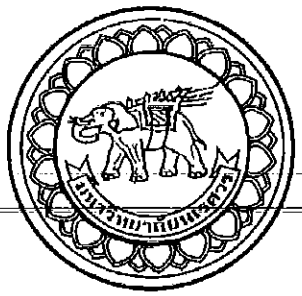


การควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ

INDUCTION MOTOR SPEED CONTROL

นายวราชัย จาดแดง รหัส 45362340

นายอุรุพงศ์ สมนึก รหัส 45363256

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร

ปีการศึกษา 2549

ใบรับรองโครงงานวิศวกรรม

หัวข้อโครงงาน	การควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ	
ผู้ดำเนินโครงงาน	นายวราชัย จาดแดง	รหัส 45362340
	นายอุรุพงศ์ สมนึก	รหัส 45363256
อาจารย์ที่ปรึกษา	ดร.สมพร เรืองสินชัยวานิช	
สาขาวิชา	วิศวกรรมไฟฟ้า	
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์	
ปีการศึกษา	2549	

.....

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวรอนุมัติให้โครงงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า

คณะกรรมการสอบโครงงานวิศวกรรม

..... ประธานกรรมการ
(ดร.สมพร เรืองสินชัยวานิช)

..... กรรมการ
(ดร.สมยศ เกียรติวนิชวิไล)

..... กรรมการ
(อาจารย์ ปิยดนัย ภาชนะพรรณ์)

หัวข้อโครงงาน	การควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ	
ผู้ดำเนินโครงงาน	นายวราชัย จาดแดง	รหัส 45362340
	นายอุรุพงศ์ สมนึก	รหัส 45363256
อาจารย์ที่ปรึกษา	ดร.สมพร เรืองสินชัยวานิช	
สาขาวิชา	วิศวกรรมไฟฟ้า	
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์	
ปีการศึกษา	2549	

.....

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการสร้างชุดอินเวอร์เตอร์ 3 เฟสสำหรับควบคุมความเร็วรอบมอเตอร์ไฟฟ้าเหนี่ยวนำขนาดไม่เกิน 0.5 แรงม้า โดยใช้ไฟฟ้า 1 เฟส 220 โวลต์ 50 เฮิรตซ์ ซึ่งอินเวอร์เตอร์ชุดนี้จะใช้อุปกรณ์เพาเวอร์มอสเฟต (Power Mosfet) เป็นตัวสวิตช์ในภาคกำลังของอินเวอร์เตอร์และใช้หลักการควบคุมสวิตช์แบบ Sinusoidal Pulse-Width Modulator (PWM) ซึ่งใช้ dsPIC30F2010 เป็นตัวสร้างสัญญาณพัลส์ PWM

Project Title	Induction motor speed control	
Name	Mr. Warachai Jaddang	ID. 45362340
	Mr. Urupong Somnuek	ID. 45363256
Project Advisor	Dr. Somporn Ruangsinchaiwanich	
Major	Electrical Engineering	
Department	Electrical and Computer Engineering	
Academic year	2006	

ABSTRACT

The Purpose of this project is contributions three phase inverter for control speed induction motors and drive load not over 0.5 HP. Input of this inverter is AC single phase 220 volt 50 Hz. This inverter using MOSFET for power switching device. The principle of inverter is operated with sinusoidal Pulse-Width Modulator and use dsPIC30F2010 build signal Pulse-Width Modulation (PWM).

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี ด้วยความช่วยเหลือจากหลายๆท่าน ผู้จัดทำจึงถือโอกาสนี้ขอกราบขอบพระคุณ

ขอขอบคุณอาจารย์สมพร เรืองสินชัยวานิช ซึ่งเป็นอาจารย์ที่ปรึกษา และคณะกรรมการสอบโครงงานทุกท่านที่ได้ให้คำปรึกษาชี้แนะแนวทางและข้อคิดเห็นต่างๆ ในการแก้ปัญหาที่เป็นประโยชน์อย่างสูงในการทำโครงการนี้ให้สำเร็จลุล่วงด้วยดี

ขอขอบคุณอาจารย์ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ทุกท่านที่ได้ประสิทธิ์ประศาสตร์วิชาความรู้ความสามารถเพียวพอที่จะทำโครงงานและทำงานได้จริง

ขอขอบคุณ พี่เนา พี่ใหม่ ที่แนะนำข้อมูล เอกสาร และพาไปหาซื้ออุปกรณ์ที่ใช้ในโครงงานนี้

ขอขอบคุณพี่ๆเพื่อนๆและน้องๆ นิสิตภาควิชาวิศวกรรมไฟฟ้าทุกคนที่ได้ให้ความช่วยเหลือในหลายๆด้าน ทั้งเรื่องเรียนและเรื่องส่วนตัวด้วยดีเสมอมา

ท้ายนี้ผู้จัดทำโครงงานขอกราบขอพระคุณบิดา มารดา และญาติพี่น้องของข้าพเจ้าที่เลี้ยงดูและคอยสนับสนุนด้านการเงิน รวมทั้งเป็นกำลังใจ ให้ผู้จัดทำเสมอมาจนสำเร็จการศึกษา

คณะผู้จัดทำโครงงาน
นายวราชัย จาดแดง
นายอุรุพงศ์ สมนึก

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญ

ในปัจจุบันนี้งานอุตสาหกรรมการผลิตหลายๆอย่างต้องใช้มอเตอร์เป็นเครื่องต้นกำลังในการขับเคลื่อน ซึ่งส่วนใหญ่แล้วจะเป็นมอเตอร์ไฟฟ้ากระแสสลับและต้องมีการควบคุมความเร็วรอบของมอเตอร์เพื่อใช้งานในลักษณะที่ต่างกัน

ดังนั้นผู้เสนอโครงงานจึงมีความสนใจที่จะทำการศึกษาและค้นคว้าข้อมูลเกี่ยวกับการควบคุมความเร็วรอบมอเตอร์เหนี่ยวนำโดยโครงงานที่ศึกษานี้จะเป็นการสร้างเครื่องควบคุมความเร็วรอบมอเตอร์ 3 เฟส หรือเรียกกันว่า อินเวอร์เตอร์ 3 เฟส ซึ่งใช้เพาเวอร์มอสเฟตเป็นตัวสวิตช์และใช้การควบคุมแบบพีดับบิวเอ็มโดยสร้างสัญญาณพัลส์พีดับบิวเอ็มจาก dsPIC Microcontroller

1.2 วัตถุประสงค์ของโครงงาน

- เพื่อออกแบบและสร้างอินเวอร์เตอร์ 3 เฟส (3 Phase Inverter) ที่สามารถควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส โดยการปรับค่าความถี่ได้

1.3 ขอบข่ายของโครงงาน

- สร้างอินเวอร์เตอร์ 3 เฟส (3 Phase Inverter) ที่ใช้ไฟฟ้า 1 เฟส 220 โวลต์ 50 เฮิรตซ์
- ศึกษาอินเวอร์เตอร์ 3 เฟส (3 Phase Inverter) ที่ใช้ควบคุมความเร็วรอบมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส 220/380 โวลต์ ขนาด 0.5 แรงม้า และสามารถปรับความถี่ได้
- ศึกษาอินเวอร์เตอร์ 3 เฟส (3 Phase Inverter) ที่ใช้มอสเฟต (MOSFET) เป็นตัวสวิตช์ของอินเวอร์เตอร์และที่ใช้ dsPIC30F2010 เป็นตัวสร้างสัญญาณพีดับเบิ้ลยูเอ็มแบบไซน์ (PWM: Sinusoidal Pulse-Width Modulator) ด้วยการเขียนโปรแกรมภาษาซี

1.4 ขั้นตอนการดำเนินงาน

- ศึกษาและค้นคว้าข้อมูลเกี่ยวกับการควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส โดยการปรับความถี่
- ศึกษาและค้นคว้าข้อมูลเกี่ยวกับมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส, อินเวอร์เตอร์ 3 เฟสและมอสเฟสกำลัง
- ศึกษาและค้นคว้าข้อมูลเกี่ยวกับ dsPIC Microcontroller

- ออกแบบและสร้างอินเวอร์เตอร์ 3 เฟสและส่วนควบคุมความถี่
- ทดสอบและทดลองการทำงานของอินเวอร์เตอร์และส่วนควบคุมความถี่
- สรุปการทดลองและจัดทำรูปเล่ม

1.5 การดำเนินงาน

ขั้นตอนการดำเนินงาน	ปี 2548		ปี 2549									
	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ย.	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.
1. ศึกษาและค้นคว้าข้อมูลเกี่ยวกับการควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟสโดยการปรับความถี่	↔											
2. ศึกษาและค้นคว้าข้อมูลเกี่ยวกับมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส, อินเวอร์เตอร์ 3 เฟส และมอสเฟสกำลัง		↔										
3. ศึกษาและค้นคว้าข้อมูลเกี่ยวกับdsPIC Microcontroller			↔	↔								
4. ออกแบบและสร้างอินเวอร์เตอร์ 3 เฟสและส่วนควบคุม					↔	↔	↔	↔	↔			
5. ทดสอบและทดลองการทำงานของอินเวอร์เตอร์และส่วนควบคุมความถี่										↔	↔	
6. สรุปการทดลองและจัดทำรูปเล่ม												↔

ตารางที่ 1.1 ขั้นตอนในการดำเนินงาน

1.6 ผลที่คาดว่าจะได้รับ

- ได้อินเวอร์เตอร์ 3 เฟส (3 Phase Inverter) ที่สามารถควบคุมความเร็วรอบมอเตอร์ไฟฟ้าเหนี่ยวนำโดยการปรับความถี่ จำนวน 1 เครื่อง
- สามารถนำอินเวอร์เตอร์ 3 เฟส (3 Phase Inverter) ที่ออกแบบและสร้างขึ้นไปใช้ในการควบคุมความเร็วรอบมอเตอร์ไฟฟ้าเหนี่ยวนำได้จริง
- ได้ความรู้เกี่ยวกับอินเวอร์เตอร์ 3 เฟสและเรื่อง dsPIC Microcontroller
- สามารถนำอินเวอร์เตอร์ 3 เฟสที่สร้างขึ้นไปใช้ในการศึกษาของนักศึกษาได้

1.7 งบประมาณของโครงงาน

- ค่าถ่ายเอกสารและเข้าเล่มรายงาน	500 บาท
- ค่าอุปกรณ์และหนังสือที่เกี่ยวข้องกับโครงงาน	1,500 บาท
รวมเป็นเงิน	2,000 บาท

บทที่ 2
หลักการและทฤษฎีที่เกี่ยวข้อง

การจัดทำโครงงานอินเวอร์เตอร์สามเฟสนี้ ได้ศึกษาเนื้อหาที่เกี่ยวข้องดังต่อไปนี้ หลักการและทฤษฎีที่เกี่ยวข้อง แบ่งออกเป็นส่วน ๆ ได้ดังนี้

2.1 มอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส (3 Phase Induction Motor)

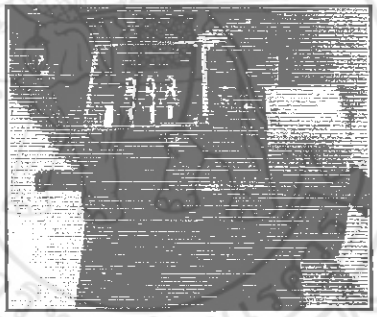
2.2 อินเวอร์เตอร์ (Inverter)

2.3 มอสเฟสกำลัง (Power Mosfet)

2.4 ข้อมูลเบื้องต้นของ dsPIC30F2010

2.5 การใช้งานโมดูล MCPWM ใน dsPIC30F10 เพื่อควบคุมมอเตอร์

2.1 มอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส (3 Phase Induction Motor)



รูปที่ 2.1 มอเตอร์เหนี่ยวนำ (Induction Motor)

มอเตอร์ที่นิยมใช้กันมากที่สุดในอุตสาหกรรมคือมอเตอร์ไฟฟ้าเหนี่ยวนำชนิดกรงกระรอก เนื่องจาก

-เชื่อถือได้สูงและต้องการบำรุงรักษาต่ำ

-ราคาถูกโดยเปรียบเทียบระหว่างมอเตอร์ไฟฟ้ากระแสตรงกับมอเตอร์เหนี่ยวนำที่กำลังเอาต์พุตเท่ากัน

- มีขนาดเล็กกว่ามอเตอร์ไฟฟ้ากระแสตรงที่กำลังเอาต์พุตเท่ากัน

- ความเร็วรอบไม่ขึ้นกับโหลดมากนัก

- นำไปใช้ในสภาพแวดล้อมที่เสี่ยงอันตรายได้

2.1.1 หลักการทำงานของมอเตอร์เหนี่ยวนำ 3 เฟส

จ่ายไฟกระแสสลับ 3 เฟสให้ขดลวดอาร์เมเจอร์ที่สเตเตอร์ จะเกิดสนามแม่เหล็กหมุนเมื่อฟลั๊กแม่เหล็กของสนามแม่เหล็กหมุนเคลื่อนตัวตัดตัวนำที่ฝังอยู่ในโรเตอร์จะเกิดการเหนี่ยวนำและเนื่องจากโรเตอร์ถูกลัดวงจรจึงเกิดแรงดันไฟฟ้าเหนี่ยวนำและแรงบิดเป็นผลให้โรเตอร์หมุนไปในทิศทางเดียวกับสนามแม่เหล็กหมุนเกิดเป็นความเร็วและเรียกความเร็วนี้ว่าความเร็วซิงโครนัส (Synchronous speed) โดยความเร็วซิงโครนัสที่มี ขั้ว P และความถี่ f จะสามารถหาได้จากสูตร

$$n_s = \frac{120f}{p} \tag{2.1}$$

สมการต่างๆที่มีความสำคัญสำหรับการควบคุมความเร็วรอบความเร็วของมอเตอร์เหนี่ยวนำโดยการปรับความถี่จะสามารถสรุปได้ดังนี้

$$\omega_s = k_7 f \tag{2.2}$$

$$s = \frac{\omega_s - \omega_r}{\omega_s} \tag{2.3}$$

$$f_r = sf \tag{2.4}$$

$$\omega_s = k_3 \phi_{ag} f \tag{2.5}$$

$$\omega_s = k_4 \phi_{ag} f_r \tag{2.6}$$

ความสัมพันธ์ของสมการต่างๆดังกล่าวทำให้สามารถสรุปความสัมพันธ์ระหว่างแต่ละค่าได้ดังนี้

- ค่า Synchronous speed เปลี่ยนแปลงตามการปรับความถี่ของแรงดันที่จ่าย
- ที่ความถี่ต่ำ ค่ากำลังงานสูญเสียที่เกิดจากความต้านทานภายในมอเตอร์และ slip frequency จะมีค่าน้อย ดังนั้นที่สภาวะ steady state ค่า slip frequency ควรมีค่าไม่เกินพิกัดความถี่ที่ระบุไว้บน nameplate
- ที่ slip frequency ต่ำๆ โดยความถี่ของแหล่งจ่ายคงที่ จะพบว่า slip (s) จะมีค่าน้อยตาม slip frequencyและการปรับความเร็วของมอเตอร์จะแปรผันอย่างเป็นเชิงเส้นกับความถี่ของแหล่งจ่าย
- สำหรับแรงบิดของมอเตอร์จะมีค่าเท่ากับค่าที่พิกัด เมื่อความถี่ที่จ่ายให้มอเตอร์มีค่าเท่ากับความถี่ของแหล่งจ่ายและค่า ควรมีค่าคงที่และเท่ากับค่าที่พิกัด

จากความสำคัญ 4 ข้อ ที่กล่าวมาข้างต้น ทำให้เราสรุปได้ว่าความเร็วรอบของมอเตอร์เหนี่ยวนำสามารถเปลี่ยนแปลงได้โดยการควบคุมความถี่ f ของแหล่งจ่ายไฟที่ให้กับมอเตอร์

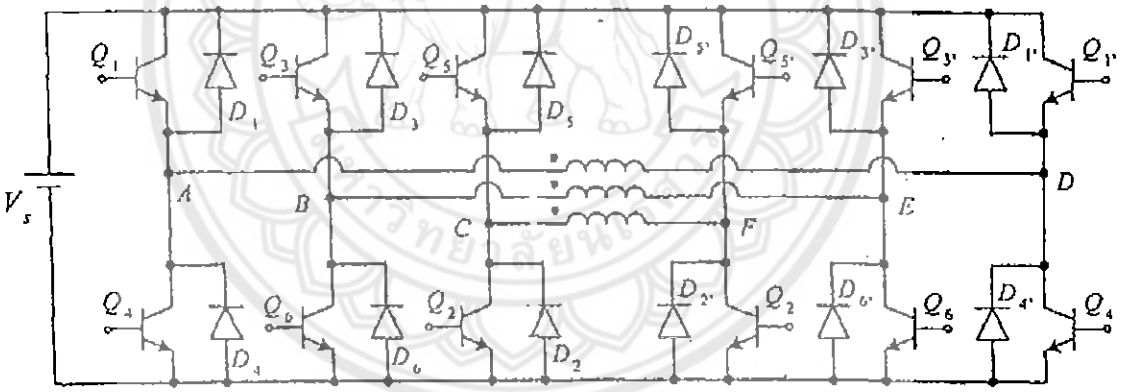
2.2 อินเวอร์เตอร์ (Inverter)

เมื่อพิจารณาอินเวอร์เตอร์ตามโครงสร้างและการนำไปใช้งานแล้วจำแนกได้เป็น 2 ประเภท คือ อินเวอร์เตอร์แบบป้อนแรงดัน (Voltage Fed Inverter) อินเวอร์เตอร์แบบนี้มีแรงดันไฟตรงมีค่าคงที่และอินเวอร์เตอร์แบบป้อนกระแส (Current Fed Inverter) อินเวอร์เตอร์แบบนี้มีกระแสคงที่

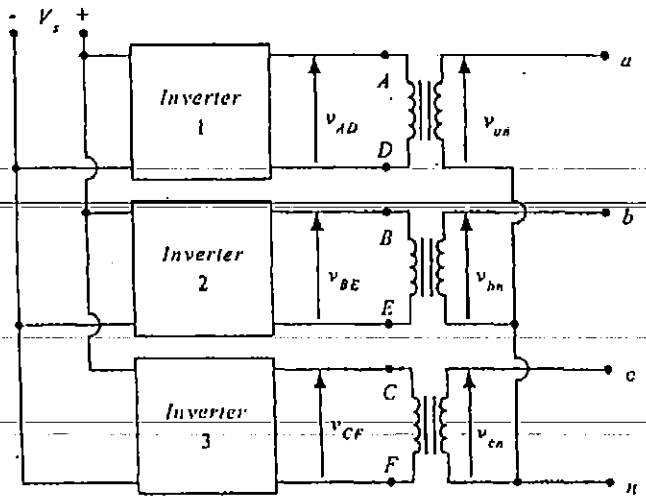
ในที่นี้จะขอกล่าวเฉพาะอินเวอร์เตอร์แบบป้อนแรงดัน (Voltage Fed Inverter) ชนิด 3เฟส เท่านั้น

2.2.1 อินเวอร์เตอร์ 3 เฟส

อินเวอร์เตอร์ 3 เฟส มักนำไปใช้ในงานที่ใช้งานที่ใช้กำลังสูงๆ เราสามารถนำวงจรกึ่งบริดจ์ (หรือบริดจ์) อินเวอร์เตอร์ 1 เฟส 3 วงจรมาต่อเป็นอินเวอร์เตอร์ 3 เฟสได้ ดังแสดงในรูปที่ 2.2 โดยที่สัญญาณขับนำทรานซิสเตอร์ของแต่ละวงจรจะต้องสวิง (Swing) กัน 120 องศา เพื่อจะทำให้ได้แรงดันสมดุล 3 เฟส ขดลวดหม้อแปลงด้านปฐมภูมิต้องแยกโดดจากกัน ส่วนขดลวดด้านทุติยภูมิ สามารถต่อได้ทั้งแบบสตาร์และเดลต้า โดยปกติแล้วมักต่อหม้อแปลงด้านทุติยภูมิแบบสตาร์ เพื่อจำกัดผลของฮาร์โมนิก 3, 9, 15 ได้ การต่อวงจรแสดงไว้ดังรูป 2.3 ซึ่งวงจรวงจรต้องใช้ทรานซิสเตอร์และไดโอดอย่างละ 12 ตัว ถ้าแรงดันด้านออกของอินเวอร์เตอร์ 1 เฟสของแต่ละวงจรมีขนาดและความถี่ไม่เท่ากัน ก็จะทำให้แรงดันด้านออกของอินเวอร์เตอร์ 3 เฟสไม่สมดุลไปด้วย

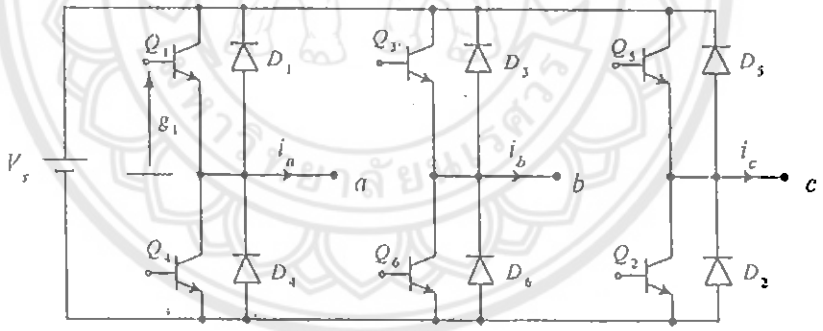


รูปที่ 2.2 อินเวอร์เตอร์ 1 เฟส จำนวน 3 วงจร ต่อเป็นอินเวอร์เตอร์ 3 เฟส



รูปที่ 2.3 ขดลวดทุติยภูมิของหม้อแปลงไฟฟ้าซึ่งต่อแบบสตาร์หรือเดลต้า

อินเวอร์เตอร์ 3 เฟสสามารถต่อได้จากทรานซิสเตอร์และไดโอดได้อย่างละ 6 ตัวได้ดังรูปที่ 2.4 สัญญาณควบคุมการขับสวิตช์สามารถสร้างได้ 2 รูปแบบคือ ช่วงการนำกระแส 180 องศา และช่วงการนำกระแส 120 องศา

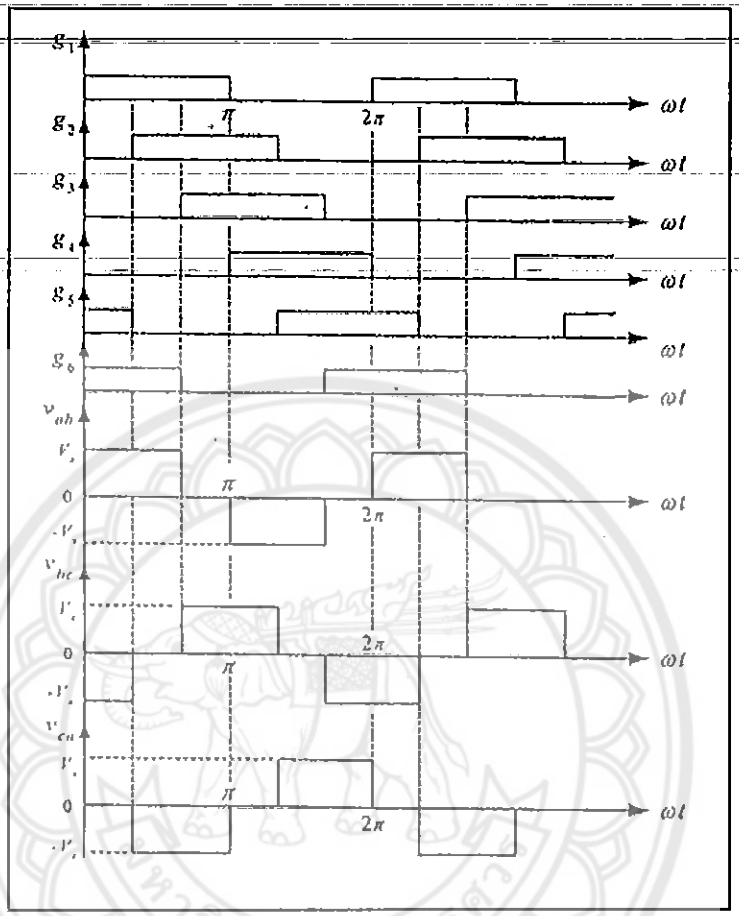


รูปที่ 2.4 อินเวอร์เตอร์ 3 เฟส ซึ่งใช้ทรานซิสเตอร์และไดโอด

A. การนำกระแส 180 องศา

ทรานซิสเตอร์แต่ละตัวจะนำกระแส 180 องศา โดยแต่ละช่วงจะมีทรานซิสเตอร์ 3 ตัวนำกระแสพร้อมกัน เมื่อทรานซิสเตอร์ Q_1 นำกระแสขั้ว a ถูกต่อเข้ากับขั้วบวกของ แหล่งจ่ายไฟตรงและเมื่อทรานซิสเตอร์ Q_4 นำกระแสขั้ว a ถูกต่อเข้ากับขั้วลบของแหล่งจ่ายไฟตรง การทำงานในแต่ละไซเกิลถูกแบ่งออกเป็น 6 โหมดใน 1 คาบ โหมดละ 60 องศา ลำดับการนำกระแสของทรานซิสเตอร์แสดงในรูปที่

2.5 ซึ่งสัญญาณขับทรานซิสเตอร์แต่ละตัวจะมีการเลื่อนเฟสไปตัวละ 60 องศา เพื่อให้แรงดันด้านออกที่สมดุล



รูปที่ 2.5 รูปคลื่นของวงจรบริดจ์อินเวอร์เตอร์ 3 เฟส

โหลดของวงจรอาจต่อได้ทั้งแบบสตาร์และแบบเดลต้า กรณีต่อโหลดแบบเดลต้า กระแสเฟสสามารถหาได้โดยตรงจากแรงดันระหว่างสายและสามารถหาค่ากระแสสายได้ ส่วนกรณีเมื่อต่อโหลดแบบสตาร์ แรงดันระหว่างสายกับนิวทรัลหาได้จากกระแสสาย (หรือเฟส) ดังแสดงในรูปที่ 2.6 แสดงการทำงาน 3 โหมดในช่วงครึ่งคาบ

โหมด 1 $(0 \le \omega t \le \pi/3)$

$$R_{eq} = R + \frac{R}{2} = \frac{3R}{2}$$

$$i_1 = \frac{V_s}{R_{eq}} = \frac{2V_s}{3R}$$

$$v_{an} = v_{cn} = \frac{i_1 R}{2} = \frac{V_s}{3}$$

$$v_{bn} = -i_1 R = \frac{-2V_s}{3}$$

โหมด 2 ($\pi/3 \le \omega t < 2\pi/3$)

$$R_{eq} = R + \frac{R}{2} = \frac{3R}{2}$$

$$i_2 = \frac{V_s}{R_{eq}} = \frac{2V_s}{3R}$$

$$v_{an} = i_2 R = \frac{2V_s}{3}$$

$$v_{bn} = v_{cn} = \frac{-i_2 R}{2} = \frac{-V_s}{3}$$

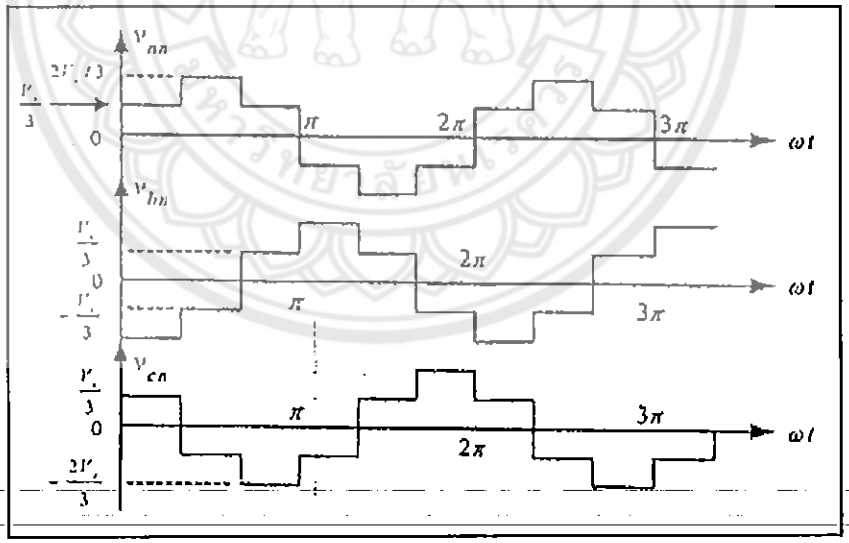
โหมด 3 ($2\pi/3 \le \omega t < \pi$)

$$R_{eq} = R + \frac{R}{2} = \frac{3R}{2}$$

$$i_3 = \frac{V_s}{R_{eq}} = \frac{2V_s}{3R}$$

$$v_{an} = v_{cn} = \frac{i_3 R}{2} = \frac{V_s}{3}$$

$$v_{bn} = -i_3 R = \frac{-2V_s}{3}$$



รูปที่ 2.6 แรงดันด้านออกเมื่อต่อโหลดตัวต้านทานแบบสตาร์

แรงดันระหว่างสายกับนิวทรัลของอินเวอร์เตอร์ 3 เฟส เมื่อโหลดเป็นตัวต้านทานแสดงได้ดังรูปที่ 2.6 ส่วนแรงดันระหว่างสาย v_{ab} แสดงได้จากรูปที่ 2.5 ซึ่งสามารถเขียนให้อยู่ในรูปของอนุกรมฟูเรียร์ได้โดยแรงดัน v_{ab} จะเลื่อนเฟสไป $\pi/6$ และไม่มีฮาร์มอนิกลำดับคู่

$$v_{ab} = \sum_{n=1,3,5..}^{\infty} \frac{4V_s}{n\pi} \cos\frac{n\pi}{6} \sin n(\omega t + \frac{\pi}{6}) \quad (2.7)$$

แรงดัน v_{bc} และ v_{ca} สามารถหาได้จากสมการ 2.7 โดยเลื่อนเฟสจาก v_{ab} ไป 120 และ 240 องศา ตามลำดับ

$$v_{bc} = \sum_{n=1,3,5..}^{\infty} \frac{4V_s}{n\pi} \cos\frac{n\pi}{6} \sin n(\omega t - \frac{\pi}{2}) \quad (2.8)$$

$$v_{ca} = \sum_{n=1,3,5..}^{\infty} \frac{4V_s}{n\pi} \cos\frac{n\pi}{6} \sin n(\omega t - \frac{7\pi}{6}) \quad (2.9)$$

จากสมการที่ 2.7, 2.8 และ 2.9 จะได้แรงดันสายที่ฮาร์มอนิกอันดับสามเท่า (n =3, 9, 15...) มีค่าเท่ากับศูนย์ แรงดันอาร์เอ็มเอสระหว่างสายด้านออกหาได้จาก

$$V_L = [\frac{2}{2\pi} \int_0^{2\pi/3} V_s^2 d(\omega t)]^{1/2} = \sqrt{\frac{2}{3}} V_s = 0.8165 V_s \quad (2.10)$$

จากสมการ 2.7 แรงดันสายอาร์เอ็มเอสของฮาร์มอนิกที่ n ของมีค่า

$$V_{Ln} = \frac{4V_s}{\sqrt{2}n\pi} \cos\frac{n\pi}{6} \quad (2.11)$$

ถ้า n=1 ส่วนประกอบหลักมูลของสายแรงดันมีค่า

$$V_{L1} = \frac{4V_s}{\sqrt{2}\pi} \cos 30° = 0.7797 V_s \quad (2.11)$$

แรงดันอาร์เอ็มเอสระหว่างสายกับนิวทรัลหาได้จากแรงดันสายได้เป็น

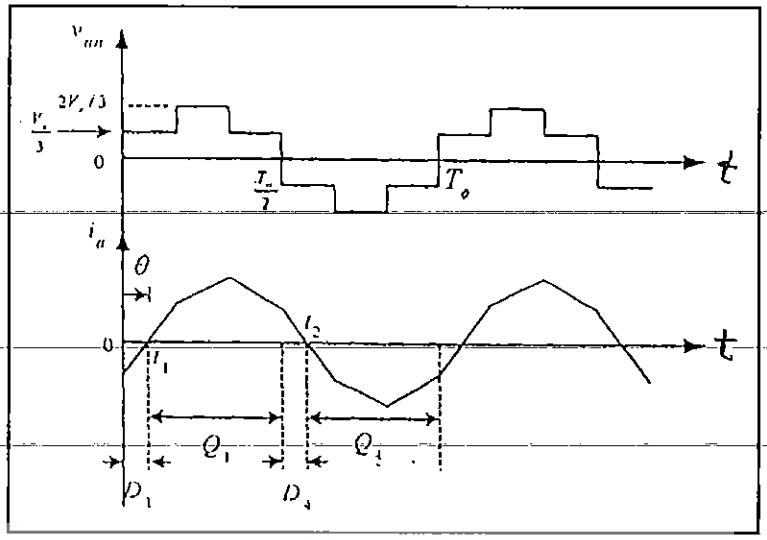
$$V_p = \frac{V_L}{\sqrt{3}} = \frac{\sqrt{2}V_s}{3} = 0.4714 V_s \quad (2.12)$$

ถ้าโหลดเป็นตัวต้านทาน ไดโอดที่ต่อคร่อมทรานซิสเตอร์จะทำงานทั้งนี้เพราะตัวต้านทานเป็นตัวอุปกรณ์ ที่ไม่สามารถเก็บพลังงานไฟฟ้าได้ แต่เมื่อโหลดเป็นตัวเหนี่ยวนำ กระแสโหลดจะล้าหลังแรงดันดังแสดงในรูป 2.7 เมื่อทรานซิสเตอร์ Q_4 ในรูป 2.4 หยุดนำกระแส กระแส i_a ส่วนที่เป็นลบจะไหลผ่านไดโอด D_1 จนกระแสโหลดกลับขั้วที่เวลา t_1ช่วงเวลา $0 \le t \le t_1$ ทรานซิสเตอร์ Q_1ไม่นำกระแส ทรานซิสเตอร์ Q_4จะนำกระแสที่ช่วงเวลา t_2 ช่วงเวลาการนำกระแสของทรานซิสเตอร์และไดโอดจะขึ้นอยู่กับค่าตัวประกอบกำลังของไดโอด

กรณีโหลดต่อวงจรแบบวาย แรงดันเฟสมีค่า $v_{an} = v_{ab}/\sqrt{3}$ และจะมีมุมประวิง 30 องศาจากสมการ กระแสสาย i_a กรณีโหลดประเภท RL จะได้เป็น

$$i_a = \sum_{n=1,3,5...}^{\infty} [\frac{4V_s}{\sqrt{3}n\pi\sqrt{R^2 + (n\omega L)^2}} \cos\frac{n\pi}{6}] \sin(n\omega t - \theta_n) \quad (2.13)$$

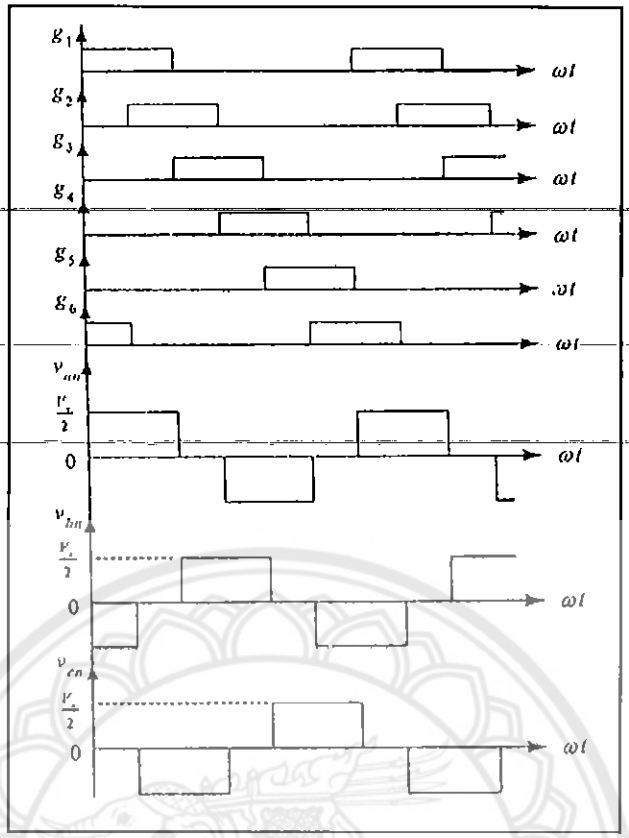
โดยที่ $\theta_n = \tan^{-1}(n\omega L / R)$



รูปที่ 2.7 รูปคลื่นเอาท์พุทของอินเวอร์เตอร์ 3 เฟส เมื่อโหลดเป็นตัวเหนี่ยวนำ (RL)

B. การนำกระแส 120 องศา

การควบคุมชนิดนี้ทรานซิสเตอร์แต่ละตัวจะมีช่วงการนำกระแส 120 องศา จะมีทรานซิสเตอร์เพียง 2 ตัวนำกระแสตลอด สัญญาณเกตแสดงในรูป 2.8 ลำดับการนำกระแสของทรานซิสเตอร์จะเป็น 61, 12, 23, 34, 45, 56, 61 ในแต่ละคาบจะแบ่งการทำงานออกเป็น 3 โหมด ในที่นี้จะคิดเฉพาะกรณีโหลดต่อวงจรแบบวาย



รูปที่ 2.8 สัญญาณขับนำและแรงดันด้านออกกรณีการนำกระแสในช่วง 120 องศา

โหมด 1 $(0 \le \omega t \le \pi/3)$ ทรานซิสเตอร์ Q_1 และ Q_6 นำกระแส

$$v_{an} = \frac{V_s}{2} \quad v_{bn} = -\frac{V_s}{2} \quad v_{cn} = 0$$

โหมด 2 $(\pi/3 \le \omega t \le 2\pi/3)$ ทรานซิสเตอร์ Q_1 และ Q_2 นำกระแส

$$v_{an} = \frac{V_s}{2} \quad v_{bn} = 0 \quad v_{cn} = -\frac{V_s}{2}$$

โหมด 3 $(2\pi/3 \le \omega t \le \pi)$ ทรานซิสเตอร์ Q_2 และ Q_3 นำกระแส

$$v_{an} = 0 \quad v_{bn} = \frac{V_s}{2} \quad v_{cn} = -\frac{V_s}{2}$$

แรงดันระหว่างสายกับนิวทรัลแสดงในรูปที่ 2.8 สามารถเขียนในรูปอนุกรมฟูเรียร์ได้ดังนี้

$$v_{an} = \sum_{n=1,3,5..}^{\infty} \frac{2V_s}{n\pi} \cos\frac{n\pi}{6} \sin n(\omega t + \frac{\pi}{6}) \tag{2.14}$$

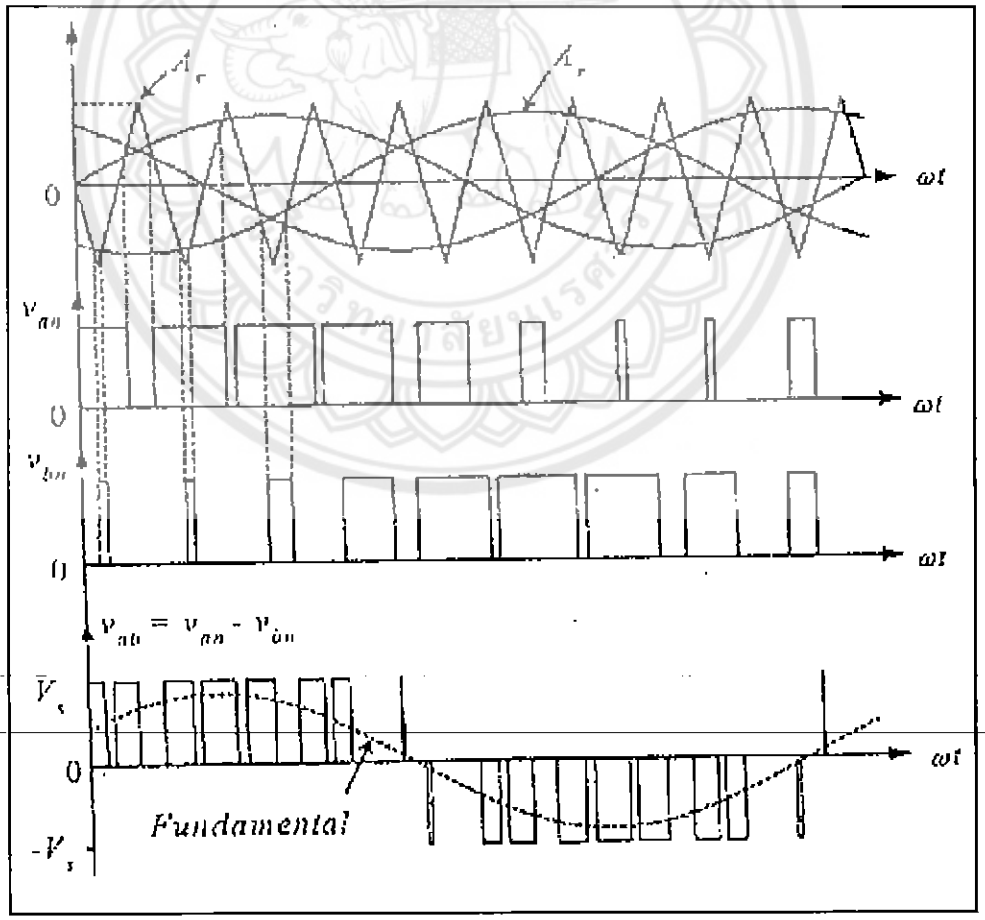
$$v_{bn} = \sum_{n=1,3,5..}^{\infty} \frac{2V_s}{n\pi} \cos\frac{n\pi}{6} \sin n(\omega t - \frac{\pi}{2}) \tag{2.15}$$

$$v_{cn} = \sum_{n=1,3,5..}^{\infty} \frac{2V_s}{n\pi} \cos\frac{n\pi}{6} \sin n(\omega t - \frac{7\pi}{6}) \tag{2.16}$$

แรงดันระหว่างเฟส a กับ b มีค่า $v_{ab} = \sqrt{3}v_{an}$ และมีมุมประวิงกับแรงดันเฟสอยู่ 30 องศาและมีการประวิงระหว่างมุมหยุดนำกระแสของทรานซิสเตอร์ Q_1 และมุมเริ่มนำกระแสของทรานซิสเตอร์ Q_4 อยู่ $\pi/6$ ซึ่งทำให้ไม่เกิดปัญหาการลัดวงจรของแหล่งจ่ายผ่านทรานซิสเตอร์ตัวบนและตัวล่าง ในทุกช่วงเวลาจะมีขั้วโหลด 2 ขั้วต่อกับแหล่งจ่ายโดยอีกขั้วจะเปิดวงจร แรงดันของขั้วที่เปิดวงจรจะขึ้นอยู่กับคุณลักษณะของโหลดซึ่งไม่สามารถทำนายได้ ดังนั้นวงจรอินเวอร์เตอร์ 3 เฟสที่ทรานซิสเตอร์มีช่วงการนำกระแส 120 องศา จึงมีการใช้ประโยชน์น้อยกว่าวงจรที่มีช่วงการนำกระแส 180 องศาที่สภาวะโหลดเดียวกัน

2.2.2 การควบคุมแรงดันอินเวอร์เตอร์ 3 เฟส

อินเวอร์เตอร์ 3 เฟส สามารถพิจารณาจากอินเวอร์เตอร์ 1 เฟส 3 ชุดโดยแรงดันด้านออกของแต่ละชุดจะเลื่อนเฟสไปชุดละ 120 องศา เช่นการสร้างสัญญาณเกตจากการมอดูเลตความกว้างพัลส์โดยใช้สัญญาณอ้างอิงรูปไซน์ดังแสดงในรูปที่ 2.9 สัญญาณอ้างอิงรูปไซน์ 3 ชุดซึ่งมีการเลื่อนเฟสไปชุดละ 120 องศา จะนำไปเปรียบเทียบกับสัญญาณพาหะรูป 3 เหลี่ยมได้สัญญาณเกตและแรงดันด้านออกแสดงในรูปที่ 2.9 โดยจะต้องหลีกเลี่ยงสภาวะการนำกระแสพร้อมกันของอุปกรณ์ที่อยู่ในกิ่งเดียวกัน



รูปที่ 2.9 การมอดูเลตความกว้างพัลส์โดยสัญญาณอ้างอิงรูปไซน์สำหรับอินเวอร์เตอร์ 3 เฟส

