned char value)
{
  unsigned char status;

  nRF_CS = 0;                                   // CSN low, init SPI transaction

  SSP1BUF = reg;
  //while (SSP1STAT.BF == 0)
  Delay_1us(10);
  status = SSP1BUF;  // free the register

  SSP1BUF = value;
  //while (SSP1STAT.BF == 0)
    Delay_1us(10);
  reg = SSP1BUF;  // free the register

  nRF_CS = 1;                                   // CSN high again

  return(status);                               // return nRF24LO1 status byte
}*/
void Tx_Mode ()
{
      CE_Pin_Active(CE_LOW);                          // Set CE pin low to enable stanby mode
      Delay_ms(200);
      LO1_Flush_TX();
      LO1_Flush_RX();
      //LO1_Clear_IRQ(MASK_IRQ_FLAGS);                // Clear interrupts
      nRF_SPI_RW_Reg(WRITE_REG + STATUS, MASK_IRQ_FLAGS);
      //ucIRQ_Source = CLEAR;
      //ucLastStat = ucLinkStat = LINK_ESTABLISH;     // LINK_ESTABLISH = 0x02

      nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);        // Enable Auto.Ack:Pipes 0-5
      nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);    // Enable Pipes 0-5
      nRF_SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x2a);   // 500?s + 86?s, 10 retrans...
      nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);        // Select RF channel 40
      nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);       // Select Addres
      nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x08);     // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
      nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);       // Set PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

      nRF_SPI_Write_Buf(WRITE_REG + TX_ADDR, &ADDRESS_P0, sizeof(ADDRESS_P0));    // Writes TX_Address to nRF24LO1
      nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // RX_Addr0 same as TX_Adr for Auto.Ack
      CE_Pin_Active(CE_HIGH);
}
/*void Tx_Mode_runtime()
{
      CE_Pin_Active(CE_LOW);                          // Set CE pin low to enable stanby mode
      //Delay_ms(200);
      LO1_Flush_TX();
      LO1_Flush_RX();
      LO1_Clear_IRQ(MASK_IRQ_FLAGS);                // Clear interrupts
      //nRF_SPI_RW_Reg(WRITE_REG + STATUS, MASK_IRQ_FLAGS);
      //ucIRQ_Source = CLEAR;
      //ucLastStat = ucLinkStat = LINK_ESTABLISH;     // LINK_ESTABLISH = 0x02

      //nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);        // Enable Auto.Ack:Pipes 0-5
      //nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);    // Enable Pipes 0-5
      //nRF_SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x2a);   // 500?s + 86?s, 10 retrans...
      //nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);        // Select RF channel 40

      //nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x08);     // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
      //nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);       // Set PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

      //nRF_SPI_Write_Buf(WRITE_REG + TX_ADDR, &ADDRESS_P0, sizeof(ADDRESS_P0));    // Writes TX_Address to nRF24LO1
      //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // RX_Addr0 same as TX_Adr for Auto.Ack
      CE_Pin_Active(CE_HIGH);
      Delay_ms(50);                                 //max 1 ms
      //CE_Pin_Active(CE_LOW);
}*/
void Rx_Mode ()
{

      CE_Pin_Active(CE_LOW);                          // Set CE pin low to enable stanby mode
      Delay_ms(1);                                    // 200ms
      LO1_Flush_TX();
      LO1_Flush_RX();

      LO1_Clear_IRQ(MASK_IRQ_FLAGS);                // Clear interrupts
```

```
    nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);              // Enable Auto.Ack:Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);          // Enable Pipes 0-5
    nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);                // Select RF channel 40
    nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);           // Select Addres
    nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x0f);           // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);             // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // Use the same address on the RX device as the TX device
    nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P1, &ADDRESS_P1, sizeof(ADDRESS_P1));

    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P2, 0x03);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P3, 0x04);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P4, 0x05);
    nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P5, 0x06);

    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); // Select same RX payload width as TX Payload width
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P1, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P2, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P3, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P4, RX_PLOAD_WIDTH);
    nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P5, RX_PLOAD_WIDTH);

    CE_Pin_Active(CE_HIGH);

}
void Rx_Mode_Runtime()
{
    //CE_Pin_Active(CE_LOW);                              // Set CE pin low to enable stanby mode
    //Delay_ms(1);                                          // 200ms
    //LO1_Flush_TX();
    //LO1_Flush_RX();

