การประยุกต์ใช้ไมโครคอนโทรเลอร์ PIC16F877A ให้ทำหน้าที่เป็น PLC

PROGRAMMABLE LOGIC CONTROL (PLC) USING THE PIC16F877A

นายณัฐวุฒิ ชูศักดิ์

นางสาวพัชรี อึ้งเลิศตระกูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2548

การประยุกต์ใช้ไมโครคอนโทรเลอร์ PIC16F877A ให้ทำหน้าที่เป็น PLC

PROGRAMMABLE LOGIC CONTROL (PLC) USING THE PIC16F877A

โดย

นายณัฐวุฒิ ชูศักดิ์

นางสาวพัชรี อึ้งเฉิดพระกูล

อาจารย์ที่ปรึกษา

อ. ชนาธิป ชัยดิลกพัฒนกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมเครื่องกล

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชา วิศวกรรมเครื่องกล

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประยุกต์ใช้ไมโครคอนโทรเลอร์ PIC16F877A ให้ทำหน้าที่เป็น PLC

Programmable Logic Controller (PLC) Using the PIC16F877A

ผู้จัดทำ

1. นาย ณัฐวุฒิ ชูศักดิ์ รหัสประจำตัว 45015415
2. นางสาว พัชรี อึ้งเลิศตระกูล รหัสประจำตัว 44010772

อาจารย์ที่ปรึกษา

(อ. ชนาธิป ชัยดิลกพัฒนกุล)

การประยุกต์ใช้ไมโครคอนโทรเลอร์ PIC16F877Aให้ทำหน้าที่เป็น PLC

ณัฐวุฒิ ชูศักดิ์ 45015415
พัชรี อึ้งเลิศตระกูล 44010772
อ. ชนาธิป ชัยดิลกพัฒนกุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

PLC (Programmable Logic controller) เป็นอุปกรณ์ที่ใช้ควบคุมการทำงานของเครื่องจักรหรือระบบต่างๆ แทนวงจรรีเลย์แบบเก่าที่มีข้อเสียคือ การเดินสายและการเปลี่ยนแปลงเงื่อนไขในการควบคุมมีความยุ่งยาก ในปัจจุบัน PLC จึงเข้ามาทดแทนวงจรรีเลย์เพราะ PLC ใช้งานได้ง่ายกว่า ปริญญานิพนธ์นี้นำเสนอการใช้ประโยชน์จากไมโครคอนโทรเลอร์ และมีการอธิบายการทำงานของไมโครคอนโทรเลอร์ ปริญญานิพนธ์นี้เป็นตัวอย่างอันดีสำหรับ นักศึกษาเครื่องกลที่สนใจนำไมโครคอนโทรเลอร์ไปใช้งาน เนื่องจากมีการอธิบายพื้นฐานทางอีเล็กทรอนิกส์อย่างละเอียด ทำให้ผู้ไม่พื้นฐานมาก่อนสามารถทำความเข้าใจได้ มีการศึกษาวงจรส่วนต่างๆของ PLC อุตสาหกรรม ส่วนไหนที่ดีแล้วก็จะนำมาเผยแพร่ บางส่วนที่ผู้เขียนมีความคิดที่แตกต่างจากเดิม ก็จะนำเสนอความคิดในงานวิจัย และทำการทดสอบและสรุปเปรียบเทียบข้อดีเสียและความแตกต่าง

ในด้านซอฟท์แวร์ การสั่งงานไมโครคอนโทรเลอร์ให้ทำงานต้องสั่งการจากผู้ใช้ งานวิจัยนี้จึงมีการพัฒนาให้การสั่งงานจากผู้ใช้ทำได้ง่ายและรวดเร็วขึ้น และทดสอบแนวคิดด้วยการเขียนโปรแกรมขึ้นมาทดลองใช้งาน มีการวิจัยโค้ดคำสั่งเพื่อให้ไมโครคอนโทรเลอร์ทำงานแต่ละคำสั่งได้เร็วสุด และวัดเวลาทำงานออกมาออกมาศึกษา นำเสนอวิธีแก้ปัญหาทางกายภาพที่เกิดกับ PLC ด้วยซอฟท์แวร์ที่พัฒนาขึ้น วิธีนี้มีข้อดีที่ลดอุปกรณ์และต้นทุนลง และมีความยืดหยุ่นกว่า เพราะปรับให้เหมาะกับการใช้งานได้

งานวิจัยนี้สร้างให้อุปกรณ์ทำงานพื้นฐานได้เหมือน PLC แต่ไม่สามารถแทน PLC ได้ในทุกกรณี แต่ได้ระบุความบกพร่องหรือความสามารถที่ขาดไป ไว้เป็นแนวทางพัฒนาต่อ และยังแสดงให้เห็นแนวทางหรือกระบวนการคิดในการการสร้าง ฮาร์ดแวร์และซอฟต์แวร์ รวมทั้งแสดงการทดสอบใช้งานอุปกรณ์ที่สร้างขึ้นกับงานด้าน Pneumatic

Programmable Logic Controller (PLC) Using the PIC16F877A

Nuttavut Choosak

Patcharee Anglertragoon

Chanatip Chaidilokpattanakul Advisor

ABSTRACT

This paper presents the design of PLC using the PIC16F877A microcontroller. PLC is widely used in the industry to automatically control engineering process. This project is developed from scratch both software and electronic circuits. The main objective is to develop , PLC that can be used in education environment and costs much less than industrial PLC.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ อ. ชนาธิป ชัยดิลกพัฒนกุล อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจเอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

ณัฐวุฒิ ชูศักดิ์
พัชรี อึ้งเลิศตระกูล

สารบัญ

หน้าที่

สารบัญตาราง

หน้าที่

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

PLC (Programmable Logic controller)เป็นอุปกรณ์ที่คิดค้นขึ้นมาเพื่อใช้ควบคุมการทำงานของเครื่องจักรหรือระบบต่างๆ แทนวงจรรีเลย์แบบเก่า ซึ่งวงจรรีเลย์มีข้อเสียคือ การเดินสายและการเปลี่ยนแปลงเงื่อนไขในการควบคุมมีความยุ่งยาก และเมื่อใช้งานไปนานๆ หน้าสัมผัสของรีเลย์จะเสื่อม ดังนั้นในปัจจุบัน PLC จึงเข้ามาทดแทนวงจรรีเลย์เพราะ PLC ใช้งานได้ง่ายกว่า สามารถต่อเข้ากับอุปกรณ์อินพุทหรือเอาท์พุทได้โดยตรง หลังจากนั้นเขียนโปรแกรมควบคุมลงไปทำให้สามารถใช้งานได้ทันที ถ้าต้องการเปลี่ยนเงื่อนไขใหม่ สามารถทำได้โดยทำการเปลี่ยนแปลงโปรแกรมเพียงอย่างเดียว

ไมโครคอนโทรเลอร์ (Microcontroller) มาจากคำ 2 คำคือ ไมโคร (micro) หมายถึงขนาดเล็ก และคำว่าคอนโทรเลอร์ (controller) หมายถึงตัวควบคุมหรืออุปกรณ์ควบคุม ดังนั้น ไมโครคอนโทรเลอร์จึงหมายถึง อุปกรณ์ควบคุมขนาดเล็ก แต่ในตัวอุปกรณ์ควบคุมขนาดเล็กนี้ได้บรรจุความสามารถที่คล้ายคลึงกับระบบคอมพิวเตอร์ที่คนส่วนใหญ่คุ้นเคย กล่าวคือภายในไมโครคอนโทรเลอร์ได้รวมเอาซีพียู หน่วยความจำ และพอร์ต ซึ่งเป็นส่วนประกอบหลักของระบบคอมพิวเตอร์เข้าด้วยกัน โดยบรรจุรวมกันอยู่ภายใต้ตัวถังเดียวกัน คุณสมบัติที่ทำให้ไมโครคอนโทรเลอร์ได้รับความนิยมมาก 3 อย่างคือ 1. ไมโครคอนโทรเลอร์มีส่วนประกอบหลักของระบบคอมพิวเตอร์อยู่ใน IC (Integrated circuit) ตัวเดียวกันทั้งหมดทำให้มีขนาดเล็กมาก และใช้สะดวกโดยต่ออุปกรณ์เพิ่มเติมอีกเล็กน้อย 2. มีความยืดหยุ่นในการประยุกต์ใช้งานสูง เนื่องจากมีความสามารถของคอมพิวเตอร์ที่เปลี่ยนการทำงานตามซอฟต์แวร์ได้ ทำให้ใช้งานได้หลากหลายมาก ไม่ว่าใช้ในโทรศัพท์ เครื่องคิดเลข วิทยุ เครื่องพิมพ์ เครื่องอ่านบาร์โค้ด ฯลฯ แต่คุณสมบัติที่สำคัญที่สุดคือ 3. ราคาถูกเนื่องจากกระบวนการสร้าง IC ใช้แรงงานและมีกระบวนการน้อยกว่าการสร้างคอมพิวเตอร์แบบที่มีส่วนประกอบแยกกัน กระบวนการสร้าง IC ใช้เครื่องจักรอัตโนมัติเป็นส่วนใหญ่ และกระบวนการใช้เวลาน้อย เหตุผลดังกล่าวทำให้ ผลิตได้จำนวนมากๆ และต้นทุนต่ำ

เนื่องจาก PLC สร้างขึ้นมาเพื่อใช้ในอุตสาหกรรมเป็นหลัก จึงต้องการความทนทานและมีความแม่นยำสูง และมีความสามารถในการทำงานเฉพาะทางหลากหลายแบบรวมอยู่ด้วย ทำให้ PLC มีราคาที่สูงขึ้นตามไปด้วย อย่างไรก็ตามไมโครคอนโทรเลอร์มีความสามารถเกินพอที่จะนำมาใช้เลียนแบบการทำงานของ PLC เนื่องจากธุรกิจไมโครคอนโทรเลอร์ใหญ่กว่า PLC ไมโครคอนโทรเลอร์จึงมีการพัฒนาที่เร็วกว่า การใช้ไมโครคอนโทรเลอร์จึงมีความเร็วการทำงานสูงกว่า และราคาถูกกว่า PLC

PLC ต้องเขียนโปรแกรมควบคุม ภาษาที่ใช้เขียนให้โปรแกรม PLC โดยพื้นฐานเป็นภาษา Ladder Diagram ก่อนที่จะมี PLC ส่วนการควบคุมจะมีการใช้สวิตซ์และรีเลย์มาต่อกันเป็นวงจรควบคุม หากลำดับการทำงานซับซ้อน วงจรรีเลย์จะออกแบบได้ยากมาก ผู้ที่จะออกแบบจะต้องมีความเข้าใจเป็นอย่างดี และมีประสบการณ์มากพอสมควร ต่อมาเมื่อมีการผลิต PLC ขึ้นมาสมัยแรกๆ ผู้ออกแบบส่วนใหญ่ยังนิยมการออกแบบให้เป็นวงจรรีเลย์อย่างเดิมอยู่ จึงมีการสร้างภาษา Ladder Diagram ขึ้นมา โดยสามารถ

นำวงจรรีเลย์มาใส่โปรแกรมได้เลย จึงสามารถนำวงจรรีเลย์แบบเก่าๆ ที่เคยมีการออกแบบไว้ มาใช้กับ PLC ได้ทันที ทำให้การใช้โปรแกรม Ladder Diagram เป็นที่แพร่หลายยิ่งขึ้น แต่การจะใช้ภาษา Ladder Diagram จะต้องมีความเข้าใจและพื้นฐานวงจรรีเลย์เป็นอย่างดีเสียก่อน ซึ่งจะต้องใช้เวลา ตัวโปรแกรมมีกฎการโปรแกรมต่างๆ เช่น เอาท์พุตต้องอยู่ทางขวา, ทำงานจากซ้ายไปขวา, ต้องปิดโปรแกรมด้วยวงจรEnd เป็นต้น ข้อจำกัดเหล่านี้ทำให้การใช้ PLC จำกัดวงอยู่ในกลุ่มผู้ใช้ที่มีความรู้ทางด้านไฟฟ้าเป็นอย่างดีเท่านั้น บ่อยครั้งที่งานจริงหรือโจทย์จะให้มาเป็นลำดับการทำงานหรือเรียกว่า Sequential Diagram โดยพื้นฐานโปรแกรม Ladder Diagram ต้องทำงานสองต่อคือนำ Sequential Diagram ไปออกแบบวงจรรีเลย์ แล้วจึงนำไปลงโปรแกรมให้กับ PLC

ปริญญานิพนธ์นี้ลองนำเสนอวิธีการโปรแกรม PLC ที่ไม่ต้องคิดวงจรรีเลย์ให้เสียเวลาก่อนนำไปโปรแกรมให้กับ PLC ผู้เขียนเห็นว่ารูปแบบการโปรแกรมที่นำเสนอทำให้การโปรแกรมทำได้ง่ายและเร็วขึ้น อย่างไรก็ตามภาษา Ladder Diagram ได้รับพัฒนายาวนาน จนครอบคลุมงานได้ทุกประเภท และรองรับงานเฉพาะทางได้หลายชนิด ในขณะที่วิธีการโปรแกรมที่คิดมาใหม่ยังไม่อาจครอบคลุมงานได้ทุกประเภท แต่ในเบื้องต้นนี้ สามารถโปรแกรมได้โดยไม่ต้องเสียเวลาคิดวงจรรีเลย์ ทำให้เร็วและง่ายขึ้นได้

จุดสำคัญอีกอย่างของปริญญานิพนธ์นี้คือการนำเสนอการใช้ประโยชน์จากไมโครคอนโทรเลอร์ และมีการอธิบายการทำงานของไมโครคอนโทรเลอร์ และการใช้งานอย่างละเอียด ไมโครคอนโทรเลอร์มีประโยชน์อย่างมากกับงานทางเครื่องกล แต่ไม่มีวิชาด้านไมโครคอนโทรเลอร์ในสาขาเครื่องกล เนื่องจากอยู่คนละสาขากัน ปริญญานิพนธ์นี้เป็นตัวอย่างอันดีสำหรับนักศึกษาเครื่องกล ที่สนใจนำไมโครคอนโทร-เลอร์ไปใช้งาน เนื่องจากปริญญานิพนธ์นี้เขียนโดยนักศึกษาเครื่องกล ดังนั้นจึงมีการอธิบายพื้นฐานทางอีเล็กทรอนิกส์อย่างละเอียด ทำให้ผู้ไม่พื้นฐานมาก่อนสามารถทำความเข้าใจได้

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 นำไมโครคอนโทรเลอร์มาสร้าง PLC แบบพื้นฐานที่มีราคาถูกกว่าและทำงานได้เร็วกว่าเพื่อผู้สนใจนำไปใช้ควบคุมเครื่องจักรหรืออุปกรณ์ไฟฟ้าทั่วไป และแสดงวิธีการสร้างPLCจากไมโครคอน-โทรเลอร์ เพื่อเป็นพื้นฐานให้ผู้อื่นสามารถนำไปพัฒนาให้เป็น PLC ที่มีประโยชน์ใช้สอยได้มากยิ่งขึ้น

1.2.2 สร้าง Software ที่ใช้สำหรับโปรแกรม PLC ที่สร้างขึ้นจากไมโครคอนโทรเลอร์ และนำเสนอวิธีการควบคุมไมโครคอนโทรเลอร์ ที่ไม่ต้องออกแบบวงจรรีเลย์ก่อนนำไปลงโปรแกรมให้กับ PLC เพื่อให้เป็นทางเลือกในการโปรแกรม PLC ที่ง่ายและรวดเร็วขึ้น

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะสร้างPLCโดยใช้ไมโครคอนโทรเลอร์ เบอร์ PIC16F877A ของบริษัท Microchip ซึ่งต้องสร้างวงจรแวดล้อมเพื่อให้ไมโครคอนโทรเลอร์ทำงานเป็น PLC ได้ องค์ประกอบหลักของ PLC เบื้องต้นประกอบด้วย ส่วนเอาท์พุต, ส่วนPower Supply, ส่วนอินพุต, ส่วน CPU, ส่วนแสดงผล, ส่วนรับการโปรแกรม จึงต้องทำการศึกษาวงจรส่วนต่างๆของ PLC อุตสาหกรรม ส่วนไหนที่ดีแล้วก็จะนำมาเผยแพร่ บางส่วนที่ผู้เขียนมีความคิดที่แตกต่างจากเดิม ก็จะนำเสนอความคิดในงานวิจัย และทำการทดสอบและสรุปเปรียบเทียบข้อดีเสียและความแตกต่าง

ในด้านซอฟท์แวร์ การสั่งงานไมโครคอนโทรเลอร์ให้ทำงานต้องสั่งการจากผู้ใช้ งานวิจัยนี้จึงมีการพัฒนาให้การสั่งงานจากผู้ใช้ทำได้ง่ายและรวดเร็วขึ้น เพื่อทดสอบแนวคิดจึงต้องเขียนโปรแกรมขึ้นมาทดลองใช้งาน โปรแกรมสั่งงานหลังผู้ใช้สั่งเสร็จแล้วจะหมดหน้าที่ไปแต่โค้ดที่อยู่ในไมโครคอนโทรเลอร์จะถูกเรียกใช้ตลอดเวลาที่ PLC ทำงาน จึงมีการวิจัยโค้ดในส่วนนี้เพื่อให้ไมโครคอนโทรเลอร์ทำงานแต่ละคำสั่งได้เร็วสุด และวัดเวลาทำงานออกมาออกมายืนยัน ปัญหาทางกายภาพบางอย่างสามารถแก้ได้ด้วยซอฟท์แวร์ที่พัฒนาขึ้น วิธีนี้มีข้อดีที่ลดอุปกรณ์และต้นทุนลง และมีความยืดหยุ่นกว่า เพราะปรับให้เหมาะกับการใช้งานได้

งานวิจัยนี้เริ่มแรกสร้างให้อุปกรณ์ทำงานพื้นฐานได้เหมือน PLC แต่ไม่สามารถแทน PLC ได้ในทุกกรณี แต่จะระบุความบกพร่องหรือความสามารถที่ขาดไปไว้เป็นแนวทางพัฒนาต่อ และไม่อาจทดลองยืนยันอายุการใช้งานของอุปกรณ์และความน่าเชื่อถือ (reliability) ของอุปกรณ์ได้ เพราะใช้ประมาณสูง

1.4 วิธีการดำเนินงาน

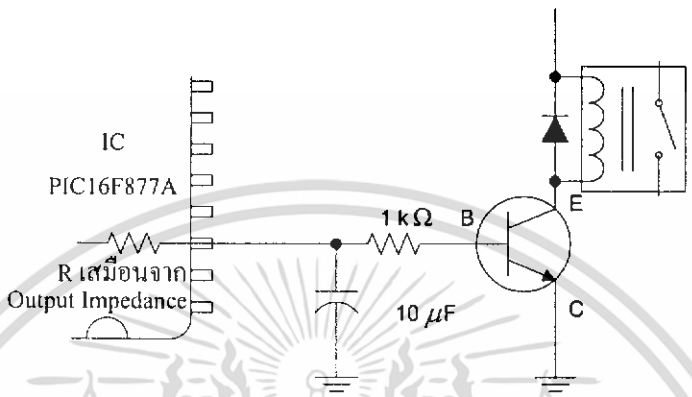
งานวิจัยในโครงงานนี้จะเริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องกับงานวิจัย ซึ่งก็มีเรื่องหลัก ๆ อยู่ 4 เรื่องด้วยกัน คือ ความรู้พื้นฐานเกี่ยวกับ PLC ทฤษฎีพื้นฐานทางอีเล็กทรอนิกส์ ทฤษฎีพื้นฐานเกี่ยวกับคอมพิวเตอร์ การเขียนโปรแกรมสมัยใหม่ ความรู้ด้านไมโครคอนโทรเลอร์ สำรวจราคาคุณสมบัติความสามารถอุปกรณ์ต่าง เปรียบเทียบและเลือกอุปกรณ์ที่จะใช้ และออกแบบวงจรส่วนต่างๆ สร้างและทดสอบทีละส่วนย่อย ซึ่งมีรายละเอียดในบทที่ 2

ออกแบบโค้ดคำสั่งของไมโครคอนโทรเลอร์ ทดลองป้อนให้ฮาร์ดแวร์ ทดสอบการตอบสนองและการทำงานระหว่างโค้ดและอุปกรณ์ รวบรวมเป็นโค้ดคำสั่ง คิด Algorithm ที่จะแปลงการติดต่อจากผู้ใช้ไปเป็นโค้ดคำสั่ง ทดสอบแนวคิดการแปลง จนได้ขั้นตอน Algorithm ที่แน่นอน ออกแบบส่วนติดต่อผู้ใช้ รวมทุกส่วนเข้าด้วยกันเป็นโปรแกรมซึ่งมีรายละเอียดในบทที่ 3 เมื่อได้องค์ประกอบหลักของทุกส่วนแล้ว

เข้าสู่การทดสอบการใช้งาน ดูปัญหาที่เกิดขึ้นจริง ข้อบกพร่อง แก้ปัญหาที่มีและหาวิธีพัฒนาให้ดีขึ้น ทดสอบหรือคำนวณSpecificationของ PLC ที่สร้าง คิดวิธีทดสอบและทดลองใช้งานจริง แสดงรายละเอียดในบทที่ 4 ต่อมาในบทที่ 5 ซึ่งเป็นบทสุดท้ายก็จะเป็นการสรุปการสรุปรายละเอียด (Specification) ของ PLC ที่สร้างปัญหาที่พบและวิธีการแก้ไข ผลลัพธ์แนวทางการพัฒนาต่อและแนวทางการในการนำประยุกต์ใช้

บทที่ 2
การออกแบบฮาร์ดแวร์

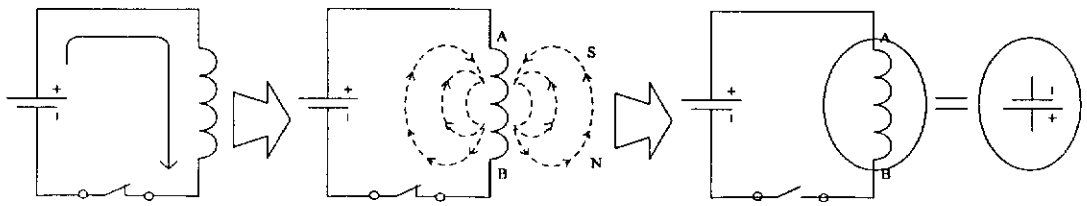
2.1 การทำงานของส่วนเอาท์พุต



รูป 2-1 วงจรภาคเอาท์พุต

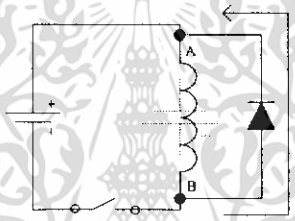
เมื่อ Microcontroller PIC16F877A ส่งไฟออกมาที่ขั้ว C0 (ขา 26)ด้วยแรงดัน 5V เกิดกระแสผ่านความต้านทาน R ซึ่งเกิดจาก Output Impedance (200 Ω) รวมกับ R 1 kΩ ที่ต่ออยู่กับขา C0 จึงมีกระแส 4.1667 mA (I = E/R = 5V/ (1000 Ω + 200 Ω) = 4.1667 mA) นี้ไหลเข้าขา B (Base) ของทรานซิสเตอร์ MPSA 42 (MPSA 42 เป็นทรานซิสเตอร์ชนิด NPN) เมื่อมีกระแสไหลผ่านขา Base ไปขา E(Emitter) ทรานซิสเตอร์จะยอมให้กระแสอีกจำนวนหนึ่งผ่านจากขา C (Collector) ไปยังขา Emitter ได้ โดยทรานซิสเตอร์จะมีอัตราขยายกระแสอยู่ ซึ่งยอมให้กระแสผ่านขา Collector ไหลไปได้มากกว่าขา Base หลายเท่า อัตราขยายกระแสนี้เรียกว่า $h_{fe} = I_C / I_E$

ทรานซิสเตอร์ MPSA 42 มีอัตราขยายกระแสประมาณ 82 เท่า เมื่อมีกระแส 4.1667 mA ผ่านเข้าขา Base จึงมีกระแส 4.1667 x 82 = 341.7 mA ไหลผ่านขา Collector กระแสนี้จะทำให้ Relay ทำงาน ต่อมาเมื่อ Controller หยุดจ่ายไฟ ทรานซิสเตอร์ก็หยุดจ่ายกระแส Relay ที่เป็นขดลวดแม่เหล็กไฟฟ้าเมื่อตอนมีกระแสเข้า กระแสผ่านขดลวดของ Relay สร้างสนามแม่เหล็กดูดให้หน้าสัมผัสไฟฟ้าติดกัน เมื่อหยุดจ่ายไฟ สนามแม่เหล็กจะยุบตัวตัดขดลวด (เหนี่ยวนำตัวเอง) เกิดแรงดันไฟฟ้าสูง (กระแสต่ำ) หากแรงดันเหนี่ยวนำนี้มากกว่าแรงดันทำงานของทรานซิสเตอร์ ก็จะทำให้ทรานซิสเตอร์เสียหายได้ (ทรานซิสเตอร์ MPSA 42 ทนแรงดันได้ประมาณ 100 V) แต่มีวิธีป้องกันทรานซิสเตอร์จากแรงดันเหนี่ยวนำนี้ โดยพิจารณาวงจรดังรูปที่ 2-2



รูปที่ 2-2 ทิศทางแรงดันของการเหนี่ยวนำตนเอง

เมื่อสวิตช์ปิด กระแสจะไหลผ่านขดลวดโดยวิ่งจาก A ไป B เกิดสนามแม่เหล็กมีทิศทางดังรูป เมื่อตัดสวิตช์ สนามแม่เหล็กจะยุบตัวตัดขดลวด ในทิศทางที่ทำให้เกิดแรงดันที่ขั้ว B เป็นบวก และขั้ว A เป็นลบ แรงดันนี้มีความต่างศักย์สูงกว่าแรงดันปกติและจะวิ่งจากขั้ว B ไปขั้ว A และอาจทำลายอุปกรณ์ใดๆ ที่ขวางการไหลของกระแส วิธีป้องกันคือให้ใส่ไดโอดลงไปในทิศทางดังรูปที่ 2-3

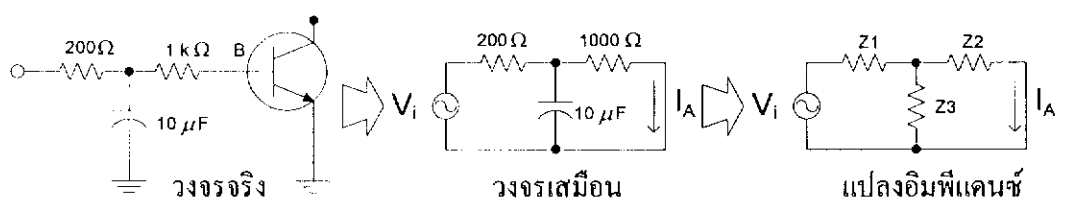


รูปที่ 2-3 การต่อไดโอด bypass แรงดัน

โดยที่ไดโอดจะบายพาสกระแสจากขั้ว B ไปยังขั้ว A โดยไม่ก่อให้เกิดความเสียหายกับอุปกรณ์อื่นๆ (ไดโอดเป็นอุปกรณ์สารกึ่งตัวนำที่ยอมให้ไฟฟ้าไหลผ่านได้ทางเดียว เช่นเดียวกับ เช็ควาล์วหรือวาล์วกันกลับ)

2.1.1 การตอบสนองทางความถี่

ต่อมา เป็นการวิเคราะห์ผลการลดทอนความถี่ที่เกิดจากตัวเก็บประจุ 10 μF ที่ต่อกับขั้ว C0 ว่ามีผลอย่างไรต่อการทำงานของรีเลย์ โดยจะวิเคราะห์จากกระแสที่ผ่านขั้ว Base ของทรานซิสเตอร์



รูปที่ 2-4 การแปลงอิมพีแดนซ์ (Impedance)

โดย Z_1 เป็นอิมพิแดนซ์ของ $R_1 = 200$

Z_2 เป็นอิมพิแดนซ์ของ $R_2 = 1000$

Z_3 เป็นอิมพิแดนซ์ของ $C_1 = 10^5 / S$

$$Z_{total} = Z_1 + (Z_2 Z_3 / Z_2 + Z_3)$$

Transfer function = output/input = I_A / V_i

Transfer function = $[1 - (Z_1 / Z_{total})] \times 1 / Z_2$

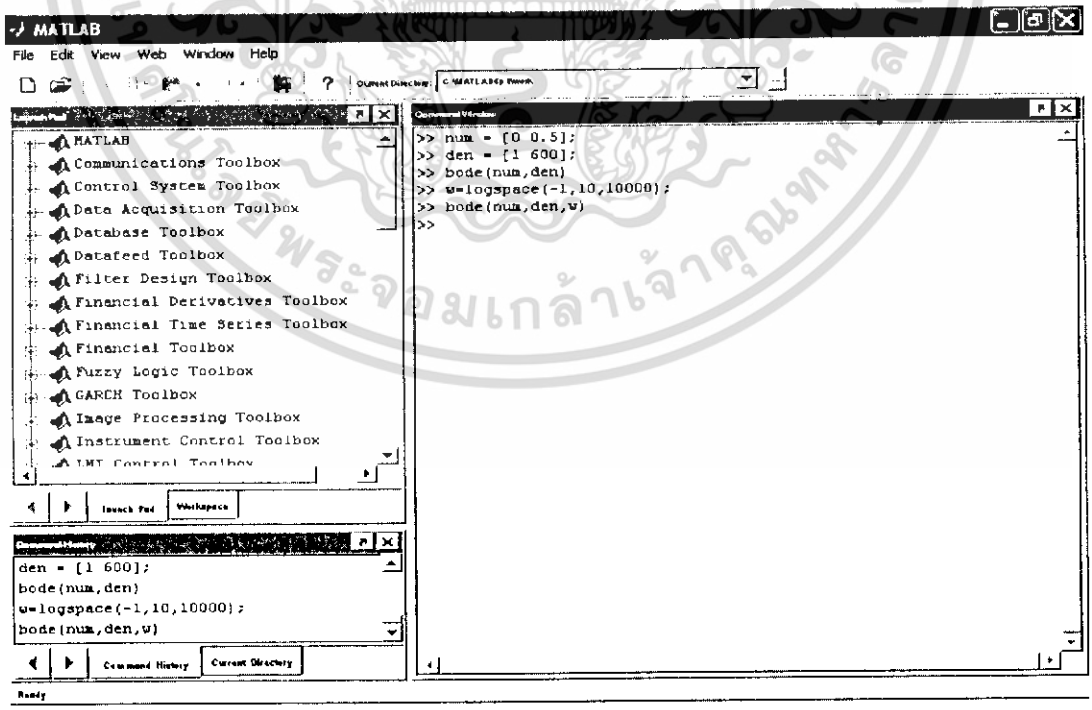
$$= Z_3 / (Z_1 Z_2 + Z_1 Z_3 + Z_2 Z_3)$$

(แทนค่าอิมพีแดนซ์) $= (10^5 / S) / [(2 \times 10^5) + (2 \times 10^7 / S) + (10^8 / S)]$

$$= 0.5 / (S+600)$$

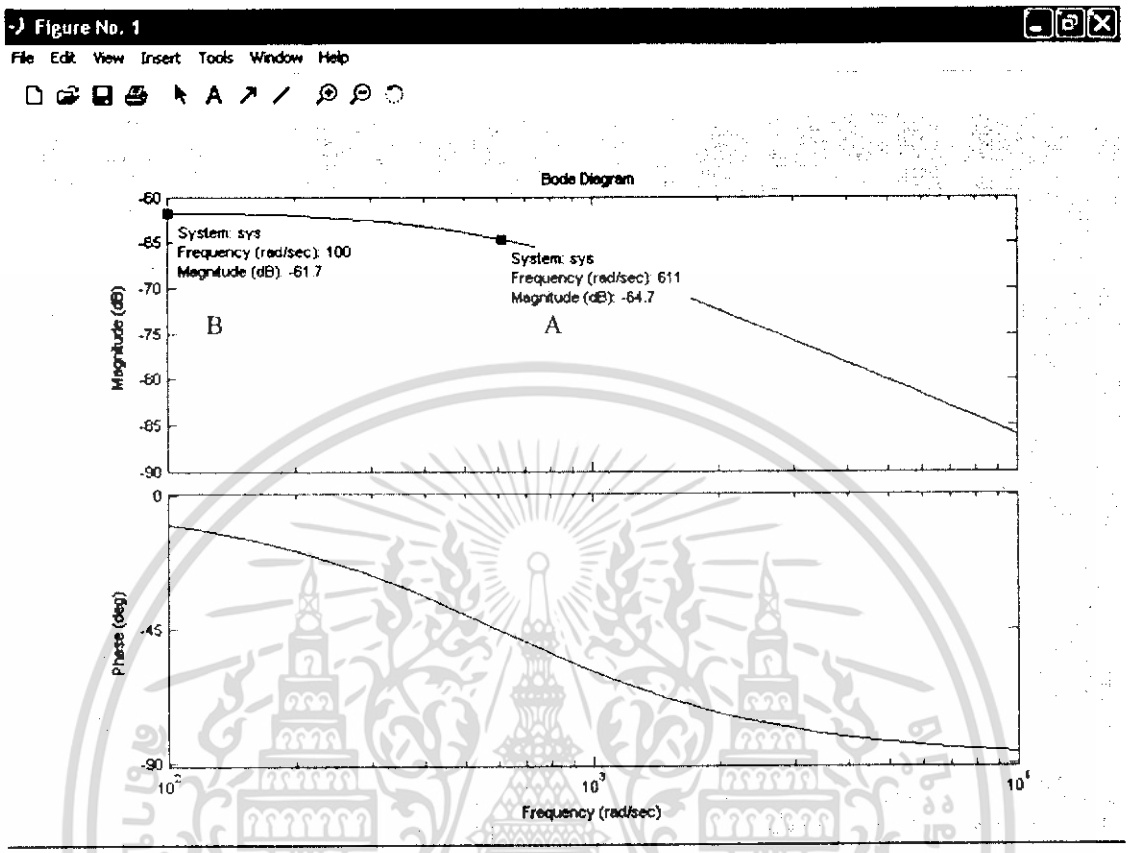
นำไปหาผลตอบสนองทางความถี่ใน Math lab ด้วยวิธีการตามนี้ (ดูรูป 2-5 ประกอบ)

```
NUM = [ 0  0.5 ] ;
DEN = [ 1  600 ] ;
bode = ( NUM , DEN )
w = logspace ( -1 , 10 , 10000) ;
bode = ( NUM , DEN , w )
```



รูปที่ 2-5 การหาผลตอบสนองทางความถี่

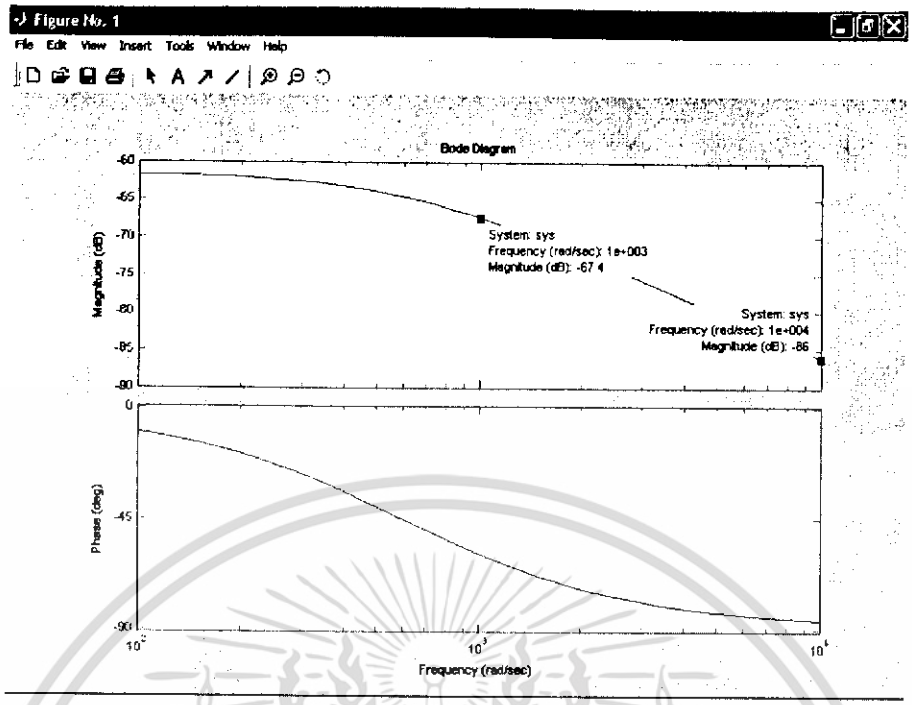
ได้ผลลัพธ์เป็นกราฟออกมาตามนี้



รูปที่ 2-6 กราฟที่ใช้หา Cut off frequency

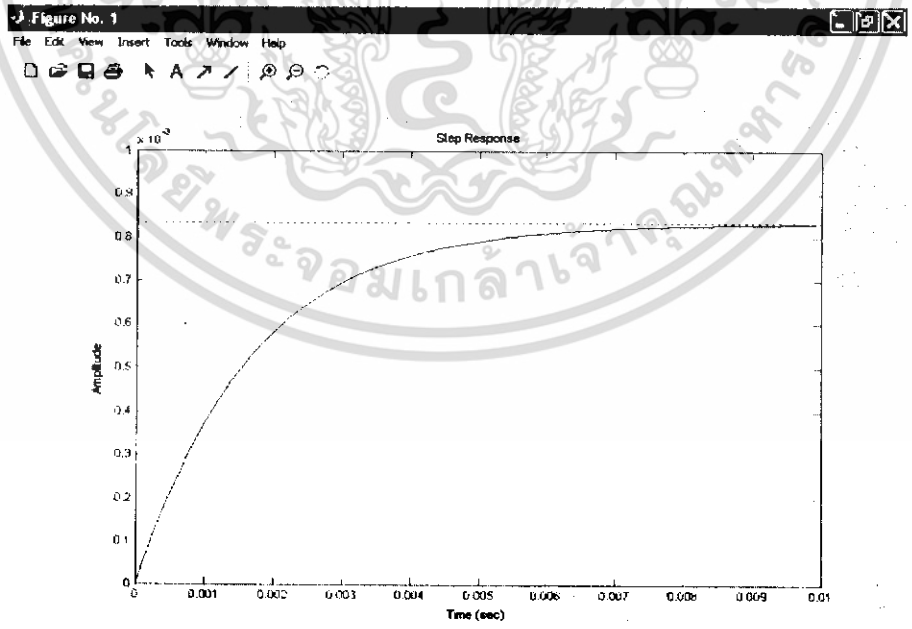
ตามนิยาม Cut off frequency เป็นความถี่ ณ. จุดที่แมกนีจูด (Magnitude) ถูกลดทอนไป -3 dB ที่จุดบนเส้นกราฟในรูป 2-6 จุด A เป็นจุด Cut off frequency เพราะตามรูปช่วงความถี่จาก 0 – 100 rad/s แมกนีจูดเป็น -61.7 dB แทบคงที่(จึงตัดกราฟช่วงต่ำกว่า 100 rad/s ไปเพื่อให้รูปแสดงเฉพาะช่วงที่สำคัญ) การที่แม้ความถี่เป็น 0 (ไฟตรง) ยังเกิดการลดทอนไป -61.7 dB เพราะตัวต้านทานในวงจร หลังจากนั้นตามรูป เมื่อความถี่สูงขึ้นถึง 611 rad/s การลดทอนเพิ่มเป็น -64.7 dB แสดงว่าจุดนี้ผลของตัวเก็บประจุทำให้(ไฟสลับ)เกิด การลดทอนเปลี่ยนไป (-64.7) – (-61.7) = -3 dB จึงแสดงว่าความถี่ Cut off frequency เท่ากับ 611 rad/s = 97.24 Hz ≈ 100 Hz