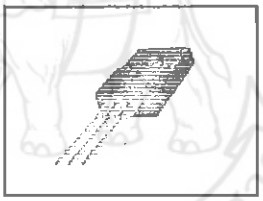
2.3 มอสเฟตกำลัง (Power MOSFET)

2.3.1 คุณสมบัติของเพาเวอร์มอสเฟต

เพาเวอร์มอสเฟตเป็นอุปกรณ์อิเล็กทรอนิกส์ที่ใช้แรงดันควบคุมและต้องการกระแสอินพุตเพียงเล็กน้อยซึ่งนิยมใช้มากในวงจรสวิตช์ชิ่งเพาเวอร์ซัพพลายในการควบคุมมอเตอร์โดยใช้อินเวอร์เตอร์ เป็นต้น เนื่องจากมีข้อดีคือ

- กำลังสูญเสียขณะสวิตช์ต่ำ
- ไม่มีช่วงแรงดันพังทลายที่ 2 (Second Breakdown)
- มีอัตราขยายสูงและวงจรขับสร้างได้ง่ายและราคาถูก
- มีความทนทานและเสถียรภาพของอุณหภูมิดี
- มีความจุหรือรับกระแสได้สูง
- สามารถนำมาต่อขนานได้ง่ายเนื่องจากสัมประสิทธิ์ความต้านทานเป็นบวก
- ใช้ในวงจรความถี่สูงได้ดี

แต่ข้อเสียของเพาเวอร์มอสเฟตก็มีเช่นเดียวกันคือแรงดันตกคร่อมขณะนำกระแสมีค่ามาก(ประมาณ 4.5โวลต์) ซึ่งเพาเวอร์ทรานซิสเตอร์จะมีเพียง 1 โวลต์



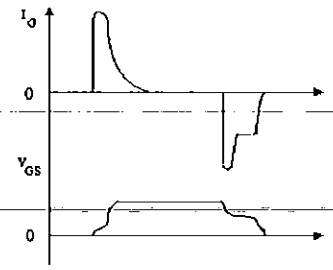
รูปที่ 2.10 ลักษณะภายนอกของเพาเวอร์มอสเฟต

2.3.2 การขับเพาเวอร์มอสเฟต

เนื่องจากอินเวอร์เตอร์เกือบทุกแบบ จะคงค่าแรงดันเอาต์พุตได้ด้วยการควบคุมช่วงเวลาการนำกระแสของทรานซิสเตอร์กำลังดังนั้นวงจรควบคุมการทำงานของอินเวอร์เตอร์โดยทั่วไปจึงมักนิยมใช้เทคนิคพัลส์วิดท์มอดูเลชั่น (Pulse Width Modulation) หรือ PWM ควบคุมช่วงเวลาการนำกระแสของทรานซิสเตอร์กำลังในอินเวอร์เตอร์ ซึ่งสามารถจะทำได้ในสองลักษณะการทำงานของวงจรควบคุม คือ ทำงานในโหมดควบคุมแรงดันและโหมดควบคุมกระแส

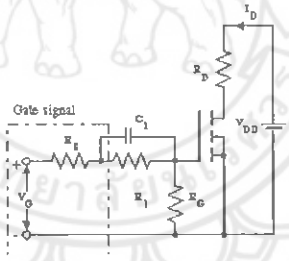
การขับเพาเวอร์มอสเฟตให้นำกระแสนั้นจะทำได้ก็ต่อเมื่อ แรงดันที่ตกคร่อมขาเกตและซอร์สมีค่าต่ำเท่ากับค่าแรงดันขีดเริ่ม (Threshold Voltage) ของมัน ทั้งนี้เนื่องมาจากลักษณะโครงสร้างภายในตัวเพาเวอร์มอสเฟตเหมือนมีตัวเก็บประจุ อาจอยู่รอบๆ ขาต่างๆ ของมัน ดังในรูปที่ 2.10 ตัวเก็บประจุเหล่านี้บังคับให้เพาเวอร์มอสเฟตต้องชาร์จประจุเข้าไปที่ตัวเก็บประจุเสียก่อนที่ตัวมันจะเริ่มนำกระแส

ในทางกลับกันในการหยุดนำกระแสของเพาเวอร์มอสเฟตต้องทำให้ตัวเก็บประจุคายประจุออกไปจนแรงดันตกคร่อมที่ขาเกต (V_{GS}) มีค่าลดลงต่ำกว่าแรงดันขีดเริ่มเพาเวอร์มอสเฟตจึงจะเริ่มหยุดนำกระแส ลักษณะของกระแสและแรงดันที่ขาเกต จึงมีลักษณะแสดงได้ดังรูปที่ 2.11

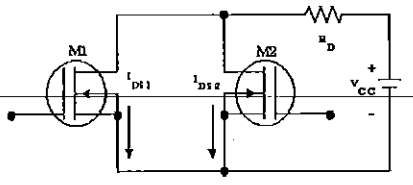


รูปที่ 2.11 ลักษณะกระแสและแรงดันที่ขาเกตขณะเพาเวอร์มอสเฟตถูกไบแอสให้นำกระแส

สำหรับวงจรในการขับเกตแสดงได้ดังรูปที่ 2.12 ในบางกรณีหากตัวเพาเวอร์มอสเฟตไม่สามารถทนพิกัดกระแสโหลด อาจทำการแก้ไขได้โดยการต่อขนานมอสเฟตหลายตัวเข้าด้วยกันโดยเพาเวอร์มอสเฟตที่นำมาต่อนั้น ควรมีคุณสมบัติเหมาะสมกันดังรูปที่ 2.13 แสดงการต่อขนานมอสเฟต



รูปที่ 2.12 วงจรขับเกต

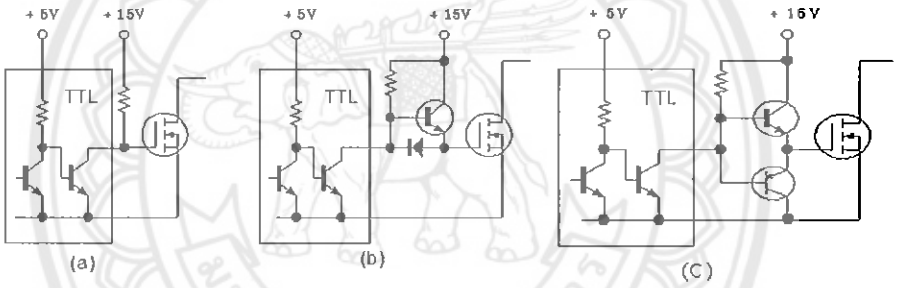


รูปที่ 2.13 การต่อขนานเพาเวอร์มอสเฟต

2.3.3 การขับมอสเฟตกำลังด้วยไอซี TTL

การขับมอสเฟตกำลังจากไอซี TTL โดยตรงนั้นเป็นไปได้แต่ไอซี TTL มีขีดจำกัดในการจ่ายและรับกระแสของมันที่เอาต์พุตซึ่งจะมีผลโดยตรงต่อความเร็วในการเปลี่ยนสถานะของมอสเฟตกำลังและทำให้เกิดกำลังสูญเสียได้สูงดังนั้นการต่อวงจรขับมอสเฟตกำลังด้วยไอซี TTLจึงจำเป็นต้องเพิ่มตัวอุปกรณ์อื่นๆ เพื่อช่วยให้เกิดการเปลี่ยนสถานะของมอสเฟตกำลังเป็นไปได้อย่างรวดเร็ว

การต่อพูลอัพรีซิสเตอร์เข้าช่วย จึงทำให้มีแรงดันสูงพอที่จะขับมอสเฟตกำลังให้ทำงาน และหยุดการนำกระแสของมอสเฟตกำลังเป็นไปได้รวดเร็วขึ้น อย่างไรก็ตาม ความเร็วขณะเริ่มนำกระแสก็ยังคงมีค่าที่จำกัดอยู่เนื่องจากกระแสที่ป้อนถูกจำกัดด้วยตัวต้านทานพูลอัพรีซิสเตอร์นั่นเอง การต่อทรานซิสเตอร์เพิ่มเข้ามาดังในรูปที่ 2.14 (a) ทรานซิสเตอร์จะช่วยจ่ายกระแสได้มากขึ้น ทำให้ความเร็วขณะเริ่มนำกระแสของมอสเฟตกำลังดีขึ้นและลดกำลังงานสูญเสียในตัวไอซี TTL เพื่อให้การคายประจุที่ขาเกตเป็นไปอย่างรวดเร็วการเพิ่มทรานซิสเตอร์ในวงจรอีกหนึ่งตัว ดังรูป 2.14 (b) ก็จะทำให้ความเร็วในขณะเริ่มหยุดนำกระแสเป็นไปได้อย่างรวดเร็วมากขึ้น



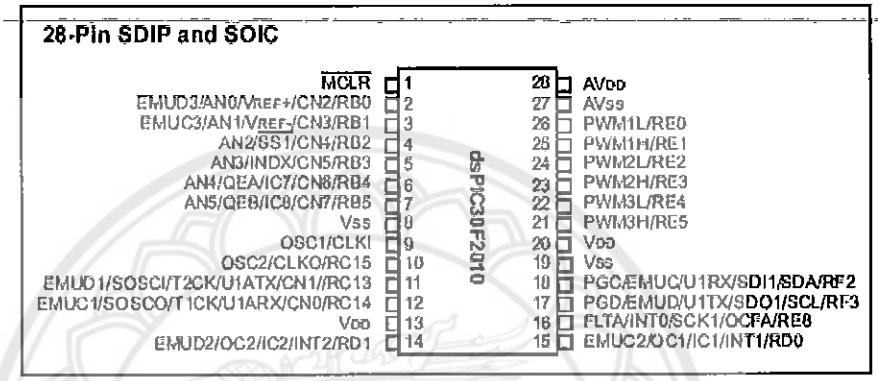
รูปที่ 2.14 (a) การขับมอสเฟตกำลังให้นำกระแสด้วยไอซี TTL และพูล-อัพรีซีสเตอร์
(b) การต่อทรานซีสเตอร์เพิ่มเข้ามา
(c) การต่อทรานซิสเตอร์เพิ่มเข้ามาอีกหนึ่งตัว

ในวงจรรูปที่ 2.14 (b) เพื่อเพิ่มความเร็วในขณะเริ่มหยุดนำกระแสด้วยทรานซิสเตอร์ที่ใช้สามารถใช้ทรานซิสเตอร์กำลังต่ำ เช่น เบอร์ 2N2222 และ เบอร์ 2N2907 ก็ทำให้วงจรขับสามารถจ่ายและรับกระแสได้สูงถึง 800 มิลลิแอมแปร์ ซึ่งก็เพียงพอแล้ว

บทที่ 3

ควบคุมมอเตอร์ด้วย ดีเอสพิค ไมโครคอนโทรลเลอร์

3.1 ข้อมูลเบื้องต้นของ dsPIC30F2010



รูปที่ 3.1 การจัดขาใช้งานไมโครคอนโทรลเลอร์ dsPIC30F2010

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
V_{DD}	13,20	อินพุต	-	-ขาต่อไฟเลี้ยงบวก ตั้ง 2.5 ถึง 5.5v
V_{SS}	8,19	อินพุต	-	-ขาต่อกราวด์
AV_{DD}	28	อินพุต	-	-ขาต่อไฟเลี้ยงบวก 5v ให้แก่โมดูล ADC ภายใน dsPIC
AV_{SS}	27	อินพุต	-	-ขาต่อไฟเลี้ยงลบ 5v ให้แก่โมดูล ADC ภายใน dsPIC

ขาพอร์ต B เป็นขาพอร์ต 2 ทิศทาง

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
EMUD3	2	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาข้อมูลสำหรับติดต่อกับ ICD กลุ่มที่ 4
AN0		อินพุต	อะนาลอก	-อินพุตอะนาลอกช่อง 0
V_{REF-}		อินพุต	อะนาลอก	-อินพุตแรงดันอ้างอิงขั้วบวก
CN2		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทาง
				ดิจิตอลช่อง2สามารถโปรแกรมให้มีการพูลอัปได้
RB0		อินพุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RB0
EMUC3	3	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาสัญญาณนาฬิกาติดต่อกับ ICD กลุ่มที่ 4
AN1		อินพุต	อะนาลอก	-อินพุตอะนาลอกช่อง 1
V_{REF-}		อินพุต	อะนาลอก	-อินพุตแรงดันอ้างอิงขั้วลบ

CN3		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง 3 สามารถโปรแกรมให้มีการพูลอัประดับต่ำได้
RB1		อินพุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RB1
AN2	4	อินพุต	ชมิตต์ทริกเกอร์	-อินพุตอะนาลอกช่อง2
SS1		อินพุต	ชมิตต์ทริกเกอร์	-อินพุต Slave Select เมื่อโมดูล SPI ทำงานในโหมดสเลฟ
CN4		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง4 สามารถโปรแกรมให้มีการพูลอัปได้
RB2		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RB2
AN3	5	อินพุต	อะนาลอก	-อินพุตอะนาลอกช่อง3
INDX		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตสัญญาณพัลส์ของโมดูลเข้ารหัสแบบควอดราเจอร์
CN5		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง5 สามารถโปรแกรมให้มีการพูลอัปได้
RB3		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RB3
AN4	6	อินพุต	อะนาลอก	-อินพุตอะนาลอกช่อง 4
QEA		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตเฟส A ของโมดูลเข้ารหัสแบบควอดราเจอร์
IC7		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตของโมดูลตรวจจับสัญญาณช่อง 7
CN6		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง6 สามารถโปรแกรมให้มีการพูลอัปได้
RB4		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RB4
AN5	7	อินพุต	อะนาลอก	-อินพุตอะนาลอกช่อง 5
QEB		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตเฟสBของโมดูลเข้ารหัสแบบควอดราเจอร์
IC8		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตของโมดูลตรวจจับสัญญาณช่อง 8
CN7		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง7 สามารถโปรแกรมให้มีการพูลอัปได้
RB5		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RB5

ขาพอร์ต C เป็นขาพอร์ต 2 ทิศทาง

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
EMUD1	11	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาข้อมูลสำหรับติดต่อกับ ICDกลุ่มที่2
SOSCI		อินพุต	ชมิตต์ทริกเกอร์/ซีมอส	-อินพุตสัญญาณนาฬิกา 32 kHz จากภายนอกเมื่อทำงานกับไทม์เมอร์ 1
				อินพุตเป็นชมิตต์ทริกเกอร์เมื่อทำงานในโหมด RC
T2CK		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตสัญญาณนาฬิกาจากภายนอกของไทเมอร์2
CN1		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง 1 สามารถโปรแกรมให้มีการพูลอัประดับต่ำได้

U1ATX		อินพุต	-	-ขาส่งข้อมูลอนุกรมของส่วนสื่อสารข้อมูลอนุกรม UART
RC13		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RC13
EMUC1	12	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาสัญญาณนาฬิกาสำหรับติดต่อกับ ICD กลุ่มที่ 2
SOSCO		อินพุต	-	-อินพุตสัญญาณนาฬิกา 32 kHz จากภายนอกเมื่อทำงานกับไทม์เมอร์ 1
T1CK		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตสัญญาณนาฬิกาจากภายนอกของไทเมอร์1
CN0		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตตรวจจับการเปลี่ยนแปลงสัญญาณทางดิจิตอลช่อง 0 สามารถโปรแกรมให้มีการพูลอัพระดับต่ำได้
U1ARX		อินพุต	ชมิตต์ทริกเกอร์	-ขาส่งข้อมูลอนุกรมของส่วนสื่อสารข้อมูลอนุกรม UART
RC14		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RC14
OSC2/CLKO	10	อินพุต/เอาต์พุต	-	-ใช้ต่อกับคริสตอลหรือเวรามิกเรโซเตอร์ เมื่อทำงานในโหมดคริสตอลและเป็นเอาต์พุตสัญญาณนาฬิกาด้วย
RC15		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RC15

ขาพอร์ต D เป็นขาพอร์ต 2 ทิศทาง

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
EMUD2	15	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาข้อมูลสำหรับติดต่อกับ ICDกลุ่มที่3
OC1		เอาต์พุต	-	-เอาต์พุตของโมดูลเปรียบเทียบข้อมูลช่อง 1
IC1		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตของโมดูลตรวจจับสัญญาณช่อง 1
INT1		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตรับสัญญาณอินเตอร์รัพจากภายนอกช่อง 1
RD0		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RD0
EMUC2	14	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาสัญญาณนาฬิกาสำหรับติดต่อกับ ICD กลุ่มที่ 3
OC2		เอาต์พุต	-	-เอาต์พุตของโมดูลเปรียบเทียบข้อมูลช่อง 2
IC2		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตของโมดูลตรวจจับสัญญาณช่อง 2
INT2		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตรับสัญญาณอินเตอร์รัพจากภายนอกช่อง 2
RD1		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RD1

ขาพอร์ต E เป็นขาพอร์ตควบคุมมอเตอร์และใช้งานทั่วไป

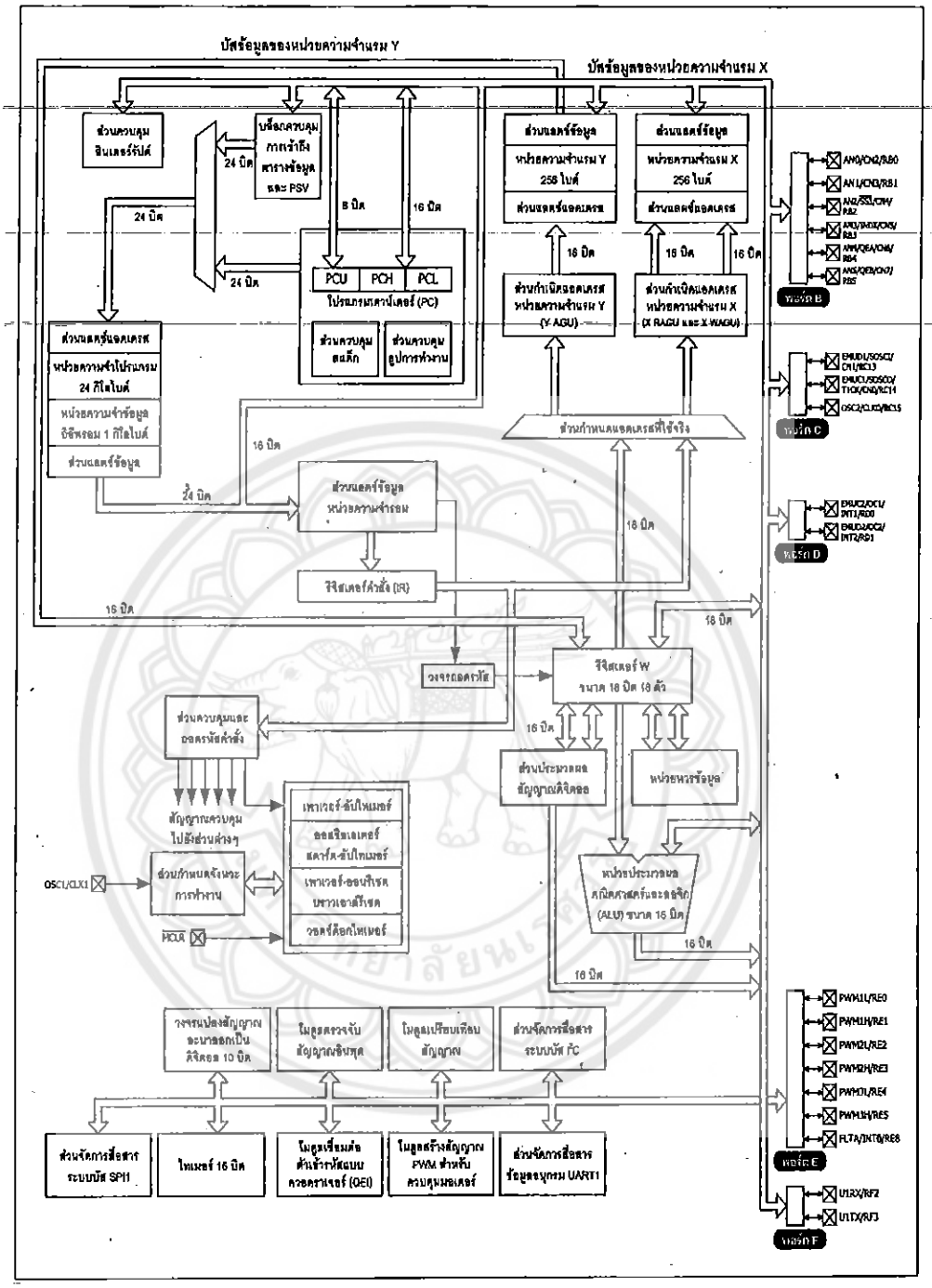
ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
PWM1L	26	เอาต์พุต	-	-เอาต์พุตสัญญาณ PWM ด้านต่ำสำหรับขับมอเตอร์ ช่อง 1
RE0		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RE0
PWM1H	25	เอาต์พุต	-	-เอาต์พุตสัญญาณ PWM ด้านสูงสำหรับขับมอเตอร์ ช่อง 1
RE1		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RE1
PWM2L	24	เอาต์พุต	-	-เอาต์พุตสัญญาณ PWM ด้านต่ำสำหรับขับ

				มอเตอร์ ช่อง 2
RE2		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RE2
PWM2H	23	เอาต์พุต	-	-เอาต์พุตสัญญาณ PWM ด้านสูงสำหรับขับ
				มอเตอร์ ช่อง 2
RE3		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RE3
PWM3L	22	เอาต์พุต	-	-เอาต์พุตสัญญาณ PWM ด้านต่ำสำหรับขับ
				มอเตอร์ ช่อง 3
RE4		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RE4
PWM3H	21	เอาต์พุต	-	-เอาต์พุตสัญญาณ PWM ด้านสูงสำหรับขับ มอเตอร์ ช่อง 3
RE5		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	- ขาพอร์ต RE5
FLTA	16	อินพุต	ชมิตต์ทริกเกอร์	-อินพุตรับสัญญาณผิดพลาดช่อง A ของ PWM
INT0		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตรับสัญญาณอินเตอร์รัพจากภายนอก ช่อง 0
OCFA		อินพุต	ชมิตต์ทริกเกอร์	-อินพุตรับสัญญาณผิดพลาดช่อง A ของโมดูลเปรียบเทียบ
SCK1		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาสัญญาณนาฬิกาอนุกรมของโมดูล
RE8		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RE8

ขาพอร์ต F เป็นขาพอร์ตสื่อสารข้อมูลอนุกรมและใช้งานทั่วไป

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียดการทำงาน
PGC	18	อินพุต	ชมิตต์ทริกเกอร์	-ขารับสัญญาณนาฬิกาสำหรับโปรแกรม
EMUC		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาสัญญาณนาฬิกาติดต่อกับ ICD กลุ่มที่ 1
U1RX		อินพุต	ชมิตต์ทริกเกอร์	-ขารับข้อมูลอนุกรมของส่วนสื่อสารข้อมูลอนุกรม UART
SDI1		อินพุต	ชมิตต์ทริกเกอร์	-ขารับข้อมูลอนุกรมของโมดูล SPI
SDA		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาข้อมูลอนุกรมของโมดูลระบบบัส I^2C
RF2		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RF2
PGD	17	อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาข้อมูลอนุกรมสำหรับโปรแกรม
EMUD		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาข้อมูลสำหรับติดต่อกับ ICD กลุ่มที่ 1
U1TX		เอาต์พุต	-	-ขาส่งข้อมูลอนุกรมของส่วนสื่อสารข้อมูลอนุกรม UART
SDO1		เอาต์พุต	-	-ขาส่งข้อมูลอนุกรมของโมดูล SPI
SCL		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาสัญญาณนาฬิกาอนุกรมของโมดูลระบบบัส I^2C
RF3		อินพุต/เอาต์พุต	ชมิตต์ทริกเกอร์	-ขาพอร์ต RF3
OSC1	9	อินพุต	ชมิตต์ทริกเกอร์	-ขาต่อคริสตอลภายนอก
CLKI		อินพุต	ชมิตต์ทริกเกอร์	-ขารับสัญญาณนาฬิกาจากภายนอก

ตารางที่ 3.1 รายละเอียดโดยสรุปของขาข่อใช้งานของ dsPIC30F2010



รูปที่ 3.2 ไดอะแกรมการทำงานและส่วนประกอบทั้งหมดของ dsPIC30F2010

dsPIC30F2010 เป็นไมโครคอนโทรลเลอร์ 16 บิต ออกแบบมาเป็นพิเศษเพื่องานประมวลผลสัญญาณดิจิตอลสำหรับสร้างระบบควบคุมอัตโนมัติที่มีความสามารถสูง

3.1.1 คุณสมบัติของซีพียู

-มี 84 คำสั่งมาตรฐาน สามารถรับรองรูปแบบการอ้างแอดเดรสได้อย่างอิสระ

-ชุดคำสั่งมีขนาด 24 บิต สามารถประมวลผลข้อมูลได้ 16 บิต

-มีหน่วยความจำแบบแฟลช ความจุ 12 กิโลไบต์ ลบและเขียนใหม่ได้น้อยกว่า 100,000 ครั้ง สามารถป้องกันการอ่านได้

-สามารถโปแกรมหน่วยความจำโปรแกรมได้ด้วยตนเองโดยใช้กระบวนการทางซอฟต์แวร์

-มีหน่วยความจำข้อมูลอีอีพรอม 1 กิโลไบต์ ลบและเขียนใหม่ได้น้อยกว่า 1, 000,000 ครั้ง

-มีหน่วยความจำข้อมูลแรม 512 ไบต์

-รีจิสเตอร์ W จัดในรูปอะเรย์ มีขนาด 16 บิต จำนวน 16 ตัว

-ความเร็วในการทำงานสูงถึง 30 ล้านคำสั่งต่อวินาที

-ความถี่สัญญาณจากภายนอก ตั้งแต่ย่านไฟตรงจนถึง 40 MHz

-ความถี่สัญญาณนาฬิกาในกรณีใช้งานร่วมกับวงจรเฟลล็อกลูปภายใน ตั้งแต่ 4 MHz ถึง 10 MHz เลือกได้ 3 ระดับ 4, 8 หรือ 16 เท่า

-รองรับแหล่งกำเนิดสัญญาณอินเตอร์รัพต์ได้สูงสุด 62 แหล่งรวมทั้งการอินเตอร์รัพต์จากภายนอก 3 แหล่ง

-สามารถกำหนดระดับความสำคัญในการตอบสนองอินเตอร์รัพต์ได้ 8 ระดับ

-มีอินเตอร์รัพต์เวกเตอร์ 48 ตำแหน่ง

-มีวงจรปรับแรงดันไฟเลี้ยงต่ำกว่ากำหนดแบบโปรแกรมได้

-มีเพาเวอร์-ออนรีเซต, เพาเวอร์อัปไทเมอร์ และออสซิลเลเตอร์–อัปไทเมอร์

-มีวอตช์ด็อกไทม์ไทเมอร์แบบโปรแกรมได้

-มีวงจรตรวจสอบการทำงานของวงจรกำเนิดสัญญาณนาฬิกาหากผิดพลาดจะเข้าสู่โหมดสัญญาณนาฬิกา RC พลังงานต่ำทันที

-รองรับการโปรแกรมในวงจรแบบอนุกรม(ICSP: In-Circuit Serial Programming)

-สามารถเลือกโหมดการใช้พลังงานได้

-ย่านเลี้ยงไฟ 2.5 ถึง 5.5 V กระแสไฟฟ้า 2.6 ถึง 44 mA ที่ไฟเลี้ยง +5V ขึ้นอยู่กับการกำหนดความเร็วในการทำงาน

3.1.2 คุณสมบัติด้านการประมวลสัญญาณดิจิตอล

-มีแอกคิวมูเลเตอร์ขนาด 40 บิต 2 ตัว รองรับการประมวลผลทางคณิตศาสตร์ได้เป็นอย่างดี

-มีหน่วยประมวลด้านการคูณและการหารเลข 17 บิตในรูปของฮาร์ดแวร์

-ทำการคูณเลข 16 บิตได้ภายในสัญญาณนาฬิกาเพียง 1 ไซเกิล

-มีตัวเลื่อนข้อมูลบาร์เรล 40 สเตจ

-มีวงจรเฟตข้อมูลคู่

3.1.3 คุณสมบัติของโมดูลฟังก์ชั่นพิเศษ

-สามารถจ่ายกระแสออกทางขาพอร์ตได้ 25 mA ทั้งแบบกระแสซิงก์และกระแสซอร์ส

-ไทเมอร์/เคาร์เตอร์ 16 บิต 3 ตัว สามารถต่อใช้งานร่วมกันเป็นไทเมอร์/เคาร์เตอร์ 32 บิตได้

-มีโมดูตรวจจับสัญญาณดิจิตอลขนาด 16 บิต 4 ชุด

-มีโมดูเปรียบเทียบข้อมูลและกำเนิดสัญญาณ PWM ความละเอียด 16 บิต 2 ชุด

-มีส่วนเชื่อมต่ออุปกรณ์อนุกรมแบบ SPI

-มีส่วนเชื่อมต่ออุปกรณ์ผ่านระบบบัส I^2Cทั้งแบบ 7 และ10 บิตกำหนดเป็นมาสเตอร์หรือสเลฟได้

-มีโมดูลสื่อสารข้อมูลอนุกรม UART พร้อมบัฟเฟอร์แบบ FIFO

-มีโมดูลสร้างสัญญาณ PWM สำหรับควบคุมมอเตอร์ 6 ช่อง

-มีโมดูลเชื่อมต่อตัวเข้ารหัสแบบควอดราเจอร์

-มีวงจรแปลสัญญาณอะนาลอกเป็นดิจิตอล ความละเอียด 10 บิต 6 ช่อง

3.1.4 สถาปัตยกรรมโดยสรุปของ dsPIC30F2010

หน่วยประมวลผลกลาง

รีจิสเตอร์หลักที่ใช้ในการทำงานคือ รีจิสเตอร์ W (Working register) โดยรีจิสเตอร์ W ได้รับการจัดให้โครงสร้างเป็นอะเรย์ ขนาด 16 บิต จึงทำให้สามารถรองรับทั้งข้อมูล, ค่าแอดเดรส หรือค่าของรีจิสเตอร์ใดๆ ที่ต้องนำมาประมวลผล

ใน dsPIC มีรีจิสเตอร์ W ให้ใช้งานถึง 16 ตัว ส่วนใหญ่ใช้ในการประมวลผลหลัก ส่วนอีกตัวหนึ่งคือ รีจิสเตอร์ W 15 จะใช้ทำงานร่วมกับตัวชี้สแต็กในการทำงานของโปรแกรมย่อยและบริการอินเตอร์รัพต์

ด้านการตอบสนองอินเตอร์รัพต์นั้น dsPIC30F2010 มาการจัดสรรพื้นที่เก็บค่าอินเตอร์รัพต์เวกเตอร์ไว้มากถึง 54 ตำแหน่ง และยังสามารถกำหนดระดับความสำคัญได้อีก 8 ระดับด้วย

หน่วยความจำ

dsPIC30F2010 มีหน่วยความจำโปรแกรม 4 กิโลเวิร์ด แอดเดรสอยู่ในช่วง 0x000100 ถึง 0x001FFE

-หน่วยความจำข้อมูลแรมของ dsPIC30F2010 ได้จัดสรรเป็น 2 ส่วนคือ หน่วยความจำข้อมูลแรม X และ Y แต่ละส่วนมีขนาด 16 บิต ความจุ 256 ไบต์ รวมเป็น 512 ไบต์ โดยแต่ละส่วนจะมีตัวกำหนดแอดเดรสแยกออกจากกันเรียกว่า AGU (Address Generation Unit)

-หน่วยความจำข้อมูลอีอีพรอม dsPIC30F2010 จัดสรรไว้ที่แอดเดรส 0x7FFC00 ถึง 0x7FFFFE มีความจุ 1 กิโลไบต์

ส่วนประมวลผลสัญญาณดิจิตอล (DSP Engine)

โดยส่วนประมวลสัญญาณดิจิตอลมีหน่วยการจักการคูณเลขขนาด 17x17 บิตความเร็วสูง, หน่วยประมวลผลทางคณิตศาสตร์และลอจิกหรือ ALU ขนาด 40 บิต แอกคิวมูเลย์เตอร์ขนาด 40 บิต 2 ตัว และตัวเลื่อนข้อมูล 2 ทิศทางแบบบาเรล ขนาด 40 บิต จึงทำให้สามารถจัดการข้อมูลขนาด 16 บิต ได้เสร็จภายในสัญญาณเพียงไซเกิลเดียว

โมดูลฟังก์ชั่นพิเศษ

-โมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล ความละเอียด 10 บิต

-โมดูลเชื่อมต่ออุปกรณ์อนุกรมหรือ SPI

-โมดูลสื่อสารระบบบัส I^2C

-โมดูลสื่อสารข้อมูลข้อมูลผ่านพอร์ตอนุกรมหรือ UART

-ไทเมอร์ขนาด 16 บิตถึง 3 ตัว

-โมดูลสร้างสัญญาณ PWM เพื่อการควบคุมมอเตอร์

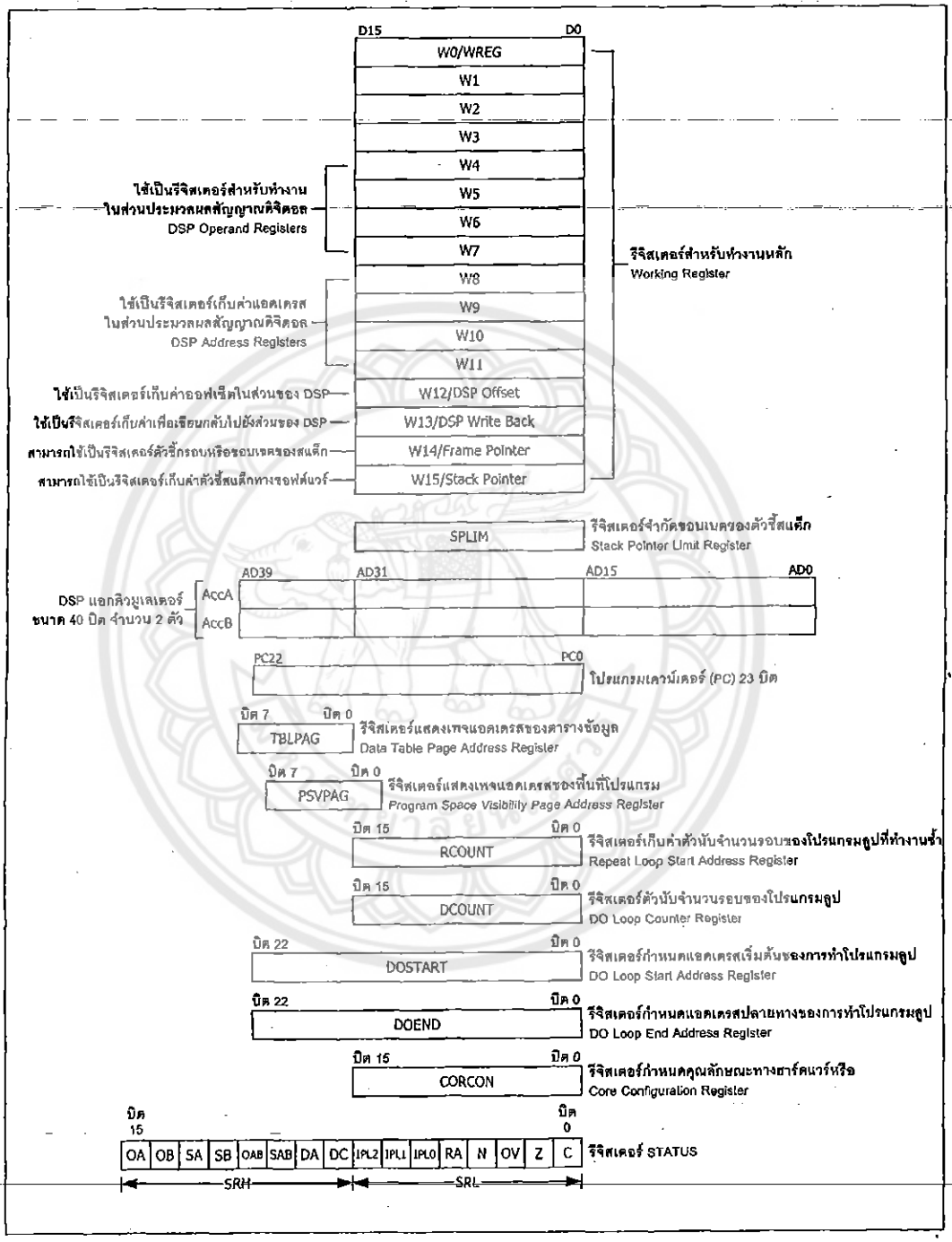
-โมดูลเข้ารหัสแบบควอดราเจอร์โดยสามารถใช้งานร่วมกันเพื่อสร้างระบบควบคุมมอเตอร์แบบปิดประสิทธิภาพสูง

พอร์ตอินพุตเอาต์พุต

dsPIC30F2010 มีพอร์ตให้ใช้งานถึง 5 พอร์ต รวม 20 ขา ดังนี้

- *พอร์ต B* มีขา 6 คือ RB0-RB5 โดยทุกขาสามารถกำหนดให้เป็นพอร์ตอินพุตหรือเอาท์พุตได้ และยังสามารถขับกระแสทั้งแบบซิงก์และซอร์สได้สูงถึง 25 mA
- *พอร์ต C* มี 3 ขา คือ RC13-RC15
- *พอร์ต D* มี 2 ขา คือ RD0 และ RDI
- *พอร์ต E* มี 7 ขา คือ RE0-RE และ RE8
- *พอร์ต F* มี 2 ขา คือ RF2 และ RF3

3.1.5 โครงสร้างทางโปรแกรมที่ควรทราบ



รูปที่ 3.3 โครงสร้างทางโปรแกรมหรือ Programmer's model ของไมโครคอนโทรลเลอร์ dsPIC

รีจิสเตอร์ W ขนาด 16 บิตมากถึง 16 ตัว โดยที่ใช้งานเป็นหลักคือ W0 ส่วนตัวที่ถูกนำไปใช้ในส่วนประมวลผลสัญญาณดิจิตอลมี 10 ตัวคือ W4-W13 ส่วน W14 ถูกนำไปใช้ร่วมเป็นตัวชี้เฟรม และ W15 ถูกนำไปใช้ร่วมในการชี้ตัวสแต็ก

แอกคิวมูเลเตอร์ 40 บิต จำนวน 2 ตัว ใช้ในงานประมวลผลดิจิตอลเป็นหลัก

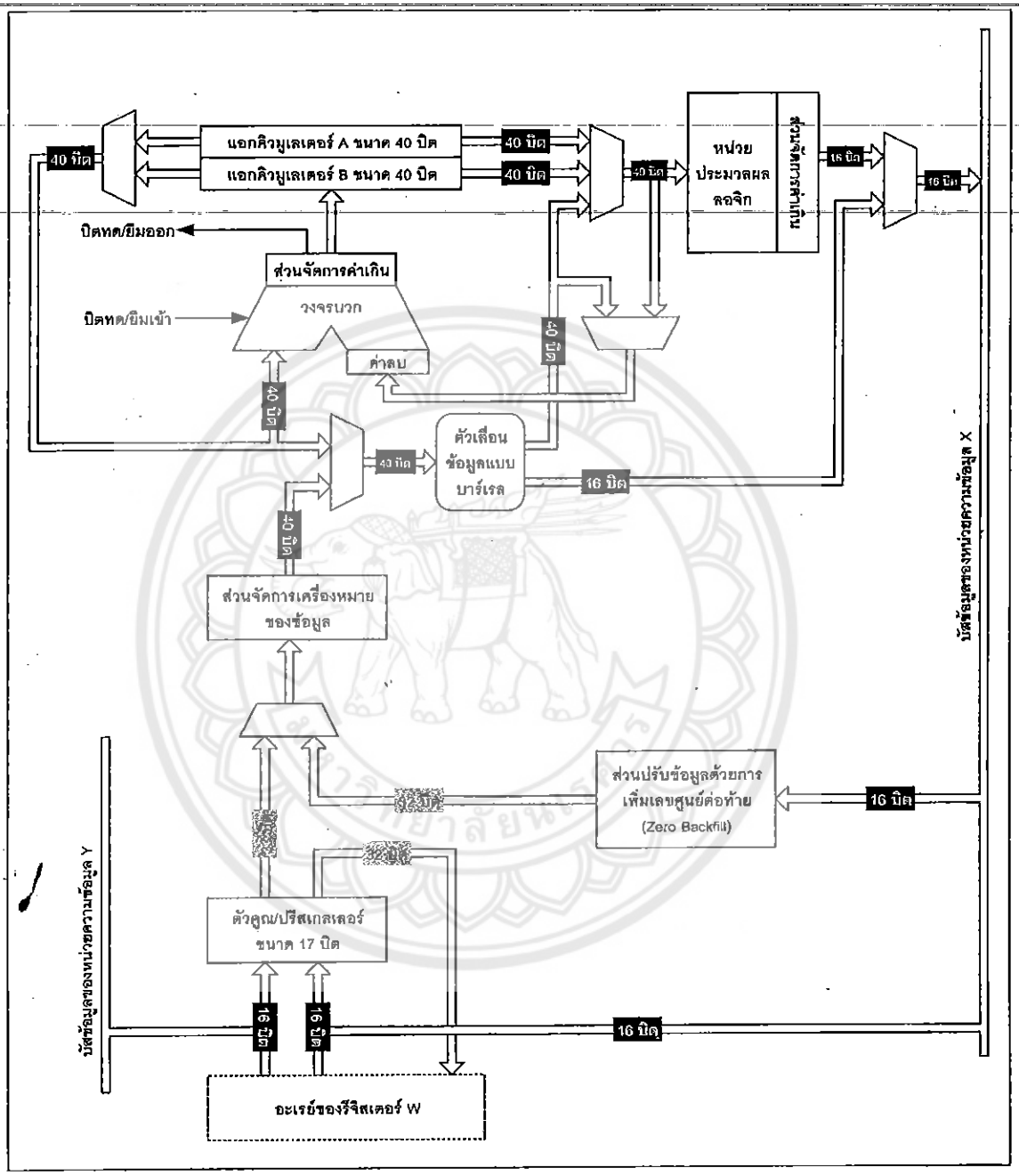
โปรแกรมเคาน์เตอร์ ขนาด 24 บิตนำมาใช้ในการแจ้งแอดเดรส 23 บิตโดยไม่สนใจบิต MSB และบิต LSB ต้องเป็น "0"

รีจิสเตอร์หลัก ประกอบด้วย

1. STATUS ซึ่งใช้แสดงสถานะทำงาน ขนาด 16 บิต
2. CORCON ใช้ควบคุมการทำงานของหน่วยประมวลผลกลาง มีขนาด 16 บิต
3. TBLPAG เป็นรีจิสเตอร์กำหนดเพจตารางของข้อมูลในหน่วยความจำโปรแกรมมีขนาด 8 บิต
4. PSVPAG เป็นรีจิสเตอร์แสดงเพจแอดเดรสของพื้นที่โปรแกรม มีขนาด 8 บิต
5. RCOUNT เป็นรีจิสเตอร์เก็บค่าตัวนับจำนวนรอบของลูปที่ทำซ้ำ
6. DCOUNT เป็นรีจิสเตอร์เก็บค่าตัวนับจำนวนรอบของลูปที่ทำงาน
7. DOSTART เป็นรีจิสเตอร์กำหนดแอดเดรสเริ่มต้นทำงานของโปรแกรมลูป

สำหรับรีจิสเตอร์ DCOUNT, DOSTART และ DOEND เป็นรีจิสเตอร์เงา (shadow register) หมายความว่า เป็นรีจิสเตอร์ที่ถูกสร้างขึ้นชั่วคราวเพื่อเก็บค่าก่อนที่จะมีการถ่ายถอดไปทำงาน จึงไม่สามารถเข้าถึงรีจิสเตอร์เหล่านี้ได้โดยตรง

3.1.6 หน่วยประมวลผลสัญญาณดิจิตอล (DSP Engine)



รูปที่ 3.4 ไดอะแกรมการทำงานโดยสรุปของหน่วยประมวลผลสัญญาณดิจิตอลภายใน dsPIC

องค์ประกอบสำคัญของหน่วยประมวลผลสัญญาณดิจิตอลใน dsPIC ได้แก่

1. มัลติพลายเออร์หรือตัวคูณ 17x17 บิตความเร็วสูง
2. ตัวเลื่อนข้อมูลแบบบาร์เรล ขนาด 40บิต
3. แอกคิวมูเลเตอร์ขนาด 40บิต ซึ่งมีด้วยกัน 2 ตัว คือ AccA และ AccB
4. ส่วนประมวลผลลอจิกที่สามารถเลือกโหมดการทำงานได้
5. ส่วนประมวลผลลอจิกในค่าอิ่มตัวที่สามารถเลือกโหมดการทำงานได้