    //LO1_Clear_IRQ(MASK_IRQ_FLAGS);                      // Clear interrupts

    //nRF_SPI_RW_Reg(WRITE_REG + EN_AA, 0x3f);            // Enable Auto.Ack:Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x3f);        // Enable Pipes 0-5
    //nRF_SPI_RW_Reg(WRITE_REG + RF_CH, 40);              // Select RF channel 40
    //nRF_SPI_RW_Reg(WRITE_REG + SETUP_AW, 0x03);         // Select Addres
    //nRF_SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x0f);         // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
    //nRF_SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);           // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, &ADDRESS_P0, sizeof(ADDRESS_P0)); // Use the same address on the RX device as the TX device
    //nRF_SPI_Write_Buf(WRITE_REG + RX_ADDR_P1, &ADDRESS_P1, sizeof(ADDRESS_P1));

    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P2, 0x03);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P3, 0x04);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P4, 0x05);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_ADDR_P5, 0x06);

    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); // Select same RX payload width as TX Payload width
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P1, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P2, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P3, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P4, RX_PLOAD_WIDTH);
    //nRF_SPI_RW_Reg(WRITE_REG + RX_PW_P5, RX_PLOAD_WIDTH);

    CE_Pin_Active(CE_HIGH);
    Delay_ms(30);
}
void Rx_Real_Mode()
{
    Rx_Mode_Runtime();
        do
        {
          //P00 = ~P00;
          uiRX_info = LO1_Read_RX_Pload(RX_pload);      // Read current payload
          // temp  = LO1_Get_FIFO() & 0x02;    temp  >> 1;

        }
        while(!(LO1_Get_FIFO() & RX_EMPTY));            // ..until FIFO empty
         //while( temp == 0 );
        temp  = LO1_Get_FIFO() & 0x01;    // temp  >> 1;

        //P14 = temp;

        //uiRX_info = LO1_Read_RX_Pload(RX_pload);
        LO1_Clear_IRQ(MASK_RX_DR_FLAG);

        /*if (RX_pload[0] == 0x31){ P00 = ~P00; RX_pload[0] = 0; }
        if (RX_pload[1] == 0x32){ P01 = ~P01; RX_pload[1] = 0; }
        if (RX_pload[2] == 0x33){ P02 = ~P02; RX_pload[2] = 0; }
        if (RX_pload[3] == 0x34){ P03 = ~P03; RX_pload[3] = 0; }
        if (RX_pload[4] == 0x35){ P04 = ~P04; RX_pload[4] = 0; }*/
        CE_Pin_Active(CE_LOW);
}
void Tx_Real_Mode()
{
    Send_Packet();
    Delay_ms(1);
    LO1_Clear_IRQ(MASK_TX_DS_FLAG);
}
unsigned char spi_master_read_write(unsigned char pLoad)
{
  Delay_ms(1);
  SPIMDAT = pLoad ;                     // Write data to SPI master
  while(!(SPIMSTAT & 0x04))();          // Wait for data available in rx_fifo
  return SPIMDAT;                       // Return data register
}
long ADE_Read (unsigned char add, unsigned char num)
{
    unsigned char i;
    unsigned long Value;

    spi_master_read_write(add);
    for(i = 0;i < num;i++)
    {
        Read[i] = spi_master_read_write(0x00);
    }

    Value = 0;
    for(i = 0;i < num;i++)
    {
        Value = Value << 8;
        Value |= Read[i];
    }
    return (Value);
}
void ADE_Write (unsigned char add, unsigned char num, unsigned char *write)
{
    unsigned char i;

    spi_master_read_write(add | 0x10);
    for(i = 0;i < num;i++)
    {
        spi_master_read_write(*write);
        write--;
    }
    //ADE_Write(add,num,(unsigned char *)&ui)       //How to
}
void Relay_onoff (int point ,int income)
{
```

