สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การออกแบบระบบควบคุมมอเตอร์ไฟฟ้ากระแสตรงของเครื่องเฉือนเนื้อทุเรียน

DC MOTOR CONTROL SYSTEM DESIGN FOR DURIAN SLICING MACHINE

T117583

พร้อมพงษ์ ภาชนนท์
ภักดิคุณ อยู่ดี
สุรสิทธิ์ วงศ์เฉลิมรัตน์

เลขหมู่.....
เลขทะเบียน 117583
วัน,เดือน,ปี - 9 ส.ค. 2554

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมแมคคาทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2553

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DC MOTOR CONTROL SYSTEM DESIGN
FOR DURIAN SLICING MACHINE

Prompong Pachanon

Pakdikoon U-dee

Surasit Wongchalermrut

THIS THESIS IS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN MECHATRONICS ENGINEERING

FAULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2010

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ปีการศึกษา 2553

สาขาวิชาวิศวกรรมการวัดและควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบระบบควบคุมมอเตอร์ไฟฟ้ากระแสตรงของเครื่องเฉือนเนื้อทุเรียน

DC MOTOR CONTROL SYSTEM DESIGN FOR DURIAN SLICING MACHINE

ผู้จัดทำ นายพร้อมพงษ์ ภาชนนท์ 50011059

นายภักดิคุณ อยู่ดี 50011152

นายสุรสิทธิ์ วงศ์เฉลิมรัตน์ 50011765

.....อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ดร.นพดล มณีรัตน์)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การออกแบบระบบควบคุมมอเตอร์ไฟฟ้ากระแสตรงของเครื่องเฉือนเนื้อทุเรียน

โดย

นายพร้อมพงษ์ ภาชนนท์ 50011059

นายภักดิคุณ อยู่ดี 50011152

นายสุรสิทธิ์ วงศ์เฉลิมรัตน์ 50011765

อาจารย์ที่ปรึกษา

ผู้ช่วยศาสตราจารย์ ดร. นพดล มณีรัตน์

ปีการศึกษา 2553

บทคัดย่อ

งานวิจัยฉบับนี้นำเสนอ การออกแบบระบบควบคุมมอเตอร์กระแสตรง สำหรับเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี โดยระบบควบคุมจะทำหน้าที่เป็นเครื่องตรวจสอบความเร็วรอบของมอเตอร์และจ่ายค่าพลังงานที่เหมาะสมให้กับมอเตอร์ในขณะนั้นๆ เพื่อควบคุมใบมีดเฉือนเนื้อทุเรียนให้มีการทำงานอย่างสม่ำเสมอ เนื่องจากงานวิจัยเดิมได้ใช้ระบบแบบพีไอในการควบคุม จุดมุ่งหมายของปริญญานิพนธ์ฉบับนี้ เพื่อศึกษาระบบควบคุมแบบฟัซซี สำหรับทำการออกแบบและสร้างระบบควบคุมชดเชยความเร็วรอบของมอเตอร์ที่ลดลงไปขณะเฉือนเนื้อทุเรียนให้มีความเร็วรอบโดยเลือกใช้ฟัซซี ระบบอัลกอริธึมแบบฟัซซีเพื่อเปรียบเทียบกับระบบควบคุมแบบพีไอในการควบคุมมอเตอร์ให้เข้าสู่จุดอ้างอิง ทางคณะผู้จัดทำจึงอาศัยหลักการของฟัซซีเซตและฟัซซีลอจิกในการควบคุม เพื่อสามารถใช้งานเครื่องเฉือนเนื้อทุเรียนแบบหมุนตามแนวแกนรัศมีได้อย่างมีประสิทธิภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DC MOTOR CONTROL SYSTEM DESIGN
FOR DURIAN SLICING MACHINE

By

Mr. Prompong Pachanon 50011059

Mr. Pakdikoon U-dee 50011152

Mr. Surasit Wongchalermrut 50011765

Thesis Advisor

Asst.Prof.Dr. Noppadol Maneerat

Academic Year 2010

Abstract

This thesis presents the DC motor control system design for durian slicing machine. The designed control system will check the motor speed and input power to the motor properly at that time to control the blade continuously. The aim of thesis is the performance comparison between the Fuzzy control system and the PI control system. We apply the fuzzy sets and fuzzy logic for the control system of durian slicing machine.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้ สามารถสำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความช่วยเหลือจาก ผศ.ดร.นพดล มณีรัตน์ ที่ได้กรุณาให้คำปรึกษาแนะนำตลอดเวลาที่ผ่านมา คณะผู้จัดทำรู้สึกซาบซึ้งและขอกราบขอบพระคุณอย่างสูง

ขอบคุณพี่และเพื่อนๆ ทุกคนที่สันบสนุนอุปกรณ์ แนะนำการออกแบบวงจรและการเขียนโปรแกรม ช่วยทำแบบจำลองและให้คำปรึกษา รวมทั้งความช่วยเหลืออื่นๆ ที่เป็นประโยชน์อย่างมากต่อโครงงานนี้

สุดท้ายนี้คณะผู้จัดทำขอกราบขอบพระคุณครอบครัวที่คอยเป็นกำลังใจที่ดีตลอดมา รวมถึงการสนับสนุนงบประมาณที่ขาด ตลอดจนเป็นแรงบันดาลใจที่ดีท่สุดที่ทำให้โครงงานนี้สำเร็จเสร็จสมบูรณ์ หากมีข้อผิดพลาดประการใด คณะผู้จัดทำต้องขออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

นายพร้อมพงษ์ ภาชนนท์

นายภักดิคุณ อยู่ดี

นายสุรสิทธิ์ วงศ์เฉลิมรัตน์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

	หน้า
บทคัดย่อ	I
บทคัดย่อ(ภาษาอังกฤษ)	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญรูป	VI
สารบัญตาราง	IX
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญ	1
1.2 วัตถุประสงค์ของโครงงาน	1
1.3 ขอบเขตของโครงงาน	1
1.4 ขั้นตอนการศึกษาและจัดทำโครงงาน	2
บทที่ 2 ทฤษฎีและความรู้ที่เกี่ยวข้อง	3
2.1 แนวคิดพื้นฐานเกี่ยวกับฟัซซีลอจิก	3
2.2 เซตแบบฉบับ	5
2.3 ฟัซซีเซต	6
2.4 ระบบกฎแบบฟัซซี	8
2.4.1 รูปแบบกฎฟัซซี	8
2.4.2 โครงสร้างพื้นฐานของการประมวลผลแบบฟัซซี	9
2.4.3 ขั้นตอนการประมวลผลแบบฟัซซีลอจิก	10
2.5 ทฤษฎีการควบคุมแบบพีไอดี	12
2.5.1 ระบบควบคุมแบบพี (สัดส่วน)	13
2.5.2 ระบบควบคุมแบบไอ (ปริพันธ์)	14
2.5.3 ระบบควบคุมแบบดี (อนุพันธ์)	15
2.5.4 สมการผลรวมของพีไอดี	16
2.6 ระบบควบคุมแบบป้อนกลับ	16
2.7 มอเตอร์ไฟฟ้ากระแสตรง	17

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

หน้า

2.8 โครงสร้างและหลักการทำงานของมอเตอร์ไฟฟ้ากระแสตรง 18

2.9 คุณสมบัติของมอเตอร์ไฟฟ้ากระแสตรง 19

2.10 การควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง 21

2.11 เอ็นโค้ดเดอร์ 24

บทที่ 3 หลักการและการออกแบบ 26

3.1 หลักการในการออกแบบ 26

3.2 ระบบจำลองเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี 27

3.2.1 มอเตอร์ 28

3.2.2 ตัวอ่านค่าความเร็ว 29

3.2.3 วงจรไฟฟ้าในระบบจำลองของเครื่องเฉือนเนื้อทุเรียน 29

3.3 เครื่องเฉือนเนื้อทุเรียน 30

บทที่ 4 การทดลองและผลการทดลอง 31

4.1 อุปกรณ์การทดลอง 31

4.2 วิธีทำการทดลอง 31

4.3 ผลการทดลอง 32

4.4 ตารางเปรียบเทียบความเสียหายของเนื้อทุเรียนโดยใช้ระบบควบคุมแบบพีไอและแบบฟัซซี 47

4.5 วิเคราะห์ผลการทดลอง 50

บทที่ 5 สรุปผลการทดลองและข้อเสนอแนะ 51

5.1 สรุปผลการทดลอง 51

5.2 ปัญหาและข้อเสนอแนะ 51

เอกสารอ้างอิง 52

ภาคผนวก 53

ภาคผนวก ก 54

ภาคผนวก ข 66

ภาคผนวก ค 80

ประวัติผู้เขียน 144

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป

หน้า

รูปที่ 2.1 ตรรกะแบบจริงเท็จ (บูลีนลอจิก) กับตรรกะแบบฟัซซีลอจิก 3

รูปที่ 2.2 ความไม่แน่นอน (uncertainly) 4

รูปที่ 2.3 ตัวอย่างเซตแบบฉบับ 5

รูปที่ 2.4 ความเป็นสมาชิกในเซตผู้ที่ไม่แต่งงาน 5

รูปที่ 2.5 การกำหนดค่าความเป็นสมาชิกของเซตวินัยและเซตแบบฟัซซี 6

รูปที่ 2.6 โครงสร้างพื้นฐานของการประมวลผลแบบฟัซซี 9

รูปที่ 2.7 ขั้นตอนการประมวลผลแบบฟัซซีลอจิก 10

รูปที่ 2.8 ขั้นตอนที่ 1 ของการประมวลผลแบบฟัซซีลอจิก 10

รูปที่ 2.9 ขั้นตอนที่ 2 ของการประมวลผลแบบฟัซซีลอจิก 11

รูปที่ 2.10 ขั้นตอนที่ 3 เป็นการหาฟัซซีเอาต์พุต 11

รูปที่ 2.11 ขั้นตอนที่ 4 ของการประมวลผลแบบฟัซซีลอกจิก 12

รูปที่ 2.12 บล็อกไดอะแกรมของระบบควบคุมแบบป้อนกลับ 17

รูปที่ 2.13 โครงสร้างของมอเตอร์ไฟฟ้ากระแสตรง 18

รูปที่ 2.14 การทำงานของมอเตอร์ไฟฟ้ากระแสตรง 18

รูปที่ 2.15 วงจรภายในของมอเตอร์ไฟฟ้ากระแสตรง 19

รูปที่ 2.16 แรงที่กระทำกับขดลวด 20

รูปที่ 2.17 ระบบควบคุมความเร็ว ที่ประกอบด้วยลูปการควบคุมป้อนกลับ
เพียงลูปเดียว 22

รูปที่ 2.18 สัญญาณ PWM ซึ่งแสดงค่าดิวตี้ไซเคิลที่ต่างๆ กัน 23

รูปที่ 2.19 โครงสร้างเอ็นโค้ดเดอร์ 24

รูปที่ 2.20 การสร้างพัลส์ของเอ็นโค้ดเดอร์ 24

รูปที่ 3.1 ระบบจำลองของเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี 28

รูปที่ 3.2 มอเตอร์ไฟฟ้ากระแสตรง 28

รูปที่ 3.3 เอ็นโค้ดเดอร์ 29

รูปที่ 3.4 บอร์ดขับมอเตอร์ 29

รูปที่ 3.5 บอร์ดควบคุม 30

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

หน้า

รูปที่ 3.6 เครื่องเฉือนเนื้อทุเรียน 30
รูปที่ 4.1 ผลการทดลองที่ความเร็ว 120 rpm ไม่มีตัวต้านทาน 32
รูปที่ 4.2 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 1 Ω 33
รูปที่ 4.3 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 2 Ω 33
รูปที่ 4.4 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 3 Ω 34
รูปที่ 4.5 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 5 Ω 34
รูปที่ 4.6 ผลการทดลองที่ความเร็ว 180 rpm ไม่มีตัวต้านทาน 35
รูปที่ 4.7 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 1 Ω 35
รูปที่ 4.8 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 2 Ω 36
รูปที่ 4.9 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 3 Ω 36
รูปที่ 4.10 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 5 Ω 37
รูปที่ 4.11 ผลการทดลองที่ความเร็ว 250 rpm ไม่มีตัวต้านทาน 37
รูปที่ 4.12 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 1 Ω 38
รูปที่ 4.13 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 2 Ω 38
รูปที่ 4.14 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 3 Ω 39
รูปที่ 4.15 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 5 Ω 39
รูปที่ 4.16 ผลการทดลองที่ความเร็ว 320 rpm ไม่มีตัวต้านทาน 40
รูปที่ 4.17 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 1 Ω 40
รูปที่ 4.18 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 2 Ω 41
รูปที่ 4.19 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 3 Ω 41
รูปที่ 4.20 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 5 Ω 42
รูปที่ 4.21 ผลการทดลองที่ความเร็ว 120 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน 42
รูปที่ 4.22 ผลการทดลองที่ความเร็ว 120 rpm ขณะเฉือนเนื้อทุเรียน 43
รูปที่ 4.23 ผลการทดลองที่ความเร็ว 180 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน 43
รูปที่ 4.24 ผลการทดลองที่ความเร็ว 180 rpm ขณะเฉือนเนื้อทุเรียน 44
รูปที่ 4.25 ผลการทดลองที่ความเร็ว 250 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน 44
รูปที่ 4.26 ผลการทดลองที่ความเร็ว 250 rpm ขณะเฉือนเนื้อทุเรียน 45

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

หน้า

รูปที่ 4.27 ผลการทดลองที่ความเร็ว 320 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน 45

รูปที่ 4.28 ผลการทดลองที่ความเร็ว 320 rpm ขณะเฉือนเนื้อทุเรียน 46

รูปที่ 4.29 ผลการทดลองที่ความเร็ว 400 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน 46

รูปที่ 4.30 ผลการทดลองที่ความเร็ว 400 rpm ขณะเฉือนเนื้อทุเรียน 47

รูปที่ 4.31 เนื้อทุเรียนของระบบควบคุมแบบพีไอ 48

รูปที่ 4.32 เนื้อทุเรียนของระบบควบคุมแบบฟัซซี 50

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

หน้า

ตารางที่ 4.1 ความเสียหายที่เกิดจากการเฉือนเนื้อทุเรียนโดยใช้
ระบบควบคุมแบบพีไอที่ความเร็ว 250 rpm 47
ตารางที่ 4.2 ความเสียหายที่เกิดจากการเฉือนเนื้อทุเรียนโดยใช้
ระบบควบคุมแบบฟัซซีที่ความเร็ว 250 rpm 49

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

จากงานวิจัยเรื่องการออกแบบระบบควบคุมเครื่องเฉือนเนื้อทุเรียนแบบหมุนตามแนวแกนรัศมีเดิม ได้มีการแก้ไขผลผลิตที่มีมากเกินความต้องการของตลาด เพื่อเร่งหาตลาดทำการจำหน่ายให้มากขึ้นทั้งตลาดภายในและต่างประเทศ โดยนำทุเรียนมาแปรรูปเป็นทุเรียนทอดโดยทางผู้จัดทำได้ทำการพัฒนาในด้านการออกแบบระบบควบคุมใบมีด จากเดิมที่ได้มีการใช้ระบบควบคุมเป็นแบบพีไอโดยใช้เอนโคดเดอร์ตรวจจับความเร็วการหมุนของมอเตอร์ แต่ในงานวิจัยเรื่องนี้ได้มีการพัฒนาโดยใช้พีไอและฟัซซีคอนโทรลเลอร์ เพื่อควบคุมการหมุนของใบมีดส่วนในด้านของตัวเครื่องเฉือนเนื้อทุเรียนได้ทำการออกแบบใหม่ให้ใบมีดอยู่ในแนวดิ่ง ดังนั้นจึงเริ่มต้นจากการศึกษาการใช้คอนโทรลเลอร์ในการควบคุมมอเตอร์เพื่อให้ได้ผลตามที่ต้องการ

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อพัฒนาระบบควบคุมเครื่องเฉือนเนื้อทุเรียนตามแนวรัศมี
2. เพื่อทำให้เครื่องเฉือนเนื้อทุเรียนทำงานได้มีประสิทธิภาพ
3. เพื่อศึกษาการควบคุมมอเตอร์โดยใช้ไมโครคอนโทรลเลอร์
4. เพื่อศึกษาโปรแกรมเพื่อใช้ควบคุมคอนโทรลเลอร์

1.3 ขอบเขตของโครงงาน

1. ศึกษาระบบควบคุมแบบพีไอดี และแบบฟัซซี
2. ศึกษาการใช้งานของมอเตอร์กระแสตรงและเซนเซอร์วัดความเร็วร่วมกับไมโครคอนโทรเลอร์ ทำให้สามารถรับค่าและคำนวณผลได้
3. ศึกษาโครงสร้างและหลักการทำงานของไมโครคอนโทรลเลอร์
4. ศึกษาการใช้งานร่วมกันระหว่างไมโครคอนโทรลเลอร์และวงจรขับมอเตอร์มอเตอร์กระแสตรงโดยใช้ไมโครคอนโทรลเลอร์ควบคุมความเร็วได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.4 ขั้นตอนการศึกษาและจัดทำโครงงาน

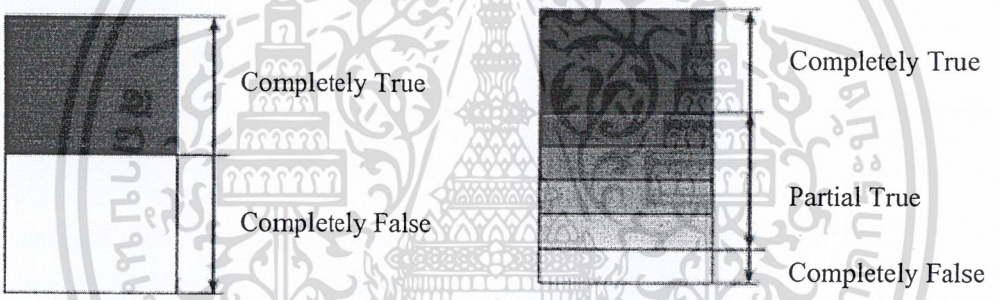
การศึกษามอเตอร์กระแสตรงนั้น จำเป็นต้องรู้และเข้าใจตัวระบบก่อนตั้งแต่เริ่มแรก คือ รวบรวมข้อมูลที่จำเป็นทั้งไมโครคอนโทรลเลอร์ มอเตอร์กระแสตรง วงจรขับมอเตอร์ และระบบควบคุมอื่นๆ ที่แตกต่างจากเดิม เพื่อนำมาเปรียบเทียบข้อดีข้อเสียระหว่างระบบควบคุมแบบพีไอ กับระบบควบคุมฟัซซี เพื่อที่จะนำไปเลือกใช้ให้เหมาะสมกับเครื่องเฉือนเนื้อทุเรียน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
ทฤษฎีและความรู้ที่เกี่ยวข้อง

2.1 แนวคิดพื้นฐานเกี่ยวกับฟัซซีลอจิก

ตรรกะแบบฟัซซี (Fuzzy logic) เป็นเครื่องมือที่ช่วยในการตัดสินใจภายใต้ความไม่แน่นอนของข้อมูลโดยยอมให้มีความยืดหยุ่นได้ใช้หลักเหตุผลที่คล้ายการเลียนแบบวิธีความคิดที่ซับซ้อนของมนุษย์ ฟัซซีลอจิกมีลักษณะที่พิเศษกว่าตรรกะแบบจริงเท็จ (Boolean logic) เป็นแนวคิดที่มีการต่อขยายในส่วนของความจริง (Partial true) โดยค่าความจริงจะอยู่ในช่วงระหว่างจริง (Completely true) กับเท็จ (Completely false) ส่วนตรรกศาสตร์เดิมจะมีค่าเป็นจริงกับเท็จเท่านั้น แสดงดังรูปที่ 2.1



(a) บูลีนลอจิก (b) ฟัซซีลอจิก

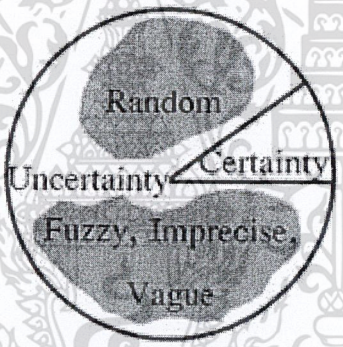
รูปที่ 2.1 ตรรกะแบบจริงเท็จ (บูลีนลอจิก) กับตรรกะแบบฟัซซี (ฟัซซีลอจิก)