ข้อมูลที่นำมาประมวลผลในหน่วยประมวลผลสัญญาณดิจิตอล (DSP) นี้มาได้จาก 2 แหล่งคือ

1. เข้ามาโดยตรงจากรีจิสเตอร์ W4 ถึง W7
2. จากบัสข้อมูลของหน่วยความจำข้อมูล X

ส่วนข้อมูลเอาต์พุตที่ได้จากหน่วย DSP นี้จะถูกส่งไปยัง 2 แหล่ง

1. ส่งไปยังแอกคิวมูเลเตอร์
2. หน่วยความจำข้อมูล X

3.1.7 การรองรับการหารเลขใน dsPIC

สำหรับ dsPIC แล้วสามารถรองรับการหารเลขได้เต็มรูปแบบโดยแบ่งเป็น 5 รูปแบบคือ

1. การหารเลขเศษส่วนหรือทศนิยม 16 บิต แบบคิดเครื่องหมาย โดยใช้คำสั่ง DIVF

2. การหารเลขตัวตั้ง 32 บิตด้วยตัวหาร 16 บิต แบบคิดเครื่องหมาย โดยใช้คำสั่ง DIV.SD

3. การหารเลขตัวตั้ง 32 บิตด้วยตัวหาร 16 บิต แบบไม่คิดเครื่องหมาย โดยใช้คำสั่งDIV.UD

4. การหารเลขจำนวนเต็ม 16 บิตแบบคิดเครื่องหมาย โดยใช้คำสั่ง DIV.SW

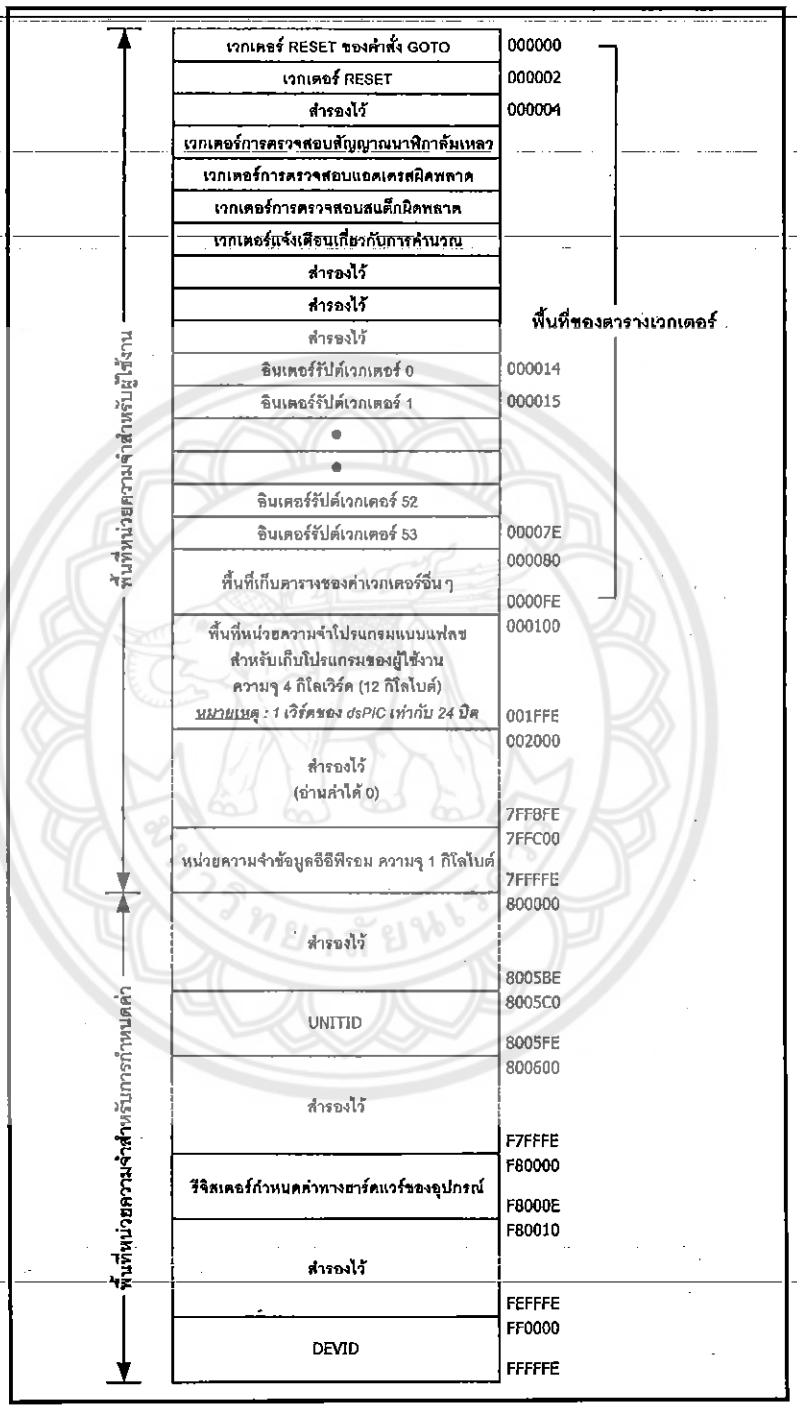
5. การหารเลขจำนวนเต็ม 16 บิตแบบไม่คิดเครื่องหมาย โดยใช้คำสั่ง DIV.UW

ผลหารของทุกคำสั่งหารเลขจะเก็บไว้ในรีจิตเตอร์ W0 ส่วนเศษของการหารเก็บไว้ในรีจิตเตอร์ W1 ในขณะที่ค่าของตัวหาร 16 บิตสามารถเก็บไว้ในรีจิตเตอร์ W ตัวใดก็ได้ เช่นเดียวกับตัวตั้งแต่ถ้าหากเป็นตัวตั้ง 32 บิตจะต้องนำค่าไปเก็บไว้ในรีจิตเตอร์ W จำนวน 2 ตัวที่อยู่ติดกันเช่น W2 กับ W3 โดย W3 เก็บค่าตัวตั้ง 16 บิตบนและ W2 เก็บค่าตัวตั้ง 16 บิตล่าง

3.1.8 การทำงานในโปรแกรมลูป

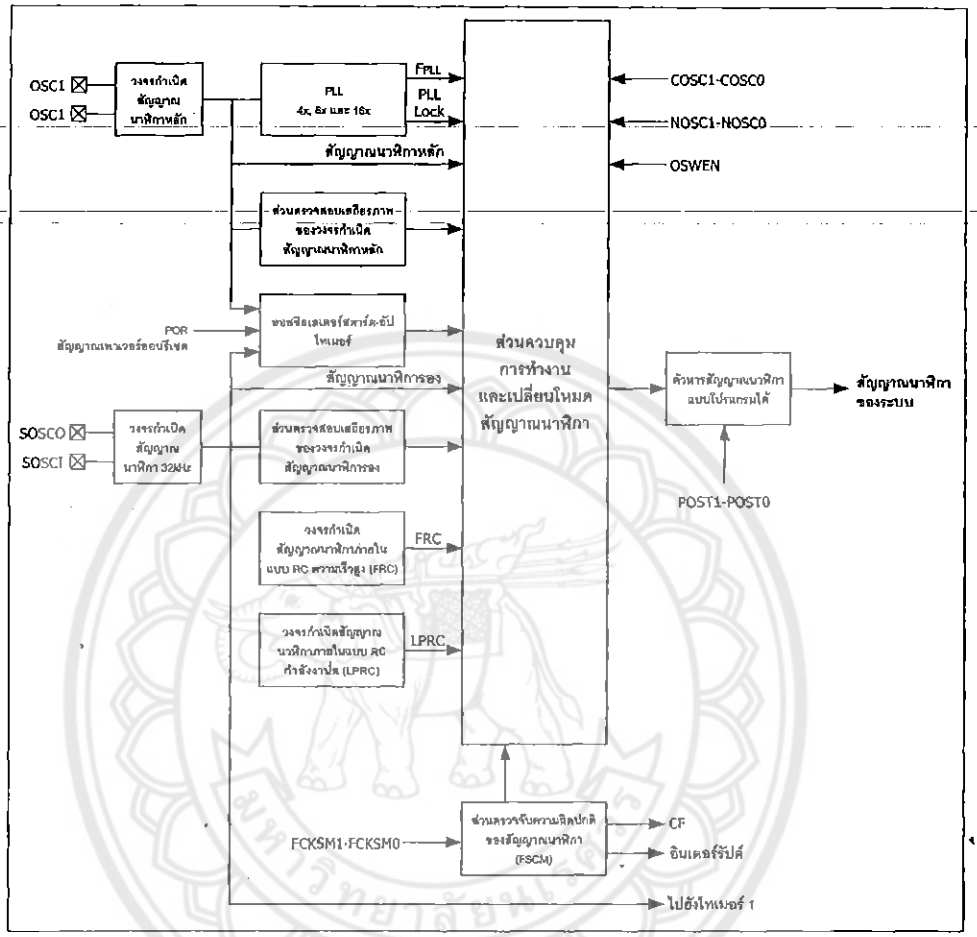
ใน dsPIC มีคำสั่งพิเศษเพื่อเข้ามาช่วยจัดการ 2 คำสั่งหลักคือ DO และ REPEATรวมทั้งมีรีจิสเตอร์เข้ามาช่วยในการเก็บค่าหลายตัว การวนทำงานในโปรแกรมลูปเป็นเรื่องปกติในการทำงานของคอนโทรลเลอร์ แต่การวนทำงานในโปรแกรมลูปปกติ ผู้พัฒนาโปรแกรมต้องสร้างโปรแกรมย่อยขึ้นมาเอง เพื่อกำหนดรอบการทำงาน ทำให้ต้องใช้พื้นที่หน่วยความจำโปรแกรมตลอดเวลาในการประมวลผลไปไม่น้อย ทว่าใน dsPIC ด้วยการกำหนดพารามิเตอร์ที่เหมาะสมให้แก่คำสั่ง DO และ REPEAT

3.1.9 การจัดสรรหน่วยความจำของ dsPIC30F2010



รูปที่ 3.5 การจัดสรรหน่วยความจำโปรแกรมในไมโครคอนโทรลเลอร์ dsPIC

3.1.10 ระบบสัญญาณนาฬิกาใน dsPIC



รูปที่ 3.6 ไดอะแกรมการทำงานของระบบสัญญาณนาฬิกาใน dsPIC30F2010

มีคุณสมบัติโดยสรุปดังนี้

- สามารถเลือกแหล่งกำเนิดสัญญาณนาฬิกาได้จากทั้งภายในและภายนอก
- มีวงจรเฟสล็อกลูป (PPL) อยู่ภายในเพื่อเพิ่มความถี่ของสัญญาณนาฬิกา
- สามารถเปลี่ยนแหล่งกำเนิดสัญญาณนาฬิกาได้ระหว่างการทำงาน
- มีโพสต์สเกลเลอร์แบบโปรแกรมได้เพื่อลดพลังงานของระบบ
- มีส่วนตรวจสอบความผิดพลาดของสัญญาณนาฬิกาและสามารถเปลี่ยนแหล่ง กำเนิดสัญญาณนาฬิกาอย่างอัตโนมัติในกรณีพบความผิดพลาด เพื่อช่วยให้ระบบยังคงทำงานต่อไปได้

โหมดวงจรกำเนิดสัญญาณนาฬิกา	รายละเอียด	ประเภทของสัญญาณนาฬิกา	ปิดควบคุมการทำงาน						การทำงานของ OSC2
			FOS1	FOS0	FPR3	FPR2	FPR1	FPR0	
XTL	คริสตอลภายนอก 200kHz-4MHz ต่อที่ขา OSC1 และ OSC2	สัญญาณนาฬิกาหลัก	1	1	0	0	0	x	OSC2
XT	คริสตอลภายนอก 4MHz-10MHz ต่อที่ขา OSC1 และ OSC2	สัญญาณนาฬิกาหลัก	1	1	0	1	0	0	OSC2
XT PLL4x	คริสตอลภายนอก 4MHz-10MHz ต่อที่ขา OSC1 และ OSC2 โดยทำงานร่วมกับ PLL 4 เท่า	สัญญาณนาฬิกาหลัก	1	1	0	1	0	1	OSC2
XT PLL8x	คริสตอลภายนอก 4MHz-10MHz ต่อที่ขา OSC1 และ OSC2 โดยทำงานร่วมกับ PLL 8 เท่า	สัญญาณนาฬิกาหลัก	1	1	0	1	1	0	OSC2
XT PLL16x	คริสตอลภายนอก 4MHz-10MHz ต่อที่ขา OSC1 และ OSC2 โดยทำงานร่วมกับ PLL 16 เท่า	สัญญาณนาฬิกาหลัก	1	1	0	1	1	1	OSC2
LP	คริสตอลภายนอก 32kHz ต่อที่ขา SOSCO และ SOSCI	สัญญาณนาฬิการอง	0	0	-	-	-	-	-
HS	คริสตอลภายนอก 10MHz-25MHz	สัญญาณนาฬิกาหลัก	1	1	0	0	1	x	OSC2
EC	สัญญาณนาฬิกาภายนอกป้อนเข้าที่ขา OSC1 ความถี่ 0-40MHz	สัญญาณนาฬิกาหลัก	1	1	1	0	1	1	CLKO
ECIO	สัญญาณนาฬิกาภายนอกป้อนเข้าที่ขา OSC1 ความถี่ 0-40MHz โดยที่ขา OSC2 เป็นขาพอร์ตอินพุตเอาต์พุต	สัญญาณนาฬิกาหลัก	1	1	1	1	0	0	I/O
EC PLL4x	สัญญาณนาฬิกาภายนอกป้อนเข้าที่ขา OSC1 ความถี่ 0-40MHz โดยที่ขา OSC2 เป็นขาพอร์ตอินพุตเอาต์พุต โดยทำงานร่วมกับ PLL 4 เท่า	สัญญาณนาฬิกาหลัก	1	1	1	1	0	1	I/O
EC PLL8x	สัญญาณนาฬิกาภายนอกป้อนเข้าที่ขา OSC1 ความถี่ 0-40MHz โดยที่ขา OSC2 เป็นขาพอร์ตอินพุตเอาต์พุต โดยทำงานร่วมกับ PLL 8 เท่า	สัญญาณนาฬิกาหลัก	1	1	1	1	1	0	I/O
EC PLL16x	สัญญาณนาฬิกาภายนอกป้อนเข้าที่ขา OSC1 ความถี่ 0-40MHz โดยที่ขา OSC2 เป็นขาพอร์ตอินพุตเอาต์พุต โดยทำงานร่วมกับ PLL 16 เท่า	สัญญาณนาฬิกาหลัก	1	1	1	1	1	1	I/O
ERC	วงจร RC ภายนอก ต่อเข้าที่ขา OSC1 โดยที่ขา OSC2 เป็นขาเอาต์พุตสัญญาณนาฬิกาความถี่ $F_{OSC}/4$	สัญญาณนาฬิกาหลัก	1	1	1	0	0	1	CLKO
ERCIO	สัญญาณนาฬิกาภายนอกป้อนเข้าที่ขา OSC1 ความถี่ 0-40MHz โดยที่ขา OSC2 เป็นขาพอร์ตอินพุตเอาต์พุต	สัญญาณนาฬิกาหลัก	1	1	1	0	0	0	I/O
FRC	สัญญาณนาฬิกาภายในจากวงจร RC ความถี่ 8MHz	สัญญาณนาฬิกาภายใน FRC	0	1	-	-	-	-	-
LPRC	สัญญาณนาฬิกาภายในจากวงจร RC ความถี่ 512kHz	สัญญาณนาฬิกาภายใน LPRC	1	0	-	-	-	-	-

หมายเหตุ : บิต FOS1, FOS2, FPR0-FPR3 อยู่ในรีจิสเตอร์ FOSC ซึ่งใช้กำหนดการทำงานของวงจรกำเนิดสัญญาณนาฬิกาใน dsPIC

รูปที่ 3.7 ตารางรายละเอียดโดยสรุปของโหมดสัญญาณนาฬิกาใน dsPIC30F2010

3.1.11 การรีเซตในไมโครคอนโทรเลอร์ dsPIC30F2010

dsPIC30F2010 รองรับกระบวนการรีเซตอยู่หลายแบบดังนี้

-เพาเวอร์-ออนรีเซต (Power-on Reset: POR) เป็นการรีเซตเนื่องการจ่ายไฟเลี้ยง

-การรีเซตที่ขา MCLR ในขณะเครื่องทำงานปกติ (EXTR)

-การรีเซตที่ขา MCLR ในโหมดสลีป (SPEEP หรือ IDLE)

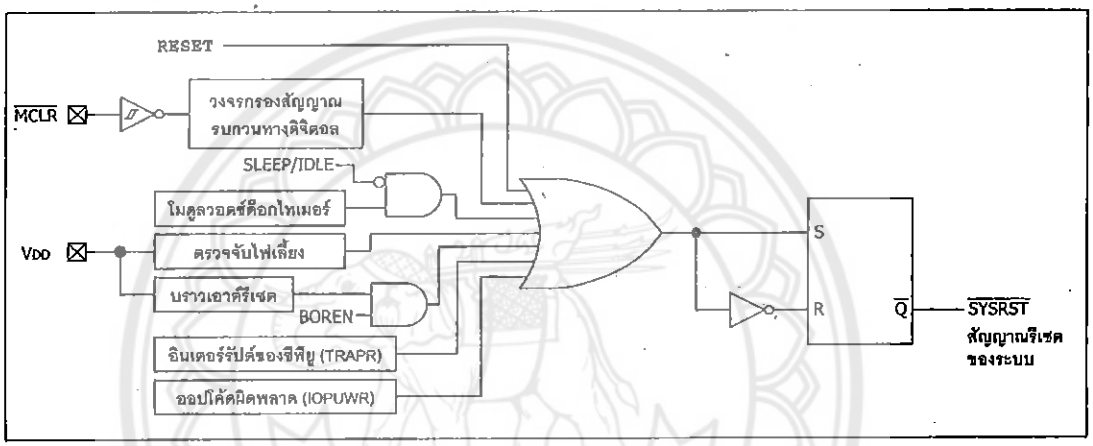
-การรีเซตเนื่องจากวอตช์ด็อกไทม์ในขณะทำงานปกติ (WDTR)

-บราวเอาต์รีเซต (Programmable Brown – out Reset: BOR) เป็นการรีเซตเนื่องจากระดับไฟเลี้ยงลดต่ำกว่าค่าที่กำหนด

-การรีเซตจากคำสั่ง RESET (SWR)

-การรีเซตเนื่องจากการอินเตอร์รัปต์ของซีพียู (TRAPR)

-การรีเซตเนื่องจากออปโค้ดผิดพลาดหรือจากใช้รีจีสเตอร์ W ตัวที่ยังไม่พร้อมทำงานไปทำงานเป็นตัวชี้แอดเดรส (Illegal opcode, or by using an uninitialized W register as an address pointer: IOPUWR)



รูปที่ 3.8 ไดอะแกรมของระบบรีเซตใน dsPIC30F2010

3.1.12 การทำงานในโหมดประหยัดพลังงานของ dsPIC30F2010

มี 2 โหมด คือ *โหมดสลีป (SLEEP)* และ*โหมด (IDLE)* หรืออาจเปรียบเทียบว่าโหมดหลับ (sleep) และโหมดสงบหรือหยุดนิ่ง (idle)

การเข้าสู่โหมดประหยัดพลังงานของ dsPIC ทำได้โดยการเรียกใช้คำสั่ง PWRSAV แล้วต่อท้ายด้วยโหมดที่ต้องการดังนี้

```
PWRSAV  # SLEEP_MODE    ; กำหนดให้ทำงานในโหมดสลีป
PWRSAV  #IDLE_MODE      ; กำหนดให้ทำงานในโหมดไอเดิล
```

โหมดสลีป

เป็นโหมดที่ประหยัดพลังงานมากที่สุด เนื่องจากซีพียู วงจรกำเนิดสัญญาณนาฬิกาของระบบโมดูลฟังชั่นพิเศษทั้งหมดที่ต้องทำงานร่วมกับวงจรกำเนิดสัญญาณนาฬิกาหลักของระบบจะถูกดิสเอเบิลหรือปิดการทำงานทั้งหมด

โหมดไอเดิล

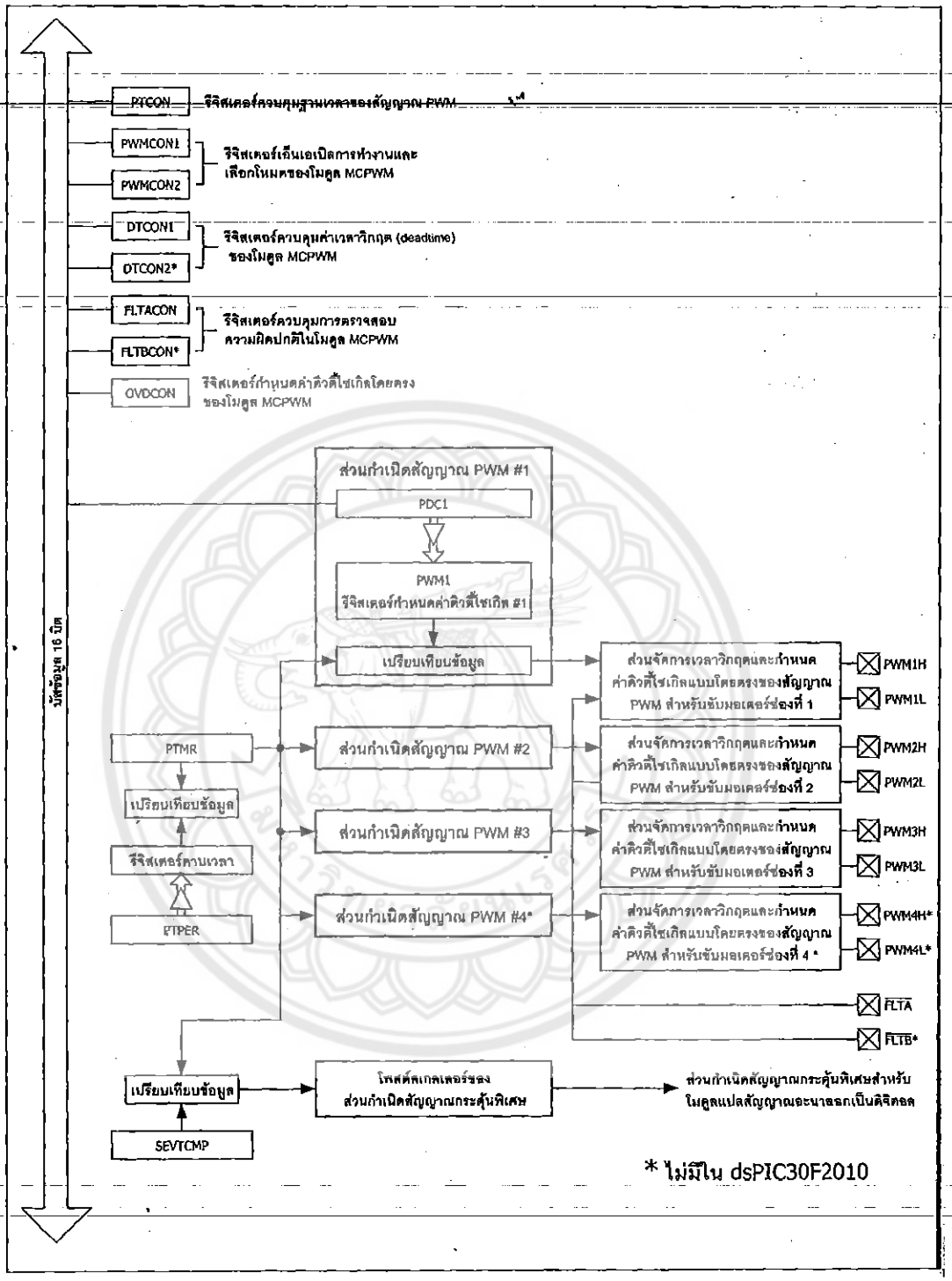
ในโหมดนี้ซีพียูจะหยุดทำงานแต่วงจรกำเนิดสัญญาณนาฬิกาของระบบโมดูลฟังก์ชั่นพิเศษทั้งหมดต้องทำงานร่วมกันกับวงจรกำเนิดสัญญาณนาฬิกาหลักของระบบจะยังคงทำงานต่อไปแต่ผู้ใช้งานสามารถเลือกให้หยุดทำงานเมื่อเข้าสู่โหมดไอเดิล

3.2 การใช้งานโมดูล MCPWM ใน dsPIC30F2010 เพื่อควบคุมมอเตอร์

โมดูลสร้างสัญญาณ PWM เพื่อควบคุมมอเตอร์ หรือเรียกว่า โมดูล MCPWM (Motor Control PWM) ใน dsPIC บรรจุโมดูลนี้ไว้ตั้งแต่ 6 ถึง 8 ช่อง สำหรับ dsPIC30F2010 ที่ใช้อ้างอิงเป็นหลักปริญญานิพนธ์เล่มนี้มี 6 ช่องจึงสามารถขับมอเตอร์แบบเฟสเดี่ยวได้ 3 ตัว และมอเตอร์ 3 เฟส ได้ 1 ตัว ดังนั้นจึงเหมาะอย่างยิ่งที่จะนำไปใช้ควบคุมอินดักชั่นมอเตอร์ 3 เฟส, สวิตช์รีลักแตนซ์มอเตอร์ (SR), มอเตอร์แบบไม่มีแปรงถ่านหรือบรัชเลสมอเตอร์ (BLDC) และในระบบเครื่องสำรองไฟฉุกเฉินหรือ UPS (Un-interrupted Power Supply)

3.2.1 คุณสมบัติโดยสรุปของโมดูล MCPWM

1. ความละเอียดของสัญญาณ PWM ที่สร้างขึ้นเท่ากับ $\frac{Tcy}{2}$
2. ในโมดูล MCPWM 1 ชุด มี 2 เอาต์พุต ใน dsPIC30F2010 มีโมดูล 3 ชุด จึงมีทั้งสิ้น 6 ช่อง
3. สามารถใช้งานของเอาต์พุตของโมดูล MCPWM แยกกันอย่างอิสระและร่วมกันเมื่อทำงานในแบบร่วมกันหรือคอมพลีเมนตารีสามารถกำหนดค่าเวลาวิกฤต (dead time) เพื่อช่วยให้การขับมอเตอร์ 3 เฟส เป็นไปอย่างมีประสิทธิภาพ
4. สามารถเลือกโหมดเอาต์พุตได้ 4 โหมด
    1. โหมดปรับขอบสัญญาณ (Edge aligned mode)
    2. โหมดสัญญาณเดียว (Single event mode)
    3. โหมดปรับสัญญากึ่งกลาง (Center aligned mode)
    4. โหมดปรับสัญญาณกึ่งกลางพร้อมปรับปรุงค่า (Center aligned mode with double updates)
5. มีอินพุตสำหรับตรวจจับความผิดพลาดในการทำงาน (FAULT) แบบโปรแกรมได้
6. สามารถสร้างสัญญาณกระตุ้นส่งไปยังโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอลเพื่อกำหนดจังหวะการทำงานให้สัมพันธ์กัน



รูปที่ 3.9 ไดอะแกรมการทำงานของโมดูล MCPWM ในไมโครคอนโทรลเลอร์ dsPIC

ส่วนประกอบหลักของโมดูลนี้คือ ส่วนกำเนิดสัญญาณ PWM ที่มีอยู่ด้วยกันสูงสุด 4 ชุด โดยใน dsPIC30F2010 จะมี 3 ชุดซึ่งได้ค่าฐานเวลามาจากรีจิสเตอร์ PTMR และ PTTER ส่วนค่าดิวตี้ไซเกิลของสัญญาณ PWM ในโมดูล MCPWM นี้สามารถกำหนดได้จากรีจิสเตอร์ดิวตี้ไซเกิลในส่วนกำเนิดสัญญาณ PWM แต่ละส่วนที่เป็นอิสระต่อกัน นอกจากนั้นยังสามารถกำหนดการทำงานของขาพอร์ตเอาต์พุตของโมดูล MCPWM โดยตรงผ่านทางรีจิสเตอร์ OVDCON

ส่วนกำเนิดสัญญาณ PWM สำหรับควบคุมมอเตอร์แต่ละชุดในโมดูล MCPWM สามารถกำเนิดให้ทำงานแยกจากกันเป็นอิสระ (Independent mode) หรือทำงานร่วมกัน (Complementary mode) เพื่อขับมอเตอร์ 3 เฟสได้ โดยกำหนดผ่านรีจิสเตอร์ PWMCON1 และ PWMCON2 และเมื่อกำหนดให้ทำงานร่วมกันจะต้องมีการจัดการสัญญาณเพื่อให้มอเตอร์ในแต่ละเฟสสามารถทำงานได้อย่างต่อเนื่อง นั่นคือการจัดการค่าเวลาหน่วงเฟส (dead time control) โดยใช้รีจิสเตอร์ DTCON1 และ DTCON2 (ไม่มีใน dsPIC30F2010)

สัญญาณที่ออกจากโมดูล MCPWM จะมีขาพอร์ต 2 ขาต่อช่องนั่นคือ ขาเอาต์พุตด้านแรงดันสูง–PWMxH และขาเอาต์พุตด้านแรงดันต่ำ–PWMxL (x คือหมายเลขของช่องเอาต์พุตมี 4 ค่า คือ 1-4 โดยใน dsPIC30F2010 มีเพียง1-3) หรือเรียกว่า คู่เอาต์พุต นอกจากนั้นยังสามารถส่งสัญญาณเอาต์พุตผ่านโพสต์สเกลเลอร์เพื่อสร้างเป็นสัญญาณกระตุ้นพิเศษให้แก่โมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC) ด้วย เพื่อให้โมดูล MCPWM สามารถทำงานสัมพันธ์กับโมดูล ADC ได้ด้วย

นอกจากนั้นในโมดูล MCPWM ยังมีอินพุตสำหรับรับสัญญาณตรวจสอบผิดปกติหรือ FAULT เพื่อป้องกันไม่ให้โมดูล MCPWM ทำงานผิดพลาดหรือเสียหายเมื่อเกิดความผิดปกติขึ้นในโมดูล MCPWM โดยในส่วนนี้มีพอร์ตอินพุตสำหรับรับสัญญาณ 2 ขา คือ FLTA และ FLTB สำหรับใน dsPIC30F2010 จะมีเพียงขาเดียวคือ FLTA โดยการทำงานในส่วนนี้ได้รับการควบคุมจากรีจิสเตอร์ FLTACON สำหรับส่วนตรวจสอบความผิดปกติชุด A และ FLTBCON สำหรับตรวจสอบความผิดปกติชุด B (ไม่มีใน dsPIC30F2010)

3.2.2 รีจิสเตอร์ที่ใช้งานในโมดูล MCPWM

PTCON (PWM Time Base Control Register)

รีจิสเตอร์ควบคุมฐานเวลาในการกำเนิดสัญญาณ PWM

บิต 15	บิต 14	บิต 13	บิต 12	บิต 11	บิต 10	บิต 9	บิต 8
PTEN	-	PTSIDL	-	-	-	-	-
R/W -0	U -0	R/W -0	U -0	U -0	U -0	U -0	U -0
บิต 7	บิต6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0

PTOPS3	PTOPS2	PTOPS1	PTOPS0	PTCKPS1	PTCKPS0	PTCKPS1	PTCKPS0
R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0

บิต 15 – PTEN (PWM Time Base Timer Enable bit)

บิตเอ็นเอเบิลการทำงานของฐานเวลา PWM

"0" = ปิดฐานเวลา

"1" = เปิดให้ฐานเวลาของสัญญาณ PWM ทำงาน

บิต 14,12 ถึง 8 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 13 – PTSIDL (PWM Time Base Stop in IDLE Mode bit)

บิตเลือกการทำงานของส่วนฐานเวลา PWM ในโหมดประหยัดพลังงาน

"0" = ฐานเวลา PWM ยังคงทำงานต่อไป แม้ว่าซีพียูจะเข้าสู่โหมดประหยัดพลังงานก็ตาม

"1" = ฐานเวลา PWM หยุดทำงานทันทีที่เข้าสู่โหมดพลังงาน

บิต 12 ถึง 8 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 7 ถึง 4 – PTOPS3 ถึง PTOPS0 (PWM Time Base Output Post scale Select bits)

บิตเลือกการลดทอนทางเอาต์พุตของฐานเวลา PWM

"0000" = 1:1

"0001" = 1:2 ผ่านโพสต์สเกลเลอร์

"1111" = 1:16 ผ่านโพสต์สเกลเลอร์

บิต 3 และ 2 –PTCKRS1และ PTCKRS0 (PWM Time Base Input Clock Prescale Select bits)

บิตเลือกอัตราลดทอนสัญญาณนาฬิกาอินพุตของฐานเวลา PWM เพื่อกำหนดคาบเวลา

"00" = คาบเวลาของสัญญาณของนาฬิกาอินพุตเท่ากับ PCY (อัตราปรีสเกลเลอร์เท่ากับ 1: 1)

"01" = คาบเวลาของสัญญาณของนาฬิกาอินพุตเท่ากับ 4 PCY (อัตราปรีสเกลเลอร์เท่ากับ 1: 4)

"10" = คาบเวลาของสัญญาณของนาฬิกาอินพุตเท่ากับ 16 PCY (อัตราปรีสเกลเลอร์เท่ากับ 1: 16)

"11" = คาบเวลาของสัญญาณของนาฬิกาอินพุตเท่ากับ 64 PCY (อัตราปรีสเกลเลอร์เท่ากับ 1: 64)

บิต 1และ 0 –PTMOD1 และ PTMOD0 (PWM Time Base Mode Select bits)

บิตเลือกโหมดการทำงานของฐานเวลา PWM

"00" = ฐานเวลา PWM ทำงานในโนโหมดเปลี่ยนแปลงค่าอิสระ

"01" = ฐานเวลา PWM ทำงานในโนโหมดทำงานครั้งเดียว

"10" = ฐานเวลา PWM ทำงานในโนโหมดนับค่าขึ้นลงอย่างต่อเนื่อง

"11" = ฐานเวลา PWM ทำงานในโนโหมดนับค่าขึ้นลงอย่างต่อเนื่องพร้อมปรับปรุงค่า PWM

PTMR (PWM Time Base Register)

รีจิสเตอร์กำหนดค่าฐานเวลา PWM

บิต 15 –PTDIR (PWM Time Base Count Direction Status)

บิตแจ้งทิศทางการนับค่าของฐานเวลา PWM

"0" = ฐานเวลา PWM กำลังนับขึ้น

"1" = ฐานเวลา PWM กำลังนับลง

บิต 14 ถึง 0 – PTMR14 ถึง PTMR0 (PWM Time Base Register Count Value bits)

บิตกำหนดค่าฐานเวลาของการกำเนิดสัญญาณ PWM

มีด้วยกัน 15 บิต เป็นค่าตัวเลขแบบไม่คิดเครื่องหมาย

PTPER (PWM Time Base Period Register)

รีจิสเตอร์กำหนดคาบเวลาของฐานเวลา PWM

บิต 15 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 14 ถึง 0 –PTPER14 ถึง 0 (PWM Time Base Period Value bits)

บิตกำหนดค่าคาบเวลาของฐานเวลา PWM

มีด้วยกัน 15 บิต เป็นค่าตัวเลขแบบไม่คิดเครื่องหมาย

SEVTCMP (Special Event Compare Register)

รีจิสเตอร์เปรียบเทียบค่าสำหรับสร้างสัญญาณกระตุ้นพิเศษ

บิต 15	บิต 14	บิต 13	บิต 12	บิต 11	บิต 10	บิต 9	บิต 8
SEVTDIR	SEVTCMP14	SEVTCMP14	SEVTCMP14	SEVTCMP14	SEVTCMP14	SEVTCMP9	**SEVTCMP8**
R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	U -0	U -0
บิต 7	บิต6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
SEVTCMP7	SEVTCMP6	SEVTCMP5	SEVTCMP4	SEVTCMP3	SEVTCMP2	SEVTCMP1	SEVTCMP0
R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0

บิต 15 – SEVTDIR (Special Event Trigger Time Base Direction bit)

บิตกำหนดการเกิดสัญญาณกระตุ้นพิเศษ (1)

"0" = เกิดสัญญาณกระตุ้นพิเศษเมื่อฐานเวลา PWM นับค่าขึ้น

"1" = เกิดสัญญาณกระตุ้นพิเศษเมื่อฐานเวลา PWM นับค่าลง

บิต 14 ถึง 0 –SEVTCMP14 ถึง SEVTCMP0 (Special Event Compare Value bit)

บิตกำหนดค่าสำหรับเปรียบเทียบ มีด้วยกัน 15 บิต เป็นค่าตัวเลขแบบไม่คิดเครื่องหมาย

หมายเหตุ [1]บิต SEVTDIR จะถูกเปรียบเทียบกับบิต PTDIR (บิต 15 ของรีจิสเตอร์ PTMR) เพื่อสร้างสัญญาณกระตุ้นพิเศษ ส่วนข้อมูลในบิต SEVTCMP14 ถึง SEVTCMP0 จะถูกเปรียบเทียบกับบิต PTMR14 ถึง PTMR0 เพื่อสร้างสัญญาณกระตุ้นพิเศษ

PWMCON1 (PWM Control Register 1)

รีจิสเตอร์ควบคุม PWM #1

บิต 15	บิต 14	บิต 13	บิต 12	บิต 11	บิต 10	บิต 9	บิต 8
-	-	-	-	PMOD4	PMOD3	PMOD2	PMOD1
U -0	U -0	U -0	U -0	R/W -0	R/W -0	R/W -0	R/W -0
บิต 7	บิต6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
PEN4H	PEN3H	PEN2H	PEN1H	PEN4H	PEN3H	PEN2H	PEN1H
R/W -1	R/W -1	R/W -1	R/W -1	R/W -1	R/W -1	R/W -1	R/W -1

บิต 15 ถึง 12 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 11 ถึง 8 – PMOD4 ถึง PMOD1 (PWM I/O Pair Mode bits)

บิตกำหนดโหมดทำงานของคู่เอาต์พุต

"0" = กำหนดให้คู่ของขาพอร์ตของโมดูล MCPWM ทำงานเป็นเอาต์พุตแบบคอมพลีเมนตารีหรือแบบทำงานร่วมกับคู่เอาต์พุตอื่น

"1" = กำหนดให้คู่ของขาพอร์ตของโมดูล MCPWM ทำงานเป็นเอาต์พุตแบบอิสระ

บิต 7 ถึง 4 –PEN4H ถึง PEN1H (PWMxH I/O Enable bits)

บิตเอนเอเบิลขาเอาต์พุตด้านแรงดันสูงของโมดูล MCPWM [1]

"0" = กำหนดให้ขา PWMxH ทำงานเป็นขาพอร์ตอินพุตเอาต์พุตปกติ

"1" = กำหนดให้ขา PWMxH เป็นขาเอาต์พุตด้านแรงดันสูงของโมดูล MCPWM

บิต 3 ถึง 0 - PEN4L ถึง PEN1L (PWMxL I/O Enable bits)

บิตเอนเอเบิลขาเอาต์พุตด้านแรงดันต่ำของโมดูล MCPWM[1]

"0" = กำหนดให้ขา PWMxL ทำงานเป็นขาพอร์ตอินพุตเอาต์พุตปกติ

"1" = กำหนดให้ขา PWMxL เป็นขาเอาต์พุตด้านแรงดันต่ำของโมดูล MCPWM

หมายเหตุ [1]หลังจากรีเซต สถานะของบิต PWMxH จะขึ้นอยู่กับค่าที่กำหนดให้ในรีจิสเตอร์FBORPOR

PWMCON2 (PWM Control Register 2)

รีจิสเตอร์ควบคุม PWM #2

บิต 15	บิต 14	บิต 13	บิต 12	บิต 11	บิต 10	บิต 9	บิต 8
-	-	-	-	SEVOPS3	SEVOPS2	SEVOPS1	SEVOPS0
U -0	U -0	U -0	U -0	R/W -0	R/W -0	R/W -0	R/W -0
บิต 7	บิต6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
-	-	-	-	-	-	OSYNC	UDIS
U -0	U -0	U -0	U -0	U -0	U -0	R/W -0	R/W -0

บิต 15 ถึง 12 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 11 ถึง 8-SEVOPS3 ถึงSEVOPS0 (PWM Special Event Trigger Output Post scale Select bits)

บิตกำหนดอัตราโพสต์สเกลเลอร์ของสัญญาณกระตุ้นพิเศษ

"0000" = 1:1

"0001" = 1:2

.

.