```
392         RX_pload[point] = income;
393
394             /*if( RX_pload[point] == 0x00 )
395             {
396                 if(Relay1 == 1)
397                 {
398                     P00 = 1;
399                     Relay1 = 0;
400                 }
401                 /*if(Relay2 == 1)
402                 {
403                     P06 = 1;
404                     Relay2 = 0;
405                 }
406                 if(Relay3 == 1)
407                 {
408                     P08 = 1;
409                     Relay3 = 0;
410                 }
411             }
412             else if( RX_pload[point] == 0x01 )
413             {
414                 if(Relay1 == 0)
415                 {
416                     P01 = 1;
417                     Relay1 = 1;
418                 }
419                 /*if(Relay2 == 1)
420                 {
421                     P06 = 1;
422                     Relay2 = 0;
423                 }
424                 if(Relay3 == 1)
425                 {
426                     P08 = 1;
427                     Relay3 = 0;
428                 }
429             }
430             else if( RX_pload[point] == 0x02 )
431             {
432                 if(Relay1 == 1)
433                 {
434                     P00 = 1;
435                     Relay1 = 0;
436                 }
437                 /*if(Relay2 == 0)
438                 {
439                     P12 = 1;
440                     Relay2 = 1;
441                 }
442                 if(Relay3 == 1)
443                 {
444                     P08 = 1;
445                     Relay3 = 0;
446                 }
447             }
448             else if( RX_pload[point] == 0x03 )
449             {
450                 if(Relay1 == 0)
451                 {
452                     P01 = 1;
453                     Relay1 = 1;
454                 }
455                 /*if(Relay2 == 0)
456                 {
457                     P12 = 1;
458                     Relay2 = 1;
459                 }
460                 if(Relay3 == 1)
461                 {
462                     P08 = 1;
463                     Relay3 = 0;
464                 }
465             }
466             else if( RX_pload[point] == 0x04 )
467             {
468                 if(Relay1 == 1)
469                 {
470                     P00 = 1;
471                     Relay1 = 0;
472                 }
473                 /*if(Relay2 == 1)
474                 {
475                     P06 = 1;
476                     Relay2 = 0;
477                 }
478                 if(Relay3 == 0)
479                 {
480                     P13 = 1;
481                     Relay3 = 1;
482                 }
483             }
484             else if( RX_pload[point] == 0x05 )
485             {
486                 if(Relay1 == 0)
487                 {
488                     P01 = 1;
489                     Relay1 = 1;
490                 }
491                 /*if(Relay2 == 1)
492                 {
493                     P06 = 1;
494                     Relay2 = 0;
495                 }
496                 if(Relay3 == 0)
497                 {
498                     P13 = 1;
499                     Relay3 = 1;
500                 }
501             }
502             else if( RX_pload[point] == 0x06 )
503             {
504                 if(Relay1 == 1)
505                 {
506                     P00 = 1;
507                     Relay1 = 0;
508                 }
509                 /*if(Relay2 == 0)
510                 {
511                     P12 = 1;
512                     Relay2 = 1;
513                 }
514                 if(Relay3 == 0)
515                 {
516                     P13 = 1;
517                     Relay3 = 1;
518                 }
519             }
520             else if( RX_pload[point] == 0x07 )
521             {
522                 if(Relay1 == 0)
523                 {
524                     P01 = 1;
525                     Relay1 = 1;
526                 }
527                 /*if(Relay2 == 0)
528                 {
529                     P12 = 1;
530                     Relay2 = 1;
531                 }
532                 if(Relay3 == 0)
533                 {
534                     P13 = 1;
535                     Relay3 = 1;
536                 }
537             }*/
538         }
```

```
void Decode_output()
{
    int DO ,output;
    DO = (3*(((Address_Route>>4) & 3))+1);
    for (output=0;output<32;output++)
    {
        TX_pload[output] = RX_pload[output];
    }
    /*if ( Send_Decode < Data_All )
    {
        Send_Decode = 0;
    }
    output = (RX_pload[37] & 0x03);*/
    /*if(output == 0x00)
    {
            TX_pload[DO]     = 0x00;
            TX_pload[DO+1]   = 0x00;
            Decode_out       = Decode_out & 0x0000;

            if( Relay1 == 1 )
                Decode_out |= 0x0001;
            else
                Decode_out |= 0x0000;
            if( Relay2 == 1 )
                Decode_out |= 0x0002;
            else
                Decode_out |= 0x0000;
            if( Relay3 == 1 )
                Decode_out |= 0x0004;
            else
                Decode_out |= 0x0000;