ต่อมาหาอัตราลดทอนพิจารณารูป 2-7 (ที่มาในการสร้างเหมือนรูป 2-6 แต่เลื่อนจุดไปตำแหน่งที่ต้องการ) จากรูปจะเห็นว่าเมื่อความถี่เปลี่ยนไปจาก 1000 rad/s ไปเป็น 10000 rad/s (เปลี่ยนไป 1 decade) แมกนีจูดเปลี่ยนไปด้วยจาก -67.4 dB เป็น -86 dB (เปลี่ยนไป -18.6 dB) จึงสรุปได้ว่า วงจรมีอัตราลดทอน -18.6 dB / decade = -5.58 dB / octave ≈ -6 dB / octave นั่นเอง (-20dB / decade = -6 dB / octave)



รูปที่ 2-7 กราฟที่ใช้หาอัตราลดทอน

สรุปแล้ววงจรที่เราออกแบบลดทอนความถี่ต่ำกว่า 100 Hz ด้วยอัตรา -6 dB / octave ต่อไปเป็นการหาอัตราการตอบสนอง (ความเฉื่อย) ทดสอบโดยการป้อน Step function (ไฟตรง, DC) ให้กับ Transfer function เพื่อดูผลตอบสนองใน Math lab ได้ผลตอบสนองเป็นกราฟดังรูป 2-8

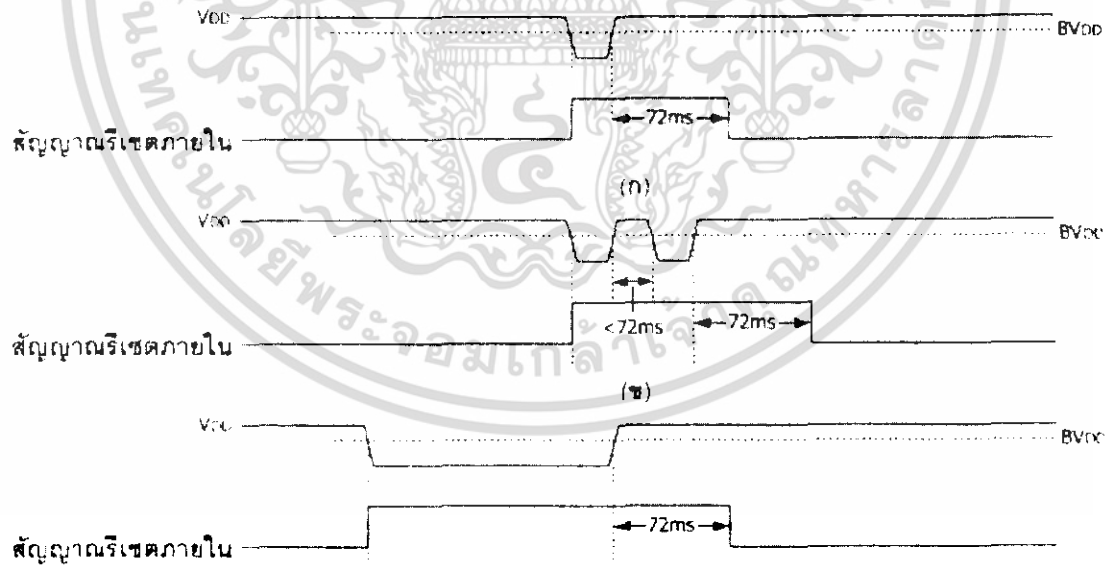


รูปที่ 2-8 ผลการตอบสนองต่อไฟตรง

2.2 การทำงานของส่วน Power Supply

ภาคจ่ายไฟได้มีการออกแบบให้ส่วน CPU (ไมโครคอนโทรเลอร์) ทำงานต่อไปได้ชั่วระยะเวลาหนึ่งเมื่อไฟดับ โดยใช้ตัวเก็บประจุสำรองไฟ ธรรมชาติของตัวเก็บประจุ เมื่อมีการจ่ายกระแสออกมาแรงดันจะลดลงเรื่อยๆ เมื่อแรงดันต่ำลงมากๆ CPU จะเริ่มทำงานผิดพลาด แต่ก่อนที่ CPU จะทำงานผิดพลาดภายใน PIC 16F877A มีวงจรตรวจจับระดับแรงดันไฟเลี้ยงต่ำกว่าที่กำหนดหรือเรียกว่า บราวเอาต์ดีเท็ก (Brown-out detect : BOD) หากพบว่าไฟเลี้ยงของไมโครคอนโทรลเลอร์ลดต่ำลงถึงจุดที่กำหนด ก็จะกำเนิดสัญญาณรีเซตภายในส่งไปยังซีพียูเพื่อเริ่มต้นการทำงานใหม่ ถ้าหากไฟเลี้ยงกลับมาอยู่ในระดับปกติ ระบบก็จะสามารถปฏิบัติงานต่อไปได้อย่างไม่ติดขัด การใช้ BOD จะช่วยแก้ปัญหาไมโครคอนโทรลเลอร์ทำงานผิดพลาดหรือหยุดทำงานอันเนื่องมาจากความไม่สม่ำเสมอของไฟเลี้ยง

สำหรับใน PIC 16F877A ระดับแรงดันที่ทำให้วงจร BOD ทำงานคือ 3.7 V ในการทำงานจริงเมื่อแรงดันลดลงถึง 3.7 V เป็นเวลามากกว่า 100 ไมโครวินาที วงจร BOD จะทำงานเพื่อสร้างสัญญาณรีเซต ในรูปที่ 2-9 แสดงไดอะแกรมเวลาการเกิดสัญญาณรีเซตเนื่องจากการทำงานของวงจร BOD จากรูปไมโครคอนโทรลเลอร์ยังคงอยู่ในสภาวะรีเซตจนกว่าระดับไฟเลี้ยงจะกลับมาอยู่ในระดับต่ำสุดที่สามารถทำงาน ซึ่งทำให้เพาเวอร์อัปไทเมอร์ (PWRT) ทำงาน หน่วงเวลา 72 มิลลิวินาที ถ้าหากในช่วงเวลานั้นทำไฟเลี้ยงเกิดลดต่ำลงจนวงจร BOD ทำงาน ไมโครคอนโทรลเลอร์ก็จะกลับไปสู่สภาวะรีเซตอีกครั้ง รอคอยไฟเลี้ยงให้กลับมาอยู่ในระดับที่สามารถทำงานได้ นั่นคือเริ่มต้นกระบวนการของเพาเวอร์อัปไทเมอร์ใหม่อีกครั้ง จนกระทั่งไฟเลี้ยงคงที่ ไมโครคอนโทรลเลอร์ก็จะกลับมาทำงานตามปกติ



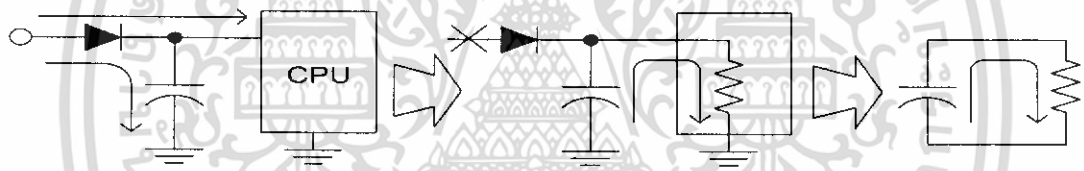
รูปที่ 2-9 ไดอะแกรมแสดงการเกิดบราวเอาต์รีเซต

เหตุจำเป็นที่ต้องมีการรีเซ็ตนี้ เพราะเมื่อบางครั้งไฟดับชั่วขณะ ไมโครคอนโทรลเลอร์ทำงานผิดพลาด (แต่ไม่ดับไปเลย) CPU อาจไปวนอยู่ในบางคำสั่งตลอดเวลา เนื่องจากคำสั่งที่ใช้เช็คเงื่อนไขอาจผิดพลาดไปแล้วจากไฟดับ ผู้ใช้จะเห็น CPU ค้างอยู่อย่างนั้นไปตลอด (หรือเรียกว่าคอมแฮงก์) สำหรับ PC ผู้ใช้จะกดรีเซ็ตให้แต่ไมโครคอนโทรลเลอร์ที่นำไปใช้เป็น PLC ซึ่งต้องทำงานอัตโนมัติจึงไม่มีคนรีเซ็ตให้ หรือหากว่ามี PLC มีจำนวนมากๆ ในโรงงานหรืออยู่ในที่เข้าถึงยาก จึงจำเป็นที่ PLC ต้องรีเซ็ตตัวเองได้เมื่อไฟตกหรือดับ

สรุป เมื่อไฟตกไมโครคอนโทรลเลอร์จะรีเซ็ตตัวเองก่อนที่จะทำงานผิดพลาด เพื่อให้ PLC ทำงานต่อไปได้อย่างปกติ จะต้องรักษาแรงดันไฟเลี้ยงไว้ให้มากกว่าแรงดันบราวเอาท์ดีเท็ค (3.7 V)

ต่อมาไมโครคอนโทรลเลอร์จะกินกระแสไฟสูงสุดเมื่อขับ load ทุก output พร้อมกัน (PLC มี 8 output) ขับกระแสสูงสุดได้ output ละ 25 mA (8 output = 2 A) จากการวัดจริงที่ไฟเลี้ยง 5V ความถี่ CPU 20 MHz ขับโหลดเต็มที่ทุกช่อง รวมทั้งบอร์ด CPU ที่สร้างขึ้นนั้น กินกระแสประมาณ 230 mA

สรุป เงื่อนไขคือ ต้องมีไฟเลี้ยงปกติ 5V แรงดันไม่ต่ำกว่า 3.7 V กินกระแสไฟ 2.3 A. ตลอดเวลา (ภายใต้เงื่อนไขนี้ สามารถมองได้ว่า ส่วน CPU ทั้งบอร์ดเป็นความต้านทาน R = E/I = 5 / 0.23 = 21.74 Ω จากวงจรจริง ซึ่งสามารถพิจารณาให้ง่ายลงได้ตามรูป 2-10



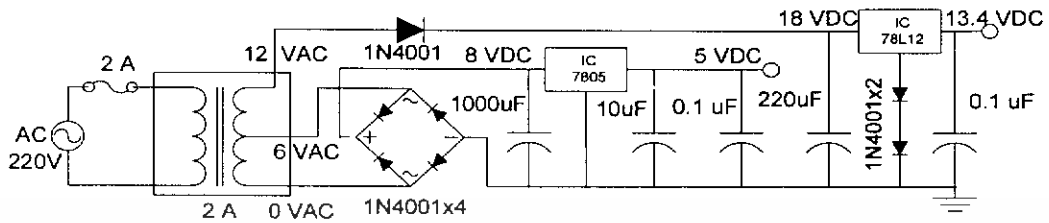
รูปที่ 2-10 การพิจารณากระแสโหลด

เมื่อไฟเลี้ยงเข้า V_{in} ตามปกติกระแสนั้นจะไหลสองทาง คือ เข้า CPU ตามปกติ กับชาร์จเข้าตัวเก็บประจุเมื่อมีประจุเต็มแล้ว กระแสก็จะไม่ไหลผ่านตัวเก็บประจุ หลังจากนั้นเมื่อไฟเลี้ยงจาก V_{in} หายไปตัวเก็บประจุจะจ่ายไฟออกมาทดแทน (เหมือนกับถังลมที่สำรองลมไว้เมื่อคอมเพรสเซอร์หยุดทำงาน ถังลมจะทำการจ่ายลมออกมา) ไดโอดในรูปที่ มีไว้กันกระแสไหลกลับไป V_{in} อย่างไรก็ตาม เมื่อไฟไหลผ่านไดโอดจะเกิดแรงดันตกคร่อมไดโอดอยู่ 0.7 V อยู่เสมอ (เป็นธรรมชาติของสารกึ่งตัวนำ เช่นเดียวกับ วาล์วกันกลับที่จะเกิดความดันตกสูญเสีย (drop pressure) เป็นความดันที่ต้องใช้ยกวาล์วหรือสปริงในเช็ควาล์ว)

สรุป เมื่อไฟไหลผ่านไดโอดแรงดันจะหายไป 0.7 V ถ้าต้องการให้ไฟเลี้ยง CPU เป็น 5 V ต้องจ่ายไฟเข้า V_{in} 5.7 V ซึ่งการที่ต้องจ่ายแรงดัน 5.7 V นี้จะเป็นผลต่อเนื่องไปถึงภาคจ่ายไฟต่อไป

ดังรูปที่ 2-10 Simple Circuit ซึ่งมีความต้านทานกับตัวเก็บประจุต่อกันเป็นวงจร RC เมื่อแก้สมการภายใต้เงื่อนไข t (time) = 0, V_0 = 5 V จะได้คำตอบเป็น $V(t) = 5e^{-t/21.7C}$ แรงดันสุดท้ายที่ CPU ทำงานได้ตามปกติคือ 3.7 V จากรายระเอียดทางเทคนิคของ PLC OMROM CPM1A เมื่อแหล่งจ่ายแรงดันหยุดชั่วขณะ PLC OMROMจะทนได้ 10 ms ซึ่งในอุปกรณ์ของเรานี้ได้ออกแบบให้ทนได้ที่ 50 ms ที่ V =

3.7 V , t = 0.055 s ได้ขนาดของ C ออกมาเท่ากับ 7641.74 μF ค่าใกล้เคียงของตัวเก็บประจุที่มีขาย คือ 6,800 μF กับ 10,000 μF เลือก 10,000 μF มาใช้ ได้จะทนเวลาได้ t = 65.46 ms

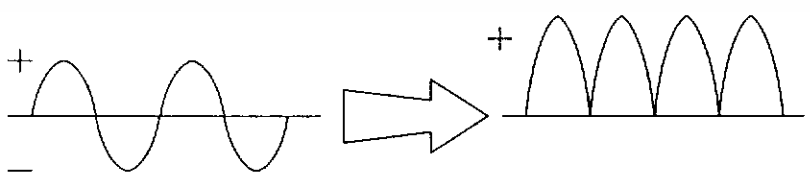


รูปที่ 2-11 วงจร Power supply และ Voltage Regulator

ต่อมาเป็นการทำงานของภาค Power supply และ Voltage Regulator จากไฟ 220 VAC เข้าหม้อแปลงขนาด 2A ลดแรงดันแบ่งเป็น 12 VAC กับ 6 VAC ไฟ 6 VAC จะไหลผ่านไดโอด 4 ตัวที่ต่อกันแบบบริดจ์ เป็นวงจรแปลงจากไฟกระแสสลับเป็นไฟกระแสตรงแบบเต็มคลื่น ดังรูปที่ 2-12

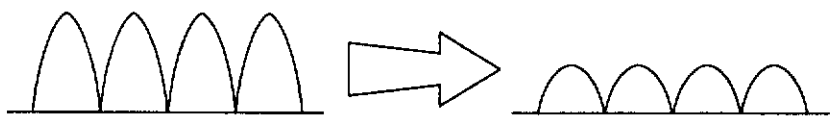
รูปที่ 2-12 เป็นวงจรแปลงจากไฟกระแสสลับเป็นไฟกระแสตรงแบบเต็มคลื่น

ไฟกระแสสลับจะมีแรงดัน 220 V. และแรงดันที่เปลี่ยนตามเวลาเป็นคลื่น Sine ซึ่งมีความถี่ 50 Hz เปลี่ยนขั้วสลับไปมาตามเวลา แรงดัน 220 V. คือแรงดันที่เรียกว่า แรงดัน RMS (Root mean square) เนื่องจากไฟกระแสสลับมีแอมพลิจูดไม่คงที่ (ตามรูปแบบฟังก์ชัน sine) แรงดัน RMS คือ แรงดันที่ได้จากการเฉลี่ยไฟกระแสสลับที่ไม่คงที่ให้เป็นไฟกระแสตรง โดยการยกกำลังสอง เพื่อให้เป็นค่าของไฟกระแสตรง ดังรูปที่ 2-13



รูปที่ 2-13 การแปลงไฟกระแสสลับเป็นไฟกระแสตรงโดยการยกกำลังสอง

แล้วถอดรากที่สองเพื่อให้ค่ากลับเป็นเท่าเดิม ดังรูปที่ 2-14

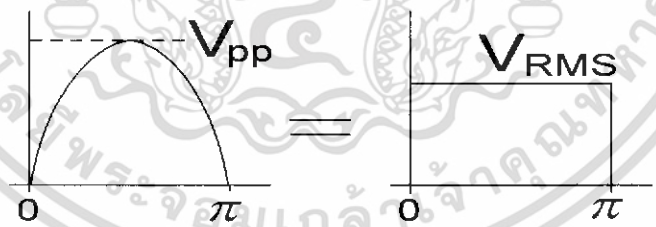


รูปที่ 2-14 การแปลงไฟกระแสสลับโดยการยกกำลังสองแล้วถอดรูท

จากนั้นแรงดันเฉลี่ยที่ได้จะเป็นไฟกระแสตรง แน่นอนว่าแรงดันเฉลี่ยนี้ต้องมีค่าลดลง แรงดันจากการเฉลี่ยนี้คือแรงดัน RMSนั่นเอง ดังรูปที่ 2-15

รูปที่ 2-15 ขนาดแรงดัน RMS จากการเฉลี่ยไฟสลับ

(Vpp คือ Volt peak หรือแรงดันยอดคลื่น Sine) พิจารณาการเฉลี่ยยอดคลื่น Sine ไปเป็นไฟกระแสตรง ดังรูปที่ 2-15



รูปที่ 2-16 ขนาดพื้นที่รูปซ้ายต้องเท่ากับพื้นที่รูปขวา

ตามนิยาม RMS พื้นที่รูปซ้ายต้องเท่ากับพื้นที่รูปขวา

พื้นที่ซ้าย = พื้นที่ขวา

$$Vpp \int_0^{\pi} \sin t dt = Vrms \times \pi$$ (ดูรูป 2-15 ประกอบ)

$$- \text{Cos t} \Big|_{0}^{\pi} \text{x Vpp} = \pi \text{Vrms}$$

$$(-\text{Cos } \pi + \text{Cos } 0)\text{Vpp} = \pi \text{Vrms}$$

$$\frac{2}{\pi} Vpp = \text{Vrms} \ , \ \text{Vpp} = \frac{\pi}{2} Vrms$$

ความรู้นี้จะนำไปใช้ในการอ้างอิงการออกแบบต่อไป ไฟบ้าน 220 VAC ซึ่งเป็นแรงดัน RMS เมื่อผ่านหม้อแปลงจะได้ไฟกระแสสลับ 6 VAC ซึ่งเป็นแรงดัน RMS เช่นเดียวกัน หลังจากผ่านไดโอดที่ต่อกันแบบบริดจ์ จะได้ไฟกระแสตรงที่มีโวลต์เฉลี่ย 6 Vrms และมีแรงดันยอด 6 Vrms x π/2 = 9.4 V และจากการผ่านไดโอด 2 ตัว ที่ใช้ในการแปลงไฟกระแสสลับเป็นกระแสตรง (ไดโอดมีแรงดันตกคร่อมตัวละ 0.7 V) ทำให้แรงดันกระแสตรง 9.4 V เหลือเท่ากับ 8 V ซึ่งจะถูกตัวเก็บประจุ 1,000 μF สะสมและคายออกมาชดเชย ทำให้ไฟกระแสตรงที่ไม่เรียบมีแอมพลิจูดเรียบขึ้นเกือบเป็นเส้นตรง หลังจากนั้นจะผ่านเข้า IC 7805 ซึ่งทำงานเหมือน Pressure regulator ของระบบลมซึ่งไฟที่ผ่านออกจาก IC จะมีแรงดัน 5VDC เสมอ ไม่ว่าไฟเข้า IC จะมีแรงดันเท่าไร IC 7805 จะปรับแรงดันให้เป็น 5V ได้ตลอดเวลาอย่างอัตโนมัติ โดยการปรับความต้านทานของตัวมันเอง ซึ่งทำงานเป็นวงจรแบ่งแรงดันอย่างง่าย เมื่อความต้านทานของตัว IC เพิ่มขึ้น แรงดันที่ตกคร่อมตัว IC ก็จะสูงขึ้นด้วย ดังนั้นแรงดันที่เหลือที่ทางออกของ IC จะลดลง รูปแบบการทำงานเช่นนี้ทำให้ IC ลดแรงดันที่ทางเข้าได้ทางเดียว (เช่นเดียวกับ Pressure regulator ของระบบนิวแมติก) ดังนั้นถ้าแรงดันเข้ามาน้อยกว่า 5 V ตัวIC จะไม่สามารถเพิ่มแรงดันให้ออกมาเป็น 5 V ได้เพราะ IC ลดแรงดันได้อย่างเดียว

เนื่องจากความต้านทานภายในและลักษณะวงจรใน IC ทำให้เกิดแรงดันสูญเสียที่ตกคร่อมตัว IC อยู่เสมอ 1.5 V โดยประมาณ (ไฟที่ผ่าน IC จะมีแรงดันลดลงไปอย่างน้อย 1.5 V เสมอ สิ่งสำคัญไมโครคอนโทรลเลอร์ต้องไม่ทำงานผิดพลาดซึ่งต้องการไฟเลี้ยงอย่างต่ำ 3.7 V. รวมกับแรงดันสูญเสียจาก IC 7805 และไดโอดเรตติไฟ สรุปว่า หม้อแปลงต้องจ่ายแรงดันไม่ต่ำกว่า 3.7 + 1.5 + 1.4 = 6.6 V (peak) = 4.2 V_{RMS} หม้อแปลงลดแรงดันจาก 220 V ลงมาเท่ากับ 6 V_{RMS} ดังนั้นถ้าจะให้จ่ายแรงดันออกมาได้ 4.2 V_{RMS} แรงดัน AC ต้องไม่ต่ำกว่า 154 VAC

สรุปคือหากแรงดันไฟบ้าน(220 V) เกิดแรงดันตกขึ้นบา PLC ของเรายังสามารถทำงานได้ ถ้าไฟไม่ตกต่ำกว่า 154 V

IC 7805 ไม่สามารถลดแรงดันเหลือเท่ากับ 5 V ได้ตลอด หากมีความดันที่เข้ามาสูงกว่าค่าหนึ่ง IC ก็จะเสียได้ ซึ่งก็คือความดันที่ 20 VDC ถ้าคำนวณย้อนกลับไป ถ้าไฟบ้าน 220 V สูงเกินปกติไปไม่เกิน 400 V ตัวIC 7805 ยังสามารถลดไฟลงเป็น 5 Vได้ แต่ตัวหม้อแปลงนั้นถนวนระบุว่าไม่ควรใช้ไฟเกิน 280 V

จากข้อมูลดังกล่าว เราจึงระบุคุณสมบัติของเครื่อง PLC เราได้ว่า สามารถใช้กับแรงดันได้ตั้งแต่ 150 - 280 V

2.2.1 การวิเคราะห์อุณหภูมิอุปกรณ์

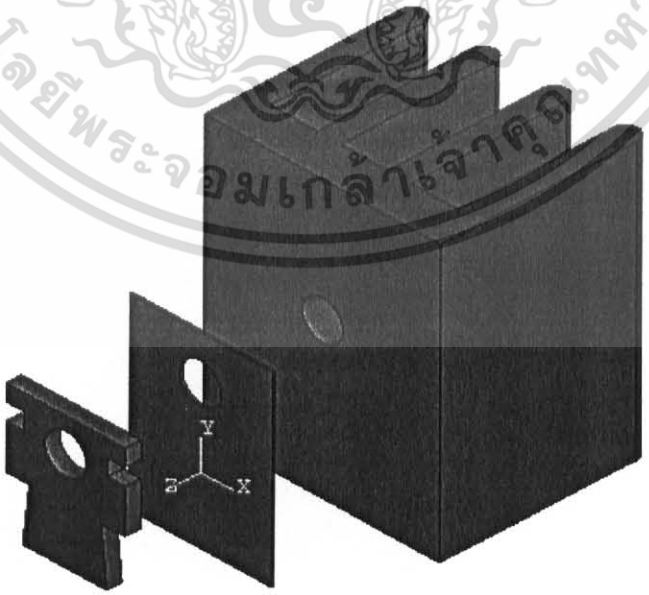
อีกเรื่องคือการทำงานของ IC 7805 ที่เรากล่าวไปแล้วว่ามันลดแรงดันไฟลงได้โดยการแบ่งแรงดันตกคร่อมที่ตัวมันเองส่วนหนึ่ง แรงดันนี้เมื่อมีกระแสไฟฟ้าไหลผ่านย่อมทำให้เกิดความร้อนขึ้น ดังนั้น IC 7805 จึงต้องมีการติด Heat sink ระบายความร้อน หาก IC มีความร้อนจากแรงดันตกคร่อมมากเกินไปหรือครีบระบายความร้อนเล็กไป ระบายความร้อนไม่ดีพอ IC ก็จะเสียได้

ทีนี้เรามากำหนด เงื่อนไขการคำนวณการระบายความร้อนกันก่อน กรณีที่จะทำให้ตัว IC เกิดความร้อนสูงสุด คือ ตอนไฟเข้าหม้อแปลงมาสูงสุดถึง 280 VAC ซึ่ง IC7805 จะเกิดแรงดันตกคร่อมสูงสุดไปด้วย (5.6 V ตกคร่อม) กระแสสูงสุดที่บอร์ดไมโครคอนโทรเลอร์และบอร์ดเอาท์พุทตอนทำงานเต็มที่อยู่ที่ 0.23 A ดังนั้นจะเกิดความร้อน = V x I = 1.3 W

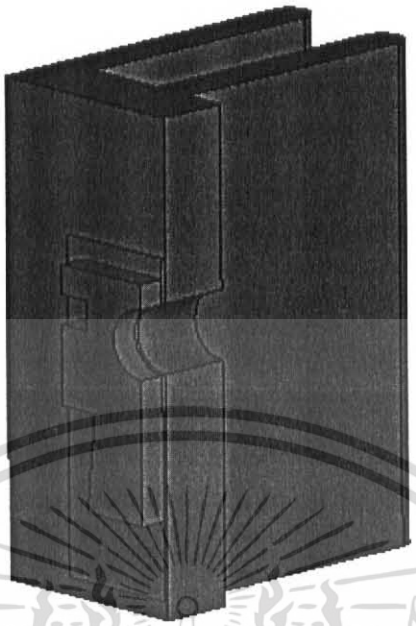
ในอุปกรณ์อิเล็กทรอนิกส์ทั้งหมด ตัวเก็บประจุชนิดอิเล็คโตรไลท์สามารถทนสภาพแวดล้อมได้แย่ที่สุดเพราะมีการบรรจุสารละลายอีเล็คโตรไลท์ที่เป็นสถานะของเหลวไว้ภายใน ในขณะที่อุปกรณ์อื่นๆ ส่วนใหญ่เป็นสถานะของแข็ง ตัวเก็บประจุนี้สามารถทนสภาวะแวดล้อมได้ที่ -10 C ถึง 60 C

ดังนั้น เงื่อนไขในการคำนวณที่ใช้มีดังต่อไปนี้ เกิดความร้อนที่ IC เท่ากับ 1.3 W และอุณหภูมิแวดล้อมเท่ากับ 60 C และเราจะทำการโมเดลสภาพแวดล้อมนี้ด้วยโปรแกรมทางไฟไนต์เอลิเมนต์ เพื่อดูค่าอุณหภูมิที่สะสมในตัว IC 7805 (IC 7805 จะเสียเมื่อสารกึ่งตัวนำมีอุณหภูมิสูงถึง 150 C) IC 7805 นั้นส่วนที่เป็นโลหะที่ใช้ระบายความร้อนนั้นสร้างอยู่บนชิ้นส่วนสารกึ่งตัวนำโดยตรง เพื่อให้การระบายความร้อนสะดวกที่สุด ดังนั้น ตัวถังนี้จะมีไฟไหลอยู่ด้วยเป็นปกติ ซึ่งตัว IC ต้องรองด้วยฉนวนซิลิโคลน ก่อนจะยึดเข้ากับ heat sink เพื่อกันไฟไหลไปที่ heat sink การโมเดลจึงต้องรวมฉนวนนี้เข้าไปด้วย

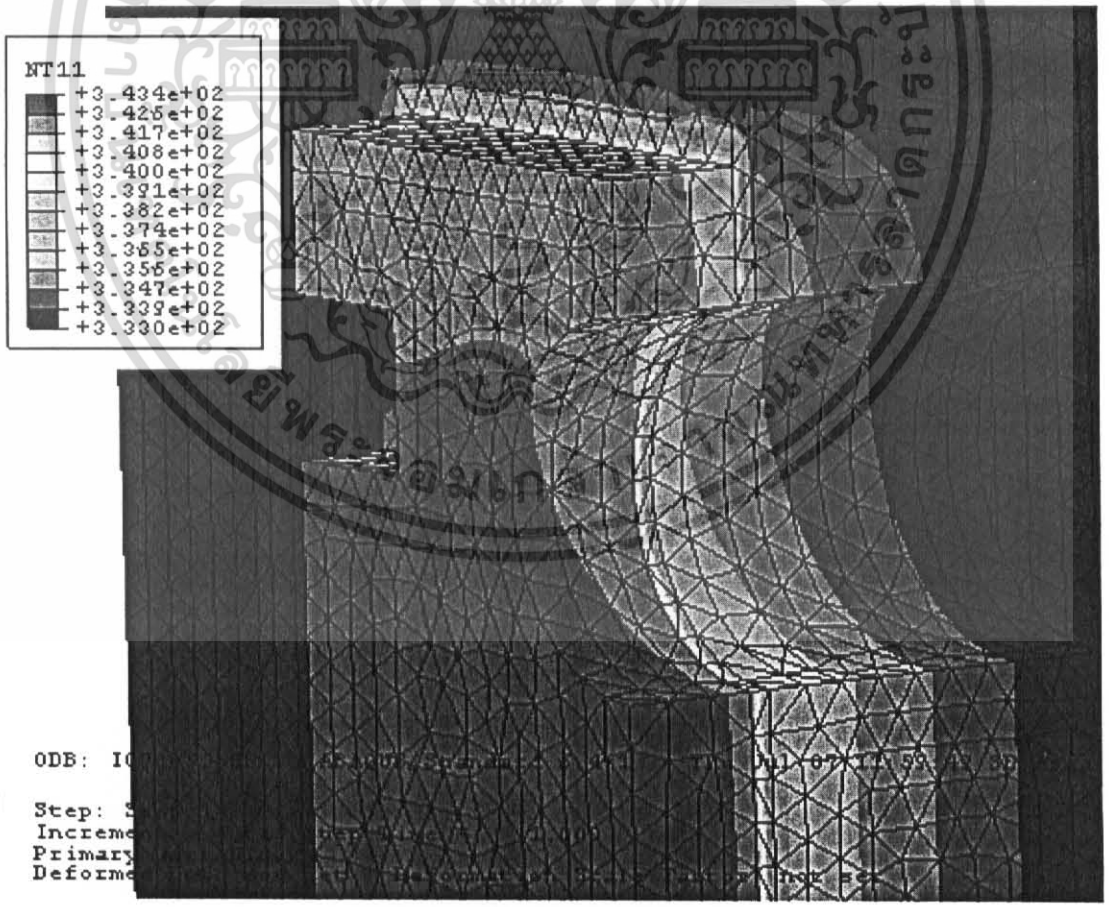
ผลจากการทำไฟไนต์เอลิเมนต์ประมาณได้ว่าอุณหภูมิสูงสุดของตัว IC อยู่ที่ 343.4 K, 70C ซึ่งไม่เกินจุดเสียหายด้วยอุณหภูมิ IC



รูปที่ 2-17 แสดงส่วนประกอบในการวิเคราะห์



รูปที่ 2-18 วิเคราะห์ครึ่งเดียว เพราะความสมมาตร (Symmetry y)



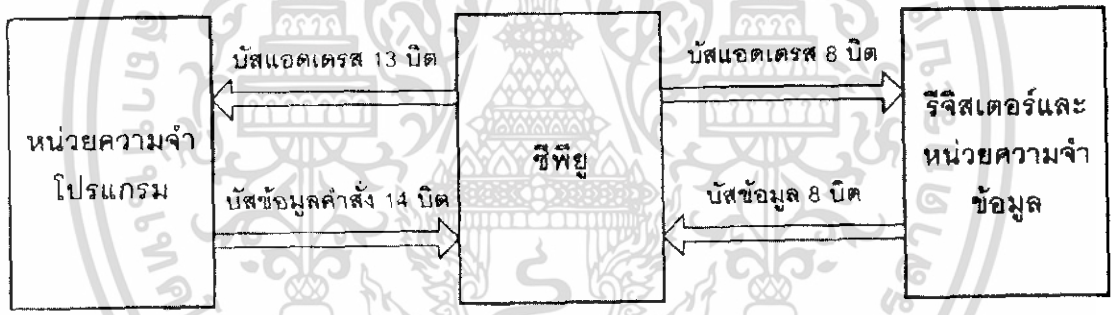
รูปที่ 2-19 แสดงจุดที่อุณภูมิสูงสุด (343.4 K, 70C)

2.3 การทำงานของส่วน CPU

2.3.1 โครงสร้างของไมโครคอนโทรลเลอร์ PIC 16F877A

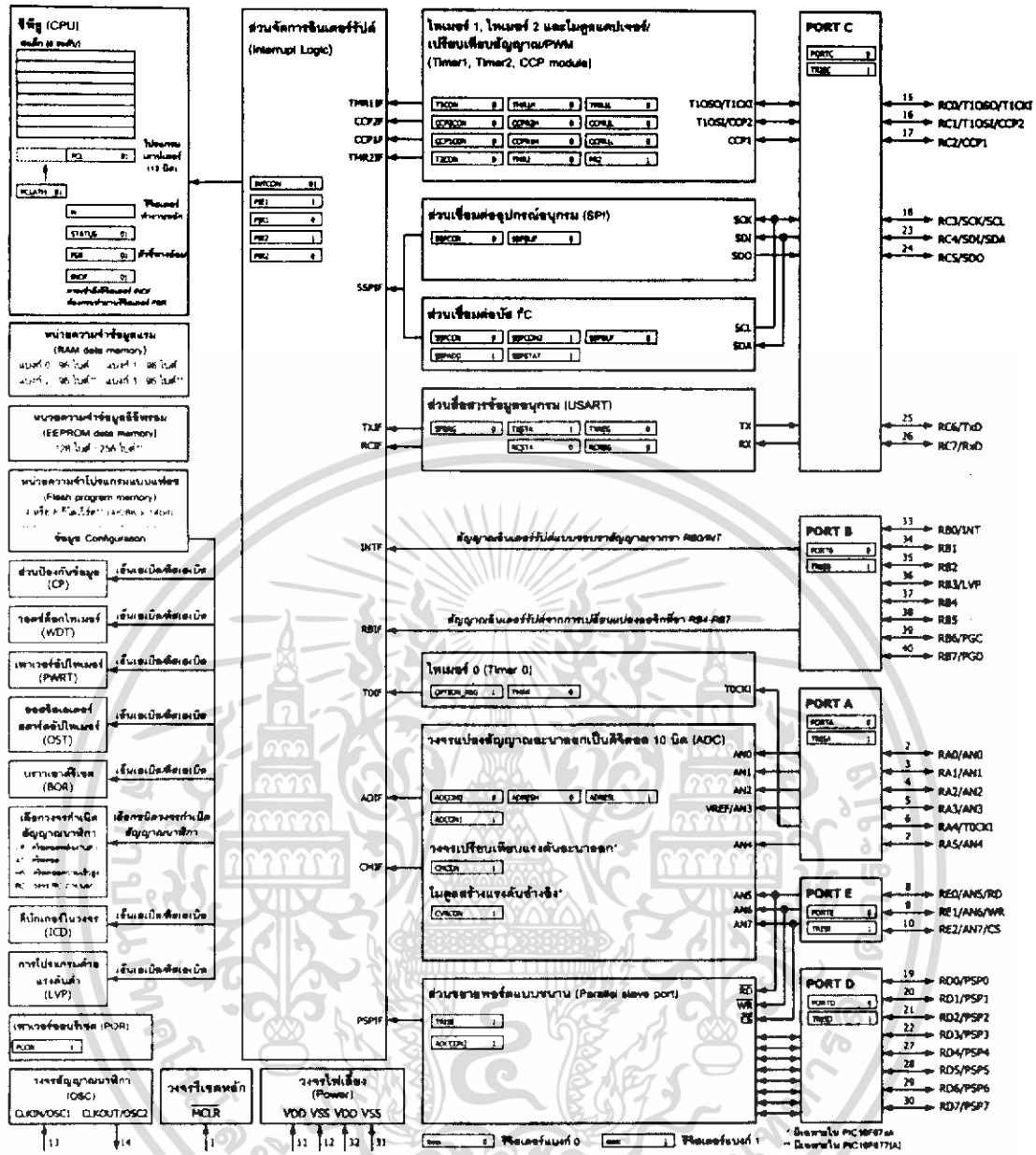
ไมโครคอนโทรลเลอร์ตระกูล PIC มีสถาปัตยกรรมแบบฮาร์วาร์ด (Harvard architecture) กล่าวคือ มีการแยกหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลออกจากกัน โดยมีบัสสำหรับติดต่อแยกกันด้วย ดังแสดงในรูปที่ 2-20 จะเห็นว่าซีพียูภายในไมโครคอนโทรลเลอร์จะติดต่อกับหน่วยความจำโปรแกรมด้วยบัสแอดเดรส 13 บิต และบัสข้อมูลหน่วยความจำโปรแกรม 14 บิต ในขณะที่บัสสำหรับติดต่อกับหน่วยความจำข้อมูลและรีจิสเตอร์ภายในเป็นแบบ 8 บิตทั้งบัสแอดเดรสและบัสข้อมูล

นอกจากการจัดสถาปัตยกรรมแบบนี้แล้ว การกระทำคำสั่งของไมโครคอนโทรลเลอร์ PIC ยังใช้กระบวนการที่เรียกว่า ไปป์ไลน์ (pipeline) ทำให้สามารถแฟตช์คำสั่งถัดไป ในขณะที่กำลังเอ็กซิคิวต์คำสั่งในปัจจุบัน ส่งผลให้ความเร็วในการทำงานของไมโครคอนโทรลเลอร์เพิ่มมากขึ้น นั่นจึงเป็นที่มาของความสามารถในการกระทำความสามารถในการกระทำคำสั่ง 1 ภายในสัญญาณนาฬิกา 1 ลูก (เฟตช์ : *fetch* เป็นกระบวนการเรียกคำสั่งออกจากหน่วยความจำโปรแกรมแล้วแปลเป็นเลขฐานสิบหกเพื่อให้ซีพียูเข้าใจ ส่วนกระบวนการเอ็กซิคิวต์(execute) เป็นการกระทำคำสั่งให้เกิดผลลัพธ์ตามที่คำสั่งนั้นๆ กำหนด)



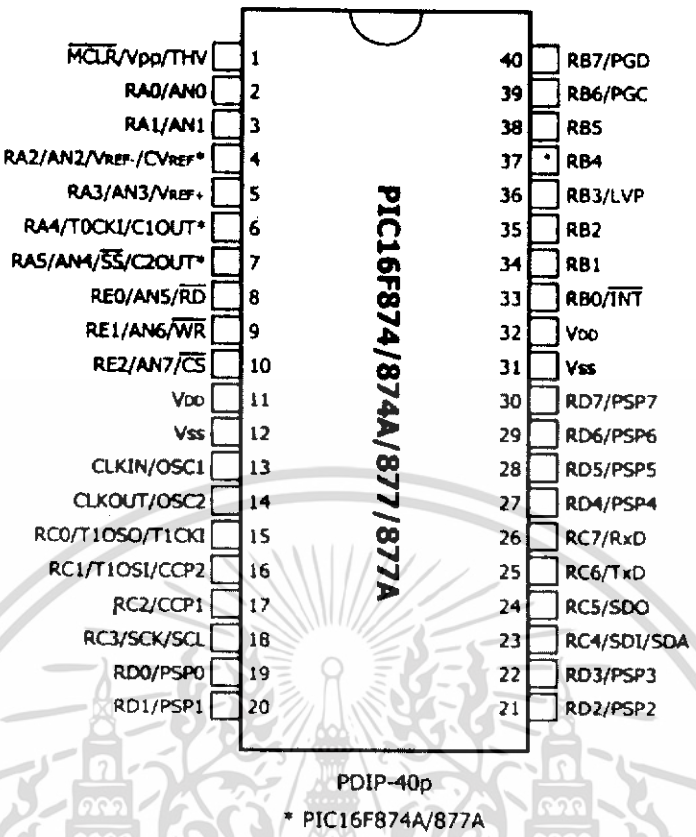
รูปที่ 2-20 โครงสร้างสถาปัตยกรรมไมโครคอนโทรลเลอร์แบบฮาร์วาร์ด