"1111" = 1:16

บิต 7 ถึง 2 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 1 -- OSYNC 1(Output Override Synchronization bit)

บิตกำหนดจังหวะการเปลี่ยนค่าสัญญาณเอาต์พุต

"0" = เปลี่ยนแปลงค่าชิงรีจิสเตอร์ OVDCON ในไซเกิลของการทำงานถัดไป

"1" = เปลี่ยนแปลงค่าชิงรีจิสเตอร์ OVDCON เมื่อจังหวะการทำงานตรงกับฐานเวลาของ PWM

บิต 0 –UDIS (PWM Update Disable bit)

บิตดิสเอเบิลการปรับปรุงค่าดิวตี้ไซเกิลของสัญญาณ PWM

"0" = เลือกเพื่อยอมให้เกิดการปรับปรุงค่าดิวตี้ไซเกิลและรีจิสเตอร์บัฟเฟอร์ที่ใช้กำหนดคาบเวลา

"1" = เลือกเพื่อไม่ยอมให้เกิดการปรับปรุงค่าดิวตี้ไซเกิลและรีจิสเตอร์บัฟเฟอร์ที่ใช้กำหนดคาบเวลา

DTCON1 (Dead Time Control Register # 1)

รีจิสเตอร์ควบคุมค่าเวลาวิกฤตหรือ Dead Time # 1

บิต 15	บิต 14	บิต 13	บิต 12	บิต 11	บิต 10	บิต 9	บิต 8
DTBPS1	DTBPS0	DTB5	DTB4	DTB3	DTB2	DTB1	DTB0

R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0
บิต 7	บิต6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
DTAPS1	DTAPS0	DTA5	DTA4	DTA3	DTA2	DTA1	DTA0

R/W -0	R/W -0	R/W -0	R/W -0	U -0	U -0	R/W -0	R/W -0

บิต 15 และ 14 -DTBPS1 และ DTBPS0 (Dead Time Unit B Prescale Select bits)

บิตเลือกอัตราปรีสเกลเลอร์ของส่วนกำเนิดค่าเวลาวิกฤตชุด B

*ไม่มีใช้งานใน dsPIC30F2010

บิต 13 ถึง 8 - DTB5 ถึง DTB0 (Unsigned 6bit Dead Time Value bits for Dead Time Unit B)

บิตกำหนดค่าเวลาวิกฤตของส่วนกำเนิดค่าเวลาวิกฤตชุด B

*ไม่มีใช้งานใน dsPIC30F2010

บิต 7 และ 6 - DTAPS1 และ DTAPS0 (Dead Time Unit A Prescale Select bits)

บิตเลือกอัตราปรีสเกลเลอร์ของส่วนกำเนิดค่าเวลาวิกฤตชุด B

"00" = เลือกคาบเวลาของสัญญาณนาฬิกาส่วนกำเนิดค่าเวลาวิกฤตชุด A เป็น Tcy

"01" = เลือกคาบเวลาของสัญญาณนาฬิกาส่วนกำเนิดค่าเวลาวิกฤตชุด A เป็น 2Tcy

"10" = เลือกคาบเวลาของสัญญาณนาฬิกาส่วนกำเนิดค่าเวลาวิกฤตชุด A เป็น 4Tcy

"11" = เลือกคาบเวลาของสัญญาณนาฬิกาส่วนกำเนิดค่าเวลาวิกฤตชุด A เป็น 8Tcy

บิต 5 ถึง 0 - DTA5 ถึง DTA0 (Unsigned 6bit Dead Time Value bits for Dead Time Unit A)

บิตกำหนดค่าเวลาวิกฤตของส่วนกำเนิดค่าเวลาวิกฤตชุด A

มีด้วยกัน 6 บิต เป็นค่าตัวเลขแบบไม่คิดเครื่องหมาย กำหนดค่าได้ตั้งแต่ 000000-111111

DTCON2 (Dead time Control Register 2)

รีจิสเตอร์ควบคุมค่าเวลาวิกฤต #2

*ไม่มีใช้งานใน dsPIC30F2010

FLTACON (Fault A Control Register)

รีจิสเตอร์ควบคุมการตรวจจับความผิดปกติชุด A

บิต 15	บิต 14	บิต 13	บิต 12	บิต 11	บิต 10	บิต 9	บิต 8
FAOV4H	FAOV4L	FAOV3H	FAOV3L	FAOV2H	FAOV2L	FAOV1H	FAOV1L

R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0
บิต 7	บิต6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
FLTM	-	-	-	FAEN4	FAEN3	FAEN2	FAEN1

R/W -0	U -0	U -0	U -0	U -0	R/W -0	R/W -0	R/W -0

บิต 15 และ 14 - FAOV4H- FAOV4L (Fault Input A PWM Override Value bits for PWM4)

บิตกำหนดลักษณะของสัญญาณ PWM ของโมดูล PWM4 จากอินพุตของส่วนตรวจจับความผิดชุด A

*2 บิตนี้ไม่มีใช้งานใน dsPIC30F2010

บิต 13 ถึง 8– FAOV3H- FAOV1L (Fault Input A PWM Override Value bits for PWM3-PWM1)

บิตกำหนดลักษณะของสัญญาณ PWM ของโมดูล PWM3 ถึง PWM1จากอินพุตของส่วนตรวจจับความผิดของการขับมอเตอร์ชุด A

"0" = ขาเอาต์พุตของสัญญาณ PWM เป็นสถานะ "ไม่แอกตีฟ" หรือเป็นสัญญาณแรงดันต่ำ เมื่อเกิดตรวจจับความผิดได้ที่ขา FLTA

"1" = ขาเอาต์พุตของสัญญาณ PWM เป็นสถานะ "แอกตีฟ" หรือเป็นสัญญาณแรงดันสูง เมื่อเกิดตรวจจับความผิดได้ที่ขา FLTA

ในการกำหนดค่าของขาเอาต์พุตคู่ (PWMxH และ PWMxL) ต้องแตกต่างกัน

บิต 7 – FLTAM (Fault A Mode bit)

บิตเลือกโหมดของส่วนตรวจจับความผิดปกติของการขับมอเตอร์ชุด A

"0" = เลือกให้ขาพอร์ตเอาต์พุตทำงานตามที่กำหนดในบิต 15 ถึง 8 ของรีจิสเตอร์ FLTACON เมื่อตรวจจับความผิดปกติได้ที่ขา FLTA

"1" = เลือกให้ขาอินพุต FLTA ทำงานในโหมดไซเกิลต่อไซเกิล (Cycle-by-Cycle mode)

บิต 6 ถึง 4 *ไม่ใช้งาน กำหนดเป็น "0"*

บิต 3 –FAEN4 (Fault Input A Enable bit for PWM4)

บิตเอ็นเอเบิลการทำงานของส่วนตรวจจับ

* บิตนี้ไม่ใช้งานใน dsPIC30F2010

บิต 2 –FAEN3 (Fault Input A Enable bit for PWM3)

บิตเอ็นเอเบิลการทำงานของส่วนตรวจจับความผิดปกติของการขับมอเตอร์ชุด A (FLTA)

"0" = เลือกให้ขา PWM3H/PWM3L ไม่ถูกควบคุมโดยอินพุต FLTA

"1" = เลือกให้ขา PWM3H/PWM3L ถูกควบคุมโดยอินพุต FLTA

บิต 1 –FAEN2 (Fault Input A Enable bit for PWM2)

บิตเอ็นเอเบิลการทำงานของส่วนตรวจจับความผิดปกติของการขับมอเตอร์ชุด A (FLTA)

"0" = เลือกให้ขา PWM2H/PWM2L ไม่ถูกควบคุมโดยอินพุต FLTA

"1" = เลือกให้ขา PWM2H/PWM2L ถูกควบคุมโดยอินพุต FLTA

บิต 0 –FAEN1 (Fault Input A Enable bit for PWM1)

บิตเอ็นเอเบิลการทำงานของส่วนตรวจจับความผิดปกติของการขับมอเตอร์ชุด A (FLTA)

"0" = เลือกให้ขา PWM1H/PWM1L ไม่ถูกควบคุมโดยอินพุต FLTA

"1" = เลือกให้ขา PWM1H/PWM1L ถูกควบคุมโดยอินพุต FLTA

FLTBCON (Fault A Control Register)

รีจิสเตอร์ควบคุมการตรวจจับความผิดปกติชุด B)

รีจิสเตอร์ตัวนี้ไม่ใช้งานใน dsPIC30F2010 ลักษณะการกำหนดค่าเหมือนกับรีจิสเตอร์ FLTBCON อย่างไรก็ตามถ้าหากมีการใช้งานส่วนตรวจจับความผิดปกติของการขับมอเตอร์ทั้ง 2 ชุดพร้อมกัน (ใน dsPIC เบอร์ที่สนับสนุนความสามารถนี้) การกำหนดที่ส่วนตรวจจับความผิดปกติของการขับมอเตอร์ชุด A จะมีนัยสำคัญสูงกว่าชุด B

OVDCON (Override Control Register)

รีจิสเตอร์กำหนดการทำงานของเอาต์พุตของโมดูล MCPWM โดยตรง

เนื่องจากโครงงานนี้กำหนดการทำงานของส่วนกำเนิดสัญญาณพัลส์ PWM เป็นแบบคอมพลีเมนตารีจะทำให้ไม่สามารถกำหนดการทำงานของเอาต์พุตได้โดยตรง จึงไม่ขออธิบายในหัวข้อนี้

PDC1 ถึง PDC4 (PWM Duty Cycle Register)

รีจิสเตอร์กำหนดค่าดิวตี้ไซเกิลของโมดูลกำเนิดสัญญาณ PWM ชุดที่ 1 ถึง 4

เป็นรีจิสเตอร์ขนาด 16 บิต ที่ใช้กำหนดค่าดิวตี้ไซเกิลของสัญญาณ PWM ในโมดูล MCPWM ทั้ง 4 ชุด โดยแยกกันอย่างอิสระคือ PDC1 สำหรับโมดูล PWM ชุดที่ 1 ไล่ไปตามลำดับจนถึง PDC4 สำหรับโมดูล PWM ชุดที่ 4 (ในdsPIC30F2010 จะไม่มีรีจิสเตอร์ PDC4)

FBORPOR (BOR AND POR Device Configuration Register)

รีจิสเตอร์กำหนดค่าสำหรับการพาวเวอร์-ออนรีเซตและบราวเอาต์รีเซต

บิตที่เกี่ยวข้องกับโมดูล MCPWM ประกอบด้วย

บิต 10 –PWMPIN (Motor control PWM Module Pin Mode bit)

บิตกำหนดการทำงานของขาพอร์ตในโมดูล MCPWM

"0"= ขาพอร์ตของโมดูล PWM ควบคุมมอเตอร์ถูกกำหนดให้เป็นขาเอาต์พุตสำหรับรับส่งสัญญาณ PWM เพื่อขับมอเตอร์ (บิต 7 ถึง 0 ของ PWMCON1=0x00)

"1" = ขาพอร์ตของโมดูล PWM ควบคุมมอเตอร์ถูกกำหนดให้เป็นขาอินพุตเอาต์พุตปกติ (บิต 7 ถึง 0 ของ PWMCON1=0xFF)

บิต 9 –HPOL (Motor control PWM Module High Side Polarity bit)

บิตกำหนดขั้วของขาเอาต์พุตของโมดูล MC PWM ด้านแรงดันสูง

"0" = กำหนดให้ขาเอาต์พุตด้านแรงดันสูงของโมดูล PWM เป็นขั้วแรงดันต่ำ

"1" = กำหนดให้ขาเอาต์พุตด้านแรงดันสูงของโมดูล PWM เป็นขั้วแรงดันสูง

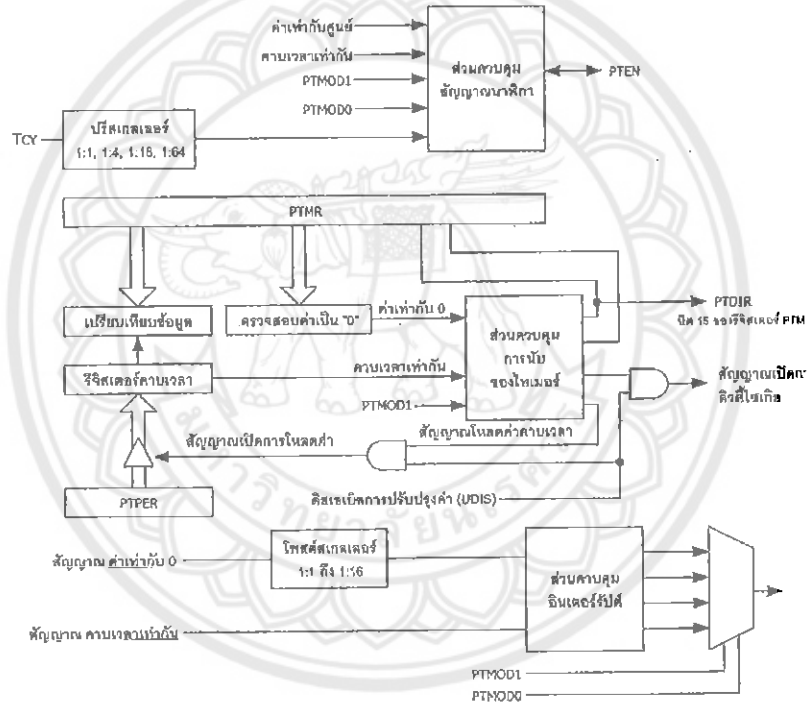
บิต 8 –LPOL (Motor control PWM Module Low Side Polarity bit)

บิตกำหนดขั้วของขาเอาต์พุตของโมดูล MC PWM ด้านแรงดันต่ำ

"0" = กำหนดให้ขาเอาต์พุตด้านแรงดันต่ำของโมดูล PWM เป็นขั้วแรงดันต่ำ

"1" = กำหนดให้ขาเอาต์พุตด้านแรงดันต่ำของโมดูล PWM เป็นขั้วแรงดันสูง

3.2.3 ฐานเวลาสัญญาณ PWM



รูปที่ 3.10 ไดอะแกรมการทำงานของส่วนกำหนดฐานเวลาของสัญญาณ PWM ที่ใช้ในโมดูล MCPWM

ไดอะแกรมของการกำหนดฐานเวลาของสัญญาณ PWM ที่ใช้ในโมดูล MCPWM ซึ่งแยกอิสระจากโมดูลเอาต์พุตเปรียบเทียบ (Output Compare: OC) ค่าฐานเวลาได้มาจากการทำงานของไทเมอร์ 15 บิตร่วมกับปรีสเกลเลอร์และโพสต์สเกลเลอร์ในโมดูล MCPWM โดยข้อมูล15 บิตนั้นบรรจุอยู่ใน 15 บิตล่างของรีจิสเตอร์

PTMR ส่วนบิต MSB คือบิต PTDIR เป็นบิตที่อ่านได้อย่างเดียวในการแสดงทิศทางในการนับค่าในปัจจุบันของฐานเวลา PWM นี้ โดยถ้าบิตนี้เป็น “0” แสดงว่า PTMR กำลังนับค่าขึ้น และเป็น “1” เมื่อกำลังนับค่าลง

การเอ็นเอเบิลให้ส่วนฐานเวลา PWM นี้ทำงานต้องเซตบิต PTEN ซึ่งเป็นบิต 15 ของรีจิสเตอร์ PTCON อย่างไรก็ตาม ค่าในรีจิสเตอร์ PTMR จะไม่ถูกเคลียร์แม้ว่าส่วนฐานเวลาของ PWM นี้จะถูกดิสเอเบิลด้วยการเคลียร์บิต PTEN

ฐานเวลา PWM ในโมดูล MCPWM สามารถกำหนดให้ทำงานได้ 4 โหมดคือ

1. โหมดเปลี่ยนแปลงค่าอิสระ (Free Running mode)
2. โหมดทำงานครั้งเดียว (Single Event mode)
3. โหมดนับค่าขึ้นหรือลงอย่างต่อเนื่อง (Continuous Up/Down Count mode)
4. โหมดนับค่าขึ้นหรือลงอย่างต่อเนื่องพร้อมการอินเตอร์รัปต์เพื่อปรับปรุงค่า (Continuous Up/Down Count mode with interrupts for double-updates)

การเลือกโหมดทำได้โดยการกำหนดค่าที่บิต PTMOD1 และ PTMOD0 ซึ่งเป็นบิต 1 และ 0 ในรีจิสเตอร์ PTCON

ฐานเวลา PWM ในโหมดเปลี่ยนแปลงค่าอิสระ

ในโหมดนี้ค่าฐานเวลาจะเพิ่มค่าขึ้นจนกระทั่งตรงกับค่าในรีจิสเตอร์ PTPER จากนั้นรีจิสเตอร์ PTMR จะรีเซตและทำการนับค่าเพิ่มขึ้นต่อเนื่องไปอีกตราบเท่าที่บิต PTEN ยังคงเซตเป็น “1” อยู่

ฐานเวลา PWM ในโหมดทำงานครั้งเดียว

ในโหมดนี้ฐานเวลาจะเริ่มนับเมื่อมีการเซตบิต PTEN เมื่อค่าของรีจิสเตอร์ PTMR ตรงกับ PTPER รีจิสเตอร์ PTMR จะรีเซต บิต PTEN จะถูกเคลียร์โดยอัตโนมัติด้วยกระบวนการทางฮาร์ดแวร์ทำให้ฐานเวลา PWM นี้หยุดทำงานตามไปด้วย

ฐานเวลา PWM ในโหมดรับค่าขึ้นหรือลงอย่างต่อเนื่อง

ในโหมดนี้ค่าฐานเวลาจะเพิ่มค่าขึ้นจนกระทั่งตรงกับค่าในรีจิสเตอร์ PTPER นอกนั้นจะกลับทิศทางการนับเป็นนับค่าลงแทน จนกระทั่งเท่ากับ “0” แล้วกลับไปเริ่มต้นนับขึ้นใหม่ บิต PTDIR ซึ่งเป็นบิต 15 ของรีจิสเตอร์ PTMR จะแสดงให้ทราบถึงทิศทางการนับในปัจจุบัน โดยเป็น “0” เมื่อนับขึ้นและ “1” เมื่อนับค่าลง

ปรีสเกลเลอร์ของฐานเวลา PWM

สัญญาณนาฬิกาที่ใช้ในการนับค่าของฐานเวลา PWM ในโมดูล MCPWM ก็คือสัญญาณนาฬิกาในการทำงานของระบบ โดยมีค่าความถี่เท่ากับ FCY (ซึ่งเท่ากับ 1/4 ของความถี่หลัก)

สัญญาณนาฬิกาจะส่งผ่านไปนับค่าในรีจิสเตอร์ PTMR โดยมีตัวลดทอนหรือปรีสเกลเลอร์เข้ามาจัดการเพื่อปรับอัตราการนับค่า ซึ่งส่งผลต่อความถี่ของสัญญาณ PWM ดูรูปที่ 3.10 ประกอบ อัตราการลดทอนของปรีสเกลเลอร์เลือกได้ 4 อัตราคือ 1:1 (ไม่ลดทอน), 1:4, 1:16 และ 1:64 โดยกำหนดจากบิต PTCKPS1 และ PTCKPS0 ซึ่งเป็นบิต 3 และ 2 ของรีจิสเตอร์ PTCON

ค่าของปรีสเกลเลอร์จะถูกเคลียร์เมื่อเกิดเหตุการณ์ดังนี้

1. เกิดการเขียนข้อมูลไปยังรีจิสเตอร์ PTMR
2. เกิดการเขียนข้อมูลไปยังรีจิสเตอร์ PTCON
3. เกิดการรีเซตขึ้นในไมโครคอนโทรลเลอร์

โพสต์สเกลเลอร์ของฐานเวลา PWM

เมื่อค่าของรีจิสเตอร์ PTMR ตรงกับค่าของรีจิสเตอร์ PTPER จะเกิดสัญญาณเอาต์พุตขึ้นเพื่อนำไปสร้างสัญญาณอินเตอร์รัปต์ และในส่วนฐานเวลา PWM นี้ได้เพิ่มเติมความสามารถพิเศษตัวหนึ่งคือ โพสต์สเกลเลอร์เพื่อปรับอัตราสัญญาณเอาต์พุตนี้ โดยแทนที่จะส่งสัญญาณเอาต์พุตออกไปทันทีก็สามารถปรับอัตราเอาต์พุตนี้ได้ โดยเลือกได้ถึง 16 ลำดับตั้งแต่ 1:1 (ไม่ปรับค่า) จนถึง 1:16 นั่นคือเกิดเหตุการณ์ค่าตรงกัน 16 ครั้ง จึงส่งสัญญาณเอาต์พุตออกมา ดังนั้นโพสต์สเกลเลอร์จะถูกใช้งานในกรณีที่ไม่ต้องการปรับค่าดิวตี้ไซเกิลในทุกๆ ไซเกิลของสัญญาณ PWM

ค่าตัวนับในโพสต์สเกลเลอร์จะถูกเคลียร์เมื่อเกิดเหตุการณ์ดังนี้

1. เกิดการเขียนข้อมูลไปยังรีจิสเตอร์ PTMR
2. เกิดการเขียนข้อมูลไปยังรีจิสเตอร์ PTCON
3. เกิดการรีเซตขึ้นในไมโครคอนโทรลเลอร์

การอินเตอร์รัพต์ในส่วนฐานเวลา PWM

สัญญาณอินเตอร์รัพต์ที่เกิดขึ้นจากฐานเวลา PWM นี้สามารถกำหนดได้จากบิต PTMOD1 และ PTMOD0 (บิต 1 และ 0 ของรีจิสเตอร์ PTCON) ร่วมกับบิต PTOPS3 ถึง PTOPS0 (บิต 7 ถึง 4 ของรีจิสเตอร์ PTCON) โดยขึ้นกับโหมดการทำงานของฐานเวลา PWM ด้วย ดังนี้

1. อินเตอร์รัพต์ของฐานเวลา PWM ในโหมดเปลี่ยนแปลงค่าอิสระในโหมดนี้การอินเตอร์รัปต์จะเกิดขึ้นเมื่อรีจิสเตอร์ PRMR มีค่าตรงกับค่าในรีจิสเตอร์ PTPER สามารถให้โพสต์สเกลเลอร์เพื่อลดค่าความถี่ของเหตุการณ์อินเตอร์รัปต์นี้ได้
2. อินเตอร์รัพต์ของฐานเวลา PWM ในโหมดทำงานครั้งเดียวในโหมดนี้การอินเตอร์รัปต์จะเกิดขึ้นเมื่อรีจิสเตอร์ PRMR มีค่าตรงกับในรีติสเตอร์

PTPER ไม่สามารถให้โพสต์สเกลเลอร์เพื่อลดค่าความถี่ของเหตุการณ์อินเตอร์รัปต์นี้ได้ นอกนั้นบิต PTEN จะถูกเคลียร์ ทำให้เกิดการดิสเอเบิลการทำงานของส่วนฐานเวลา PWM นี้ตามไปด้วย

3. อินเตอร์รัปต์ของฐานเวลา PWM ในโหมดนับค่าขึ้นลงอย่างต่อเนื่องในโหมดนี้การอินเตอร์รัปต์จะเกิดขึ้นทุกครั้งที่รีจิสเตอร์ PRMR เป็น "0" จากนั้นค่าฐานเวลา PWM จะเริ่มนับค่าขึ้น สามารถใช้โพสต์สเกลเลอร์เพื่อลดค่าความถี่ของเหตการณ์อินเตอร์รัปต์นี้ได้

4. อินเตอร์รัปต์ของฐานเวลา PWM ในโหมดนับค่าขึ้นลงอน่างต่อเนื่องพร้อมการอินเตอร์รัปต์เพื่อปรับปรุงค่า ในโหมดนี้การอินเตอร์รัปต์จะเกิดขึ้นทุกครั้งที่รีจิสเตอร์ PRMR เป็น "0" และทุกครั้งที่ค่าของคาบเวลาตรงกัน ไม่สามารถใช้โพสต์สเกลเลอร์เพื่อลดค่าความถี่ของเหตุการณ์อินเคอร์รัปต์นี้ได้ นอกจากนั้นในโหมดนี้ยังกำหนดให้เกิดการปรับปรุงค่าดิวตี้ไซเกิล 2 ครั้งต่อคาบเวลา โดยสามารถเลือกได้ว่าต้องการให้ทำงานที่ขอบขาขึ้นหรือลงของสัญญาณ PWM

คาบเวลาของสัญญาณ PWM

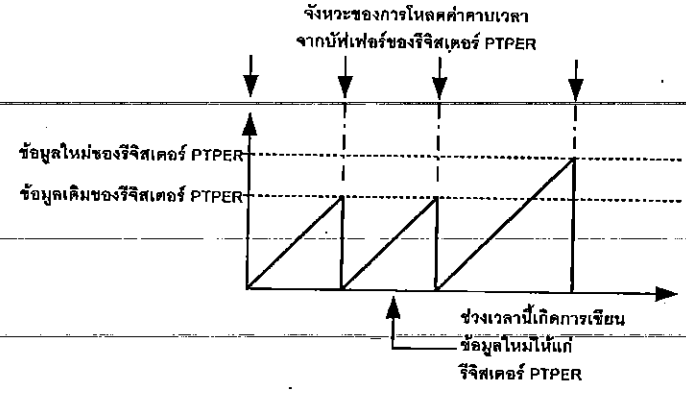
รีจิสเตอร์ PTPER ถูกใช้สำหรับกำหนดค่าการนับคาบเวลาของรีจิสเตอร์ PRMR ผู้พัฒนาต้องระบุข้อมูลขนาด 15 บิต ลงในบิต 0 ถึง 14 ของรีจิสเตอร์ PTPER เมื่อโมดูลนี้ทำงานจนกระทั่งค่าของรีจิสเตอร์ PRMR เท่ากับ PTPER ค่าฐานเวลาจะรีเซตเป็น"0" หรือเปลี่ยนทิศทางการนับค่าในสัญญาณนาฬิกาถูกถัดไป ขึ้นอยู่กับการกำหนดโหมดทำงาน

คาบเวลาของฐานเวลามีขนาดของบัฟเฟอร์เป็น 2 เท่าเพื่อรองรับการเปลี่ยนแปลงค่าในระหว่างการทำงานได้ โดยปราศจากการรบกวน นั่นคือรีจิสเตอร์ PTPER จะมีรีจิสเตอร์บัฟเฟอร์สำหรับรองรับค่าที่ต้องการเปลี่ยนแปลงใหม่ ในระหว่างที่กำลังทำงานกับค่าเดิม โดยรีจิสเตอร์บัฟเฟอร์นี้ผู้ใช้งานไม่สามารถเข้าถึงได้ ข้อมูลสำหรับกำหนดค่าคาบเวลาจะถูกเขียงลงในรีจิสเตอร์ PTPER แตกต่างกันไปตามโหมดการทำงานของฐานเวลา PWM ดังนี้

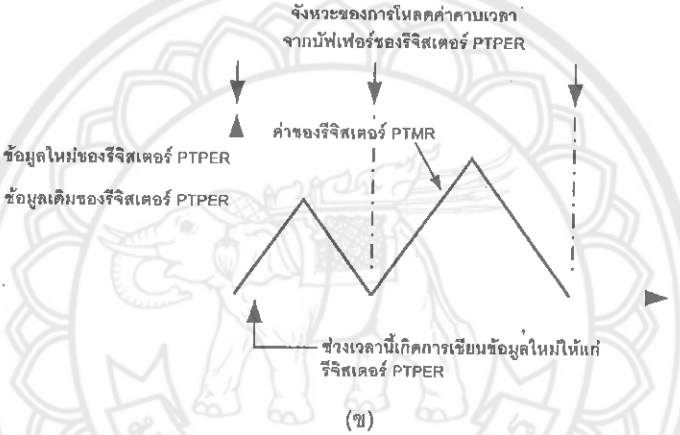
1.ในโหมดเปลี่ยนแปลงค่าอิสระและโหมดทำงานครั้งเดียว ข้อมูลจากรีจิสเตอร์ PTPERจะถูกโหลดลงในรีจิสเตอร์คาบเวลาเมื่อรีจิสเตอร์ PRMR ถูกรีเซตเป็น "0" หลังจากที่ค่าของรีจิสเตอร์ PRMR ตรงกับค่าของ PTPER ดังแสดงด้วยไดอะแกรมเวลาในรูปที่ 3.11 (ก)

2.ในโหมดนับค่าขึ้นลง ข้อมูลจากรีจิสเตอร์ PTPER จะถูกโหลดลงในรีจิสเตอร์คาบเวลาเมื่อรีจิสเตอร์ PRMR มีค่าเป็น "0" ดังแสดงด้วยไดอะแกรมเวลาในรูปที่ 3.11 (ข)

นอกจากนั้นข้อมูลในรีจิสเตอร์ PTPER จะถูกโหลดไปยังรีจิสเตอร์คาบเวลาอย่างอัตโนมัติเมื่อฐานเวลา PWM ถูกดิสเอเบิลโดยกำหนดบิต PTEN เป็น "0"



(ก)



(ข)

รูปที่ 3.11 ไดอะแกรมเวลาแสดงการโหลดค่าของรีจิสเตอร์ PTPER

ก. เมื่อฐานเวลาของ PWM ทำงานในโหมดเปลี่ยนแปลงค่าอิสระและโหมดทำงานครั้งเดียว

ข. เมื่อฐานเวลาของ PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่อง

ข้อมูลสำหรับกำหนดคาบเวลาของสัญญาณ PWM ที่เขียนไปยังรีจิสเตอร์ PTPER สามารถคำนวณได้จากสมการต่อไปนี้

$$PEPER = \frac{F_{CY}}{F_{PWM} \times PTMR\_prescaler} - 1$$

<u>ตัวอย่างที่ 1</u>

ความถี่ของการทำงาน (F_{CY}) เท่ากับ 20 MHz

ความถี่ของสัญญาณ PWM (F_{PWM}) เท่ากับ 20 kHz

อัตราปรีสเกลเลอร์เท่ากับ 1:1

ดังนั้น ค่าของ PTPER เท่ากับ $\frac{20000000}{20000} - 1 = 1000-1=999$ (ฐานสิบ)

3.2.4 หน่วยเปรียบเทียบค่าดิวตี้ไซเกิลของสัญญาณ PWM

ในโมดูล MCPWM มีส่วนกำเนิดสัญญาณ PWM สูงสุด 4 ชุด (สำหรับใน dsPIC30F2010 มี 3 ชุด) จึงมีรีจิสเตอร์ขนาด 16 บิตสำหรับกำหนดค่าดิวตี้ไซเกิล 4 ตัว คือ PDC1 ถึง PDC4 สำหรับในอธิบายจะใช้ PDCx แทน (x แทนตัวเลข 1 ถึง 4 ซึ่งเป็นหมายเลขของส่วนกำเนิดสัญญาณ PWM)

ความละเอียดสูงสุดของค่าดิวตี้ไซเกิลจะขึ้นอยู่กับความถี่สัญญาณนาฬิกาหลักของอุปกรณ์นั้นๆ และความถี่ของสัญญาณ PWM โดยสามารถคำนวณได้จากสมการดังนี้

$$\text{ความละเอียด} = \frac{\log\left(\frac{2T_{PWM}}{T_{CY}}\right)}{\log(2)}$$

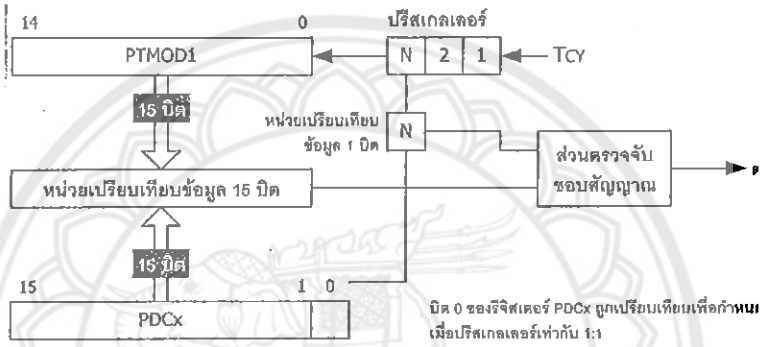
ค่าเวลา T_{CY} (ความถี่ F_{CY})	ค่าของรีจิสเตอร์ PTPER	ความละเอียดของ PWM	ความถี่สัญญาณ PWM
33 นาโนวินาที (30MHz)	0x7FFF	16 บิต	915kHz
33 นาโนวินาที (30MHz)	0x3FF	11 บิต	29.3kHz
50 นาโนวินาที (50MHz)	0x7FFF	16 บิต	610Hz
50 นาโนวินาที (50MHz)	0x1FF	10 บิต	39.1kHz
100 นาโนวินาที (10MHz)	0x7FFF	16 บิต	305Hz
100 นาโนวินาที (10MHz)	0xFF	9 บิต	39.1kHz
200 นาโนวินาที (5MHz)	0x7FFF	16 บิต	153Hz
200 นาโนวินาที (5MHz)	0x7F	8 บิต	39.1kHz

หมายเหตุ : ความถี่ของสัญญาณ PWM จะเป็นครึ่งหนึ่งของค่าที่แสดงนี้เมื่อทำงานในแบบปรับแต่งสัญญาณกึ่งกลาง

ตารางที่ 3.2 ความสัมพันธ์ระหว่างความถี่สัญญาณนาฬิกาในการทำงาน, ค่าของรีจิสเตอร์PTPER, ความละเอียดและความถี่ของสัญญาณ PWM เมื่ออัตราปรีสเกลเป็น 1:1

ในตารางที่ 3.2 แสดงให้เห็นถึงความสัมพันธ์ของความถี่ในการทำงานของ dsPIC กับความละเอียดของสัญญาณ PWM โดยค่าของความถี่ที่แสดงในตารางนั้นเป็นค่าที่เกิดขึ้นเมื่อกำหนดให้ส่วนกำเนิดสัญญาณ PWM ทำงานในโหมดปรับขอบสัญญาณ (Edge-aligned mode) และค่าความถี่จะลดลงครึ่งหนึ่งเมื่อทำงานในโหมดปรับสัญญาณกึ่งกลาง (Center aligned mode)

ในโมดูล MCPWM สามารถสร้างสัญญาณ PWM ที่มีความละเอียดเท่ากับ $T_{CY}/2$ ได้ โดยค่าของรีจิสเตอร์ PTMR จะเพิ่มขึ้นทุกๆคาบเวลา T_{CY} ที่อัตราปรีสเกลเลอร์เป็น 1:1 เพื่อให้ได้ความละเอียด $T_{CY}/2$ นี้ ค่าของรีจิสเตอร์ PDCx บิต 15 ถึง 1 จะถูกเปรียบเทียบกับค่ารีจิสเตอร์ PTMR บิต 14 ถึง 0 โดยบิต 0 ของรีจิสเตอร์ PDCx จะถูกใช้กำหนดขอบสัญญาณเริ่มต้นของสัญญาณ PWM แต่ถ้าหากที่ฐานเวลา PWM เลือกอัตราปรีสเกลเลอร์เป็นค่าอื่น ค่าบิต 0 ของรีจิสเตอร์ PDCx จะถูกเปรียบเทียบกับบิตนัยสำคัญสูงสุดของตัวนับในปรีสเกลเลอร์แทน เพื่อกำหนดขอบสัญญาณเริ่มต้นของสัญญาณ PWM ดังแสดงกระบวนการเปรียบเทียบในรูปที่ 3.12



รูปที่ 3.12 กระบวนการเปรียบเทียบข้อมูลเพื่อกำหนดค่าดิวตี้ไซเกิลของสัญญาณ PWM ในโมดูลMCPWM

3.2.5 โหมดการทำงานของส่วนกำเนิดสัญญาณ PWM ในโมดูล MCPWM

มีด้วยกัน 4 แบบหลักคือ

1.โหมดปรับของสัญญาณ (Edge aligned mode)

2.โหมดสัญญาณเดี่ยว (Single event mode)

3.โหมดปรับสัญญาณกึ่งกลาง (Center aligned mode)

4.โหมดปรับสัญญาณกึ่งกลางพร้อมปรับปรุงค่า (Center aligned mode with double updates)

ซึ่งจะเพิ่มสัมพันธ์กับโหมการทำงานของฐานเวลา PWM โดย

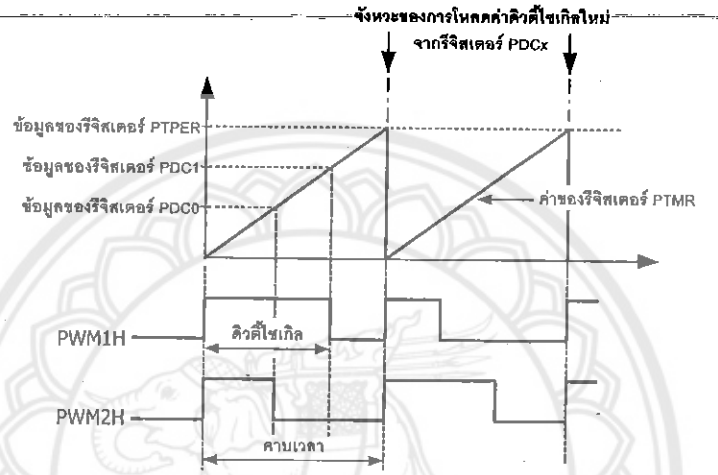
-เมื่อฐานเวลา PWM ทำงานในโหมดเปลี่ยนแปลงค่าอิสระ ส่วนกำเนิดสัญญาณ PWM จะทำงานในโหมดปรับขอบสัญญาณ

-เมื่อฐานเวลา PWM ทำงานในโหมดทำงานครั้งเดียว ส่วนกำเนิดสัญญาณ PWM จะทำงานในโหมดสัญญาณเดี่ยว

-เมื่อฐานเวลา PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่อง ส่วนกำเนิดสัญญาณ PWM จะทำงานในโหมดปรับสัญญาณกึ่งกลาง

-เมื่อฐานเวลา PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่องพร้อมปรับปรุงค่า ส่วนกำเนิดสัญญาณ PWM จะทำงานในโหมดปรับสัญญาณกึ่งกลางพร้อมปรับปรุงค่า

การทำงานของส่วนกำเนิดสัญญาณ PWM ในโหมดปรับขอบสัญญาณ



รูปที่ 3.13 ไดอะแกรมเวลาแสดงการเกิดสัญญาณ PWM เมื่อทำงานในโหมดปรับขอบสัญญาณ

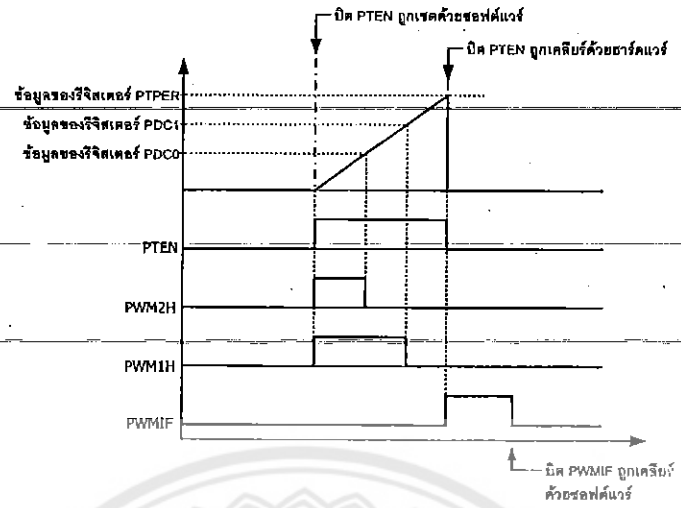
เริ่มต้นด้วยการกำหนดให้ฐานเวลา PWM ทำงานในโหมดเปลี่ยนแปลงค่าอิสระ ดังนั้นคาบเวลาของสัญญาณ PWM จะถูกกำหนดโดยค่าที่โหลดให้แก่รีจิสเตอร์ PTPER ส่วนดิวตี้ไซเกิลถูกกำหนดค่าในรีจิสเตอร์ PDCx

ในกรณีที่ค่าดิวตี้ไซเกิลไม่เป็นศูนย์ วงจรเอาต์พุตของส่วนกำเนิดสัญญาณ PWM ทุกชุดที่ได้รับการเอนเอเบิลจะทำงานที่จุดเริ่มต้นของคาบเวลาของสัญญาณ PWM หรือเมื่อค่าในรีจิสเตอร์ PTMR เท่ากับ 0 และหยุดทำงานเมื่อค่ารีจิสเตอร์ PTMR ตรงกับค่าดิวตี้ไซเกิลของส่วนกำเนิดสัญญาณ PWM

ถ้าหากค่าในรีจิสเตอร์ PDCx เป็นศูนย์ วงจรเอาต์พุตของส่วนกำเนิดสัญญาณ PWM จะไม่ทำงานใดๆ นั่นหมายความว่าในโหมดนี้จะสามารถกำเนิดสัญญาณ PWM ได้ก็ต่อเมื่อค่าในรีจิสเตอร์ PDCx ต้องมากกว่าค่าที่กำหนดในรีจิสเตอร์ PTPER

การทำงานของส่วนกำเนิดสัญญาณ PWM ในโหมดสัญญาณเดี่ยว

ในโมดูล MCPWM จะกำเนิดสัญญาณพัลส์ออกมาเพียงลูกเดียวหรือพัลส์เดี่ยวเมื่อฐานเวลาPWM ถูกกำหนดให้ทำงานในโหมดทำงานครั้งเดียว (ค่าของ PTMOD1 และ PTMOD0 เท่ากับ “0”)



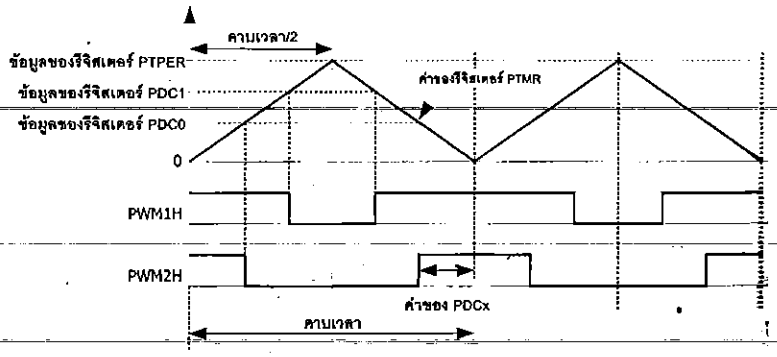
รูปที่ 3.14 ไดอะแกรมเวลาแสดงการเกิดสัญญาณ PWM เมื่อทำงานในโหมดสัญญาณเดี่ยว

ในรูปที่ 3.14 แสดงไดอะแกรมเวลาของการกำเนิดสัญญาณ PWM ในโหมดนี้ โดยการกำเนิดสัญญาณในลักษณะนี้จะถูกนำไปใช้ขับสวิตช์รีลักแตนซ์มอเตอร์ความเร็วสูง โหมดนี้จะเริ่มต้นการทำงานเมื่อ บิต PTEN ถูกเซตเพื่อเอ็นเอเบิลการทำงานของฐานเวลา PWM ลงจรเอาต์พุตของโมดูล MCPWM จะทำงานทันที

เมื่อค่าในรีจิสเตอร์ PTMR ตรงกับค่าดิวตี้ไซเกิลในรีจิสเตอร์ PDCx วงจรเอาต์พุตจะหยุดทำงานเมื่อค่าในรีจิสเตอร์ PTMR เท่ากับค่าในรีจิสเตอร์ PTPER ค่าในรีจิสเตอร์ PTMR จะถูกเคลียร์วงจร เอาต์พุตของโมดูล MCPWM จะหยุดทำงานทั้งหมด และเกิดการอินเตอร์รัพต์ขึ้นถ้ามีการเอ็นเอเบิลไว้หลังจากนั้นส่วนกำเนิดสัญญาณ PWM จะหยุดทำงาน จนกว่าบิต PTEN จะถูกเซตด้วยซอฟต์แวร์อีกครั้ง

การทำงานของส่วนกำเนิดสัญญาณ PWM ในโหมดปรับกึ่งกลางสัญญาณ

โมดูล MCPWM จะกำเนิดสัญญาณพัลส์ PWM ในโหมดนี้เมื่อฐานเวลา PWM ถูกกำหนดให้ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่อง (ค่าของ PTMOD1 และ PTMOD0 เท่ากับ “10” หรือ “11”)



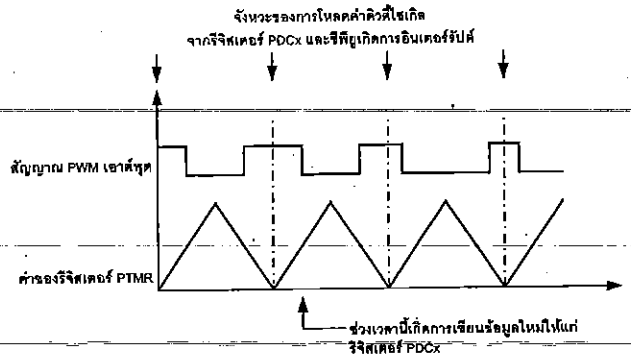
รูปที่ 3.15 ไดอะแกรมเวลาของการกำเนิดสัญญาณ PWM ในโหมดปรับสัญญาณกึ่งกลาง

ในรูปที่ 3.15 แสดงไดอะแกรมเวลาของการกำเนิดสัญญาณ PWM ในโหมดนี้ วงจรเอาต์พุตของโมดูล MCPWM จะอยู่ในสภาวะทำงานเมื่อค่าของรีจิสเตอร์ PDCx เท่ากับค่าของ PTMR และฐานเวลา PWM อยู่ในภาวะนับลง (บิต PTDIR เป็น "1") วงจรเอาต์พุตของโมดูล MCPWM จะอยู่ในสภาวะหยุดทำงานเมื่อฐานเวลา PWM อยู่ในภาวะนับขึ้น (บิต PTDIR เป็น "0") และค่าของรีจิสเตอร์ PTMR เท่ากับค่าของรีจิสเตอร์กำหนดค่าดิวตี้ไซเกิล

ถ้าหากค่าในรีจิสเตอร์ PDCx เป็นศูนย์ วงจรเอาต์พุตส่วนกำเนิดสัญญาณ PWM จะไม่ทำงานใดๆ นั่นหมายความว่า ในโหมดนี้จะสามารถกำเนิดสัญญาณ PWM ได้ก็ต่อเมื่อค่าในรีจิสเตอร์ PDCx ต้องมากกว่าค่าที่กำหนดในรีจิสเตอร์ PTPER

3.2.6 การเปลี่ยนค่าดิวตี้ไซเกิลสัญญาณ PWM ของโมดูล MCPWM

รีจิสเตอร์กำหนดค่าดิวตี้ไซเกิลทั้ง 4 ตัว PDC1 ถึง PDC4 ต่างก็มีรีจิสเตอร์บัฟเฟอร์เพื่อป้องกันสัญญาณรบกวนเมื่อมีการปรับปรุงค่าของสัญญาณ PWM โดยดิวตี้ไซเกิลของสัญญาณ PWM จะถูกปรับปรุงค่าตามข้อมูลที่เขียนลงในรีจิสเตอร์ PDCx จากนั้นค่าจากรีจิสเตอร์ PDCx จะถูกโหลดไปยังบัฟเฟอร์เพื่อทำการเปรียบเทียบ เมื่อมีการเปลี่ยนแปลงข้อมูลในรีจิสเตอร์ PDCx เรียบร้อย ข้อมูลนั้นจึงจะถูกส่งไปยังบัฟเฟอร์เพื่อทำงานต่อไป ทำให้ไม่เกิดการติดขัดหรือเกิดการผิดพลาดในขณะเปลี่ยนค่าดิวตี้ไซเกิล