            TX_pload[DO+1] = Decode_out & 0xff;
    }
    /*else if(output == 0x01)
    {
            TX_pload[DO]     = 0x10;
            TX_pload[DO+1]   = 0x00;

                cs_on1;
                delay_nop
                delay_nop
                Irms1 = ADE_Read (0x16 ,3);
                cs_off1;
                delay_nop

                Ireal = (Irms1 - 13600)/120;
                Power = (Vrms*Ireal)/100;

                TX_pload[DO+1] |= Power;
                Power = Power >> 8;
                TX_pload[DO] |= Power;
    }
    else if(output == 0x02)
    {
            TX_pload[DO]     = 0x20;
            TX_pload[DO+1]   = 0x00;

                cs_on2;
                delay_nop
                delay_nop
                Irms2 = ADE_Read (0x16 ,3);
                cs_off2;
                delay_nop

                Ireal = (Irms2 - 1500)/160;
                Power = (Vrms*Ireal)/100;

                TX_pload[DO+1] |= Power;
                Power = Power >> 8;
                TX_pload[DO] |= Power;
    }
    else if(output == 0x03)
    {
            TX_pload[DO]     = 0x30;
            TX_pload[DO+1]   = 0x00;

                cs_on3;
                delay_nop
                delay_nop
                Irms3 = ADE_Read (0x16 ,3);
                cs_off3;
                delay_nop

                Ireal = (Irms3 - 1500)/160;
                Power = (Vrms*Ireal)/100;

                TX_pload[DO+1] |= Power;
                Power = Power >> 8;
                TX_pload[DO] |= Power;
    )*/
}
void Decode_input()
{
    int DI;
    DI = (3*(((Address_Route>>4) & 3))+1);
    switch ( (RX_pload[DI] & 0xf0) >> 4 )
    {
        case 0x00:

            /*Decode_in = RX_pload[DI] & 0x0f;
            Decode_in = Decode_in << 4;
            Decode_in = RX_pload[DI+1] & 0xff;*/

            /*if( Result == 0 )
            {
                Sum = RX_pload[DI+1];
                Relay_onoff(DI+1,Sum);
                Result = Sum;
            }
            else if( (RX_pload[DI+1] > Result) && (Result != 0) )
            {
                Sum = RX_pload[DI+1] | Result;
                Relay_onoff(DI+1,Sum);
                Result = Sum;
            }
            else if( (RX_pload[DI+1] <= Result) && (Result != 0) )
            {
                Sum = RX_pload[DI+1] & Result;
                Relay_onoff(DI+1,Sum);
                Result = Sum;
            )*/

            //Sum = RX_pload[DI+1];
            //Relay_onoff(DI+1 ,Sum);
            switch (RX_pload[2])
                {
                    case 0x61:
                    P00 = 0;
                    P01 = 1;
                break;
                case 0x62:
                    P00 = 1;
                    P01 = 0;
                break;
                }
            Delay_ms(50);
            /*P07 = 1;
            P06 = 0;
            P05 = 0;
            P10 = 0;
            P12 = 0;
            P13 = 0;*/

        break;
    }
}
```

```
void Check_Device()
{
    int CD,set,pload;
    CD = ((Address_Route>>4) % 9);
    if(CD == 0)
    {
        Rx_Real_Mode();

        for(set=3;set<=31;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=3;pload<=24;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[0] = Address_Route;
    }
    else if(CD == 1)
    {
        Rx_Real_Mode();

        for(set=0;set<=2;set++)
        TX_pload[set] = RX_pload[set];

        for(set=6;set<=24;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[3] = Address_Route;
    }
    else if(CD == 2)
    {
        Rx_Real_Mode();

        for(set=0;set<5;set++)
        TX_pload[set] = RX_pload[set];

        for(set=9;set<=24;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[6] = Address_Route;
    }
    else if(CD == 3)
    {
        Rx_Real_Mode();

        for(set=0;set<=8;set++)
        TX_pload[set] = RX_pload[set];

        for(set=12;set<=24;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[9] = Address_Route;
    }
    else if(CD == 4)
    {
        Rx_Real_Mode();

        for(set=0;set<=11;set++)
        TX_pload[set] = RX_pload[set];

        for(set=15;set<=24;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[12] = Address_Route;
    }
    else if(CD == 5)
    {
        Rx_Real_Mode();

        for(set=0;set<=14;set++)
        TX_pload[set] = RX_pload[set];

        for(set=18;set<=24;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[15] = Address_Route;
    }
    else if(CD == 6)
    {
        Rx_Real_Mode();

        for(set=0;set<=17;set++)
        TX_pload[set] = RX_pload[set];

        for(set=21;set<=24;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[18] = Address_Route;
    }
    else if(CD == 7)
    {
        Rx_Real_Mode();

        for(set=0;set<=20;set++)
        TX_pload[set] = RX_pload[set];

        for(set=24;set<=30;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[21] = Address_Route;
    }
```