ความเป็นฟัซซี (Fuzziness) มีชื่อเรียกว่า มัลติวาลานซ์ (Multivalance) ซึ่งมีค่าที่ความเป็นสมาชิกมากกว่า 2 ค่า และแตกต่างกับไบวาลานซ์ (Bivalance) ที่มีความเป็นสมาชิกเพียง 2 ค่า ฟัซซีเซต (Fuzzy set) เป็นเครื่องมือทางคณิตศาสตร์ที่สื่อถึง "ความไม่แน่นอน (Uncertainty)" สามารถตัดสินใจได้มากกว่า 2 กรณี ซึ่งหากกำหนดว่าคนที่อ้วนคือคนที่มีน้ำหนักมากกว่า 75 กิโลกรัม คอมพิวเตอร์จะให้ผลว่าคนที่มีน้ำหนัก 74.50 กิโลกรัมไม่จัดเป็นคนที่อ้วน จะสร้างและกำหนดรูปแบบ (Modeling) ของลักษณะความไม่แน่นอนที่เป็นความคลุมเครือ ความไม่ตายตัว รวมถึงความขาดข้อมูลบางส่วน โดยทฤษฎีของฟัซซีเซตจะใช้ลักษณะความหมายตัวแปร (Linguistic) มากกว่าปริมาณ (Quantitative) ของตัวแปร เช่น การหาความหมายของ "คนที่อ้วน" จะไม่สามารถนิยามค่าความอ้วนที่ตรงกันและระบุเป็นหนึ่งเดียว (Identical) สำหรับคนที่อ้วน นาย ก. จะให้ความหมายของ "คนอ้วน" หมายถึง คนที่มีน้ำหนักมากกว่า 70 กิโลกรัม นาย ข.ให้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้นำเอกสารนี้ไปเผยแพร่ต่อ ไม่ว่าในรูปแบบใดๆ ทั้งสิ้น

ความหมายว่าเป็นคนที่มีน้ำหนักมากกว่า 75 กิโลกรัม ซึ่งทั้งสองคนต่างแสดงความหมายของคำว่าคนที่อ้วน โดยเปรียบเทียบและในมุมมองของตัวเองตามน้ำหนักของตน ในการทำงานในมุมมองแบบฐานสอง (Binary sense) จะได้ผลเป็น ใช่ หรือแต่จะเห็นว่าบุคคลนี้เป็นคนอ้วนน้ำหนักเกือบจะ 75 กิโลกรัม และถึงแม้ว่าบุคคลนี้จะมีน้ำหนัก 75 กิโลกรัม แต่หากพิจารณาจากกลุ่มคนที่มีน้ำหนักเฉลี่ย 90 กิโลกรัม บุคคลนี้ก็จะไม่จัดอยู่ในกลุ่มคนที่อ้วนแสดงให้เห็นว่าความอ้วนไม่ได้มีลักษณะความไม่แน่นอนแบบสุ่ม จากการศึกษาปัญหาทั่วๆ ไปจะแสดงถึงรูปแบบลักษณะการกระจายของปัญหา

รูปที่ 2.2 เป็นการแสดงให้เห็นว่าแนวทางในการตัดสินใจของปัญหาทั้งหมดมีเพียงส่วนน้อยที่เป็นสิ่งที่แน่นอน (Certainty) ที่เหลือคือสิ่งที่ไม่แน่นอน ซึ่งประกอบด้วยความไม่แน่นอนที่มีลักษณะแบบสุ่มและความไม่แน่นอนที่มีลักษณะเป็นฟัซซีหรือคลุมเครือ ซึ่งมีมากกว่าร้อยละ 40 เพราะปัญหาส่วนมากเกี่ยวข้องกับการตัดสินใจของมนุษย์ ซึ่งจะตัดสินใจตามพื้นฐานความคิดของตนเป็นหลัก



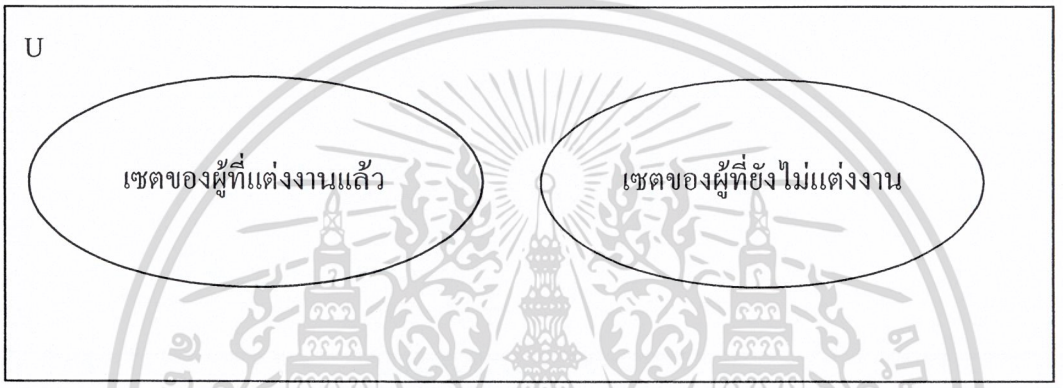
รูปที่ 2.2 ความไม่แน่นอน (Vncertainly)

ฟัซซีจะสร้างวิธีทางคณิตศาสตร์ที่แสดงถึงความคลุมเครือ ความไม่แน่นอนของระบบที่เกี่ยวข้องกับความคิดความรู้สึกของมนุษย์ เมื่อพิจารณาส่วนประกอบต่างๆ ในความไม่แน่นอนเพื่อกำหนดเงื่อนไขในการตัดสินใจ (Decision making) โดยอาศัยเซตของความไม่เป็นสมาชิก (Set membership)

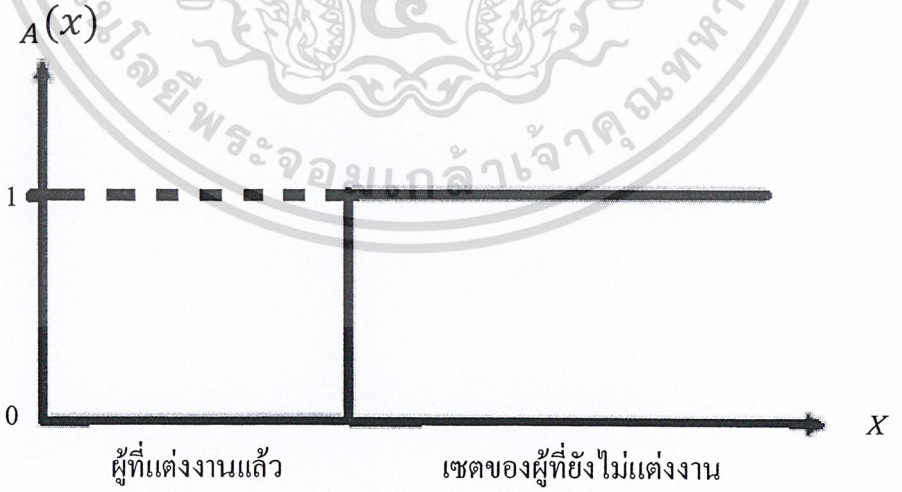
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2 เซตแบบฉบับ

ในเซตแบบฉบับ (Classical set) หรือเซตทวินัย (Crisp set) เป็นเซตที่มีค่าความเป็นสมาชิกเป็น 0 หรือ 1 {0, 1} เท่านั้น เซตในทฤษฎีเซตแบบฉบับจะมีขอบเขตแบบแข็ง (Sharp boundary) ซึ่งเป็นขอบเขตที่ตัดขาดจากกันแบบทันทีทันใด เซตแบบฉบับมีการกำหนดค่าความเป็นสมาชิกตามแนวคิดเลขฐานสอง โดยที่ตัวแปรหนึ่งๆจะมีค่าความเป็นสมาชิกเพียงสองค่า คือ 0 ไม่เป็นสมาชิก และ 1 เป็นสมาชิกตัวอย่าง เช่น เซตของคู่แต่งงานจะสามารถบอกได้ว่าอย่างแน่ชัดว่าเป็นกลุ่มผู้แต่งงานหรือไม่แต่งงาน



รูปที่ 2.3 ตัวอย่างเซตแบบฉบับ



รูปที่ 2.4 ความเป็นสมาชิกในเซตผู้ที่ไม่แต่งงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 2.3 แสดงตัวอย่างของเซตย่อยสองเซต คือ เซตของผู้ที่แต่งงานและเซตของผู้ที่ไม่แต่งงาน จะเห็นได้ว่าคนหนึ่งคนจะเป็นสมาชิกภาพได้เพียงเซตเดียวเท่านั้น แต่งงานหรือไม่แต่งงาน ในรูปที่ 2.4 แสดงฟังก์ชันความเป็นสมาชิกของเซตผู้ที่ไม่แต่งงาน จะเห็นได้ว่า ผู้ที่แต่งงานแล้วจะมีค่าความเป็นสมาชิกในเซตของผู้ไม่แต่งงานเป็น 0 ส่วนผู้ที่ไม่แต่งงานมีค่าความเป็นสมาชิกภาพของเซตผู้ที่ไม่แต่งงานเป็น 1 ค่าความเป็นสมาชิกของทั้งสองเซตจะตัดขาดจากกันอย่างทันทีทันใด รูปแบบคณิตศาสตร์ของเซตแบบฉบับมีรูปดังนี้

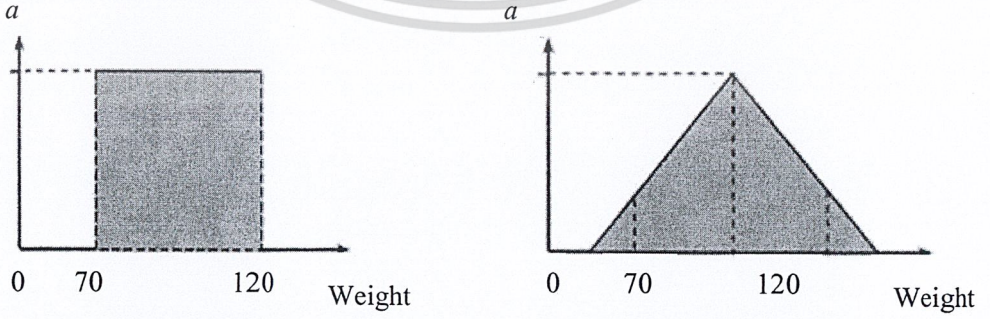
$$\mu_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \qquad (2.1)$$

เมื่อ A เป็นเซตแบบฉบับหรือเซตแบบทวินัย x เป็นสมาชิกในเซต μ_A เป็นค่าความเป็นสมาชิกในเซต และ $\mu_A(x)$ เป็นฟังก์ชันความเป็นสมาชิกในเซต A

2.3 ฟัซซีเซต

ฟัซซีเซต (Fuzzy Set) เป็นเซตที่มีขอบเขตที่ราบเรียบ ทฤษฎีฟัซซีเซตจะครอบคลุมทฤษฎีเซตแบบฉบับ โดยฟัซซีเซตยอมให้มีค่าความเป็นสมาชิกของเซตระหว่าง 0 และ 1 ในโลกแห่งความเป็นจริงเซตไม่ใช่มีเฉพาะเซตแบบฉบับเท่านั้น จะมีเซตแบบฟัซซีด้วยฟัซซีเซตจะมีขอบเขตแบบฟัซซีไม่ใช่เปลี่ยนแปลงทันทีทันใดจากขาวเป็นดำ ตัวอย่างเช่นเซตของคู่แต่งงานที่มีความสุข จะเห็นได้ว่าสมาชิกในเซตนี้จะไม่มีเฉพาะคู่แต่งงานที่มีความสุขระดับเดียวกันหมด บางคู่จะมีความสุขมาก บางคู่มีความสุขน้อยแตกต่างกันไป การใช้เซตแบบดั้งเดิมจึงไม่เหมาะสม

ยกตัวอย่างเกี่ยวกับความอ้วน นิยามคำว่าคนอ้วนในเซตทวินัยอาจกำหนดเป็นคนที่มีน้ำหนักตั้งแต่ 70 ถึง 120 กิโลกรัม โดยนิยามแบบฟัซซีเซตอาจกำหนดเป็นคนที่มีความอ้วนประมาณ 80 กิโลกรัม ซึ่งเป็นการให้นิยามที่ไม่แสดงถึงขอบเขตที่แน่นอน



(ก.) แบบบูลีนลอจิก (ข.) แบบฟัซซี

รูปที่ 2.5 การกำหนดค่าความเป็นสมาชิกของเซตวินัยและเซตแบบฟัซซี

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

นิยามของฟัซซีเซต กำหนดให้ X เป็นเซตที่ไม่ว่างฟัซซีเซต A สามารถแสดงลักษณะเฉพาะได้จากฟังก์ชันความเป็นสมาชิก

$$\mu_A(x): X \longrightarrow [0,1] \tag{2.2}$$

เมื่อ μ_A สามารถตีความเป็นค่าของความเป็นสมาชิกภาพของตัวประกอบ X ในฟัซซีเซต A สำหรับแต่ละ (อ่านว่า "x เป็นสมาชิกของ X") ฟัซซีเซต สามารถเขียนเป็นเซตของคู่ลำดับ (Tuples)

$$\underline{A} = \{(x\, \mu_A(x)) \mid x \in X\} \tag{2.3}$$

เมื่อ $\underline{A}$ หมายถึง ฟัซซีเซต $\underline{A}$ x หมายถึง สมาชิกของเซต (Set membership) μ_A หมายถึง ฟังก์ชันความเป็นสมาชิก (Membership function) μ_A (x) บางครั้งแทนด้วย $\underline{A}(x)$ X หมายถึง เอกภพสัมพัทธ์ (Universe) หรือประชากร ถ้า $X = \{x_1, x_2, x_3, \ldots, x_n\}$ เป็นเซตจำกัด และ $\underline{A}$ เป็นฟัซซีเซตใน X ซึ่งเป็นชนิดไม่ต่อเนื่อง (Discrete) และจำกัดสัญลักษณ์ (Notation) ของฟัซซีเซต เขียนได้เป็น

$$A = \left\{\frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \cdots + \frac{\mu_A(x_n)}{x_n}\right\} = \left\{\sum_{i=1}^{n} \frac{\mu_A(x_n)}{x_n}\right\} \tag{2.4}$$

เมื่อพจน์ $\mu_A(x_1)|x_1, i = 1.2 \ldots \ldots n$ หมายถึง ค่าความเป็นสมาชิก $\mu_A(x_1)$ ของ x_1 ในเซต A และ เครื่องหมายบวก "+" หมายถึง ยูเนี่ยน(Union)

ถ้าเอกภพสัมพัทธ์ X เป็นต่อเนื่อง (continuous) สัญกรณ์ (notation) ของฟัซซีเซต A เขียนได้เป็น

$$A = \int \frac{\mu_{A\sim}}{X} \tag{2.5}$$

ทฤษฎีฟัซซีเซตสามารถแก้ปัญญาข้อจำกัดของเซตแบบดั้งเดิมได้ โดยฟัซซีเซตยอมให้มีค่าหรือดีกรีของความเป็นสมาชิก (Degree of membership) ซึ่งแสดงด้วยค่าตัวเลขระหว่าง 0 และ 1 หรือเขียนเป็นสัญลักษณ์ [0, 1] โดย 0 หมายถึง ไม่เป็นสมาชิกในเซต 1 หมายถึง เป็นสมาชิกในเซต และค่าระหว่าง 0 กับ 1 เป็นสมาชิกบางส่วนในเซต การทำเช่นนี้ทำให้เกิดความราบเรียบในการเปลี่ยนจากพื้นที่นอกเซตไปอยู่ในเซตของสมาชิกต่างๆ โดยมีฟังก์ชันสมาชิก (Membership function) เป็นฟังก์ชันจัดเทียบ (Mapping function) วัตถุในโดเมนใดๆ ให้เป็นค่าความเป็นสมาชิกในฟัซซีเซต

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ความเป็นสมาชิกสำหรับฟัซซีเซต มีจำนวนระดับความเป็นสมาชิกเป็นอนันต์ คือ ค่าต่อเนื่องในช่วงตั้งแต่ 0 ถึง 1 ซึ่งครอบคลุมการกำหนดสมาชิกแบบฉบับ และเซตแบบฉบับหรือเซตทวินัย (Crisp set) จะกำหนดตามดังสมการที่ (2.6)

$$\mu_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \tag{2.6}$$

เมื่อ A เป็นเซตแบบฉบับหรือเซตแบบทวินัย x เป็นสมาชิกในเซต μ_A เป็นค่าความเป็นสมาชิกในเซต และ $\mu_A(x)$ เป็นฟังก์ชันความเป็นสมาชิกในเซต A

2.4 ระบบกฎแบบฟัซซี

ในระบบปัญญาประดิษฐ์(Artificial intelligence) หรือเครื่องจักรอัจริยะ (Machine intelligence) มีวิธีการหลายวิธีในการที่จะแสดงองค์ความรู้ของมนุษย์ในรูปแบบต่างๆ เช่น ตรรกะ (Logic) เฟรม (Frames) โครงข่ายความหมาย (Semantic nets) ภววิทยา (Ontology) และกฎ (Rules) ซึ่งแบบหลังสุดเป็นวิธีหนึ่งที่นิยมใช้ในระบบฟัซซี

2.4.1 รูปแบบกฎฟัซซี

ระบบฟัซซีองค์ความรู้สามารถแสดงในรูปประโยค

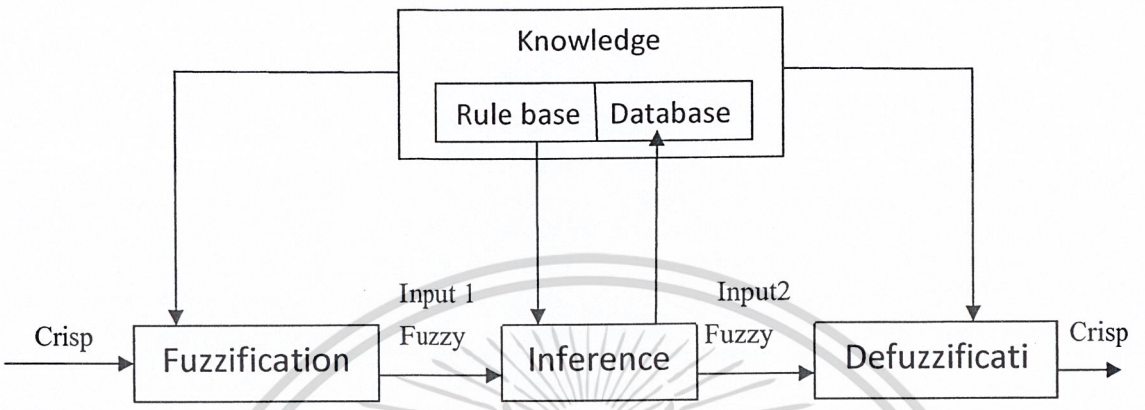
ถ้า ข้อตั้ง (ข้อนำ) ดังนั้น ข้อยุติ (ข้อตาม)
IF premise (antecedent), THEN conclusion (consequent)

ข้อความข้างต้นเป็นที่รู้จักกันดีในนาม “รูปแบบฐานกฎถ้า-ดังนั้น” หรือ รูปแบบนิรนัย ในรูปแบบการแสดงอนุมาน หากหาข้อยุติหรือข้อตามการแสดงรูปแบบองค์ความรู้นี้เรียกว่า องค์ความรู้ตื้น (Shallow knowledge) ซึ่งค่อนข้างมีความเหมาะสมในบริบทของภาษา เนื่องจากเป็นการแสดงประสบการณ์ของมนุษย์และองค์ความรู้เชิงศึกษาสำนัก (Heuristics) ในรูปแบบประโยคภาษามนุษย์ที่ใช้ในการสื่อสารทั่วไป แต่ไม่เป็นรูปแบบองค์ความรู้ที่ลึกล้ำแบบที่เป็นการรู้เองเป็นโครงสร้างเป็นฟังก์ชัน หรือเป็นพฤติกรรมของวัตถุรอบๆ ตัวเราอย่างที่เรียกว่า อุปนัย (Inductive) ระบบกฎฟัซซีเป็นสิ่งที่มีประโยชน์ในการจัดรูปแบบของระบบที่ซับซ้อนที่สามารถสังเกตได้โดยมนุษย์ เพราะระบบเหล่านี้สามารถแสดงด้วยตัวแปรภาษาในข้อนำและข้อตามของกฎได้ตัวแปรภาษาสามารถนำแสดงเชิงธรรมชาติด้วยฟัซซีเซตและตัวเชื่อมตรรกะของเซตเหล่านั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.2 โครงสร้างพื้นฐานของการประมวลผลแบบฟัซซีลอจิก