แสดงดังในรูปที่ 2-21 ส่วนประกอบหลักก็จะเหมือนกับไมโครคอนโทรลเลอร์ PIC 16F84 แต่จะมีส่วนที่เพิ่มเติมเข้ามามากพอควร ได้แก่ วงจรบราวเอาต์รีเซต (brown-out reset), ส่วนแก้ไขข้อมูลในวงจรหรือดีบักเกอร์ (In-circuit debugger), วงจรโปรแกรมข้อมูลด้วยแรงดันต่ำ (low-voltage programming), ไทเมอร์ที่มีมากถึง 3 ตัว, วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลขนาด 10 บิต,



รูปที่ 2-21 โครงสร้างไมโครคอนโทรลเลอร์ PIC 16F877A

วงจรเชื่อมต่ออุปกรณ์อนุกรม (SPI : Serial Peripheral Interfacing), วงจรเชื่อมต่อระบบบัส 12C, วงจรสื่อสารอนุกรม (USART :Universal Synchronous Asynchronous Receiver Transmitter) และโมดูลเปรียบเทียบสัญญาณ-ตรวจจับสัญญาณ-วงจรมอดูเลชั่นทางความกว้างของพัลส์หรือ PWM(CCP : Compare Capture Pulse-width modulation) นอกจากนั้นในอนุกรม PIC 16F87x จะมีวงจรเปรียบเทียบแรงดันอะนาลอกและโมดูลสร้างแรงดันอ้างอิงเพิ่มเติมเข้ามาอีกด้วย



รูปที่ 2-22 การจัดขาไมโครคอนโทรลเลอร์ PIC 16F877A

นอกจากนั้นขนาดของหน่วยความจำทั้งส่วนโปรแกรม,ข้อมูล, รีจิสเตอร์ และหน่วยความจำอีอีพรอมในไมโครคอนโทรลเลอร์ PIC 16F87x ก็มีเพิ่มมากขึ้น โดยสามารถสรุปคุณสมบัติทางเทคนิคและการจัดขาของ PIC 16F87x

PIC 16F87x มีรีจิสเตอร์พิเศษตัวหนึ่งที่บรรจุข้อมูลสำหรับกำหนดการทำงานทั้งหมดเอาไว้นั่นคือ Configuration word โดยภายใน Configuration word จะบรรจุข้อมูลของการเลือกป้องกันการอ่านข้อมูล, เลือกความสามารถการรีเซตอัตโนมัติเมื่อไฟเลี้ยงลดต่ำถึงค่าที่กำหนด, ควบคุมการทำงานของวอตช์ด็อกไทเมอร์ หรือกระทั่งการเลือกชนิดของวงจรกำเนิดสัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ การกำหนดข้อมูลสำหรับรีจิสเตอร์ตัวนี้สามารถกระทำได้ 2 ทางคือ ด้วยคำสั่ง _CONFIG ในส่วนต้นของโปรแกรมภาษาแอสเซมบลีแล้วแอสเซมเบลอร์ด้วยMPASMซึ่งบรรจุอยู่ในชุดของโปรแกรม MPASM อันเป็นซอฟต์แวร์สำหรับพัฒนาไมโครคอนโทรลเลอร์ PIC ของ Microchip ผู้ผลิตไมโครคอนโทรลเลอร์PIC นั่นเอง ทางที่สอง คือกำหนดที่ซอฟต์แวร์ที่ใช้ในการโปรแกรมหน่วยความจำของไมโครคอนโทรลเลอร์ การกำหนดสามารถกระทำในทางใดทางหนึ่งหรือทั้งสองทางก็ได้ แต่ถ้าการกำหนดทั้งสองทางแตกต่างกัน การกำหนดที่ซอฟต์แวร์ของเครื่องโปรแกรมจะมีนัยสำคัญสูงกว่า

รีจิสเตอร์ Configuration มีขนาด 14 บิต เท่ากับขนาดของเวิร์ดในหน่วยความจำโปรแกรม ตำแหน่งของ Configuration word อยู่ที่แอดเดรส 0x2007 ดังมีรายละเอียดของการกำหนดข้อมูลแต่ละบิต

2.3.2 การกำเนิดสัญญาณนาฬิกา

วงจรของส่วนสร้างสัญญาณนาฬิกาของไมโครคอนโทรเลอร์ PIC 16F877 มีได้ 4 แบบ คือ

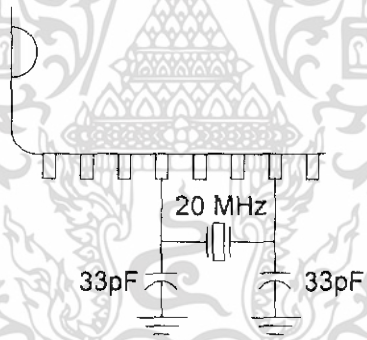
LP: คริสตอลแบบกำลังงานต่ำ

XT: คริสตอลแบบทั่วไปความถี่ไม่เกิน 4 MHz

HS: คริสตอลแบบความถี่สูงกว่า 4 MHz

RC: ใช้วงจร RC ภายนอก

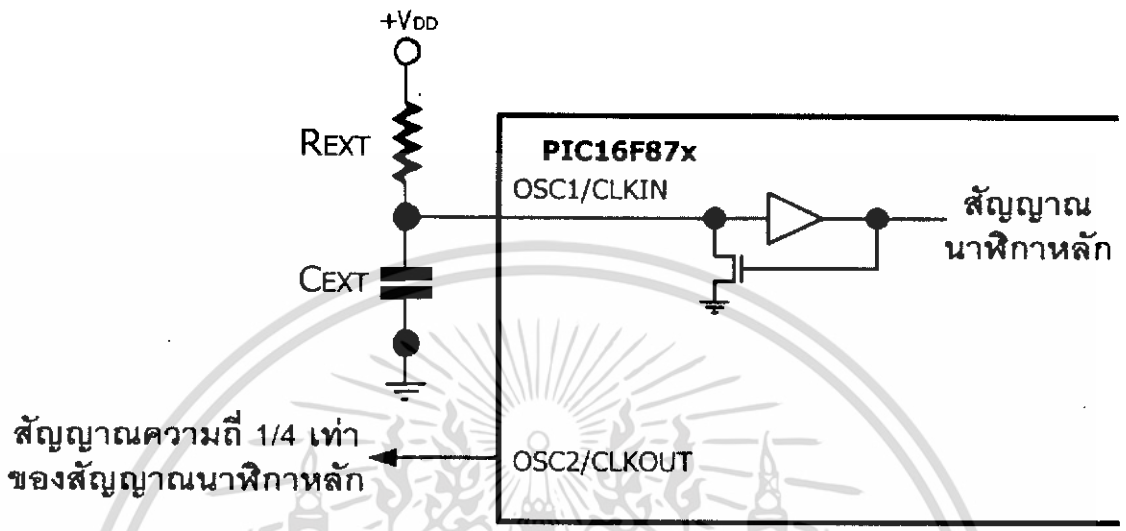
การกำเนิดสัญญาณนาฬิกา ด้วยวงจร RC นั้น ทำให้ความถี่ไม่แม่นยำ เนื่องจากตัวเก็บประจุสร้างให้มีค่าแม่นยำได้ยาก (โดยทั่วไปมีค่าผิดพลาดประมาณ 5% - 20%) และ RC Oscillator ยังไม่เหมาะกับการกำเนิดความถี่สูงมากๆ จึงควรใช้คริสตอลกำเนิดความถี่แทน วงจรจึงไม่จำเป็นต้องใช้โหมดกำเนิดความถี่แบบ LP เพราะวงจรส่วน output กินไฟมากว่าส่วนกำเนิดสัญญาณนาฬิกาหลายเท่า การใช้คริสตอลแบบประหยัดพลังงานจึงทำให้ทั้งระบบกินไฟน้อยมาก เพื่อให้ PLC ใช้เวลาประมวลผลแต่ละคำสั่งน้อยลง จึงเลือกใช้สัญญาณนาฬิกาสูงสุดที่ไมโครคอนโทรลเลอร์ทำได้คือ 20 MHz ดังนั้นการกำเนิดสัญญาณนาฬิกาจึงต้องใช้โหมด HS การต่อวงจรภายนอกเพื่อกำเนิดสัญญาณนาฬิกาในโหมด HS ทำได้โดยใช้ตัวเก็บประจุสองตัวและคริสตอลที่มีความถี่ธรรมชาติ 20 MHz เข้ากับไมโครคอนโทรเลอร์ที่ขา Osc1 และ Osc2 ตามรูปที่ 2-23



รูปที่2-23 วงจรกำเนิดสัญญาณนาฬิกา

การเลือกขนาดตัวเก็บประจุ 33 pF เป็นไปตามตัวอย่างวงจรที่ผู้ผลิตไมโครคอนโทรเลอร์แนะนำ ความผิดพลาดของค่าตัวเก็บประจุไม่มีผลทำให้ความถี่ที่เกิดผิดพลาดไปด้วย เพราะการ Oscillate จะเกิดความถี่ตามความถี่ธรรมชาติของคริสตอล (20 MHz) ค่าของตัวเก็บประจุที่ผิดพลาดจะทำให้การ Oscillate มี Amplitude ลดลงเท่านั้น ไม่เหมือนกับวงจร RC Oscillator ที่ความถี่ธรรมชาติที่กำเนิดออกมาจะได้รับผลกระทบจากค่าความผิดพลาดของค่า R และ C ตัวเก็บประจุ 33 pF นี้ ควรใช้ชนิดเซรามิคเพราะราคาถูกและมีอิมพิแดนซ์เหมาะกับการทำงานที่ความถี่สูง (20 MHz) ในขณะที่ตัวเก็บประจุชนิดอีเล็คโตรไลท์จะมีอิมพิแดนซ์สูงมากที่ความถี่สูงกว่า 1 MHz ขึ้นไป

PIC 16F87x สามารถเลือกโหมดของสัญญาณนาฬิกาเพื่อกำหนดจังหวะการทำงานได้มากถึง 4 โหมด โดยการกำหนดที่บิต FOSC0 และ FOSC1 ในรีจิสเตอร์ Configuration word ซึ่งในการทำงานจะต้องเลือกโหมดใดโหมดหนึ่ง ดังมีรายละเอียดต่อไปนี้

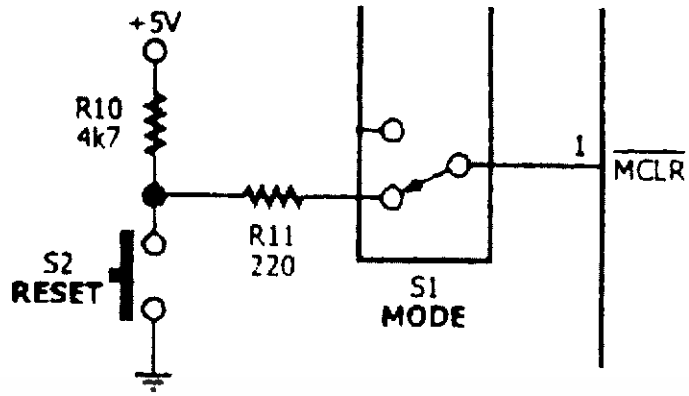


รูปที่ 2-24 การต่อตัวต้านทานและตัวเก็บประจุเพื่อกำหนดความถี่สัญญาณนาฬิกาเมื่อ ทำงานในโหมด RC

1. โหมด LP ใช้กับคริสตอลหรือเซรามิกเรโซเนเตอร์พลังงานต่ำความถี่ 32 kHz – 200 kHz
2. โหมด XT ใช้กับคริสตอลหรือเซรามิกเรโซเนเตอร์มาตรฐานความถี่ 200 kHz – 4 MHz
3. โหมด HS ใช้กับคริสตอลหรือเซรามิกเรโซเนเตอร์ความถี่สูง 4 MHz – 20 MHz
4. โหมด RC (External Resistor/Capacitor) สามารถกำหนดค่าความถี่ได้จากค่าของตัวต้านทานและตัวเก็บประจุที่ต่อภายนอกเข้ากับขา OSC1/CLKIN ดังแสดงในรูปที่ 2-24 ความถี่สูงสุดคือ 4 MHz อย่างไรก็ตามความถี่ของสัญญาณนาฬิกาในโหมดนี้ไม่อาจกำหนดลงไปได้อย่างชัดเจน เนื่องจากต้องพิจารณาถึงองค์ประกอบที่สามารถเปลี่ยนแปลงได้ในขอบเขตที่กว้าง ไม่ว่าจะเป็นค่าของแรงดันไฟเลี้ยง, ค่าของต้านทานและตัวเก็บประจุ ซึ่งต้องรวมไปถึงค่าความผิดพลาดของอุปกรณ์ทั้งสองด้วย อย่างไรก็ตามค่าของตัวต้านทานที่เหมาะสมอยู่ในย่าน 3 kΩ – 100 kΩ ส่วนค่าของตัวเก็บประจุควรมากกว่า 20 pF นอกจากนั้นที่ขา OSC2/CLKOUT จะมีสัญญาณนาฬิกาความถี่ 1/4 เท่าของความถี่สัญญาณนาฬิกาหลักส่งออกมา

2.3.3 กระบวนการรีเซตใน PIC 16F877A

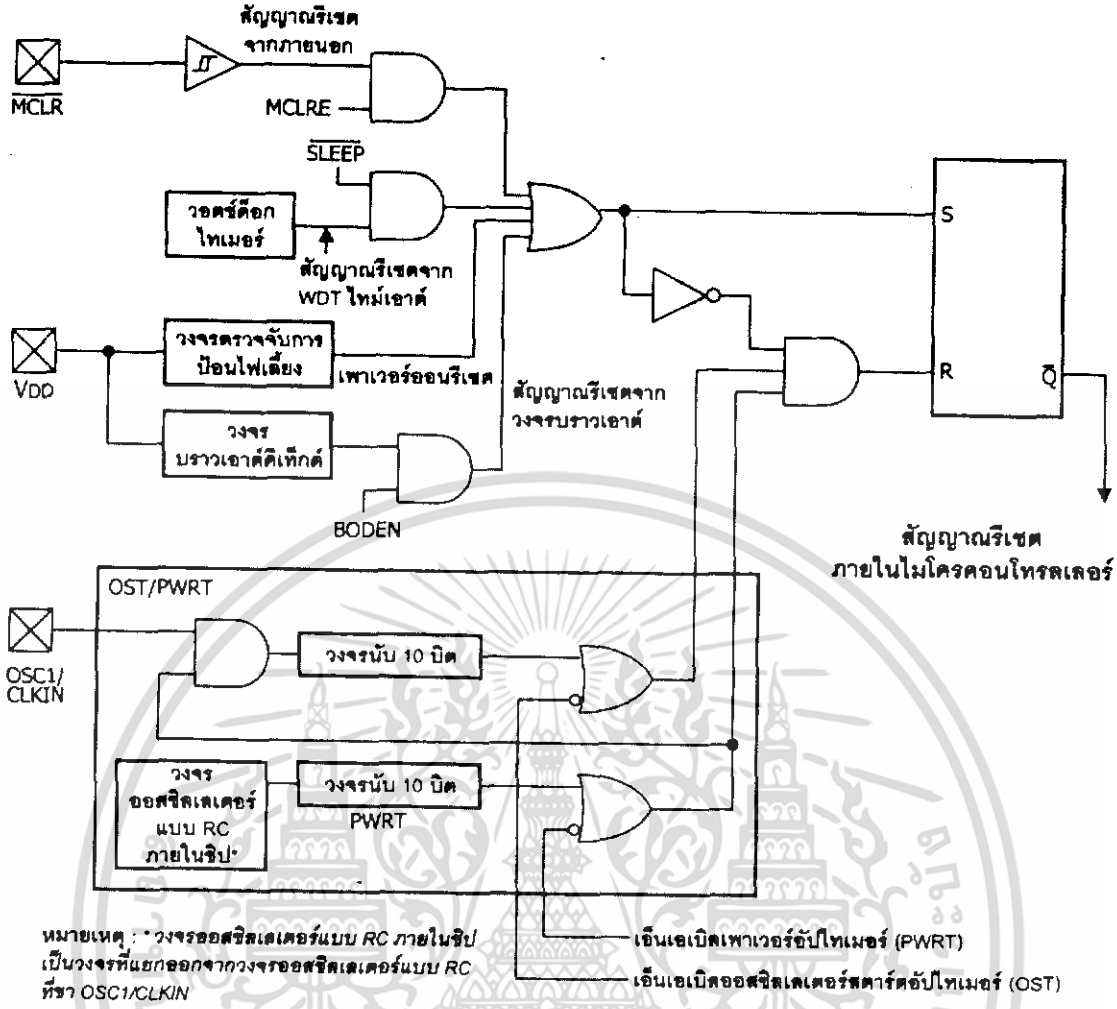
วงจร Reset โดยผู้ใช้ประกอบด้วย R10, R11, S2, S1 โดยภายใต้การใช้งานปกติจะต้องกำเนิดลอจิก "1" ให้กับขา 1 ($\overline{MCLR}$) เมื่อกด S2 (Reset) จะเป็นการกำเนิดลอจิก "0" ให้กับขา 1 ($\overline{MCLR}$) เพื่อเป็นสัญญาณรีเซ็ต



รูปที่ 2-25 วงจรรีเซต

การรีเซต (Reset) เป็นกระบวนการที่สำคัญมาในระบบไมโครคอนโทรลเลอร์ เป็นการกำหนดให้ซีพียูภายในไมโครคอนโทรลเลอร์เริ่มต้นการทำงานใหม่ ทั้งนี้เพื่อประโยชน์ในการแก้ไขความผิดปกติหรือการทำงานที่ผิดพลาดของไมโครคอนโทรลเลอร์ อันทำให้ไมโครคอนโทรลเลอร์ทำงานค้างอยู่ในสถานะใดสถานะหนึ่งหรือหยุดการทำงาน เมื่อทำการรีเซตไมโครคอนโทรลเลอร์ก็จะกลับมาเริ่มต้นการทำงานใหม่ ทำให้ระบบโดยรวมยังกลับมาทำงานได้ต่อไป การรีเซตในไมโครคอนโทรลเลอร์ PIC 16F87x มีด้วยกัน 6 ประเภท ดังนี้

1. เพาเวอร์ออนรีเซต (power-on reset) เป็นการรีเซตที่เกิดขึ้นหลังจากเริ่มต้นจ่ายไฟเลี้ยงใหม่ให้แก่ไมโครคอนโทรลเลอร์
2. การรีเซตที่ขา $\overline{MCLR}$ ในระหว่างการทำงานปกติ
3. การรีเซตที่ขา $\overline{MCLR}$ ขณะทำงานในโหมดสลีป
4. การรีเซตจากวอตช์ด็อกไทเมอร์ ในขณะทำงานปกติ
5. วอตช์ด็อกไทเมอร์เวกอัปขณะทำงานในโหมดสลีป
6. การรีเซตเนื่องจากไฟเลี้ยงลดต่ำลงจากที่กำหนดโดยวงจรบราวเอาต์ดีเท็ก (BOD : Brown-out Detect) เรียกว่า บราวเอาต์รีเซต (brown-out reset)



รูปที่ 2-26 แสดงกลไกของการเกิดสัญญาณรีเซตในไมโครคอนโทรลเลอร์PIC 16F87x

การรีเซตในแต่ละประเภทจะส่งผลกระทบต่อรีจิสเตอร์ที่แตกต่างกัน รีจิสเตอร์ส่วนใหญ่จะเข้าสู่สภาวะรีเซต (reset state) เมื่อเกิดเพาเวอร์ออนรีเซต, การรีเซตที่ขา $\overline{MCLR}$, การรีเซตเนื่องจากวอชด็อกไทเมอร์ และการรีเซตเนื่องจากบราวเอาต์ดีเท็ก และจะไม่ได้รับผลกระทบในกรณีที่วอชด็อกไทเมอร์เวกอัป ผู้ใช้งานสามารถตรวจสอบสาเหตุของการรีเซตได้จากบิตแสดงสถานะ 4 บิตคือ บิต $\overline{TO}$ กับ $\overline{PD}$ ในรีจิสเตอร์ STATUS และบิต $\overline{POR}$ กับ $\overline{BOR}$ ในรีจิสเตอร์ PCON ทั้งหมดของPIC 16F87x หลังจากที่มีการรีเซตในลักษณะต่างๆ เกิดขึ้น

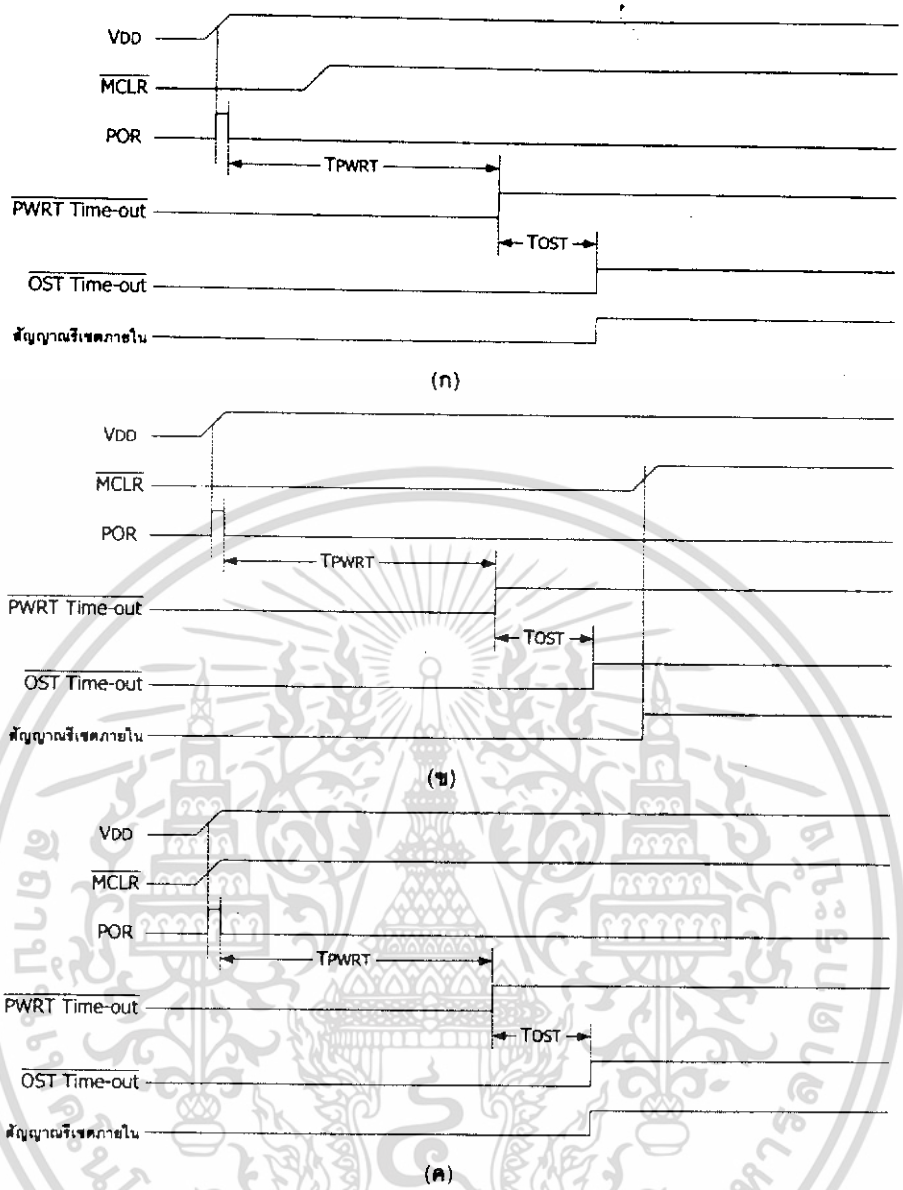
เพาเวอร์ออนรีเซต (Power-on reset : POR) เป็นการรีเซตที่มีนัยสำคัญสูงสุดของไมโครคอนโทรลเลอร์ เกิดขึ้นเมื่อหลังจากมีการจ่ายไฟเลี้ยงใหม่ให้แก่ไมโครคอนโทรลเลอร์ จึงเป็นการรีเซตที่เกิดขึ้นแรกสุด โดยวงจร POR ในไมโครคอนโทรลเลอร์จะควบคุมให้ไมโครคอนโทรลเลอร์อยู่ในสภาวะรีเซตจนกว่าไฟเลี้ยงจะเพิ่มขึ้นถึงจุดที่สูงเพียงพอให้ส่วนประกอบทั้งหมดในไมโครคอนโทรลเลอร์พร้อมทำงานได้ โดยปกติจะใช้เวลาประมาณ 0.05 V. ต่อมิลลิวินาที ถ้าไฟเลี้ยง 5 V. ก็จะใช้เวลาสูงสุดประมาณ 100 มิลลิวินาที

หลังจากที่เกิดเพาเวอร์ออนรีเซตจะมีไทเมอร์อีก 2 ตัวที่ทำงานเพื่อเตรียมความพร้อมให้แก่ตัวไมโครคอนโทรลเลอร์ ได้แก่ เพาเวอร์อัปไทเมอร์ (Power-up timer : PWRT) และออสซิลเลเตอร์สตาร์ตอัปไทเมอร์ (Oscillator start-up timer) มีรายระเอียดดังนี้

เพาเวอร์อัปไทเมอร์ (PWRT) ไทเมอร์ตัวนี้โดยปกติจะมีคาบเวลาคงที่ที่ 72 มิลลิวินาที จะทำงานหลังจากที่เกิดเพาเวอร์ออนรีเซตหรือบราวเอาต์รีเซต โดยจะเป็นตัวกำหนดให้ไมโครคอนโทรลเลอร์อยู่ในสภาวะรีเซต จนกว่าไฟเลี้ยงจะเพิ่มถึงจุดที่ทำงานได้ อย่างไรก็ตามสามารถที่จะดิสเอเบิลไทเมอร์ตัวนี้ได้โดยการเซตบิต PWRTE ใน Configuration word แต่ถ้าหากมีการเอ็นเอเบิลบราวเอาต์รีเซตไว้ ต้องเอ็นเอเบิลไทเมอร์ตัวนี้เสมอ อย่างไรก็ตามค่าเวลาของเพาเวอร์อัปไทเมอร์อาจไม่เท่ากันในไมโครคอนโทรลเลอร์แต่ละตัว และจะเปลี่ยนแปลงตามค่าของไฟเลี้ยงและอุณหภูมิด้วย

ออสซิลเลเตอร์สตาร์ตอัปไทเมอร์(OST) เป็นไทเมอร์ที่มีคาบเวลา 1,024 ไซเคิลของคาบเวลาสัญญาณนาฬิกาหลัก (TOSC) ที่ขาอินพุตสัญญาณนาฬิกา OSC1 ไทเมอร์ตัวนี้จะทำงานต่อจากเพาเวอร์อัปไทเมอร์ หน้าที่ของมันคือหน่วงเวลาเพื่อให้เกิดความแน่ใจว่าคริสตอลหรือเซรามิกเรโซเนเตอร์ที่ใช้ในการกำเนิดสัญญาณนาฬิกาได้เริ่มต้นทำงานและสร้างสัญญาณนาฬิกาที่มีเสถียรภาพเพียงพอแล้ว ไทเมอร์ OST จะทำงานก็ต่อเมื่อเลือกโหมดสัญญาณนาฬิกาเป็น LP, XT หรือ HS เท่านั้น และทำงานเฉพาะเมื่อเกิดเพาเวอร์ออนรีเซตหรือเมื่อไมโครคอนโทรลเลอร์เวกอัปออกจากโหมดสลีป

จังหวะการเกิดเพาเวอร์ออนรีเซต ในรูปที่ 2-27 แสดงไดอะแกรมเวลาของการเกิดสัญญาณรีเซตอันเนื่องมาจากการเริ่มต้นจ่ายไฟเลี้ยงใหม่ โดยจะมีเหตุการณ์ที่เกิดขึ้นได้ 3 ลักษณะ โดยในรูปที่ 2-27 (ก) และ(ข) เป็นไดอะแกรมเวลาที่อาจเกิดขึ้นในกรณีที่ไม่ต่อขา $\overline{MCLR}$ เข้ากับไฟเลี้ยง ส่วนในรูปที่ 2-27 (ค) เป็นกรณีที่ต่อขา $\overline{MCLR}$ เข้ากับไฟเลี้ยง พิจารณารูปที่ 2-27(ก) และ(ข) ไปพร้อมๆ กัน เมื่อเริ่มต้นจ่ายไฟเลี้ยง V_{DD} แรงดันจะเพิ่มค่าขึ้น พร้อมกันนั้นสัญญาณ POR ภายในจะเกิดการแอกตีฟ ทำให้ PWRT เริ่มต้นทำงาน หลังจากนั้น OST จะทำงานรับช่วงต่อ เมื่อครบเวลาแล้วสัญญาณรีเซตภายในจึงหยุดลง ทำให้ไมโครคอนโทรลเลอร์สามารถเริ่มต้นทำงานได้ พิจารณาที่ขา $\overline{MCLR}$ ในกรณีที่ไม่ต่อกับไฟเลี้ยง สถานะที่ขานี้จะเป็น "0" หรือ "1" อย่างไม่แน่นอน ถ้าหากเกิดเป็นลอจิก "1" ขึ้นก่อนที่ไทเมอร์ทั้งสองตัวคือ PWRT และ OST จะทำงานสิ้นสุด หลังจากที่ OST หยุดทำงาน ไมโครคอนโทรลเลอร์ก็จะทำงานได้ทันที7 แต่ถ้าหากขา $\overline{MCLR}$ ยังคงเป็น "0" หลังจากที่ PWRT และ OST หยุดทำงานแล้ว ตัวไมโครคอนโทรลเลอร์ยังคงอยู่ในสภาวะรีเซตต่อไปอีกชั่วขณะ หรือบางทีอาจยาวนาน ทั้งนี้ไม่สามารถควบคุมหรือกำหนดได้ จนกระทั่งขา $\overline{MCLR}$ เป็นลอจิก "1" สัญญาณรีเซตภายในก็จะหยุดลงดังในรูป 2-27(ข)

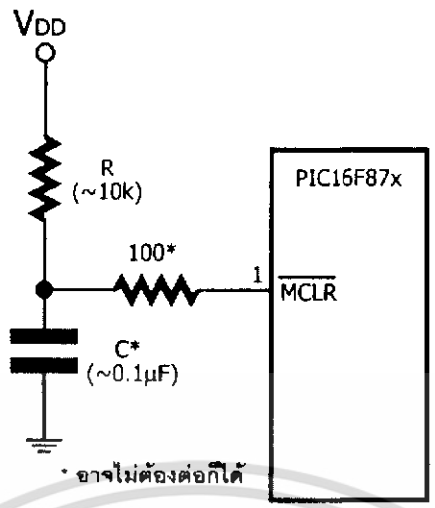


รูปที่ 2-27 ไดอะแกรมเวลาแสดงจังหวะการเกิดเพาเวอร์ออนรีเซต

(ก) และ (ข) เป็นกรณีที่ไม่ต่อขา $\overline{MCLR}$ เข้ากับไฟเลี้ยง

(ค) กรณีต่อขา $\overline{MCLR}$ เข้ากับไฟเลี้ยงโดยผ่านตัวต้านทาน จะเห็นได้ว่า เกิดการรีเซตอย่างสมบูรณ์ภายใต้ขอบเขตของเวลาที่มีความแน่นอน

แต่ถ้าหากต่อขา $\overline{MCLR}$ เข้ากับไฟเลี้ยงโดยผ่านตัวต้านทานพูลอัป จะทำให้ขา $\overline{MCLR}$ เป็นลอจิก "1" ไปจ่ายไฟเลี้ยง V_{DD} หลังจากนั้นสัญญาณรีเซตภายในก็จะเกิดขึ้น และหยุดลงหลังจาก OST หยุดทำงาน ทำให้ไมโครคอนโทรลเลอร์สามารถทำงานได้ทันทีอย่างแน่นอน ในรูปที่ 2-28 แสดงตัวอย่างการต่อวงจรเพาเวอร์ออนรีเซตเข้าที่ขา $\overline{MCLR}$ เพื่อควบคุมสถานะลอจิกที่ขา $\overline{MCLR}$ นี้



รูปที่ 2-28 วงจรเพาเวอร์คอนรีเซตสำหรับไมโครคอนโทรลเลอร์ PIC 16F87x

การรีเซตที่ขา $\overline{MCLR}$ เกิดขึ้นเมื่อขา $\overline{MCLR}$ ได้รับลอจิก "0" ในการใช้งานปกติ ขานี้จะต่อตัวต้านทานพูลอัปหรือวงจรเพาเวอร์ออนรีเซตตามรูปที่ 2-28 ซึ่งหลังจากจ่ายไฟแล้ว สถานะที่ขา $\overline{MCLR}$ จะถูกรักษาไว้ที่ระดับลอจิก "1" ในกรณีที่ต้องให้เกิดการรีเซตสามารถต่อขา $\overline{MCLR}$ นี้ลงกราวด์ชั่วขณะ แล้วปลดออก จะทำให้เกิดการรีเซตแล้ว ดังนั้นในทางปฏิบัติอาจต่อสวิตช์กดติดปล่อยดับเข้าระหว่างขา $\overline{MCLR}$ นี้กับกราวด์ หากต้องการให้เกิดการรีเซต ก็เพียงกดสวิสต์แล้วปล่อยเท่านั้น

การรีเซตแบบนี้จะทำให้ไมโครคอนโทรลเลอร์กลับมาเริ่มทำงานที่แอดเดรส 0x0000 ใหม่ ค่าในรีจิสเตอร์บางตัวไม่เปลี่ยนแปลง และข้อมูลในหน่วยความจำข้อมูลแรมจะไม่ได้รับผลกระทบถ้าหากไมโครคอนโทรลเลอร์ในโหมดสลีป และมีการป้อนลอจิก "0" เข้าที่ขา $\overline{MCLR}$ นอกจากทำให้เกิดการรีเซตแล้วยังทำให้เกิดการเวกอัปออกจากโหมดสลีปด้วย

การรีเซตเนื่องจากวอตช์ด็อกไทเมอร์ จะเกิดขึ้นเมื่อมีการเอ็นเอเบิลวอตช์ด็อกไทเมอร์ให้ทำงาน และไมโครคอนโทรลเลอร์ไม่สามารถเคลียร์ค่าวอตช์ด็อกไทเมอร์ได้ทันภายในคาบเวลาของวอตช์ด็อกไทเมอร์ อันเนื่องจากไมโครคอนโทรล-เลอร์ทำงานผิดพลาด วนอยู่กับการทำงานบางอย่าง จนไม่สามารถกลับมาทำงานตามขั้นตอนปกติได้ จนกระทั่งวอตช์ด็อกไทเมอร์เกิดไทม์เอาต์ วอตช์ด็อกไทเมอร์ก็จะสร้างสัญญาณรีเซตภายในส่งมายังซีพียู ทำให้เกิดการรีเซตไมโครคอนโทรลเลอร์ขึ้น

แต่ถ้าหากในขณะนั้นไมโครคอนโทรลเลอร์ทำงานอยู่ในโหมดสลีป การที่วอตช์ด็อกไทเมอร์ไทม์เอาต์จะเป็นการเวกอัปหรือกระตุ้นให้ไมโครคอนโทรลเลอร์ออกจากโหมดสลีป กลับมาทำงานในโหมดปกติ

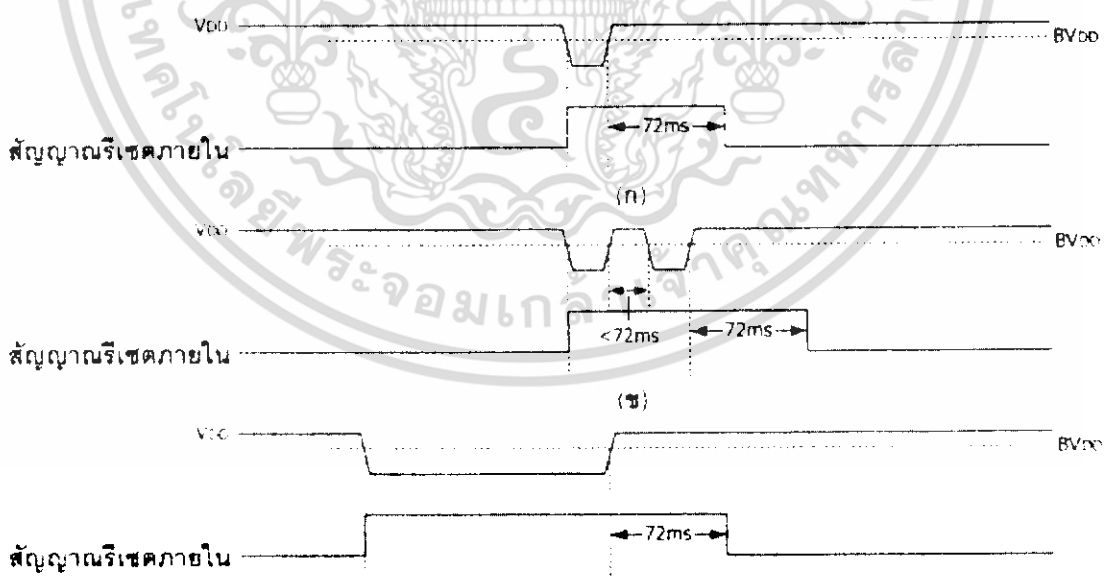
บราวเอาต์รีเซต ภายใน PIC 16F87x มีวงจรตรวจจับระดับแรงดันไฟเลี้ยงต่ำกว่าที่กำหนดหรือเรียกว่า บราวเอาต์ดีเท็ก (Brown-out detect: BOD) หากพบว่าไฟเลี้ยงของไมโครคอนโทรลเลอร์ลดต่ำลงถึงจุดที่กำหนด ก็จะกำเนิดสัญญาณรีเซตภายในส่งไปยังซีพียูเพื่อเริ่มต้นการทำงานใหม่ ถ้าหากไฟเลี้ยงกลับมาอยู่ในระดับปกติ ระบบก็จะสามารถปฏิบัติงานต่อไปได้อย่างไม่ติดขัด การใช้ BOD จะช่วย

แก้ปัญหาไมโครคอนโทรลเลอร์ทำงานผิดพลาดหรือหยุดทำงานอันเนื่องมาจากความไม่สม่ำเสมอของไฟเลี้ยง

สำหรับใน PIC 16F87x ระดับแรงดันที่ทำให้วงจร BOD ทำงานคือ 3.7 – 4.35 V ในการทำงานจริงเมื่อแรงดันลดลงถึง 4V เป็นเวลามากกว่า 100 ไมโครวินาที วงจร BOD จะทำงานเพื่อสร้างสัญญาณรีเซต ในรูปที่ 2-29 แสดงไดอะแกรมเวลาการเกิดสัญญาณรีเซตเนื่องจากการทำงานของวงจร BOD จากรูปไมโครคอนโทรลเลอร์ยังคงอยู่ในสภาวะรีเซตจนกว่าระดับไฟเลี้ยงจะกลับมาอยู่ในระดับต่ำสุดที่สามารถทำงาน ซึ่งทำให้เพาเวอร์อัปไทเมอร์ (PWRT) ทำงาน หน่วงเวลา 72 มิลลิวินาที ถ้าหากในช่วงเวลานั้นทำไฟเลี้ยงเกิดลดต่ำลงจนวงจร BOD ทำงาน ไมโครคอนโทรลเลอร์ก็จะกลับไปสู่สภาวะรีเซตอีกครั้ง รอคอยไฟเลี้ยงให้กลับมาอยู่ในระดับที่สามารถทำงานได้ นั่นคือเริ่มต้นกระบวนการของเพาเวอร์อัปไทเมอร์ใหม่อีกครั้ง จนกระทั่งไฟเลี้ยงคงที่ ไมโครคอนโทรลเลอร์ก็จะกลับมาทำงานตามปกติ