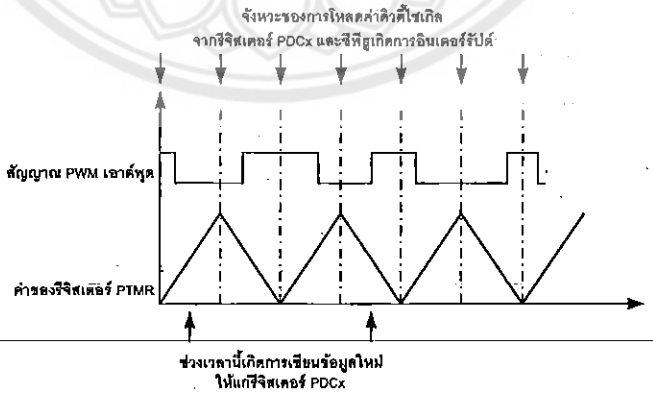


รูปที่ 3.16 ไดอะแกรมเวลาแสดงการเปลี่ยนค่าดิวตี้ไซเกิลในการกำเนิดสัญญาณ PWM

เมื่อฐานเวลา PWM ทำงานในโหมดเปลี่ยนแปลงค่าอิสระหรือโหมดทำงานครั้งเดียว (ค่าของ PTMOD1 และ PTMOD0 เท่ากับ “0x”) ค่าดิวตี้ไซเกิลของสัญญาณ PWM จะถูกปรับปรุงเมื่อค่าของรีจิสเตอร์ PTMR เท่ากับ PTPER และเมื่อรีจิสเตอร์ PTMR เกิดการรีเซตเป็นศูนย์

เมื่อฐานเวลา PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่อง (ค่าของ PTMOD1 และ PTMOD0 เท่ากับ “10”) ค่าดิวตี้ไซเกิลของสัญญาณ PWM จะถูกปรับปรุงเมื่อค่าของรีจิสเตอร์ PTMR เป็นศูนย์ และฐานเวลา PWM นับค่าขึ้น

เมื่อฐานเวลา PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่องพร้อมปรับปรุงค่า (ค่าของ PTMOD1 และ PTMOD0 เท่ากับ “11”) ค่าดิวตี้ไซเกิลของสัญญาณ PWM จะถูกปรับปรุงเมื่อค่าของรีจิสเตอร์ PTMR เป็นศูนย์ และค่าของรีจิสเตอร์ PTPER



รูปที่ 3.17 ไดอะแกรมเวลาแสดงการเปลี่ยนค่าดิวตี้ไซเกิลเมื่อฐานเวลา PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่องพร้อมปรับปรุงค่า

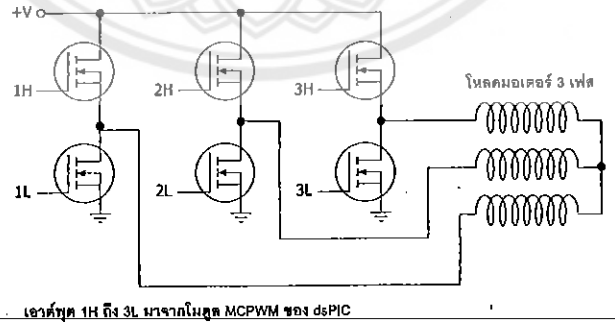
อย่างไรก็ตาม การเขียนข้อมูลไปยังรีจิสเตอร์ PDCx จะปรับปรุงค่าดิวตี้ไซเกิลในทันที ถ้าฐานเวลา PWM ถูกดิสเอเบิลอยู่ (บิต PTEN เป็น "0") ดังนั้นเมื่อเอ็นเอเบิลให้ส่วนฐานเวลา PWM ทำงานค่าดิวตี้ไซเกิลของสัญญาณ PWM จำถูกเปลี่ยนเป็นค่าใหม่ตามไปด้วย

นอกนั้นในโมดูล MCPWM ยังมีความสามารถพิเศษอีกประการหนึ่งคือ การป้องกันการเปลี่ยนค่าดิวตี้ไซเกิล ทำได้โดยการเซตบิต UDIS (PWM Update Disable bit) ซึ่งเป็นบิต 2 ของรีจิสเตอร์ PWMCON2 โดยมีขั้นตอนดังนี้

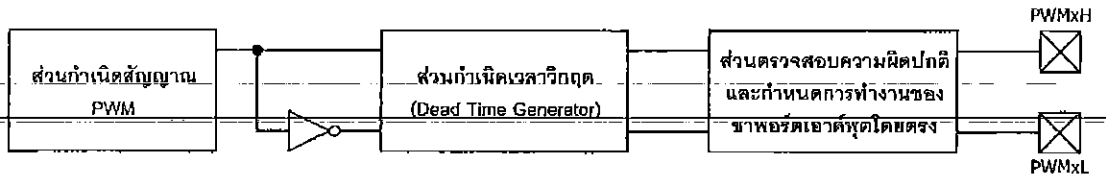
1. เซตบิต UDIS
2. เขียนข้อมูลที่ต้องการและค่าดิวตี้ไซเกิลในรีจิสเตอร์ PTPER และ PDCx
3. หลังจากนั้นจะไม่สามารถเปลี่ยนแปลงค่าดิวตี้ไซเกิลจนกว่าจะเคลียร์บิต UDIS

3.2.7 การทำงานร่วมกันของส่วนกำเนิดสัญญาณ PWM หรือการทำงานในแบบคอมพลีเมนตารี (Complementary PWM Output Mode)

การทำงานในลักษณะนี้ของโมดูล MCPWM เหมาะสำหรับการนำไปสร้างสัญญาณเพื่อขับโหลดแบบอินเวอร์เตอร์ดังตัวอย่างในรูปที่ 3.18 ตัวอย่างของโหลดแบบอินเวอร์เตอร์ได้แก่ มอเตอร์อินดักชั่นไฟสลับ 3 เฟส (ACIM : AC Induction Motor) และมอเตอร์แบบไม่มีแปลงถ่าน (BLDC : Brushless DC motor) ในการทำงานแบบคอมพลีเมนตารีนี้ วงจรเอาต์พุตของโมดูล PWM ในคู่ที่นำมาใช้งานนั้นไม่สามารถกำหนดให้ทำงานได้พร้อมกัน นั่นคือ เอาต์พุต PWMxH และ PWMxL ต้องมีสภาวะที่ตรงกันข้าม ทำให้มอสเฟตที่ต่ออยู่กับขาเอาต์พุตนั้นสลับกันทำงาน โดยกระบวนการทำงานภายในของโมดูล PWM เมื่อทำงานในแบบคอมพลีเมนตารีแสดงด้วยไดอะแกรมในรูปที่ 2.33



รูปที่ 3.18 วงจรตัวอย่างในการนำขาเอาต์พุตของโมดูล MCPWM ใน dsPIC มาทำงานร่วมกันในแบบคอมพลีเมนตารี



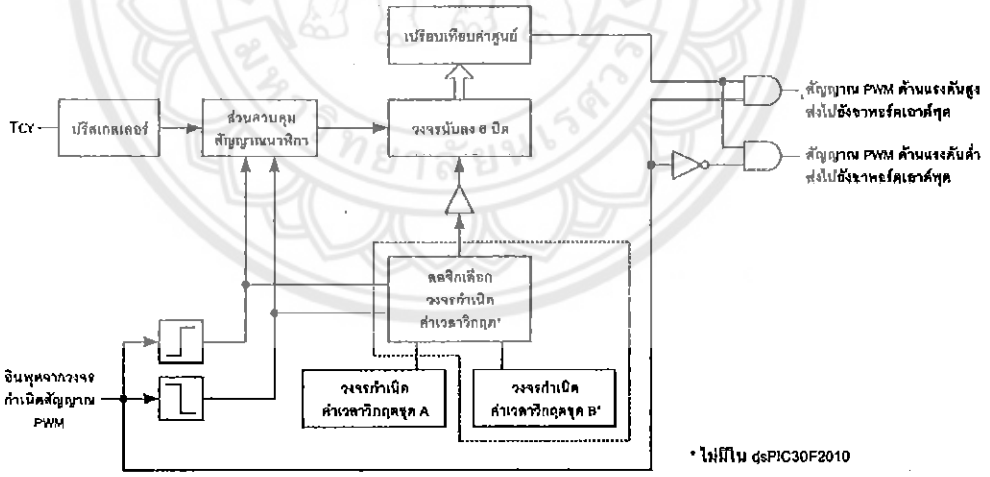
รูปที่ 3.19 ไดอะแกรมการทำงานเบื้องต้นของโมดูล MCPWM เมื่อทำงานในแบบคอมพลีเมนตารี

จะเห็นได้ว่า มีส่วนประกอบที่สำคัญเพิ่มขึ้นมา 2 ส่วนคือ ส่วนกำเนิดของช่วงเวลาวิกฤต (Dead Time Generator) และส่วนตรวจสอบความผิดปกติและกำหนดการทำงานของขาพอร์ตเอาต์พุตโดยตรง (Override and Fault Logic) ซึ่งได้อธิบายต่อไป

ในกำหนดให้โมดูล MCPWM อย่างไรก็ตาม เมื่อเกิดการรีเซต โมดูล MCPWM จะถูกกำหนดให้ทำงานในแบบคอมพลีเมนตารีเป็นค่าตั้งต้น ดังนั้นหากต้องการให้ทำงานในแบบอิสระจะต้องมีการเขียนโปรแกรมเพื่อกำหนดรูปแบบการทำงานใหม่เสมอ

3.2.8 การควบคุมเวลาวิกฤต (Dead Time Control)

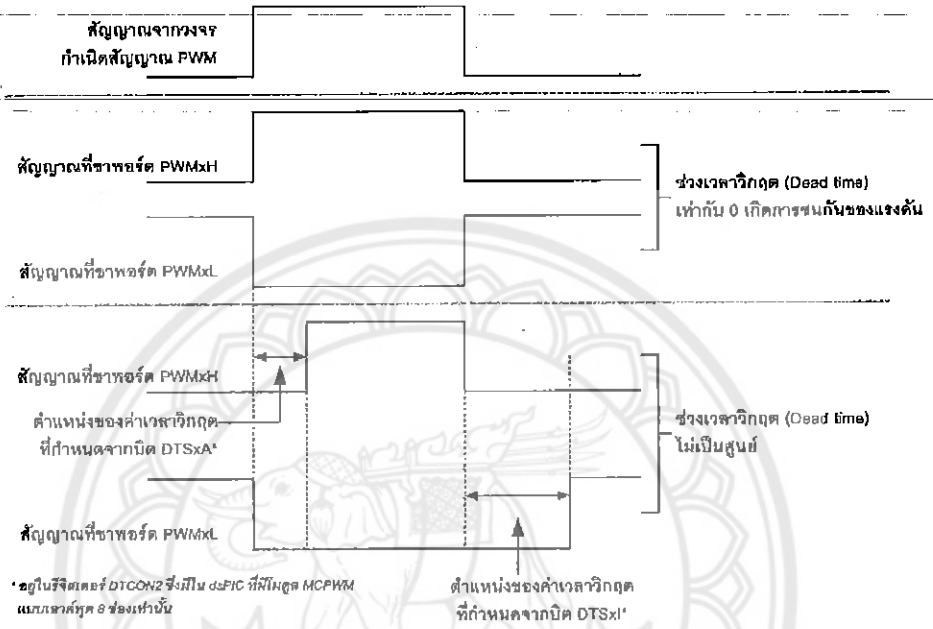
ส่วนกำเนิดช่วงเวลาวิกฤต



รูปที่ 3.20 ไดอะแกรมของการทำงานของส่วนกำเนิดช่วงเวลาวิกฤตของวงจรเอาต์พุตของโมดูล MCPWM

ในรูปที่ 3.21 เป็นไดอะแกรมการทำงานของส่วนกำเนิดช่วงเวลาวิกฤตของแต่ละคู่วงจรเอาต์พุตในโมดูล MCPWM มีวงจรนับลง 6 บิต ซึ่งถูกใช้กำเนิดค่าเวลาวิกฤต โดยมีส่วนตรวจจับขอบขาขึ้นและลงของ

สัญญาณเชื่อมต่อเข้ากับเอาต์พุตของส่วนเปรียบเพื่อกำหนดค่าดิวตี้ไซเกิล ค่าเวลาวิกฤตจะถูกโหลดไปยังวงจรนับลงเมื่อตรวจจับขอบขาของสัญญาณ PWM ได้ จะเริ่มการนับค่าลงจนเป็น "0" เอาต์พุตของโมดูล PWM จึงมีการเปลี่ยนแปลงสถานะ



รูปที่ 3.21 ไดอะแกรมเวลาของการกำหนดช่วงเวลาวิกฤตลงในเอาต์พุตของโมดูล MCPWM โดยค่าเวลาวิกฤตค่าที่สองจะเกิดขึ้นได้ใน dsPIC ที่มีโมดูล MCPWM แบบ 8 เอาต์พุตเท่านั้น

ใน dsPIC30F2010 ซึ่งมีโมดูล MCPWM แบบ 6 ช่อง จึงมีส่วนกำเนิดช่วงเวลาวิกฤตเพียงชุดเดียว คือชุด A ค่าของเวลาวิกฤตที่ต้องการจึงสามารถกำหนดลงในบิต DPA5 ถึง DTA0 (บิต 5 ถึง 0) ของรีจิสเตอร์ DTCON1 ส่วนอัตราปรีสเกลเลอร์สามารถกำหนดได้ที่บิต DTAPS1 และ DTAPS (บิต 7 และ6) ของรีจิสเตอร์ DTCON1 โดยกำหนดได้ตั้งแต่ TCY ถึง 8TCY

ย่านของเวลาวิกฤต

ค่าเวลา T_{CY} (ความถี่ F_{CY})	อัตราของปรีสเกลเลอร์	ความละเอียด	ย่านของเวลาหน่วงเฟส
33 นาโนวินาที (30MHz)	$4T_{CY}$	130 นาโนวินาที	9-130 ไมโครวินาที
50 นาโนวินาที (20MHz)	$4T_{CY}$	200 นาโนวินาที	12-200 ไมโครวินาที
100 นาโนวินาที (10MHz)	$2T_{CY}$	200 นาโนวินาที	12-200 ไมโครวินาที

ตารางที่ 3.3 ตัวอย่างของย่านเวลาวิกฤตความถี่ในการทำงานค่าต่างๆและอัตราปรีสเกลเลอร์เท่ากับ $2T_{CY}$ และ $4T_{CY}$

ค่าเวลาสามารถกำหนดได้จากอัตราปรีสเกลเลอร์ร่วมกับข้อมูล 6 บิตที่กำหนดลงในวงจรนับของส่วนกำเนิดเวลาวิกฤต (ดูรูปที่ 3.21 ประกอบ) ซึ่งก็คือค่าของบิต DPA5 ถึง DTA0 (บิต 5 ถึง 0) ของรีจิสเตอร์ DTCON1 ในกรณีใช้ส่วนกำเนิดเวลาวิกฤตชุด A ในกรณีใช้ส่วนกำเนิดเวลาวิกฤต ชุด B ค่าเวลาวิกฤตจะกำหนดได้จากค่าของบิต DTB5 ถึง DTB0 (บิต 5 ถึง 0) ของรีจิสเตอร์ DTCON2และอัตราปรีสเกลเลอร์ที่บิต DTBPS1 และ DTBPS0 (บิต 15 และบิต 14) ของรีจิสเตอร์ DTCON1ข้อมูล 6 บิตที่ใช้ในการกำหนดค่าเวลาวิกฤต (DT) สามารถคำนวณได้จากสมการนี้

$$DT = \frac{Dead\_Time}{\Pr escaler \times T_{CY}}$$

3.2.9 การควบคุมขั้วทางเอาต์พุตของโมดูล MCPWM

การควบคุมขาพอร์ตเอาต์พุตของโมดูล MCPWM ให้ทำงานเป็นขาพอร์ตสำหรับจ่ายสัญญาณ PWM ต้องกระทำผ่านบิต PENxx ในรีจิสเตอร์ PWMCON1 และเมื่อกำหนดให้ทำงานกับโมดูล MCPWM แล้ว การควบคุมจากรีจิสเตอร์ PORT และ PRIS จะถูกดิสเอเบิลไปโดยอัตโนมัติ

นอกจากนั้นยังสามารถกำหนดขั้วแรงดันที่ขาพอร์ตเอาต์พุตได้ด้วย เดยกระทำผ่านบิต HPOL, LPOL และ PWMPIN ในรีจิสเตอร์ FBORPOR โดยเริ่มจากการเคลียร์บิต PWMPIN เพื่อกำหนดให้ขาพอร์ตของโมดูล MCPWM เป็นขาเอาต์พุตสำหรับส่งสัญญาณ PWM เพื่อขับมอเตอร์

ส่วนบิต HPOL (บิต 9 ของรีจิสเตอร์ FBORPOR) ใช้กำหนดขั้วของขาเอาต์พุตของโมดูล MCPWM ด้านแรงดันสูง ปกติแล้วกำหนดเป็น "1" นั่นคือ กำหนดให้ขาเอาต์พุตด้านแรงดันสูงของโมดูล MCPWM หรือ PWMxH เป็นขั้วแรงดันสูง

ทางด้านบิต LPOL (บิต 8 ของรีจิสเตอร์ FBORPOR) ใช้กำหนดขั้วของขาเอาต์พุตของโมดูล MCPWM ด้านแรงดันต่ำ หรือขา PWMxL โดยปกติมีค่าเป็น "0" เพื่อกำหนดให้ขา PWMxL ของโมดูล MCPWM เป็นขั้วแรงดันต่ำ

3.2.10 อินพุตตรวจสอบความผิดปกติของโมดูล MCPWM

ในโมดูล MCPWM ของ dsPIC มีอินพุตสำหรับรับสัญญาณตรวจสอบความผิดปกติสูงสุด 2 ช่องคือ $\overline{FLTA}$ และ $\overline{FLTB}$ สำหรับใน dsPIC30F2010 มีเพียงช่องเดียวคือ $\overline{FLTA}$ โดยขาอินพุตนี้ทำงานด้วยลอจิก "0" ดังนั้นเมื่อนำมาใช้งานควรต่อตัวต้านทานพูลอัปด้วย การใช้งานขาอินพุตตรวจสอบความผิดปกตินี้ควรกระทำด้วยกระบวนการอินเตอร์รัปต์ เนื่องจากใน dsPIC ได้เตรียมอินเตอร์รัปต์เวกเตอร์บิตแฟลกแจ้งการเกิดอินเตอร์รัปต์ บิตเอ็นเอเบิลการอินเตอร์รัปต์ และบิตกำหนดระดับความสำคัญของอินเตอร์รัปต์แบบนี้ไว้อย่างพร้อมสรรพ จึงให้ผู้พัฒนาโปรแกรมควบคุมไม่จำเป็นต้องเขียนโปรแกรมเพื่อวนตรวจสอบ

การทำงานของขาอินพุตตรวจสอบความผิดปกติ $\overline{FLTA}$ ถูกควบคุมด้วยรีจิสเตอร์ FLTACON ในขณะที่ขาอินพุตตรวจสอบความผิดปกติ $\overline{FLTB}$ จะถูกควบคุมด้วยรีจิสเตอร์ FLTBCON

สำหรับการอธิบายในหัวข้อนี้จะขออธิบายเพียงช่องเดียวคือ อินพุต $\overline{FLTA}$

บิตเอ็นเอเบิลการตรวจสอบความผิดปกติ

ในรีจิสเตอร์ FLTACON มีบิตสำหรับเอ็นเอเบิลการตรวจสอบความผิดปกตินี้ 4 บิต คือ FAEN1ถึง FAEN4 (ถ้าเป็นรีจิสเตอร์ FLTBCON จะเป็นบิต FBEN1 ถึง FBEN4) ทำงานด้วยลอจิก "1" โดยแต่ละบิตจะตรวจสอบการทำงานของโมดูล PWM แต่ละชุดแยกจากกัน นั่นคือบิต FAEN1 ใช้เอ็นเอเบิลการตรวจสอบความผิดปกของโมดูล PWM1 เป็นต้น ส่วน FAEN4 ไม่ใช้งานใน dsPIC30F2010

การกำหนดผลหลังจากตรวจสอบพบความผิดปกติ

ในรีจิสเตอร์ FLTACON และ FLTBCON มีบิตควบคุมอยู่ 8 ที่ใช้ในการกำหนดการทำงานของขาเอาต์พุตของโมดูล MCPWM เมื่อตรวจสอบพบความผิดปกติที่อินพุต $\overline{FLTA}$ และ $\overline{FLTB}$ นั่นคือบิต FAOVxH และ FAOVxL (x คือหมายเลขของโมดูล PWM) โดยจับเป็นคู่เรียงลำดับจาก FAOV4H กับ FAOV4L (บิต 15 และ 14) ไปจนถึง FAOV1H กับ FAOV1L (บิต 9 และ 8) สำหรับใน dsPIC30F2010 ไม่มีการใช้งานบิต FAOV4H กับ FAOV4L ต้องกำหนดเป็น "0"

การกำหนดให้ขาพอร์ตเอาต์พุตของโมดูล MCPWM ทำงานเมื่ออินพุต $\overline{FLTA}$ แอกตีฟหรือตรวจสอบพบความผิดพลาดขึ้น ทำได้ 2 แบบคือ

1. ขาเอาต์พุตของโมดูล MCPWM เป็นสถานะ "ไม่แอกตีฟ" หรือเป็นสัญญาณแรงดันต่ำ เมื่อเกิดตรวจจับความผิดปกติได้ที่ขา $\overline{FLTA}$

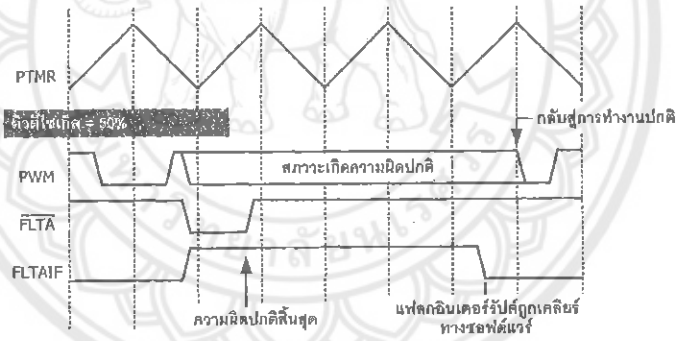
2. ขาเอาต์พุตของโมดูล โมดูล MCPWM เป็นสถานะ "แอกตีฟ" หรือเป็นสัญญาณแรงดันสูง เมื่อเกิดตรวจจับความผิดปกติได้ที่ขา $\overline{FLTA}$

ในการกำหนดลักษณะทำงานของขาเอาต์พุตคู่ (PWMxH และ PWMxL) จะต้องแตกต่างกันและต้องคำนึงถึงการกำหนดขั้วของแรงดันจากบิต HPOL และ LPOL ในรีจิสเตอร์ FBORPOR ในตอนเริ่มต้อนการทำงานหรือในภาวการณ์ทำงานปกติด้วย นั่นหมายความว่า สถานการณ์ทำงานของขาพอร์ตเอาต์พุตของโมดูล MCPWM ควรเปลี่ยนไปเมื่อตรวจจับพบความผิดปกติที่อินพุต $\overline{FLTA}$ หรือ $\overline{FLTB}$

โหมดการทำงานเมื่อตรวจสอบพบความผิดปกติ

A. โหมดแลตซ์ (Latched mode)

เมื่ออินพุต $\overline{FLTA}$ เกิดลอจิก "0" (นั่นคือ ตรวจสอบพบความผิดปกติ) ขาเอาต์พุตของโมดูล MCPWM จะเปลี่ยนสถานการณ์ทำงานตามที่กำหนดไว้ในรีจิสเตอร์ FLTACON และทำงานอยู่ในสถานะที่เปลี่ยนมานี้จนกว่าขาอินพุต $\overline{FLTA}$ จะกลับไปเป็นลอจิก "1" และบิตแฟลกอินเตอร์รัปต์ FLTAIF จะถูกเคลียร์ด้วยกระบวนการทางซอฟต์แวร์ เมื่อเงื่อนไขทั้งสองเป็นจริง โมดูล MCPWM จะกลับไปทำงานตามปกติที่คาบเวลาถัดไปของฐานเวลาPWM

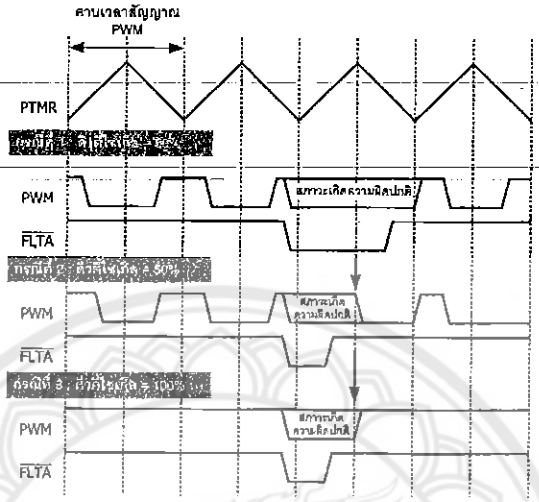


รูปที่ 3.22 ไดอะแกรมเวลาที่แสดงให้เห็นถึงการทำงานของโมดูล MCPWM เมื่อมีการตรวจสอบพบความผิดปกติที่อินพุต $\overline{FLTA}$ ในโหมดแลตซ์

B. โหมดไซเกิลต่อไซเกิล (Cycle-by-Cycle mode)

เมื่ออินพุต FLTA เกิดลอจิก "0" (นั่นคือ ตรวจสอบพบความผิดปกติ) ขาเอาต์พุตของโมดูล MCPWM จะเปลี่ยนสถานการณ์ทำงานตามที่กำหนดไว้ในรีจิสเตอร์ FLTACON และทำงานอยู่ในสถานะที่เปลี่ยนมานี้จนกว่าขาอินพุต $\overline{FLTA}$ จะกลับไปเป็นลอจิก "1" โมดูล MCPWM จะกลับไปทำงานเมื่อเริ่มต้นคาบเวลาถัดไปของฐานเวลา PWM หรือที่จุดกึ่งกลางของคาบเวลาในโหมดปรับสัญญาณกึ่งกลาง ดังนั้นในโหมดนี้

จะไม่สนใจการเคลียร์บิตแฟลก FLTAIF เพียงขาอินพุต $\overline{FLTA}$ กลับไปสู่ลอจิก "1" หรือการตรวจจับความผิดปกติสิ้นสุดลง โมดูล MCPWM ก็จะกลับไปทำงานตามปกติทันที



รูปที่ 3.23 ไดอะแกรมเวลาที่แสดงให้เห็นถึงการทำงานของโมดูล MCPWM เมื่อมีการตรวจสอบพบความผิดปกติที่อินพุต FLTA ในโหมดไซเกิลต่อไซเกิล

การเลือกโหมดทำงานนี้กำหนดได้ที่บิต FLTAM (บิต 7) ในรีจิสเตอร์ FLTBCON ในกรณีใช้งานอินพุตตรวจสอบความผิดปกติชุด B

การทำให้โมดูล MCPWM ออกจากโหมดการทำงานในสภาวะผิดปกติกลับไปทำงานตามปกติมีอีก 1 วิธี คือ เคลียร์บิต PTEN ส่งผลให้เกิดการดิสเอเบิลการทำงานของฐานเวลา PWM ในโมดูล MCPWM ลงกระบวนการทำงานภายในโมดูล MCPWM จำทำการเรียกค่าตั้งต้นการทำงานกลับมาใหม่ในทันทีทำให้สภาวะผิดปกตินั้นถูกยกเลิกไป โมดูล MCPWM ก็จะสามารถกลับมาทำงานตามปกติได้

ในกรณีที่ใช้งานขาอินพุต $\overline{FLTA}$ และ $\overline{FLTB}$ และเกิดเป็นลอจิก "0" พร้อมกัน โมดูล MCPWM จะเลือกทำงานตามที่กำหนดจากอินพุต FLTA ก่อน นั่นคือ อินพุต FLTA มีระดับความสำคัญสูงกว่าอินพุต FLTB และเมื่อเงื่อนไขของความผิดปกติที่อินพุต $\overline{FLTA}$ สิ้นสุดลง แต่ที่อินพุต $\overline{FLTB}$ ยังเป็นลอจิก"0" โมดูล MCPWM ก็จะถูกกำหนดให้ทำงานตามที่กำหนดในรีจิสเตอร์ FLTBCON ต่อไป

3.2.11 การกำเนิดสัญญาณกระตุ้นพิเศษของโมดูล MCPWM

อีกหนึ่งความสามารถของโมดูล MCPWM คือ กำเนิดสัญญาณกระตุ้นแก่โมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC) เพื่อให้ทำงานสัมพันธ์กับฐานเวลา PWM จึงทำให้การสุ่มสัญญาณและเวลาในการแปลง

สัญญาณสามารถที่จะกำหนดให้เกิดขึ้นในช่วงเวลาใดๆภายในคาบเวลาของสัญญาณ PWM สัญญาณกระตุ้นพิเศษนี้จะช่วยให้สามารถลดค่าเวลาหน่วงของการแปลงสัญญาณในโมดูล ADC ลงได้

การทำงานในส่วนนี้ใช้รีจิสเตอร์ SEVTCMP เป็นหลัก โดยมีบิตกำหนดอัตราโพสต์สเกลเลอร์ 4 บิตคือ SEVOPS3 ถึง SEVOPS0 (บิต 11 ถึง 8 ในรีจิสเตอร์ PWMCON2) ใช้ในการควบคุมร่วมกัน ค่าในรีจิสเตอร์ PTMR 15 บิตจะถูกโหลดไปยังรีจิสเตอร์ SEVTCMP ในบิต 14 ถึง 0

เมื่อฐานเวลา PWM ทำงานในโหมดนับค่าขึ้นลงอย่างต่อเนื่อง บิต SEVTDIR ซึ่งเป็นบิตกำหนดการเกิดสัญญาณกระตุ้นพิเศษ ในรีจิสเตอร์ SEVTCMP จะถูกนำมาใช้กำหนดจังหวะในการสร้างสัญญาณกระตุ้นพิเศษ โดยถ้าบิตนี้เป็น "0" สัญญาณกระตุ้นพิเศษจะเกิดขึ้นเมื่อฐานเวลา PWM นับค่าขึ้น และถ้าเป็น "1" สัญญาณกระตุ้นพิเศษจะเกิดขึ้นเมื่อฐานเวลา PWM นับค่าลง

สัญญาณกระตุ้นที่มีโพสต์สเกลเลอร์เพื่อปรับอัตราของสัญญาณได้ 16 ระดับในกรณีที่โมดูลADC ไม่ต้องการให้ทำงานสัมพันธ์กับทุกไซเกิลของสัญญาณ PWM การกำหนดอัตราโพสต์สเกลเลอร์ทำได้โดยการกำหนดค่าที่บิต SEVOPS3 ถึง SEVOPS0 ในรีจิสเตอร์ PWMCON2 ค่าของโพสต์สเกลเลอร์จะถูกเคลียร์เมื่อเกิดการเขียนข้อมูลมายังรีจิสเตอร์ SEVTCMP และเมื่อไมโครคอนโทรลเลอร์เกิดการรีเซต

3.2.12 การทำงานของโมดูล MCPWM ในโหมดประหยัดพลังงาน

การทำงานของโมดูล MCPWM ในโหมดสลีป

เมื่อ dsPIC เข้าสู่โหมดสลีป ระบบสัญญาณนาฬิกาจะถูกกำหนดให้หยุดทำงาน ทำให้สัญญาณนาฬิกาสำหรับฐานเวลา PWM หยุดทำงานตามไปด้วย วงจรเอาต์พุตของโมดูล MCPWM ทั้งหมดหยุดทำงาน สิ่งที่ต้องระมัดระวังอย่างมากคือ ถ้าหากในขณะนั้นโมดูล MCPWM ถูกใช้ขับโหลดอยู่ ก่อนหน้าที่จะเข้าสู่โหมดสลีปต้องมีการเตรียมการเพื่อกำหนดให้โมดูล MCPWM อยู่ในสภาวะปลอดภัย วิธีการหนึ่งที่จะช่วยได้คือ กำหนดการทำงานโดยตรงที่ขาพอร์ตเอาต์พุตโดยใช้รีจิสเตอร์ OVDCON ซึ่งจะต้องเขียนข้อมูลเพื่อกำหนดให้ขาพอร์ตของโมดูล MCPWM หยุดทำงานทั้งหมด

การทำงานของโมดูล MCPWM ในโหมดไอเดิล

เมื่อ dsPIC เข้าสู่โหมดไอเดิล ระบบสัญญาณนาฬิกาหลักยังคงทำงานอยู่ แต่ซีพียูจะหยุดการกระทำคำสั่ง ส่วนโมดูล MCPWM สามารถเลือกให้หยุดหรือทำงานต่อไปในโหมดไอเดิลนี้ได้ หากต้องการให้ทำงานต่อไปต้องเคลียร์บิต PTSIDL ซึ่งเป็นบิต 13 ในรีจิสเตอร์ PTCON นอกจากนั้นการอินเตอร์รัพต์ทุกแบบที่เกิดขึ้นในโมดูล MCPWM จะช่วยให้ dsPIC สามารถเวก-อัปออกจากโหมดไอเดิลนี้ได้

บทที่ 4
ขั้นตอนออกแบบและการดำเนินงาน

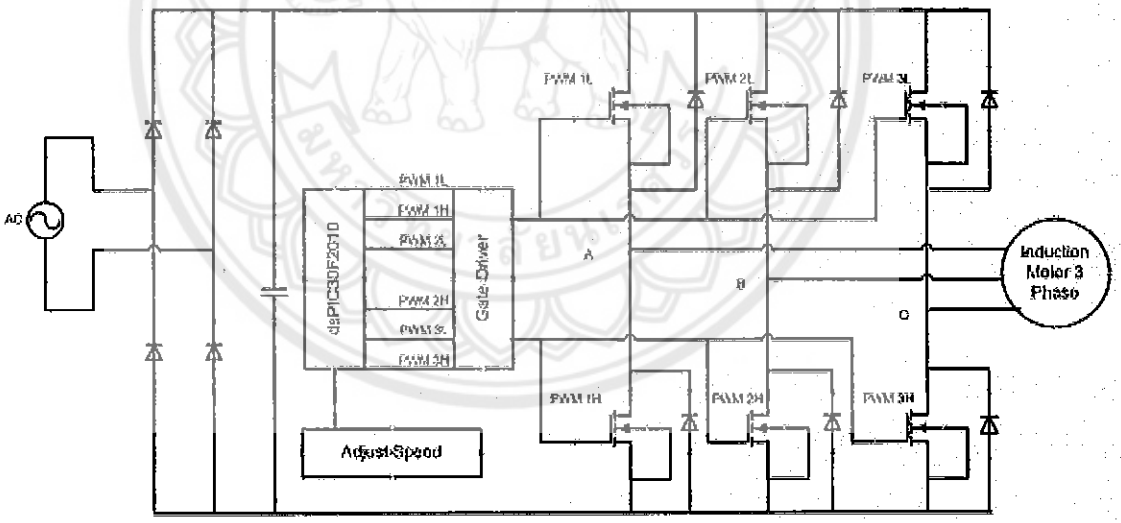
4.1 ขั้นตอนและการศึกษาข้อมูล

การศึกษาข้อมูลและทฤษฎีที่เกี่ยวข้องในการสร้างอินเวอร์เตอร์ 3 เฟส เพื่อควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำสามารถแบ่งส่วนการทำงานออกเป็น 3 ส่วนใหญ่ๆ คือ

1. ส่วนควบคุมความเร็วรอบมอเตอร์ประกอบด้วยวงจรปรับความถี่และวงจรสร้างสัญญาณพัลส์ ป้อนให้กับขาเกตของไอซีที่ใช้ขับเพาเวอร์มอสเฟต

2. ส่วนโปรแกรม ซึ่งประกอบด้วยโปรแกรมการใช้งานโมดูล MCPWM ใน dsPIC30F2010, โปรแกรมใช้งานไทเมอร์3, โปรแกรมใช้งานโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอลภายใน dsPIC และโปรแกรมสื่อสารข้อมูลอนุกรม

3. ส่วนอินเวอร์เตอร์ประกอบด้วยแหล่งจ่ายไฟ 310โวลต์และอินเวอร์เตอร์แบบฟูลบริดจ์ 3 เฟส ทำหน้าที่กำเนิดไฟฟ้ากระแสสลับ 3 เฟส



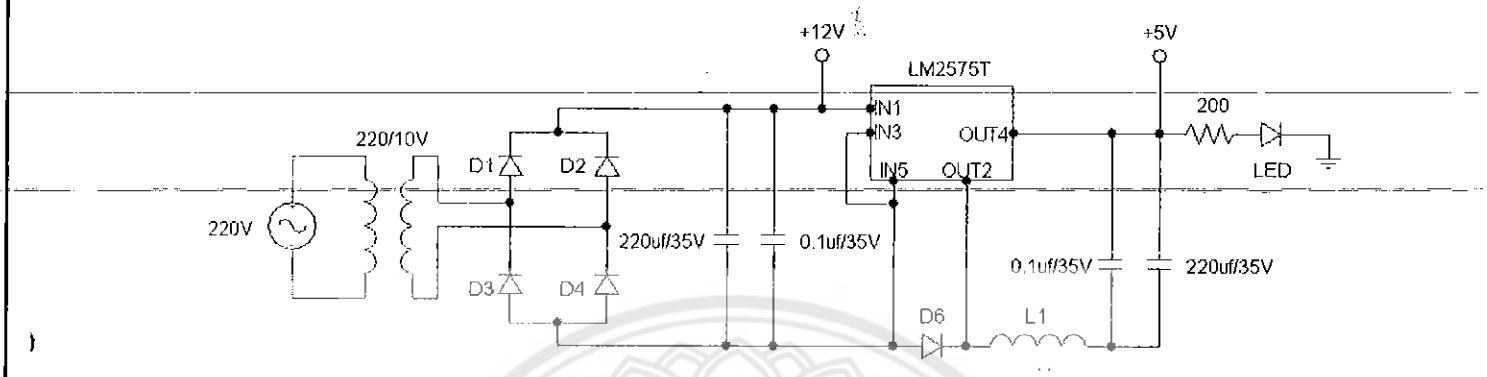
รูปที่ 4.1 รูปแบบเครื่องควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำ

จากรูปที่ 4.1 โครงสร้างของเครื่องอินเวอร์เตอร์ควบคุมความเร็วรอบของมอเตอร์ไฟฟ้าเหนี่ยวนำที่ใช้ dsPIC30F2010 เป็นตัวควบคุมปรับความถี่ และกำเนิดสัญญาณพัลส์เพื่อไปป้อนให้กับขาเกตเพื่อทำให้เพาเวอร์มอสเฟตทำงาน

4.2 ออกแบบวงจรกำเนิดสัญญาณพัลส์, ปรับความถี่ของมอเตอร์

ในโครงงานนี้บอร์ดควบคุมและแหล่งจ่ายไฟจะอยู่บนบอร์ดเดียวกันแต่จะอธิบายเป็น 2 ส่วน

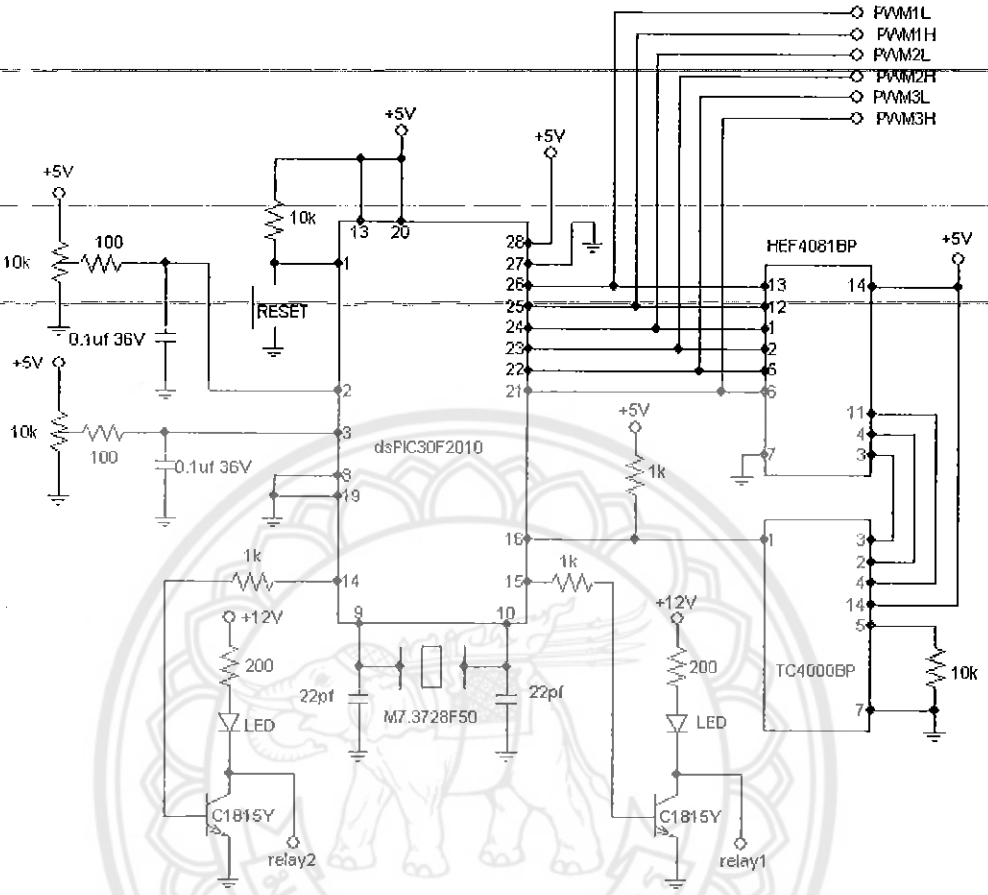
4.2.1 วงจรแหล่งจ่ายไฟกระแสตรง 5 โวลต์และ12 โวลต์



รูปที่ 4.2 วงจรแหล่งจ่ายไฟกระแสตรง 5 และ12 โวลต์บนบอร์ดควบคุม

จากวงจรแหล่งจ่ายไฟกระแสตรง แผงวงจรจะรับไฟกระแสสลับขนาด 220 โวลต์ผ่านหม้อแปลงลดแรงดันเหลือ 10 โวลต์ จากนั้นจะเข้าวงจรบริดจ์เร็กติไฟล์ (เบอร์ 2KBP06M ขนาด 600V 2A) จะทำให้ได้ไฟกระแสตรง 12 โวลต์เพื่อจ่ายให้กับรีเลย์ 2 ตัว และทำให้ไฟกระแสตรงเหลือ 5 โวลต์เพื่อจ่ายให้กับไอซีบนบอร์ดควบคุมในโครงงานนี้ใช้ โวลต์เตจ เร็กกรูเลเตอร์ (เบอร์ LM2575T) เป็นตัวลดแรงดันลงจาก 12 โวลต์เป็น 5 โวลต์แล้วจ่ายให้กับบอร์ดควบคุมเพื่อให้บอร์ดทำงานต่อไป

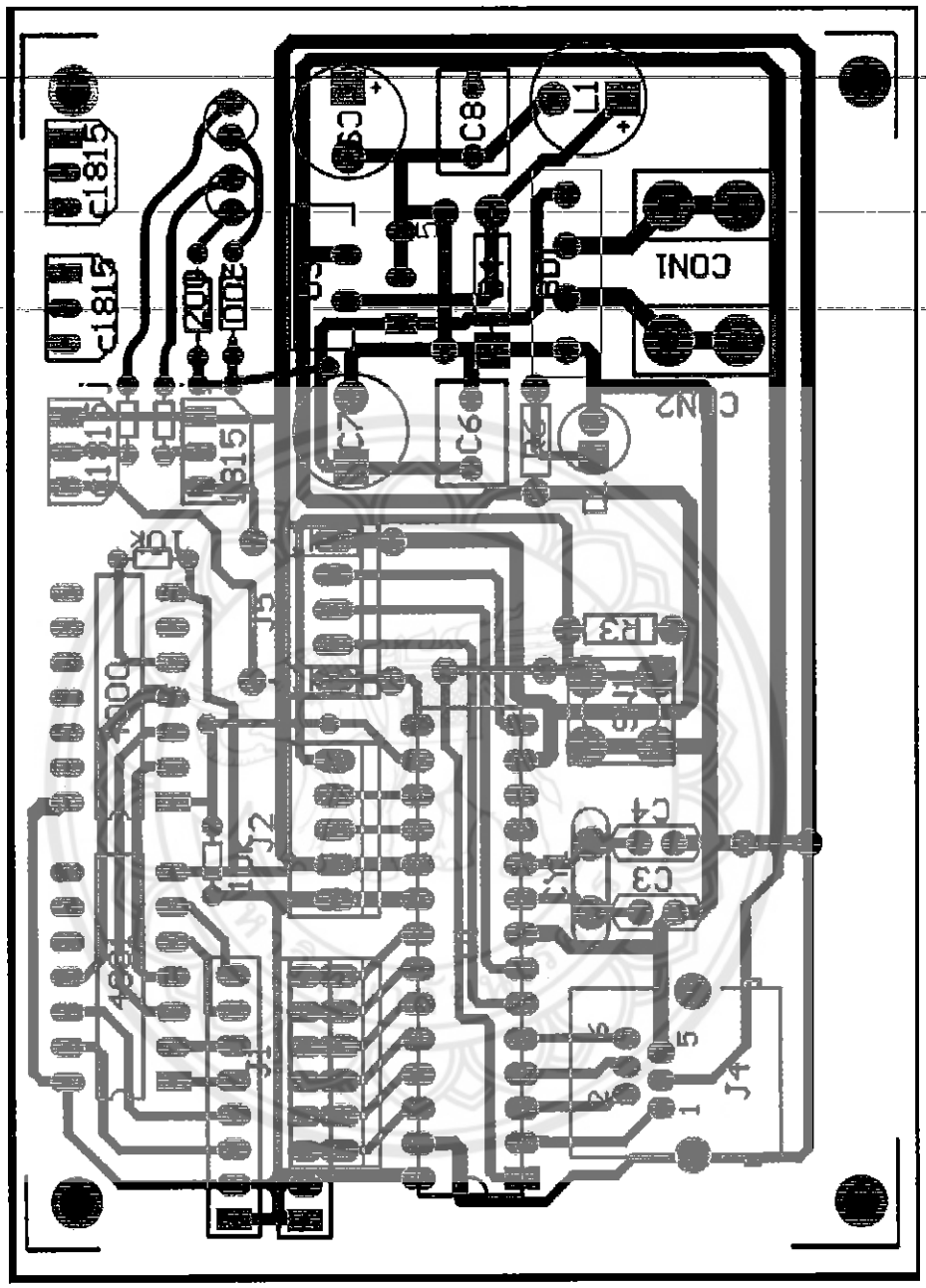
4.2.2 วงจรส่วนกำเนิดสัญญาณพัลส์และปรับความถี่มอเตอร์