โครงสร้างพื้นฐานของการประมวลผลแบบฟัซซีซึ่งประกอบด้วยส่วนที่สำคัญ 4 ส่วนดังนี้ ดังรูปที่ 2.6



รูปที่ 2.6 โครงสร้างพื้นฐานของการประมวลผลแบบฟัซซี

ส่วนที่แปลงการอินพุตทั่วไปเปลี่ยนเป็นการอินพุตแบบตัวแปรฟัซซี (Fuzzification) หรือในรูปแบบเซตฟัซซีหรือเรียกว่าเป็นตัวแปรภาษา (Linguistic Variable)

ฐานความรู้ (Knowledge Base) เป็นส่วนที่จัดเก็บรวบรวมข้อมูลในการควบคุมประกอบ 2 ส่วนคือ ฐานกฎ (Rule Base) และฐานข้อมูล (Database)

ฐานกฎ (Rule Base) ส่วนของการกำหนดวิธีการควบคุมซึ่งได้จากผู้เชี่ยวชาญในรูปแบบของชุดข้อมูลแบบกฎของภาษา (Linguistic Rule)

ฐานข้อมูล (Database) เป็นการจัดเตรียมส่วนที่จำเป็นเพื่อที่จะใช้ในการกำหนดกฎการควบคุม และการจัดการข้อมูลของตรรกศาสตร์ฟัซซี

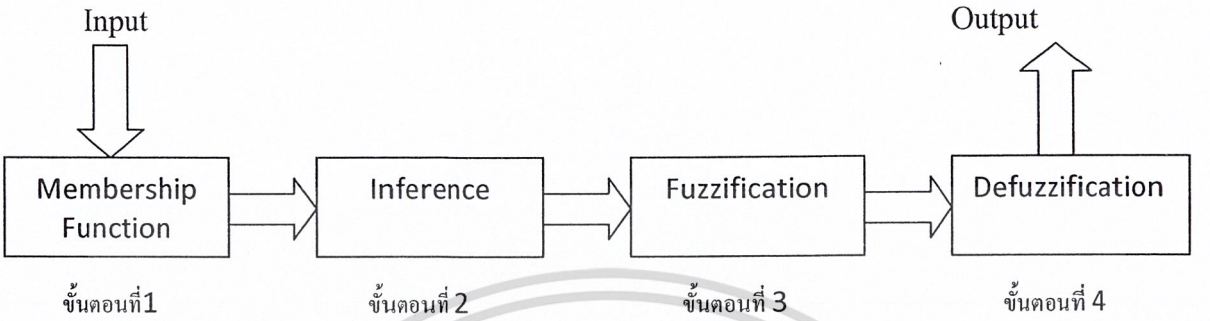
เครื่องอนุมานหรือการตีความ (Inference Engine) เป็นส่วนที่ทำหน้าที่ตรวจสอบข้อเท็จจริงและกฎ เพื่อใช้ในการตีความหาเหตุผลเหมือนกลไกสำหรับควบคุมการใช้ความรู้ในการแก้ไขปัญหา รวมทั้งการกำหนดวิธีการของการตีความเพื่อหาคำตอบ

ส่วนที่แปลงการเอาต์พุตให้อยู่ในช่วงที่เหมาะสม (Defuzzification) เป็นการทำการแปลงข้อมูลที่อยู่ในรูปแบบฟัซซีให้เป็นค่าที่สรุปผลหรือค่าการควบคุมระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

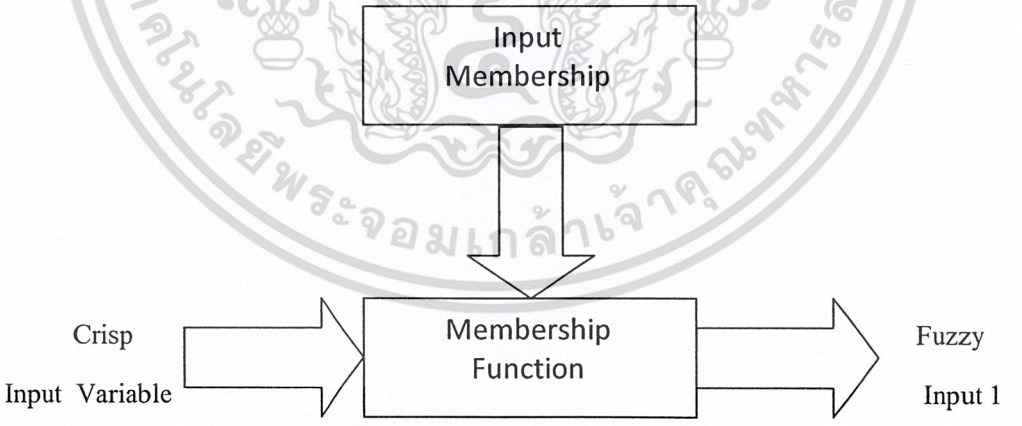
2.4.3 ขั้นตอนการประมวลผลแบบฟัซซีลอจิก

ขั้นตอนการประมวลผลแบบฟัซซีลอจิกมีรูปแบบการทำงานเป็น 4 ส่วน แสดงดังรูปที่ 2.7



รูปที่ 2.7 ขั้นตอนการประมวลผลแบบฟัซซีลอจิก

ขั้นตอนที่ 1 เป็นการแปลงการอินพุตแบบทวินัยเปลี่ยนเป็นการอินพุตแบบตัวแปรฟัซซี โดยจะสร้างฟังก์ชันความเป็นสมาชิก โดยไม่จำเป็นต้องมีลักษณะเดียวกันขึ้นกับคุณลักษณะของแต่ละการอินพุต (Input) และความสำคัญต่อการเอาต์พุต (Output) ที่น่าสนใจ โดยฟังก์ชันจะมีลักษณะเป็นการกำหนดภาษาสามัญเพื่อให้เป็นฟัซซีการอินพุต ดังรูปที่ 2.8

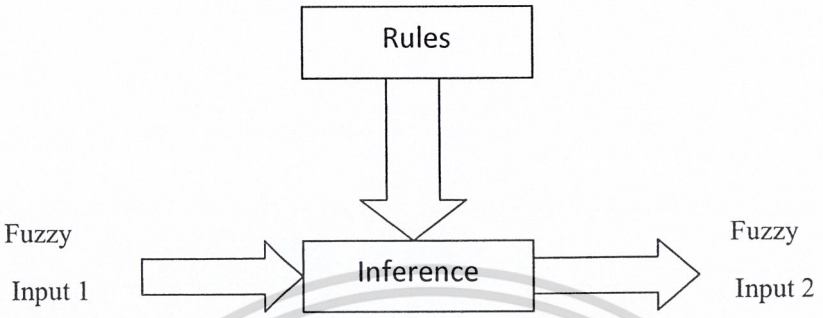


รูปที่ 2.8 ขั้นตอนที่ 1 ของการประมวลผลแบบฟัซซีลอจิก

ขั้นตอนที่ 2 เป็นการสร้างความสัมพันธ์ระหว่างการอินพุตทั้งหมดที่เกี่ยวข้องกับเอาต์พุตที่อาศัยหลักการของการหาเหตุและผลอาจจะสร้างการเก็บข้อมูลการคาดการณ์จากการตัดสินใจของมนุษย์หรือค่าจากการทดลอง โดยเขียนเป็นกฎการควบคุมระบบ ซึ่งจะมีลักษณะอยู่ในรูปแบบถ้า

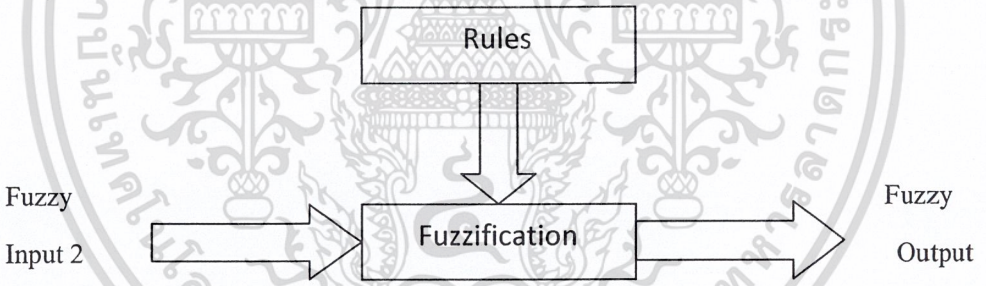
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(If) และ (And) หรือ (Or) ซึ่งเป็นภาษาสามัญนำกฎทั้งหมดมาประมวลผลรวมกันเพื่อการหาตัดสินใจที่เหมาะสม ดังรูปที่ 2.9



รูปที่ 2.9 ขั้นตอนที่ 2 ของการประมวลผลแบบฟัซซีลอจิก

ขั้นตอนที่ 3 เป็นการหาฟัซซีเอาต์พุตโดยการนำกฎการควบคุมที่สร้างขึ้นในขั้นตอนที่ 2 มาประมวลผลกับฟัซซีอินพุต โดยใช้วิธีการทางคณิตศาสตร์เพื่อนำค่าที่ได้ประมวลผล ดังรูปที่ 2.10

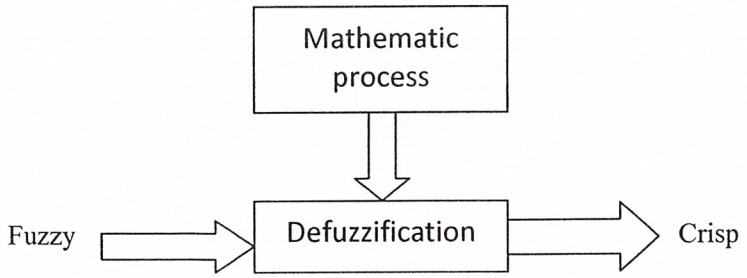


รูปที่ 2.10 ขั้นตอนที่ 3 เป็นการหาฟัซซีเอาต์พุต

วิธีการทำเป็นค่าคลุมเครือ (Fuzzification) วิธีการที่นิยมใช้ในการตีความหาเหตุผลเลือกใช้ Max-Min method และ Max-Dot method

ขั้นตอนที่ 4 เป็นขั้นตอนสุดท้ายหรือขั้นตอนการสรุปเหตุผลฟัซซีโดยจะเปลี่ยนฟัซซีเอาต์พุตให้เป็นทวินัยเอาต์พุตตามรูปที่ 2.11 และด้วยวิธีทางคณิตศาสตร์ เช่น วิธีการหาจุดศูนย์ถ่วง (Central of Gravity) เพื่อนำค่าที่ได้มาใช้ในการตัดสินใจเพื่อควบคุมระบบในสถานการณ์นั้นๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.11 ขั้นตอนที่ 4 ของการประมวลผลแบบฟัซซีลอกจิก

วิธีการทำค่าฟัซซีให้เป็นค่าปกติ (Defuzzification) วิธีการที่เป็นเทคนิคการเลือกค่าสูงสุดหรือสรุปหาเหตุผลจากหลายๆ เซตมาเพียงค่าเดียวซึ่งเป็นการใช้ค่าสูงสุดของค่าระดับการเป็นสมาชิกจากการกระทำหลายๆ แบบและเลือกกระทำเพียงรูปแบบเดียว

วิธีการหาจุดศูนย์ถ่วง (Central of Gravity: COG) เป็นวิธีการเฉลี่ยผลที่ได้จากการตีความหาเหตุที่นิยมใช้ในปัจจุบัน ค่าที่ได้จะคำนวณจุดศูนย์ถ่วงโดยรวมจะหาได้จากการประมาณค่าจากสมการ

$$\mathbf{COG} = \frac{\sum_{i=1}^{N} \propto_i w_i}{\sum_{i=1}^{N} \propto_i} \tag{2.7}$$

โดยสมการ ได้กำหนดค่าของสมการดังนี้

COG = ค่าของจุดศูนย์ถ่วง (Central of Gravity)

N = ค่าตั้งแต่ตำแหน่งที่ 1 ถึงตำแหน่งที่ i

$\propto_i$ = ค่าฟัซซีของเอาต์พุตในเซตฟัซซีตำแหน่งที่ i

w_i = พื้นที่ใต้โค้งของเซตฟัซซีตำแหน่งที่ i

2.5 ทฤษฎีการควบคุมแบบพีไอดี

การควบคุมแบบพีไอดี ได้ชื่อตามการรวมกันของเทอมของตัวแปรทั้งสามตามสมการ

$$MV(t) = P_{out} + I_{out} + D_{out} \tag{2.8}$$

เมื่อ

P_{out} = สัญญาณขาออกของเทอมสัดส่วน

I_{out} = สัญญาณขาออกของเทอมปริพันธ์

D_{out} = สัญญาณขาออกของเทอมอนุพันธ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.5.1 ระบบควบคุมแบบพี (สัดส่วน)

เทอมของสัดส่วน (บางครั้งเรียก อัตราขยาย) จะเปลี่ยนแปลงเป็นสัดส่วนของค่าความผิดพลาด การตอบสนองของสัดส่วนสามารถทำได้โดยการคูณค่าความผิดพลาดด้วยค่าคงที่ K_p, หรือที่เรียกว่าอัตราขยายสัดส่วน

เทอมของสัดส่วนจะเป็นไปตามสมการที่ 2.9

$$P_{out} = K_p e(t) \qquad (2.9)$$

เมื่อ

P_{out} = สัญญาณขาออกของเทอมสัดส่วน

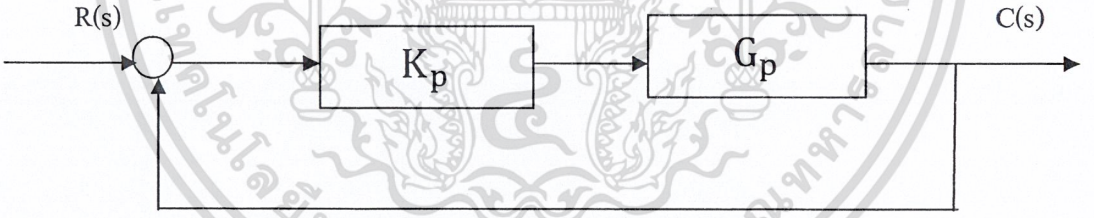
K_p = อัตราขยายสัดส่วน, ตัวแปรปรับค่าได้

e = ค่าความผิดพลาด = SP − PV

t = เวลา

และจะได้ฟังก์ชันถ่ายโอนของระบบควบคุมแบบพีดังสมการที่ 2.10

$$G_c(s) = K_p$$



บล็อกไดอะแกรมของระบบควบคุมแบบพี

ข้อดีของตัวควบคุมแบบสัดส่วน

1. ลดผลตอบสนองส่วนที่เป็นค่าชั่วครู่ทำให้มีผลตอบสนองที่ไวขึ้น

ข้อเสียของตัวควบคุมแบบสัดส่วน

2. ถ้าใช้งานเดี่ยวๆ จะไม่สามารถขจัด Steady-state error ได้ซึ่งสามารถลดผลกระทบได้ด้วยการเพิ่มค่า Gain ให้มีค่าสูงๆแต่การเพิ่มค่า Gain ให้มีค่าสูงอาจจะทำให้เกิด Overshoot ที่สูงขึ้นด้วยและการเพิ่มค่า Gain ให้มีค่าสูงอาจจะทำไม่ได้ในทางปฏิบัติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.5.2 ระบบควบคุมแบบไอ (ปริพันธ์)

ผลจากเทอมปริพันธ์ (บางครั้งเรียก reset) เป็นสัดส่วนของขนาดความผิดพลาดและระยะเวลาของความผิดพลาด ผลรวมของความผิดพลาดในทุกช่วงเวลา (ปริพันธ์ของความผิดพลาด) จะให้ออฟเซตสะสมที่ควรจะเป็นในก่อนหน้า ความผิดพลาดสะสมจะถูกคูณโดยอัตราขยายปริพันธ์ ขนาดของผลของเทอมปริพันธ์จะกำหนดโดยอัตราขยายปริพันธ์, K_i,

เทอมปริพันธ์จะเป็นไปตามสมการ

$$I_{out} = K_i \int_0^t e(\tau)\, d\tau \qquad (2.10)$$

เมื่อ

I_{out} = สัญญาณขาออกของเทอมปริพันธ์

K_i = อัตราขยายปริพันธ์, ตัวแปรปรับค่าได้

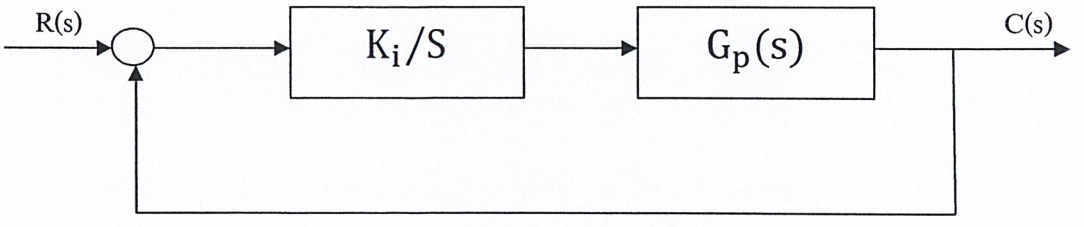
e = ความผิดพลาด = SP − PV

t = เวลา

τ = ตัวแปรปริพันธ์หุ่น

และจะได้ฟังก์ชันถ่ายโอนของระบบควบคุมแบบไอ:

$$G_c(s) = K_i/S \qquad (2.11)$$



บล็อกไดอะแกรมของระบบควบคุมแบบไอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ข้อดีของตัวควบคุมแบบไอ

1. ขจัด Steady-state error ได้

ข้อด้อยของตัวควบคุมแบบปริพันธ์

1. ไม่สามารถใช้งานเดี่ยวๆได้ต้องใช้งานร่วมกับตัวควบคุมแบบสัดส่วน
2. ไม่สามารถขจัด overshoot ได้
3. การใช้ค่า Gain ที่ไม่เหมาะสมอาจทำให้ผลตอบสนองเกิดการแกว่งได้

2.5.3 ระบบควบคุมแบบดี (อนุพันธ์)

อัตราการเปลี่ยนแปลงของความผิดพลาดจากกระบวนการนั้นคำนวณหาจากความชันของความผิดพลาดทุกๆเวลา (นั่นคือ เป็นอนุพันธ์อันดับหนึ่งสัมพันธ์กับเวลา) และคูณด้วยอัตราขยายอนุพันธ์ K_d ขนาดของผลของเทอมอนุพันธ์ (บางครั้งเรียก อัตรา) ขึ้นกับ อัตราขยายอนุพันธ์ K_d

เทอมอนุพันธ์เป็นไปตามสมการ

$$D_{out} = K_d \frac{d}{dt} e(t) \tag{2.12}$$

เมื่อ

D_{out} = สัญญาณขาออกของเทอมอนุพันธ์

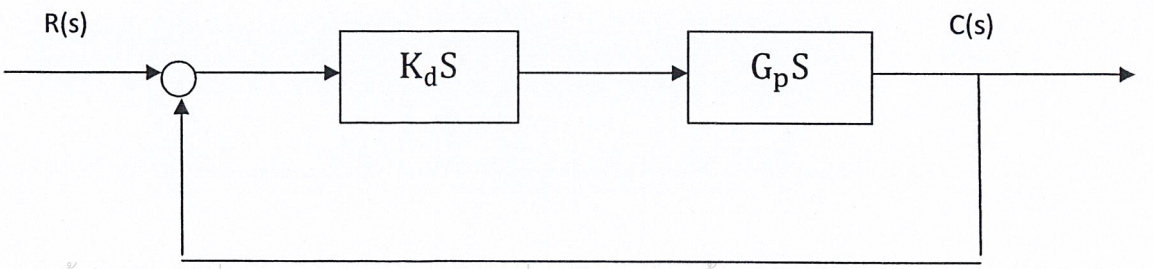
K_d = อัตราขยายอนุพันธ์, ตัวแปรปรับค่าได้

e = ความผิดพลาด = SP − PV

t = เวลา

จะได้ฟังก์ชันถ่ายโอนระบบควบคุมแบบดี

$$G_c(s) = K_d s \tag{2.13}$$



บล็อกไดอะแกรมของระบบควบคุมแบบดี

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ข้อดีของตัวควบคุมแบบอนุพันธ์