```
    else if(CD == 6)
    {
        Rx_Real_Mode();

        for(set=0;set<=23;set++)
        TX_pload[set] = RX_pload[set];

        for(set=27;set<=30;set++)
        TX_pload[set] = RX_pload[set];

        for(pload=0;pload<=30;pload++)
        {
            if((pload % 3) == 0)
            TX_pload[pload]= TX_pload[pload] & 0xf0;
        }
        TX_pload[24] = Address_Route;
    }
    CE_Pin_Active(CE_LOW);
}
void Read_Data ()
{
    int SD;
    SD = ((Address_Route>>4) % 9);
    if(SD == 0)
    {
    if(((Address_Route & 0xf0) == (RX_pload[0] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[6] & 0x0f)) || (((Address_Route & 0x0f)+15) == (RX_pload[30] & 0x0f)))
        {
          Decode_input();
          /*if(RX_pload[2] == 0x01)
          { Delay_ms(100); P00 = ~P00; Check = 1; }*/
          //P00 = ~P00;
          Check = 1;
        }
    if(RX_pload[6] == 0x00)
        {
        Decode_input();
          Check = 1;
        if(((Address_Route & 0xf0) == (RX_pload[0] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[3] & 0x0f)) || (((Address_Route & 0x0f)+15) == (RX_pload[30] & 0
          {
          Decode_input();
          Check = 1;
          }
        }
    }
    else if(SD == 1)
    {
        if(((Address_Route & 0xf0) == (RX_pload[3] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[0] & 0x0f)))
        //if(((Address_Route>>4) == (RX_pload[3]>>4)) && (((Address_Route&0x0f)-1) && (RX_pload[0]&0x0f)))
          {
            Decode_input();
            /*if(RX_pload[5] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }*/
            //P00 = ~P00;
            Check = 1;
          }
    }
    else if(SD == 2)
    {
        if(((Address_Route & 0xf0) && (RX_pload[6] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[3] & 0x0f)))
          {
            Decode_input();
            /*if(RX_pload[8] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }*/
            //if (TX_pload[7] == 0x00) P00 = ~P00;
            //if (TX_pload[8] == 0x01) P01 = ~P01;
            //P00 = ~P00;
            Check = 1;
          }
    }
    else if(SD == 3)
    {
        if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[15] & 0x0f)) || (((Address_Route & 0x0f)-2) == (RX_pload[3] & 0x0f
          {
            if(RX_pload[11] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }
          }
        if(RX_pload[15] == 0x00)
          {
          if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[12] & 0x0f)) || (((Address_Route & 0x0f)-2) == (RX_pload[3] & 0x
            ()//P02 = ~P02;
          }
    }
    else if(SD == 4)
    {
        if(((Address_Route & 0xf0) && (RX_pload[12] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[6] & 0x0f)))
          {
            if(RX_pload[14] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }
          }
    }
    else if(SD == 5)
    {
        if(((Address_Route & 0xf0) && (RX_pload[15] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[12] & 0x0f)))
          {
            if(RX_pload[17] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }
          }
    }
    else if(SD == 6)
    {
        if(((Address_Route & 0xf0) && (RX_pload[18] & 0xf0)) && ((((Address_Route & 0x0f)+2) == (RX_pload[24] & 0x0f)) || (((Address_Route & 0x0f)-3) == (RX_pload[6] & 0x0
          {
            if(RX_pload[20] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }
          }
        if(RX_pload[24] == 0x00)
          {
          if(((Address_Route & 0xf0) && (RX_pload[9] & 0xf0)) && ((((Address_Route & 0x0f)+1) == (RX_pload[21] & 0x0f)) || (((Address_Route & 0x0f)-3) == (RX_pload[2] & 0x
            ()//P02 = ~P02;
          }
    }
    else if(SD == 7)
    {
        if(((Address_Route & 0xf0) && (RX_pload[21] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[18] & 0x0f)))
          {
            if(RX_pload[23] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }
          }
    }
    else if(SD == 8)
    {
        if(((Address_Route & 0xf0) && (RX_pload[24] & 0xf0)) && (((Address_Route & 0x0f)-1) == (RX_pload[21] & 0x0f)))
          {
            if(RX_pload[26] == 0x01)
            { Delay_ms(100); P00 = ~P00; Check = 1; }
          }
    }
}
void RF_Mode() interrupt 9 using 1
{
    RF = 0;
    //P07 = ~P07;
    Rx_flag = 1;
    RF = 1;
}
```