จากรูปที่ 2-29(ก) หากไฟเลี้ยงตกลงชั่วขณะแต่นานพอที่จะทำให้วงจร BOD ตรวจจับได้ ก็จะเกิดสัญญาณรีเซตภายในขึ้น หลังจากที่ไฟเลี้ยงกลับมาอยู่ในระดับที่เพียงพอต่อการทำงาน จะมีการหน่วงเวลาไปอีก 72 มิลลิวินาที

ในรูปที่ 2-29(ข) กรณีไฟเลี้ยงเกิดการกระเพื่อม จะมีช่วงเวลาหนึ่งที่มีค่าสูงถึงระดับที่ทำงานได้ แต่ทว่ามีระยะเวลาน้อยกว่า 72 มิลลิวินาที จึงยังคงทำให้ไมโครคอนโทรลเลอร์อยู่สภาวะรีเซต จะเห็นประโยชน์ของเพาเวอร์อัปไทเมอร์ในเวลานี้ เพราะมันจะช่วยให้ไมโครคอนโทรลเลอร์ยังไม่ต้องทำงานจนกว่าจะแน่ใจว่า มีการจ่ายไฟเลี้ยงที่ถูกต้องและมีระยะเวลาที่นานพอจะแน่ใจได้ว่า ไม่ใช่แรงดันกระชากที่ปะปนเข้ามาในสายเลี้ยง

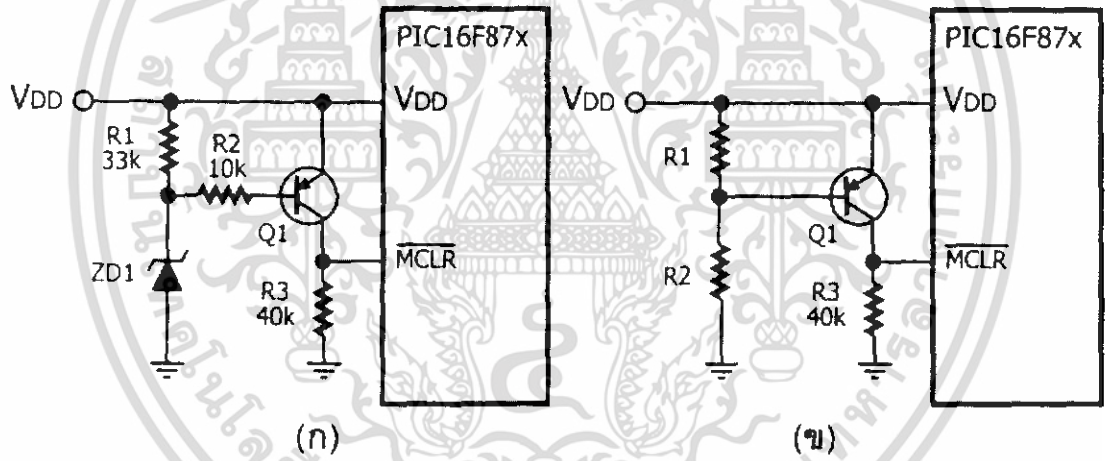


รูปที่ 2-29 ไดอะแกรมแสดงการเกิดบราวเอาต์รีเซตในลักษณะต่างๆ

สำหรับในรูปที่ 2-29 (ค) แสดงให้เห็นถึงการตกลงของไฟเลี้ยงเป็นเวลายาวนาน ไม่ใช่ชั่วขณะซึ่งวงจรบราวเอาต์ดีเท็กต์สามารถตรวจสอบได้ และควบคุมให้ไมโครคอนโทรลเลอร์อยู่ในสภาวะรีเซตยาวนานต่อเนื่องและหน่วงเวลาต่อไปอีก 72 มิลลิวินาทีหลังจากที่ไฟเลี้ยงกลับมาอยู่ในระดับปกติ

ผู้ใช้งานสามารถเลือกให้วงจร BOD ทำงานหรือไม่ก็ได้ โดยการกำหนดที่บิต BODEN ใน Configuration word ถ้าต้องการใช้งานให้เซตบิต BODEN แต่ถ้าต้องการดิสเอเบิลหรือไม่เลือกใช้งานให้เคลียร์บิต BODEN

อย่างไรก็ตาม ในกรณีที่มีความจำเป็นต้องดิสเอเบิลวงจร BOD ภายในไมโครคอนโทรลเลอร์แต่ยังคงต้องการการรีเซตแบบนี้อยู่ ก็สามารถทำได้โดยการต่อวงจรภายนอกเพิ่มเติม ดังแสดงตัวอย่างวงจรในรูปที่ 2-30 ซึ่งก็คือการป้อนลอจิก "0" ให้แก่ขา $\overline{MCLR}$ เพื่อรีเซตไมโครคอนโทรลเลอร์นั่นเอง หากแต่การเกิดสัญญาณรีเซตนี้จะมีเงื่อนไขคือ เมื่อแรงดันไฟเลี้ยงลดต่ำกว่าที่กำหนด ทรานซิสเตอร์ Q1 จะหยุดทำงาน ทำให้แรงดันตกคร่อม R3 เป็น "0" และแรงดันตกคร่อม R3 ก็คือแรงดันที่ป้อนให้แก่ขา $\overline{MCLR}$ เมื่อแรงดันเป็นศูนย์ จึงเกิดเป็นระดับลอจิก "0" ส่งเข้าที่ขา $\overline{MCLR}$ ไมโครคอนโทรลเลอร์จึงเกิดการรีเซตขึ้นในที่สุด



รูปที่ 2-30 ตัวอย่างการต่อวงจรบราวเอาต์ดีเท็กต์เข้าที่ขา $\overline{MCLR}$ ในกรณีที่ไม่เลือกใช้ความสามารถของวงจรบราวเอาต์ดีเท็กต์ภายในไมโครคอนโทรลเลอร์

(ก) ไมโครคอนโทรลเลอร์จะเกิดการรีเซตเมื่อแรงดันไฟเลี้ยงลดต่ำกว่าระดับ Vz + 0.7 V. โดย Vz คือแรงดันเบรกดาวน์ของซีเนอร์ไดโอด ZD1

(ข) ในวงจรนี้ที่ภาวะปกติ Q1จะทำงานตลอด ขา $\overline{MCLR}$ ได้รับไฟเลี้ยงตลอดเวลา เมื่อไฟเลี้ยงลดลงจนทำให้ Q1 หยุดทำงาน ก็จะไม่มีแรงดันป้อนที่ขา $\overline{MCLR}$ ทำให้เกิดการรีเซตขึ้น ระดับแรงดันที่ทำให้ Q1 หยุดทำงานสามารถคำนวณได้จาก $V_{DD} = \frac{0.7(Ri + R2)}{R1}$ การกำหนดค่าของ R1 และ R2 ต้องเหมาะสมพอที่จะทำให้ทรานซิสเตอร์ทำงานได้เมื่อระดับได้เมื่อระดับไฟเลี้ยงเป็นปกติ

2.3.4 พอร์ตอินพุตเอาต์พุตของไมโครคอนโทรลเลอร์ PIC16F87x

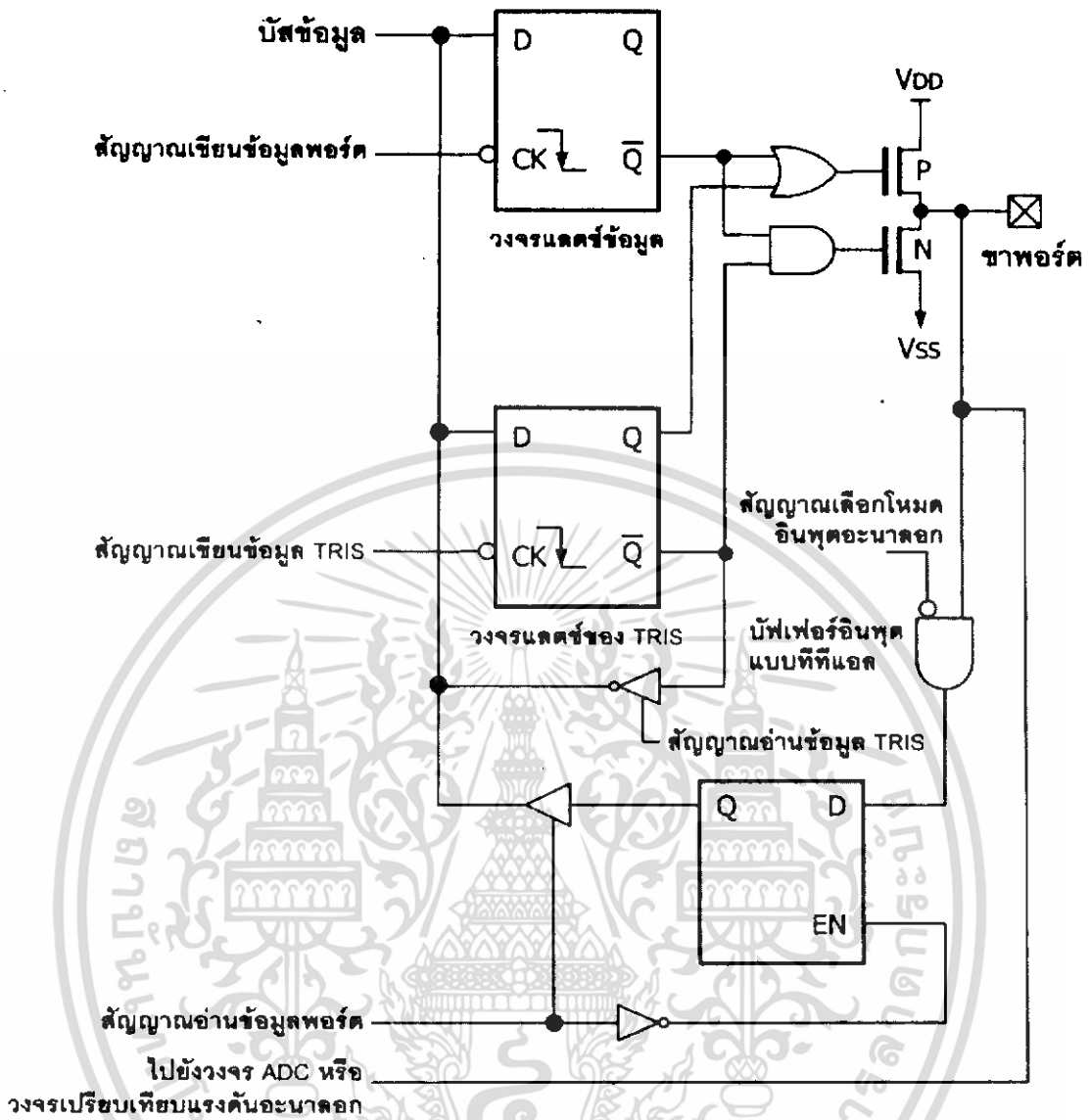
PIC16F87x มีพอร์ตให้ใช้งานตั้งแต่ 3-5 พอร์ต จำนวน 20-33 บิต ขึ้นอยู่กับเบอร์ของไมโครคอนโทรลเลอร์ ดังอธิบายไปแล้ว และด้วยความสามารถของพอร์ตใน PIC16F87x ที่สามารถทำงานได้หลายอย่าง จึงจำเป็นอย่างยิ่งที่ผู้ใช้งานต้องทำความเข้าใจถึงโครงสร้างทางฮาร์ดแวร์และการกำหนดหรือเลือกฟังก์ชั่นการทำงานให้แก่ขาพอร์ตแต่ละขาด้วยกระบวนการทางซอฟต์แวร์ ทั้งนี้เพื่อให้สามารถใช้งานพอร์ตทั้งหมดของ PIC16F87x ได้อย่างมีประสิทธิภาพสูงสุด

ในส่วนนี้จะกล่าวถึงภาพรวมของพอร์ตทั้งหมด ตั้งแต่พอร์ต A ถึง E โดยจะเน้นไปที่โครงสร้างทางฮาร์ดแวร์และฟังก์ชั่นการทำงานในภาพรวม สำหรับหน้าที่หรือฟังก์ชั่นพิเศษที่ขาพอร์ตนั้นๆ สามารถทำได้จะกล่าวถึงในรายละเอียดต่อไป

ความสามารถในการจ่ายกระแสเอาต์พุตของขาพอร์ตที่ไฟเลี้ยง +5 V. คือ 25 mA. ต่อขาทั้งกระแสซิงก์และกระแสซอร์ซ ในขณะที่กระแสเอาต์พุตรวมของพอร์ต A,B และ E มีค่าสูงสุด 200 mA ส่วนกระแสเอาต์พุตรวมของพอร์ต C และ D มีค่าสูงสุด 200 mA. ดังนั้นในการออกแบบเพื่อขับโหลดทางเอาต์พุตของขาพอร์ตต้องระวังเรื่องกระแสเอาต์พุตรวมที่ไมโครคอนโทรลเลอร์สามารถขับได้ด้วย

พอร์ต A มีทั้งสิ้น 6 ช่องหรือ 6 บิต กำหนดชื่อขาเป็น RA0-RA5 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORT A มีแอดเดรสอยู่ที่ 0x05 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต แต่ใช้งานจริงเพียง 6 บิต ที่เหลือ 2 บิตต้องกำหนดให้เป็น "0""ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISA ซึ่งมีแอดเดรสอยู่ที่ 0x85 (แบงก์ 1) มีขนาด 8 บิตและใช้เพียง 6 บิตเช่นกัน 2 บิตบนคือบิต 6 และบิต 7 ต้องกำหนดให้เป็น "0" บิต 0 ของ TRISA ใช้กำหนดทิศทางของขาพอร์ต RA0 ไล่เรียงลำดับจนถึงบิต 5 ของ TRISA ใช้กำหนดทิศทางของขาพอร์ต RA5 หากต้องการกำหนดให้ขาพอร์ตในบิตโตเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้น และในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุตให้เขียนข้อมูล "0" ไปยังบิตนั้น

โครงสร้างทางฮาร์ดแวร์ พอร์ต A สามารถทำงานเป็นขาพอร์ตอินพุตเอาต์พุทปกติและเป็นขาอินพุตสัญญาณอะนาลอกสำหรับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลขนาด 10 บิตภายในไมโครคอนโทรลเลอร์ โดยขา RA0-RA3 และ RA5 จะมีการทำงานที่เหมือนกัน ส่วน RA4 จะแตกต่างตรงที่ขานี้นอกจากเป็นขาพอร์ตอินพุตเอาต์พุตปกติแล้ว ยังใช้เป็นขาอินพุตสำหรับไทเมอร์ 0 ภายในไมโครคอนโทรลเลอร์ด้วย และขา RA4 นี้ไม่สามารถใช้งานเป็นขาอินพุตรับสัญญาณอะนาลอกได้



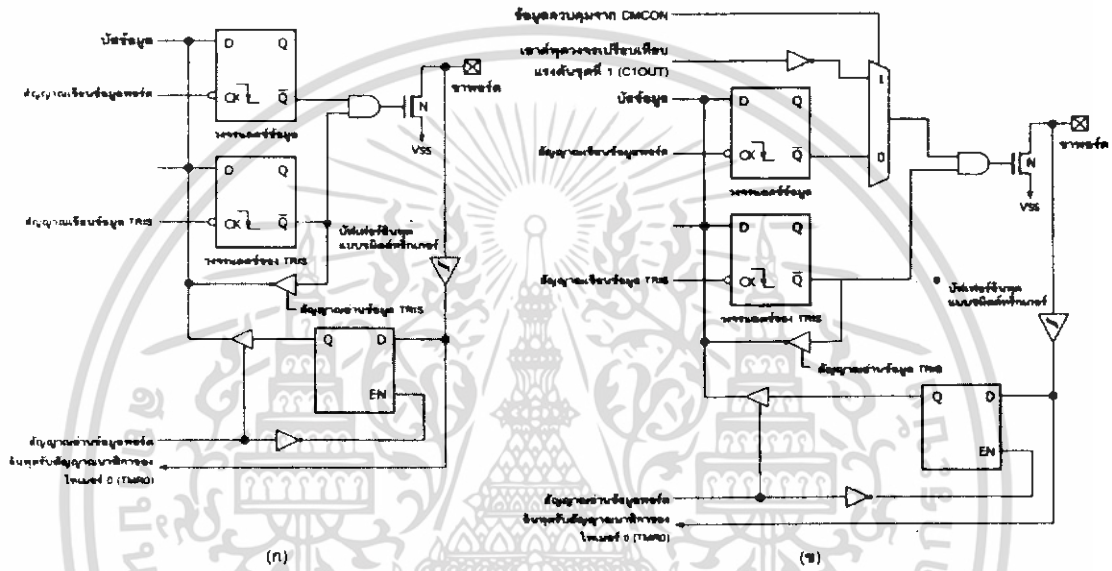
รูปที่ 2-31 โครงสร้างขา RA0-RA3 ของพอร์ต A

ในรูปที่ 2-31 แสดงไดอะแกรมของพอร์ต A ในบิต RA0-RA3 จะเห็นได้ว่าที่ขาพอร์ตจะมีแอนด์เกตทำหน้าที่เลือกลักษณะการทำงานของขาพอร์ตเมื่อเป็นขาอินพุต หากเลือกให้ขาพอร์ตนี้เป็นขาพอร์ตอินพุตดิจิตอล สัญญาณเลือกโหมดอินพุตอะนาลอกจะต้องเป็นลอจิก "0" แต่ถ้าหากต้องการกำหนดให้เป็นขาอินพุตสัญญาณอะนาลอก สัญญาณเลือกโหมดต้องเป็นลอจิก "1" สัญญาณอะนาลอกที่ป้อนเข้ามายังขานี้ จะผ่านเข้าสู่วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลหรือวงจรเปรียบเทียบแรงดันอะนาลอก (เฉพาะในอนุกรม PIC16F87xA) โดยตรง

เมื่อขาพอร์ต RA0-RA3 ทำงานเป็นขาพอร์ตอินพุตดิจิตอล จะสามารถรับสัญญาณดิจิตอลระดับทีทีแอล (0-5V.) ได้โดยตรง หากทำงานเป็นเอาต์พุทจะสามารถขับโหลดที่ต้องการกระแส 20 mA ได้ หากนำมาขับ LED ต้องต่อตัวต้านทานจำกัดกระแส หรือถ้าใช้ไฟเลี้ยง 3 V ก็จะขับ LED ได้โดยตรง

ในรูปที่ 2-32 เป็นไดอะแกรมของขาพอร์ต RA4/T0CKI โดยขานี้จะเป็นขาพอร์ตอินพุตเอาต์พุตปกติ และขาอินพุตรับสัญญาณนาฬิกาจากภายนอกของโมดูลไทเมอร์ 0 วงจรอินพุตบัฟเฟอร์ที่ขาพอร์ตนี้เป็นแบบชมิตต์ทริกเกอร์ ทั้งนี้เพื่อจัดการให้สัญญาณอินพุตที่เข้ามามีความสมบูรณ์มากที่สุด และจะต้องต่อตัวต้านทานพูลอัปค่าประมาณ 4.7 kΩ - 10 kΩ ที่ขานี้เสมอเมื่อใช้งานเป็นอินพุต

สำหรับใน PIC16F87xA ขาพอร์ต RA4 ยังใช้เป็นขาเอาต์พุตของโมดูลเปรียบเทียบแรงดันอะนาลอกชุดที่ 1 (C1OUT) ด้วย โครงสร้างภายในจึงมีการเปลี่ยนแปลงจากเดิมไปเล็กน้อย ดังแสดงในรูปที่ 2-32 (ข)



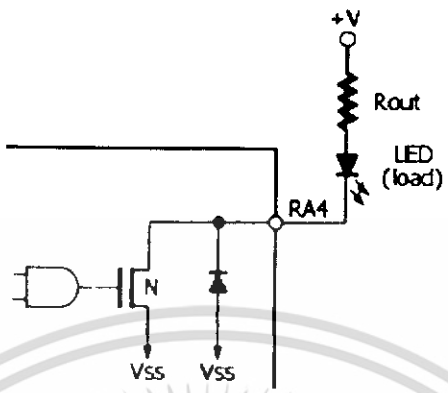
รูปที่ 2-32 โครงสร้างขา RA4 ของพอร์ต A

ในกรณีที่ใช้งานเป็นพอร์ตเอาต์พุต วงจรเอาต์พุตจะเป็นแบบเดรนเปิด (open drain) ถ้าหากเทียบกับวงจรทรานซิสเตอร์ก็คือ วงจรคอกเล็กเตอร์เปิด (open collector) นั่นเอง ในการใช้งานจึงต่อตัวต้านทานอนุกรมกับโหลดและไฟเลี้ยงของโหลด ดังวงจรในรูปที่ 2-33 โดยปกติถ้าหากไม่มีความจำเป็นควรกำหนดให้ขา RA4/T0CKI นี้เป็นอินพุตจะดีที่สุด โดยต้องต่อตัวต้านทานพูลอัปเสมอ หรือถ้าหากจำเป็นต้องใช้เป็นเอาต์พุตก็ต้องไม่ลืมว่า ขานี้เป็นเอาต์พุตแบบเดรนเปิด ต้องต่อขับโหลดในแบบกระแสซิงก์เท่านั้น

สำหรับในอนุกรม PIC16F87xA ขาพอร์ต RA4 ยังใช้เป็นขาเอาต์พุตของวงจรเปรียบเทียบแรงดันอะนาลอกชุดที่ 1 ด้วย โดยโครงสร้างของเอาต์พุตนี้ก็ยังคงเป็นแบบเดรนเปิดเช่นเดียวกับการใช้ขา RA4 เป็นพอร์ตดิจิตอลเอาท์พุตปกติ

โครงสร้างของพอร์ต RA5 ขานี้ก็มีความพิเศษไม่น้อย โดยนอกจากเป็นขาพอร์ตอินพุตเอาต์พุตดิจิตอลแล้ว ยังสามารถเลือกให้ทำงานเป็น ขาพอร์ตอินพุตสัญญาณอะนาลอก สำหรับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลในช่อง 4 และเป็นขาเอาต์พุตของวงจรเปรียบเทียบแรงดัน อะนาลอกชุดที่ 2

ในไมโครคอนโทรลเลอร์อนุกรม PIC16F87xA นอกจากนั้นยังใช้เป็นขา Slave Select (SS) ในกรณีที่เลือกใช้การสื่อสารข้อมูลอนุกรมซิงโครนัสในแบบ SPI ซึ่งจะได้กล่าวถึงในรายละเอียดต่อไป



รูปที่ 2-33 การต่ออุปกรณ์เข้าที่ขาพอร์ต RA4 เมื่อใช้งานเป็นพอร์ตเอาต์พุต

การติดต่อเพื่อกำหนดการทำงานและเขียนข้อมูลไปยังพอร์ต A ในขั้นแรกต้องทำการเตรียมความพร้อมให้แก่รีจิสเตอร์ PORT A โดยการเลือกแบงก์สำหรับติดต่อกับรีจิสเตอร์ PORT A จากนั้นส่งข้อมูล "0" เพื่อเคลียร์ข้อมูลทั้งหมด และเนื่องจากพอร์ต A ทำงานกับสัญญาณอะนาลอกได้ และค่าเริ่มต้นของพอร์ต A ที่ทำงานกับสัญญาณอะนาลอกจะถูกกำหนดให้เป็นอินพุตอะนาลอกทั้งหมด ดังนั้นหากต้องการใช้งานเป็นพอร์ตดิจิตอลต้องกำหนดข้อมูล 0x06 หรือ 0x07 แล้วเขียนลงในรีจิสเตอร์ ADCONI เพื่อดิสเอเบิลการทำงานกับสัญญาณอะนาลอก

จากนั้นจึงเลือกแบงก์ใหม่เพื่อติดต่อกับรีจิสเตอร์ TRISA แล้วเขียนข้อมูลเพื่อกำหนดทิศทางของขาพอร์ตตามที่ต้องการลงในรีจิสเตอร์ TRISA

พอร์ต B มี 8 บิต กำหนดชื่อขาเป็น RB0-RB7 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORT B มีแอดเดรสอยู่ที่ 0x06 (แบงก์ 0) และ 0x106 (แบงก์ 2) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของขนาด พอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISB ซึ่งมีแอดเดรสอยู่ที่ 0x86 (แบงก์ 1) และ 0x186 (แบงก์ 3) มีขนาด 8 บิต เช่นเดียวกับพอร์ต A บิต 0 ของ TRISB ใช้กำหนดทิศทางของขาพอร์ต RB0 ไล่เรียงลำดับจนถึงบิต 7 ของ TRISB ใช้กำหนดทิศทางของขาพอร์ต RB7 หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้น ในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุตให้เขียนข้อมูล "0" ไปยังบิตนั้น

โครงสร้างทางฮาร์ดแวร์พอร์ต B สามารถใช้งานในลักษณะต่างๆ ได้ 5 แบบคือ

1. เป็นขาพอร์ตอินพุตเอาต์พุตปกติ
2. เป็นขาอินพุตสัญญาณอินเตอร์รัปต์จากภายนอก โดยใช้ขา RB0/ $\overline{INT}$
3. เป็นขาพอร์ตอินพุตสำหรับรับแรงดันโปรแกรมระดับต่ำ (low voltage programming) โดยใช้ขา RB3/PGM

4. เป็นขาข้อมูลอนุกรมและสัญญาณนาฬิกาอนุกรมสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์ ซึ่งใช้ 2 ขาคือ RB7/PGD และ RB6/PGC

5. ใช้เป็นแหล่งกำเนิดสัญญาณอินเตอร์รัปต์แบบตรวจสอบการเปลี่ยนแปลงข้อมูลหรือระดับสัญญาณที่ขา RB4-RB7

ในพอร์ต B ในบิต RB6-RB3 จะเห็นได้ว่าที่ขาพอร์ตจะมีวงจรพูลอัปแบบโปรแกรมได้ต่ออยู่ นั่นคือหากต้องการกำหนดให้เป็นขาอินพุต ต้องทำการเขียนข้อมูล "0" ไปยังบิต RBPU ในรีจิสเตอร์ OPTION_REG เพื่อเอ็นเอเบิลวงจรพูลอัปภายในขาพอร์ต B ในขณะที่หากกำหนดเป็นเอาต์พุต การพูลอัปที่ขาพอร์ต B นี้จะถูกยกเลิกโดยอัตโนมัติ นอกจากนั้นการพูลอัปนี้จะได้รับการยกเลิกเมื่อเกิดเพาเวอร์ออนรีเซตขึ้น ในกรณีใช้ขา RB0/ $\overline{INT}$ เพื่อรับสัญญาณอินเตอร์รัปต์จากภายนอก สัญญาณจะผ่านเข้าไปยังวงจรบัฟเฟอร์แบบชมิตต์ทริกเกอร์เพื่อให้สัญญาณที่ได้มีความแม่นยำและมีเสถียรภาพ

ในพอร์ต (RB4-RB7) โดยขาพอร์ตในกลุ่มนี้มีความสามารถพิเศษพอสมควร โดยสามารถเลือกให้ทำงานเป็นขาพอร์ตอินพุตเอาต์พุตปกติ, ขาอินพุตรับแรงดันสำหรับการโปรแกรม (RB3), ขาสัญญาณสำหรับการโปรแกรม (RB6-RB7) และทำงานเป็นแหล่งกำเนิดสัญญาณอินเตอร์รัปต์ในแบบตรวจสอบการเปลี่ยนแปลงที่ขาพอร์ต RB4-RB7

วงจรอินพุตบัฟเฟอร์ที่ขาพอร์ตนี้มีทั้งแบบทีทีแอลและชมิตต์ทริกเกอร์ ทั้งนี้เพื่อจัดการให้สัญญาณอินพุทที่เข้ามามีความเหมาะสมและสมบูรณ์มากที่สุด และยังคงสามารถรองรับการพูลอัปภายในแบบอัตโนมัติได้ ในกรณีที่มีการเอ็นเอเบิลการตอบสนองอินเตอร์รัปแบบตรวจสอบการเปลี่ยนแปลงลอจิกที่พอร์ต RB4-RB7 หากเกิดการอินเตอร์รัปต์ขึ้น บิต RBIF (บิต 0 ในรีจิสเตอร์ INTCON) จะเซตและหลังจากตอบสนองการอินเตอร์รัปต์แล้ว ต้องเคลียร์บิต RBIF ด้วยกระบวนการทางซอฟต์แวร์เสมอ

พอร์ต C มีทั้งสิ้น 8 บิต กำหนดชื่อขาเป็น RC0-RC7 รีจิสเตอร์ที่ใช้เก็บข้อมูลคือ PORT C มีแอดเดรสอยู่ที่ 0x07 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISC มีแอดเดรสอยู่ที่ 0x87 (แบงก์ 1) มีขนาด 8 บิต เช่นเดียวกับพอร์ต A และ B บิต 0 ของ TRISC ใช้กำหนดทิศทางของขาพอร์ต RC0 ไล่เรียงลำดับจนถึงบิต 7 ของ TRISC ใช้กำหนดทิศทางของขาพอร์ต RC7 หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้น และทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุต ให้เขียนข้อมูล "0" ไปยังบิตนั้น

โครงสร้างทางฮาร์ดแวร์ พอร์ต C สามารถใช้งานในลักษณะต่างๆ ได้หลายรูปแบบ และเป็นขาพอร์ตที่มีความสามารถสูงมาก ไม่ว่าจะเป็นขาพอร์ตอินพุตเอาต์พุตปกติ, ขาอินพุตเอาต์พุตออสซิลเลเตอร์ของโมดูลไทเมอร์ 1, ขาอินพุตสำหรับรับสัญญาณนาฬิกาของโมดูลไทเมอร์ 1, ขาเชื่อมต่อระบบบัส I ^{2}C, ขาเชื่อมต่อแบบ SPI (Serial Peripheral Interface), ขาเชื่อมต่อพอร์ตอนุกรมแบบ USART , ขาอินพุตวงจรแคปเจอร์ (capture) หรือวงจรตรวจจับสัญญาณ, ขาเอาต์พุตของวงจรเปรียบเทียบสัญญาณ (compare) และขาเอาต์พุตวงจร PWM (Pulse Width Modulation) หรืออาจกล่าวได้ว่าพอร์ต C เป็นพอร์ตสำหรับเชื่อมต่ออุปกรณ์ภายนอก (peripheral function port) ที่มีความสมบูรณ์แบบมากที่สุด สามารถสรุปหน้าที่และการทำงานที่หลากหลายของขาพอร์ต C

ในพอร์ต C ในบิต RC0-RC2 และ RC5-RC7 จะเห็นได้ว่ามีสัญญาณควบคุมการทำงานของขาพอร์ตมากมาย ทั้งนี้เนื่องจากขาพอร์ต C สามารถทำงานได้หลากหลายนั่นเอง สัญญาณควบคุมที่สำคัญคือ สัญญาณเลือกการทำงานระหว่างเป็นพอร์ตปกติหรือเป็นขาเชื่อมต่ออุปกรณ์พิเศษ (PORT/PERIPHERAL Select) และสัญญาณควบคุมการส่งข้อมูลของวงจรเชื่อมต่ออุปกรณ์ (Peripheral Output Enable) สำหรับข้อมูลของวงจรเชื่อมต่ออุปกรณ์ที่ส่งออกและรับเข้ามาจะผ่านทางขาพอร์ตปกติ แต่เมื่อผ่านวงจรสำหรับเลือกสัญญาณข้อมูลแล้ว สายสัญญาณข้อมูลของพอร์ต (data bus) กับข้อมูลของวงจรเชื่อมต่ออุปกรณ์ (Peripheral Output/peripheral input) จะแยกกัน

ใน พอร์ต RC3-RC4 ทั้ง 2 ขานี้มีความพิเศษตรงที่สามารถใช้งานเป็นขาเชื่อมต่ออุปกรณ์อนุกรมแบบซิงโครนัส ซึ่งแบ่งเป็นระบบ SPI และระบบบัส I ^{2}C จึงทำให้ต้องเพิ่มสายสัญญาณควบคุมอินพุตเพิ่มเข้ามาอีก 1 เส้น เพื่อเลือกสัญญาณอินพุทระหว่าง SPI และบัส วงจรอินพุตบัฟเฟอร์ของขาพอร์ต C นี้เป็นแบบชมิตต์ทริกเกอร์ทั้งหมด ทั้งนี้เพื่อจัดการให้สัญญาณอินพุตที่เข้ามามีความเหมาะสมและสมบูรณ์มากที่สุด และยังคงสามารถรองรับการพูลอัปภายในแบบอัตโนมัติได้

พอร์ต D มี 8 บิต กำหนดชื่อขาเป็น RD0-RD7 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORT D มีแอดเดรสอยู่ที่ 0x08 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISD มีแอดเดรสอยู่ที่ 0x88 (แบงก์ 1) มีขนาด 8 บิต หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้น และในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุต ให้เขียนข้อมูล "0" ไปยังบิตนั้น สำหรับพอร์ต D นี้จะมีเฉพาะในไมโครคอนโทรลเลอร์ PIC รุ่น 40 ขาเท่านั้น สำหรับในอนุกรม PIC16F87x จะมีในเบอร์ PIC16F871, PIC16F874(A) และ PIC16F877(A)

โครงสร้างทางฮาร์ดแวร์ พอร์ต D สามารถใช้งานได้ 2 โหมดคือ เป็นขาพอร์ตอินพุตเอาต์พุตปกติ และเป็นส่วนขยายพอร์ตแบบขนาน (Parallel Slave Port : PSP) สำหรับเชื่อมต่อกับอุปกรณ์ภายนอกที่มีการจัดระบบบัสแบบไมโครโปรเซสเซอร์คือ มีสายข้อมูล 8 เส้น สายสัญญาณควบคุม 3 เส้นคือ สายสัญญาณควบคุมการอ่าน (RD : Read) , เขียน (WR : Write) และเลือกอุปกรณ์ (CS : Chip Select)

ในพอร์ต D ซึ่งมีโครงสร้างเหมือนกันทุกบิต เมื่อทำงานในโหมดพอร์ตอินพุตเอาต์พุตปกติ วงจรอินพุตจะเป็นแบบชมิตต์ทริกเกอร์ แต่เมื่อทำงานในโหมดขยายพอร์ตแบบขนานหรือ PSP วงจรอินพุตจะเปลี่ยนเป็นแบบทีทีแอล การเลือกโหมดทำงานของพอร์ต D เป็นพอร์ตปกติ และถ้าเป็น "I"พอร์ต D จะทำงานในโหมด PSP

พอร์ต E มี 3 บิต กำหนดชื่อขาเป็น RE0-RE2 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORT E มีแอดเดรสอยู่ที่ 0x09 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต แต่ใช้งานเพียง 3 บิตล่างคือ บิต 0 - บิต 2 เท่านั้น ที่เหลือกำหนดให้เป็น "0" ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISE ซึ่งมีแอดเดรสอยู่ที่ 0x89 (แบงก์ 1) มีขนาด 8 บิต โดยใช้สามบิตล่างในการกำหนดทิศทางของพอร์ต E ส่วนที่เหลือใช้ควบคุมการทำงานในโหมด PSP ของพอร์ต D

พอร์ต E สามารถใช้งานเป็นพอร์ตอินพุตเอาต์พุตปกติ, ขาอินพุตอะนาลอกของโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล และขาควบคุมการติดต่อกับอุปกรณ์ภายนอกแบบ PSP ทั้งนี้ขึ้นอยู่กับการกำหนดข้อมูลของรีจิสเตอร์ที่ใช้ควบคุมการทำงานของพอร์ตนี้ เช่นเดียวกับพอร์ต D พอร์ต E จะมีเฉพาะ

ในไมโครคอนโทรลเลอร์ PIC รุ่น 40 ขาเท่านั้น สำหรับในอนุกรม PIC16F87x จะมีในเบอร์ PIC16F871, PIC16F874(A), PIC16F877(A)

รีจิสเตอร์ TRISE เมื่อมีการกำหนดให้พอร์ต D ทำงานในโหมด PSP 4 บิตบนของรีจิสเตอร์ TRISE จะใช้เป็นบิตแสดงสถานะวงจรบัฟเฟอร์ที่ใช้ในการถ่ายทอดข้อมูลของส่วนขยายพอร์ตแบบขนาน หรือ PSP ส่วน 3 บิตล่าวใช้ในการเลือกโหมดการทำงานของพอร์ต D

ถ้าหากพอร์ต E ทำงานเป็นพอร์ตอินพุตเอาต์พุตปกติ 3 บิตล่างรีจิสเตอร์ TRISE จะใช้ในการกำหนดทิศทางการถ่ายทอดสัญญาณข้อมูลของพอร์ต ดังมีรายละเอียดการทำงานของแต่ละบิตของรีจิสเตอร์ TRISE ดังนี้

IBF (Input Buffer Fuli status bit) : บิตแสดงสถานะบัฟเฟอร์ทางอินพุตของ PSP

"0" แสดงว่า ไม่มีข้อมูลรับเข้ามาในบัฟเฟอร์

"1" แสดงว่า มีข้อมูลเข้ามาแล้วและกำลังรอการอ่านจากซีพียู

OBF (Output Buffer Full status bit) : บิตแสดงสถานะบัฟเฟอร์ทางเอาต์พุตของ PSP

"0" แสดงว่า ข้อมูลในบัฟเฟอร์ถูกส่งออกไปแล้ว

"1" แสดงว่า ยังมีข้อมูลเดิมอยู่ในบัฟเฟอร์ทางเอาต์พุต

IBOV (Input Buffer Overflow Detect bit) : บิตแสดงการรับข้อมูลเกินของบัฟเฟอร์อินพุต

"0" แสดงว่า ไม่มีการรับข้อมูลเกินหรือโอเวอร์โฟลวเกินขึ้น

"1" แสดงว่า เกิดการเขียนข้อมูลมายังบัฟเฟอร์อินพุต ทั้งที่ยังคงมีข้อมูลเดิมอยู่

PSPMODE (Parallel Slave Port Mode select bit) : บิตเลือกการทำงานของพอร์ต D และ E

"0" แสดงว่า กำหนดให้ทำงานในโหมดพอร์ตอินพุตเอาต์พุตปกติ

"1" แสดงว่า กำหนดให้ทำงานในโหมด PSP

บิต 3 : ไม่มีการใช้งาน กำหนดค่าเป็น "0"

DIR-RE2 ถึง DIR-RE0 (Direction control bit for RE2-RE0) : บิตควบคุมทิศทางของพอร์ต E เมื่อทำงานในโหมดพอร์ตอินพุตเอาต์พุตปกติ

"0" เป็นเอาต์พุต

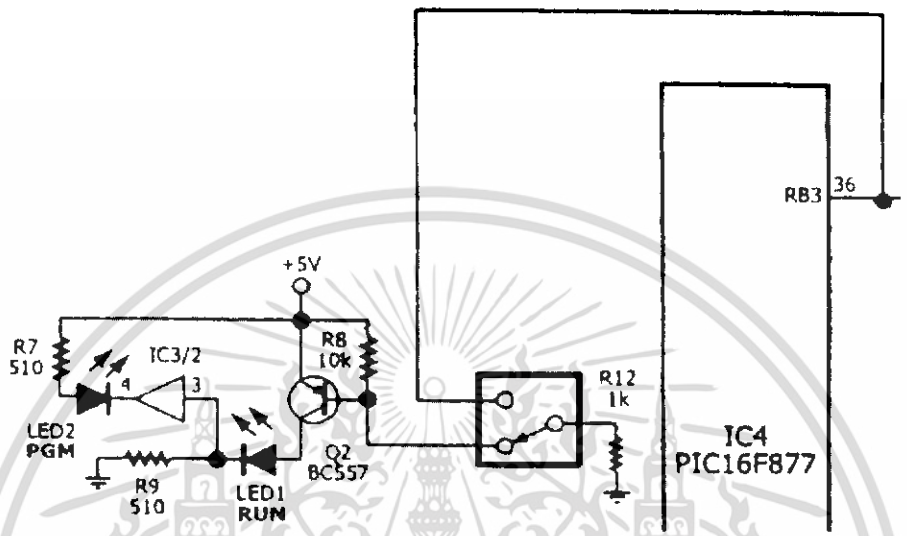
"0" เป็นอินพุต

โครงสร้างทางฮาร์ดแวร์ในพอร์ต E ในบิต RE0-RE2 เมื่อทำงานในโหมดพอร์ตอินพุตเอาต์พุตปกติ จะเห็นได้ว่ามีความคล้ายคลึงกับขาพอร์ตอื่นๆ ของ PIC16F87x ถ้าหากทำงานในโหมดพอร์ตอินพุตเอาต์พุตปกติ วงจรอินพุตของพอร์ตนี้จะเป็นแบบชมิตต์ทริกเกอร์ ในขณะที่หากทำงานในโหมดการแปลงสัญญาณอะนาลอกเป็นดิจิตอล วงจรอินพุตจะเปลี่ยนเป็นแบบทีทีแอล