1. ลดผลตอบสนองที่เป็น Overshoot

ข้อด้อยของตัวควบคุมแบบอนุพันธ์

1.ไม่สามารถใช้งานเดี่ยวๆได้ต้องใช้งานร่วมกับตัวควบคุมแบบสัดส่วน

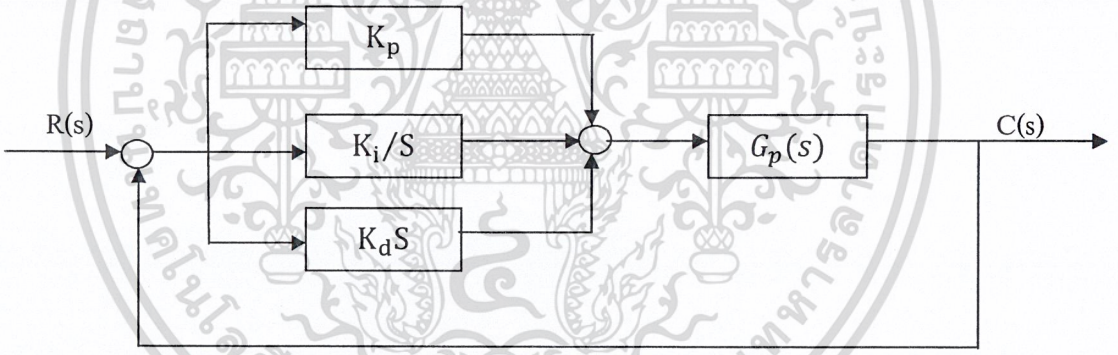
2.ไม่สามารถขจัด Steady-state error ได้

2.5.4 สมการผลรวมของพีไอดี

เทอมสัดส่วน ปริพันธ์ และอนุพันธ์ จะนำมารวมกันเป็นสัญญาณขาออกของการควบคุมแบบพีไอดี กำหนดให้ u(t) เป็นสัญญาณขาออก

สมการสุดท้ายของวิธีพีไอดี คือ

$$u(t) = MV(t) = K_p e(t) + K_i \int_o^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t) \qquad (2.14)$$



บล็อกไดอะแกรมของระบบควบคุมแบบพีไอดี

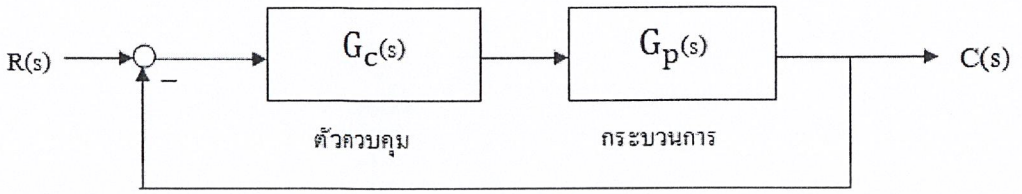
2.6 ระบบควบคุมแบบป้อนกลับ (Feedback Control System)

ระบบควบคุมแบบป้อนกลับ คือ ระบบควบคุมที่มีการนำเอาเอาท์พุทที่ได้จากกระบวนการกลับเข้ามาเป็นส่วนหนึ่งของข้อมูลที่จะนำมาพิจารณาเป็นอินพุทที่จะให้กับกระบวนการนั้น โดยมักมีโครงสร้างดังรูปที่ 2.12 การที่จะทราบค่าเอาท์พุทได้จะต้องมีการวัดข้อมูลของเอาท์พุท เมื่อทราบค่าเอาท์พุทแล้วจะนำเอาค่าเอาท์พุทที่ได้ไปเปรียบเทียบกับค่าอ้างอิงที่ต้องการ เพื่อหาค่า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

ความแตกต่างระหว่างค่าที่ได้จริงกับค่าที่ต้องการ แล้วส่งสัญญาณไปสู่ตัวควบคุมซึ่งประมวลผลเพื่อหาอินพุทที่จะป้อนให้กระบวนการ



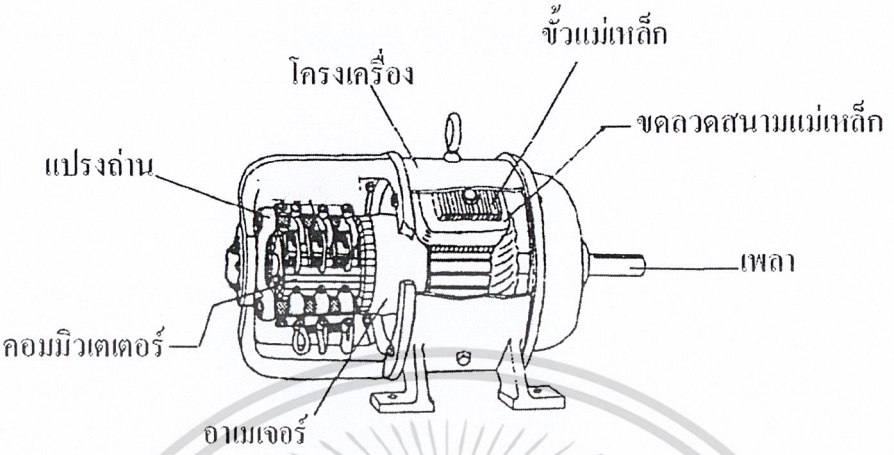
รูปที่ 2.12 บล็อกไดอะแกรมของระบบควบคุมแบบป้อนกลับ

2.7 มอเตอร์ไฟฟ้ากระแสตรง

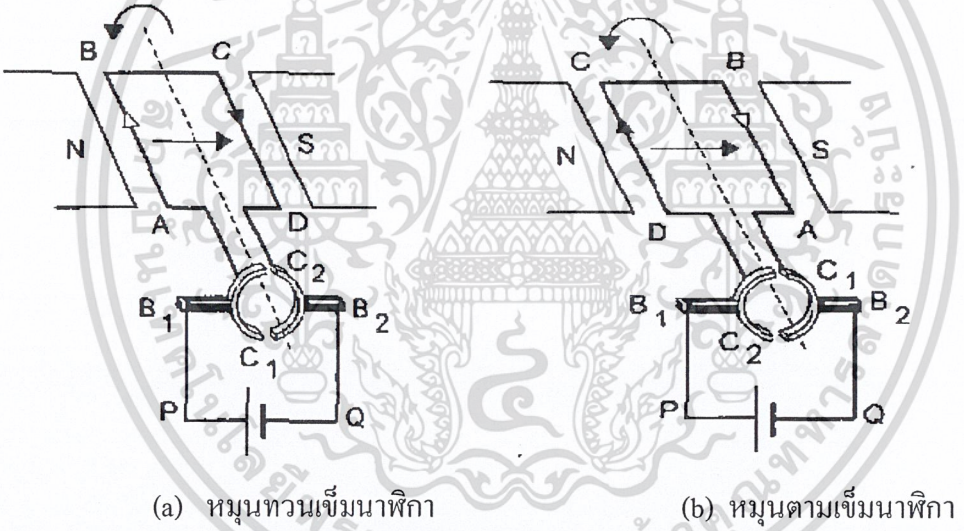
มอเตอร์ไฟฟ้ากระแสตรง (DC motor) คือ อุปกรณ์ที่ใช้เปลี่ยนพลังงานไฟฟ้าเป็นพลังงานกล เพื่อนำพลังงานกลที่ได้ไปขับเคลื่อนสิ่งต่างๆ ตามต้องการ มอเตอร์ไฟฟ้ากระแสตรงนั้นจะใช้การขับเคลื่อนในแบบที่มีอัตราเร็วไม่สูงมากนัก เนื่องจากมอเตอร์ไฟฟ้ากระแสตรงนั้นมีแรงบิดเริ่มต้นที่สูง (Starting torque) สามารถควบคุมอัตราเร็วได้ค่อนข้างง่าย แต่มีข้อเสีย คือ มีโครงสร้างที่ค่อนข้างซับซ้อนมาก จึงไม่เหมาะที่จะใช้ในงานที่มีอัตราเร็วค่อนข้างสูงมากๆ แรงทางกลที่เกิดขึ้นอาศัยหลักการที่ว่า เมื่อกระแสไหลในตัวนำซึ่งอยู่ในสนามแม่เหล็กก็จะทำให้เกิดแรงดันไฟฟ้าเหนี่ยวนำเกิดขึ้นในขดลวดและขดลวดตัวนำจะหมุนไปตามสนามแม่เหล็กที่เกิดขึ้นโดยการดูดของขั้วแม่เหล็กที่ต่างกันผลักกันระหว่างวงขั้วแม่เหล็กที่เหมือนกัน ซึ่งแรงดันไฟฟ้าเหนี่ยวนำที่เกิดขึ้นจะทำให้เพลาของมอเตอร์หมุน ก็จะได้พลังงานกลไปใช้งานแต่ก็จะมีแรงดันไฟฟ้าเหนี่ยวนำเกิดขึ้นในทิศทางตรงกันข้ามกับแรงดันๆไฟฟ้าที่จ่ายจากภายนอก แรงดันไฟฟ้าเหนี่ยวนำที่เกิดขึ้นนี้ เรียกว่า แรงดันไฟฟ้าต้านกลับ (Counter e.m.f หรือ Back e.m.f)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
117583

2.8 โครงสร้างและหลักการทำงานของมอเตอร์ไฟฟ้ากระแสตรง



รูปที่ 2.13 โครงสร้างของมอเตอร์ไฟฟ้ากระแสตรง



(a) หมุนทวนเข็มนาฬิกา (b) หมุนตามเข็มนาฬิกา

รูปที่ 2.14 การทำงานของมอเตอร์ไฟฟ้ากระแสตรง

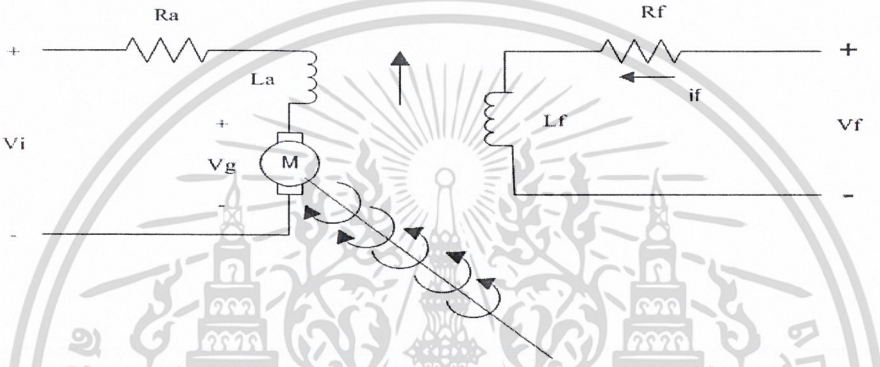
รูปที่ 2.13 และรูปที่ 2.14 อธิบายโครงสร้างและหลักการทำงานของมอเตอร์ไฟฟ้ากระแสตรงซึ่งประกอบด้วยขดลวดที่วางอยู่ระหว่างขั้วแม่เหล็ก โดยปลายของขดลวดทั้งสองข้างต่อเข้ากับคอมมิวเตเตอร์ (Commutator) ด้านล่ะซี่ ซึ่งจะมีแปรงถ่านต่อไว้และแปรงถ่านทั้งสองต่อเข้ากับแหล่งจ่ายไฟฟ้ากระแสตรงจากภายนอก ขดลวดตัวนำนั้นจะต้องหมุนอยู่ในสนามแม่เหล็กเมื่อมีการผ่านกระแสไฟฟ้าเข้าไปยังขดลวด ในสนามแม่เหล็กจะทำให้เกิดแรงแม่เหล็กซึ่งมีสัดส่วนของแรงขึ้นกับกระแสแรงของสนามแม่เหล็ก โดยแรงจะเกิดขึ้นเป็นมุมฉากกับกระแสสนามแม่เหล็ก ขณะที่ทิศทางของแรงกลับตรงกันข้ามกัน ถ้าหากกระแสของสนามแม่เหล็กไหลย้อนกลับจะทำให้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เกิดการเปลี่ยนแปลงของกระแสและสนามแม่เหล็ก เป็นผลทำให้ทิศทางของแรงเปลี่ยนไปด้วย คุณสมบัตินี้ทำให้มอเตอร์ไฟฟ้ากระแสตรงกลับทิศทางการหมุนได้

สนามแม่เหล็กของมอเตอร์ส่วนหนึ่งเกิดขึ้นจากแม่เหล็กถาวร ซึ่งจะถูกยึดติดกับแผ่นสนามแม่เหล็กหรือเหล็กกล้า โดยปกติส่วนนี้จะเป็นส่วนที่ยึดกับที่ และขดลวดเหนี่ยวนำจะพันอยู่กับส่วนที่เป็นแกนหมุนของมอเตอร์

2.9 คุณสมบัติของมอเตอร์ไฟฟ้ากระแสตรง



รูปที่ 2.15 วงจรภายในของมอเตอร์ไฟฟ้ากระแสตรง

จากรูปที่ 2.15 จะได้ว่าแรงดันไฟฟ้าที่มีผลต่อการใช้งานจริงของมอเตอร์จึงมีค่าเท่ากับแรงดันไฟฟ้าที่จ่ายให้ลบด้วยแรงดันไฟฟ้าต้านกลับ เขียนเป็นสมการได้ คือ

$$I_a R_a = V - E_b \tag{2.15}$$

หรือ

$$E_b = V - I_a R_a \tag{2.16}$$

เมื่อ E_b คือ แรงดันไฟฟ้าต้านกลับ (V)

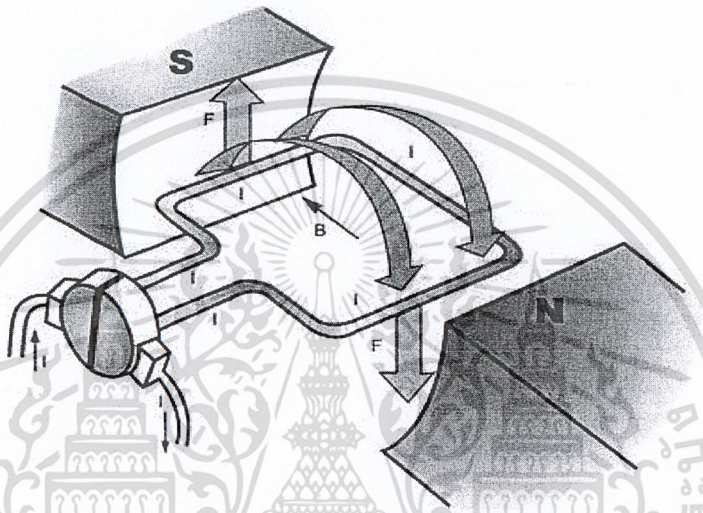
V คือ แรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์ (V)

I_a คือ กระแสที่ไหลในอาร์เมเจอร์ (A)

R_a คือ ความต้านทานของขดลวดอาร์เมเจอร์ (Ω)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ค่าแรงดันไฟฟ้าต้านกลับนี้จะมีค่าไม่เท่ากับแรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์ แรงดันไฟฟ้าต้านกลับที่เกิดขึ้น สามารถเขียนเปรียบเทียบได้เหมือนกับในอาร์เมเจอร์นั้นมีแหล่งจ่ายไฟฟ้ากระแสตรงซ่อนอยู่ และจ่ายไฟออกมาตรงกันข้ามกับแรงดันไฟฟ้าที่จ่ายเข้าไป เมื่อมีกระแสไหลผ่านขดลวดอาร์เมเจอร์ที่วางอยู่ในสนามแม่เหล็กจะเกิดแรงไฟฟ้าเหนี่ยวนำ ทำให้มีแรงผลักดันที่อาร์เมเจอร์ เกิดแรงบิดที่สม่ำเสมอราบเรียบไม่กระตุก เพราะมีขดลวดหลายๆ ชุดที่ติดอยู่กันผลัดกันทำงานทีละชุดตามลำดับ



รูปที่ 2.16 แรงที่กระทำกับขดลวด

จากรูปที่ 2.16 แรงที่กระทบบนขดลวดตัวนำต่อหนึ่งขดในกระแสไหลผ่านตัวนำนั้นจะเป็นสัดส่วนโดยตรงกับจำนวนกระแสที่ไหล ความเข้มของสนามแม่เหล็กและความยาวของตัวนำในส่วนที่ตัดผ่านสนามแม่เหล็ก (หรือความยาวของตัวนำส่วนที่ใช้งานจริง) สามารถเขียนเป็นสมการได้ดังนี้

$$F = BI_aL \tag{2.17}$$

เมื่อ F คือ แรงที่กระทำที่ขดลวดตัวนำ (N)

B คือ ความหนาแน่นของเส้นแรงแม่เหล็ก (W_b/m^2)

L คือ ความยาวของตัวนำในส่วนที่ผ่านสนามแม่เหล็ก (m)

I_a คือ กระแสที่ไหลในขดลวดตัวนำ (A)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แรงที่กระทำ F นี้จะยังส่งผลให้เกิดแรงบิดซึ่งแรงบิด (Torque) คือ การหมุนหรือการบิดของโมเมนต์ของแรงๆ หนึ่งที่กระทำรอบๆ แกนของมอเตอร์มีขนาดเท่ากับผลคูณของแรงกันรัศมีเขียนเป็นสมการได้ดังนี้

$$T = Fr \tag{2.18}$$

เมื่อ T คือ แรงบิดที่เกิดขึ้น (N-m)

r คือ ระยะห่างระหว่างจุดศูนย์กลางของเพลาถึงตัวนำ (M)

คุณลักษณะการทำงานของมอเตอร์ไฟฟ้ากระแสตรงเมื่อพิจารณาในกรณีความสัมพันธ์ของแรงบิดกันกระแสที่ไหลในอาร์เมเจอร์ จะได้ว่าแรงบิดที่เกิดขึ้นจากมอเตอร์ไฟฟ้ากระแสตรงนั้นขึ้นอยู่กับกระแสที่ไหลในอาร์เมเจอร์และเส้นแรงแม่เหล็กที่เกิดขึ้นจากฟิลด์คอยล์หรือจากขั้วแม่เหล็กถาวรสามารถเขียนเป็นความสัมพันธ์ได้ คือ

$$T = K_e \phi I_a \tag{2.19}$$

เมื่อ K_e คือ ค่าคงที่ของมอเตอร์

ϕ คือ เส้นแรงแม่เหล็ก (W_b)

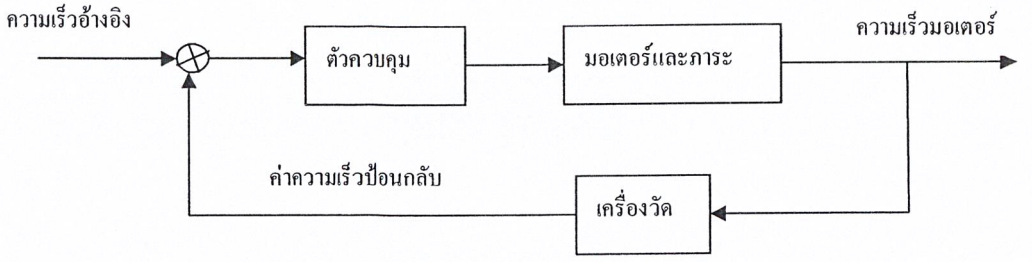
2.10 การควบคุมความเร็วมอเตอร์ไฟฟ้ากระแสตรง

2.10.1 หลักการควบคุมความเร็ว

รูปที่ 2.17 แสดงบล็อกไดอะแกรมของระบบควบคุมความเร็วของมอเตอร์ เครื่องวัดรอบในลูปแบบป้อนกลับจะวัดความเร็วของมอเตอร์ แล้วส่งป้อนกลับมาในรูปของสัญญาณไฟฟ้า (แรงดันหรือ กระแส) ซึ่งแปรตามความเร็วเพลาของมอเตอร์ในที่นี้ลูปป้อนกลับจะทำให้ความเร็วเอาท์พุทของมอเตอร์มีค่าคงที่มากขึ้น