```
void main(){

//int LOOP,TEST;
  RF = 1;
  EA = 1;

 SPIMCON0 = 0;
 S0CON = 0x50;            //Mode 1 8-bit UART.
 S0BUF = 0x00;
 S0RELH = 0x03;
 S0RELL = 0xF3;
 ADCON = 0x80;

 PCON = PCON| 0x80;
 SPIMCON0 = 0x55;

  nRF_CE = 0;
  nRF_CS = 1;
  P0DIR = 0x10;

 SPIMCON0 = 0;
 S0CON = 0x50;             //Mode 1 8-bit UART.
 S0BUF = 0x00;
 S0RELH = 0x03;
 S0RELL = 0xF3;
 ADCON = 0x80;

 PCON = PCON| 0x80;
 SPIMCON0 = 0x55;

  nRF_CE = 0;
  nRF_CS = 1;
  P0DIR = 0x10;
  P1DIR = 0xc0;          // for SPI P1DIR = 0xcf;

    P02 = 0;
    P07 = 0;
    P06 = 0;
    P05 = 0;
    P10 = 0;
    P12 = 0;
    P13 = 0;

  RFCKEN = 1;
  //Delay_ms(500);

    CE_Pin_Active(CE_LOW);                                  // Set CE pin low to enable stanby mode
    //Delay_ms(200);
    LO1_Flush_TX();
    LO1_Flush_RX();

    //P0 = 0x00;
    //Delay_ms(500);
    Rx_Mode();

    //TIO = 0;

    cs_on1;
    delay_nop
    delay_nop
    ui = 0x0040;
    ADE_Write(0x09,2,(unsigned char *)&ui);
    Delay_ms(200);
    ui = 0x000c;
    ADE_Write(0x09,2,(unsigned char *)&ui);
    uc = 0x00;
    ADE_Write(0x0f,1,(unsigned char *)&uc);
    uc = 0x00;
    ADE_Write(0x0d,1,(unsigned char *)&uc);
    uc = 0x00;
    ADE_Write(0x0e,1,(unsigned char *)&uc);
    cs_off1;
    delay_nop

    cs_on2;
    delay_nop
    delay_nop
    ui = 0x0040;
    ADE_Write(0x09,2,(unsigned char *)&ui);
    Delay_ms(200);
    ui = 0x000c;
    ADE_Write(0x09,2,(unsigned char *)&ui);
    uc = 0x00;
    ADE_Write(0x0f,1,(unsigned char *)&uc);
    uc = 0x00;
    ADE_Write(0x0d,1,(unsigned char *)&uc);
    uc = 0x00;
    ADE_Write(0x0e,1,(unsigned char *)&uc);
    cs_off2;
    delay_nop

    cs_on3;
    delay_nop
    delay_nop
    ui = 0x0040;
    ADE_Write(0x09,2,(unsigned char *)&ui);
    Delay_ms(200);
    ui = 0x000c;
    ADE_Write(0x09,2,(unsigned char *)&ui);
    uc = 0x00;
    ADE_Write(0x0f,1,(unsigned char *)&uc);
    uc = 0x00;
    ADE_Write(0x0d,1,(unsigned char *)&uc);
    uc = 0x00;
    ADE_Write(0x0e,1,(unsigned char *)&uc);
    cs_off3;
    delay_nop

   while(1)
  {
        if(Rx_flag == 1)
        {
            P02 = ~P02;
            Check_Device();
            Read_Data();


            if(Check == 1)
            {
                RF = 0;
                Decode_output();
                Tx_Mode();
                Tx_Real_Mode();
                Delay_ms(10);
                Check = 0;
                RF = 1;
            }
            Rx_Mode();
            Rx_flag = 0;
        }
  }
}
```