ถึงแม้ว่าพอร์ต E ใน PIC16F87x มีจำนวนน้อยเพียง 3 บิต แต่สามารถเลือกรูปแบบการทำงานได้มากถึง 3 แบบ คือเป็นพอร์ตอินพุตเอาต์พุตปกติ, อินพุตสำหรับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลขนาด 10 บิตในไมโครคอนโทรลเลอร์ และพอร์ตสัญญาณควบคุมสำหรับติดต่อกับอุปกรณ์ในโหมด PSP ดังนั้นในการเลือกรูปแบบการทำงานต้องระมัดระวังเช่นเดียวกับพอร์ต C ที่ได้กล่าวมาแล้วก่อนหน้านี้

2.4 การทำงานของส่วนแสดงผล

วงจรแสดงโหมด RUN และ PQM โดยหลอด LED 2 หลอด ถ้าหลอดเขียวติดแสดงว่าอยู่ในโหมด RUN ตามปกติ ถ้าหลอดแดงติดก็คืออยู่ในโหมด PQM (โปรแกรม)



รูปที่ 2-34 วงจรส่วนแสดงโหมด

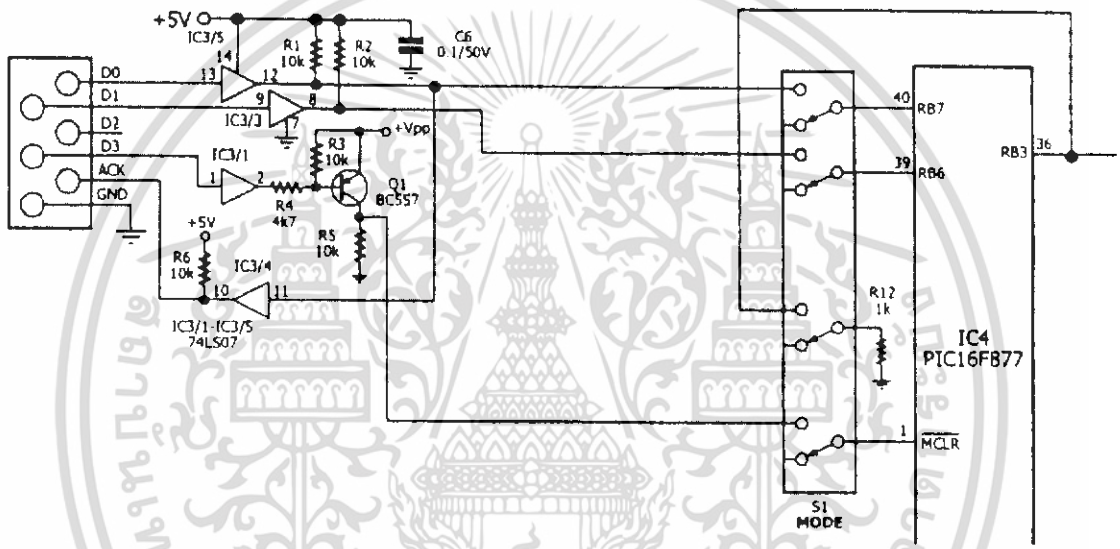
การทำงานส่วนนี้เริ่มจาก S1 อยู่ในโหมด RUN ตามปกติ กระแสจะผ่านขา B ของทรานซิสเตอร์ BC 557 ผ่าน S1 ไปที่ R2 ซึ่งมีค่า 1 kΩ ทำให้กระแสส่วนใหญ่ไหลผ่านขา E ของทรานซิสเตอร์ BC 557 ไปผ่าน LED1 และมี R9 ซึ่งมีค่า 510 Ω จำกัดกระแสที่ผ่าน LED (เมื่อกระแสไหลผ่าน LED ที่เป็นไดโอดประเภทหนึ่งจะเกิดแรงดันตกคร่อม 1.7 V. ซึ่งมากกว่าไดโอดธรรมดา และ LED ขนาด 5 mm จะรับกระแสต่อเนื่องได้สูงสุด 20 mA. ขณะที่ LED ขนาด 2.5 mA. ที่ใช้จะรับกระแสสูงสุดประมาณ 8 mA) กระแสที่ผ่าน R9 510 Ω นั้น จะเกิดแรงดันตกคร่อมเป็นลอจิก1 ให้กับ IC 3/2 ที่ทำงานเป็นบัฟเฟอร์ปล่อยไฟ 5 V. (ลอจิก1) ให้กับขาแคโทดของ LED2 (PQM) เมื่อทั้งขั้ว K และ A ของ LED มีไฟ 5 V. ทั้งคู่ ความต่างศักย์จึงเป็น 0 โวลต์ ไม่มีกระแสไหล ดังนั้น LED2 จะดับ

เมื่อสับ S1 เข้าโหมดโปรแกรม กระแสของขา B ทรานซิสเตอร์จะลงกราวด์ผ่าน S1 ไม่ได้ ไม่มีไฟไหลไป LED1 จึงดับและเกิดลอจิก "0" ด้วย (0 โวลต์หรือกราวด์) ทำให้เกิดความต่างศักย์ที่ LED2 (PQM) ทำให้กระแสไฟไหลเข้า LED ติดสว่าง โดยมี R7 510 Ω จำกัดกระแสไว้ ขณะเดียวกัน S1 จะต่อขา 36 (RB3) ของไมโครคอนโทรเลอร์ลงกราวด์โดย R12 ทำให้เกิดลอจิก "0" ขา RB3 เป็นขา low voltage programming จะใช้ในการโปรแกรมด้วยไฟแรงต่ำ 5 V. ซึ่งเราไม่จำเป็นต้องใช้ จึงนำไปต่อกับ R12 ไว้ตอนโปรแกรมและขานี้จะกลับไปทำงานเป็น Port B เมื่ออยู่ในโหมด RUN ตามปกติ ดังนั้น S1 จึงช่วยตัดต่อในระหว่างการ RUN และโปรแกรม

2.5 การทำงานของส่วนรับการโปรแกรม

ขา ACK และ IC 3/5 (74 LS 07) และ RG ใช้เช็คการต่อของสายดาวน์โหลดระหว่างคอมพิวเตอร์เข้ากับบอร์ด หากสายขาดหรือมีความผิดปกติ โดยถ้าการต่อสายถูกต้อง โปรแกรมที่ใช้เขียนไมโครคอนโทรเลอร์จะเช็คโดยการส่งลอจิก "0" มาตามสาย D0 ถ้าสาย D0 ต่อกับบอร์ดแล้ว ลอจิก "0" จะถูกส่งไปที่บัฟเฟอร์ IC 3/5 และส่งมันกลับมาตามสาย ACK เข้าสู่คอม ดังนั้นโปรแกรมจะทราบได้ทันทีว่าสายต่อถูกต้องหรือไม่ โดยดูจากลอจิก "0" ที่ส่งไปนั้นถูกส่งกลับมาตามสาย ACK หรือไม่

สาย D0, D1 และ Buffer IC 3/5, 3/3 ใช้เป็นขาสัญญาณโปรแกรม ส่วนสาย D3 และอุปกรณ์ที่ต่อกับสายนี้ใช้ทำการ Reset ไมโครคอนโทรลเลอร์ เมื่อทำการโปรแกรมเสร็จแล้ว ก่อนจะอ่านข้อมูลเพื่อทำการเปรียบเทียบกับต้นฉบับ เพื่อตรวจสอบความถูกต้องของการโปรแกรม



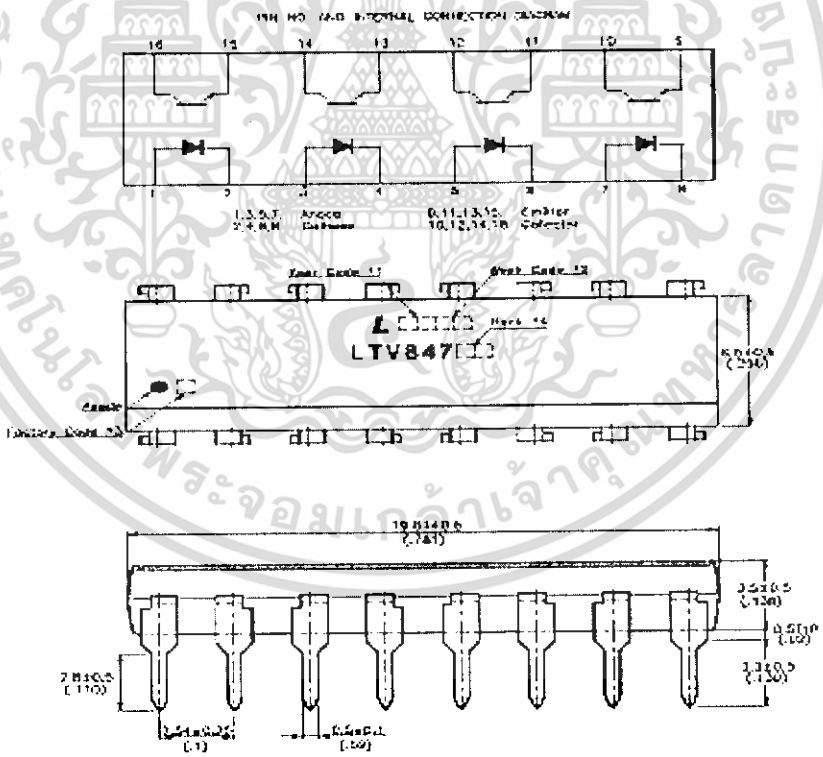
รูปที่ 2-35 วงจรส่วนรับการโปรแกรม

2.6 การทำงานของภาคอินพุต

PLC ปกติจะมีภาคอินพุตหลายแบบให้เลือกติดตั้งใช้งาน ให้เหมาะกับงานแต่ละแบบ เช่นรับค่าแบบอะนาลอก หรือเป็นอินพุตแบบดิจิตอล ส่วนไมโครคอนโทรเลอร์ PIC16F877A ปกติภาคอินพุตก็มี

ให้เลือกใช้ทั้งแบบอะนาลอกและดิจิตอล ตามรายละเอียดในหัวข้อที่2.3.4ทำให้สามารถต่อกับแหล่งกำเนิดสัญญาณได้โดยตรง การเชื่อมต่อโดยตรงนี้จะได้ความสามารถในการตรวจจับสัญญาณสูงสุดที่ไมโครคอนโทรเลอร์ PIC16F877 ทำได้แต่การเชื่อมต่อโดยตรงอาจทำให้ไมโครคอนโทรเลอร์เสียหายได้โดยตรงจากความผิดปกติจากอินพุต ดังนั้นเราจึงมีการสร้างภาคอินพุตเป็นบัฟเฟอร์ป้องกันไมโครคอนโทรเลอร์เสียหาย ภาคอินพุตในส่วนนี้สามารถเลือกว่าจะติดตั้งหรือไม่ก็ได้ หากใช้จะมีข้อดีดังนี้ ทนแรงดันกระชากได้สูงโดยไม่เสียหาย การต่อสลับขั้ว (บวก, ลบ) ได้โดยไม่ทำให้เสียหาย แยกวงจรทางไฟฟ้าระหว่างอินพุตและ PLC ออกจากกัน เมื่อมีแรงดันสูงมากเข้ามาจะชำรุดเฉพาะภาคอินพุต โดยไม่มีผลกระทบต่อภาค CPU และข้อมูลในหน่วยความจำ (ภาคอินพุตราคาถูกกว่าส่วน CPU มาก) ทนสัญญาณรบกวนได้ดีขึ้น ส่วนข้อเสียคือไม่สามารถวัดสัญญาณอะนาลอกได้ ใช้ได้แต่กับลอจิก 0 และ 1 มีการหน่วงเวลาของสัญญาณไปเล็กน้อย (lag) มีผลกระทบจากอุณหภูมิเล็กน้อย ทำให้ความสามารถในการแยกความแตกต่างของลอจิก 0 และ 1ลดลง อินพุตอิมพิแดนซ์ลดลงมาก ไม่สามารถวัดสัญญาณความถี่สูงๆได้เหมือนการต่อตรงเข้ากับไมโครคอนโทรเลอร์

การทำงานเราจะใช้การเชื่อมโยงทางแสงด้วยPhotocoupler ทำให้อินพุตและไมโครคอนโทรเลอร์แยกกันทางไฟฟ้า Photocouplerที่เราใช้คือเบอร์ LTV-847 ของ LITE ON

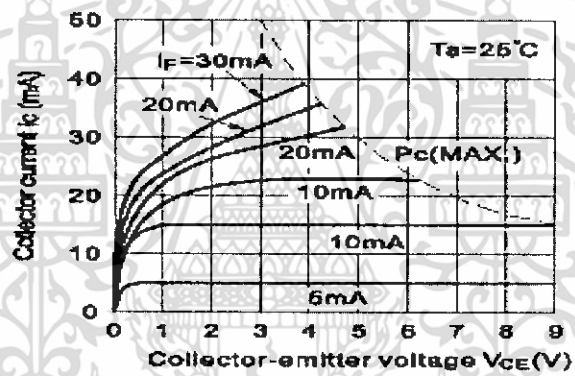


รูปที่ 2-36 Photocoupler LTV-847

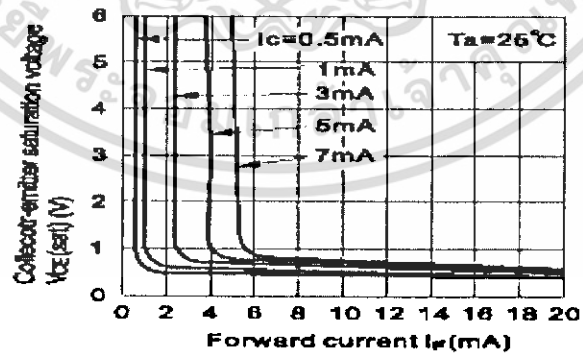
Parameter		Symbol	Rating	Unit
Input	Forward Current	IF	50	mA
	Reverse Voltage	VR	6	V

	Power Dissipation	P	70	mW
Output	Collector-Emitter Voltage	VCEO	35	V
	Emitter-Collector Voltage	VECO	6	V
	Collector Current	IC	50	mA
	Collector Power Dissipation	PC	150	mW
Total Power Dissipation		Ptot	200	mW
Operating Temperature		Topr	-30~+100	°C
Storage Temperature		Tstg	-55~+125	°C
*1.Isolation Voltage		Viso	5	KVrms
*2.Soldering Temperature		Tsol	260	°C

ตารางที่2-1 Absolute Maximum Ratings (Ta=25 C) ของ Photocoupler LTV-847



รูปที่ 2-37 ความสัมพันธ์ระหว่าง I_C และ V_{CE}



รูปที่ 2-38 ความสัมพันธ์ระหว่าง V_{CE} และ I_F

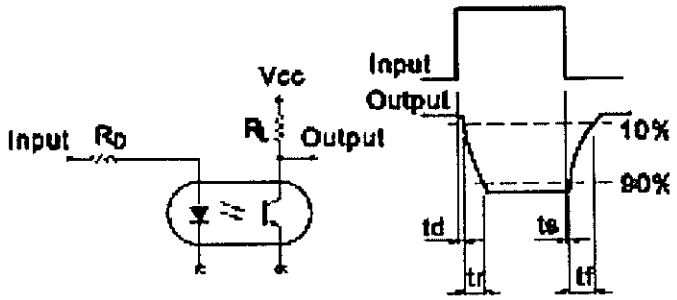
Parameter		Symbol	Min.	Typ.	Max.	Unit	Conditions
Input	Forward Voltage	VF	-	1.2	1.4	V	IF=20mA
	Reverse Current	IR	-	-	10	μA	VR=4V

	Terminal Capacitance	Ct	-	30	250	pF	V=0, f=1KHz
Output	Collector Dark Current	ICEO	-	-	100	nA	VCE=20V
	Collector-Emitter Breakdown Voltage	BVCEO	35	-	-	V	IC=0.1mA
	Emitter-Collector Breakdown Voltage	BVECO	6	-	-	V	IE=10 μA
Transfer Characteristics	Current Transfer Ratio	CTR	50	-	600	%	IF=5mA, VCE=5V
	Collector Current	IC	2.5	-	30	mA	RBE= ∞
	Collector-emitter Saturation Voltage	VCE(sat)	-	0.1	0.2	V	IF=20mA, IC=1mA
	Isolation Resistance	RISO	5×10^{10}	10^{11}	-	Ω	DC500V, 40~60% R.H.
	Floating Capacitance	Cf	-	0.6	1.0	pF	V=0, f=1MHz
	Cut-off Frequency	fc	-	80	-	KHz	VCE=5V, IC=2mA RL=100 Ω, -3dB
	Response Time (Rise)	tr	-	4	18	μs	VCE=2V, IC=2mA RL=100 Ω
	Response Time (Fall)	tf	-	3	18	μs	

ตารางที่2-2 Electrical/Optical Characteristics (Ta=25 C) ของ Photocoupler LTV-847

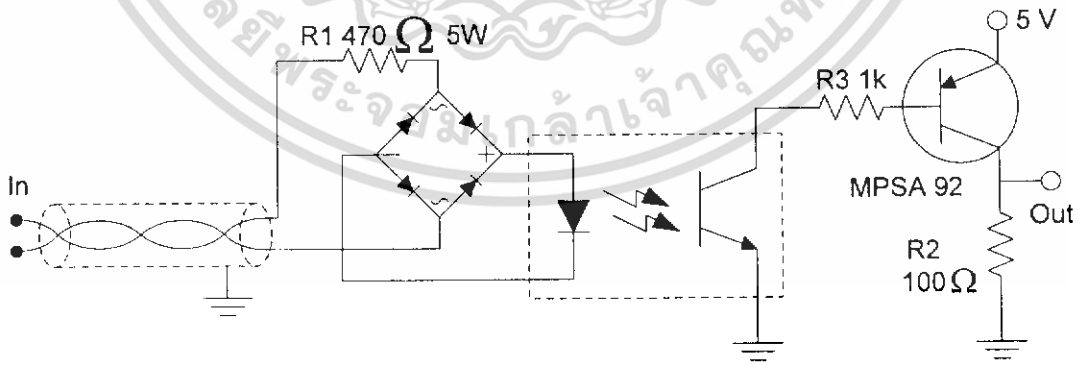
PART NO	DESCRIPTION	PACKAGE	STOCK	PRICE
LTV817C	IC: PHOTOCOUPLER 1 CHANNEL, LITE-ON	PDIP	1602	2.90
LTV827	IC:PHOTOCOUPLER 2 CHANNEL , LITEON DUAL,TRANSISTOR OUTPUT,VCE=55V,IC=50MA	PDIP	421	10.48
LTV847	IC:PHOTOCOUPLER 4 CHANNEL , LITEON QUAD,TRANSISTOR OUTPUT,VCE=55V,IC=50MA	PDIP	108	17.03

รูปที่ 2-39 ราคา Photocoupler LTV-847 (บาท)



รูปที่ 2-40 การหน่วงสัญญาณของ Photocoupler LTV-847

ตามตาราง 2-1 กระแสขับ LED สูงสุดที่ 50 mA เราจะออกแบบให้ภาคอินพุตใช้กับแรงดันได้สูงสุด 26.4 V(24 V + 10%) เพื่อจำกัดกระแสไม่ให้เกิน 50 mA เราต้องใช้ความต้านทาน R1 = V/I = (26.4-0.7-0.7-1.7)/0.05 = 466 Ω ค่าที่มีใกล้เคียงคือ 470 Ω (ต้องใช้ค่ามากกว่าเท่านั้น), (ตัวเลข 0.7 คือแรงดันคร่อมไดโอด ส่วนตัวเลข 1.7 คือแรงดันคร่อม LED) ความร้อนที่ความต้านทาน R1เท่ากับ V^2/R = $(26.4-0.7-0.7-1.7)^2$ /470 = 1.16 W ดังนั้นตัวต้านทานนี้ต้องใช้ขนาด 5 W และเราจะออกแบบให้ให้ภาคอินพุตตรวจจับแรงดันได้ต่ำสุด 3.6 V ที่แรงดันนี้มีกระแสไหล (3.6-0.7-0.7-1.7)/470 = 1 mA(กระแสนี้คือ If ตามรูป 2-35) เมื่อให้แรงดัน Collector – Emitter (Vce ตามรูป 2-36) จะได้ Collector current (Ic) 0.3 mA ภาคอินพุตของไมโครคอนโทรเลอร์จะจัดแรงดันสูงกว่า 2.6 V เป็นลอจิก "1" ทรานซิสเตอร์ MPSA 92 มีอัตราขยายกระแสประมาณ 82 เท่า เมื่อมีกระแส 0.3 mA ผ่านจากขา Base ไปลงกราวด์ทาง Photocoupler (ดูรูป2-39 ประกอบ) จึงมีกระแส 0.3 x 82 = 24.6 mA ไหลผ่านขา Collector และผ่าน R2 เมื่อเราต้องการให้แรงดันคร่อม R2 เป็น 2.6 V (เพื่อให้เป็นลอจิก "1" พอดี) ต้องใช้ R2 = V/I = 2.6/0.0246 = 100 Ω และสรุปคุณสมบัติภาคอินพุตเป็นตารางที่ 2-3



รูปที่ 2-41 วงจรภาคอินพุต

รายการ	รายละเอียดทางเทคนิค
แรงดันอินพุต	24 VDC +10% / -15%
อินพุตอิมพีแดนซ์	470 Ω
กระแสด้านอินพุต(ทั่วไป)	50 mA
แรงดันตรวจจับต่ำสุด	3.6 V
อุณหภูมิทำงาน	-30°C ~ +100°C
ความต้านทานฉนวน(ระหว่างอินพุตและ CPU)	5000 Vrms
Terminal Capacitance	30 - 250 pF
Cut-off Frequency	80 kHz , -3dB
เวลาตอบสนอง (ขาขึ้น Rise)	4 - 18 µs
เวลาตอบสนอง (ขาลง Fall)	3 - 18 µs

ตารางที่ 2-3 คุณสมบัติของภาคอินพุต

บทที่ 3
การออกแบบซอฟต์แวร์

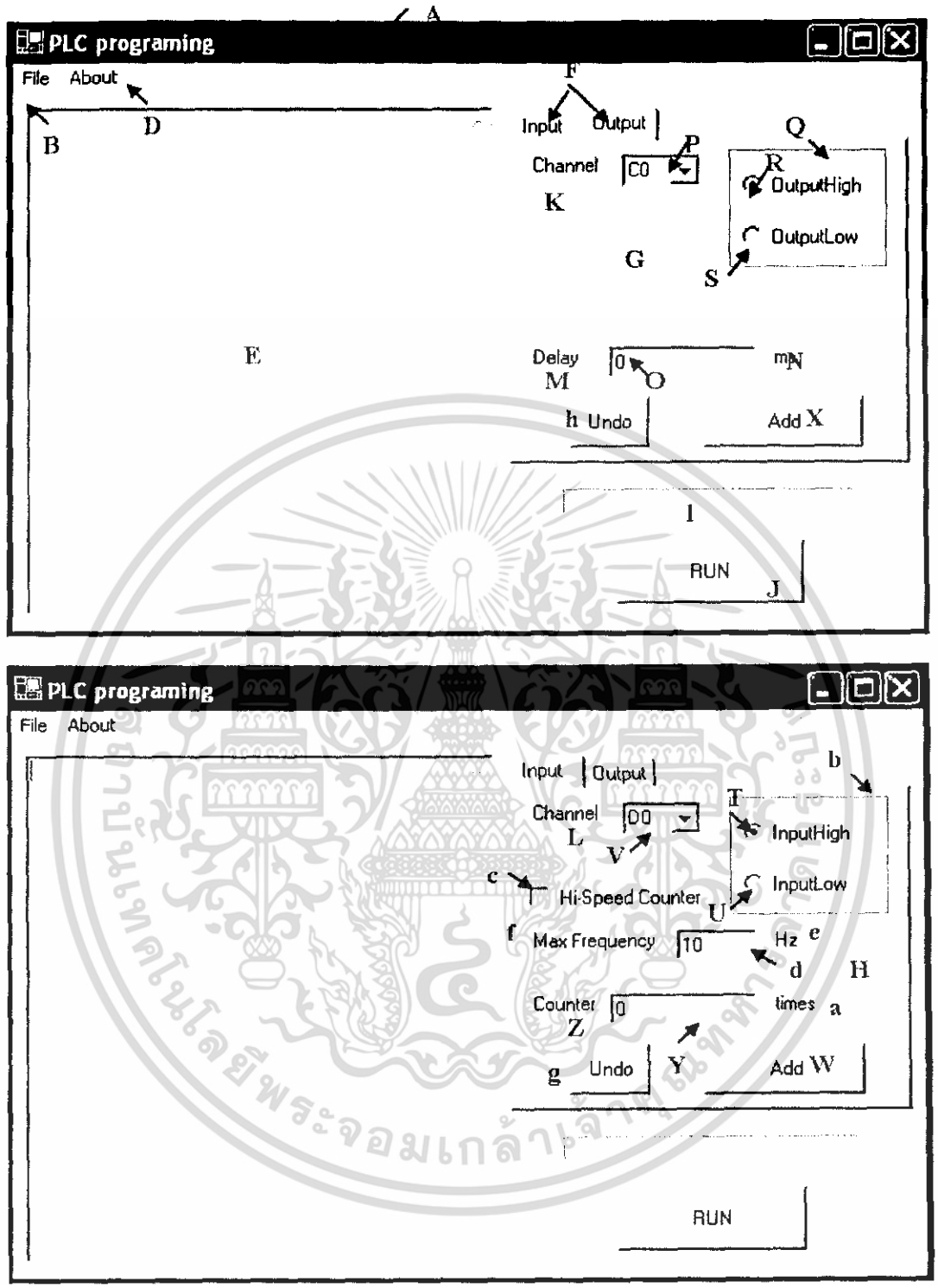
3.1 ส่วนติดต่อผู้ใช้ (User interface)

รายการ	รายละเอียดทางเทคนิค
แรงดันอินพุต	24 VDC +10% / -15%
อินพุตอิมพีแดนซ์	470 Ω
กระแสด้านอินพุต(ทั่วไป)	50 mA
แรงดันตรวจจับต่ำสุด	3.6 V
อุณหภูมิทำงาน	-30°C ~ +100°C
ความต้านทานฉนวน(ระหว่างอินพุตและ CPU)	5000 Vrms
Terminal Capacitance	30 - 250 pF
Cut-off Frequency	80 kHz , -3dB
เวลาตอบสนอง (ขาขึ้น Rise)	4 - 18 μs
เวลาตอบสนอง (ขาลง Fall)	3 - 18 μs

ตารางที่ 2-3 คุณสมบัติของภาคอินพุต

บทที่ 3
การออกแบบซอฟต์แวร์

3.1 ส่วนติดต่อผู้ใช้ (User interface)



รูปที่ 3-1 แสดงส่วนประกอบของ ส่วนติดต่อผู้ใช้

	ชื่อ	หน้าที่
A	From1	เป็นกรอบสำหรับ Object ในโปรแกรม
B	menuItem1	เป็น เมนู File ซึ่งบรรจุ menuItem2 ไว้ภายในจะปรากฏ เมื่อ click down
C	menuItem2	เป็น เมนู Exit ใช้ปิดโปรแกรม
D	menuItem3	เป็น เมนู About แสดงรายละเอียดและเวอร์ชั่นของโปรแกรม

E	txtOutput	แสดงคำสั่งที่ผู้ใช้ทำไปทั้งหมด
F	tabControl1	บรรจุ tabPage1,2 ให้ผู้ใช้ไม่ไปใช้ Out และ In ซ้อนกันในคำสั่งเดียวกัน
G	tabPage2	บรรจุปุ่มต่างๆที่ใช้เกี่ยวกับ output ของ PLC
H	tabPage1	บรรจุปุ่มต่างๆที่ใช้เกี่ยวกับ input ของ PLC
I	progressBar1	แสดงความคืบหน้าขณะโปรแกรมทำงาน
J	btnRUN	ปุ่ม RUN กดสั่งให้โปรแกรมแปลCode สำหรับ PLC ขึ้นมาจากตัวเลือกทั้งหลายที่ผู้ใช้เลือกไปทั้งหมด
K	label4	แสดงคำว่า Channel เพื่อใช้สื่อความหมายของกล่องตัวเลือกข้างๆว่ามีไว้ใช้เลือกช่องของ PLC
L	label1	แสดงคำว่า Channel เพื่อใช้สื่อความหมายของกล่องตัวเลือกข้างๆว่ามีไว้ใช้เลือกช่องของ PLC แม้ว่าจะแสดงคำว่า Channel และอยู่ตำแหน่งเดียวกับ label4 แต่มันอยู่บนคนละ tabPage กันจึงเป็นคนละตัวและคนละชื่อกัน
M	label3	แสดงคำว่า Delay เพื่อใช้สื่อความหมายของกล่องตัวเลือกข้างๆว่ามีไว้ใช้กรอกเวลาที่ต้องการให้ Timer
N	label6	แสดงคำว่า ms เพื่อใช้บอกว่าเวลาที่กรอกเป็น ms
O	textdelay	เป็นช่องกรอกเวลาที่ต้องการให้ Timer
P	comboBox1	บรรจุชื่อ Port Output ทั้งหมดที่ PLC มีให้ผู้ใช้
Q	groupBox1	บรรจุกลุ่มของตัวเลือก Output เพื่อแสดงว่าอยู่ในกลุ่มเดียวกัน
R	radioButtom1	เป็นช่องตัวเลือกว่าจะให้ Output เป็น Hi หรือ Low เนื่องจาก Output เป็นทั้ง Hi และ Low พร้อมกันไม่ได้ ดังนั้นปุ่มนี้จะต้องทำงานสัมพันธ์กับปุ่ม radioButtom2(OutputLow) ด้านล่าง ถ้ากด Hi แล้ว Low จะต้องดับไปหรือ เลือก Low แล้ว Hi จะต้องดับไปแทน
S	radioButtom2	ใช้เลือก Output ให้เป็น Low ติดดับสลับกับ radioButtom1 ไม่สามารถเลือกทั้ง radioButtom1และ radioButtom2 พร้อมกันได้
T	rdo3	เป็นช่องตัวเลือกว่าจะให้ Input จับสัญญาณจาก การเปลี่ยนแปลง Low ไป เป็น Hi จึงจะทำงาน ( ) เนื่องการเลือกปุ่ม rdo3 จะให้ผลตรงข้ามกับ rdo4 ดั้งนั้นสถานการณ์ทั้ง 2 จึงเกิดพร้อมกันไม่ได้ ปุ่ม rdo3 และ rdo4 จะสร้างมาให้เลือกพร้อมกันไม่ได้
U	rdo4	เป็นช่องตัวเลือกว่าจะให้ Input จับสัญญาณจาก การเปลี่ยนแปลง Hi ไป เป็น Low จึงจะทำงาน ( )
V	conboBox2	บรรจุชื่อ Port Input ทั้งหมดของ PLC ที่ผู้ใช้เลือกใช้ได้

W	btnAddIn	เพิ่มคำสั่งเกี่ยวกับการ Input ของ Port PLC เมื่อกด Add รายการคำสั่งที่เลือกไว้จะถูกนำไปบันทึกเพิ่มในจอ txtOutput
X	btnAddOut	เพิ่มคำสั่งเกี่ยวกับการ Output ของ Port PLC เมื่อกด Add รายการคำสั่งที่เลือกไว้จะถูกนำไปบันทึกเพิ่มในจอ txtOutput
Y	txtCounter	เป็นช่องกรอกจำนวนครั้งที่ต้องการให้ Timer นับ
Z	label2	แสดงคำว่า Counter เพื่อใช้สื่อความหมายของกล่องตัวเลือกข้างๆว่ามีไว้ใช้กรอกจำนวนครั้งที่ต้องการให้ Timer นับ
a	label5	แสดงคำว่า Times เพื่อใช้บอกว่าหน่วยเป็น ครั้ง
b	groupBox2	บรรจุกลุ่มของตัวเลือก Input เพื่อแสดงว่าอยู่ในกลุ่มเดียวกัน
c	checkBox1	ใช้เคาน์เตอร์ความเร็วสูง
d	textBox1	เป็นช่องกรอกความถี่สูงสุดที่จะใช้กับ Counter
e	label8	แสดงคำว่า Hz เพื่อใช้บอกว่าความถี่ที่ใช้หน่วยเป็น Hz
f	label7	แสดงคำว่า Max Frequency เพื่อใช้สื่อความหมายของกล่องตัวเลือกข้างๆว่ามีไว้ใช้กรอกความถี่สูงสุดที่จะใช้กับ Counter
g	UndoO	ย้อนกลับไปแก้ไขคำสั่งที่ Add ไปแล้วได้ 1 ครั้ง
h	UndoI	ย้อนกลับไปแก้ไขคำสั่งที่ Add ไปแล้วได้ 1 ครั้ง

ตารางที่ 3-1 รายชื่อและหน้าที่ของ Object

Object A (From1) สร้างโดยพิมพ์ using System.Windows.Forms; ไปที่ส่วนหัวของโค้ดประโยคดังกล่าวประกาศว่าโค้ดใดๆภายใต้ Namespace เดียวกับ คำสั่ง using ด้านบนสามารถเข้าถึงหรือใช้โค้ด ใน Namespace Forms (ซึ่งบรรจุอยู่ใน Global Namespace Windows และบรรจุอยู่ใน Global Namespace System อีกที) ได้โดยไม่ต้องมาอ้างถึงอีก (การใช้ using จะทำให้ใช้โค้ดได้โดยไม่ต้องอ้างถึงยาวๆอีก) ฟังชั่นใดๆที่จำเป็นในการสร้างวินโดวส์แอปพลิเคชั่น ถูกจัดเตรียมไว้ให้แล้ว หลังจากนั้นพิมพ์

```
namespace WindowsApplication2{
        public class Form1 : System.Windows.Forms.Form
        { ..... . . ...
//
                        // Form1
                        //
                        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
```

```
this.ClientSize = new System.Drawing.Size(544, 337);
this.Controls.Add(this.progressBar1);
this.Controls.Add(this.btnRUN);
this.Controls.Add(this.tabControl1);
this.Controls.Add(this.txtOutput);
this.Menu = this.mainMenu1;
this.Name = "Form1";
this.Text = "PLC programing";
this.tabControl1.ResumeLayout(false);
this.tabPage1.ResumeLayout(false);
this.groupBox2.ResumeLayout(false);
this.tabPage2.ResumeLayout(false);
this.groupBox1.ResumeLayout(false);
this.ResumeLayout(false);
```

WindowsApplication2 มาจากชื่อ Folder ที่บรรจุโปรเจคอยู่ ซึ่ง C# สร้างขึ้นตอน new project ชื่อ Folder นี้เป็นชื่อเดียวกับชื่อแอปพลิเคชั่น แล้วต้องไปกำหนด Properties ของ class Form1 ตามนี้ Size: 552, 392 Text: PLC programming

(mainMenu1) ที่ต้องทำก็มีประกาศ private System.Windows.Forms.MainMenu mainMenu1; // mainMenu1

```
//
this.mainMenu1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {this.menuItem1,
this.menuItem3});
```

ใต้ class Form1 ไม่ต้องกำหนด Properties ให้เพราะใช้คุณสมบัติ DinamicProperties ที่จะไปกำหนดตามจำนวน MenuItem ที่จะมาอยู่ด้วย

Object B (menuItem1) ที่ต้องทำก็มีประกาศ private System.Windows.Forms.MenuItem menuItem1;

```
// menuItem1
//
this.menuItem1.Index = 0;
```

```
this.menuItem1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {

this.menuItem2});
this.menuItem1.Text = "File";
```

ใต้ class Form1 กำหนด Properties ชื่อปุ่มเป็น Text: File และมี event ด้วยคือกดแล้วจะมี menuItem2 (Exit) เลื่อนลงมาซึ่งต้องไปทำใน Form [Design]*

Object C (menuItem2) ที่ต้องทำก็มีประกาศ private System.Windows.Forms.MenuItem menuItem2; ใต้ class Form1 กำหนด Properties ชื่อปุ่มเป็น Text: Exit และมี event ด้วยเมื่อ Click จะเป็นการปิดโปรแกรม เราใช้ event Click โดย พิมพ์ private void menuItem2_Click(object sender, System.EventArgs e)

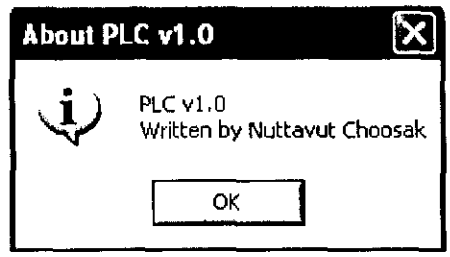
```
{
    Application.Exit();
}
// menuItem2
    //
    this.menuItem2.Index = 0;
    this.menuItem2.Text = "Exit";
    this.menuItem2.Click += new System.EventHandler(this.menuItem2_Click);
```

ที่ใต้ namespace WindowsApplication2 คำสั่งด้านบนเป็น Method ของobject menuItem2 ทำงานคล้ายฟังชั่น คำว่า Method เป็นศัพท์เฉพาะที่ใช้ในเรื่องการโปรแกรมเชิงวัตถุ(OOP) มันคล้ายกับฟังชั่นแต่อันที่จริงมันคือคำสั่งที่อยู่ใน Object ในกรณีคำสั่ง Method ด้านบน นี้เป็นของ Object menuItem2 ที่เป็นปุ่มในเมนูของโปรแกรมเรา Method นี้มีชื่อว่า menuItem2_Click สรุปง่ายๆว่าเมื่อปุ่ม menuItem2(Exit) ถูกกด Method (หรือฟังชั่น)นี้จะถูกเรียกขึ้นมา เนื่องจากคำว่าvoid ที่อยู่ใน Method (หรือฟังชั่น)นี้กำหนดว่า เมื่อ Method (หรือฟังชั่น)นี้ทำงานจบมันจะไม่ Return ค่าใดกลับ โค้ดที่บรรจุอยู่ใน {....} คือสิ่งที่เราอยากให้มันทำ ซึ่งคือการปิดโปรแกรม คำสั่ง Application.Exit(); จะทำให้โปรแกรมปิดตัวลง

Object D (menuItem3) ที่ต้องทำก็มีประกาศ private System.Windows.Forms.MenuItem menuItem3; ใต้ class Form1 กำหนด Properties ชื่อปุ่มเป็น Text: About และมี event ด้วยเมื่อ click จะเป็นการเรียกหน้าต่าง Information ขึ้นมาแสดงรายละเอียดของโปรแกรม เราใช้ event Click โดย พิมพ์

```
private void menuItem3_Click(object sender, System.EventArgs e)
        {
            MessageBox.Show ("PLC v1.0\r\nWritten by Nuttavut Choosak", "About
PLC v1.0",
                MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }
// menuItem3
        //
        this.menuItem3.Index = 1;
        this.menuItem3.Text = "About";
        this.menuItem3.Click += new System.EventHandler(this.menuItem3_Click);
```