ระบบควบคุมแบบป้อนกลับถูกใช้รวมอยู่ในระบบเครื่องมือ เครื่องจักร หรือสว่านกำลังเพื่อชดเชยความแตกต่างของโหลด (วัสดุที่ถูกตัดหรือถูกเจาะ) วัสดุเนื้อแข็งจะหน่วงความเร็วของสว่าน หรือเครื่องมือที่ใช้ในการตัดและสว่านจะมีความเร็วเพิ่มขึ้นในวัสดุเนื้ออ่อนในลูปป้อนกลับความเร็วมอเตอร์ ที่ประกอบด้วยเครื่องวัดความเร็ว ความเร็วของเครื่องมือจับคงมีค่าคงที่เนื่องจากเมื่อเครื่องมือที่ใช้ในการตัดมีความเร็วลดลง สัญญาณป้อนกลับจะควบคุมมอเตอร์ให้เพิ่มความเร็วขึ้นในขณะเดียวกันเมื่อเครื่องมือตัดชิ้นงานที่เป็นวัสดุอ่อน ลูปป้อนกลับจะป้องกันไม่ให้มอเตอร์เร่งความเร็วเกินขนาด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.17 ระบบควบคุมความเร็วที่ประกอบด้วยลูปการควบคุมป้อนกลับเพียงลูปเดียว

2.10.2 ประเภทของการควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง

แบ่งออกเป็น 3 ประเภทดังนี้

1.การควบคุมด้วยตัวต้านทานที่ปรับค่าได้

2.การควบคุมด้วยวิธีเปลี่ยนค่าแรงดัน

3.การควบคุมความกว้างพัลส์

2.10.2.1 การควบคุมด้วยตัวต้านทานที่ปรับค่าได้

เป็นรูปแบบพื้นฐานที่สดของการควบคุมมอเตอร์ คือ ใช้ตัวต้านทานปรับค่าได้อนุกรมกับมอเตอร์ โดยตัวต้านทานที่ปรับค่าได้จะเป็นตัวกำหนดความเร็วในการหมุนของมอเตอร์ การบังคับแบบนี้ไม่มีประสิทธิภาพเพราะกำลังไฟฟ้าสูญเสียไปในตัวต้านทาน มักนิยมใช้กับมอเตอร์ตัวเล็กๆ การบังคับแบบนี้ให้คุณสมบัติการสตาร์ทดี (ให้แรงบิดที่สูงความเร็วที่ต่ำ) แต่จะให้ความเร็วสูงมากเมื่อมอเตอร์อยู่ในภาวะที่มีโหลดน้อยๆ ดังนั้นการบังคับแบบนี้มีประโยชน์เฉพาะภาวะที่แรงต้านคงที่ เช่น การบังคับความเร็วของเครื่องจักรเย็บผ้า เป็นต้น

2.10.2.2 การควบคุมด้วยวิธีเปลี่ยนค่าแรงดัน

วิธีการนี้ดีกว่าวิธีการแรกแต่จะซับซ้อนกว่าต้องใช้อุปกรณ์อิเล็กทรอนิกส์ที่มีอัตราขยายกำลังสูง และมอเตอร์จะถูกป้อนด้วยแรงดันที่เปลี่ยนแปลงค่าได้ จากแหล่งจ่ายที่มีอิมพีแดนซ์ต่ำ ข้อดีของการควบคุมวิธีนี้ คือ ถ้าความเร็วลดลงจากผลของแรงบิด แรงดันที่ป้อนให้กับมอเตอร์จะเพิ่มขึ้นเพื่อรักษาระดับความเร็ว ส่วนข้อเสียจากการควบคุมวิธีนี้ คือ เมื่อมอเตอร์มีความเร็วต่ำแรงดันที่ป้อนกลับให้กับมอเตอร์จะมีค่าต่ำเช่นกัน

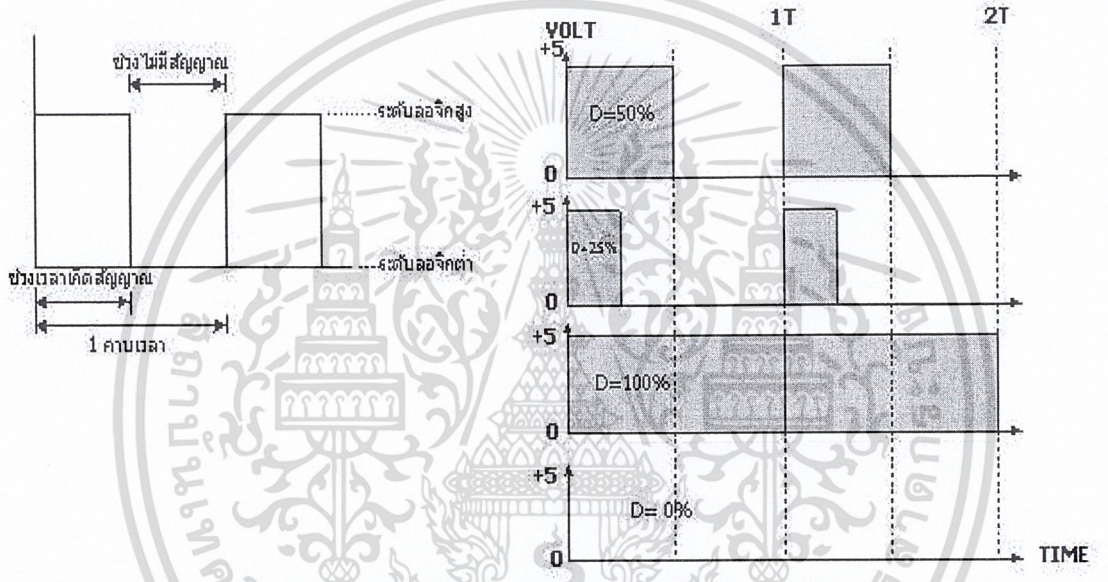
2.10.2.3 การควบคุมความเร็วของมอเตอร์แบบการควบคุมความกว้างของสัญญาณพัลส์

การควบคุมความกว้างพัลส์ (Pulse Width Modulation : PWM) คือ การปรับเปลี่ยนที่สัดส่วน และความกว้างของสัญญาณพัลส์ โดยความถี่ของสัญญาณพัลส์จะไม่มีการเปลี่ยนแปลงหรือเป็นการเปลี่ยนแปลงที่ค่าดิวตี้ไซเคิล (Duty cycle) นั่นเอง ซึ่งค่าดิวตี้ไซเคิล คือ ช่วงความกว้างของพัลส์ที่มีสถานะลอจิกสูง โดยคิดสัดส่วนเป็นเปอร์เซ็นต์จากความกว้างของพัลส์ทั้งหมด ยกตัวอย่างเช่น ถ้าหากค่าดิวตี้ไซเคิลมีค่าเท่ากับ 50% ก็หมายถึงในแต่ละรูปสัญญาณพัลส์จะมีช่วง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สัญญาณที่เป็นสถานะลอจิกสูงอยู่ครึ่งหนึ่งและสถานะลอจิกต่ำอยู่อีกครึ่งหนึ่ง ดังรูปที่ 2.18 และในทำนองเดียวกันถ้าหากค่าดิวตี้ไซเคิลมีค่ามาก หมายความว่า ความกว้างของพัลส์ที่เป็นสถานะลอจิกสูงจะมีความกว้างมากขึ้น หากค่าดิวตี้ไซเคิ้ลมีค่าเท่ากับ 100% ก็หมายความว่าจะไม่มีสถานะลอจิกต่ำเลย ซึ่งค่าดิวตี้ไซเคิ้ลสามารถหาได้จากค่าความสัมพันธ์ดังนี้

$$\text{ค่าดิวตี้ไซเคิ้ล} = \frac{\text{ช่วงของสัญญาณพัลส์}}{\text{คาบเวลาทั้งหมดของสัญญาณ}} \text{ X } 100\,\% \qquad (2.20)$$



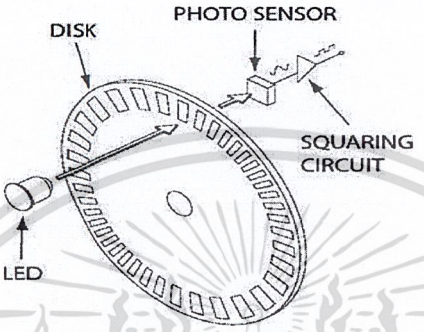
รูปที่ 2.18 สัญญาณ PWM ซึ่งแสดงค่าดิวตี้ไซเคิล ที่ต่างๆกัน

เนื่องจากโครงงานนี้ได้จัดทำเครื่องเฉือนเนื้อทุเรียน ซึ่การควบคุมความเร็วของใบมีดที่ใช้ในการเฉือนเนื้อทุเรียน คือ ถ้าความเร็วลดลงจากผลของแรงบิด แรงดันที่ป้อนให้กับมอเตอร์จะเพิ่มขึ้นเพื่อรักษาระดับความเร็ว ดังนั้นในโครงงานนี้จึงเลือกใช้การควบคุมความเร็วมอเตอร์แบบเปลี่ยนค่าแรงดันไฟฟ้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

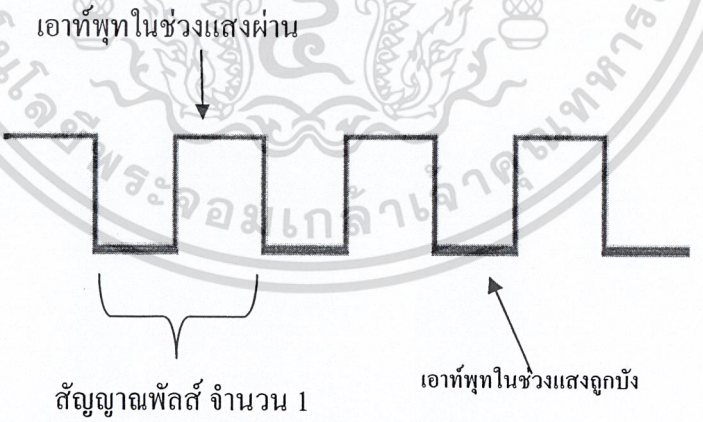
2.11 เอ็นโค้ดเดอร์ (Encoder)

เอ็นโค้ดเดอร์มีลักษณะเป็นแผ่นกลมมีแกนอยู่ตรงกลาง และที่แผ่นกลมจะมีช่องเล็กที่แสงสามารถส่องผ่านได้เป็นจำนวนมาก เรียกว่า ช่องสลิท (Slit) ซึ่งที่ด้านหนึ่งของแผ่นกลมนี้จะมีตัวส่งแสงอินฟาเรด (Infared) ไปยังตัวรับสัญญาณแสงอินฟาเรดซึ่งจะอยู่ในด้านตรงกันข้าม ดังรูปที่ 2.19



รูปที่ 2.19 โครงสร้างของเอ็นโค้ดเดอร์

เมื่อแกนหมุนหมุนทำให้แผ่นกลมหมุนไปตัดลำแสงอินฟาเรด ดังนั้นชุดรับแสงอินฟาเรดจึงมีแสงมากระทบเป็นช่วงๆ ตามจังหวะที่แสงผ่านช่องสลิทจึงทำให้สัญญาณเอาท์พุทของชุดรับแสงอินฟาเรดมีลักษณะเป็นสัญญาณพัลส์ ดังรูปที่ 2.20



รูปที่ 2.20 การสร้างพัลส์ของเอ็นโค้ดเดอร์

จำนวนพัลส์ที่ได้ออกมานี้ จะเป็นตัวที่บ่งชี้ว่ามอเตอร์หมุนไปกี่องศา หรือกี่รอบ ซึ่งสามารถคำนวณได้จากความสัมพันธ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$\text{จำนวนรอบที่มอเตอร์หมุน} = \frac{\text{จำนวนพัลส์}}{\text{ความละเอียดของเอ็นโค้ดเดอร์}} \quad (2.21)$$

โดยที่ค่าความละเอียดของเอ็นโค้ดเดอร์ใช้หน่วยเป็นจำนวนพัลส์ต่อรอบ (pulse/round:ppr) เช่น 1000 ppr ก็หมายถึงว่า เมื่อมอเตอร์หมุนไป 1 รอบจะมีสัญญาณพัลส์ออกมา 1000 พัลส์ เป็นต้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
หลักการและการออกแบบ

การออกแบบระบบควบคุมเพื่อใช้ขับมอเตอร์กระแสตรงนั้น จำเป็นต้องพิจารณาระบบที่เหมาะสมเพื่อต้องการให้ระบบมีเสถียรภาพมากยิ่งขึ้น ดังนั้นคณะผู้จัดทำได้นำระบบฟัซซีเข้ามาทดสอบในการออกแบบระบบควบคุม เพื่อต้องการศึกษาการใช้งานจริงและหาระบบควบคุมที่เหมาะสม

3.1 หลักการในการออกแบบ

การทำฟัซซิฟิเคชัน กฎพื้นฐาน และดีฟัซซิฟิเคชัน

ขั้นตอนการทำฟัซซิฟิเคชัน ทำได้โดยใช้สองอินพุท คือ สัญญาณผิดพลาด e(nT) และอัตราการเปลี่ยนแปลงของสัญญาณผิดพลาด r(nT) ร่วมกับเอาท์พุทของตัวควบคุม $u_{pd}(nT)$ โดยที่อินพุทของตัวควบคุมฟัซซีพีดี คือ สัญญาณผิดพลาด (error) และ อัตราสัญญาณผิดพลาด (rate) โดยพื้นฐานของฟังก์ชันความเป็นสมาชิกได้แสดงตามกฎการควบคุมต่อไปนี้

(r1) If error = ep AND rate = rp THEN output = oz

(r2) If error = ep AND rate = rn THEN output = op

(r3) If error = en AND rate = rp THEN output = on

(r4) If error = en AND rate = rn THEN output = oz

โดยที่

ep คือ ค่าสัญญาณผิดพลาดทางด้านบวก

en คือ ค่าสัญญาณผิดพลาดทางด้านลบ

rp คือ ค่าอัตราการเปลี่ยนแปลงของสัญญาณผิดพลาดทางด้านบวก

rn คือ ค่าอัตราการเปลี่ยนแปลงของสัญญาณผิดพลาดทางด้านลบ

oz คือ ค่าสัญญาณเอาท์พุทที่ศูนย์

op คือ ค่าสัญญาณเอาท์พุททางด้านบวก

on คือ ค่าสัญญาณเอาท์พุททางด้านลบ

และเอาท์พุทของระบบฟัซซีพีดี คือ $\Delta U_{pd}(nT)$ โดยจะใช้ AND ซึ่งเป็นตรรกศาสตร์ Zadeh's logical "AND" สามารถหาได้จาก

$$\mu_A \text{ AND } \mu_B = \min\{\mu_A, \mu_B\} \qquad (3.1)$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กฎข้อที่ 1 (r1) เงื่อนไข ep (สัญญาณผิดพลาดเป็นค่าบวก) หมายถึง เอาท์พุทของระบบมีค่ามากกว่าจุดที่กำหนด (sp > y) และเงื่อนไข rp (อัตราการเปลี่ยนแปลงของสัญญาณผิดพลาดเป็นบวก) ในกรณีนี้เอาท์พุทของระบบก่อนหน้านี้มีลักษณะลดลง ดังนั้นตัวควบคุมจะปรับให้เอาท์พุทของระบบมีค่าเพิ่มขึ้นโดยอัตโนมัติ จากกฎข้อที่หนึ่งเอาท์พุทที่ได้เท่ากับศูนย์ (output = oz)

กฎข้อที่ 2 (r2) เงื่อนไข ep (สัญญาณผิดพลาดเป็นค่าบวก) หมายถึง เอาท์พุทมีค่ามากกว่าจุดที่กำหนด (sp > y) และเงื่อนไข rn (อัตราการเปลี่ยนแปลงของสัญญาณผิดพลาดเป็นลบ) ในกรณีนี้ตัวควบคุมขั้นก่อนหน้านี้มีการปรับให้เพิ่มขึ้น จากกฎข้องสองเอาท์พุทที่ได้มีค่าเป็นบวก (output = op)

กฎข้อที่ 3 (r3) เงื่อนไข en (สัญญาณผิดพลาดเป็นค่าลบ) หมายถึง เอาท์พุทมีค่าต่ำกว่าจุดที่กำหนด (sp > y) และเงื่อนไข rn (อัตราการเปลี่ยนแปลงของสัญญาณผิดพลาดเป็นบวก) ในกรณีนี้ตัวควบคุมก่อนหน้านี้มีการปรับให้ลดลง ดังนั้นทำการควบคุมเหมือนกฎข้อที่สอง จากกฎข้อที่สามเอาท์พุทที่ได้มีค่าเป็นลบ (output = on)

กฎข้อที่ 4 (r4) เงื่อนไข en (สัญญาณผิดพลาดเป็นค่าบวก) หมายถึง เอาท์พุทที่มีค่าต่ำกว่าจุดที่กำหนด (sp > y) และเงื่อนไข rn (อัตราการเปลี่ยนแปลงของสัญญาณผิดพลาดเป็นลบ) ในกรณีนี้ตัวควบคุมขึ้นก่อหน้านี้ได้ปรับให้ระบบเพิ่มขึ้น2 จากกฎขอที่สี่เอาท์พุทที่ได้เท่ากับศูนย์ (output = oz)

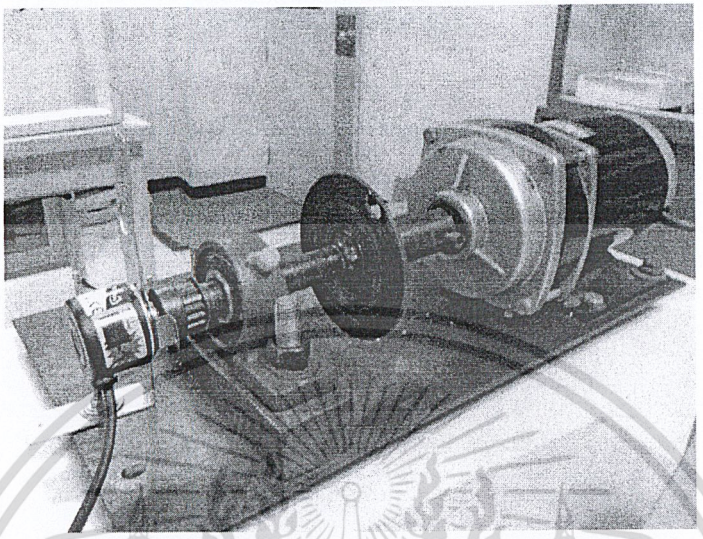
ในขั้นตอนการทำดีฟัซซีฟิเคชัน ใช้เป็นตรรกศาสตร์ Zadeh's logical "AND" และฟังก์ชันความเป็นสมาชิก ค่าที่ใช้โดยทั่วไปใช้หลักจุดกึ่งกลางมวล (Center Of Mass) และนำมาใช้ในการดีฟัซซีฟิเคชันดังแสดงในสมการที่ 3.12 ดังนี้

$$\Delta u_{pd}(nT) = \frac{\sum\{\text{membership value of input} \times \text{membership value of output}\}}{\sum\{\text{membership value of point}\}} \qquad (3.1)$$

3.2 ระบบจำลองของเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี

ระบบจำลองเป็นการทดลองการทำงานของมอเตอร์เมื่อมีแรงมากระทำกับเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี โดยทำการทดลองการจำลองแรงที่มากระทำโดยใส่ตัวต้านทานกระเบื้องให้กับระบบให้เหมือนกับการที่ใบมีดของเครื่องเฉือนเนื้อทุเรียนกำลังตัดเนื้อทุเรียน โดยอุปกรณ์ในระบบจำลองดังรูปที่ 3.1