รูปที่ 4.3 วงจรของส่วนควบคุมกำเนิดสัญญาณพัลส์และปรับความถี่

จากวงจรโครงงานนี้จะใช้ ไอซี dsPIC30F2010 เป็นตัวควบคุมความเร็วมอเตอร์และเป็นตัวกำเนิดสัญญาณพัลส์ (PWM) โดยdsPIC30F2010 จะรับไฟกระแสตรง 5 โวลต์จากแหล่งจ่ายไฟกระแสตรงข้างบนเข้าที่ขา 13 และ20 ส่วนขาที่ 28 และ27 จะต่อไฟเลี้ยงบวกและลบ 5 โวลต์ให้แก่โมดูล ADC ภายใน dsPIC ตามลำดับ ปุ่มปรับความถี่ของโครงงานนี้จะต่อเข้าที่ขาที่ 2 ของ dsPIC30F2010 สัญญาณเอาต์พุต PWM ที่จะนำไปขับเพาเวอร์มอสเฟตจะออกมาเป็นคู่ ทั้งหมด 3 คู่ คือขาที่ 25, 26, 23, 24 และ 21, 22 ดังรูปที่ 4.3

4.3 ลายวงจรบนบอร์ดกำเนิดสัญญาณพัลส์และแหล่งจ่ายไฟของโครงงานนี้

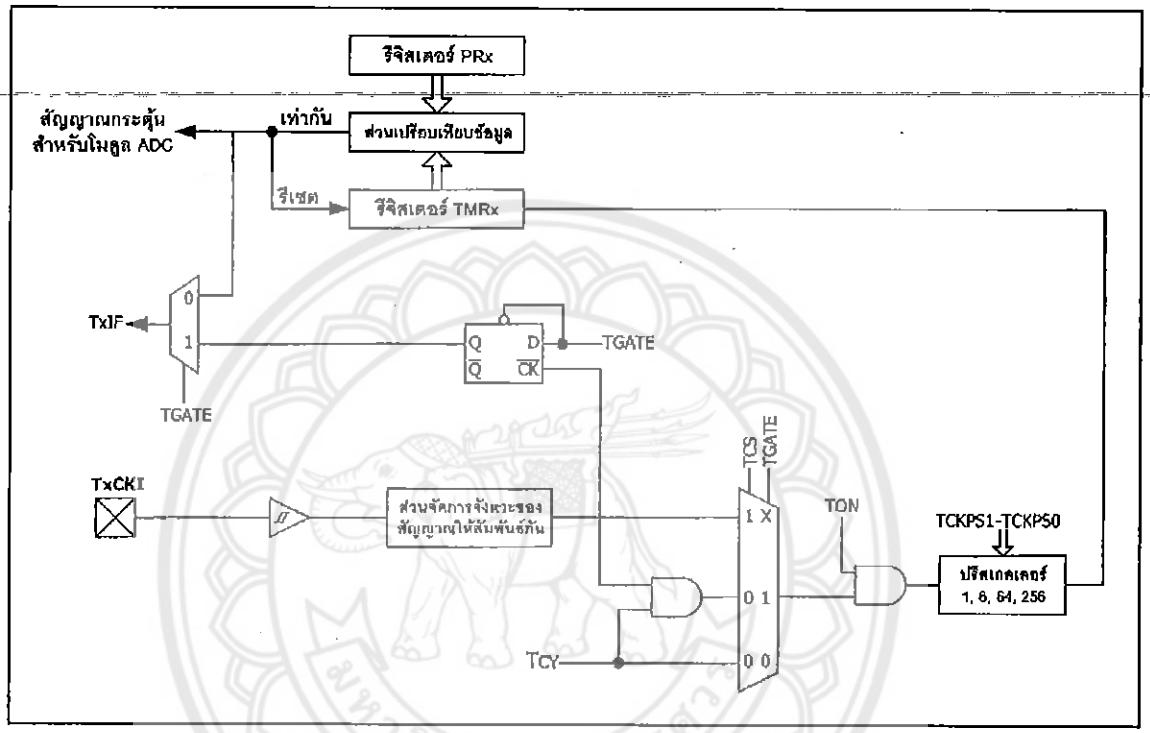


รูปที่ 4.4 ลายวงจรบนบอร์ดควบคุม

4.4 ส่วนโปรแกรมของโครงงาน

4.4.1 โปรแกรมไทเมอร์ใน dsPIC30F2010

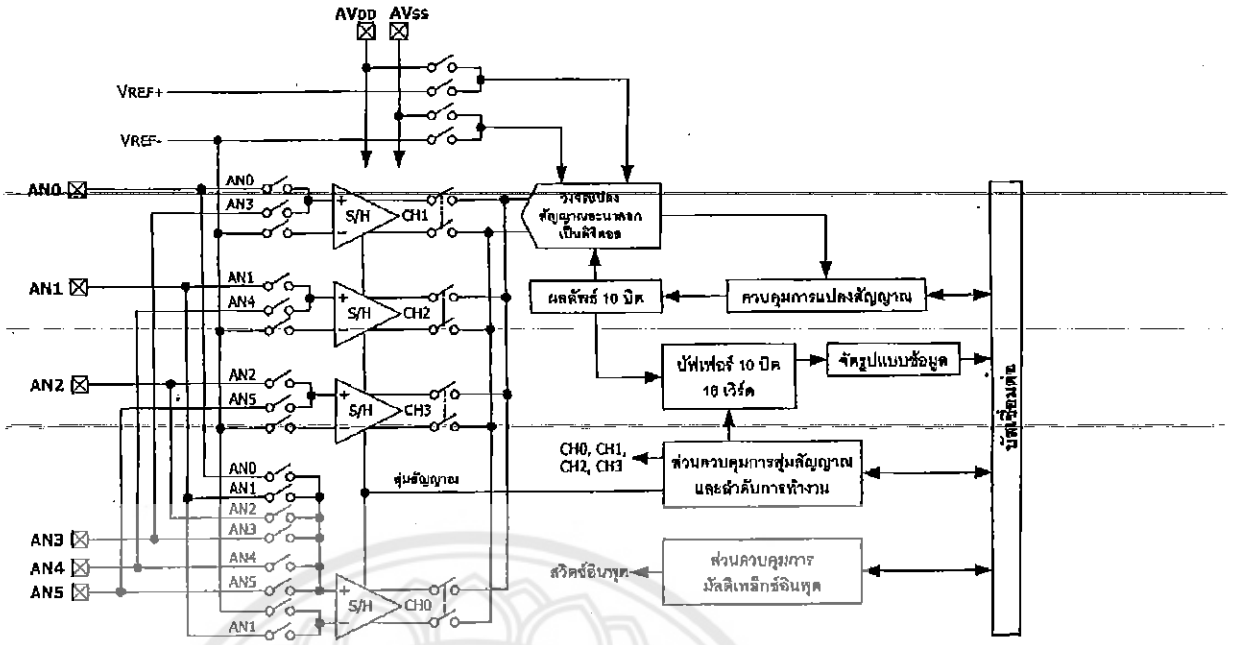
มีใช้งาน 3 ตัวคือT1,T2 และT3 ในโครงงานนี้เลือกใช้ไทเมอร์ 3 ซึ่งทำงานด้วยฐานเวลา C เพราะในโครงงานนี้ต้องการไทเมอร์ที่กำเนิดสัญญาณกระตุ้นการทำงานไปยังโมดูล ADC เพื่อทำให้ไทเมอร์และโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอลทำงานสัมพันธ์กัน



รูปที่ 4.5 ไดอะแกรมการทำงานของไทเมอร์ 3 ซึ่งทำงานด้วยฐานเวลาแบบ C

4.4.2 โปรแกรมใช้งานโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอลภายใน dsPIC (ADC)

โปรแกรมนี้เป็นโปรแกรมการทำงานของปุ่มปรับความถี่และปุ่มปรับทอร์คมอเตอร์โดยปุ่มปรับความถี่จะอยู่พอร์ตอินพุตที่ AN0 (ขา 2) และปุ่มปรับทอร์คมอเตอร์จะอยู่ที่พอร์ต AN1 (ขา3) ของ dsPIC30F2010



รูปที่ 4.6 ไดอะแกรมการทำงานของโมดูล ADC ใน dsPIC30F2010

ในรูปที่ 4.6 เป็นไดอะแกรมการทำงานของโมดูล ADC ใน dsPIC30F2010 ซึ่งขาพอร์ตอินพุตอะนาลอกทั้งสิ้น 6 ขาคือ AN0-AN5 โดยมี 2 ขาที่สามารถใช้รับแรงดันอ้างอิงเพื่อขยายผ่านของแรงดันอินพุตภายในโมดูลมีวงจรสุ่มและเก็บค่าสัญญาณ (Sample and Hold: S/H) จำนวน 4 ชุด โดยทำงานร่วมกันส่วนควบคุมการมัลติเพล็กซ์สัญญาณอินพุต ทำให้สามารถจัดสรรวงจร S/H ให้สามารถรองรับกับสัญญาณอินพุตอะนาลอกทั้ง 6 ช่องได้ด้วยความเร็วสูงสุด

สัญญาณที่ผ่านจากวงจร S/H จะถูกป้อนเข้าสู่วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลแบบซักเซสซีฟ แอ็ปพรอกซิเมชั่น ขนาด 10 บิต ข้อมูลที่ได้จากการแปลงจะถูกพักไว้ในหน่วยความจำแรม จากนั้นจะได้รับการจัดรูปแบบตามที่ผู้พัฒนาโปรแกรมกำหนดจากนั้นข้อมูลจะถูกถ่ายทอดลงบนบัสข้อมูลเพื่อส่งไปยังซีพียูต่อไป

อีกองค์ประกอบหนึ่งที่ทำให้โมดูล ADC สามารถแปลงสัญญาณได้รวดเร็วคือภายในโมดูล ADC มีบัฟเฟอร์ความจุ 16 เวิร์ด นั่นคือ สามารถรองรับข้อมูลที่ได้จากการแปลงสูงสุด 16 ชุดข้อมูล ดังนั้นเมื่อแปลงสัญญาณครั้งหนึ่งก็นำมาเก็บไว้ที่บัฟเฟอร์ หากบัฟเฟอร์ยังไม่เต็มก็สามารถกลับไปแปลงสัญญาณต่อได้ทันที โดยไม่ต้องรอให้การถ่ายทอดข้อมูลไปยังรีจีสเตอร์ที่ใช้เก็บค่าการแปลงเสร็จสิ้น

การกำหนดค่าเพื่อใช้งานโมดูล ADC

มีขั้นตอนโดยสรุปดังนี้

1. ตั้งค่าของโมดูล ADC

1.1 เลือกขาพอร์ตให้ทำงานเป็นอินพุตอะนาลอกที่รีจิสเตอร์ ADPCFG

1.2 เลือกแหล่งจ่ายแรงดันอ้างอิงให้เหมาะสมกับย่านแรงดันอะนาลอกทางอินพุตที่บิต 15 ถึง 13 ของรีจีสเตอร์ ADCON2

1.3 เลือกสัญญาณนาฬิกาสำหรับการเปลี่ยนแปลงสัญญาณที่บิต 5 ถึง 0 ของรีจีสเตอร์ ADCON3

1.4 กำหนดจำนวนช่องของวงจร S/H ที่ต้องใช้ที่บิต 9 และ 8 ของรีจีสเตอร์ ADCON2 และรีจีสเตอร์ ADPCFG

1.5 กำหนดวิธีการสุ่มสัญญาณที่บิต 3 ของรีจีสเตอร์ ADCON1และรีจีสเตอร์ ADCSSL

1.6 กำหนดจำนวนอินพุตที่ต้องทำงานร่วมกับวงจร S/H ที่รีจีสเตอร์ ADCHS

1.7 เลือกลำดับการสุ่มและแปลงสัญญาณที่บิต 7 ถึง 0 ของรีจีสเตอร์ ADCON1และบิต 12 ถึง 18 ของรีจีสเตอร์ ADCON3

1.8 เลือกรูปแบบของผลลัพธ์ที่ต้องการที่บิต 9 และ 8 ของรีจีสเตอร์ ADCON1

1.9 เลือกการอินเตอร์รัปต์ที่บิต 9 ถึง 5 ของรีจีสเตอร์ ADCON2

1.10 เปิดการทำงานของโมดูล ADC ที่บิต 15 ของรีจีสเตอร์ ADCON1

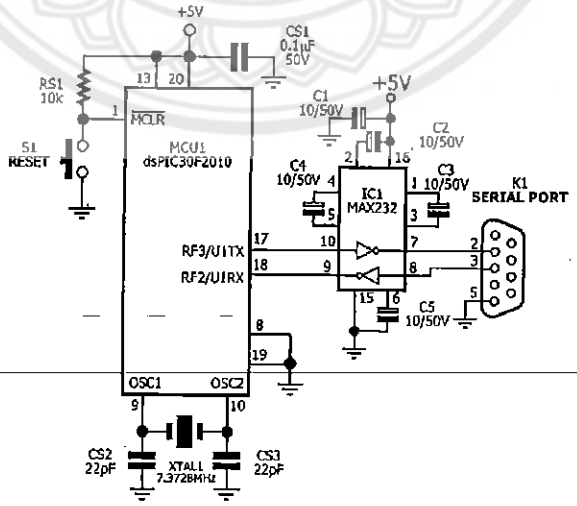
2. กำหนดการอินเตอร์รัปต์ (ถ้าต้องการ)

2.1 เคลียร์บิต ADIF

2.2 เลือกระดับความสำคัญของอินเตอร์รัปต์

4.4.3 โปรแกรมสื่อสารข้อมูลอนุกรม

โปรแกรมนี้เป็นการเขียนเพื่อใช้งานโมดูลUARTซึ่งเป็นโมดูลสื่อสารข้อมูลอนุกรมระหว่าง dsPIC30F2010 กับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม



รูปที่ 4.7 วงจรสื่อสารข้อมูลระหว่าง dsPIC30F2010 กับคอมพิวเตอร์ผ่านพอร์ตอนุกรม

รายละเอียดของโปรแกรมนี้

1. ฟังก์ชั่น uartl_init ทำหน้าที่กำหนดคุณสมบัติเริ่มต้นของโมดูล UART1 เพื่อใช้สำหรับการสื่อสารข้อมูลอนุกรมระหว่างไมโครคอนโทรลเลอร์กับเครื่อง PC โดยกำหนดให้รับส่งข้อมูลในรูปแบบอินเตอร์รัปต์ที่อัตราบอด 9,600 บิตต่อวินาที ซึ่งผู้พัฒนาต้องทำการผนวก (include) ไลบรารี uart.h เข้ามาภายในโปรแกรม เพื่อเรียกใช้งานฟังก์ชั่นต่าง ๆ เกี่ยวกับโมดูล UART1 ของ dsPIC30F2010

2. ค่าอัตราบอดในการสื่อสารเท่ากับ 9,600 บิตต่อวินาที เป็นการกำหนดค่าจากตัวแปร baudvalue โดยค่าที่ถูกกำหนดจะถูกโหลดไปยังรีจีสเตอร์ U1BRG อีกทอดนึ่งในขั้นตอนการเรียกใช้งานฟังก์ชั่น OpenUART1 ภายในโปรแกรม ซึ่งควบคุมอัตราบอดในการสื่อสารข้อมูลอนุกรม

3. ฟังก์ชัน void _ISR _U1TXInterrupt (void) ทำหน้าที่ตอบสนองอินเตอร์รัพต์เนื่องจากการส่งข้อมูลของ UART1 เสร็จสิ้น

4. ฟังก์ชัน void _ISR _U1TXInterrupt (void) ทำหน้าที่ตอบสนองอินเตอร์รัพต์เนื่องจากการส่งข้อมูลของ UART1

4.4.4 โปรแกรมใช้งานโมดูล MCPWM ใน dsPIC30F2010

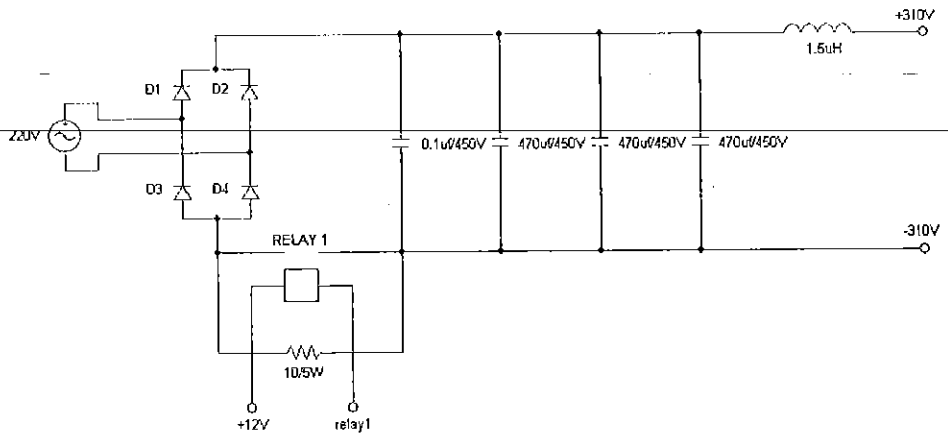
เป็นโปรแกรมหลักของโครงงานในครั้งนี้ซึ่งรายละเอียดนั้นได้อธิบายไว้อย่างละเอียดในบทที่ 3

หมายเหตุ ตัวโปรแกรมที่ใช้งานของโครงงานนี้ทั้งหมดอยู่ในภาคผนวก ก.

4.5 ออกแบบวงจรอินเวอร์เตอร์

ในส่วนนี้จะประกอบไปด้วย ส่วนของแหล่งจ่ายกำลังไฟกระแสตรงและส่วนของวงจรขับเพาเวอร์มอสเฟต ซึ่งแหล่งจ่ายไฟกระแสตรงมีอยู่ 2 ระดับแรงดันคือ 310 โวลต์หรือแหล่งจ่ายไฟหลักจำนวน 1 ชุดที่จ่ายเข้าขา DRAIN ของเพาเวอร์มอสเฟสเพื่อสร้างไฟกระแสสลับ 3 เฟส ออกมาและ 24 โวลต์ จำนวน 6 ชุดที่จ่ายให้กับ ไอซี TLP 250 ซึ่งเป็นไอซีของวงจรขับนำเกตที่ใช้ในการสวิตช์ขาเพาเวอร์มอสเฟต

4.5.1 แหล่งจ่ายไฟกระแสตรง 310 โวลต์



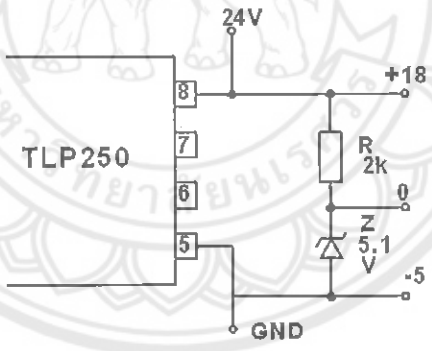
รูปที่ 4.8 วงจรแหล่งจ่ายไฟกระแสตรง 310โวลต์

แหล่งจ่ายแรงดันของระบบประกอบด้วยวงจรเรียงกระแสแบบบริดจ์และวงจรกรองแรงดัน โดยจะรับแรงดันไฟกระแสสลับ 220 โวลต์ 50 เฮิร์ต 1เฟส และเพื่อเปลี่ยนให้เป็นแรงดันไฟตรงสูง-สุด (V_p) ประมาณ 310 โวลต์ ในโครงงานนี้จึงใช้วงจรบริดจ์เร็กติไฟล์ (เบอร์ KBPC5010 ขนาด 1000V 50A) เป็นตัวแปลงกระแสไฟฟ้าจากกระแสสลับ 220 โวลต์เป็นกระแสตรง 310 โวลต์ ใช้ตัวเก็บประจุทั้งหมด 4 ตัวคือ 0.1μF/450V จำนวน 1 ตัวและ 470μF/450V จำนวน 3 ตัวเป็นตัวกรองแรงดันให้แรงดันเรียบ

4.5.2 วงจรขับนำเกต

วงจรนี้จะมีทั้งหมด 6 ชุดเพื่อขับนำเกตที่ขาเพาเวอร์มอสเฟตทั้งหมด 6 ตัวซึ่งในแต่ละวงจรจะมีแหล่งจ่ายไฟกระแสตรงขนาด 24 โวลต์ไปเลี้ยงวงจรขับนำเกต

วงจรขับนำเกตที่ใช้ในโครงงานนี้เลือกใช้วงจรขับเกตแบบ Opto Couple โดยใช้ไอซีขับเกตเบอร์ TLP250 จากรูปที่ 4.9 ทางด้านอินพุตจะรับสัญญาณขับเกตผ่าน Photo Diode ส่งสัญญาณไปทางวงจรขยายด้านเอาต์พุต ซึ่งใช้ทรานซิสเตอร์ทำงานแบบ Push Pull ส่งไปยังขาเกตของเพาเวอร์มอสเฟต ขณะนำกระแส ระดับของแรงดันขับเกต 13 โวลต์ และขณะหยุดกระแสมีค่า -5 โวลต์ โดยใช้ซีเนอร์ไดโอด Z1 เป็นตัวยกระดับแรงดันที่ขาเดรนขึ้นเป็น 5.1 โวลต์

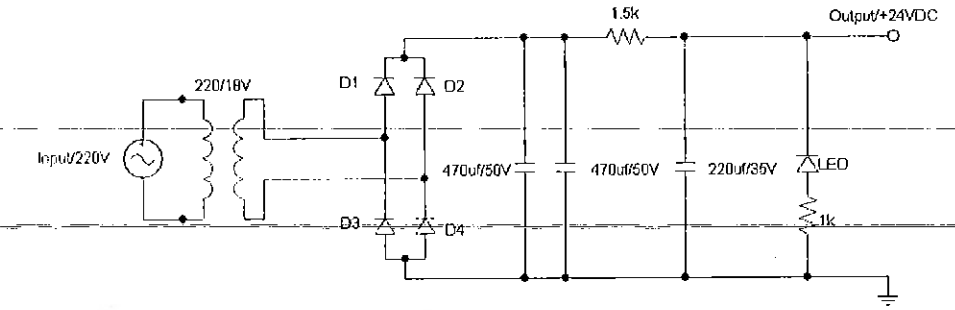


รูปที่ 4.9 การขับนำเขตด้วยไอซี TLP250

การต่อแหล่งจ่ายไฟตรงนั้น จะพิจารณาจากช่วงการนำกระแสและหยุดนำกระแสของเพาเวอร์มอสเฟตโดยช่วงการนำกระแสของเพาเวอร์มอสเฟตต้องการแรงดันขับเกตที่ +18 โวลต์ ส่วนในช่วงหยุดนำกระแสเพาเวอร์มอสเฟตต้องการแรงดันขับเกตที่ -5 โวลต์ ซึ่งเป็นการป้องกันสภาวะตกค้างของประจุภายในตัวไปในตัวด้วย จากหลักการดังกล่าว จึงได้ออกแบบแหล่งจ่ายที่ระดับ 24 โวลต์ จ่ายให้กับวงจรทำให้ได้แรงดันตามหลักการของเพาเวอร์มอสเฟตดังรูปที่ 4.10

แหล่งจ่ายไฟกระแสตรง 24 โวลต์

ในวงจรขับนำเกตทั้ง 6 วงจรจะมีแหล่งไฟกระแสตรงทุกวงจรลักษณะของวงจรจะเป็นดังรูป 4.10

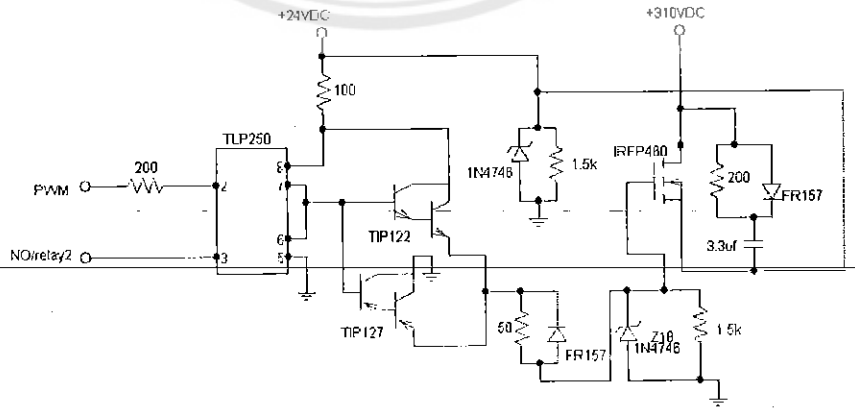


รูปที่ 4.10 วงจรแหล่งจ่ายไฟกระแสตรง 24 โวลต์

จากวงจรรูป 4.10 มีหลักการทำงานคือ วงจรจะรับไฟกระแสสลับขนาด 220 โวลต์ผ่านหม้อแปลงลดแรงดันให้เหลือ 18 โวลต์จากนั้นก็เข้าวงจรบริดจ์เร็กติไฟล์ (เบอร์ 2KBP06M ขนาด 600V 2A) เพื่อแปลงเป็นกระแสตรง โดยใช้ตัวเก็บประจุ 3 ตัวขนาด 470μF/50V 2 ตัว และ 220μF/35V 1 ตัว เป็นกรองแรงดันให้เรียบ เอาท์พุตที่ได้จะเป็นไฟกระแสตรง 24 โวลต์ และจ่ายให้กับวงจรขับนำเกตต่อไป

วงจรขับนำเกตของโครงงานนี้

วงจรขับนำเกตในโครงงานนี้ใช้ไอซี TLP250 เป็นไอซีสวิตช์ ON-OFF ของขาเพาเวอร์มอสเฟต ซึ่งสัญญาณพัลส์ (PWM) ที่ถูกสร้างจากบอร์ดควบคุมจะเข้าที่ขา 2 ของไอซีและรับไฟกระแสตรง 24 โวลต์เพื่อเลี้ยงวงจรเข้าที่ขา 8 และขา 5 ต่อลงกราวด์โดยต่อตัวต้านทานที่ขาเข้าเพื่อกำจัดกระแสด้วย



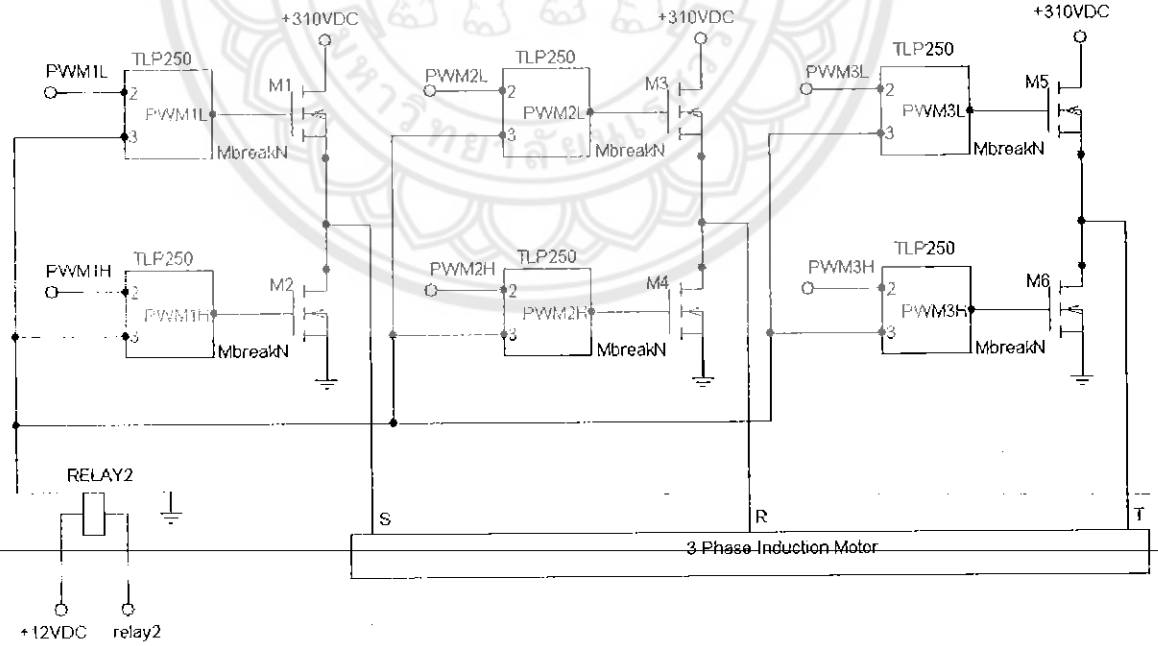
รูปที่ 4.11 วงจรขับนำเกตโดยใช้ไอซี TLP250 ของโครงงานนี้

ในส่วนของวงจรจะเป็นส่วนที่ใช้ในการขับนำสัญญาณที่ได้จากเอาต์พุตของภาค PWM เพื่อที่จะเป็นตัวกำหนดให้เพาเวอร์มอสเฟตทำงานได้ในลำดับที่เหมาะสม จากรูปที่ 4.11 จะเป็นรูปของวงจรขับนำเกตโดยใช้ไอซี TLP250 เป็นตัวขับนำเกต โดยมีการทำงานของวงจรมีดังนี้

-ในสภาวะที่ไม่มีสัญญาณอินพุตเข้ามา วงจรจะจ่ายแรงดัน -5 โวลต์ ให้กับขาเกตของเพาเวอร์มอสเฟส เพื่อเป็นการป้องกันไม่ให้เพาเวอร์มอสเฟสทำงาน

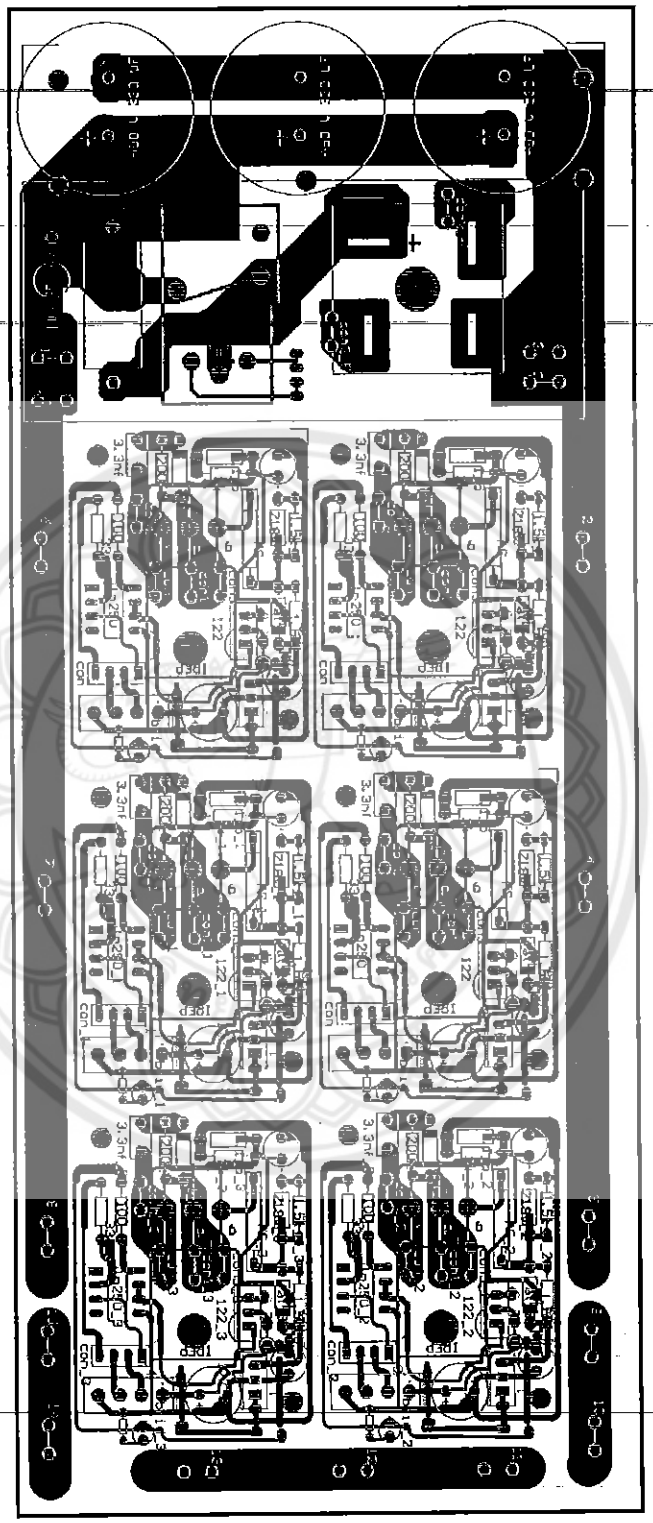
-ในสภาวะทำงาน ทางด้านอินพุทจะรับสัญญาณ PWM ผ่าน Photo Diode ส่งสัญญาณไปทางวงจรขยายด้านเอาต์พุต ซึ่งใช้ทรานซิสเตอร์ทำงานแบบ Push Pull ส่งไปยังขาเกตของเพาเวอร์มอสเฟสขณะนำกระแส ระดับของแรงดันขับเกต +18 โวลต์ให้เอาต์พุตที่ได้จากขา 1 และขา 3 มีลักษณะเหมือนเอาต์พุตที่ได้มาจากภาค PWM เพียงแต่ระดับแรงดันที่ได้ต่างกัน คือ เอาต์พุตที่ได้ +18 โวลต์

และเมื่อนำวงจรขับนำเกตทั้ง 6 วงจรมาต่อเข้าด้วยกันจะได้ตามรูปที่ 4.12 โดยที่ขา DRAIN ของเพาเวอร์มอสเฟตตัวที่ 1L จะรับไฟกระแสตรง 310 โวลต์จากแหล่งจ่ายและเพาเวอร์มอสเฟตตัวที่ 1H จะรับไฟกระแสตรง 310 โวลต์จากขา SOURCE ของตัวที่ 1L อีกทีและได้เอาพุตออกที่ขานั้นด้วย ซึ่งตัวที่ 2L, 2H, 3L และ3H ก็จะต่อแบบตัวที่ 1L และ 1H เช่นกัน ส่วนขา SOURCE ของเพาเวอร์มอสเฟตตัวที่ 1H, 2H และ3H จะต่อลงกราวด์ ในโครงงานนี้ใช้เพาเวอร์มอสเฟส IRFP460 ขนาด 500 โวลต์ 20 แอมแปร์

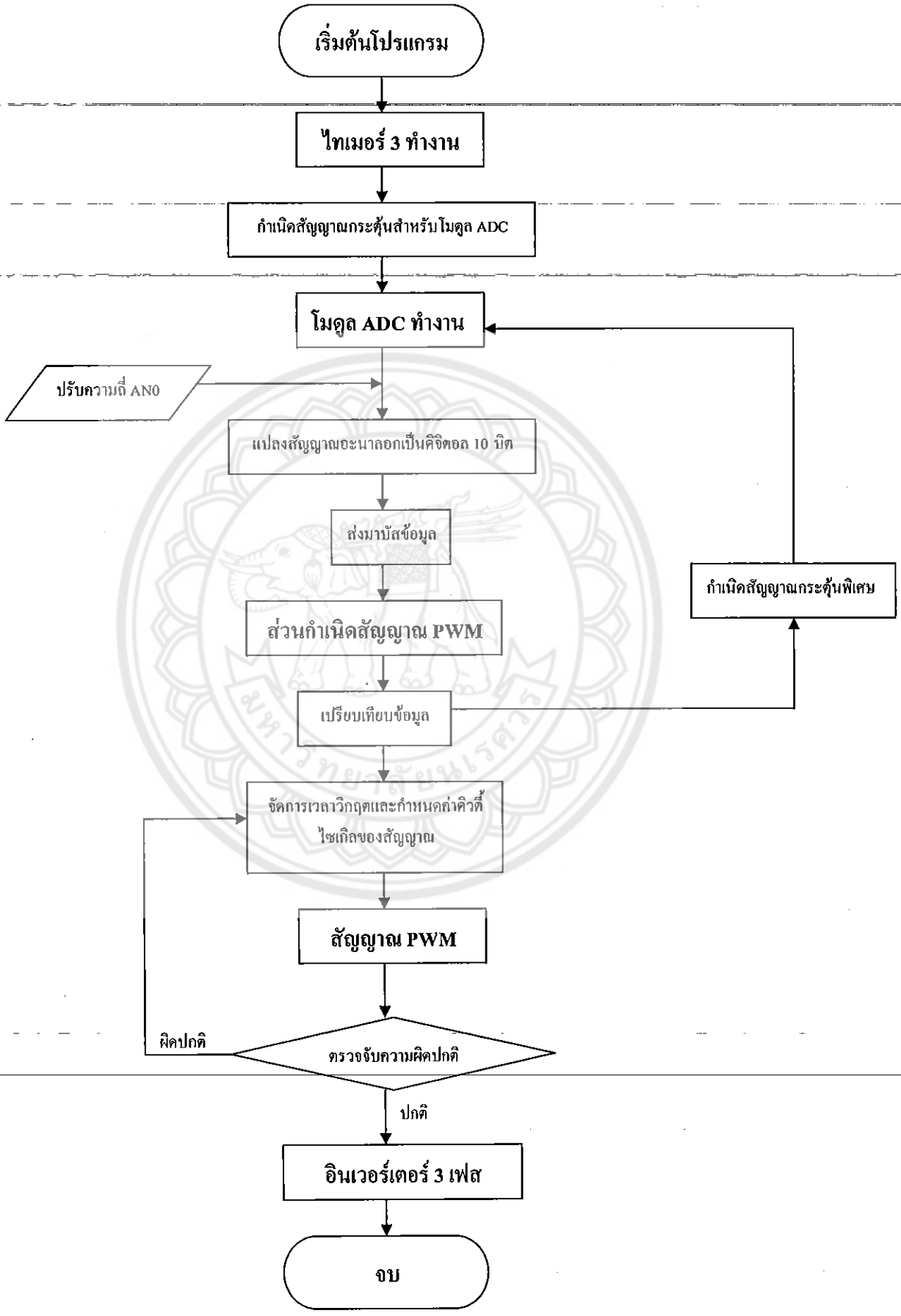


รูปที่ 4.12 วงจรขับนำเกตเมื่อนำวงจรทั้ง 6 วงจรมาต่อกัน

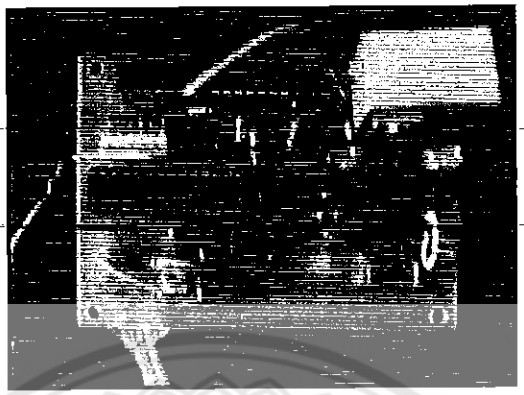
4.6 ลายวงจรส่วนของอินเวอร์เตอร์ 3 เฟสและแหล่งจ่ายไฟของโครงงานนี้



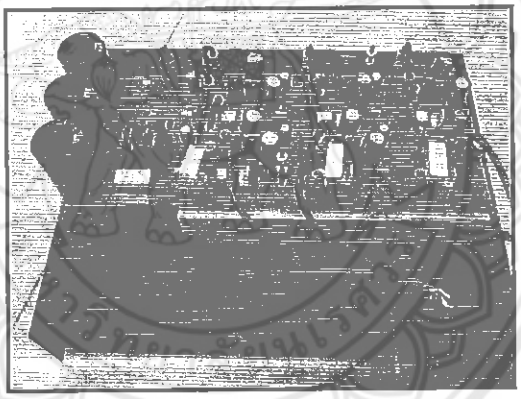
รูปที่ 4.13 ลายวงจรส่วนของอินเวอร์เตอร์ 3 เฟสและแหล่งจ่ายไฟของโครงงานนี้


เริ่มต้นโปรแกรม
ไทเมอร์ 3 ทำงาน
กำเนิดสัญญาณกระตุ้นสำหรับโมดูล ADC
โมดูล ADC ทำงาน
ปรับความถี่ AN0
แปลงสัญญาณอะนาลอกเป็นดิจิตอล 10 บิต
ส่งมาบัสข้อมูล
ส่วนกำเนิดสัญญาณ PWM
เปรียบเทียบข้อมูล
กำเนิดสัญญาณกระตุ้นพิเศษ
จัดการเวลาวิกฤตและกำหนดค่าดิวตี้ไซเกิลของสัญญาณ
สัญญาณ PWM
ผิดปกติ
ตรวจจับความผิดปกติ
ปกติ
อินเวอร์เตอร์ 3 เฟส
จบ

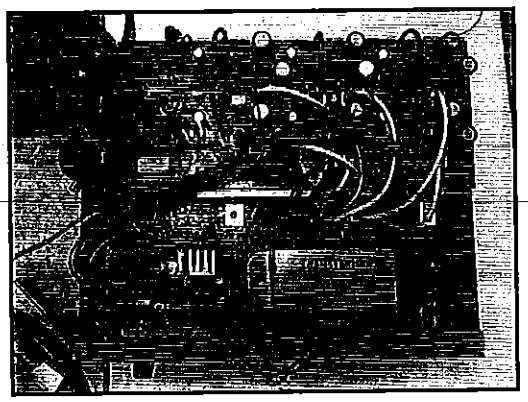
บทที่ 5
ผลการทดลองและวิเคราะห์ผลการทดลอง