คล้ายกับปุ่มก่อนหน้าเมื่อเกิด Event Click ที่ปุ่ม Method menuItem3_Click จะถูกเรียกขึ้นมาและทำงานตามคำสั่งที่บรรจุอยู่ในวงเล็บปีกกา คำสั่งนั้นเป็นการเรียกหน้าต่าง Information ขึ้นมาแสดงรายละเอียดของโปรแกรม คำสั่งนี้มีรูปแบบดังนี้ MessageBox.Show (<Text>, <Title>, <MessageBoxButtons>,< MessageBoxIcon>); ในกรณีของเรา <Text>จะแทนด้วย ข้อมูล เวอร์ชั่นโปรแกรม และผู้เขียนโปรแกรม ต้องมี "....." ล้อมข้อความเพื่อให้โปรแกรมรู้ว่าในนั้นเป็นข้อความที่ไม่ต้องนำไป compile(แปล)ด้วย และส่วน\r\n ที่แทรกอยู่คือ สายลำดับหลีกเลี่ยงตามตารางยูนิโค้ด \r จะแสดงอักขระ Carriage return ส่วน \n จะทำให้ขึ้นบรรทัดใหม่ <Title>จะแทนด้วยชื่อหน้าต่าง MessageBox และต้องมี "....." ล้อมข้อความเช่นกัน <MessageBoxButtons> แทนด้วย MessageBoxButtons.OK จะทำให้เกิดปุ่ม OK ใน MessageBox
< MessageBoxIcon>จะแทนด้วย MessageBoxIcon.Asterisk คำสั่งนี้ทำให้แสดงรูป icon Information สรุปคือเมื่อปุ่ม About ถูกกด จะแสดงหน้าต่างดังรูป



รูปที่ 3-2 MessageBox

Object E (txtOutput) ที่ต้องทำก็มีประกาศ private System.Windows.TextBox txtOutput; menuItem2;

```
// txtOutput
//
this.txtOutput.Location = new System.Drawing.Point(8, 8);
this.txtOutput.Multiline = true;
this.txtOutput.Name = "txtOutput";
this.txtOutput.ReadOnly = true;
this.txtOutput.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.txtOutput.Size = new System.Drawing.Size(280, 320);
this.txtOutput.TabIndex = 0;
this.txtOutput.Text = "";
```

ใต้ class Form1

txtOutput เป็น Object หนึ่ง ซึ่งเราจะส่งข้อความมาแสดงที่นี่ เมื่อผู้ใช้กดปุ่ม Add ดังนั้นเราจะไปพูดถึงอีกทีที่ปุ่ม Add และต้องตั้ง Properties Location: 8, 8 Size: 280, 320

Object F (tabControl1)ที่ต้องทำก็มีประกาศ private System.Windows.Forms.TabControl tabControl1;

```
// tabControl1
//
this.tabControl1.Controls.Add(this.tabPage1);
this.tabControl1.Controls.Add(this.tabPage2);
this.tabControl1.Location = new System.Drawing.Point(296, 8);
this.tabControl1.Name = "tabControl1";
this.tabControl1.SelectedIndex = 0;
this.tabControl1.Size = new System.Drawing.Size(240, 216);
this.tabControl1.TabIndex = 1;
```

ใต้ class Form1

และต้องตั้ง Properties Location: 296, 8 Size: 240, 216 TabIndex: 1 Padding: 6, 3 ItemSize: 42, 18

Object G (tabPage2)ที่ต้องทำก็มีประกาศ private System.Windows.Forms. TabPage tabPage2;

// tabPage2

```
//
this.tabPage2.Controls.Add(this.groupBox1);
this.tabPage2.Controls.Add(this.label6);
this.tabPage2.Controls.Add(this.btnAddOut);
this.tabPage2.Controls.Add(this.textDelay);
this.tabPage2.Controls.Add(this.label3);
this.tabPage2.Controls.Add(this.comboBox1);
this.tabPage2.Controls.Add(this.label4);
this.tabPage2.Location = new System.Drawing.Point(4, 22);
this.tabPage2.Name = "tabPage2";
this.tabPage2.Size = new System.Drawing.Size(232, 190);
this.tabPage2.TabIndex = 1;
this.tabPage2.Text = "Output";
```

ใต้ class Form1

และต้องตั้ง Properties Text: Input Location: 4, 22 Size: 232, 190

Object H (tabPage1)ที่ต้องทำก็มีประกาศ private System.Windows.Forms. TabPage tabPage1;

ใต้ class Form1

tabPage1

```
//
this.tabPage1.Controls.Add(this.groupBox2);
this.tabPage1.Controls.Add(this.label5);
this.tabPage1.Controls.Add(this.btnAddIn);
this.tabPage1.Controls.Add(this.txtCounter);
this.tabPage1.Controls.Add(this.label2);
this.tabPage1.Controls.Add(this.comboBox2);
this.tabPage1.Controls.Add(this.label1);
this.tabPage1.Location = new System.Drawing.Point(4, 22);
```

```
this.tabPage1.Name = "tabPage1";
this.tabPage1.Size = new System.Drawing.Size(232, 190);
this.tabPage1.TabIndex = 0;
this.tabPage1.Text = "Input";
```

และต้องตั้ง Properties Text: Input Location: 4, 22 Size: 232, 190

Object I (progressBar1) ที่ต้องทำก็มีประกาศ private System.Windows.Forms. ProgressBar progressBar1;ใต้ class Form1

```
// progressBar1
//
this.progressBar1.Location = new System.Drawing.Point(328, 240);
this.progressBar1.Name = "progressBar1";
this.progressBar1.Size = new System.Drawing.Size(176, 16);
this.progressBar1.TabIndex = 3;
```

และต้องตั้ง Properties TabIndex: 3 Locations: 328, 240 Size: 176, 16

Object J (btnRUN)ที่ต้องทำก็มีประกาศ private System.Windows.Forms. Button btnRUN;ใต้ class Form1 และต้องตั้ง Properties Text: RUN TabIndex: 2 Location: 360, 272 Size 112, 40 และมี event ด้วยเมื่อ click จะเป็นการรันของโปรแกรมดังนั้นหลังจากกดปุ่มนี้จะเกิดกระบวนการเช็คต่างๆและเงื่อนไขการเรียกฟังชั่นมากมายซ้อนกันอยู่และทีการสร้างไปถึงการเขียนไฟล์เกิดขึ้นด้วยดังนั้นเราจะยกการอธิบายเหตุการณ์หลังกดปุ่มไปไว้ในส่วนAlgorithm ทีหลัง เราใช้ event Click โดย พิมพ์

```
private void btnRUN_Click(object sender, System.EventArgs e)
        {
            FileStream aFile = new FileStream ("PLC.c",FileMode.Create);
            StreamWriter sw = new StreamWriter(aFile);
            sw.WriteLine("#define _PIC_16F877A_\r\n");
            sw.WriteLine("#include <16F877A.h>\r\n");
            sw.WriteLine("#define CLOCK_SP 20000000\r\n");
            sw.WriteLine("#fuses HS\r\n");
            sw.WriteLine("#fuses NOLVP,NOWDT\r\n");
```

```
sw.WriteLine("#fuses NOPROTECT\r\n");
sw.WriteLine("#use delay (clock=20000000)\r\n");
sw.WriteLine("#use fast_io(B)\r\n");
sw.WriteLine("#use fast_io(C)\r\n");
sw.WriteLine("void main() {\r\n");
sw.WriteLine("int i;\r\n");
sw.WriteLine("set_tris_b(0xFF);\r\n");
sw.WriteLine("set_tris_c(0x00);\r\n");
sw.WriteLine("set_tris_d(0xFF);\r\n");
sw.WriteLine("while (TRUE) {\r\n");
sw.WriteLine("{0}\r\n",Code);
sw.WriteLine("}");
sw.WriteLine("}");
sw.Close();

progressBar1.Maximum = 200000;


for(int i = 1; i < 200000 ; ++i)
{
    progressBar1.Value += 1;
}


MessageBox.Show ("The compilation are complete", "MessageBox",
    MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

}

// btnRUN

//

this.btnRUN.Location = new System.Drawing.Point(360, 272);
this.btnRUN.Name = "btnRUN";
this.btnRUN.Size = new System.Drawing.Size(112, 40);
```

```
this.btnRUN.TabIndex = 2;
this.btnRUN.Text = "RUN";
this.btnRUN.Click += new System.EventHandler(this.btnRUN_Click);
```

Object K (label4) ที่ต้องทำก็มีประกาศprivate System.Windows.Forms .Label label4;ใต้ class Form1

```
// label4
//
this.label4.Location = new System.Drawing.Point(8, 16);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(48, 16);
this.label4.TabIndex = 7;
this.label4.Text = "Channel";
```

และต้องตั้ง Properties Location: 8, 16 Text: Channel Size: 240, 216 TabIndex: 7 ItemSize: 48, 16

Object L (label1) ที่ต้องทำก็มีประกาศ private System.Windows.Forms .Label label4;ใต้ class Form1

```
// label1
//
this.label1.Location = new System.Drawing.Point(8, 16);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(48, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Channel";
```

และต้องตั้ง Properties Location: 8, 16 Text: Channel Size: 240, 216 TabIndex: 7 ItemSize: 48, 16

Object M (label3) ที่ต้องทำก็มีประกาศprivate System.Windows.Forms .Label label4;ใต้ class Form1

// label3

```
//
this.label3.Location = new System.Drawing.Point(8, 104);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(48, 16);
this.label3.TabIndex = 11;
this.label3.Text = "Delay";
```

และต้องตั้ง Properties Location: 8, 104 Text: Delay Size: 240, 216 TabIndex: 11 ItemSize: 48, 16

Object N (label6) ที่ต้องทำก็มีประกาศprivate System.Windows.Forms .Label label4;ใต้ class Form1

// label6

```
//
this.label6.Location = new System.Drawing.Point(152, 104);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(48, 16);
this.label6.TabIndex = 14;
this.label6.Text = "ms";
```

และต้องตั้ง Properties Location: 152, 104 Text: ms Size: 240, 216 TabIndex: 14 ItemSize: 48, 16

Object O (textdelay)ที่ต้องทำก็มีประกาศprivate System.Windows.Forms TextBox textDelay;menuItem2; ใต้ class Form1

// textDelay

```
this.textDelay.Location = new System.Drawing.Point(56, 104);
this.textDelay.Name = "textDelay";
this.textDelay.Size = new System.Drawing.Size(88, 20);
this.textDelay.TabIndex = 12;
this.textDelay.Text = "0";
```

txtDelay เป็น Object หนึ่งเป็นช่องกรอกเวลาที่ต้องการให้PLC แต้เราไม่ได้ส่งข้อความมาแสดงที่นี่เหมือน Odject TextOutput ทั้งสองเป็น Object แบบ TextBox เหมือนกันแต่ใช้คนละ Method กัน TextOutput เราใช้แสดงบันทึกคำสั่งที่ผู้ใช้สั่งไป แต่txtDelay เราใช้รับตัวเลขจากผู้ใช้ด้วย Method this.textDelay.Text ส่วนคำว่า textDelay เป็นชื่อ Object ที่ต้องการ คำสั่งนี้จะอ่านค่าใน TextBox ส่งกลับมาเป็น String ไม่ใช้ค่าตัวเลข ดังนั้นเมื่อเราจะนำไปสั่ง PLC เราต้องแปลง String นี้เป็น Integer ก่อน ในภาษา C# การแปลงชนิดข้อมูลใช้คนละวิธีกับ ภาษาC โดย ภาษา C# มีคำสั่งเฉพาะที่ใช้แปลงชนิดข้อมูลโดยตรง คำสั่งยาวกว่ามากภาษาC มาก แต่เข้าใจได้โดยง่ายและปลอดภัยกว่ามากการแปลงค่าแบบนี้เรียกว่า การแปลงค่าเชิงประจักษ์ (Explicit conversion) ภาษาC# เองก็มีวิธีการแปลงชนิดข้อมูลง่ายๆแบบภาษาC เช่นกัน แปลงค่าแบบนี้เรียกว่า การแปลงค่าโดยนัย (Implicit conversion) อาจมีปัญหาได้ถ้าเราเปลี่ยนชนิดข้อมูลที่เก็บค่าได้มากกว่าไปสู่ ชนิดข้อมูลที่เก็บค่าได้น้อยกว่า เพราะค่าอาจผิดพลาดไปโดยไม่มีการแจ้งเตือน การผิดพลาดชนิดนี้เรียกว่าการเกิด Overflow อย่างไรก็ตามโปรแกรมในขั้นแรกของเราใช้การแปลงค่าโดยนัยและไม่มีการเช็คว่าผู้ใช้ป้อนตัวเลขหรืออักษรเข้ามาด้วยหรือไม่ ข้อความใดๆหรือตัวเลขแม้กระทั่งจุดทศนิยมหรือเว้นวรรคจะถูกแปลเป็น ค่าตัวเลขทั้งหมด ความผิดพลาดนี้ไม่ทำให้โปรแกรมหยุดทำงาน แต่การหน่วงเวลาของ PLC จะไม่ถูกต้อง ดังนั้นในขั้นทดลองนี้ถ้ากรอกเลขถูกโปรแกรมก็จะทำงานได้ถูกต้องแน่นอน และ Object นี้ต้องตั้ง Properties Location: 56, 104 Size: 88, 20

ต่อมาในเวอร์ชั่นถัดมาได้มีการป้องกัน ดังที่บอกไว้แล้วโดยปกติช่องกรอกตัวเลขนี้ห้ามผู้ใช้กรอกข้อความ ตัวเลขที่ติดลบ หรือทศนิยมลงไป สรุปคือกดได้แค่เลข 0 – 9 เท่านั้น ตามรหัส ASCII (American Standard Code for Information Interchange) รหัสมาตรฐานที่ใช้แทนตัวอักขระต่างๆ ค่าแอสกีสำหรับตัวอักษร 1 – 9 นั้นอยู่ระหว่าง 48 – 57 โค้ดที่เราเพิ่มเข้าไปจะไม่ยอมให้ผู้ใช้พิมพ์สิ่งที่ไม่ใช่ตัวเลขลงไป ในโค้ดจะเห็นว่าเราอนุญาตเพิ่มขึ้นมาอีกค่าคือ 8 เพราะค่าแอสกีของ 8 คือปุ่ม Backspace ที่ใช้ลบสิ่งที่พิมพ์ผิด เราจึงอนุญาตสำหรับปุ่มนี้ด้วย โค้ดส่วนนี้แยกมาต่างหากข้างล่าง

```
private void txtCounter_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
        {
                if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar !=8)
                        e.Handled = true;
        }
```

Object P (comboBox1) ที่ต้องทำก็มีประกาศ private System.Windows.Forms.ComboBox comboBox1;

```
//
// comboBox1
//
this.comboBox1.Items.AddRange(new object[] {
```

```
                "C0",
                "C1",
                "C2",
                "C3",
                "C4",
                "C5",
                "C6",
                "C7"}
);
                this.comboBox1.Location = new System.Drawing.Point(64, 16);
                this.comboBox1.MaxDropDownItems = 12;
                this.comboBox1.Name = "comboBox1";
                this.comboBox1.Size = new System.Drawing.Size(48, 21);
                this.comboBox1.TabIndex = 8;
                this.comboBox1.Text = "C0";
```

ใต้ class Form1

Object นี้เป็นกล่องตัวเลือกที่บังคับให้ผู้ใช้เลือก Port ได้เฉพาะที่มีให้ดังนั้นจึงไม่มีปัญหาให้โปรแกรมผิดพลาด C0 C1 C2 C3 C4 C5 C6 C7 เป็น Port (PLC) ที่โปรแกรมอนุญาตให้ใช้ได้

Objects หลังจากนี้ลงไปจะเป็นชนิดที่เคยอธิบายไปแล้วดังนั้นจึงขอละคำอธิบายส่วนที่ซ้ำซ้อนไป

Object Q (groupBox1) สร้างโดยพิมพ์โค้ดตามนี้

```
// groupBox1
                this.groupBox1.Controls.Add(this.radioButton1);
                this.groupBox1.Controls.Add(this.radioButton2);
                this.groupBox1.Location = new System.Drawing.Point(128, 8);
                this.groupBox1.Name = "groupBox1";
                this.groupBox1.Size = new System.Drawing.Size(96, 80);
                this.groupBox1.TabIndex = 15;
                this.groupBox1.TabStop = false;
```

Object R (radioButtom1) สร้างโดยพิมพ์โค้ดตามนี้

```
// radioButton1
//
this.radioButton1.Checked = true;
this.radioButton1.Location = new System.Drawing.Point(8, 16);
this.radioButton1.Name = "radioButton1";
this.radioButton1.Size = new System.Drawing.Size(80, 24);
this.radioButton1.TabIndex = 9;
this.radioButton1.TabStop = true;
this.radioButton1.Text = "OutputHigh";
```

Object S (radioButtom2) สร้างโดยพิมพ์โค้ดตามนี้

```
// radioButton2
//
this.radioButton2.Location = new System.Drawing.Point(8, 48);
this.radioButton2.Name = "radioButton2";
this.radioButton2.Size = new System.Drawing.Size(80, 24);
this.radioButton2.TabIndex = 10;
this.radioButton2.Text = "OutputLow";
```

Object T (rdo3) สร้างโดยพิมพ์โค้ดตามนี้

```
// rdo3
//
this.rdo3.Checked = true;
this.rdo3.Location = new System.Drawing.Point(8, 16);
this.rdo3.Name = "rdo3";
this.rdo3.Size = new System.Drawing.Size(72, 24);
this.rdo3.TabIndex = 2;
this.rdo3.TabStop = true;
this.rdo3.Text = "InputHigh";
```

Object U (rdo4) สร้างโดยพิมพ์โค้ดตามนี้

```
// rdo4
//
this.rdo4.Location = new System.Drawing.Point(8, 48);
this.rdo4.Name = "rdo4";
this.rdo4.Size = new System.Drawing.Size(72, 24);
this.rdo4.TabIndex = 3;
this.rdo4.Text = "InputLow";
```

Object V (conboBox2) สร้างโดยพิมพ์โค้ดตามนี้

```
// comboBox2
//
this.comboBox2.Items.AddRange(new object[] {
"D0",
"D1",
"D2",
"D3",
"D4",
"D5",
"D6",
"D7",
"B0",
```

```
                "B1",

                "B2",

                "B3"});
            this.comboBox2.Location = new System.Drawing.Point(64, 16);
            this.comboBox2.MaxDropDownItems = 12;
            this.comboBox2.Name = "comboBox2";
            this.comboBox2.Size = new System.Drawing.Size(48, 21);
            this.comboBox2.TabIndex = 1;
            this.comboBox2.Text = "D0";
```

Object W (btnAddIn) สร้างโดยพิมพ์โค้ดตามนี้

```
// btnAddIn
            //
            this.btnAddIn.Location = new System.Drawing.Point(112, 136);
            this.btnAddIn.Name = "btnAddIn";
            this.btnAddIn.Size = new System.Drawing.Size(96, 32);
            this.btnAddIn.TabIndex = 6;
            this.btnAddIn.Text = "Add";
            this.btnAddIn.Click += new System.EventHandler(this.btnAddIn_Click);
```

Object X (btnAddOut) สร้างโดยพิมพ์โค้ดตามนี้

```
// btnAddOut
            //
            this.btnAddOut.Location = new System.Drawing.Point(112, 136);
            this.btnAddOut.Name = "btnAddOut";
            this.btnAddOut.Size = new System.Drawing.Size(96, 32);
            this.btnAddOut.TabIndex = 13;
            this.btnAddOut.Text = "Add";
            this.btnAddOut.Click += new System.EventHandler(this.btnAddOut_Click);
```

Object Y (txtCounter) สร้างโดยพิมพ์โค้ดตามนี้

```
// txtCounter
                    //
                    this.txtCounter.Location = new System.Drawing.Point(56, 104);
                    this.txtCounter.Name = "txtCounter";
                    this.txtCounter.Size = new System.Drawing.Size(88, 20);
                    this.txtCounter.TabIndex = 5;
                    this.txtCounter.Text = "0";

private void txtCounter_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
        {
                    if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar !=8)
                            e.Handled = true;
        }
```

Object Z (label2) สร้างโดยพิมพ์โค้ดตามนี้

```
// label2
                    //
                    this.label2.Location = new System.Drawing.Point(8, 104);
                    this.label2.Name = "label2";
                    this.label2.Size = new System.Drawing.Size(48, 16);
                    this.label2.TabIndex = 4;
                    this.label2.Text = "Counter";
```

Object a (label5) สร้างโดยพิมพ์โค้ดตามนี้

```
// label5
                    //
                    this.label5.Location = new System.Drawing.Point(152, 104);
                    this.label5.Name = "label5";
```

```
this.label5.Size = new System.Drawing.Size(40, 16);
this.label5.TabIndex = 7;
this.label5.Text = "times";
```

Object b (groupBox2) สร้างโดยพิมพ์โค้ดตามนี้

```
// groupBox2
//
this.groupBox2.Controls.Add(this.rdo4);
this.groupBox2.Controls.Add(this.rdo3);
this.groupBox2.Location = new System.Drawing.Point(128, 8);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(96, 80);
this.groupBox2.TabIndex = 8;
this.groupBox2.TabStop = false;
```

Object c (checkBox1) สร้างโดยพิมพ์โค้ดตามนี้

```
// checkBox1
//
this.checkBox1.Location = new System.Drawing.Point(8, 56);
this.checkBox1.Name = "checkBox1";
this.checkBox1.Size = new System.Drawing.Size(120, 24);
this.checkBox1.TabIndex = 10;
this.checkBox1.Text = "Hi-Speed Counter";
this.checkBox1.CheckedChanged += new
System.EventHandler(this.checkBox1_CheckedChanged);

private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
        {
            this.textBox1.ReadOnly = this.checkBox1.Checked? false:true ;
```

Object d (textBox1) สร้างโดยพิมพ์โค้ดตามนี้

```
// textBox1
                    //
                    this.textBox1.Location = new System.Drawing.Point(96, 88);
                    this.textBox1.MaxLength = 5;
                    this.textBox1.Name = "textBox1";
                    this.textBox1.ReadOnly = true;
                    this.textBox1.Size = new System.Drawing.Size(48, 20);
                    this.textBox1.TabIndex = 11;
                    this.textBox1.Text = "10";

private void textBox1_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
            {
                    if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar !=8)
                            e.Handled = true;
            }
```

Object e (label8) สร้างโดยพิมพ์โค้ดตามนี้

```
// label8
                    //
                    this.label8.Location = new System.Drawing.Point(152, 88);
                    this.label8.Name = "label8";
                    this.label8.Size = new System.Drawing.Size(24, 23);
                    this.label8.TabIndex = 12;
                    this.label8.Text = "Hz";
```

Object f (label7) สร้างโดยพิมพ์โค้ดตามนี้

```
// label7
                    //
                    this.label7.Location = new System.Drawing.Point(8, 88);
                    this.label7.Name = "label7";
                    this.label7.Size = new System.Drawing.Size(112, 23);
                    this.label7.TabIndex = 9;
```

```
this.label7.Text = "Max Frequency";
```

Object g (UndoO) สร้างโดยพิมพ์โค้ดตามนี้

```
private System.Windows.Forms.Button UndoO;
// UndoO
                //
                this.UndoO.Location = new System.Drawing.Point(32, 160);
                this.UndoO.Name = "UndoO";
                this.UndoO.Size = new System.Drawing.Size(48, 32);
                this.UndoO.TabIndex = 16;
                this.UndoO.Text = "Undo";
                this.UndoO.Click += new System.EventHandler(this.UndoO_Click);

private void UndoO_Click(object sender, System.EventArgs e)
        {
                Code = OldCode ;
                Dis = OldDis ;
                this.txtOutput.Text=Dis ;
                this.UndoI.Enabled = false ;
                this.UndoO.Enabled = false ;
        }
```

บรรทัดบนสุดสั่งสร้างปุ่ม Undo อีก 10 บรรทัดถัดมาเป็นการกำหนด Properties ของปุ่มเช่น ข้อความบนปุ่ม ขนาด ตำแหน่ง บรรทัดถัดมาที่ขึ้นว่า **private void UndoO_Click** ระบุเงื่อนไขว่าจะให้ทำอะไรเมื่อปุ่มถูกกด คือส่งค่าเก่าที่เก็บไว้ใน OldCode, OldDis คืนให้เพื่อย้อนกลับไปคำสั่งก่อนหน้า (Undo) และสั่งแสดงผลใหม่เพื่อให้ผู้ใช้เห็นการเปลี่ยนแปลง และสั่งให้ปุ่ม Undo จางไปไม่ให้กดได้อีก เพราะโปรมแกรมมีหน่วยความจำสำรองให้ย้อนกลับได้ครั้งเดียว

Object h (UndoI) สร้างโดยพิมพ์โค้ดตามนี้

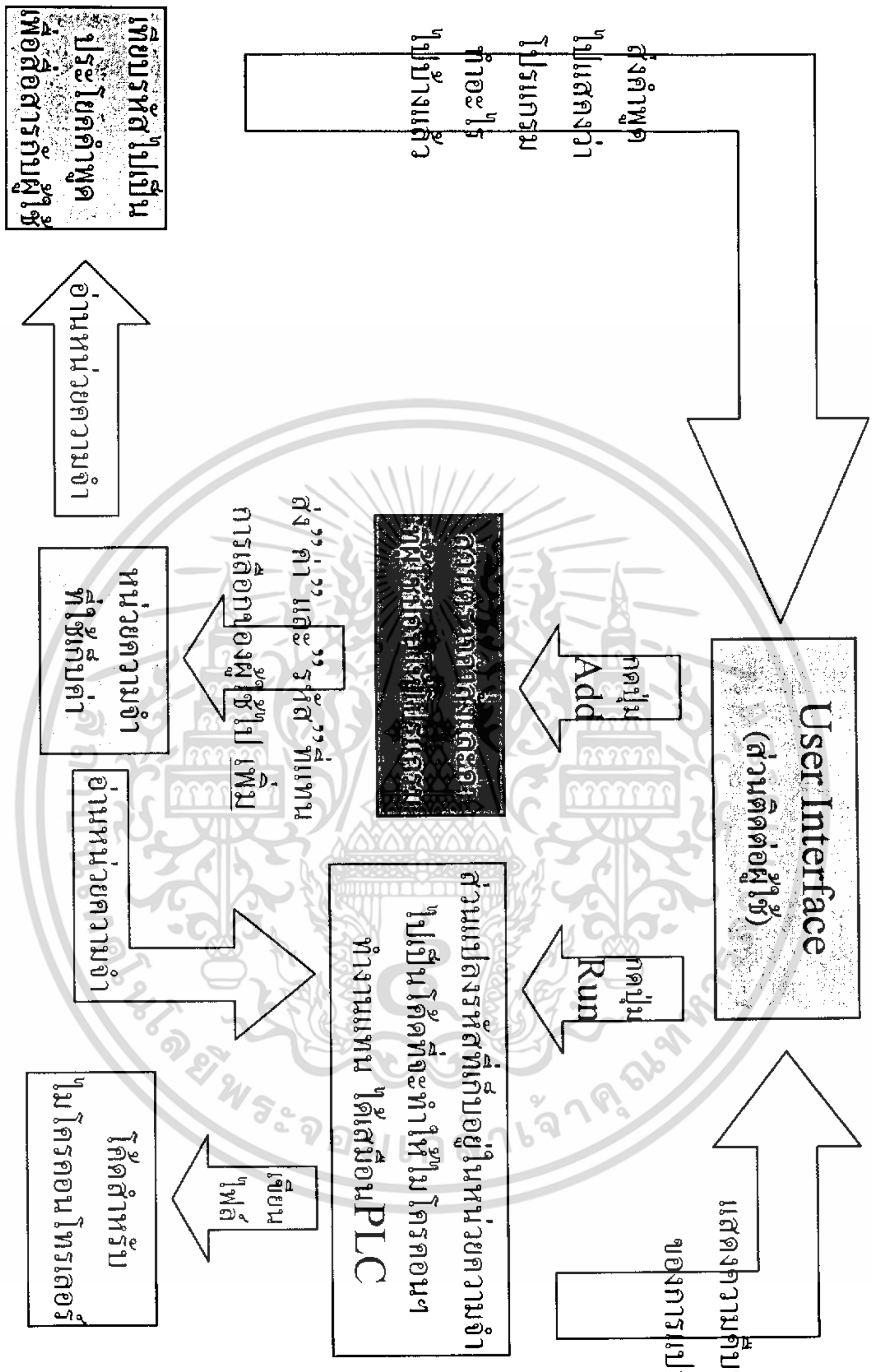
```
private System.Windows.Forms.Button UndoI;
// UndoI
                //
                this.UndoI.Location = new System.Drawing.Point(32, 160);
```

```
this.UndoI.Name = "UndoI";
this.UndoI.Size = new System.Drawing.Size(48, 32);
this.UndoI.TabIndex = 13;
this.UndoI.Text = "Undo";
this.UndoI.Click += new System.EventHandler(this.UndoI_Click);
```

```
private void UndoI_Click(object sender, System.EventArgs e)
        {
            Code = OldCode ;
            Dis = OldDis ;
            this.txtOutput.Text=Dis ;
            this.UndoI.Enabled = false ;
            this.UndoO.Enabled = false ;
        }
```

บรรทัดบนสุดสั่งสร้างปุ่ม Undo อีก 10 บรรทัดถัดมาเป็นการกำหนด Properties ของปุ่มเช่น ข้อความบนปุ่ม ขนาด ตำแหน่ง บรรทัดถัดมาที่ขึ้นว่า **private void UndoO_Click** ระบุเงื่อนไขว่าจะให้ทำอะไรเมื่อปุ่มถูกกด คือส่งค่าเก่าที่เก็บไว้ใน OldCode, OldDis คืนให้เพื่อย้อนกลับไปคำสั่งก่อนหน้า (Undo) และสั่งแสดงผลใหม่เพื่อให้ผู้ใช้เห็นการเปลี่ยนแปลง และสั่งให้ปุ่ม Undo จางไปไม่ให้กดได้อีก เพราะโปรแกรมมีหน่วยความจำสำรองให้ย้อนกลับได้ครั้งเดียว

3.2 ส่วนตรรกะ (Algorithm)



รูปที่ 3-3 ผังการทำงานของโปรแกรม

3.2.1 ส่วนตรวจคำสั่งและค่าที่ผู้ใช้ป้อนให้โปรแกรม ประกอบด้วยโค้ดดังนี้

```
private void btnAddOut_Click(object sender, System.EventArgs e)
                {
                        string Display;
                        Display = (string)(this.radioButton1.Checked?"high":"low");
                        Display += comboBox1.Text;
                        Display += "\r\n"+"delay_ms("+this.textDelay.Text+");"+"\r\n";
                        string Dout;
                        Dout += comboBox1.Text;
                        Dout +=
(string)(this.radioButton1.Checked?"high":"low");"+this.textDelay.Text+" ms"+"\r\n";
                        Dis += Dout;
                        this.txtOutput.Text=Dis ;

                }

private void btnAddIn_Click(object sender, System.EventArgs e)
                {
                        string Display2 , x , y ;
                    Display2 = (string)(this.rdo3.Checked? "!" : " " ) ;
                        Display2 += comboBox2.Text ;
                        y = (string)(this.rdo3.Checked? " " ; "!" ) ;
                        y += " input(PIN_" ;
                        y += comboBox2.Text ;
                        y += " )){ } ; " ;
                        x = Display2 ;
                        Display2 += " \r\n" ;

                        Display2 += "for ( i=0 ; i<" + this.txtCounter.Text + " ; i++ ) { "
+"\r\n"+ y +"\r\n"+ x ;
                        string Dout2;
                        Dout2 += comboBox2.Text;
                        Dout2 += (string)(this.rdo3.Checked? "High" : "low" ) ;
```

```
            Dout2 += "\r\n"+"Counter "+this.txtCounter.Text+" times"+"\r\n";
            this.txtOutput.Text=Dis ;

        }
```

3.2.2 ส่วนเทียบรหัสไปเป็นประโยคคำพูดเพื่อสื่อสารกับผู้ใช้ และแสดงผล ประกอบด้วยโค้ดดังนี้

```
private void btnAddOut_Click(object sender, System.EventArgs e)
        {
            string Display;
            Display = "output_";
            Display += (string)(this.radioButton1.Checked?"high":"low");
            Display += "(PIN_";
            Display += comboBox1.Text;
            Display += ");";
            Display += "\r\n"+"delay_ms("+this.textDelay.Text+");"+"\r\n";
            Code += Display;
            string Dout;
            Dout = "Channel ";
            Dout += comboBox1.Text;
            Dout += " output";
            Dout += (string)(this.radioButton1.Checked?"high":"low");
            Dout += "\r\n"+"delay "+this.textDelay.Text+" ms"+"\r\n";
            Dis += Dout;
            this.txtOutput.Text=Dis ;

        }

        private void btnAddIn_Click(object sender, System.EventArgs e)
        {
            string Display2 , x , y ;
            Display2 = "while(" ;
            Display2 += (string)(this.rdo3.Checked? "!" : " " ) ;
            Display2 += " input(PIN_" ;
```

```
                Display2 += comboBox2.Text ;
                Display2 += " )){ } ; " ;

                y = "while(" ;
                y += (string)(this.rdo3.Checked? " " : "!" ) ;
                y += " input(PIN_" ;
                y += comboBox2.Text ;
                y += " )){ } ; " ;
                x = Display2 ;
                Display2 += " \r\n" ;

                Display2 += "for ( i=0 ; i<" + this.txtCounter.Text + " ; i++ ) { "
+"\r\n"+ y +"\r\n"+ x ;
                Display2 += " \r\n" ;
                Display2 += "delay_ms(300);" ;
                Display2 += "}" ;
                Display2 += " \r\n" ;
                Code += Display2 ;
                string Dout2;
                Dout2 = "Channel ";
                Dout2 += comboBox2.Text;
                Dout2 += " input";
                Dout2 += (string)(this.rdo3.Checked? "High" : "low" ) ;
                Dout2 += "\r\n"+"Counter "+this.txtCounter.Text+" times"+"\r\n";
                Dis += Dout2;
                this.txtOutput.Text=Dis ;

        }
```

จะเห็นว่าส่วนสร้างประโยคคำพูดเพื่อสื่อสารกับผู้ใช้ และส่วนส่วนตรวจคำสั่งและค่าที่ผู้ใช้ป้อนให้โปรแกรมจะทำงานด้วย Event เดียวกัน และใช้หน่วยความจำตัวเดียวกันด้วยดังนั้นจึงได้เห็น โค้ดหลายบรรทัดที่เหมือนกันและตัวแปรหลายตัวที่ใช้ด้วยกัน

3.2.3 ส่วนแปลงรหัสไปเป็นโค้ดไมโครคอนฯ, แสดงความคืบหน้าของการแปล, และเขียนไฟล์

```
using System;
using System.IO;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsApplication2
{
        /// <summary>
        /// Summary description for Form1.
        /// </summary>
private void btnRUN_Click(object sender, System.EventArgs e)
                {string Display3 , a , b ;
                    Display3 = "while(" ;
                    Display3 += (string)(this.rdo3.Checked? "!" : " " ) ;
                    Display3 += " input(PIN_" ;
                    Display3 += comboBox2.Text ;
                    Display3 += " )){ } ; " ;

                    b = "while(" ;
                    b += (string)(this.rdo3.Checked? " " : "!" ) ;
                    b += " input(PIN_" ;
                    b += comboBox2.Text ;
                    b += " )){ } ; " ;
                    a = Display3 ;
                    Display3 += " \r\n" ;

                    Display3 += "for ( i=0 ; i<" + this.txtCounter.Text + " ; i++ ) { "
+"\r\n"+ b +"\r\n"+ x ;
                    Display2 += " \r\n" ;
                    Display2 += "delay_ms(300);" ;
```

```
Display2 += "}" ;
Display2 += " \r\n" ;
Code += Display3 ;

FileStream aFile = new FileStream ("PLC.c",FileMode.Create);
StreamWriter sw = new StreamWriter(aFile);
sw.WriteLine("#define _PIC_16F877A_\r\n");
sw.WriteLine("#include <16F877A.h>\r\n");
sw.WriteLine("#define CLOCK_SP 20000000\r\n");
sw.WriteLine("#fuses HS\r\n");
sw.WriteLine("#fuses NOLVP,NOWDT\r\n");
sw.WriteLine("#fuses NOPROTECT\r\n");
sw.WriteLine("#use delay (clock=20000000)\r\n");
sw.WriteLine("#use fast_io(B)\r\n");
sw.WriteLine("#use fast_io(C)\r\n");
sw.WriteLine("void main() {\r\n");
sw.WriteLine("int i;\r\n");
sw.WriteLine("set_tris_b(0xFF);\r\n");
sw.WriteLine("set_tris_c(0x00);\r\n");
sw.WriteLine("set_tris_d(0xFF);\r\n");
sw.WriteLine("while (TRUE) {\r\n");
sw.WriteLine("{0}\r\n",Code);
sw.WriteLine("}");
sw.WriteLine("}");
sw.Close();

progressBar1.Maximum = 200000;


for(int i = 1; i < 200000 ; ++i)
{
        progressBar1.Value += 1;
}
```

```
                MessageBox.Show ("The compilation are complete", "MessageBox",
                        MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

        }

        private void menuItem3_Click(object sender, System.EventArgs e)
        {
                MessageBox.Show ("PLC v1.0\r\nWritten by Nuttavut Choosak", "About
PLC v1.0",
                        MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }

        private void menuItem2_Click(object sender, System.EventArgs e)
        {
                Application.Exit();
        }

[STAThread]
        static void Main()
        {
                Application.Run(new Form1());
        }
        string Code;
        string Dis;
private System.ComponentModel.IContainer components;

        public Form1()
        {
                //
                // Required for Windows Form Designer support
```

```
            //
            InitializeComponent();

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }
```

3.3 โค้ดทั้งหมดของโปรแกรม

```
using System;
using System.IO;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

```
namespace WindowsApplication2
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.TabControl tabControl1;
        private System.Windows.Forms.TabPage tabPage1;
        private System.Windows.Forms.TabPage tabPage2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Button btnAddOut;
        private System.Windows.Forms.TextBox textDelay;
        private System.Windows.Forms.RadioButton radioButton2;
        private System.Windows.Forms.RadioButton radioButton1;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.TextBox txtOutput;
        private System.Windows.Forms.Button btnAddIn;
        private System.Windows.Forms.TextBox txtCounter;
        private System.Windows.Forms.Button btnRUN;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.RadioButton rdo4;
        private System.Windows.Forms.RadioButton rdo3;
        private System.Windows.Forms.ComboBox comboBox2;
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem2;
```

```
private System.Windows.Forms.MenuItem menuItem3;
private System.Windows.Forms.ProgressBar progressBar1;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.CheckBox checkBox1;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Button UndoO;
private System.Windows.Forms.Button UndoI;
private System.ComponentModel.IContainer components;

public Form1()
{
        //
        // Required for Windows Form Designer support
        //
        InitializeComponent();

        //
        // TODO: Add any constructor code after InitializeComponent call
        //
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if (components != null)
                {
                        components.Dispose();
                }
        }
```

```
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.txtOutput = new System.Windows.Forms.TextBox();
            this.tabControl1 = new System.Windows.Forms.TabControl();
            this.tabPage1 = new System.Windows.Forms.TabPage();
            this.label8 = new System.Windows.Forms.Label();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.checkBox1 = new System.Windows.Forms.CheckBox();
            this.label7 = new System.Windows.Forms.Label();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.rdo4 = new System.Windows.Forms.RadioButton();
            this.rdo3 = new System.Windows.Forms.RadioButton();
            this.label5 = new System.Windows.Forms.Label();
            this.btnAddIn = new System.Windows.Forms.Button();
            this.txtCounter = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.comboBox2 = new System.Windows.Forms.ComboBox();
            this.label1 = new System.Windows.Forms.Label();
            this.tabPage2 = new System.Windows.Forms.TabPage();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.radioButton1 = new System.Windows.Forms.RadioButton();
            this.radioButton2 = new System.Windows.Forms.RadioButton();
            this.label6 = new System.Windows.Forms.Label();
            this.btnAddOut = new System.Windows.Forms.Button();
            this.textDelay = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
```