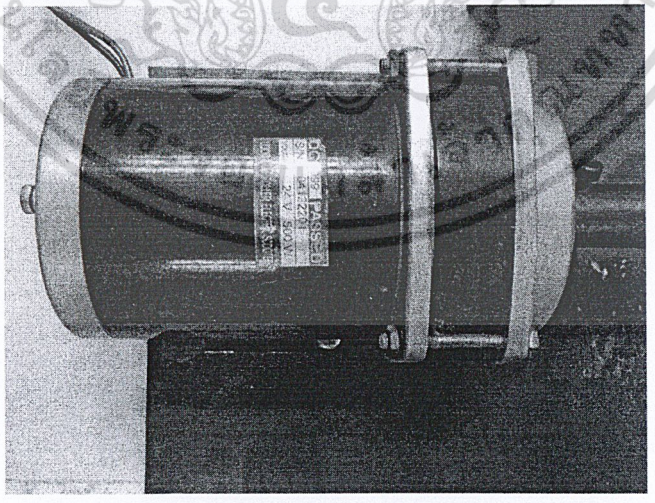
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.1 ระบบจำลองของเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี

3.2.1 มอเตอร์

เป็นมอเตอร์ที่ใช้จริงในเครื่องเฉือนเนื้อทุเรียนตามแนวแกนรัศมี เป็นมอเตอร์ไฟฟ้ากระแสตรง 24 V 27 A 500 W ดังรูปที่ 3.2

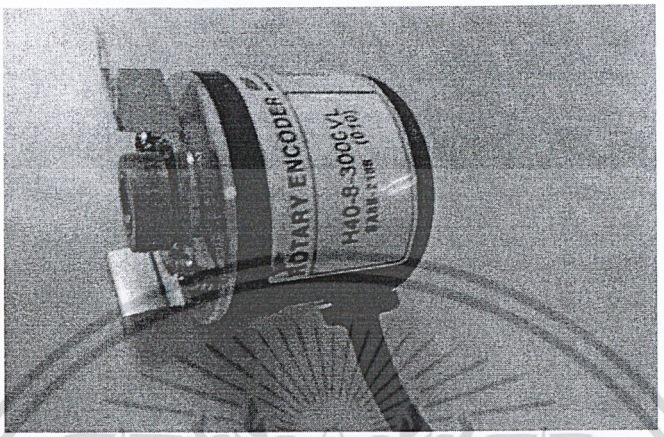


รูปที่ 3.2 มอเตอร์ไฟฟ้ากระแสตรง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2.2 ตัวอ่านค่าความเร็ว

เป็น Rotery Encoder Mecapion รุ่น H40-8-3000VL ซึ่งสามารถให้สัญญาณได้ถึง 3000 พัลส์/รอบ ดังรูปที่ 3.3



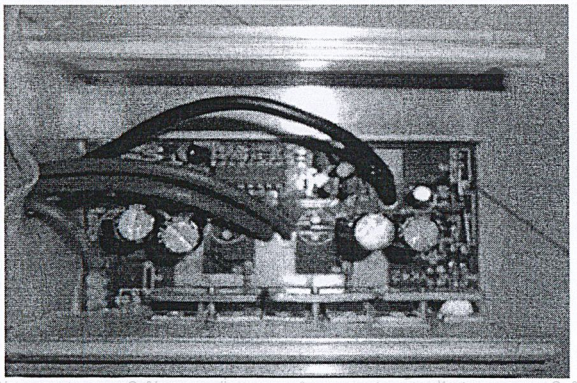
รูปที่ 3.3 เอ็นโค้ดเดอร์

3.2.3 วงจรไฟฟ้าในระบบจำลองของเครื่องเฉือนเนื้อทุเรียนแบบหมุนตามแนวแกนรัศมี

วงจรไฟฟ้าในระบบจำลองของเครื่องเฉือนเนื้อทุเรียนแบบหมุนตามแนวแกนรัศมี ประกอบด้วยวงจรขับมอเตอร์ วงจรควบคุม ซึ่งรายละเอียดและหน้าที่ของทุกวงจรประกอบด้วย ดังนี้

3.2.3.1 บอร์ดขับมอเตอร์

เป็นวงจรขับมอเตอร์กระแสตรงในการควบคุมมอเตอร์ไฟฟ้ากระแสตรง ทำให้สามารถควบคุมมอเตอร์ได้อย่างสมบูรณ์แบบ คือสามารถควบคุมความเร็วได้ด้วยอาศัยการหมุนค่าตัวต้านทาน ดังรูปที่ 3.4

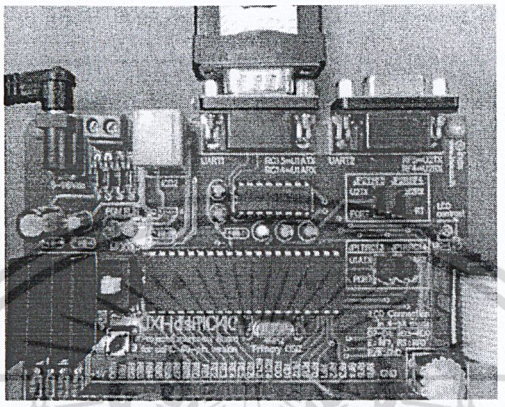


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 3.4 บอร์ดขับมอเตอร์

3.2.3.2 บอร์ดควบคุม

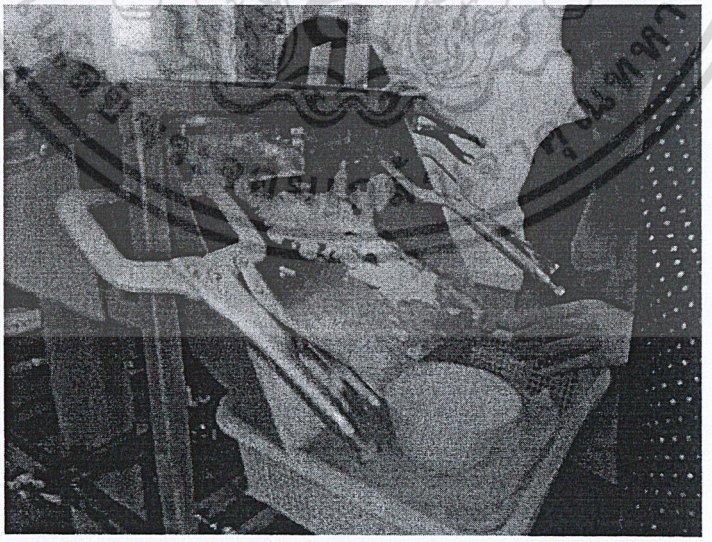
วงจรควบคุมความเร็วมอเตอร์ที่รับสัญญาณป้อนกลับจากเอ็นโค้ดเดอร์ เพื่อทำการปรับความเร็วรอบให้เป็นไปตามความต้องการ โดยใช้ไมโครคอนโทรลเลอร์ตระกูล dsPIC 30F4011 เป็นตัวประมวลผล ดังรูปที่ 3.5



รูปที่ 3.5 บอร์ดควบคุม

3.3 เครื่องเฉือนเนื้อทุเรียน

เครื่องเฉือนเนื้อทุเรียนได้ทำการออกแบบโดยนักศึกษาสาขาวิชาวิศวกรรมอาหาร ให้มีการเฉือนตามแนวแกนรัศมีให้ที่ป้อนเนื้อทุเรียนมาตามแนวดิ่ง ดังรูปที่ 3.6



รูปที่ 3.6 เครื่องเฉือนเนื้อทุเรียน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4

การทดลองและผลการทดลอง

การทดลองเครื่องเฉือนเนื้อทุเรียนได้แบ่งการทดลองออกเป็นภาคจำลองโหลด และภาคปฏิบัติจริงกับเครื่องเฉือนเนื้อทุเรียน โดยมีวิธีการทดลองดังต่อไปนี้

4.1 อุปกรณ์การทดลอง ประกอบด้วย

1.มอเตอร์ไฟฟ้ากระแสตรง 24 V 27 A 500 W

2.วงจรควบคุม dsPIC 30F4011

3.วงจงควบคุมมอเตอร์ไฟฟ้ากระแสตรง

4.แหล่งจ่ายกระแสไฟฟ้า

5.เอ็นโค้ดเดอร์ Rotery Encoder Mecapion รุ่น H40-8-3000VL

6.ตัวต้านทาน

7.เครื่องเฉือนเนื้อทุเรียน

4.2 วิธีการทดลอง

ขั้นตอนแรกให้มอเตอร์หมุนตามที่ต้องการ จากนั้นจึงทำการใส่โหลดให้กับมอเตอร์ หลังจากนั้นทำการเก็บผลทดลองเพื่อหาค่าที่เหมาะสมที่สุดในการควบคุมมอเตอร์ เพื่อให้ได้เนื้อทุเรียนที่มีประสิทธิภาพ

4.2.1 ตอนที่ 1 การทดลองภาคจำลองโหลด

1. จัดเตรียมอุปกรณ์การทดลอง
2. เตรียมตัวต้านทานเซรามิกขนาด 1 , 2 , 3 และ 5 Ω ขนาด 10 W
3. นำอุปกรณ์การทดลองติดตั้งเข้าด้วยกัน
4. ทำการหมุนมอเตอร์ด้วยโปรแกรมฟัซซีที่ความเร็วเริ่มต้นที่ 120 rpm
5. นำตัวต้านทานขนาด 1, 2 , 3 และ 5 Ω มาต่อคร่อมกับมอเตอร์ตามลำดับ
6. บันทึกค่าความเร็วออกมาเป็นกราฟด้วยโปรแกรม GUI Matlab
7. ทำการเปลี่ยนความเร็วมอเตอร์จาก 120 rpm เป็น 180, 250 และ 320 rpm ตามลำดับ
8. ทำการทดลองตามข้อ 5 และข้อ 6
9. เปลี่ยนโปรแกรมการหมุนของมอเตอร์จากระบบควบคุมฟัซซีเป็นระบบควบคุมพีไอ
10. ทำการทดลองตามข้อ 5 ถึง ข้อ 8

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

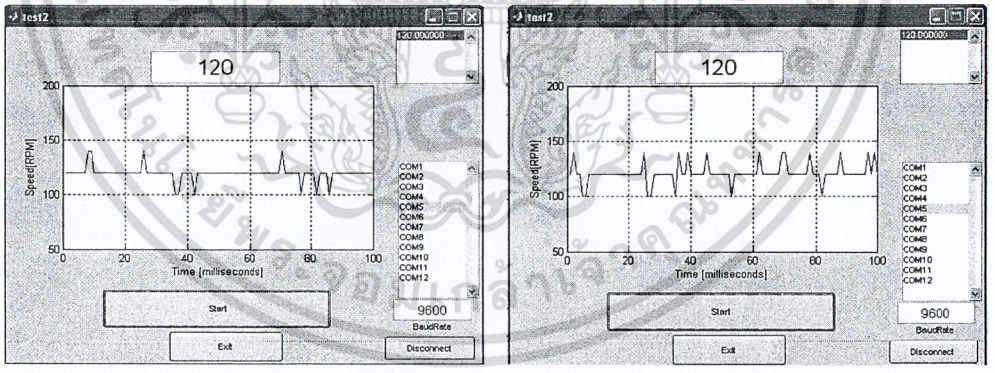
4.2.2 ตอนที่ 2 การทดลองภาคปฏิบัติจริงกับเครื่องเฉือนเนื้อทุเรียน

1. จัดเตรียมอุปกรณ์การทดลอง
2. ติดตั้งเอ็นโค้ดเดอร์เข้ากับเครื่องเฉือนเนื้อทุเรียน
3. ทำการหมุนมอเตอร์ด้วยโปรแกรมฟัซซีด้วยความเร็ว 120, 180, 250, 320 และ 400 rpm
4. บันทึกค่าความเร็วออกมาเป็นกราฟด้วยโปรแกรม GUI Matlab
5. นำเนื้อทุเรียนมาเฉือนด้วยความเร็ว 120 rpm
6. บันทึกค่าความเร็วขณะเฉือนเนื้อทุเรียนด้วยโปรแกรม GUI Matlab
7. ทำการทดลองตามข้อ 5 แต่เปลี่ยนความเร็วจาก 120 เป็น 180, 250, 320 และ 400 rpm
8. บันทึกค่าความเร็วจากกราฟด้วยโปรแกรม GUI Matlab
9. เปลี่ยนโปรแกรมจากระบบควบคุมฟัซซีเป็นระบบควบคุมพีไอ
10. ทำการทดลองตามข้อ 5 ถึง ข้อ 8

4.3 ผลการทดลอง

4.3.1 ผลการทดลองภาคจำลองโหลด

ที่ความเร็ว 120 rpm ไม่มีตัวต้านทาน

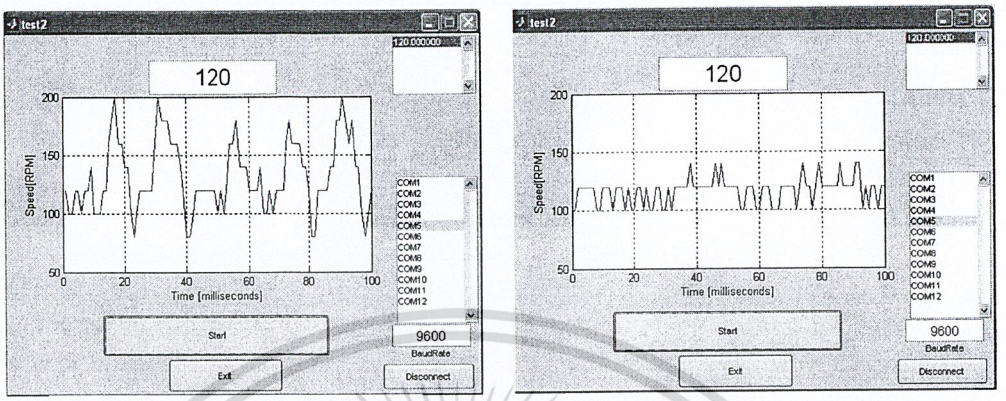


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.1 ผลการทดลองที่ความเร็ว 120 rpm ไม่มีตัวต้านทาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

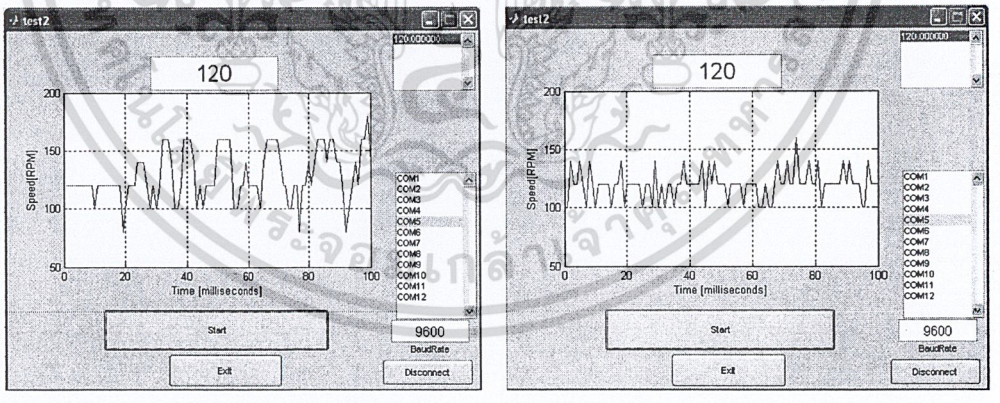
ที่ความเร็ว 120 rpm ตัวต้านทาน 1 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.2 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 1 Ω

ที่ความเร็ว 120 rpm ตัวต้านทาน 2 Ω

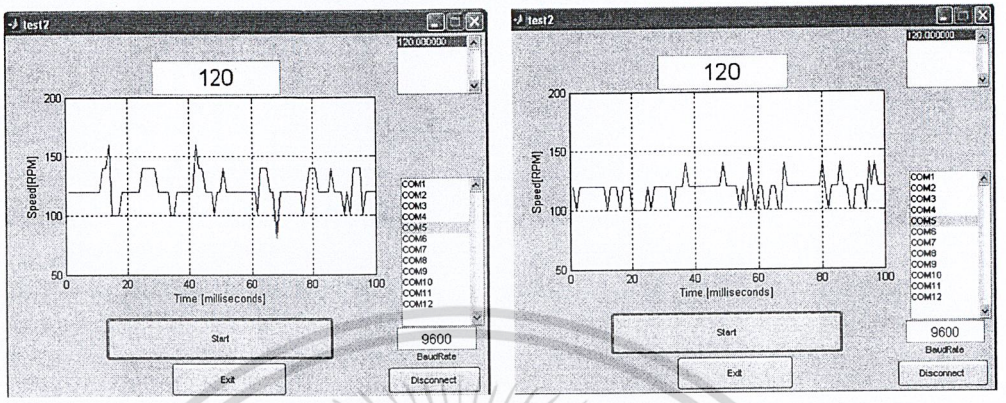


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.3 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 2 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

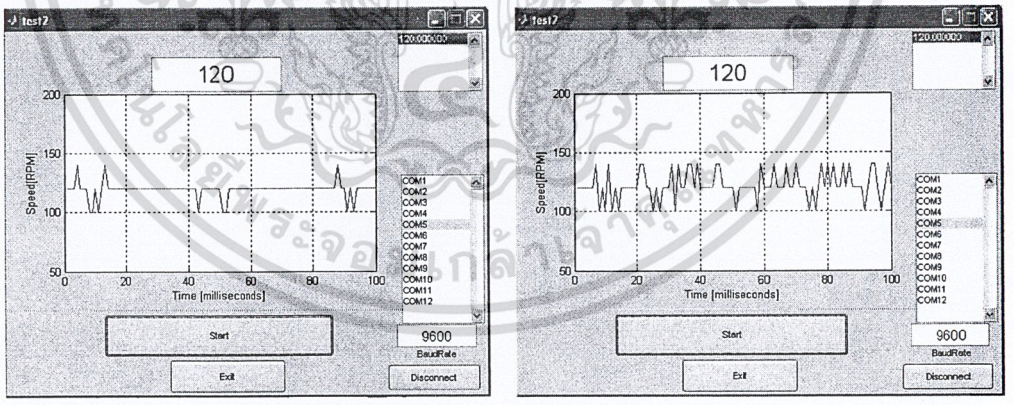
ที่ความเร็ว 120 rpm ตัวต้านทาน 3 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.4 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 3 Ω

ที่ความเร็ว 120 rpm ตัวต้านทาน 5 Ω

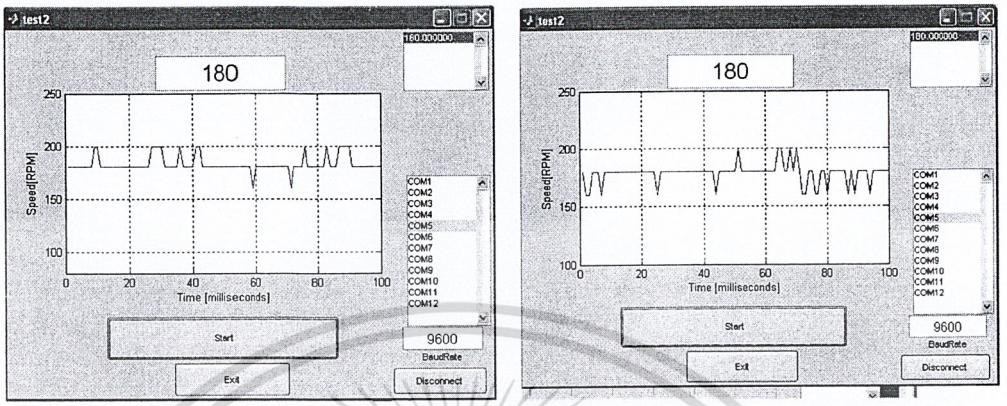


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.5 ผลการทดลองที่ความเร็ว 120 rpm มีตัวต้านทาน 5 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

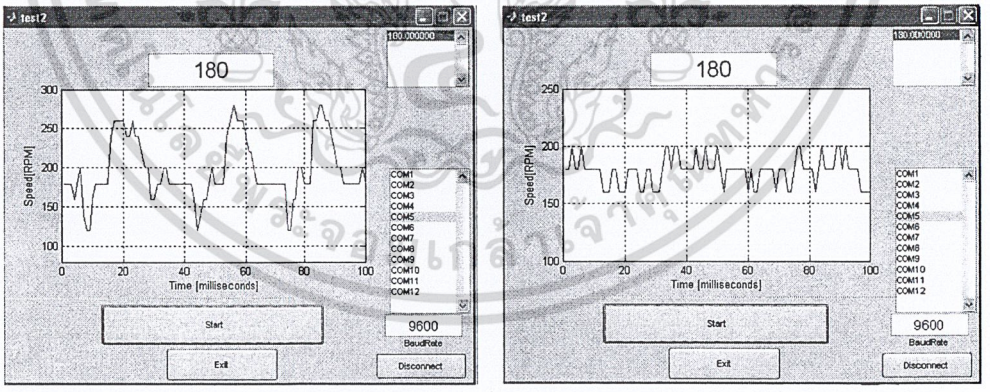
ที่ความเร็ว 180 rpm ไม่มีตัวต้านทาน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.6 ผลการทดลองที่ความเร็ว 180 rpm ไม่มีตัวต้านทาน