รูปที่ 5.1 บอร์ดวงจรควบคุมที่สร้างขึ้น

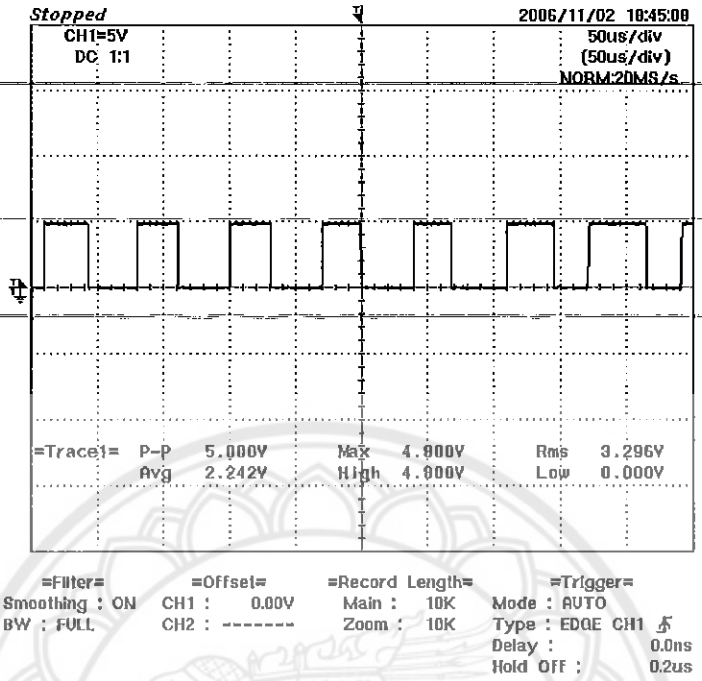


รูปที่ 5.2 บอร์ดวงจรอินเวอร์เตอร์ที่สร้างขึ้น

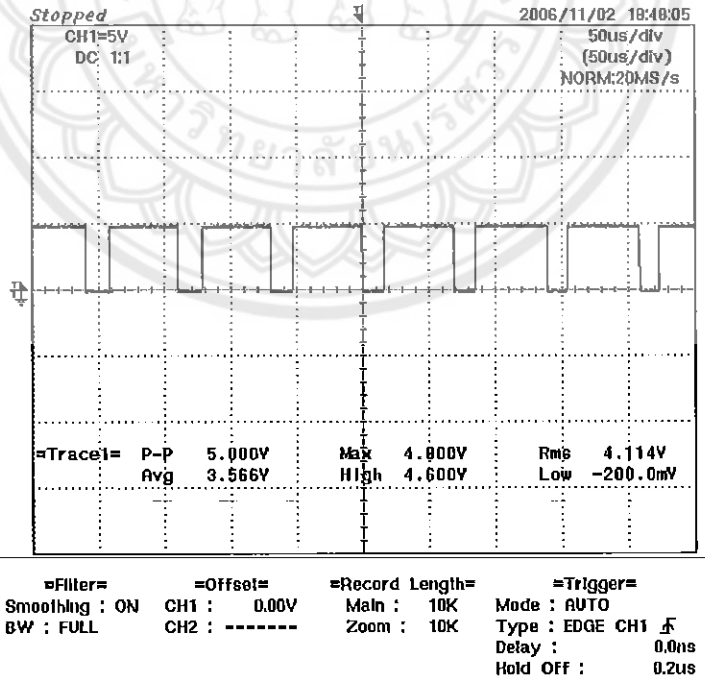


รูปที่ 5.3 เครื่องควบคุมความเร็วรอบมอเตอร์เหนี่ยวนำที่สร้างและประกอบเสร็จสมบูรณ์

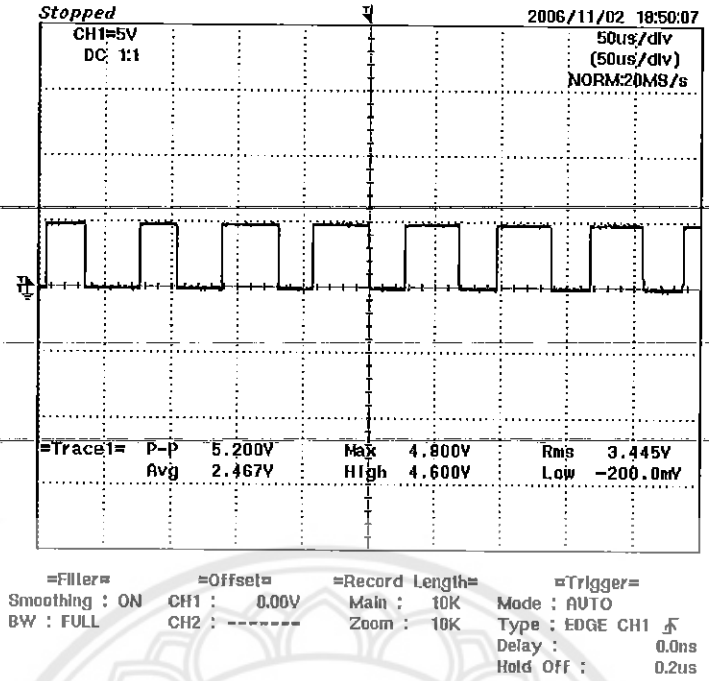
5.1 สัญญาณ PWM ที่สร้างขึ้นจากไอซี dsPIC30F2010



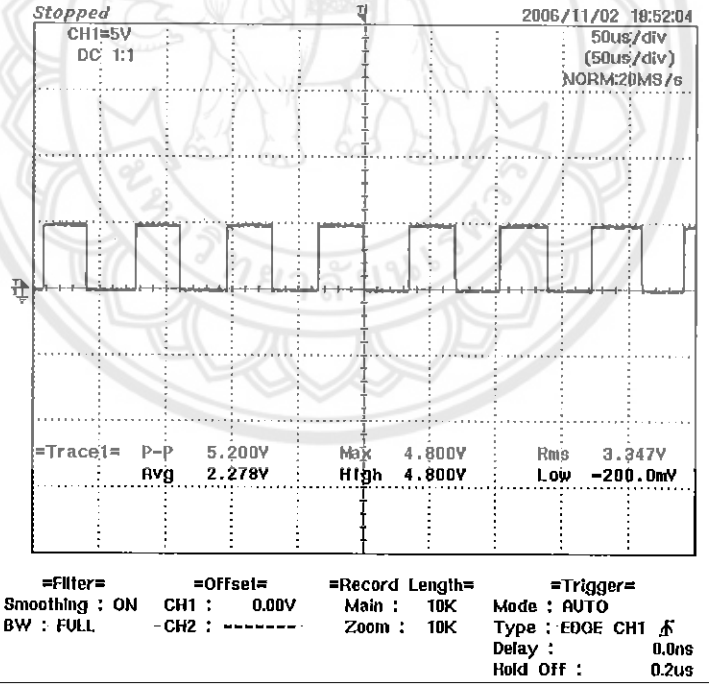
รูปที่ 5.4 สัญญาณ PWM 1L



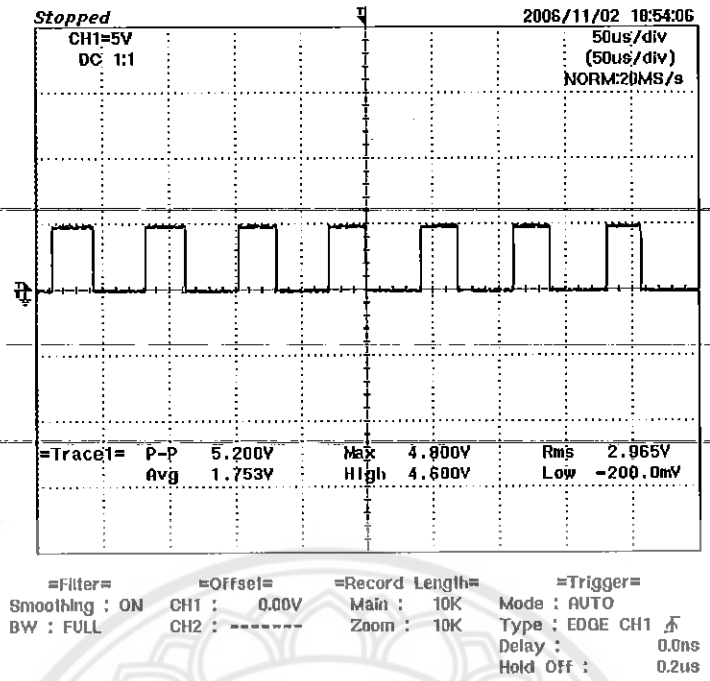
รูปที่ 5.5 สัญญาณ PWM 1H



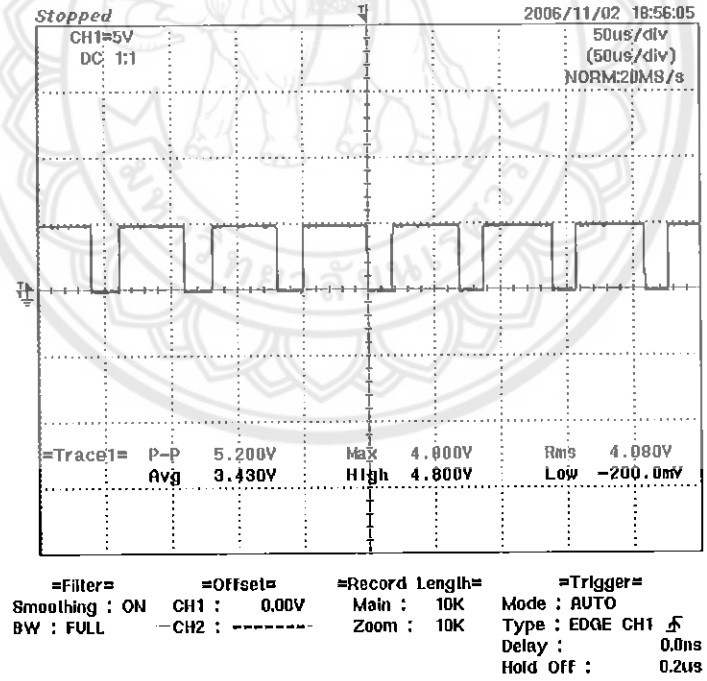
รูปที่ 5.6 สัญญาณ PWM 2L



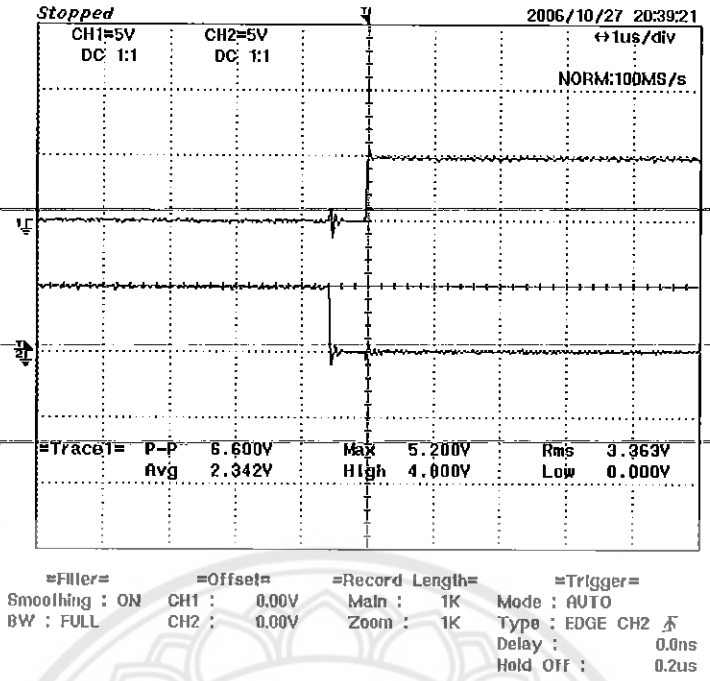
รูปที่ 5.7 สัญญาณ PWM 2H



รูปที่ 5.8 สัญญาณ PWM 3L



รูปที่ 5.9 สัญญาณ PWM 3H



รูปที่ 5.10 สัญญาณ Dead time ของค้านแรงดันต่างกันแต่อยู่กิ่งเดียวกัน

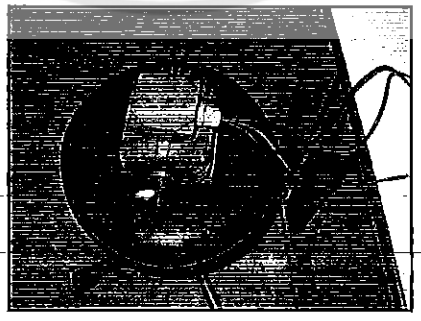
5.2 ความเร็วรอบมอเตอร์

ในโครงงานนี้ใช้มอเตอร์เหนี่ยวนำไฟฟ้า 3 เฟส ขนาด 4 Pole, ½ HP, 220/380V, 50/60 Hz เป็นมอเตอร์ทดลอง ซึ่งการหาความเร็วรอบมอเตอร์หาได้จากสูตร

$$n_s = \frac{120f}{P}$$

โดยที่ P คือ จำนวนขั้วของมอเตอร์

และ f คือความถี่ที่ป้อนให้กับมอเตอร์



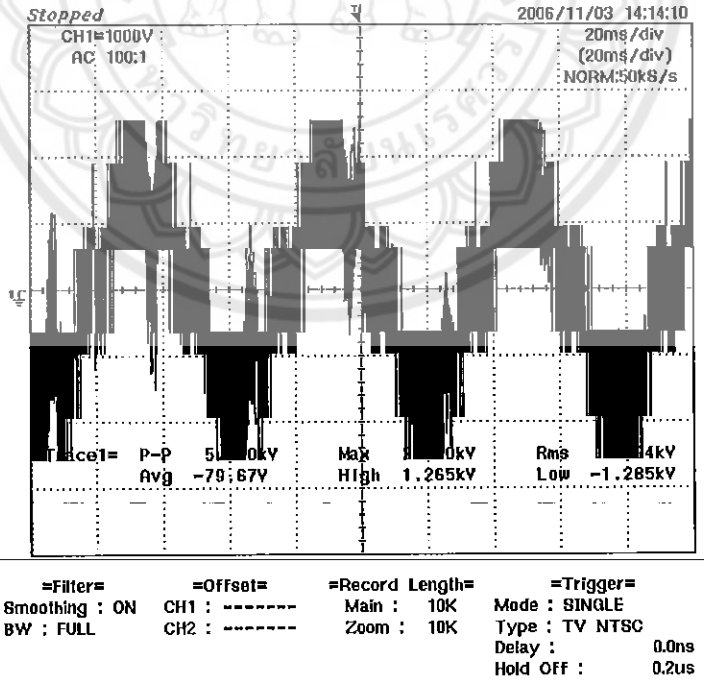
รูปที่ 5.11 มอเตอร์ที่ใช้ทดลองในโครงงานนี้

ผลการทดลองวัดค่าความเร็ว ณ ความถี่ต่างๆ (เอาต์พุต 220V ปรับทอร์คปานกลาง)

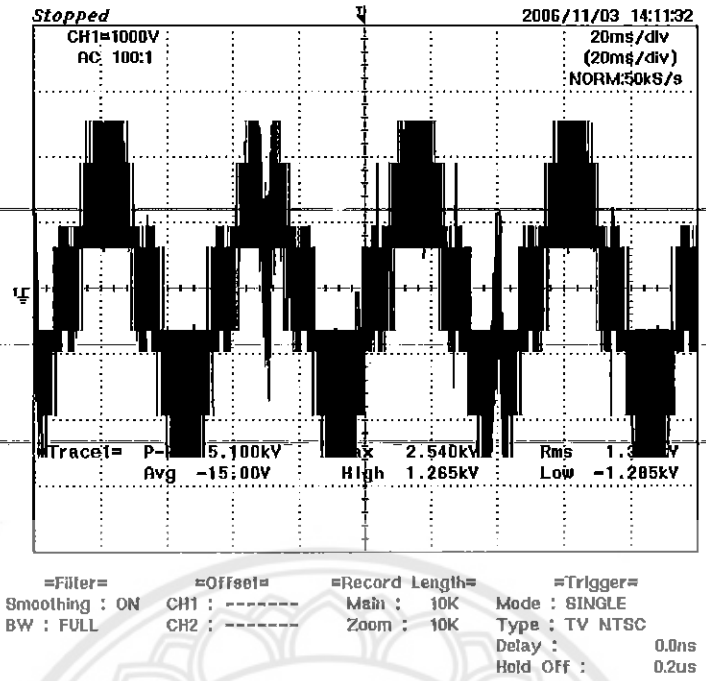
ความถี่ที่ปรับ	ความเร็วรอบที่ได้จากการคำนวณ	ความเร็วรอบที่วัดที่ได้จากการทดลอง	แรงดันเฟสเอาต์พุต (V)			กระแสเฟส (A)		
			V_{ab}	V_{ac}	V_{bc}	i_a	i_b	i_c
15	450	420	46.17	44.57	60.16	0.51	0.53	0.53
25	750	714	88.35	87.31	102.11	0.52	0.50	0.52
35	1050	1023	107.12	107.68	124.45	0.56	0.53	0.55
45	1350	1337	135.24	137.85	155.33	0.52	0.49	0.52
55	1650	1595	150.87	154.45	179.65	0.53	0.54	0.52
65	1950	1920	168.55	165.23	202.34	0.53	0.52	0.53

ตารางที่ 5.1 ผลการทดลองวัดค่าความเร็วรอบ ณ ความถี่ต่างๆ

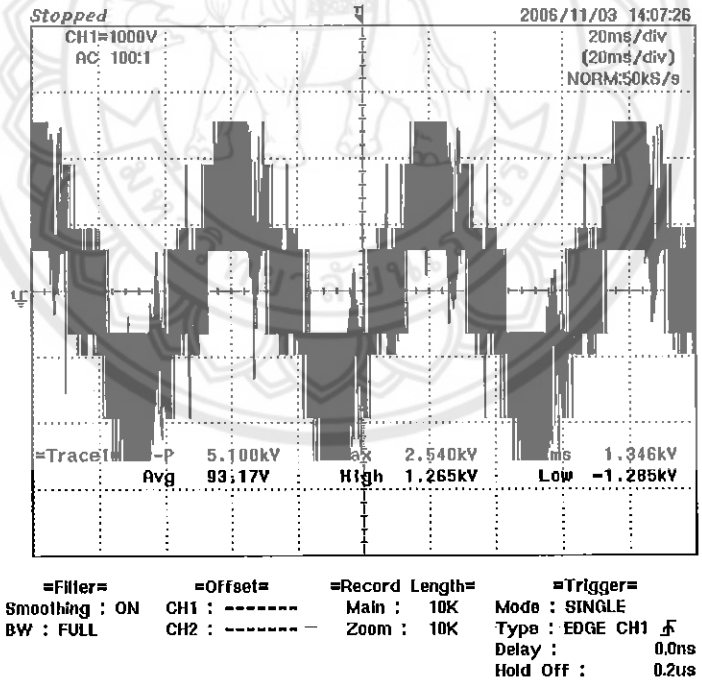
5.3 สัญญาณแรงดันและกระแสเอาต์พุตขณะต่อมอเตอร์ไฟฟ้าเหนี่ยวนำ 3 เฟส



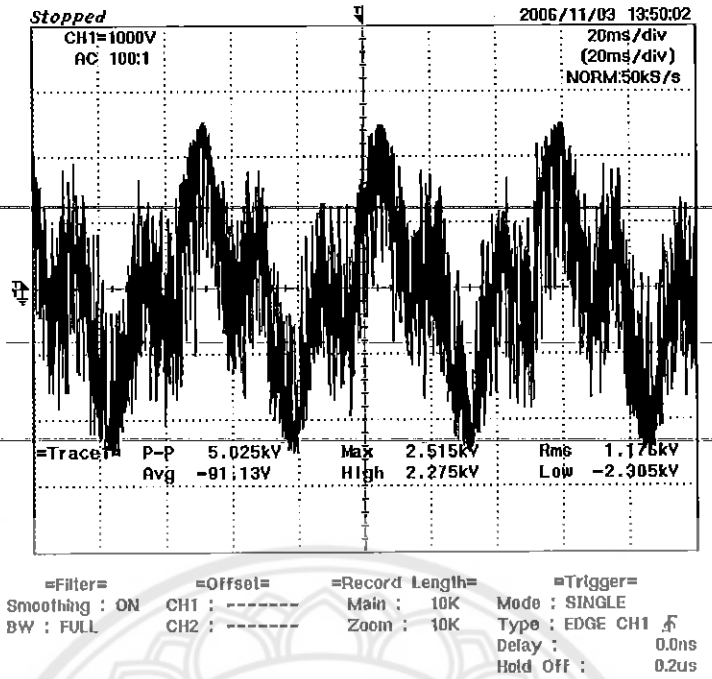
รูปที่ 5.12 สัญญาณแรงดัน V_{an}



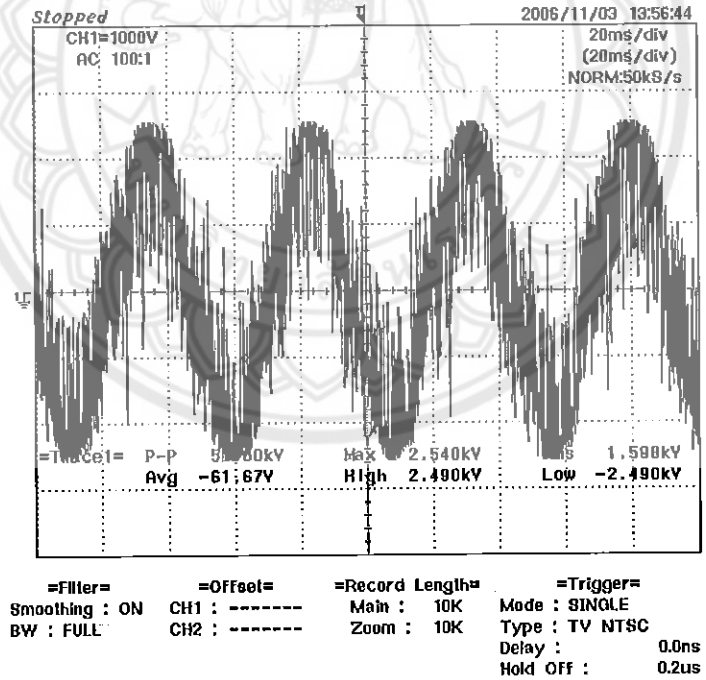
รูปที่ 5.13 สัญญาณแรงดัน V_{bn}



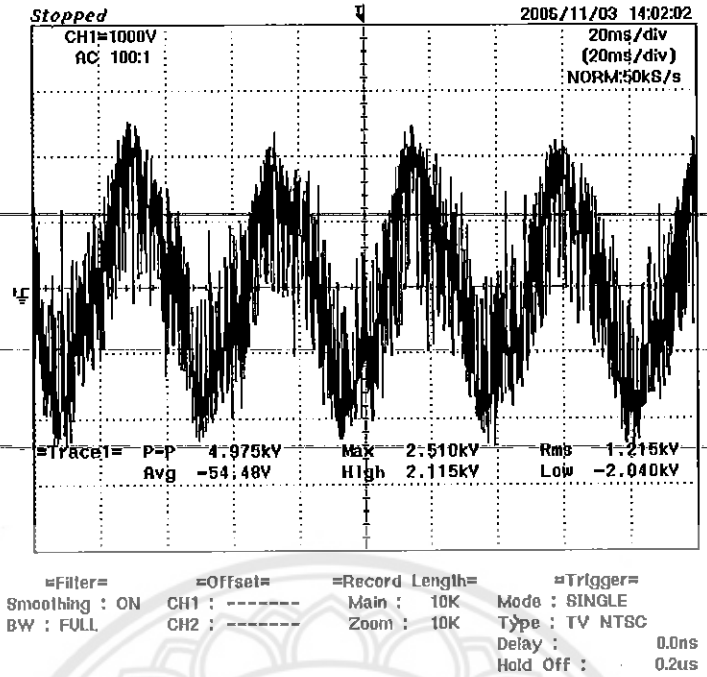
รูปที่ 5.14 สัญญาณแรงดัน V_{cn}



รูปที่ 5.15 สัญญาณกระแส i_a



รูปที่ 5.16 สัญญาณกระแส i_b



รูปที่ 5.17 สัญญาณกระแส i_c

5.4 วิเคราะห์ผลการทดลอง

5.4.1 สัญญาณ PWM

จากการวัดค่าสัญญาณ PWM ที่วัดได้ออกมาจากบอร์ดควบคุม วัดจากขา 26 (1L), 25 (1H), 24 (2L), 23 (2H), 22 (3L) และ21 (3H) สัญญาณ PWM ที่วัดได้เป็นสัญญาณ PWM (Pulse Width Modulation) ซึ่งเป็นแบบ sine wave จากทฤษฎี และมีสัญญาณ Dead Time อยู่ที่ประมาณ 0.8 μS โดย สัญญาณ PWM ที่ออกมาด้านแรงดันต่างกันจะทำงานที่แรงดันตรงข้ามกันและที่ด้านแรงดันเหมือนกันจะทำงานที่แรงดันเหมือนกันแต่สัญญาณทั้งหมดจะทำงานไม่พร้อมกันโดยมีช่วงการทำงานต่างกันเพื่อป้องกันการสวิตช์ของเพาเวอร์มอสเฟต

5.4.2 ความเร็วรอบของมอเตอร์

จากตารางผลการทดลองจะเห็นได้ว่าความเร็วรอบมอเตอร์ความเร็วรอบที่ได้จากการคำนวณกับผลการทดลองมีค่าใกล้เคียงแต่ความเร็วที่ได้จากการทดลองแต่แรงดันเฟสที่ออกมานั้นยังค่าไม่เท่ากันอันนี้เนื่องมาจากการทำงานด้านโปรแกรมยังไม่สมบูรณ์เท่าที่ควรจึงทำให้เพาเวอร์มอสเฟตทำงานไม่สมบูรณ์ไปด้วยแต่ก็ถือว่าผลการทดลองออกมาเป็นที่น่าพอใจ

5.4.3 สัญญาณแรงดันและกระแส

รูปสัญญาณของผลการทดลองที่ออกมาจะเห็นได้ชัดว่ามีสัญญาณรบกวน[1] เกิดขึ้นจำนวนมากและที่รูปสัญญาณกระแสเกิดฮาร์มอนิก [2] จำนวนมากเหมือนกันจึงทำให้รูปสัญญาณเพี้ยนไปมาก แต่อย่างไรก็ตามก็ยังเห็นว่ารูปสัญญาณแรงดันและกระแสที่ออกมาจากการทดลองยังคงเป็นรูปไซน์ตามทฤษฎี

หมายเหตุ

(1) สัญญาณรบกวน (Noise) เป็นปรากฏการณ์ที่เกิดสัญญาณไฟฟ้าที่ไม่ต้องการ และมีความถี่ต่ำกว่า 200 kHz ปะปนเข้ามาในสัญญาณของแรงดัน หรือ กระแสในสายกำลัง ซึ่งอาจจะเกิดขึ้นได้จากการที่ระบบไฟฟ้าไม่มีการต่อสายลงดิน (grounding) ที่ถูกต้องเหมาะสม ซึ่งอาจเกิดร่วมกับความผิดพลาดทางไฟฟ้าแบบอื่นด้วย ในขณะที่มีการใช้งานอุปกรณ์สวิตชิ่งอื่นๆในระบบ ผลของสัญญาณรบกวนอาจจะทำให้วงจรควบคุมทางอิเล็กทรอนิกส์ หรือ ไมโครคอนโทรลเลอร์ ทำงานผิดพลาด หรือ หยุดทำงานได้

(2) ฮาร์มอนิก (Harmonic) คือองค์ประกอบของสัญญาณที่มีรูปร่างเป็นไซน์ที่มีความถี่เป็นจำนวนเต็มเท่าของความถี่ที่สัญญาณหลักมูล (Fundamental Frequency) เช่น ความถี่ในระบบไฟฟ้าบ้านเรามีค่า 50 เฮิรตซ์ ฮาร์มอนิกของสายกำลังจะมีค่าความถี่ต่างๆขึ้นอยู่กับอันดับของฮาร์มอนิก เช่น ฮาร์มอนิกอันดับ 3 (3^{rd} Harmonic) จะมีความถี่เท่ากับ 150 เฮิรตซ์ ฮาร์มอนิกอันดับ 5 (5^{th} Harmonic) จะมีความถี่เท่ากับ 250 เฮิรตซ์ เป็นต้น ซึ่งเมื่อมีองค์ประกอบที่ฮาร์มอนิกต่างๆปะปนเข้ามาในระบบจะส่งผลให้รูปคลื่นของแรงดัน หรือ กระแส มีขนาดและเฟสเปลี่ยนไป หรือที่เราเรียกว่าเกิดความผิดเพี้ยนของรูปคลื่น (Distortion Waveform) นั่นเอง มักจะเกิดในระบบไฟฟ้าที่มีการใช้งานโหลดที่ไม่เป็นเชิงเส้น ปรากฏการณ์เช่นนี้จะมีผลให้อุปกรณ์ไฟฟ้าบางประเภท หยุดทำงาน หรือทำงานผิดพลาด และ อาจสร้างความเสียหายกับโหลด เช่น มอเตอร์ ได้ ถ้าองค์ประกอบของฮาร์มอนิกมีขนาดใหญ่มาก

บทที่ 6
สรุปผลการทดลองและข้อเสนอแนะ

6.1 สรุปผลการทดลอง

โครงงานนี้เป็นการสร้างเครื่องควบคุมความเร็วรอบมอเตอร์ไฟฟ้าเหนี่ยวนำหรือที่เรียกทั่วไปว่า อินเวอร์เตอร์ ซึ่งในโครงงานนี้ใช้ dsPIC30F2010 เป็นตัวสร้างสัญญาณ PWM

จากการทดลองการทำงานของเครื่องอินเวอร์เตอร์เครื่องนี้สามารถสรุปคุณสมบัติได้ดังนี้

1. ใช้ไฟฟ้ากระแสสลับเป็นอินพุตขนาด 220โวลต์ 1 เฟส
2. สามารถปรับความเร็วรอบด้วยการปรับความถี่ได้ตั้งแต่ 15-65 Hz.
3. สามารถขับมอเตอร์ไฟฟ้า 3 เฟส ขนาด 0.5 แรงม้า
4. ใช้การควบคุมสวิตช์มอสเฟตแบบ PWM (Pulse-Width Modulator)

และจากการทดลองสร้างเครื่องอินเวอร์เตอร์ในโครงงานนี้สรุปข้อดีได้ว่า

1. หาแหล่งจ่ายไฟง่ายเพราะใช้ไฟ 1 เฟส 220โวลต์ขับมอเตอร์เหนี่ยวนำ 3 เฟส
2. วงจรควบคุมและสร้างสัญญาณพัลส์มีขนาดเล็ก
3. ประหยัดค่าใช้จ่ายลงเนื่องจากไอซีมีราคาไม่แพงและหาซื้อได้ง่าย
4. การสร้างวงจรควบคุมไม่ยุ่งยากและซับซ้อนอย่างวงจรควบคุมที่ใช้ไอซีตัวอื่นๆสร้าง

6.2 ปัญหา ข้อเสนอแนะและแนวทางแก้ไข

1. ในโครงงานนี้การเขียนโปรแกรมด้วยภาษาซีเป็นสิ่งที่จำเป็นมากควรศึกษามาให้ดี
2. การวัดสัญญาณต่างๆต้องระวังไฟดูดเพราะโครงงานนี้ใช้ไฟ 220โวลต์
3. ในการวัดค่าสัญญาณ ระวังการช็อตเซอร์กิต (short circuit) ของกราวด์เครื่องกับกราวด์เครื่องวัด
4. อุปกรณ์จำพวกเพาเวอร์มอสเฟตเวลาสั่งซื้อควรสั่งซื้อเผื่อชำรุดด้วยอย่าสั่งมาพอดี เพราะอุปกรณ์พวกนี้ชำรุดง่าย เวลาชำรุดจะได้เปลี่ยนเลยจะได้ไม่ทำให้เสียเวลา
5. การขันน็อตระหว่างแผ่นระบายความร้อนกับเพาเวอร์มอสเฟต ควรขันให้พอดีไม่หลวมหรือแน่นเกินไปเพราะจะทำให้เพาเวอร์มอสเฟตชำรุดหรือแตกหักได้
6. ในการปรับความถี่ของเครื่องควรค่อยๆปรับไม่ควรรีบปรับจนเกินไปเพราะจะทำให้เครื่องเสียหายได้
7. สามารถปรับทิศทางการหมุนได้โดยการสลับเฟสมอเตอร์
8. โครงงานนี้เกิดสัญญาณรบกวน (Noise) จำนวนมากดังนั้นควรบรรจุกล่องมิดชิดและกั้นระหว่างวงจรอินเวอร์เตอร์กับบอร์ดไมโครคอนโทรลเลอร์เพื่อป้องกันสัญญาณรบกวนระหว่างกัน

6.3 แนวทางในการพัฒนาต่อไป

จากการทดลองในโครงงานนี้สามารถนำไปประยุกต์ใช้กับงานควบคุมมอเตอร์โดยใช้มอเตอร์ไฟฟ้าเหนี่ยวนำได้ เช่น ควบคุมความเร็วในรถรางไฟฟ้า

เอกสารอ้างอิง

[1] รศ.ดร.วีระเชษฐ์ ขันเงิน. **อิเล็กทรอนิกส์กำลัง.** พิมพ์ครั้งที่ 2. กรุงเทพฯ: หจก.วิเจ พริ้นติ้ง. 2547

[2] สุระพล เธียรมนตรี. **อิเล็กทรอนิกส์กำลัง.** พิมพ์ครั้งที่ 1. กรุงเทพฯ: 2545

[3] พรจิต ประทุมสุวรรณ. **พื้นฐานการขับเคลื่อนมอเตอร์ไฟฟ้าด้วยอิเล็กทรอนิกส์กำลัง.** กรุงเทพฯ: เรือนแก้วการพิมพ์. 2547

[4] ยืน ภู่สุวรรณ. **อิเล็กทรอนิกส์อุตสาหกรรม.** พิมพ์ครั้งที่ 2. กรุงเทพฯ: ซีเอ็ดยูเคชั่น. 2525

[5] MUHAMMAD H.RAHID. **POWER ELECTRICS.** NEW JERSEY: PRINTICE, 1997

[6] นคร ภักดีชาติ, ชัยวัฒน์ ลิ้มพรจิตรวิไล. **คู่มือการทดลอง dsPIC Microcontroller เบื้องต้นด้วยโปรแกรมภาษาซี กับ MPLAB C30.** กรุงเทพฯ: อินโนเวตีฟ เอ็กเพอริเมนต์. 2547

ภาคผนวก

ภาคผนวก ก

โปรแกรมกำเนิดสัญญาณพัลส์ของ dsPIC30F2010

โปรแกรม

```
//-------------------------------------------------------------------------------------------------------------//
// Program              : Test UART1
// Description          : Gets character and send to display to terminal program(Echo)
// Frequency            : 7.3738 MHz at PLL 4x
// Filename             : uart_01.c
// C compiler           : C30 Compiler by Microchip Technology
//-------------------------------------------------------------------------------------------------------------//
#include<p30f2010.h>  // Header file for dsPIC30F2010
#include<uart.h>      // Module function for uart
#include <stdio.h>
#include <pwm.h>
#include <math.h>
#include<adc10.h>     // Module function for 10 bit ADC
#define re1  LATDbits.LATD0
#define run  LATDbits.LATD1

//-----------------------------------------------------------time
#include<timer.h>             // Module function for Timer

unsigned char deat,deat_b;
unsigned int index1,index2,index3,VOL;
unsigned long level,level_b;
unsigned int test;

unsigned char table[360]={0x7f,0x81,0x83,0x86,0x88,0x8a,0x8c,0x8f,0x91,0x93,
0x95,0x97,0x9a,0x9c,0x9e,0xa0,0xa2,0xa4,0xa6,0xa9,
0xab,0xad,0xaf,0xb1,0xb3,0xb5,0xb7,0xb9,0xbb,0xbd,
0xbf,0xc1,0xc3,0xc4,0xc6,0xc8,0xca,0xcc,0xcd,0xcf,
0xd1,0xd3,0xd4,0xd6,0xd8,0xd9,0xdb,0xdc,0xde,0xdf,
0xe1,0xe2,0xe3,0xe5,0xe6,0xe7,0xe9,0xea,0xeb,0xec,
```

```
0xed,0xef,0xf0,0xf1,0xf2,0xf3,0xf3,0xf4,0xf5,0xf6,
0xf7,0xf8,0xf8,0xf9,0xfa,0xfa,0xfb,0xfb,0xfc,0xfc,
0xd1,0xcf,0xce,0xcc,0xca,0xc8,0xc6,0xc4,0xc3,0xc1,0xbf,0xbd,0xbb,0xb9,0xb7,0xb5,0xb3,
0xb1,0xaf,0xad,0xab,0xa9,0xa6,0xa4,0xa2,0xa0,0x9e,0x9c,0x9a,0x97,0x95,0x93,0x91,0x8f,
0x8c,0x8a,0x88,0x86,0x83,0x81,0x7f,0x7d,0x7b,0x78,0x76,0x74,0x72,0x6f,0x6d,0x6b,0x69,
0x67,0x65,0x62,0x60,0x5e,0x5c,0x5a,0x58,0x56,0x53,0x51,0x4f,0x4d,0x4b,0x49,0x47,0x45,
0x43,0x41,0x3f,0x3d,0x3b,0x3a,0x38,0x36,0x34,0x32,0x31,0x2f,0x2d,0x2b,0x2a,0x28,0x26,
0x25,0x23,0x22,0x20,0x1f,0x1d,0x1c,0x1b,0x19,0x18,0x17,0x15,0x14,0x13,0x12,0x11,0x0f,
0x0e,0x0d,0x0c,0x0b,0x0b,0x0a,0x09,0x08,0x07,0x06,0x06,0x05,0x04,0x04,0x03,0x03,0x02,
0x02,0x01,0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x01,0x01,0x01,0x02,0x02,0x03,0x03,0x04,0x04,0x05,0x06,0x06,0x07,0x08,0x09,
0x0a,0x0b,0x0b,0x0c,0x0d,0x0e,0x0f,0x11,0x12,0x13,0x14,0x15,0x17,0x18,0x19,0x1b,0x1c,
0x1d,0x1f,0x20,0x22,0x23,0x25,0x26,0x28,0x2a,0x2b,0x2d,0x2f,0x30,0x32,0x34,0x36,0x38,
0x3a,0x3b,0x3d,0x3f,0x41,0x43,0x45,0x47,0x49,0x4b,0x4d,0x4f,0x51,0x53,0x55,0x58,0x5a,
0x5c,0x5e,0x60,0x62,0x64,0x67,0x69,0x6b,0x6d,0x6f,0x72,0x74,0x76,0x78,0x7b,0x7d};

//------------------------------------ Interrupt service routine for timer3-----------------------------------//
//-----------------------------------------------------------------------------------------------------------//
void _ISR _T3Interrupt(void)
{
SetDCMCPWM(1,(table[index1]*2),0);
SetDCMCPWM(2,(table[index2]*2),0);
SetDCMCPWM(3,(table[index3]*2),0);

index1=index1+2;
index2=index2+2;
index3=index3+2;
}
//------------------------------------ Function initialize ACD module ----------------------------------------//
//-----------------------------------------------------------------------------------------------------------//
void adc_init()
```

```
{
    unsigned int Channel, PinConfig, Scanselect, Adcon3_reg, Adcon2_reg,
        Adcon1_reg;
    ADCON1bits.ADON = 0;                    // Turn off ADC
    Channel = ADC_CH0_POS_SAMPLEA_AN0 &// Channel 0 positive input select AN0
        ADC_CH0_POS_SAMPLEA_AN1 &   // Channel 0 positive input select AN1
        ADC_CH0_POS_SAMPLEA_AN2 &   // Channel 0 positive input select AN2
        ADC_CH0_POS_SAMPLEA_AN3 &   // Channel 0 positive input select AN3
        ADC_CH0_NEG_SAMPLEA_NVREF ;// Channel 0 negative VREF

    SetChanADC10(Channel);              // Set channel configuration
    ConfigIntADC10(ADC_INT_DISABLE);    // Disable interrupt for ADC
    PinConfig       = ENABLE_AN0_ANA &  // Enable AN0-AN3 analog port
                      ENABLE_AN1_ANA &
                      ENABLE_AN2_ANA &
                      ENABLE_AN3_ANA ;

    Scanselect      = SKIP_SCAN_AN4 &           // Scan for AN0-AN3
                      SKIP_SCAN_AN5 &
                      SKIP_SCAN_AN6 &
                      SKIP_SCAN_AN7;

    Adcon3_reg      = ADC_SAMPLE_TIME_30 &      // Sample for 10 time
                ADC_CONV_CLK_INTERNAL_RC &      // Internal Clock
                      ADC_CONV_CLK_13Tcy;

    Adcon2_reg      = ADC_VREF_AVDD_AVSS &      // Vref at Vdd and Vss
                      ADC_SCAN_ON &             // Enable scan for ADC
                      ADC_ALT_BUF_OFF &         // Disable alternate buffer
                ADC_ALT_INPUT_OFF &         // Disable alternate input
                      ADC_CONVERT_CH0&          // Select CH0 convert
```

```
                    ADC_SAMPLES_PER_INT_16;        // 16 sample between interrupt

        Adcon1_reg  = ADC_MODULE_ON &              // Enable module ADC
                    ADC_IDLE_CONTINUE &            // ADC run on idle mode
                    ADC_FORMAT_INTG &              // Output value integer format
                    ADC_CLK_MANUAL &               // ADC manual clock
                    ADC_SAMPLE_SIMULTANEOUS &      // ADC sampling simultaneous
                    ADC_AUTO_SAMPLING_ON;          // ADC auto sampling

        OpenADC10(Adcon1_reg, Adcon2_reg, Adcon3_reg,PinConfig, Scanselect); // Turn on
ADC module
}
//-----------------------------------------------------------------------------------------//
//---------------------------------- Interrupt service routine  TX UART1 ------------------//
//-----------------------------------------------------------------------------------------//
void _ISR _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;   // Clear TX interrupt flag
}
//-----------------------------------------------------------------------------------------//
//---------------------------------- Interrupt service routine  RX UART1 ------------------//
//-----------------------------------------------------------------------------------------//
void _ISR _U1RXInterrupt(void)
{
        IFS0bits.U1RXIF = 0;   // Clear RX interrupt flag
}
//-----------------------------------------------------------------------------------------//
//---------------------------------- Function for configuration UART1 ---------------------//
//-----------------------------------------------------------------------------------------//
void uart1_init()
{
```

```
unsigned int baudvalue;                 // Keep baud rate value for Load into U1BRG
unsigned int U1MODEvalue;   // Keep value for Load into U1MODE
unsigned int U1STAvalue;                // Keep value for Load into U1STA

CloseUART1();                   // Disable UART1
ConfigIntUART1( UART_RX_INT_EN &        // Enable RX interrupt UART1
                UART_RX_INT_PR6 &       // Set RX interrupt Priority ==>6
                UART_TX_INT_EN          // Enable TX interrupt UART1
                &UART_TX_INT_PR2);      // Set RX interrupt Priority ==>2
baudvalue = 47;         // Baud rate 9600 bps
U1MODEvalue =   UART_EN &                       // Enable UART1
                UART_IDLE_CON &  // UART1 working in idle mode
                UART_RX_TX&     //      UART1      normal
pin(TX==>RF3 pin,RX==>RF2 pin)
                UART_DIS_WAKE & // Disable wake-up on start
UART
                UART_DIS_LOOPBACK &// Disable loop back mode

                UART_DIS_ABAUD & // Disable autobaud mode
                UART_NO_PAR_8BIT &// Set data 8 bit ,no parity bit
                UART_1STOPBIT; // Set 1 stop bit

U1STAvalue  = UART_INT_TX_BUF_EMPTY &   // Interrupt on buffer empty mode
              UART_TX_PIN_NORMAL &      // TX Break bit normal
              UART_TX_ENABLE &          // Enable TX for UART
              UART_INT_RX_3_4_FUL&      // UART interrupt receive mode
              UART_ADR_DETECT_DIS &     // Disable detect address mode
              UART_RX_OVERRUN_CLEAR;    // Reset buffer over run

OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);// Execute configuration for UART1
}
```

```
//-------------------------------------------------------------------------------------------------//
//---------------------------------------- Main Program ----------------------------------------//
//-------------------------------------------------------------------------------------------------//
int main()
{ // char _1[] ="12415415212512"
  index1=0;
  index2=120;
  index3=60;
        char Txdata[] = "\r\nUart iup  key...ERROR \r\n";       // Table character
        char  dat;                  // Buffer for receive character
//  unsigned int num,num2;
//-------------------------------------------------------------------------AD
unsigned int result[4], old_result[4],i;    // Keep data
adc_init();                                 // Initial ADC
        uart1_init();               // Initial UART1
        putsUART1 ((unsigned int *)Txdata);             // Send string into terminal
        while(BusyUART1());             // Wait until success

//-------------------------------------------------------------------------PWM
//PWM_IPCLK_SCALE64 =0xA0FF;
        unsigned int config1,config2,config3;
        unsigned int config;
 //  unsigned char updatedisable;

ConfigIntMCPWM(PWM_INT_EN&PWM_FLTA_EN_INT&0x00);
SetMCPWMFaultA(PWM_OVA1H_ACTIVE &
                        PWM_OVA2H_ACTIVE &
                        PWM_FLTA_MODE_LATCH &
                        PWM_FLTA1_EN &
                                        PWM_FLTA3_EN &
                        PWM_FLTA2_EN);
```

```
//sptime=0;
        config1=(PWM_EN&PWM_OP_SCALE1&PWM_IPCLK_SCALE1&PWM_MOD_D
BL);
        config2=(PWM_MOD1_COMP&
                        PWM_MOD2_COMP&
                PWM_MOD3_COMP&
                PWM_PEN2H&PWM_PEN2L&
                PWM_PEN1H&PWM_PEN1L&
                PWM_PEN3H&PWM_PEN3L);
        config3=(PWM_SEVOPS1&PWM_OSYNC_PWM&PWM_UEN);
    OpenMCPWM(255,0x00,config1,config2,config3);    //ตัวหน้า 170-20.97 k

///----------------------------------------------------------------------TIMMER
 TRISBbits.TRISB4=0 ;
                // Configuration for RD1 to output
TRISDbits.TRISD0 = 0;               // Configuration for RD1 to output
TRISDbits.TRISD1 = 0;               // Configuration for RD1 to output
        ConfigIntTimer23(T2_INT_PRIOR_1 &       // Timer1 interrupt priority 1
                        T2_INT_ON);             // Enable interrupt for timer1
        WriteTimer23(0);                        // Clear count value at TMR1 register

        OpenTimer23(T2_ON &                     // Start timer2/3
                        T2_GATE_OFF &           // Disable gate pin for timer2/3
                        T2_IDLE_STOP &          // Stop timer2/3 in idle mode
                        T2_PS_1_1 &             // Presceler 1:1
                        T2_32BIT_MODE_ON &      // Enable mode 32 bit Timer2/3
                        T2_SOURCE_INT,7372800);

//-------------------------------------------------------------------------------------------------------------
test=0;
deat=60;
```

```
level_b=7370000;
deat=60;
VOL=70;
while(1)                        // Infinite loop
    {
  for(i=0;i<4;i++)              // Loop 4 time for read ADC keep to result array
        {
            ADCON1bits.SAMP = 1;          // Start Sampling
            result[i] = ReadADC10(i);     // Keep value for ADC value
        }
//------------------------------------------------------------------------------ ADC print
   putsUART1((unsigned int *)"\r\n RUN INVERTER");
if(test>=250){
      if(result[0] > 1000){run=0;}else{run=1;}
      deat=result[1]/15;
          if(level > level_b){level_b=level_b+3;}
       if(level < level_b){level_b=level_b-3;}
       test=250;
          }
else{
   if(test >= 100) {  re1= 1; }
   if(result[0] < 1015){  if(test >= 30){PORTDbits.RD1 = 1;} }
   }
 test++;
      if(result[0] < 160){result[0]=160;}//60hz
       level=7372730-((result[0]*3)+84);
       if(level > 7372700){level=7372700;}

//------------------------------------------------------------------------------VOL
 VOL=0;
      if( (result[1]/2) < (180) ){if(deat < 20  ){VOL=20;}else { deat++;  } }
}
```

```
        if( (result[1]/2) > (180) ){if(deat  > 59 ){VOL=60;}  else{ deat--; }
        SetMCPWMDeadTimeGeneration(deat);//??? ??????? 0-60

        }
        return 0;
    }
```