```
this.comboBox1 = new System.Windows.Forms.ComboBox();
this.label4 = new System.Windows.Forms.Label();
this.btnRUN = new System.Windows.Forms.Button();
this.mainMenu1 = new System.Windows.Forms.MainMenu();
this.menuItem1 = new System.Windows.Forms.MenuItem();
this.menuItem2 = new System.Windows.Forms.MenuItem();
this.menuItem3 = new System.Windows.Forms.MenuItem();
this.progressBar1 = new System.Windows.Forms.ProgressBar();
this.UndoO = new System.Windows.Forms.Button();
this.UndoI = new System.Windows.Forms.Button();
this.tabControl1.SuspendLayout();
this.tabPage1.SuspendLayout();
this.groupBox2.SuspendLayout();
this.tabPage2.SuspendLayout();
this.groupBox1.SuspendLayout();
this.SuspendLayout();
//
// txtOutput
//
this.txtOutput.Location = new System.Drawing.Point(8, 8);
this.txtOutput.Multiline = true;
this.txtOutput.Name = "txtOutput";
this.txtOutput.ReadOnly = true;
this.txtOutput.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.txtOutput.Size = new System.Drawing.Size(280, 320);
this.txtOutput.TabIndex = 0;
this.txtOutput.Text = "";
//
// tabControl1
//
this.tabControl1.Controls.Add(this.tabPage1);
this.tabControl1.Controls.Add(this.tabPage2);
this.tabControl1.Location = new System.Drawing.Point(296, 8);
this.tabControl1.Name = "tabControl1";
```

```
this.tabControl1.SelectedIndex = 0;
this.tabControl1.Size = new System.Drawing.Size(240, 224);
this.tabControl1.TabIndex = 1;
//
// tabPage1
//
this.tabPage1.Controls.Add(this.UndoI);
this.tabPage1.Controls.Add(this.label8);
this.tabPage1.Controls.Add(this.textBox1);
this.tabPage1.Controls.Add(this.checkBox1);
this.tabPage1.Controls.Add(this.label7);
this.tabPage1.Controls.Add(this.groupBox2);
this.tabPage1.Controls.Add(this.label5);
this.tabPage1.Controls.Add(this.btnAddIn);
this.tabPage1.Controls.Add(this.txtCounter);
this.tabPage1.Controls.Add(this.label2);
this.tabPage1.Controls.Add(this.comboBox2);
this.tabPage1.Controls.Add(this.label1);
this.tabPage1.Location = new System.Drawing.Point(4, 22);
this.tabPage1.Name = "tabPage1";
this.tabPage1.Size = new System.Drawing.Size(232, 198);
this.tabPage1.TabIndex = 0;
this.tabPage1.Text = "Input";
//
// label8
//
this.label8.Location = new System.Drawing.Point(152, 88);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(24, 23);
this.label8.TabIndex = 12;
this.label8.Text = "Hz";
//
// textBox1
//
```

```
            this.textBox1.Location = new System.Drawing.Point(96, 88);
            this.textBox1.MaxLength = 5;
            this.textBox1.Name = "textBox1";
            this.textBox1.ReadOnly = true;
            this.textBox1.Size = new System.Drawing.Size(48, 20);
            this.textBox1.TabIndex = 11;
            this.textBox1.Text = "10";
            this.textBox1.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.textBox1_KeyPress);
            //
            // checkBox1
            //
            this.checkBox1.Location = new System.Drawing.Point(8, 56);
            this.checkBox1.Name = "checkBox1";
            this.checkBox1.Size = new System.Drawing.Size(120, 24);
            this.checkBox1.TabIndex = 10;
            this.checkBox1.Text = "Hi-Speed Counter";
            this.checkBox1.CheckedChanged += new
System.EventHandler(this.checkBox1_CheckedChanged);
            //
            // label7
            //
            this.label7.Location = new System.Drawing.Point(8, 88);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(112, 23);
            this.label7.TabIndex = 9;
            this.label7.Text = "Max Frequency";
            //
            // groupBox2
            //
            this.groupBox2.Controls.Add(this.rdo4);
            this.groupBox2.Controls.Add(this.rdo3);
            this.groupBox2.Location = new System.Drawing.Point(128, 0);
            this.groupBox2.Name = "groupBox2";
```

```
this.groupBox2.Size = new System.Drawing.Size(96, 80);
this.groupBox2.TabIndex = 8;
this.groupBox2.TabStop = false;
//
// rdo4
//
this.rdo4.Location = new System.Drawing.Point(8, 48);
this.rdo4.Name = "rdo4";
this.rdo4.Size = new System.Drawing.Size(72, 24);
this.rdo4.TabIndex = 3;
this.rdo4.Text = "InputLow";
//
// rdo3
//
this.rdo3.Checked = true;
this.rdo3.Location = new System.Drawing.Point(8, 16);
this.rdo3.Name = "rdo3";
this.rdo3.Size = new System.Drawing.Size(72, 24);
this.rdo3.TabIndex = 2;
this.rdo3.TabStop = true;
this.rdo3.Text = "InputHigh";
//
// label5
//
this.label5.Location = new System.Drawing.Point(152, 128);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(40, 16);
this.label5.TabIndex = 7;
this.label5.Text = "times";
//
// btnAddIn
//
this.btnAddIn.Location = new System.Drawing.Point(112, 160);
this.btnAddIn.Name = "btnAddIn";
```

```
this.btnAddIn.Size = new System.Drawing.Size(96, 32);
this.btnAddIn.TabIndex = 6;
this.btnAddIn.Text = "Add";
this.btnAddIn.Click += new System.EventHandler(this.btnAddIn_Click);
//
// txtCounter
//
this.txtCounter.Location = new System.Drawing.Point(56, 128);
this.txtCounter.MaxLength = 9;
this.txtCounter.Name = "txtCounter";
this.txtCounter.Size = new System.Drawing.Size(88, 20);
this.txtCounter.TabIndex = 5;
this.txtCounter.Text = "0";
this.txtCounter.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.txtCounter_KeyPress);
//
// label2
//
this.label2.Location = new System.Drawing.Point(8, 128);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(48, 16);
this.label2.TabIndex = 4;
this.label2.Text = "Counter";
//
// comboBox2
//
this.comboBox2.Items.AddRange(new object[] {
  "D0",
  "D1",
  "D2",
```

```
"D3",
"D4",
"D5",
"D6",
"D7",
"B0",
"B1",
"B2",
"B3"});
this.comboBox2.Location = new System.Drawing.Point(64, 8);
this.comboBox2.MaxDropDownItems = 12;
this.comboBox2.Name = "comboBox2";
this.comboBox2.Size = new System.Drawing.Size(48, 21);
this.comboBox2.TabIndex = 1;
this.comboBox2.Text = "D0";
//
// label1
//
this.label1.Location = new System.Drawing.Point(8, 8);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(48, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Channel";
//
// tabPage2
```

```
//
this.tabPage2.Controls.Add(this.UndoO);
this.tabPage2.Controls.Add(this.groupBox1);
this.tabPage2.Controls.Add(this.label6);
this.tabPage2.Controls.Add(this.btnAddOut);
this.tabPage2.Controls.Add(this.textDelay);
this.tabPage2.Controls.Add(this.label3);
this.tabPage2.Controls.Add(this.comboBox1);
this.tabPage2.Controls.Add(this.label4);
this.tabPage2.Location = new System.Drawing.Point(4, 22);
this.tabPage2.Name = "tabPage2";
this.tabPage2.Size = new System.Drawing.Size(232, 198);
this.tabPage2.TabIndex = 1;
this.tabPage2.Text = "Output";
//
// groupBox1
//
this.groupBox1.Controls.Add(this.radioButton1);
this.groupBox1.Controls.Add(this.radioButton2);
this.groupBox1.Location = new System.Drawing.Point(128, 0);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(96, 80);
this.groupBox1.TabIndex = 15;
this.groupBox1.TabStop = false;
//
// radioButton1
//
this.radioButton1.Checked = true;
this.radioButton1.Location = new System.Drawing.Point(8, 16);
this.radioButton1.Name = "radioButton1";
this.radioButton1.Size = new System.Drawing.Size(80, 24);
this.radioButton1.TabIndex = 9;
this.radioButton1.TabStop = true;
this.radioButton1.Text = "OutputHigh";
```

```
//
// radioButton2
//
this.radioButton2.Location = new System.Drawing.Point(8, 48);
this.radioButton2.Name = "radioButton2";
this.radioButton2.Size = new System.Drawing.Size(80, 24);
this.radioButton2.TabIndex = 10;
this.radioButton2.Text = "OutputLow";
//
// label6
//
this.label6.Location = new System.Drawing.Point(152, 128);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(48, 16);
this.label6.TabIndex = 14;
this.label6.Text = "ms";
//
// btnAddOut
//
this.btnAddOut.Location = new System.Drawing.Point(112, 160);
this.btnAddOut.Name = "btnAddOut";
this.btnAddOut.Size = new System.Drawing.Size(96, 32);
this.btnAddOut.TabIndex = 13;
this.btnAddOut.Text = "Add";
this.btnAddOut.Click += new System.EventHandler(this.btnAddOut_Click);
//
// textDelay
//
this.textDelay.Location = new System.Drawing.Point(56, 128);
this.textDelay.MaxLength = 9;
this.textDelay.Name = "textDelay";
this.textDelay.Size = new System.Drawing.Size(88, 20);
this.textDelay.TabIndex = 12;
this.textDelay.Text = "0";
```

```
            this.textDelay.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.textDelay_KeyPress);
            //
            // label3
            //
            this.label3.Location = new System.Drawing.Point(8, 128);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(48, 16);
            this.label3.TabIndex = 11;
            this.label3.Text = "Delay";
            //
            // comboBox1
            //
            this.comboBox1.Items.AddRange(new object[] {
              "C0",
              "C1",
              "C2",
              "C3",
              "C4",
              "C5",
              "C6",
              "C7"});
            this.comboBox1.Location = new System.Drawing.Point(64, 8);
            this.comboBox1.MaxDropDownItems = 12;
            this.comboBox1.Name = "comboBox1";
            this.comboBox1.Size = new System.Drawing.Size(48, 21);
```

```
this.comboBox1.TabIndex = 8;
this.comboBox1.Text = "C0";
//
// label4
//
this.label4.Location = new System.Drawing.Point(8, 8);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(48, 16);
this.label4.TabIndex = 7;
this.label4.Text = "Channel";
//
// btnRUN
//
this.btnRUN.Location = new System.Drawing.Point(360, 280);
this.btnRUN.Name = "btnRUN";
this.btnRUN.Size = new System.Drawing.Size(112, 40);
this.btnRUN.TabIndex = 2;
this.btnRUN.Text = "RUN";
this.btnRUN.Click += new System.EventHandler(this.btnRUN_Click);
//
// mainMenu1
//
this.mainMenu1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {


this.menuItem1,



this.menuItem3});
//
// menuItem1
//
this.menuItem1.Index = 0;
```

```
                this.menuItem1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {


this.menuItem2});
                this.menuItem1.Text = "File";
                //
                // menuItem2
                //
                this.menuItem2.Index = 0;
                this.menuItem2.Text = "Exit";
                this.menuItem2.Click += new System.EventHandler(this.menuItem2_Click);
                //
                // menuItem3
                //
                this.menuItem3.Index = 1;
                this.menuItem3.Text = "About";
                this.menuItem3.Click += new System.EventHandler(this.menuItem3_Click);
                //
                // progressBar1
                //
                this.progressBar1.Location = new System.Drawing.Point(328, 248);
                this.progressBar1.Name = "progressBar1";
                this.progressBar1.Size = new System.Drawing.Size(176, 16);
                this.progressBar1.TabIndex = 3;
                //
                // UndoO
                //
                this.UndoO.Location = new System.Drawing.Point(32, 160);
                this.UndoO.Name = "UndoO";
                this.UndoO.Size = new System.Drawing.Size(48, 32);
                this.UndoO.TabIndex = 16;
                this.UndoO.Text = "Undo";
                this.UndoO.Click += new System.EventHandler(this.UndoO_Click);
```

```
            //
            // UndoI
            //
            this.UndoI.Location = new System.Drawing.Point(32, 160);
            this.UndoI.Name = "UndoI";
            this.UndoI.Size = new System.Drawing.Size(48, 32);
            this.UndoI.TabIndex = 13;
            this.UndoI.Text = "Undo";
            this.UndoI.Click += new System.EventHandler(this.UndoI_Click);
            //
            // Form1
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(544, 337);
            this.Controls.Add(this.progressBar1);
            this.Controls.Add(this.btnRUN);
            this.Controls.Add(this.tabControl1);
            this.Controls.Add(this.txtOutput);
            this.Menu = this.mainMenu1;
            this.Name = "Form1";
            this.Text = "PLC programing";
            this.tabControl1.ResumeLayout(false);
            this.tabPage1.ResumeLayout(false);
            this.groupBox2.ResumeLayout(false);
            this.tabPage2.ResumeLayout(false);
            this.groupBox1.ResumeLayout(false);
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
```

```
[STAThread]
static void Main()
{
        Application.Run(new Form1());
}
string OldCode, Code = null ;
string OldDis, Dis = null ;

private void btnAddOut_Click(object sender, System.EventArgs e)
{
        OldCode = Code ;
        OldDis = Dis ;
        string Display;
        Display = "output_";
        Display += (string)(this.radioButton1.Checked?"high":"low");
        Display += "(PIN_";
        Display += comboBox1.Text;
        Display += ");";
        Display += "\r\n"+"delay_ms("+this.textDelay.Text+");"+"\r\n";
        Code += Display;
        string Dout;
        Dout = "Channel ";
        Dout += comboBox1.Text;
        Dout += " output";
        Dout += (string)(this.radioButton1.Checked?"high":"low");
        Dout += "\r\n"+"delay "+this.textDelay.Text+" ms"+"\r\n";
        Dis += Dout;
        this.txtOutput.Text=Dis ;
        this.UndoI.Enabled = true ;
        this.UndoO.Enabled = true ;

}

private void btnAddIn_Click(object sender, System.EventArgs e)
```

```
{

                OldCode = Code ;
                OldDis = Dis ;
                string Display2 , x , y ;
                Display2 = "while(" ;
                Display2 += (string)(this.rdo3.Checked? "!" : " " ) ;
                Display2 += " input(PIN_" ;
                Display2 += comboBox2.Text ;
                Display2 += " )){ } ; " ;


                y = "while(" ;
                y += (string)(this.rdo3.Checked? " " : "!" ) ;
                y += " input(PIN_" ;
                y += comboBox2.Text ;
                y += " )){ } ; " ;
                x = Display2 ;
                Display2 += " \r\n" ;

                Display2 +=  "for ( i=0 ; i<" +  this.txtCounter.Text  + "  ; i++ ) { "
+"\r\n"+  y  +"\r\n"+  x    ;
                Display2 += " \r\n" ;


                ushort z ;
                z = Convert.ToUInt16(this.textBox1.Text);
                z = Convert.ToUInt16(1000/z);
                Display2 += "delay_ms(" ;
                Display2 += z ;
                Display2 += ");" ;
                Display2 += "}" ;
                Display2 += " \r\n" ;
                Code += Display2 ;
                string Dout2;
                Dout2 = "Channel ";
```

```
            Dout2 += comboBox2.Text;
            Dout2 += " input";
            Dout2 += (string)(this.rdo3.Checked? "High" : "low" ) ;
            Dout2 += "\r\n"+"Counter "+this.txtCounter.Text+" times"+"\r\n";
            Dis += Dout2;
            this.txtOutput.Text=Dis ;
            this.UndoI.Enabled = true ;
            this.UndoO.Enabled = true ;

}

private void btnRUN_Click(object sender, System.EventArgs e)
{
            FileStream aFile = new FileStream ("PLC.c",FileMode.Create);
            StreamWriter sw = new StreamWriter(aFile);
            sw.WriteLine("#define _PIC_16F877A_\r\n");
            sw.WriteLine("#include <16F877A.h>\r\n");
            sw.WriteLine("#define CLOCK_SP 20000000\r\n");
            sw.WriteLine("#fuses HS\r\n");
            sw.WriteLine("#fuses NOLVP,NOWDT\r\n");
            sw.WriteLine("#fuses NOPROTECT\r\n");
            sw.WriteLine("#use delay (clock=20000000)\r\n");
            sw.WriteLine("#use fast_io(B)\r\n");
            sw.WriteLine("#use fast_io(C)\r\n");
            sw.WriteLine("void main() {\r\n");
            sw.WriteLine("int i;\r\n");
            sw.WriteLine("set_tris_b(0xFF);\r\n");
            sw.WriteLine("set_tris_c(0x00);\r\n");
            sw.WriteLine("set_tris_d(0xFF);\r\n");
            sw.WriteLine("while (TRUE) {\r\n");
            sw.WriteLine("{0}\r\n",Code);
            sw.WriteLine("}");
            sw.WriteLine("}");
            sw.Close();
```

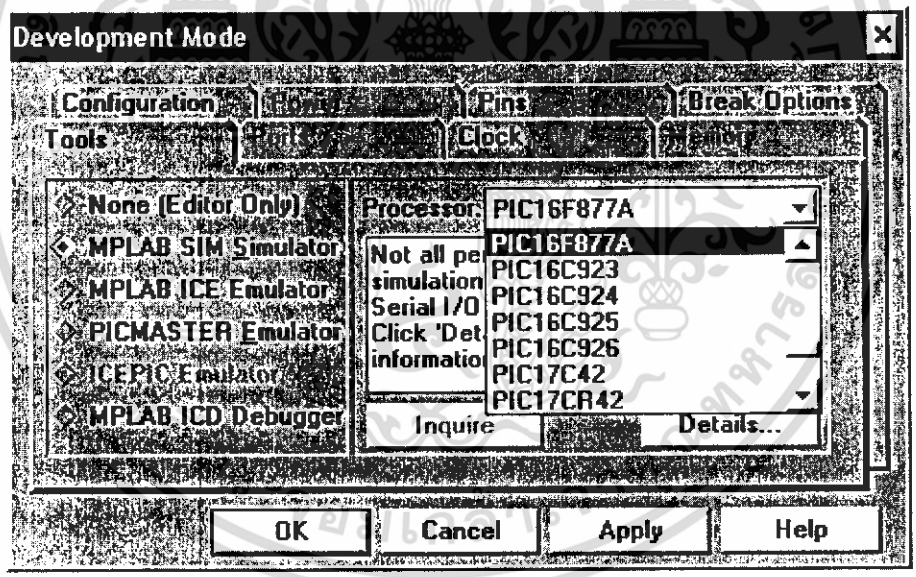
```
            progressBar1.Maximum = 200000;


            for(int i = 1; i < 200000 ; ++i)
            {
                progressBar1.Value += 1;
            }


            MessageBox.Show ("The compilation are complete", "MessageBox",
                MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

        }

        private void menuItem3_Click(object sender, System.EventArgs e)
        {
            MessageBox.Show ("PLC v1.0\r\nWritten by Nuttavut Choosak", "About
PLC v1.0",
                MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }

        private void menuItem2_Click(object sender, System.EventArgs e)
        {
            Application.Exit();

        }


        private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
        {
            this.textBox1.ReadOnly = this.checkBox1.Checked? false:true ;
        }
```

```
        private void textBox1_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
        {
                if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar !=8)
                        e.Handled = true;
        }

        private void txtCounter_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
        {
                if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar !=8)
                        e.Handled = true;
        }

        private void textDelay_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
        {
                if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar !=8)
                        e.Handled = true;
        }

        private void UndoI_Click(object sender, System.EventArgs e)
        {
                Code = OldCode ;
                Dis = OldDis ;
                this.txtOutput.Text=Dis ;
                this.UndoI.Enabled = false ;
                this.UndoO.Enabled = false ;
        }