ที่ความเร็ว 180 rpm ตัวต้านทาน 1 Ω

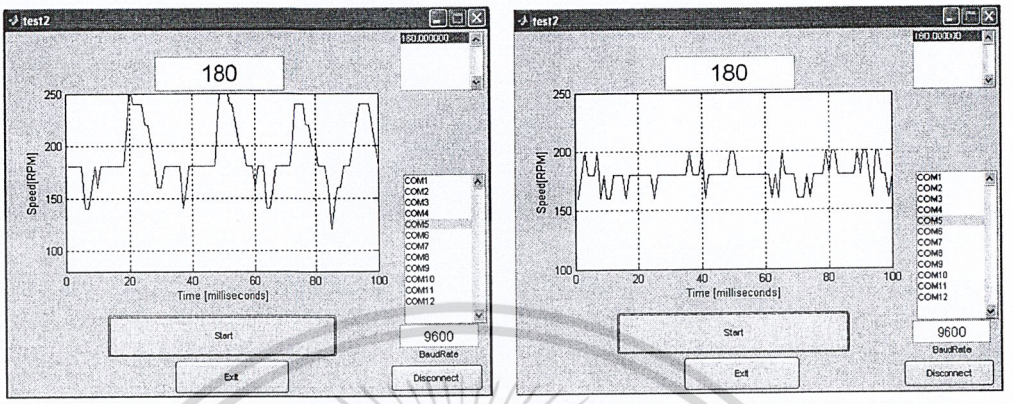


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.7 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 1 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

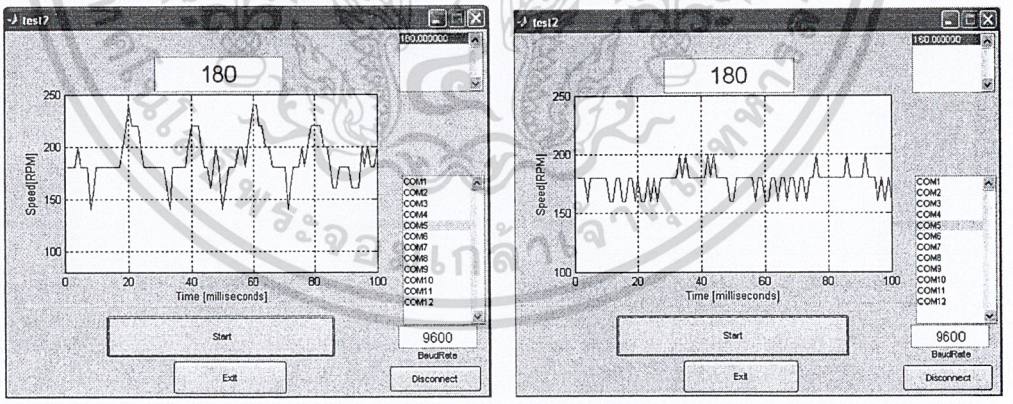
ที่ความเร็ว 180 rpm ตัวต้านทาน 2 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.8 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 2 Ω

ที่ความเร็ว 180 rpm ตัวต้านทาน 3 Ω

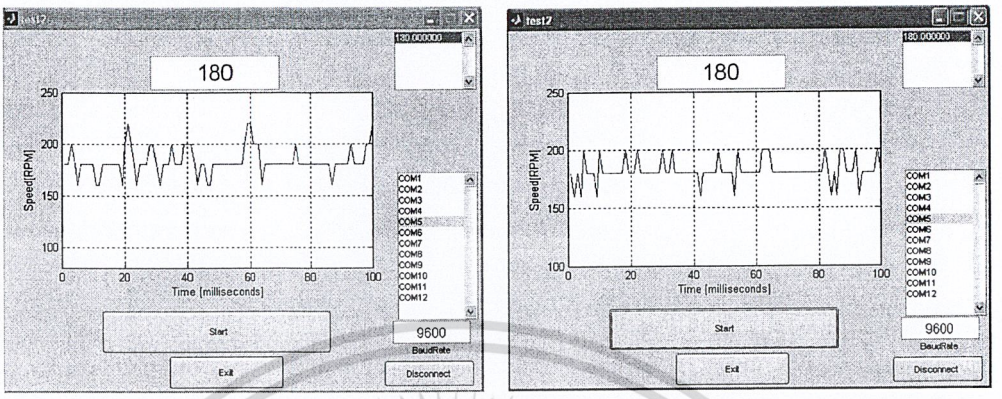


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.9 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 3 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

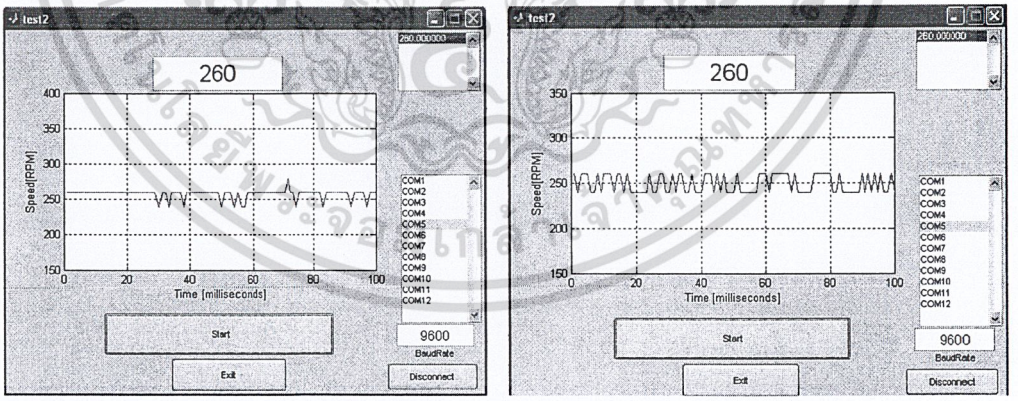
ที่ความเร็ว 180 rpm ตัวต้านทาน 5 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.10 ผลการทดลองที่ความเร็ว 180 rpm มีตัวต้านทาน 5 Ω

ที่ความเร็ว 250 rpm ตัวต้านทาน ไม่มีตัวต้านทาน

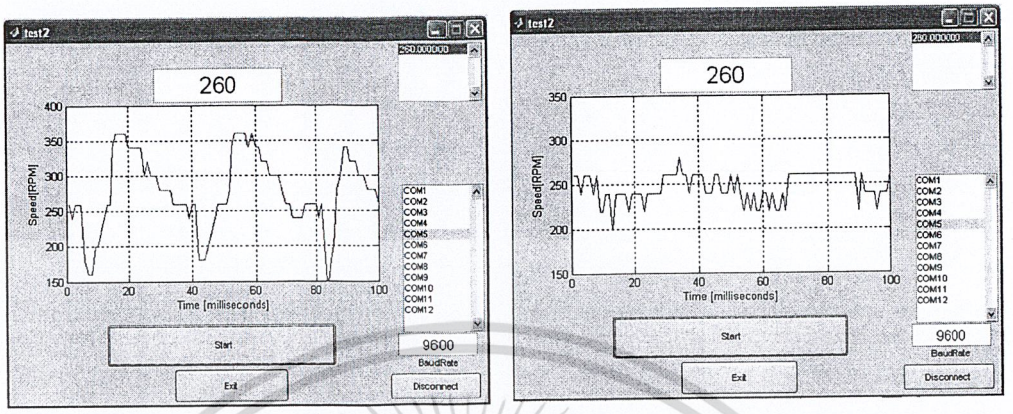


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.11 ผลการทดลองที่ความเร็ว 250 rpm ไม่มีตัวต้านทาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

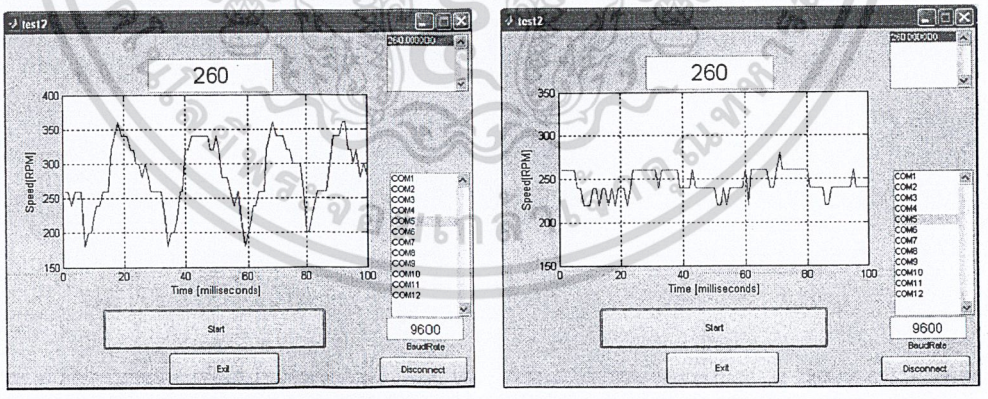
ที่ความเร็ว 250 rpm ตัวต้านทาน ตัวต้านทาน1 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.12 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 1 Ω

ที่ความเร็ว 250 rpm ตัวต้านทาน ตัวต้านทาน2 Ω

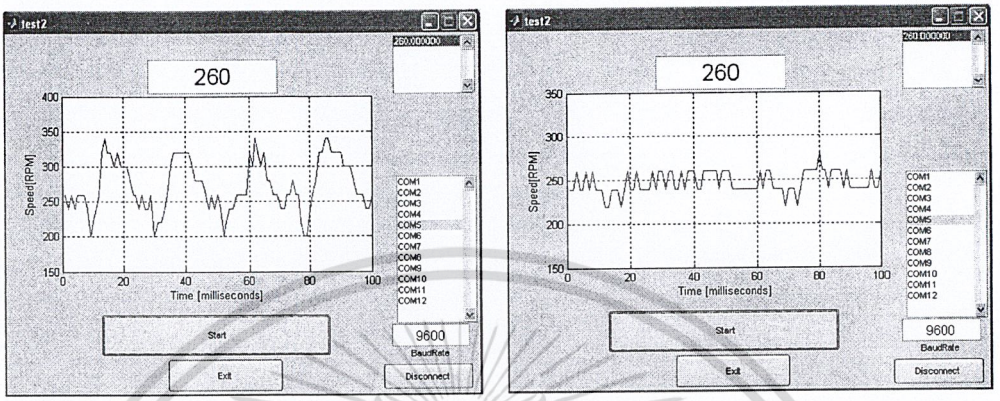


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.13 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 2 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

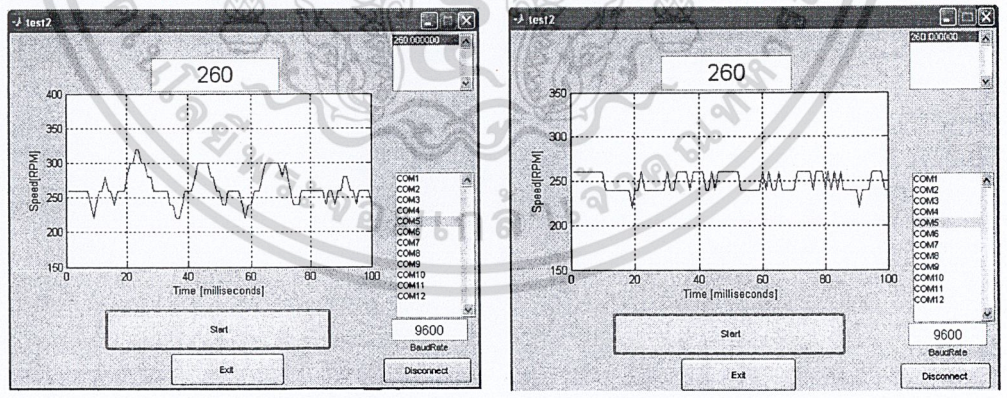
ที่ความเร็ว 250 rpm ตัวต้านทาน ตัวต้านทาน3 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.14 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 3 Ω

ที่ความเร็ว 250 rpm ตัวต้านทาน5 Ω

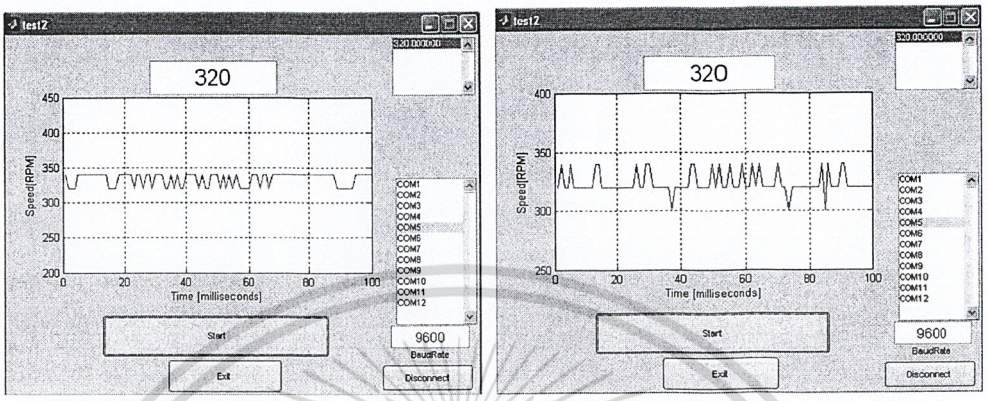


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.15 ผลการทดลองที่ความเร็ว 250 rpm มีตัวต้านทาน 5 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

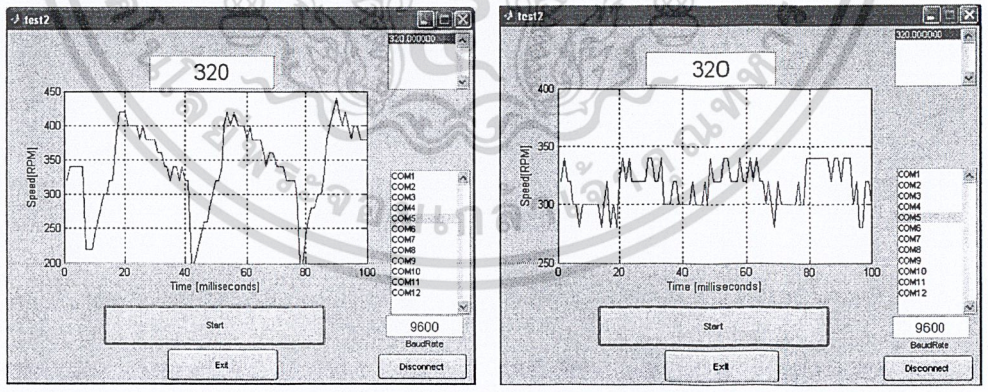
ที่ความเร็ว 320 rpm ตั ไม่มีตัวต้านทาน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.16 ผลการทดลองที่ความเร็ว 320 rpm ไม่ มีตัวต้านทาน

ที่ความเร็ว 320 rpm ตัวต้านทาน1 Ω

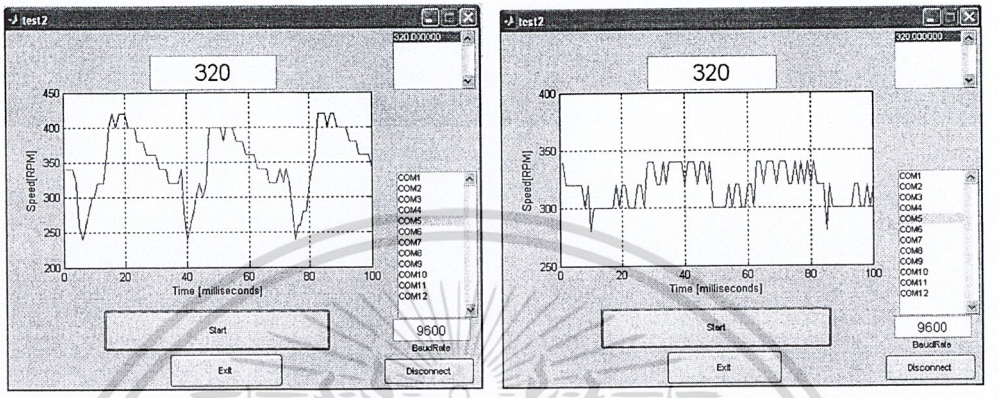


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.17 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 1 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

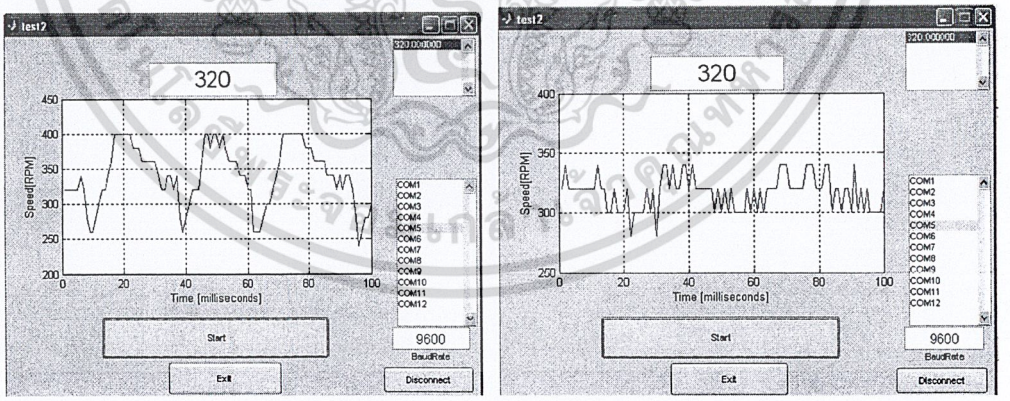
ที่ความเร็ว 320 rpm ตัวต้านทาน2 Ω



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.18 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 2 Ω

ที่ความเร็ว 320 rpm ตัวต้านทาน3 Ω

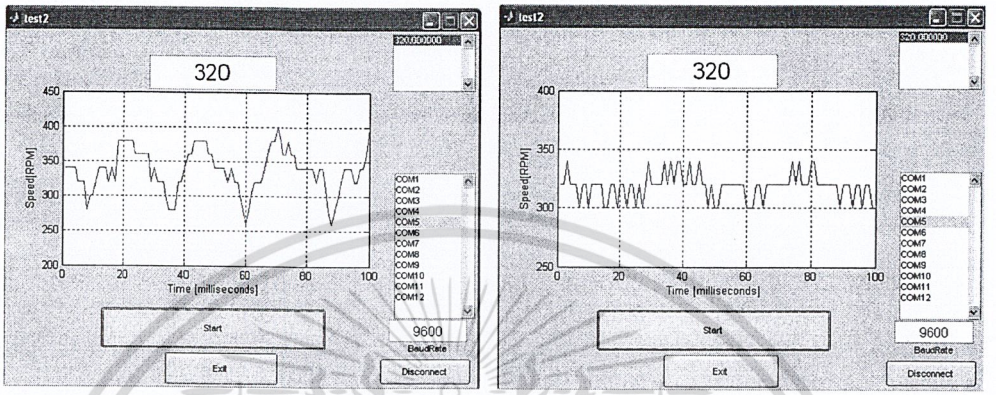


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.19 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 3 Ω

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ที่ความเร็ว 320 rpm ตัวต้านทาน5 Ω