ภาคผนวก ข

แนะนำฟังก์ชั่นและการเขียนโปรแกรมภาษา C สำหรับ MPLAB C30

แนะนำฟังก์ชั่นและการเขียนโปรแกรมภาษา C สำหรับ MPLAB C30

ในการพัฒนาโปรแกรมภาษา C สำหรับควบคุมไมโครคอนโทรลเลอร์ dsPIC ด้วย MPLAB C30 คอมไพเลอร์นั้น รูปแบบไวยากรณ์หลักๆมีลักษณะไปในทำนองเดียวกับมาตรฐาน ANCI C อันประกอบด้วย การประกาศตัวแปร, ชุดคำสั่งควบคุม, โอเปอร์เรเตอร์ต่างๆ, รูปแบบการใช้งานฟังก์ชัน เป็นต้น โดยการทำงานจะเริ่มต้นที่ฟังก์ชัน main () หรือที่เรียกว่า โปรแกรมหลัก ดังตัวอย่างต่อไปนี้

```
main (viod)
{                                   // statement of main are enclosed in braces
int  x,y,result                     // defines 3 variables of type int
x = 0×25;                           /* Assign 0×25 to variables x */
y = 0×0F;                           /* Assign 0×0F to variables y */
result = x & y;                     /* And x with y and assigns it to result */
return 0;                           // Retern 0 to the caller of main
}                                   // End of main function
```

บรรทัดแรกเป็นประกาศจุดเริ่มต้นของโปรแกรม ในบรรทัดถัดมาเป็นวงเล็บปีกกาเปิด {อันเป็นประกาศให้ทราบว่า บรรทัดแรกของโปรแกรมในฟังก์ชัน main เริ่มที่นี่ ซึ่งในโปรแกรมภาษา C ทุกโปรแกรมจะมี และประกาศฟังก์ชัน main () ได้เพียงหนึ่งฟังก์ชันเท่านั้น

เครื่องหมาย // กับ /* และ */ ถูกใช้ในการระบุตำแหน่งของคำอธิบายโปรแกรมหรือคอมเมนต์ (comment) โดย // จะถูกใช้ในคำอธิบายโปรแกรมภายในบรรทัดเดียวกัน โดยข้อความหรือตัวอักษรที่อยู่ถัดจากเครื่องหมาย // นี้ จะไม่ได้รับการคอมไพล์ จนกว่าจะขึ้นบรรทัดใหม่ ในขณะที่ /* และ */ ต้องใช้ร่วมกัน โดยคอมไพเลอร์จะไม่ตีความหรือแปลข้อความที่อยู่ระหว่างเครื่องหมาย /* และ */

จากนั้นอีก 5 บรรทัดเป็นรายละเอียดของโปรแกรม โดยในทุกคำสั่งของโปรแกรมภาษา C จะต้องปิดท้ายด้วยเครื่องหมายเซมิโคลอน (;) เสมอ

บรรทัดสุดท้ายเป็นวงเล็กปีกกาปิด } ใช้แจ้งตำแหน่งบรรทัดสุดท้ายของฟังก์ชัน

1. กระบวนการทำงานของ MPLAB C30 คอมไพเลอร์

MPLAB C30 คอมไพเลอร์ เป็นตัวแปรภาษา C หรือ C คอมไพเลอร์ที่พัฒนาต่อจาก GCC คอมไพเลอร์ โดยมีกระบวนการทำงานที่เป็นไปตามข้อกำหนดในมาตรฐาน ANSI x3.159-1989 โดยไฟล์ผลลัพธ์ที่ได้ตะนำไปใช้ในการโปรแกรมลงในไมโครคอนโทรลเลอร์ dsPIC

ซอร์สโปรแกรมภาษา C ที่บันทึกเป็นไฟล์ .c ได้รับการแปลงไฟล์ภาษาแอสเซมบลีของ dsPIC ซึ่งเป็นไฟล์ .S ด้วย C คอมไพเลอร์ จากนั้นจะได้รับการแอสเซมเบลอร์ร่วมกับซอร์สโปรแกรม

ภาษาแอสเซมบลีตัวอื่นที่มาจากไฟล์ผนวก (include file) หรือจากการแทรกโปรแกรมภาษาแอสเซมบลี จะได้เป็นไฟล์ออปเจ็กในแบบ COFF (Common Object File Format) ซึ่งเป็นไฟล์ .o จากนั้นจะเข้าสู่กระบวนการเชื่อมโยงกับไฟล์ไลบรารีต่างๆด้วย MPLABLINK30 ลิงเกอร์ เพื่อสร้างไฟล์เอ็กซิคิวต์ COFF หรือไฟล์ออปเจ็กที่สามารถนำไปใช้งานได้ ซึ่งบันทึกได้เป็นไฟล์ .cof

จากไฟล์ .cof สามารถนำไปใช้ได้ 2 ทาง คือ

(1) ส่งต่อไปยัง MPLAB IDE เพื่อทำการทดสอบและดีบัก

(2) นำไปแปลงเป็นไฟล์ .hex ในรูปแบบของ intel HEX เพื่อดาวน์โหลดไปยังหน่วยความจำโปรแกรมภายใน dsPIC หรือส่งไปยังโปรแกรมการทำงาน (คอมมานด์ไลน์ซิมูเลเตอร์)

2. ชนิดข้อมูลที่ MPLAB C30 สนับสนุน

ชนิดข้อมูล	คำอธิบาย
char, signed char	ชนิดข้อมูลจำนวนเต็ม 8 บิต มีค่า -128 ถึง 127
unsigned char	ชนิดข้อมูลจำนวนเต็ม 8 บิต มีค่า 0 ถึง 255
short, signed short	ชนิดข้อมูลจำนวนเต็ม 16 บิต มีค่า -32,768 ถึง 32,767
unsigned short	ชนิดข้อมูลจำนวนเต็ม 16 บิต มีค่า 0 ถึง 65,535
int, signed int	ชนิดข้อมูลจำนวนเต็ม 16 บิต มีค่า -32,768 ถึง 32,767
unsigned int	ชนิดข้อมูลจำนวนเต็ม 16 บิต มีค่า 0 ถึง 65,535
long, signed long	ชนิดข้อมูลจำนวนเต็ม 32 บิต มีค่า -2^{31} ถึง $2^{31}-1$
unsigned long	ชนิดข้อมูลจำนวนเต็ม 32 บิต มีค่า 0 ถึง $2^{31}-1$
long long, signed long long	ชนิดข้อมูลจำนวนเต็ม 64 บิต มีค่า -2^{63} ถึง $2^{63}-1$
unsigned long long	ชนิดข้อมูลจำนวนเต็ม 64 บิต มีค่า 0 ถึง $2^{64}-1$
float	ชนิดข้อมูลทศนิยม 32 บิต มีค่า1.175×10^{-38} ถึง 3.403×10^{38}
double	ชนิดข้อมูลทศนิยม 32 บิต มีค่า1.175×10^{-38} ถึง 3.403×10^{38}
long double	ชนิดข้อมูลทศนิยม 64 บิต 2.225×10^{-308} ถึง 1.798×10^{308}
pointer	ชนิดตัวแปรชี้ตำแหน่งข้อมูล
array	ชนิดตัวแปรอะเรย์
structure	ชนิดตัวแปรโครงสร้าง

3. การประกาศตัวแปร

การประกาศตัวแปรในภาษา C ทำได้ด้วยการระบุชนิดข้อมูลแล้วตามด้วยการระบุชื่อตัวแปร รูปแบบ

```
type variable_name;
```

โดยที่ type คือ ชนิดข้อมูลที่ต้องการกำหนด

variable_name คือ ชื่อตัวแปรที่ประกาศ

เช่น

```
char x;                 // ประกาศตัวแปรชื่อ x เป็นชนิด char
int j;                  // ประกาศตัวแปรชื่อ j เป็นชนิด int
float mc;               // ประกาศตัวแปรชื่อ mc เป็นชนิด float
```

นอกจากนี้ยังสามารถกำหนดค่าเริ่มต้นของตัวแปรในส่วนของการประกาศได้ดังนี้

```
int y = 0x39;           // ประกาศตัวแปรชื่อ y เป็นชนิด int กำหนดค่าเริ่มต้นเป็น 0x39
float result = 26.54;   // ประกาศตัวแปรชื่อ result เป็นชนิด float กำหนดค่า เริ่มต้นเป็น 26.54
char echo = 'z' ;       // ประกาศตัวแปรชื่อ echo เป็นชนิด charกำหนดค่าเริ่มต้นให้เป็นรหัสแอส
                        กี้ที่ตัวกับอักขระ 'z'
```

3.1 ตัวแปรแบบอะเรย์

เมื่อต้องการประกาศชุดตัวแปรที่มีชนิดเดียวกัน ควรใช้การประกาศตัวแปรแบบอะเรย์ ดังนี้

```
char name [5] ;
        // ประกาศตัวแปรอะเรย์ชื่อ name มีสมาชิกทั้งหมด 5 ตัว คือ
        // name [0] ถึง name [4] โดยที่สมาชิกทุกตัวมีชนิดข้อมูลเป็น char
char string [12] = "Hello world"
```

ประกาศเป็นตัวแปรอะเรย์ชื่อ string มีสมาชิกทั้งสิ้น 15 ตัว คือ name [0] ถึง name [14] โดยที่สมาชิกทุกตัวมีสมาชิกข้อมูลเป็น char เรียกข้อมูลชนิดนี้ว่า "สายอักขระ" โดยมีการกำหนดค่าเริ่มต้นพร้อมกับประกาศ ดังนั้นหลังจากประกาศ ค่าของ string[0] ถึง string[11] เท่ากับ

```
string[0] = 'H'
string[1] = 'e'
string[2] = 'l'
string[3] = 'l'
string[4] = 'o'
string[5] = ' '
string[0] = 'w'
string[0] = 'o'
```

```
string[0] = 'r'
string[0] = 'l'
string[0] = 'd'
string[0] = 0
```

3.2 ตัวแปรแบบโครงสร้าง

เมื่อต้องการจัดกลุ่มตัวแปรที่มีการทำงานสอดคล้องกัน ทำได้โดยการประกาศตัวแปรแบบโครงสร้าง ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 1

```
struct var              // ประกาศตัวแปรโครงสร้างชื่อ var
{
    char a ;            // สมาชิกของ var ชื่อตัวแปร a มีชนิดข้อมูลเป็นตัวอักขระ
    int b ;             // สมาชิกของ var ชื่อตัวแปร b มีชนิดข้อมูลเป็นเลขจำนวนเต็ม
    float c ;           // สมาชิกของ var ชื่อตัวแปร c มีชนิดข้อมูลเป็นเลขทศนิยม
}
```

จากการประกาศตัวแปรโครงสร้าง var ในตัวอย่างที่ A1-1 เป็นการสร้างชนิดข้อมูลใหม่ขึ้นมาและสามารถสำเนาชนิดข้อมูลโครงสร้างแบบ var ไปใช้งานได้ โดยประกาศตัวแปรได้ดังนี้

ตัวอย่างที่ 2

```
struct var var1 ;       // กำหนดให้ var1 เป็นตัวแปรโครงสร้างแบบ var (ที่ประกาศก่อนหน้านี้)
                        // ดังนั้น var1 มีสมาชิกคือ a,b,c ที่เข้าถึงได้ด้วยโอเปเรเตอร์ "." (จุด)
var1.a = 'x' ;          // กำหนดค่าข้อมูลของสมาชิก a ใน var1 เป็นอักขระ x (char)
var1.b = 5132 ;         // กำหนดค่าข้อมูลของสมาชิก b ใน var1 เป็นอักขระ 5132 (int)
var1.c = 42.69 ;        // กำหนดค่าข้อมูลของสมาชิก c ใน var1 เป็นอักขระ 42.69 (float)
```

3.3 ตัวแปรแบบพอยน์เตอร์

เมื่อมีความจำเป็นต้องเข้าถึงที่อยู่ของข้อมูลหรือตำแหน่งแอดเดรสของข้อมูล ก็สามารถประกาศใช้งานตัวแปรชนิดพอยน์เตอร์ได้ดังนี้

ตัวอย่างที่ 3

```
char *p ;       // ประกาศตัวแปรพอยน์เตอร์ชื่อ p เป็นชนิดอักขระสำหรับเก็บค่าแอดเดรส
char x=120 ;    // ประกาศชื่อ x เป็นชนิดอักขระและกำหนดค่าเริ่มต้นเป็น 120
char y ;        // ประกาศชื่อ y เป็นชนิดอักขระ
p = & x ;       // โหลดค่าแอดเดรสตัวแปร x ไปยังพอยน์เตอร์ p
y = *p ;        // โหลดค่าข้อมูลในแอดเดรสที่พอยน์เตอร์ p ชี้อยู่ไปยัง y ทำให้ y = 120
```

4. การกำหนดค่าคงที่แบบมาโคร

เมื่อต้องการกำหนดค่าคงที่แบบมาโครให้กับโปรแกรม ทำได้โดยใช้คีย์เวิร์ด #define สามารถกำหนดได้ทั้งแบบค่าคงที่และแบบกำหนดเป็นนิพจน์ดังนี้

ตัวอย่างที่ 4

```
#define num 15          // กำหนด num แทนด้วยตัวเลข 15
#define sum (x,y) x+y   // กำหนดให้ sum (x,y) ถูกแทนด้วยนิพจน์ x+y
```

5. ตัวดำเนินการทางคณิตศาสตร์

สรุปได้ดังนี้

โอเปอเรเตอร์ (Operater)	ความหมาย
+	การบวก
-	การลบ
*	การคูณ
/	การหาร
%	การหารแบบเอาเศษหรือโมดูโล (modulo)
++	การเพิ่มค่าขึ้นอีกหนึ่งค่า
--	การลดค่าลงอีกหนึ่งค่า
+=	การบวกขึ้นอีกเท่ากับค่าทางขวามือ
-=	การลดลงอีกเท่ากับค่าทางขวามือ
*=	การคูณด้วยเท่ากับค่าทางขวามือ
/=	การหารด้วยเท่ากับค่าทางขวามือ
%=	การหารด้วยเท่ากับค่าทางขวามือแบบเอาเศษ

ตัวอย่างที่ 5

```
int main ()
{
int x=2 ;       // กำหนดให้ x เท่ากับ 2                    (x=2)
x++ ;           // เพิ่มค่า x ขึ้น 1 ค่า                   (x=3)
x*=6            // คูณค่า x ด้วย 6 แล้วเก็บค่าไว้ที่ x      (x=18)
x%=4            // หาร x ด้วย 4 แล้วเก็บค่าเศษไว้ที่ x      (x=2)
}
```

6. ตัวดำเนินการตรวจสอบเงื่อนไข

เป็นตัวกำเนินการที่ใช้ร่วมกับคำสั่งควบคุม เช่น if, for, while ในการตรวจสอบเงื่อนไข มีดังนี้

โอเปอเรเตอร์ (Operater)	ความหมาย
= =	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าหรือเท่ากับ
<=	น้อยกว่าหรือเท่ากับ
!	ไม่ใช่ (NOT)
&&	และ (AND)
||	หรือ (OR)

7. ตัวดำเนินการทางบิต (Bitwise Operation)

การกระทำในกลุ่มนี้ เป็นการกระทำที่เข้าถึงในระดับบิตของข้อมูล โดยที่ค่าของแต่ละบิตเป็นได้ 2 ค่าคือ "1" หรือ "0" สามารถสรุปได้ดังนี้

โอเปอเรเตอร์ (Operater)	ความหมาย
= =	เท่ากับ
~	กลับค่าของบิตข้อมูล
&	การ AND แบบบิต
|	การ OR แบบบิต
^	การ exclusive OR แบบบิต
<<	เลื่อนบิตไปทางซ้าย
>>	เลื่อนบิตไปทางขวา
<<=	เลื่อนบิตไปทางซ้ายแล้วให้เท่ากับ
>>=	เลื่อนบิตไปทางขวาแล้วให้เท่ากับ
&=	แอนด์ (AND) แล้วให้เท่ากับ
|=	ออร์(OR)แล้วให้เท่ากับ
^=	เอ็กคลูซีฟ-ออร์(exclusive)แล้วให้เท่ากับ

ตัวอย่างที่ 6

```
FORTB = PORTB | 0×40;        // นำค่าข้อมูลรีจิสเตอร์ PORTB มาออร์กับค่า 0×40
```

```
                                    // แล้วเก็บค่าผลลัพธ์ที่ได้ไปยังรีจิสเตอร์ PORTB
```

ตัวอย่างที่ 7

```
LATD = var << 4;                    // นำค่าข้อมูลของตัวแปร var มาเลื่อนบิตไปทางซ้าย 4 ครั้ง
                                    // แล้วเก็บค่าผลลัพธ์ที่ได้ไปยังรีจิสเตอร์ LATD
```

ตัวอย่างที่ 8

```
a = a ^ 0×F0;                       // นำค่าข้อมูลของตัวแปร a มาเอ็กคลูซีฟ-ออร์กับค่า 0×F0
                                    // แล้วเก็บค่าผลลัพธ์ที่ได้ไปยังตัวแปร a
```

ตัวอย่างที่ 9

```
dat &= 0xAA;                        // นำค่าของตัวแปร datมาแอนด์กับค่า 0xAA เก็บค่าผลลัพธ์
                                    ที่ได้ไปยังตัวแปร dat
```

ตัวอย่างที่ 10

```
PORTDbits.RD1 =! PORTDbits.RD1;     // นำข้อมูลบิต 1 ของพอร์ต D กลับค่าลอจิก
```

8. คำสั่งควบคุมในภาษา C

8.1 คำสั่ง if และ if-else

ใช้เมื่อต้องการให้โปรแกรมตัดสินใจเลือกในหลายๆทางเลือกที่มีการกำหนดไว้

รูปแบบ

```
if (expression_1)
        statement_1
else if (expression_2)
        statement_2
else if (expression_3)
        statement_3
        ...
else
        statement_n
```

โดยที่ expression_1 ถึง expression_n คือ เงื่อนไขสำหรับในแต่ละทางเลือกทำโปรแกรม

statement_1 ถึง statement_n คือ ชุดคำสั่งสำหรับตอบสนองเมื่อ expression ของเงื่อนไขนั้นๆเป็นจริง

ตัวอย่างที่ 11

```
if (a != 0)
```

```
  PORTD = 0x55;
else
  PORTD = 0xAA; ;
```

ตัวอย่างนี้เป็นการตัดสินใจที่มีทางเลือก 2 ทาง โดยพิจารณาจากค่าของตัวแปร a

ถ้า a มีค่าไม่เป็น 0 พอร์ต D จะถูกกำหนดค่าเป็น 0x55

ถ้า a มีค่าเป็น 0 พอร์ต D จะถูกกำหนดค่าเป็น 0xAA

ตัวอย่างที่ 12

```
if (x == 1)
  return (10) ;
else if (x == 5)
  return (20) ;
else
  return (-1) ;
```

ตัวอย่างนี้เป็นการตัดสินใจที่มีทางเลือก 3 ทาง โดยพิจารณาค่าตัวแปร x

ถ้า x = 1 ให้คืนค่าเป็น 10

ถ้า x = 5 ให้คืนค่าเป็น 20

นอกเหนือจากนี้ให้คืนค่าเป็น -1

8.2 คำสั่ง switch-case

เมื่อต้องการตัดสินใจเลือกในหลายๆทาง นอกจากคำสั่ง if และ if-else ผู้พัฒนาอาจเลือกใช้คำสั่ง switch-case ช่วยได้ แต่มีข้อจำกัดคือ สามารถใช้ตรวจสอบค่าตัวแปรหรือค่าคงที่เท่านั้น

รูปแบบ

```
switch (expression)
{
        case   const_1:  statement_1;
                 break;
        case   const_2:  statement_2;
                 break;
        ...
        default          :        statement_n;
}
```

โดยที่ expression คือข้อมูลที่นำมาพิจารณาว่าตรงกับ case ใด ก็จะทำตามคำสั่งที่กำหนดใน statement นั้นๆ

break คือ คำสั่งออกจาก case หลังจากทำงานที่ statement เรียบร้อยแล้ว

default คือ เงื่อนไขนอกเหนือจาก case ต่างๆที่ระบุ ถ้าไม่ตรงกับ case ใดๆเลยจะเข้าสู่ statement_n (อาจไม่กำหนดก็ได้)

ตัวอย่างที่ 13

```
switch (i)
{
case 1 : x = '1' ;
   break ;
case 1 : x = '2' ;
   break ;
case 1 : x = '3' ;
   break ;
case 1 : x = 'E' ;
   break ;
}
```

ตัวอย่างนี้เป็นการตัดสินใจที่มีทางเลือก 4 ทาง โดยพิจารณาจากค่าตัวแปร i

ถ้า i = 1 ให้กำหนดค่า x เป็นอักขระ 1 (หรือ 0×31)

ถ้า i = 2 ให้กำหนดค่า x เป็นอักขระ 1 (หรือ 0×32)

ถ้า i = 3 ให้กำหนดค่า x เป็นอักขระ 1 (หรือ 0×33)

นอกเหนือจากนี้ กำหนดค่า x เป็นอักขระ E (หรือ 0×45)

8.3 คำสั่ง for

เมื่อต้องการให้โปรแกรมมีการทำงานวนรอบโดยการตัดสินใจจากเงื่อนไขที่กำหนด ซึ่งโดยมากมักจะทราบค่าจำนวนรอบของการทำงาน (หรืออาจไม่ทราบก็ได้) ผู้พัฒนาอาจเลือกใช้คำสั่ง for ในการจัดการ

รูปแบบ

```
for (initialize ; condition ; incremental)
   statement ;
```

โดยที่ initialize คือ ค่าเริ่มต้นที่กำหนดจากตัวแปรที่นำมาเป็นเงื่อนไขในการทำงานแบบวนรอบ

condition คือ เงื่อนไขที่ใช้ในการตรวจสอบว่าจะให้โปรแกรมทำคำสั่งภายใน statement หรือไม่ ถ้าเป็นจริงจะทำคำสั่งใน statement แต่ในทางกลับกันถ้าเป็นเท็จจะไม่มีการทำคำสั่งใน statement

incremental คือ คำสั่งในการกระทำกับตัวแปรที่นำมาเป็นเงื่อนไข

ตัวอย่างที่ 14

```
sum = 0;
for (x = 0;x<10;x++)
sum = sum+x;
```

ตัวอย่างนี้จะเป็นการบวกค่าข้อมูลของตัวแปร sum กับตัวแปร x โดยทุกครั้งจะเก็บค่าผลลัพธ์ไว้ที่ sum เสมอ โดยจะมีการบวกทั้งสิ้น 10 ครั้ง x = 0...9 นั่นก็หมายความว่าค่าผลลัพธ์ของ sum คือ 1+2+3+4+5+6+7+8+9 ซึ่งผลลัพธ์สุดท้ายมีค่าเป็น 45

8.4 คำสั่ง while

เมื่อต้องการให้โปรแกรมมีการทำงานวนรอบโดยการตัดสินใจจากเงื่อนไขที่กำหนด โดยเปรียบกับเงื่อนไข ผู้พัฒนาอาจเลือกใช้รูปแบบคำสั่ง while

รูปแบบ

```
while (expression)
    statement;
```

โดยที่ expression คือ เงื่อนไขที่ใช้ในการตรวจสอบว่า จะให้โปรแกรมทำคำสั่งภายใน statement หรือไม่ ถ้าเป็นจริง จะมีการทำคำสั่งใน statement ถ้าเป็นเท็จ จะไม่มีการทำคำสั่งใน statement

ตัวอย่างที่ 15

```
while (1)
{
value = value + 1;
}
```

ในตัวอย่างนี้เงื่อนไขของคำสั่ง while จะเป็นจริงตลอดเวลา และเพิ่มค่าตัวแปร value ขึ้นหนึ่งค่าไปตลอด

ตัวอย่างที่ 16

```
value = 0;
while (value<5)
{
value++;
}
```

ตัวอย่างนี้จะมีการเพิ่มค่าตัวแปร value ขึ้นหนึ่งค่า จนกระทั่งเท่ากับ 5 จึงออกจากการวนรอบคำสั่งเพิ่มค่าตัวแปร เนื่องจากเงื่อนไขในการตรวจสอบเป็นเท็จ

8.5 คำสั่ง do-while

มีลักษณะการทำงานที่คล้ายกับคำสั่ง while แตกต่างเพียงในคำสั่งนี้จะทำงานใน statement ก่อนแล้วจึงเปรียบเทียบเงื่อนไข

รูปแบบ

```
do
        statement;
while (expression);
```

9. ฟังก์ชั่น

ในภาษา C จะมีฟังก์ชั่นหลักคือ main () เรียกว่า ส่วนโปรแกรมหลัก จะเป็นส่วนที่โรแกรมกระทำคำสั่งภายในไล่เรียงลำดับกันไป และจะกระโดดไปกระทำคำสั่งภายในฟังก์ชั่นอื่นๆที่มีการประกาศไว้ก็ต่อเมื่อมีการเรียกใช้งานฟังก์ชั่นนั้นๆภายในส่วนของโปรแกรมหลักนี้เอง หรืออาจกระโดดไปกระทำคำสั่งเนื่องจากการอินเตอร์รัปต์จากแหล่งกำเนิดใดๆ สุดท้ายก็จะกลับเข้ามาในโปรแกรมหลักอยู่ดี นอกจากนี้ภายในฟังก์ชั่นเองก็ยังสามารถเรียกใช้งานฟังก์ชั่นได้อีกด้วย ดังนั้นการทำงานกับฟังก์ชั่นจึงถือว่าเป็นส่วนที่มีบทบาทสำคัญในการเขียนโปรแกรมเลยทีเดียว

9.1 การประกาศฟังก์ชั่นและการเรียกใช้งานฟังก์ชั่น

รูปแบบ

```
return_type     fundtion_name (parameter1, parameter2, ...)
{
        command_list;
}
```

โดยที่ return_type คือ ชนิดข้อมูลที่มีการคืนค่าผลลัพธ์ออกมาและภายในฟังก์ชั่นชนิดนี้จะใช้คำสั่ง return(value) เพื่อส่งค่าผลลัพธ์ออกมา ซึ่งชนิดของข้อมูล value ที่ส่งออกมาควรจะเป็นชนิดเดียวกับ return_type เพื่อป้องกันการผิดพลาดที่อาจเกิดจากการรับข้อมูล สำหรับฟังก์ชั่นที่ไม่มีการคืนค่า (return value) ที่ตำแหน่ง return_type จะต้องประกาศเป็น void

parameter คือ ส่วนที่มีการประกาศชนิดข้อมูลและชื่อของตัวแปรที่นำมารับค่าที่ส่งผ่านให้กับฟังก์ชั่น อาจมี 1 ตัว หรือหลายๆตัว หรืออาจไม่มีเลยก็ได้ ขึ้นอยู่กับการใช้งานของฟังก์ชั่นนั้นๆ ถ้าฟังก์ชั่นไม่มี parameter รับค่า อาจประกาศเป็น void หรือเว้นว่างไม่ประกาศก็ได้

command_list คือ คำสั่งต่างๆที่อยู่ในตัวฟังก์ชั่น ซึ่งเมื่อจบ 1 คำสั่งจะต้องปิดท้ายด้วยเครื่องหมาย ; เสมอ

หมายเหตุ ที่ตำแหน่งของ value นั้นอาจจะเป็นค่าคงที่,ตัวแปร หรือผลลัพธ์จากการทำคำสั่งใดๆก็ได้

9.2 ตัวอย่างประกอบการประกาศและเรียกใช้งานฟังก์ชั่น

ตัวอย่างที่ 17

```
unsigned char x;
```

```
...
void f1  (void)
{
x++;                    // เพิ่มค่า x ขึ้น 1 ค่า
PORTE = X;              // โหลดค่าของ x ให้กับรีจิสเตอร์ PORTE
}
```

f1 เป็นฟังก์ชั่นแบบไม่มีการคืนค่าและไม่มีการส่งผ่านพารามิเตอร์

ตัวอย่างการใช้งานฟังก์ชั่น f1

```
unsigned char x;
...
int main ()
{
  ...
  f1 () ;          // เรียกฟังก์ชั่น f1
  ...
  return 0;
}
```

ตัวอย่างที่ 18

```
void f2 (unsigned char x, unsigned char y)
{
PORTE = x+y;  //  นำค่าที่ส่งผ่านมาทางพารามิเตอร์  x  และ  y  บวกกันแล้วโหลดให้กับรีจิสเตอร์
PORTE
}
```

f2 เป็นฟังก์ชั่นแบบไม่มีการคืนค่า แต่มีการส่งผ่านพารามิเตอร์ 2 ตัว คือ x และ y ซึ่งมีชนิดข้อมูลเป็นแบบ char

ตัวอย่างเรียกฟังก์ชั่นใช้งาน f2

```
int mian ()
{
  ...
  f2 (0x20 , 0x14) ;          // ส่งผ่านค่า 0x20 ให้กับ x และ 0x20 ให้กับ y
  ...
  return 0 ;
```

```
}
```

ตัวอย่างที่ 19

```
...
int f3 (int a, int b)
{
  return (a*b) ;                    // คืนค่าผลลัพธ์จากการคูณระหว่าง a กับ b
}
```

f3 เป็นฟังก์ชั่นแบบคืนค่าผลลัพธ์ชนิด int และมีการส่งผ่านพารามิเตอร์ 2 ตัว คือ a และ b ซึ่งมีชนิดข้อมูลเป็นแบบ int

ตัวอย่างเรียกใช้งานฟังก์ชั่น f3

```
int main ()
{
  int result ;
  ...
  result = f3 (26,40) ;     // โหลดค่าผลลัพธ์การคูณระหว่าง 26 กับ 40 ให้กับตัวแปร result
  ...
  return 0 ;
}
```

10. ไลบรารีและการใช้งาน

ในภาษา C โดนทั่วๆไปจะมีกลุ่มฟังก์ชั่นตามมาตรฐานที่ใช้พัฒนาโปรแกรมมาให้อยู่พอสมควร โดยจะถูกจัดหรือจำแนกเป็นหมวดหมู่ตามประเภทการใช้งานของฟังก์ชั่นออกได้เป็นหลายๆกลุ่ม โดยแต่ละกลุ่มฟังก์ชั่นที่ถูกบรรจุรวมกันเรียกว่า "ไลบรารี" ไฟล์ของไลบรารีจะมีนามสกุล .h

สำหรับไลบรารีในภาษา C ที่เป็นมาตรฐานการใช้งานอันได้แก่ stdio.h, string.h, math.h, stdlib.h และอื่นๆอีกหลายตัว นั้นมาพร้อมกับ C คอมไพเลอร์ทั่วๆไป ซึ่ง MPLAB C30 เองก็จะทำงานมีไลบรารีหลักๆเหล่านี้ติดตั้งมาเรียบร้อยแล้ว นอกจากนี้ยังมีไลบรารีของกลุ่มฟังก์ชั่นที่สนับสนุนการทำงานของโมดูลต่างๆภายในไมโครคอนโทรลเลอร์ dsPIC อีกหลายตัว เช่น adc12.h, uart.h, dsp.h, pwm.h, can.h, i2c.h, spi.h, timer.h เป็นต้น

10.1 การใช้งานไลบรารี

เมื่อผู้พัฒนาต้องการใช้งานฟังก์ชั่นซึ่งบรรจุอยู่ภายในไลบรารีใดๆ ผู้พัฒนาจำเป็นที่จะต้องประกาศรายชื่อไลบรารีเพื่อแสดงให้คอมไพเลอร์มองเห็นฟังก์ชั่นทุกตัวที่ถูกบรรจุภายในไลบรารีนั้นด้วยคีย์เวิร์ด #incudc<xxx.h>

โดยที่ xxx.h คือชื่อไลบรารีที่บรรจุฟังก์ชั่นที่ต้องการเรียกใช้งาน

ตัวอย่างที่ 20

```
#incude<math.h>                    // ผนวกไลบรารี math.h เพื่อเรียกใช้งานฟังก์ชั่น sqrt
                                   // สำหรับหาค่ารากที่สองของตัวเลขใดๆ
...
int main ()
{
  float m ;
  ...
  m = sqrt (5) ;                   // โหลดค่าผลลัพธ์จากการหาค่ารากที่สองของ 5 ให้กับ m
  ...
  return 0 ;
}
```

สำหรับตัวอย่างนี้เป็นการแสดงตัวอย่างเรียกการใช้งานฟังก์ชั่น sqrt ซึ่งถูกบรรจุอยู่ภายในไลบรารี math.h

ตัวอย่างที่ 21

```
#incude <p30f2010.h>               // ประกาศไลบรารีสำหรับรีจิสเตอร์หลักของ dsPIC30F2010
#incude<timer.h>                   // ประกาศไลบรารีของฟังก์ชั่นและมาโครต่างๆที่จัดการเกี่ยวกับ
                                   // การทำงารของโมดูลไทเมอร์ของ dsPIC
...
int main (void)
{
TRISDbits.TRISD1 = 0 ;// การเข้าถึงบิตภายในรีจิสเตอร์ของ dsPIC30F2010 จาก
                                   // p30f2010.h
...
PORTDbits.RD1 = 1 ;                // ฟังก์ชั่นภายในไลบรารี timer.h
                                   // p30f2010.h
...
ConfigIntTimer1 (...) ;            // ฟังก์ชั่นภายในไลบรารี timer.h
...
WriteTimer1 (0) ;                  // ฟังก์ชั่นภายในไลบรารี timer.h
```

```
...
OpenTimer1 (...) ;                    // ฟังก์ชั่นภายในไลบรารี timer.h
...
}
```

10.2 การสร้างไลบรารีเพื่อใช้งาน

นอกจากไลบรารีที่มาพร้อมกับ MPLAB C30 คอมไพเลอร์ ผู้พัฒนาสามารถสร้างไลบรารีสำหรับพัฒนาโปรแกรมเพิ่มเติมได้ ด้วยการสร้างไฟล์นามสกุล .h โดยบันทึกชื่อและพาธผู้พัฒนาต้องการ แล้วบรรจุฟังก์ชั่นต่างๆที่ต้องการลงภายในไฟล์ไลบรารี เพียงเท่านี้ก็จะได้ไฟล์ไลบรารีที่ต้องการ

ในกรณีที่จัดเก็บไลบรารีไว้ในพาธที่อยู่นอกไลบรารีหลักของคอมไพเลอร์ อันจะส่งผลให้คอมไพเลอร์มองไม่เห็นไลบรารีดังกล่าว ผู้พัฒนาจำเป็นจะต้องเพิ่มรายชื่อพาธให้แก่คอมไพเลอร์ด้วย โดยผู้พัฒนาสามารถตรวจสอบได้ที่โปรแกรม MPLAB โดยเข้าไปที่ Project → Set Language Tool Location ที่หัวข้อ Include Search Path,$ (INCDIR) ปรากฏหน้าต่างแจ้งพาธดังรูปที่ 1.2 และผู้พัฒนาต้องการให้คอมไพเลอร์เพิ่มการมองที่พาธ E:\dsPIC_project\include_2010 ซึ่งเป็นพาธที่จัดเก็บไลบรารีที่สร้างขึ้น ผู้พัฒนาเพียงพิมพ์ต่อท้ายชื่อพาธที่มีอยู่เดิม คั่วด้วยเครื่องหมาย ; ซึ่งจะได้ผลลัพธ์ตามรูปที่ 1.3

สำหรับการเรียกใช้งานฟังก์ชั่นภายในไลบรารีดังกล่าวนั้น จะทำในลักษณะเดียวกับตัวอย่างที่ 1.20 และ 1.21 คือประกาศ include ชื่อไฟล์ไลบรารีเพื่อแจ้งกับคอมไพเลอร์ก่อน หลังจากนั้นภายในโปรแกรมก็สามารถเรียกใช้งานได้ตามปกติ

ตัวอย่างที่ 22

ไฟล์ไลบรารี calculate.h มีรายละเอียดดังนี้

```
int sum (int x, int.y)
{
  return (x+y) ;
{
  int mul (int x, int y)
}
  return (x*y) ;
}
```

โปรแกรมหลักที่เรียกใช้งานไฟล์ไลบรารี calculate.h มีรายละเอียดดังนี้

```
#include<calaulate.h>          // ประกาศไลบรารีเพื่อให้คอมไพเลอร์เห็นฟังก์ชั่น sum และ mul
...
int main ()
```

```
{
  int dat ;
  …
  dat = mul (5,12) ;              // เรียกใช้งานฟังก์ชั่น mul จากไลบรารี calculate.h
  ...
  return 0 ;
}
```

11. รูปแบบการแทรกคำสั่งแอสเซมบลีร่วมกับ MPLAB C30 หรือ In line assembly

เมื่อผู้พัฒนาต้องการแทรกคำสั่งแอสเซมบลีลงภายในโปรแกรมภาษา C ของ MPLAB C30 รูปแบบที่ใช้มีลักษณะดังนี้

```
__asm__("insstruction") ;
```

หรือ

```
asm("insstruction") ;
```

โดยที่ instruction คือ คำสั่งภาษแอสเซมบลีที่ต้องการแทรก

เช่น

```
__asm__("MOV #0x55,WO") ;  // โหลดค่า 0x55 ไปยัง WO
```

12. มาโครที่มาพร้อมกับ MPLAB C30

นอกเหนือจากฟังก์ชั่นที่หลากหลายซึ่งถูกจำแนกบรรจุภายในไลบรารีต่างๆที่มาพร้อมกับ MPLAB C30 แล้วนั้น ยังมีกลุ่มมาโครอีกส่วนหนึ่งที่สนับสนุนการทำงานเทียบได้กับฟังก์ชั่นใน MPLAB C30 โดยถูกบรรจุไว้ภายในไลบรารีแบบเฮดเดอร์ (*.h) ประจำรหัสเบอร์ของไมโครคอนโทรลเลอร์ที่ใช้พัฒนา เช่น ในการพัฒนาโปรแกรมสำหรับควบคุมไมโครคอนโทรลเลอร์ dsPIC30F2010 ที่ส่วนหัวของโปรแกรม ผู้พัฒนาต้องผนวกไฟล์ p30f2010.h ด้วยไดเร็กตีฟ #include เพื่อให้คอมไพเลอร์รู้จักทรัพยากรต่างๆภายในไมโครคอนโทรลเลอร์ dsPIC30F2010 ซึ่งได้แก่ รีจิสเตอร์และบิตควบคุมทั้งหมดและรวมถึงมาโครกำหนดคุณสมบัติทางฮาร์ดแวร์ด้วย

ผู้พัฒนาสามารถดูรายละเอียดเกี่ยวกับกลุ่มมาโครเหล่านี้ได้จากพาธ C: \ Program Files \ Microchip \ MPLAB C30 \ support \ h

12.1 มาโครกำหนดคุณสมบัติของ Configuration

_FOSC (CSW_FSCM_ON & XT_PLL4);

ใช้เปิดการทำงานของแหล่งกำเนิดสัญญาณนาฬิกาภายนอก โดยกำหนดให้ทำงานในโหมด PLL4X

_FBORPOR (PBOR_ON & BORV_20 & PWRT_64 & MCLR_DIS) ;

ใช้เปิดการทำงานของพาวเวอร์-ออนรีเซตที่ระดับแรงดัน 2V, กำหนดค่าเริ่มต้นของพาวเวอร์-อัปไทเมอร์ที่ 64 มิลลิวินาที และกำหนดให้ขา MCLR ใช้งานเป็นพอร์ตอินพุตเอาต์พุตปกติ

นอกจากนี้ในกลุ่มนี้ยังมีมาโครอื่นๆที่ใช้ในการกำหนดคุณสมบัติในการทำงานของ dsPIC อีก ซึ่งสามารถดูรายละเอียดเพิ่มเติมได้จาก C: \ Program Files \ Microchip \ MPLAB C30 \ support \ h

12.2 มาโครแบบแทรกคำสั่งด้วยแอสเซมบลี

#define Nop () {__asm__ volatile ("Nop") ; }

กำหนดให้ Nop () แทนด้วยคำสั่งหน่วงเวลา 1 แมชีนไซเกิล

#define ClrWdt () {__asm__ volatile ("ClrWdt") ; }

กำหนดให้ ClrWdt () แทนด้วยคำสั่งเคลียร์วอตช์ด็อกไทเมอร์

#define Sleep () {__asm__ volatile ("Sleep") ; }

กำหนดให้ Sleep () แทนด้วนคำสั่งเข้าสู่โหมดสลีป

#define Idle () {__asm__ volatile ("Idle") ; }

กำหนดให้ Idle () แทนด้วยคำสั่งเข้าสู้โหมดไอเดิล

นอกจากนี้ยังมีมาโครอีกหลายตัวซึ่งมิได้กล่าวถึงในที่นี้ สามารถดูรายละเอียดเพิ่มเติมจากไฟล์เฮดเดอร์ที่กล่าวถึงในข้างต้น

ประวัติผู้เขียนโครงงาน

ชื่อ นายวราชัย จาดแดง

ภูมิลำเนา 523/2 ม.7 ต.ท่าตะโก อ.ท่าตะโก จ.นครสวรรค์

ประวัติการศึกษา

- จบมัธยมศึกษาจากโรงเรียนท่าตะโกพิทยาคม จ.นครสวรรค์
- ปัจจุบันกำลังศึกษาในระดับปริญญาตรีชั้นปีที่ 4

สาขาวิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์

มหาวิทยาลัยนเรศวร

E-mail: moouanjompuan@hotmail.com

ชื่อ นายอุรุพงศ์ สมนึก

ภูมิลำเนา 36/3 ม.4 ต.ป่าแดง อ.ชาติตระการ จ.พิษณุโลก

ประวัติการศึกษา

- จบมัธยมศึกษาจากโรงเรียนชาติตระการวิทยา จ.พิษณุโลก
- ปัจจุบันกำลังศึกษาในระดับปริญญาตรีชั้นปีที่ 4

สาขาวิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์

มหาวิทยาลัยนเรศวร

E-mail: sillyfools_iq180@hotmail.com