        private void UndoO_Click(object sender, System.EventArgs e)
        {
                Code = OldCode ;
```

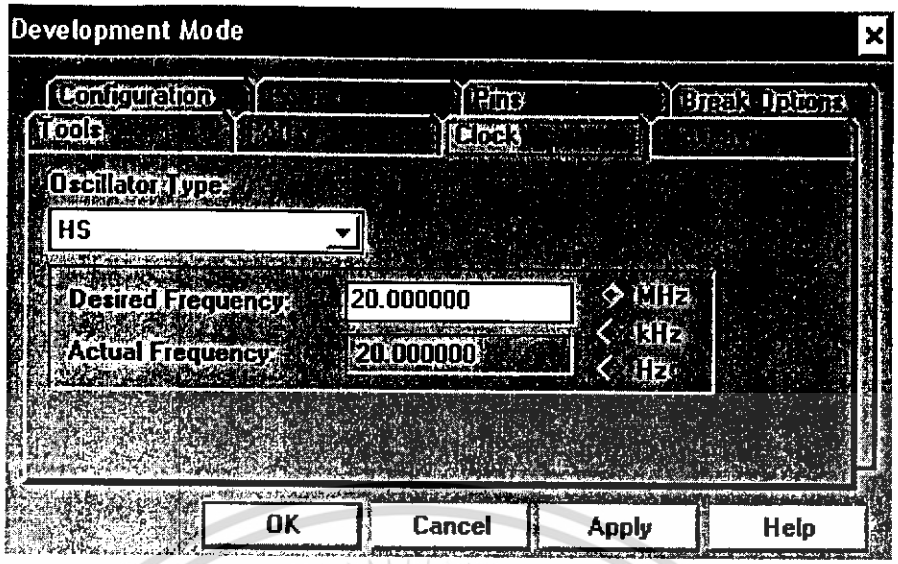
```
                    Dis = OldDis ;
                    this.txtOutput.Text=Dis ;
                    this.UndoI.Enabled = false ;
                    this.UndoO.Enabled = false ;
                }
            }
}
```

3.4 การหาเวลาที่ไมโครคอนโทรเลอร์ใช้ในการทำงานในแต่ละคำสั่ง

การหาเวลาที่ไมโครคอนโทรเลอร์ใช้ในการทำงานแต่ละคำสั่งนั้นเราวัดออกมาด้วย โปรแกรม MPLAB ที่ผู้ผลิตไมโครคอนโทรเลอร์ สร้างขึ้นและเผยแผ่ให้ใช้ฟรีเพื่อสนับสนุนสินค้า โปรแกรม MPLABปกติใช้แปลภาษาแอสแซมบลีเป็นแมนชีนโค้ด แต่เราใช้ความสามารถในการ Simulate ทำการจำลองโปรแกรมเป็น IC PIC16F877A เพื่อศึกษาการทำงาน และจับเวลาว่าคำสั่งที่เราสร้างให้ทำงานเลียนแบบ PLC ใช้เวลาเท่าไรเพื่อระบุคุณสมบัติ PLC เรา

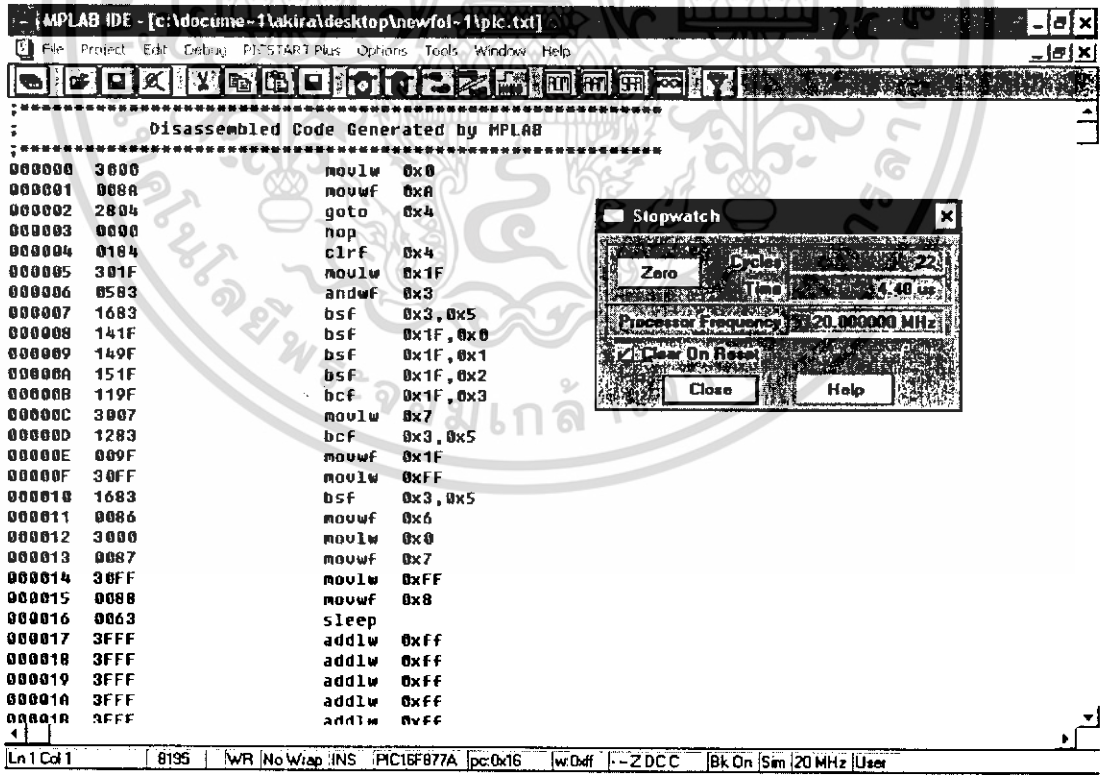


รูปที่ 3-4 เลือก ไมโครคอนโทรเลอร์ที่เราใช้ (PIC16F877A) และเลือกใช้การ simulate โดยเลือกตามภาพ (ปกติจะอยู่ที่ none editor only)



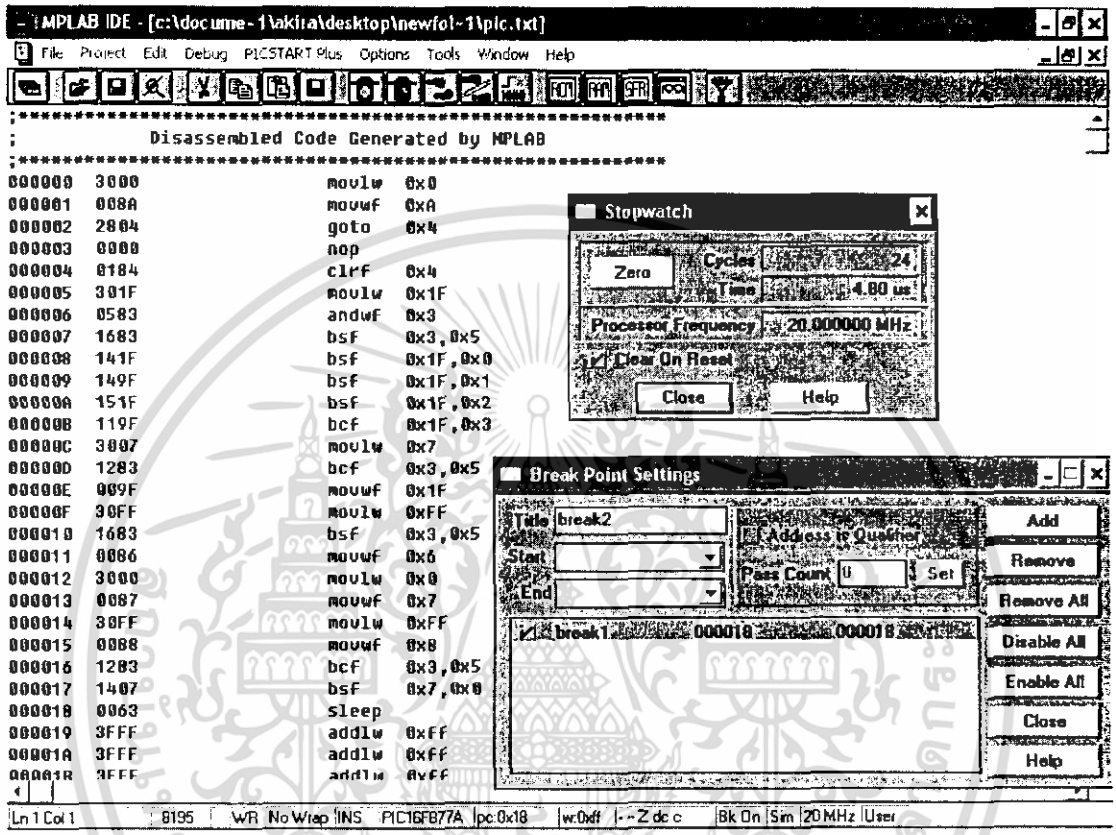
รูปที่ 3-5 เลือกโหมดสัญญาณนาฬิกาเป็น HS (คริสตอลความถี่สูง) และใช้ความถี่ในโปรแกรมให้เหมือนกับในHardwareจริงคือ 20 MHz

จากการวัดด้วยโปรแกรม หลังจากเกิดการรีเซ็ตหรือเปิดเครื่องครั้งแรก ไมโครคอนโทรเลอร์เราใช้เวลาเตรียมก่อนจะทำคำสั่งใดๆที่ผู้ใช้สั่งไปได้โปรแกรมจับเวลาได้ตามรูป 3-6



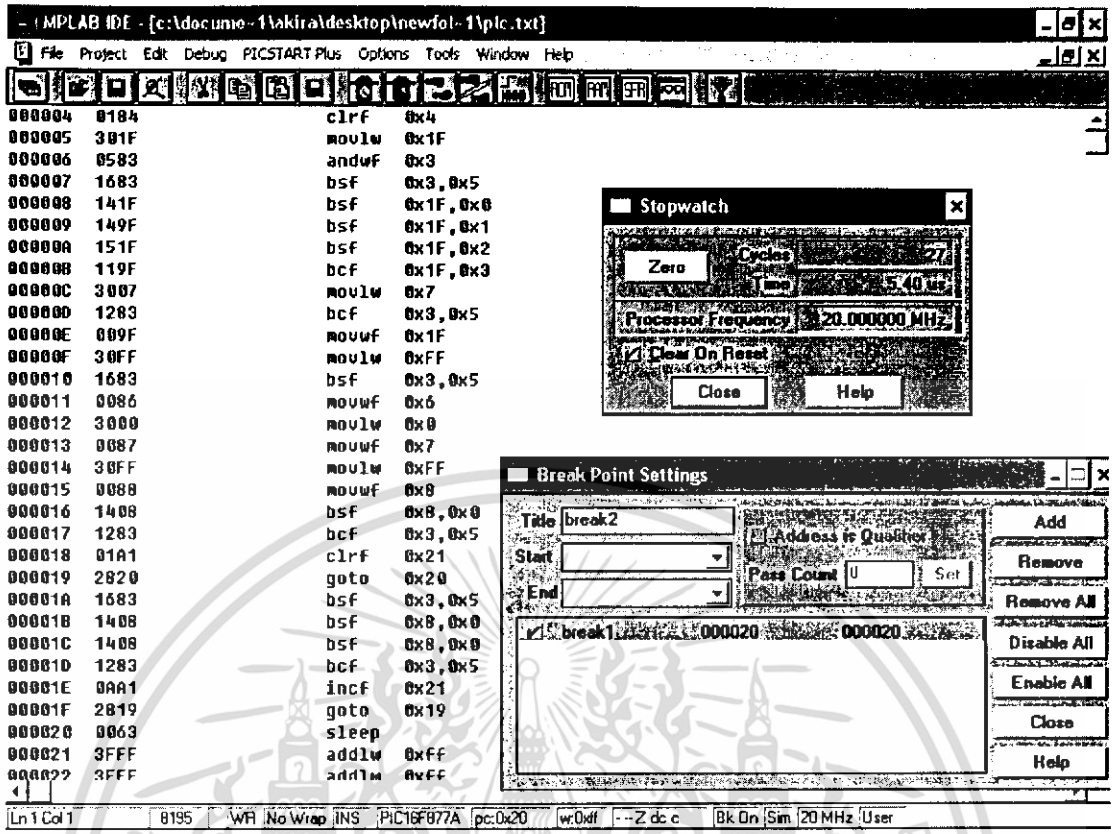
รูปที่ 3-6 ใช้เวลา 4.40 ไมโครวินาที

หลังจากนั้นเราลองวัดเวลาในการเปิดเครื่องครั้งแรกไปจบการทำงานจบคำสั่ง(ที่PLCเราใช้) เราจะได้เวลาที่ PLC เตรียมตัว บวกกับ ที่ PLC ทำหนึ่งคำสั่ง เหตุผลที่เราไม่วัดเวลาการทำคำสั่งโดยตรงเพราะไม่สามารถแยกให้แค่เฉพาะส่วนคำสั่ง compileได้ เราได้เวลารวมนี้ออกมาดังรูป 3-7



รูปที่ 3-7 ใช้เวลา 4.80 ไมโครวินาที

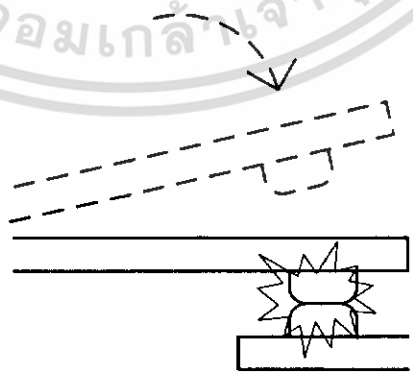
เมื่อเรา ลบ เวลารวมด้วยเวลาเตรียมตัว เราจะได้เวลาที่PLCใช้ทำหนึ่งคำสั่ง 4.8 - 4.4 = 0.4 us เราจึงสรุปได้ว่าคำสั่งสั้นสุดนั้นใช้เวลา 0.4 us ต่อมาเราวัดเวลาเตรียม กับเวลาทำคำสั่งที่นานสุดได้ตามรูป 3-8



รูปที่ 3-8 ใช้เวลา 5.40 ไมโครวินาที

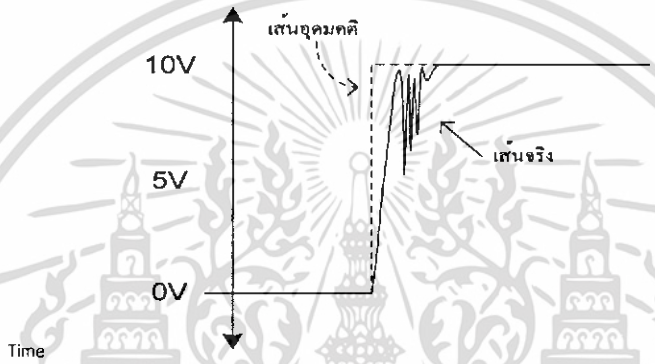
เมื่อเรา ลบ เวลารวมด้วยเวลาเตรียมตัว เราจะได้เวลาที่PLCใช้ทำหนึ่งคำสั่ง 5.4 - 4.4 = 1 us เราจึงสรุปได้ว่าคำสั่งนานสุดนั้นใช้เวลา 1 us

3.5 การเกิดการกระดอนของสวิทช์ (Switch bounce) และการป้องกันด้วยซอฟต์แวร์



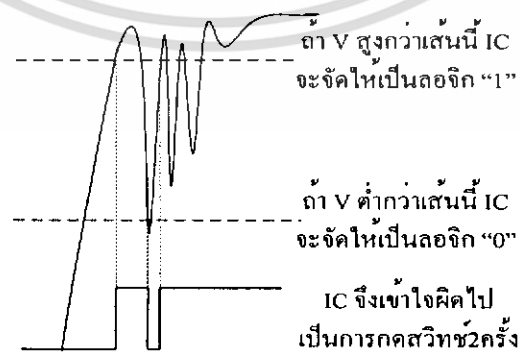
รูปที่ 3-9 การเกิด Switch bounce (การกระดอนของหน้าสัมผัส)97

เพื่อให้สวิทช์ตัดหรือต่อได้ที่ความเร็วสูงๆ แผ่นหน้าสัมผัสสวิทช์จะต้องมีน้ำหนักเบา และการตัดต่อที่ความถี่สูงๆ หน้าสัมผัสสวิทช์จะกระทบกันด้วยความเร็วสูง เมื่อหน้าสัมผัสกระแทกกันย่อมกระดอนกลับ หน้าสัมผัสเมื่อตัดกระแสไฟฟ้าจะเกิดการอาร์คขึ้นทำให้หน้าสัมผัสเกิดออกไซด์ทำให้การนำไฟฟ้าไม่สะดวก ผิวสัมผัสจะนำไฟฟ้าดีขึ้นเมื่อมีแรงกดผิวสัมผัสเข้าด้วยกัน การกระดอนของหน้าสัมผัสจะทำให้แรงกดไม่คงที่ทำให้การไหลของกระแสแกว่งไปมาด้วย การกระดอนของหน้าสัมผัสและออกไซด์ของหน้าสัมผัสนี้เกิดกับสวิทช์ทุกชนิด(มากน้อยตามคุณภาพสวิทช์) ผลของความเหนี่ยวนำความต้านทานของสายรวมทั้งความเป็นตัวเก็บประจุของฉนวนจะไปเพิ่มให้ การเปลี่ยนระดับสัญญาณจากการตัดต่อสวิทช์จะยิ่งต่างจากในอุดมคติ รูปร่างสัญญาณจะผิดจากอุดมคติไปดังรูป



รูปที่ 3-10 กราฟแรงดันและเวลา ของสวิทช์ที่เกิดการ Switch bounce

ผลของ Switch bounces จะกระทบต่อการตรวจจับสัญญาณของภาคอินพุตไมโครคอนโทรเลอร์ การกระดอนของหน้าสัมผัสอาจทำให้ภาคอินพุตเข้าใจผิดได้ว่า มีการกดและปล่อยสวิทช์ไปมาหลายครั้ง (ทั้งที่สวิทช์ต่อครั้งเดียว) เพราะไมโครคอนโทรเลอร์ที่เราใช้แทน PLC มีศักยภาพสูงจนสามารถจับสัญญาณได้ถึงความถี่ 5 MHz ทำให้ในการทดสอบช่วงแรกๆ การกดสวิทช์ครั้งเดียวแต่ส่วน Counter นับได้ 2-3 ครั้ง

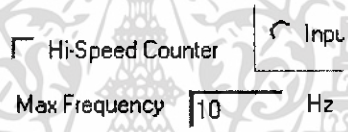


รูปที่ 3-11 ผลของ Switch bounces ทำให้ภาคอินพุตเข้าใจผิดว่ามีการกดสวิทช์หลายครั้ง

การแก้ปัญหานี้ตามปกติจะใช้วงจร Low pass filter กรองสัญญาณรบกวนออกไป การใช้ Low pass filter เป็นวิธีพื้นฐานที่ใช้กันทั่วไป เมื่อPLC เราใช้งานกับสวิทช์ทั่วไปใช้กับตัดต่อ 4ครั้งต่อวินาทีโดยไม่มีผิดพลาด สวิทช์ความเร็วสูงผู้ผลิตเองก็ระบุความถี่ที่ใช้งานได้โดยไม่ผิดพลาดไว้ที่ 10 Hz ดังนั้นเราจึงออกแบบไว้ว่าความถี่สูงกว่า 20 Hz (เผื่อไว้ 2 เท่า) เป็นสัญญาณรบกวนได้ และออกแบบLow pass filter กรองความถี่สูงกว่า 20Hz ทิ้ง

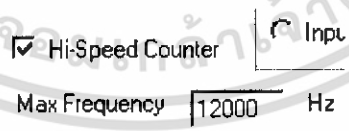
อย่างไรการป้องกัน Switch bounce ด้วยวงจร Low pass filter เป็นวิธีทาง Hardware หมายความว่าเมื่อสร้างเสร็จแล้วจะแก้ไขอะไรไม่ได้ (เพราะเป็น Hardware) เช่นเราสร้างไว้ใช้กับสวิทช์ออกแบบไว้กรองความถี่สูงกว่า 20Hz ทิ้ง หากจะนำไปใช้กับ photoelectric sensor ซึ่งทำงานได้ถึง 10 kHz และไม่มี Switch bounce (เนื่องจากไม่มีชิ้นส่วนเคลื่อนไหว) สัญญาณความถี่สูงจึงผ่าน Low pass filter ไม่ได้

ผู้สร้างได้คิดวิธีป้องกัน Switch bounce ด้วยวิธีทาง Software ขึ้นมา Hardware (Low pass filter) เมื่อกำหนดความถี่ Cut off (ความถี่ที่สัญญาณผ่านไม่ได้) ไปแล้วจะปรับเปลี่ยนไม่ได้อีก แต่วิธีทาง Software ที่คิดขึ้นมานี้จะสามารถเปลี่ยนความถี่ Cut off ได้โดยอิสระจากโปรแกรม ทำให้ PLC เรามีความยืดหยุ่นในการใช้งานมากขึ้น การใช้งานนั้นเราออกแบบโปรแกรมไว้ดังนี้



รูปที่ 3-12 ค่าเริ่มต้นของโปรแกรม ใช้กับสวิทช์ธรรมดา

โดยค่าเริ่มต้นนั้นโปรแกรมจะถูกตั้งไว้เป็น ใช้กับสวิทช์ธรรมดา ไม่ใช้เคาน์เตอร์ความเร็วสูง ความถี่ Cut off อยู่ที่ 10 Hz และจะไม่สามารถแก้ความถี่ cut off ในช่องได้ (สีทึบ) ในโหมดนี้ใช้กับสวิทช์ทั่วไปที่เกิด Switch bounce ได้ทันทีโดยไม่ต้องทำอะไร

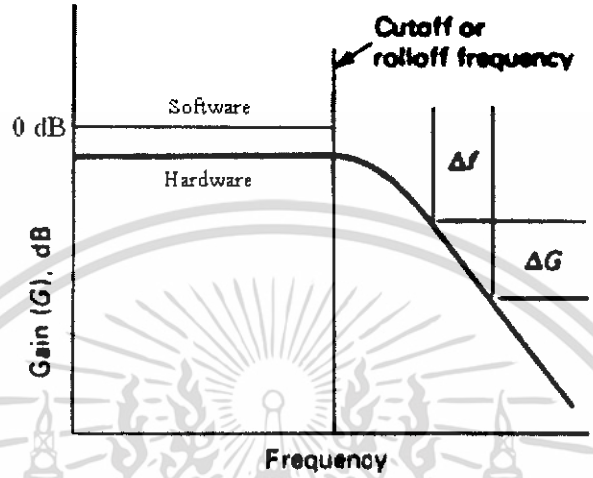


รูปที่ 3-13 ใช้การนับสำหรับความเร็วสูง

หากเลือกที่ช่อง Hi-Speed Counter ช่องความถี่ cut off จะสามารถเปลี่ยนความถี่ได้ เราจะเลือกได้ว่าจะกั้นความถี่สูงกว่าเท่าใด โดยกรอกตัวเลขได้เลย เช่น photoelectric sensor ซึ่งทำงานได้ถึง 10 kHz เราอาจป้อนเผื่อไปเป็น 12 kHz ได้

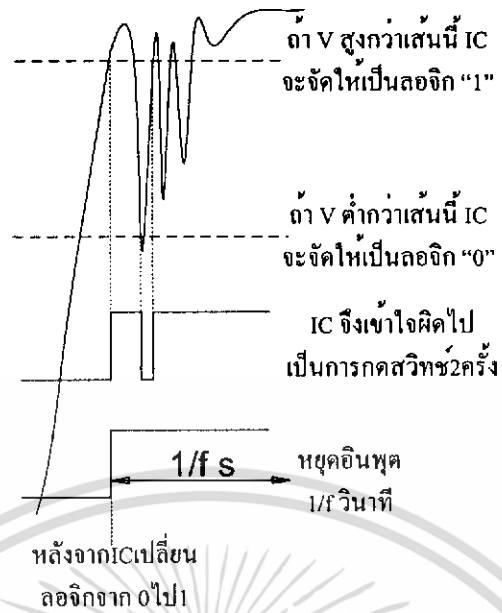
ลักษณะพิเศษอีกอย่างของการกรองความถี่ทางSoftware ที่คิดขึ้นมาใหม่นี้คือ วงจร Low pass filter จะมีสัญญาณสูญเสียไปเสมอจากความต้านทานในวงจรและการรั่วไหลของกระแสในตัวเก็บประจุ

และการสกัดกั้นความถี่ไม่เป็นไปอย่างทันทีทันใด คือความถี่สูงกว่าความถี่ cut off จะมีการลดทอนที่สูง แต่ที่ความถี่ใกล้ความถี่ cut off การลดทอนจะต่ำกว่า (คือไม่สามารถตัดสัญญาณอย่างทันทีทันใด) แต่การกรองความถี่ทางSoftware จะเกือบเป็นอุดมคติคือตัดสัญญาณที่ความถี่สูงกว่า cut off ทันทีและมีการลดทอนสูงมากตลอดย่าน โดยไม่มีการสูญเสียของสัญญาณปกติ (เพราะไม่มีความต้านทานมาขวางสัญญาณ)



รูปที่ 3-14 กราฟการลดทอนสัญญาณ

วิธีทาง Software ของเรานั้นเรียบง่ายมาก โดยเรายอมให้สัญญาณทั้งหมดเข้าสู่ส่วนอินพุตก่อน เมื่อสัญญาณไฟฟ้ากลายเป็นข้อมูลหมดแล้วเราค่อยแก้ไขข้อมูลเดิมให้เป็นไปตามที่เราต้องการ ขบวนการมีดังนี้ เมื่อผู้ใช้ป้อนความถี่ cut off ให้โปรแกรมในช่อง Max frequency โปรแกรมจะเปลี่ยนความถี่เป็นคาบเวลา (T=1/f) ในเมื่อเราไม่ต้องการความถี่ที่สูงกว่าผ่านก็หมายความว่าเราจะไม่ยอมให้เกิดข้อมูลที่มีคาบเวลาต่ำกว่า T (ที่คำนวณได้)เกิดขึ้น มีวิธีการทาง Software หลายแบบที่จะเปลี่ยนข้อมูลให้ไม่มีความถี่สูงเกิด cut off แต่ทุกวิธีต้องใช้งาน CPU ไม่มากก็น้อย แต่เราก็พบวิธีที่ไม่ต้องใช้ CPU โดยหยุดการทำงานของภาคอินพุตของ IC เป็นเวลานานเท่ากับT=1/f วินาที ระหว่างที่หยุดการทำงานของภาคอินพุตข้อมูลจะค้างอยู่แบบเดิม(เช่นเป็นลอจิก 0 ก็จะเป็น 0 ต่อไป)จนกว่าจะเริ่มให้ภาคอินพุตทำงานใหม่ และเราจะเริ่มกระบวนการหยุดภาคอินพุตทุกครั้งที่ภาคอินพุตเปลี่ยนลอจิกจาก 0ไป1 หรือ 1ไป0 พูดง่ายๆว่าการหยุดการทำงานภาคอินพุตสั้นๆแบบนี้จะทำให้ความสามารถการจับสัญญาณที่ไวเกินไปจนได้สัญญาณรบกวนมาด้วย ลดความไวลงเหลือรับได้แต่สัญญาณที่ต้องการ ปกติ IC จับสัญญาณได้ถึงความถี่ 5 MHz แต่วิธีการใช้ภาคอินพุตแบบไม่เต็มประสิทธิภาพนี้จะทำให้มันจับสัญญาณได้ต่ำลงเหลือแค่ 1 Hz ได้กระบวนการ Low pass filter ด้วย Software นี้อธิบายได้ด้วยภาพ 3-14



รูปที่ 3-15 การหยุดการทำงานภาคอินพุต

หลังจาก IC เปลี่ยนลอจิกจาก 0ไป 1 (หรือ 1ไป0) คงข้อมูลเดิมไว้ 1/f วินาที ทำให้ไม่เกิดข้อมูลที่มีความถี่เกิน f Hz เนื่องจากเราสามารถเปลี่ยนความถี่ f ได้ ผมจึงเรียกกระบวนการนี้ว่าวงจรกรองความถี่แบบโปรแกรมได้

บทที่ 4
การทดสอบการใช้งาน

4.1 ตัวอย่างวิธีการใช้โปรแกรมและ PLC ที่สร้างขึ้น กับงานทั่วไป

วิธีการใช้โปรแกรมโดยอธิบายด้วย 2 ตัวอย่าง โจทย์ตัวอย่าง 1 เป็นตัวอย่างการใช้งานง่ายๆ

1.มีไฟเข้าขั้ว input B3

2.พอขั้ว input B3 มีไฟเข้าแล้วให้ output C0 ปล่อยไฟบวกออกมา (output เป็น High)

3.มีไฟเข้าขั้ว input B3 มา 5 ครั้ง (สวิทช์ถูกกด ปล่อย 5 ครั้ง)

4.ต้องการให้ output C0 (ที่ปล่อยไฟบวกออกมาตั้งแต่ข้อ 2) ไม่ปล่อยไฟออกมา 0.5วินาที (output เป็น Low นาน 0.5วินาที)

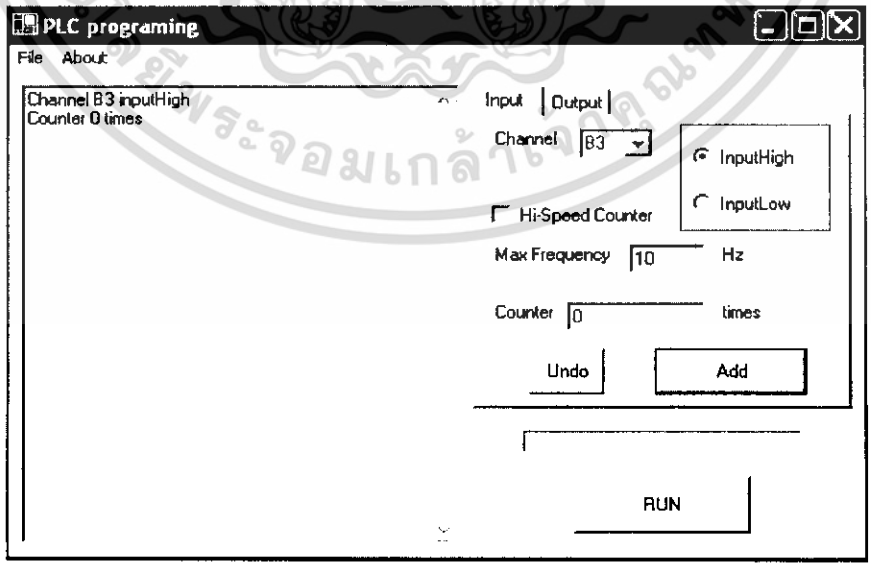
5. ให้ output C0 กลับเป็นบวกอีกครั้ง (output เป็น High)

ลำดับการสั่ง PLC คือ

1. (รับ) input B3 เป็น high
2. (สั่ง) output C0 เป็น High
3. (รับ) input B3 เป็น high พร้อม Counter (อีก) 4 ครั้ง
4. (สั่ง) output C0 เป็น Low พร้อม Delay 500 ms
5. (สั่ง) output C0 เป็น High

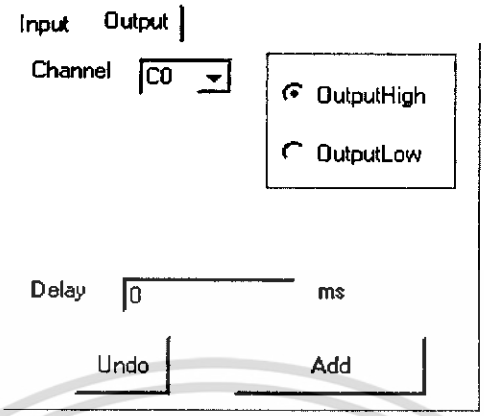
นำลำดับไปสั่งโปรแกรม

1.เลือก ตัวเลือกตามภาพ แล้วกดปุ่ม Add(หน้าต่างด้านซ้ายแสดงการทำงานที่สั่งไปแล้ว)

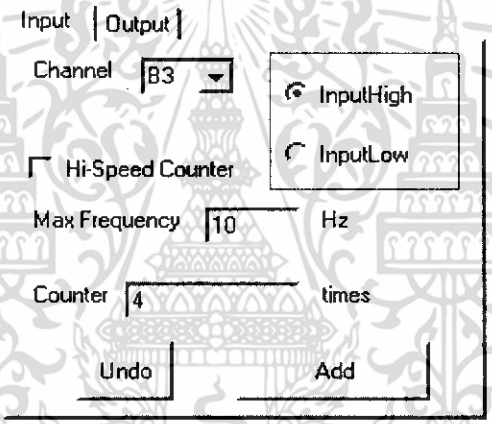


รูปที่ 4-1 สั่ง (รับ) input B3 เป็น high

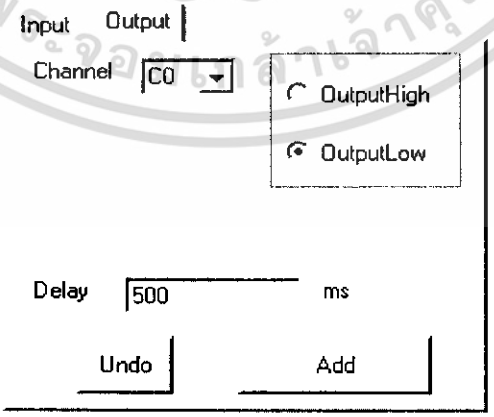
2.การสั่งข้อ 2 ถึง 5 เลือกตัวเลือกไปตามภาพ (กด Add ด้วย)



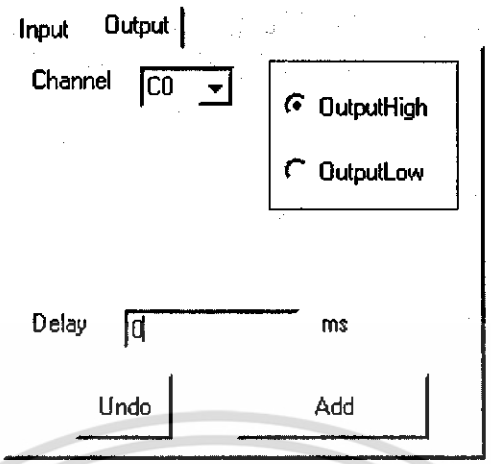
รูปที่ 4-2 (สั่ง) output C0 เป็น High



รูปที่ 4-3 (รับ) input B3 เป็น high พร้อม Counter (อีก) 4 ครั้ง

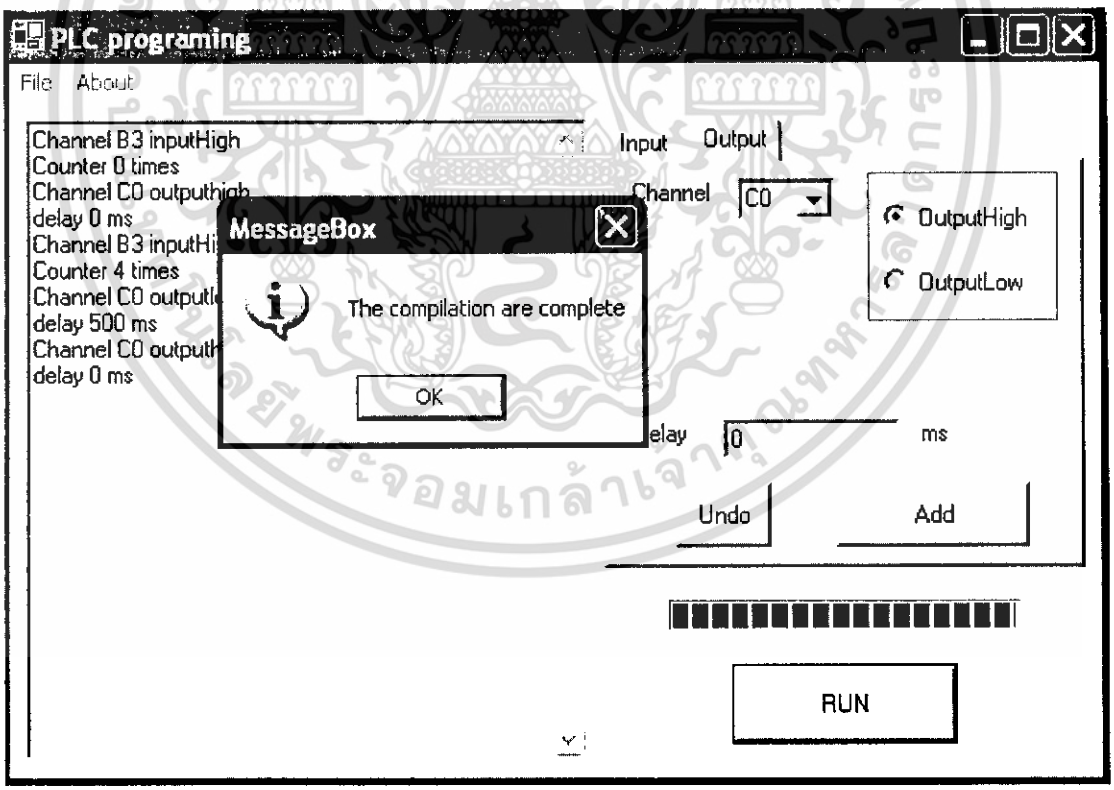


รูปที่ 4-4 (สั่ง) output C0 เป็น Low พร้อม Delay 500 ms



รูปที่ 4-5 (สั่ง) output C0 เป็น High

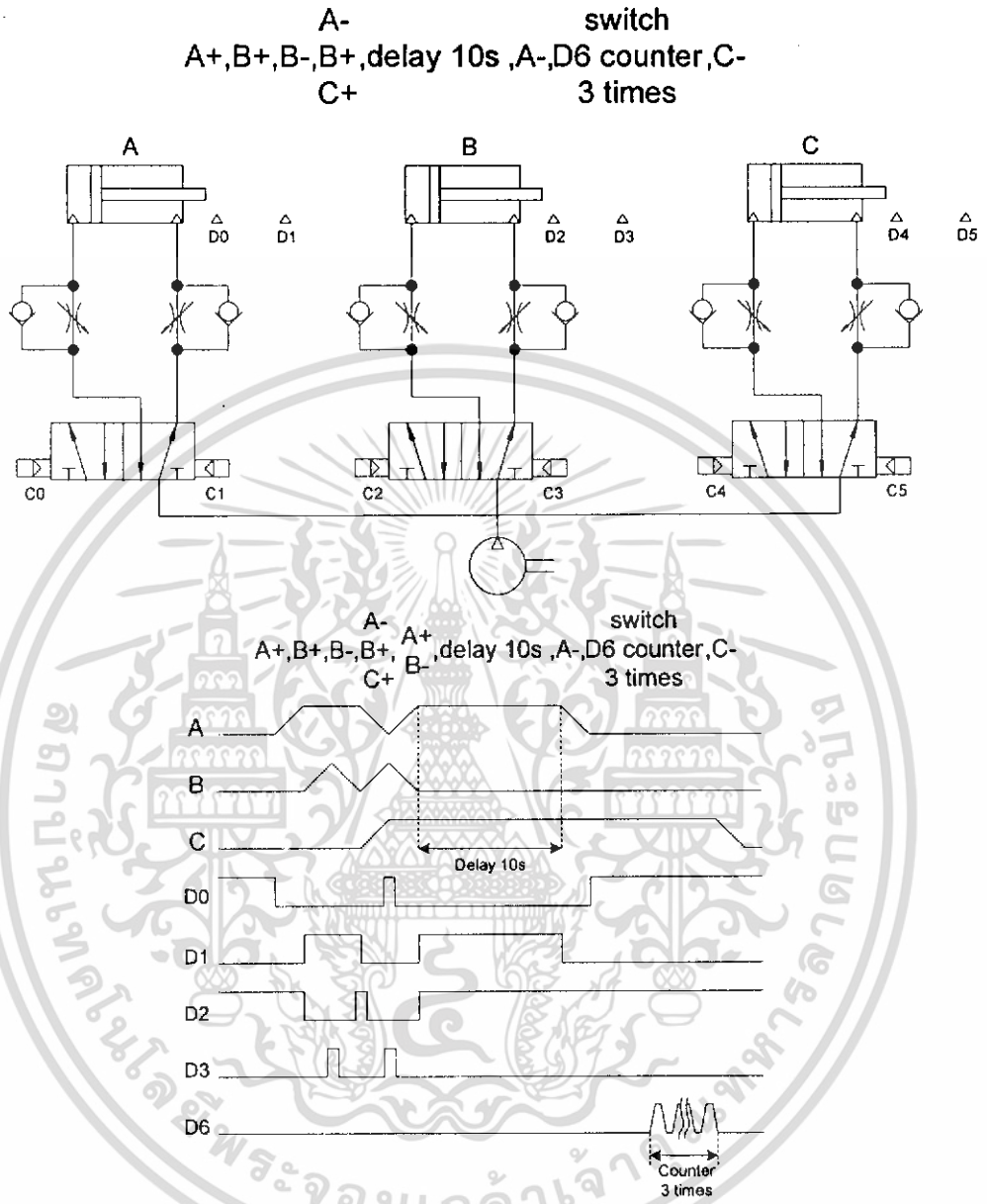
กดปุ่ม RUN โปรแกรมจะสร้าง TextFile ชื่อ PLC ออกมา เป็น Code ที่พร้อมนำไปโปรแกรมใส่ไมโครคอนโทรเลอร์ ต่อไป (ผ่าน Port ขนาน)



รูปที่ 4-6 compile เสร็จสมบูรณ์

4.2 ตัวอย่างวิธีการใช้โปรแกรมและ PLC ที่สร้างขึ้น กับระบบ Pneumatic

โจทย์ตัวอย่าง 2 เป็นตัวอย่างการใช้งานในระบบ Pneumatic



รูปที่ 4-7 วงจรและลำดับการทำงานของกระบอกสูบ

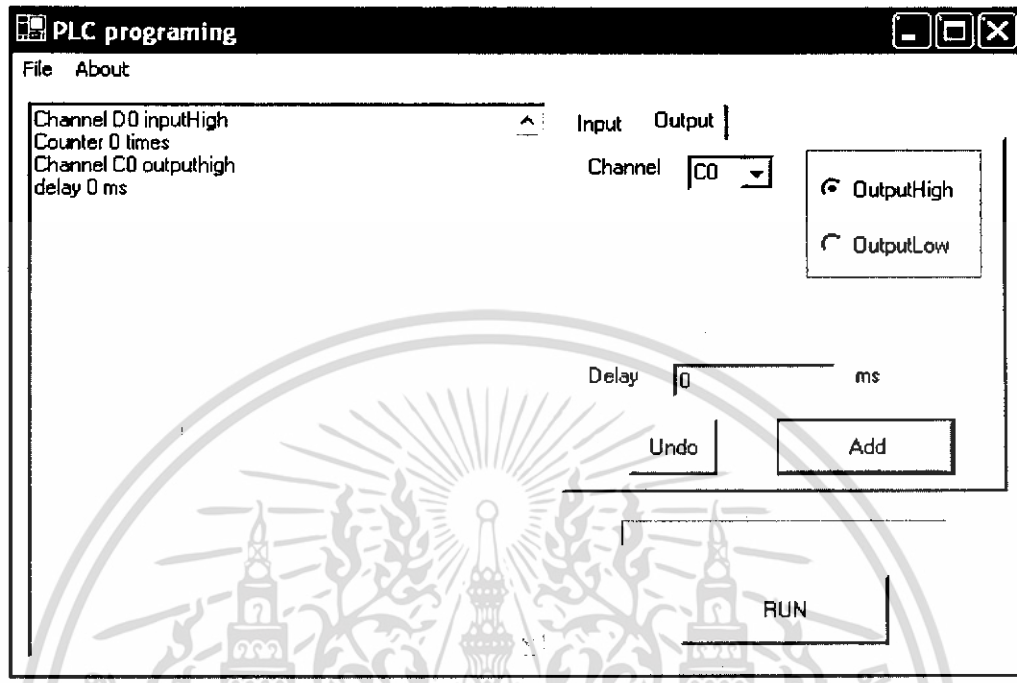
ลำดับการทำงานของวงจรนี้จะเห็นว่ามีช่วงที่มีไฟเข้าวาล์วตัวเดียว2ทาง(ขั้ว C0 และขั้ว C1)พร้อมกัน ในจังหวะ B+, B- การทำงานแบบนี้ปกติต้องใช้วงจรพวก แคสเคส หรือ ชีฟรีจีสเตอร์ เพื่อตัดไฟที่มาที่วาล์วพร้อมกันออกไปด้านหนึ่งก่อน แต่PLCเราตัดไฟด้วยsoftwareที่ทำได้ง่ายกว่า เอกลักษณ์อย่างหนึ่งของPLCเราคือการทำงานไปตาม Sequence (A+,B+,B-B+...) PLCจะมองเห็นแต่สวิทช์ที่อยู่ตาม Sequence ที่เราสั่ง สวิทช์อื่นที่แม้จะต่ออยู่แต่ทำงานไม่ตาม Sequence เรามันก็ไม่สนใจ เช่น สวิทช์ D1 จะทำให้ เกิด A+ หลังจากนั้นพอเปลี่ยนเป็นลำดับ B+ แล้วPLCจะไม่สนใจ สวิทช์ D1 อีก ผู้โปรแกรมPLCจึงสนใจแค่เรื่องเฉพาะหน้า

I/O	port	logic	delay	counter	Detail
out	C0	hi	-	-	A+
in	D1	hi	-	-	รับรู้ว่ากระบอกสูบ A+ เลื่อนสุดแล้ว
out	C2	hi	-	-	B+
in	D3	hi	-	-	รับรู้ว่ากระบอกสูบ B+ เลื่อนสุดแล้ว
out	C2	low	-	-	ตัดไฟ เมื่อB+เสร็จแล้ว
out	C3	hi	-	-	B-
in	D2	hi	-	-	รับรู้ว่ากระบอกสูบ B- เลื่อนสุดแล้ว
out	C0	low	-	-	ตัดไฟ เมื่อA+เสร็จแล้ว
out	C1	hi	-	-	A-
out	C3	low	-	-	ตัดไฟ เมื่อB-เสร็จแล้ว
out	C2	hi	-	-	B+
out	C4	hi	-	-	C+
in	D0	hi	-	-	รับรู้ว่ากระบอกสูบ A- เลื่อนสุดแล้ว
out	C1	low	-	-	ตัดไฟ เมื่อA-เสร็จแล้ว
out	C0	hi	-	-	A+
out	C2	low	-	-	ตัดไฟ เมื่อB+เสร็จแล้ว
out	C3	hi	-	-	B-
in	D2	hi	-	-	รับรู้ว่ากระบอกสูบ B- เลื่อนสุดแล้ว
out	C3	low	10000	-	ตัดไฟ เมื่อB-เสร็จแล้ว และdelay10s
out	C0	low	-	-	ตัดไฟ เมื่อA+เสร็จแล้ว
out	C1	hi	-	-	A-
in	D0	hi	-	-	รับรู้ว่ากระบอกสูบ A- เลื่อนสุดแล้ว
in	D6	hi	-	2	D6ถูกกด3ครั้ง(นับปกติไปแล้ว1ครั้ง จึงcounterอีก2)
out	C4	low	-	-	ตัดไฟ เมื่อC+เสร็จแล้ว
out	C5	hi	-	-	C-
in	D4	hi	-	-	รับรู้ว่ากระบอกสูบ C- เลื่อนสุดแล้ว
out	C5	low	-	-	ตัดไฟ เมื่อC-เสร็จแล้ว
out	C1	low	-	-	ตัดไฟ เมื่อA-เสร็จแล้ว
out	C3	low	-	-	ตัดไฟ เมื่อB-เสร็จแล้ว

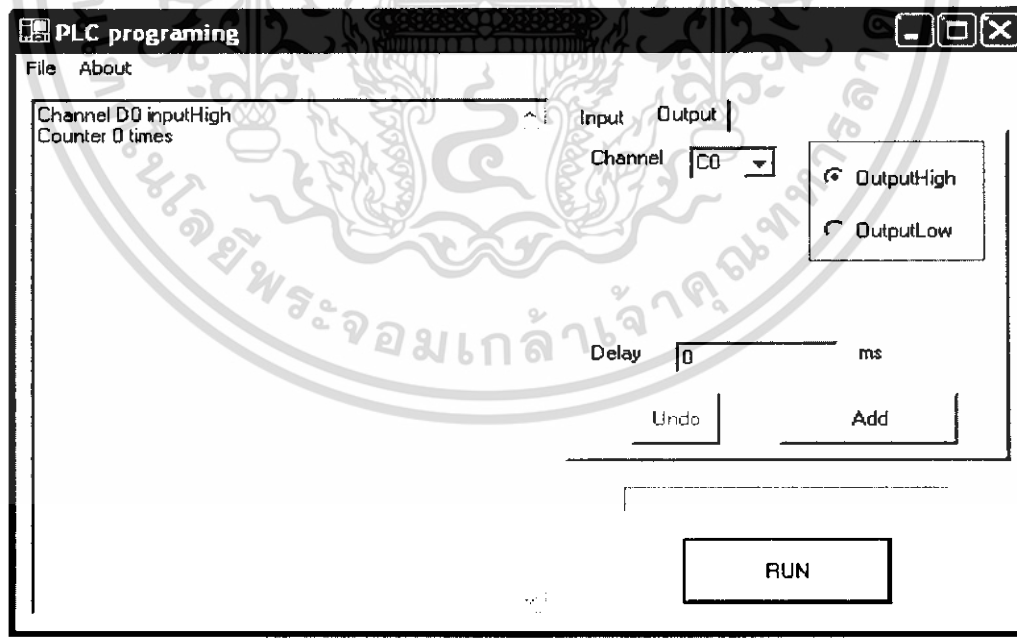
I/O	port	logic(hi, low)	delay	counter
Input | Output |	Channel 3	InputHigh InputLow	Delay 0	Counter 0

ตารางที่ 4-1 Sequence Diagrams ตามตัวอย่าง (ที่เป็นรูป) สามารถแปลงเป็นตารางได้ดังนี้

กรอกหรือเลือกค่าในตารางลงโปรแกรมตามช่อง(ดูรูปของชื่อช่องได้)เมื่อกรอกเสร็จแต่ละบรรทัดกดปุ่มAddเมื่อAddไปแล้วย้อนกลับไปแก้ได้ด้วยปุ่มUndo



รูปที่ 4-8 แก้ไขได้ด้วยการกดปุ่มUndo

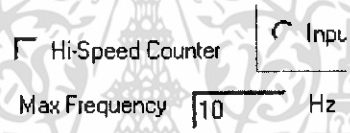


รูปที่ 4-9 หลังจาก Undoไปแล้วปุ่ม Undo จะจางไปไม่ให้กดได้อีก เพราะโปรแกรมมีหน่วยความจำสำรองให้ย้อนกลับได้ครั้งเดียว

การแก้ปัญหา Switch bounce ตามปกติจะใช้วงจร Low pass filter กรองสัญญาณรบกวนออกไป การใช้ Low pass filter เป็นวิธีพื้นฐานที่ใช้กันทั่วไป เมื่อPLC เราใช้งานกับสวิทช์ทั่วไปใช้กับตัดต่อ 4ครั้งต่อวินาทีโดยไม่มีผิดพลาด สวิทช์ความเร็วสูงผู้ผลิตเองก็ระบุความถี่ที่ใช้งานได้โดยไม่ผิดพลาดไว้ที่ 10 Hz ดังนั้นเราจึงออกแบบไว้ว่าความถี่สูงกว่า 20 Hz (เผื่อไว้2เท่า) เป็นสัญญาณรบกวนได้ และออกแบบ Low pass filter กรองความถี่สูงกว่า 20Hz ทิ้ง

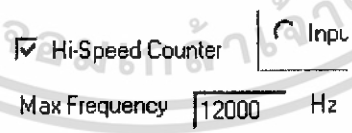
อย่างไรการป้องกัน Switch bounce ด้วยวงจร Low pass filter เป็นวิธีทาง Hardware หมายความว่าเมื่อสร้างเสร็จแล้วจะแก้ไขอะไรไม่ได้ (เพราะเป็น Hardware) เช่นเราสร้างไว้ใช้กับสวิทช์ออกแบบไว้กรองความถี่สูงกว่า 20Hz ทิ้ง หากจะนำไปใช้กับ photoelectric sensor ซึ่งทำงานได้ถึง 10 kHz และไม่มี Switch bounce (เนื่องจากไม่มีชิ้นส่วนเคลื่อนไหว) สัญญาณความถี่สูงจึงผ่าน Low pass filter ไม่ได้

ผู้สร้างได้คิดวิธีป้องกัน Switch bounce ด้วยวิธีทาง Software ขึ้นมา Hardware (Low pass filter) เมื่อกำหนดความถี่ Cut off (ความถี่ที่สัญญาณผ่านไม่ได้) ไปแล้วจะปรับเปลี่ยนไม่ได้อีก แต่วิธีทาง Software ที่คิดขึ้นมานี้จะสามารถเปลี่ยนความถี่ Cut off ได้โดยอิสระจากโปรแกรม ทำให้ PLC เรามีความยืดหยุ่นในการใช้งานมากขึ้น การใช้งานนั้นเราออกแบบโปรแกรมไว้ดังนี้



รูปที่ 4-10 ค่าเริ่มต้นของโปรแกรม ใช้กับสวิทช์ธรรมดา

โดยค่าเริ่มต้นนั้นโปรแกรมจะถูกตั้งไว้เป็น ใช้กับสวิทช์ธรรมดา ไม่ใช้เคาน์เตอร์ความเร็วสูง ความถี่ Cut off อยู่ที่ 10 Hz และจะไม่สามารถแก้ความถี่ cut off ในช่องได้ (สีทึบ) ในโหมดนี้ใช้กับสวิทช์ทั่วไปที่เกิด Switch bounce ได้ทันทีโดยไม่ต้องทำอะไร



รูปที่ 4-11 ใช้การนับสำหรับความเร็วสูง

หากเลือกที่ช่อง Hi-Speed Counter ช่องความถี่ cut off จะสามารถเปลี่ยนความถี่ได้ เราจะเลือกได้ว่าจะกั้นความถี่สูงกว่าเท่าใด โดยกรอกตัวเลขได้เลย เช่น photoelectric sensor ซึ่งทำงานได้ถึง 10 kHz เราอาจป้อนเผื่อไปเป็น 12 kHz ได้

บทที่ 5
บทวิจารณ์และสรุป

5.1 สรุปรายละเอียด (Specification) ของ PLC ที่สร้าง

20จุด I/O AC เอาท์พุตรีเลย์	input 12	output 8
DC เอาท์พุตรีเลย์	input 12	output 8
สายเคเบิลเชื่อมต่อ(IEEE 1284)	ยาว 0.95 M	
Software PLC V1.0	for Windows XP	
แหล่งจ่ายแรงดันไฟฟ้า/ความถี่	input 220 VAC, 50/60 Hz	
	output 2000mA ที่24VD	
ย่านแรงดันขณะทำงาน	AC 160 - 280 VAC	
	DC 20.4 - 26.4 VDC	
การใช้กำลังไฟฟ้า	20 W	
ช่วงอุณหภูมิทำงาน	0'c - 55'c	
เมื่อแหล่งจ่ายแรงดันหยุดชั่วขณะทนได้	50 ms	
น้ำหนัก	2000 g	
ช่วงเวลาทำงานในแต่ละคำสั่ง	0.4 - 1μs	
ความจุของโปรแกรม	8192 words	
หน่วยความจำรีจิสเตอร์	368 bytes	
หน่วยความจำข้อมูลอีอีพรอม	256 bytes	
ไทม์เมอร์	32 ตัว	
	ความละเอียด 1 ms	
เคาท์เตอร์	32 ตัว	
	ความเร็วสูงสุด 99999 Hz	
แรงดันอินพุต	แรงดัน on ต่ำสุด 2.6 V	
	แรงดัน off สูงสุด 0.8 V	
อินพุตอิมพีแดนซ์	1MΩ	
รีเลย์เอาท์พุตทนกระแส	10A ที่ 24VDC +10% -15%	
	2A ที่ 220VAC +5% -10%	
ความเร็วการสวิทช์รีเลย์	5 Hz	

ตารางที่ 5-1 คุณสมบัติของ PLC ที่สร้าง

รายการ	รายละเอียดทางเทคนิค
แรงดันอินพุต	24 VDC +10% / -15%
อินพุตอิมพีแดนซ์	470 Ω
กระแสด้านอินพุต(ทั่วไป)	50 mA
แรงดันตรวจจับต่ำสุด	3.6 V
อุณหภูมิทำงาน	-30°C ~ +100°C
ความต้านทานฉนวน(ระหว่างอินพุตและ CPU)	5000 Vrms
Terminal Capacitance	30 - 250 pF
Cut-off Frequency	80 kHz , -3dB
เวลาตอบสนอง (ขาขึ้น Rise)	4 - 18 µs
เวลาตอบสนอง (ขาลง Fall)	3 - 18 µs

ตารางที่ 5-2 คุณสมบัติของภาคอินพุต (อุปกรณ์เสริม)

5.2 ปัญหาที่พบและวิธีการแก้ไข

ปัญหาสวิทช์เกิดการกระดอนของหน้าสัมผัส สามารถแก้ได้โดยกระบวนการทางซอฟท์แวร์ รายละเอียดอยู่ในหัวข้อที่ 3.5 ระบบคอมพิวเตอร์ไวต่อระบบไฟฟ้ามาก ปัญหาการรีเซตตัวเองจากความผิดปกติของระบบจ่ายไฟ(ไฟตกไฟเกินสายไม่แน่น)กระบวนการแก้ไขแสดงอยู่ในหัวข้อ 2.2

5.3 ผลลัพธ์และแนวทางการพัฒนาต่อ

ไมโครคอนโทรเลอร์สามารถนำมาใช้แทนPLC ได้เป็นอย่างดี มีราคาถูกกว่ามาก เนื่องจากธุรกิจไมโครคอนโทรเลอร์ใหญ่กว่า PLC ไมโครคอนโทรเลอร์จึงมีการพัฒนาที่เร็วกว่า การใช้ไมโครคอนโทร-เลอร์จึงมีความเร็วการทำงานสูงกว่า และราคาถูกกว่า PLC โครงงานนี้สามารถแทน PLC ได้ในคำสั่งพื้นฐาน จากการทดสอบโครงงานสามารถประยุกต์กับงานได้หลากหลายมาก ด้านซอฟท์แวร์การสั่งงานแบบโครงสร้างช่วยให้เข้าใจการทำงานได้ง่ายกว่า

ความสามารถที่ควรพัฒนาต่อยอดไปอีก คือการทำงานกับระบบอนาลอก รับอินพุตเป็นอนาลอก และให้เอาพุตเป็นอนาลอก และการใช้งานเป็นอุปกรณ์ควบคุมแบบป้อนกลับ(Close Loop) จะเสริมความสามารถให้มีประโยชน์มากยิ่งขึ้น และทดลองยืนยันอายุการใช้งานของอุปกรณ์และความน่าเชื่อถือ (reliability) ของอุปกรณ์ การทนฝุ่นละอองความชื้นการสั่นสะเทือน เพื่อให้นำไปใช้งานจริงได้ในอุตสาหกรรม

บรรณานุกรม

[1] Karli watsan, David Espinosa, Zach Greenvoss, Jacop hammer Pederser, Christian Nagel, John D. Reid, Matthew Reynolds, Morgan Skinner, Eric white : " Beginning Visual C# " ,2002 By Wrox press.

[2] Semon Robinson, K. scott Allen, Zach Greenvoss, Christian Nagel, Morgan Skinner, Karli watsan : " Professional C# 2nd edtion " , 2002 By Wrox press.

[3] รศ. ธีรศิลป์ ทุมวิภาต, สุภาพร จำปาทอง : " เรียนรู้ PLC ขั้นต้นด้วยตนเอง " , SE-EDUCATION Public.

[4] รศ. ธีรศิลป์ ทุมวิภาต, สุภาพร จำปาทอง : " เรียนรู้ PLC ขั้นกลางด้วยตนเอง " , SE-EDUCATION Public.

[5] ผศ. เรวัต ธรรมาอธิรมย์ : " Start Microsoft Visual .NET " , SPC. BOOKS.

[6] สุรสิทธิ์ คิวประสมศักดิ์, นันทนี แขมโสภา : " อินไซด์ Visual Basic .NET ฉบับสมบูรณ์ " , Provision.

[7] บัญชา ปะสีละเตสัง : " เริ่มต้นการเขียนโปรแกรม Visual Basic .NET " , SE- EDUCATION Public.

[8] ประจิน พลังสันติกุล : " เรียนรู้การใช้งาน ccsc คอมไพเลอร์Innovative Emperiment Co.,Ltd.

[9] ณัฏฐพล วงศ์สุนทรชัย, ชัยวัฒน์ ลิ้มพรจิตรวิไล : " ปฏิบัติการไมโครคอนโทรลเลอร์ PIC 16F87x " ,Innovative Emperiment Co.,Ltd.

[10] สัจจะ จรัสรุ่งรวีวร : " คู่มือการเขียนโปรแกรมและใช้งาน Visual Basic 6 " , Info Press.