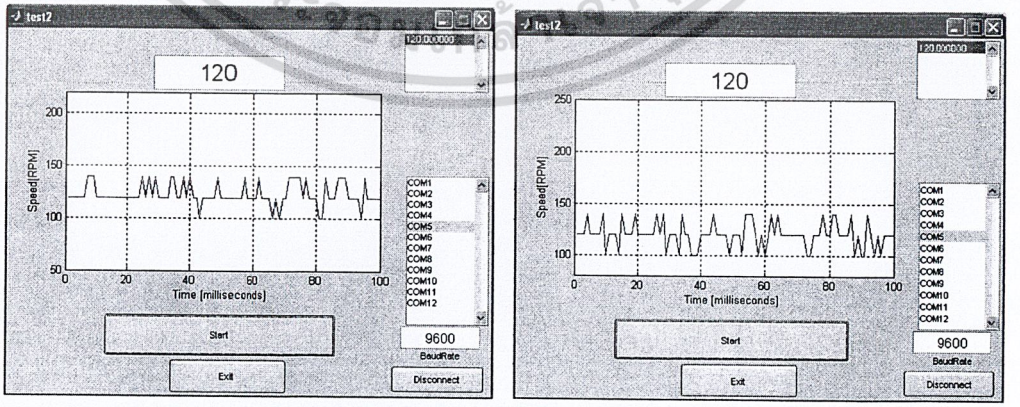


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.20 ผลการทดลองที่ความเร็ว 320 rpm มีตัวต้านทาน 5 Ω

4.3.2 ผลการทดลองภาคปฏิบัติจริงกับเครื่องเฉือนเนื้อทุเรียน

ความเร็ว 120 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

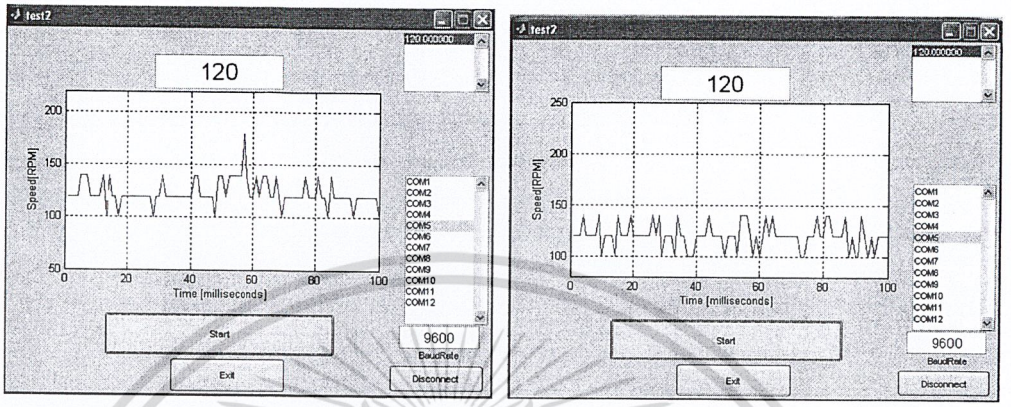


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 4.21 ผลการทดลองที่ความเร็ว 120 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

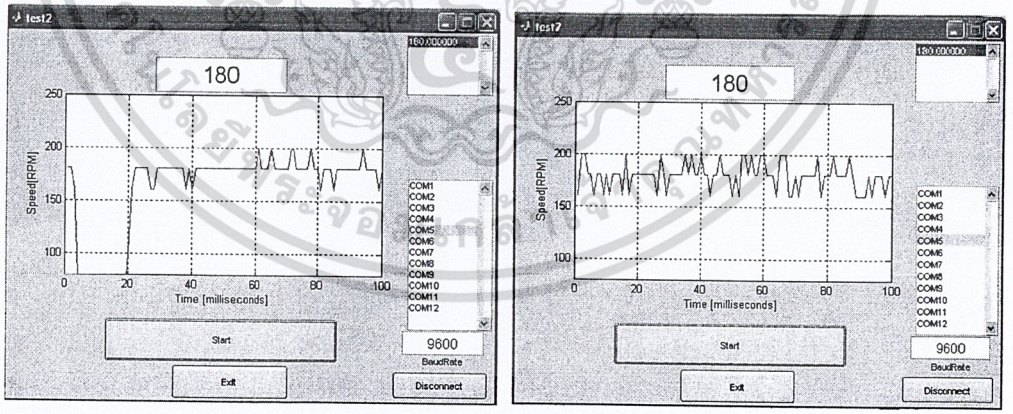
ความเร็ว 120 rpm ขณะเฉือนเนื้อทุเรียน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.22 ผลการทดลองที่ความเร็ว 120 rpm ขณะเฉือนเนื้อทุเรียน

ความเร็ว 180 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

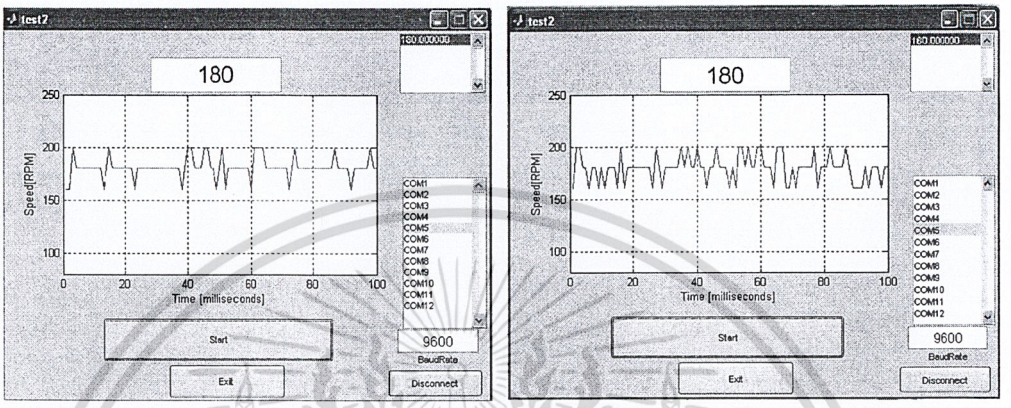


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.23 ผลการทดลองที่ความเร็ว 180rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

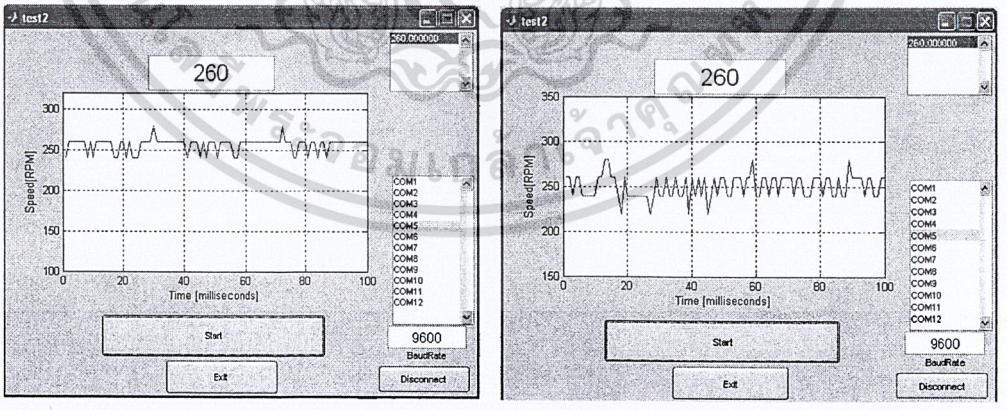
ความเร็ว 180 rpm ขณะเฉือนเนื้อทุเรียน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.24 ผลการทดลองที่ความเร็ว 180 rpm ขณะเฉือนเนื้อทุเรียน

ความเร็ว 250 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

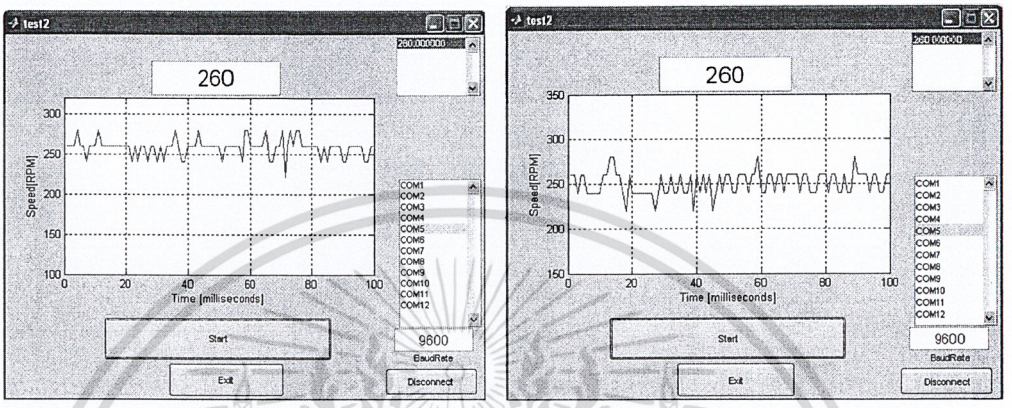


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.25 ผลการทดลองที่ความเร็ว 250 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

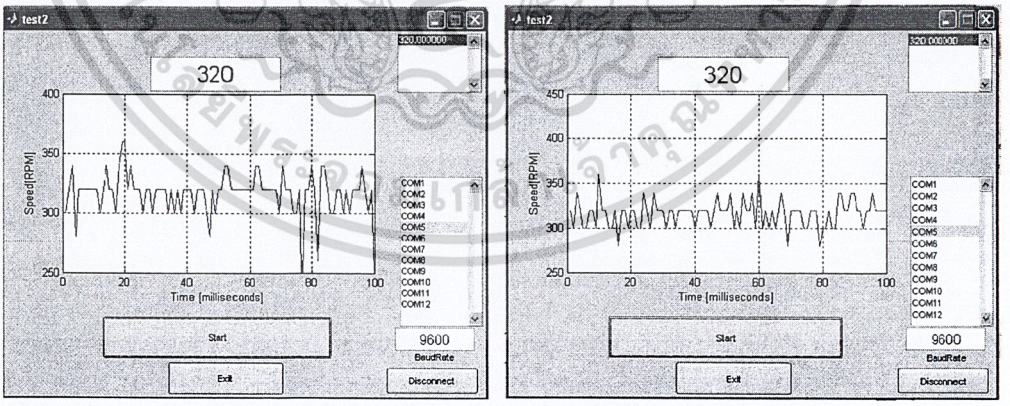
ความเร็ว 250 rpm ขณะเฉือนเนื้อทุเรียน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.26 ผลการทดลองที่ความเร็ว 250 rpm ขณะเฉือนเนื้อทุเรียน

ความเร็ว 320 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

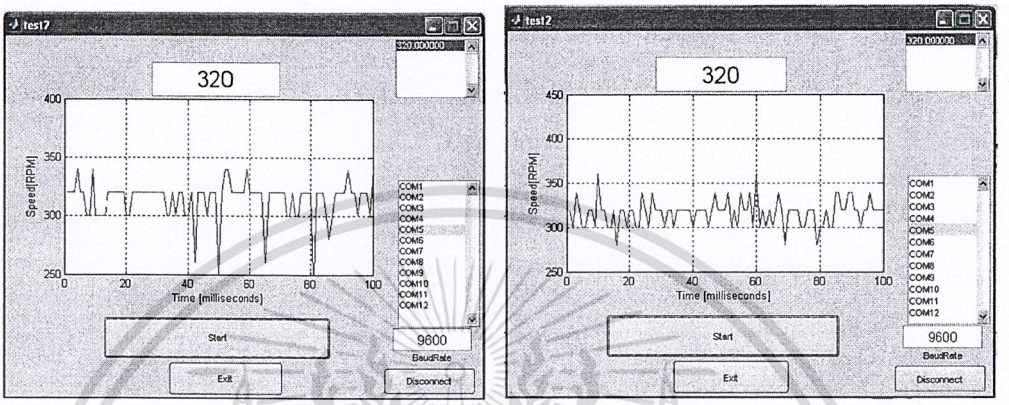


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.27 ผลการทดลองที่ความเร็ว 320 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

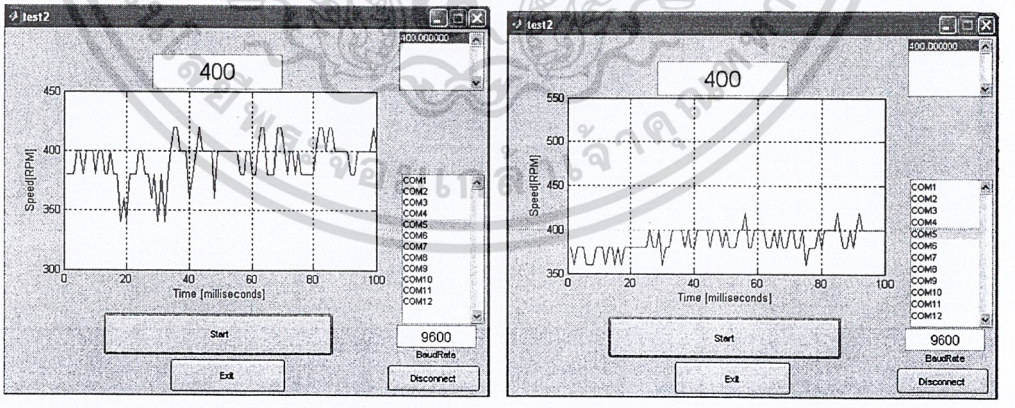
ความเร็ว 320 rpm ขณะเฉือนเนื้อทุเรียน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.28 ผลการทดลองที่ความเร็ว 320 rpm ขณะเฉือนเนื้อทุเรียน

ความเร็ว 400 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

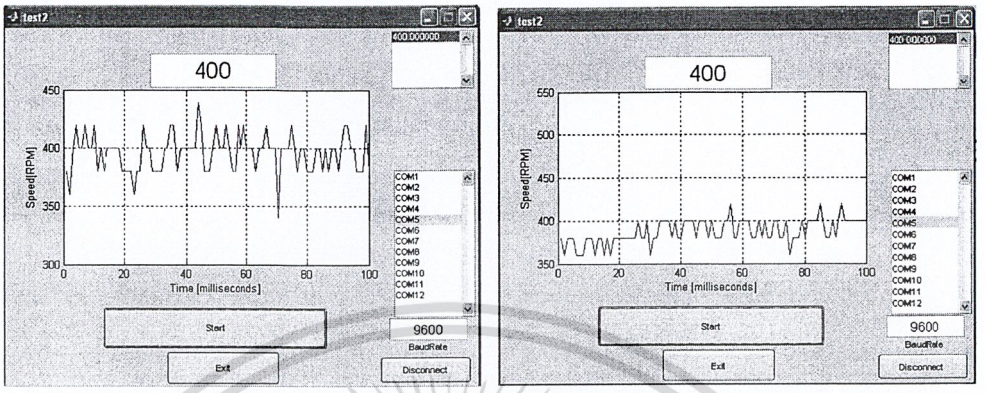


(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.29 ผลการทดลองที่ความเร็ว 400 rpm ขณะไม่ได้เฉือนเนื้อทุเรียน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ความเร็ว 400 rpm ขณะเฉือนเนื้อทุเรียน



(ก.) ระบบควบคุมแบบฟัซซี (ข.) ระบบควบคุมแบบพีไอ

รูปที่ 4.30 ผลการทดลองที่ความเร็ว 400 rpm ขณะเฉือนเนื้อทุเรียน

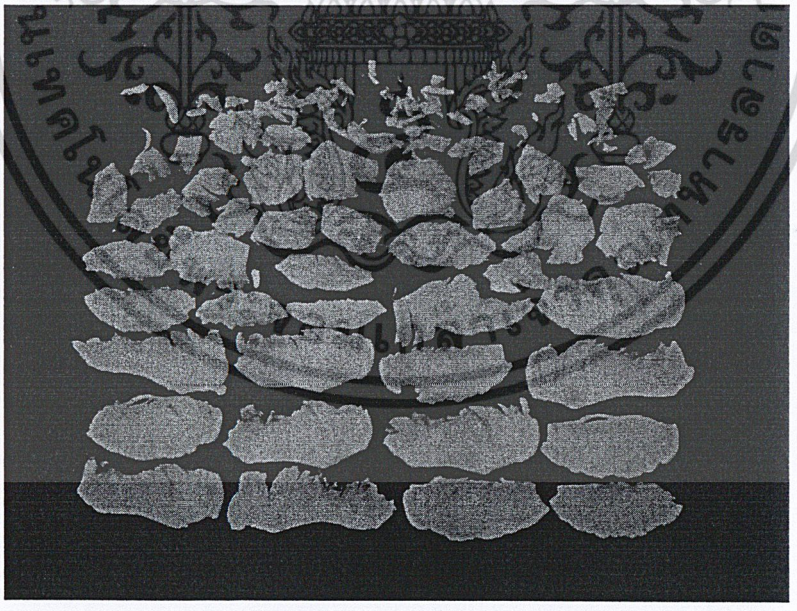
4.4 ตารางเปรียบเทียบความเสียหายของเนื้อทุเรียนโดยใช้ระบบควบคุมแบบพีไอและแบบฟัซซี

ตารางที่ 4.1 ความเสียหายที่เกิดจากการเฉือนทุเรียนโดยใช้ระบบควบคุมแบบพีไอที่ความเร็ว 250 rpm

ลำดับที่	จำนวน pixels ของชิ้นทุเรียน	พื้นที่ของชิ้นทุเรียน (cm^2)	อัตราส่วนพื้นที่เทียบกับชิ้นทุเรียนที่ใหญ่ที่สุด	จำนวนชิ้นที่เสียหายสะสม
1	124085	41.212	0.777	-
2	91115	30.262	0.571	-
3	86186	28.625	0.540	-
4	88048	29.243	0.552	-
5	52223	17.345	0.327	1
6	113884	37.824	0.713	-
7	18750	6.227	0.117	2
8	21126	7.017	0.132	3
9	84880	28.191	0.532	-
10	47019	15.616	0.295	4
11	50411	16.743	0.316	5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

12	46024	15.286	0.288	6
13	159622	53.015	1.000	-
14	38293	12.718	0.240	7
15	119264	39.611	0.747	-
16	44729	14.856	0.280	8
17	99760	33.133	0.625	-
18	102435	34.021	0.642	-
19	159078	52.834	0.997	-
20	45197	15.011	0.283	9
21	48945	16.256	0.307	10
22	119693	39.753	0.750	-
23	93543	31.068	0.586	-
24	48396	16.074	0.303	11
ความเสียหายที่เกิดจากการเฉือนทุเรียน				45.83%



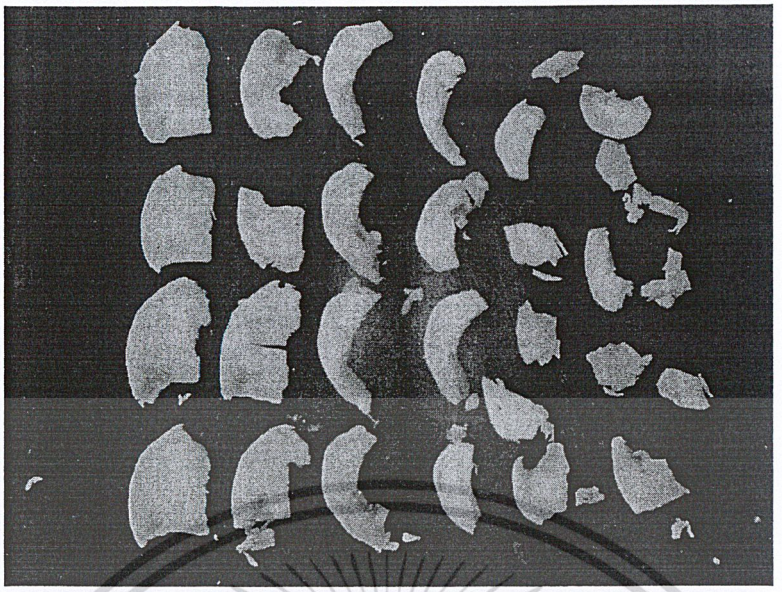
รูปที่ 4.31 เนื้อทุเรียนของระบบควบคุมแบบพีไอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 4.2 ความเสียหายที่เกิดจากการเฉือนทุเรียนโดยใช้ระบบควบคุมแบบฟัซซีที่ความเร็วรอบ 250 rpm

ลำดับที่	จำนวน pixels ของชิ้นทุเรียน	พื้นที่ของชิ้นทุเรียน (cm^2)	อัตราส่วนพื้นที่เทียบกับชิ้นทุเรียนที่ใหญ่ที่สุด	จำนวนชิ้นที่เสียหายสะสม
1	48434	24.612	0.258	1
2	32236	16.381	0.172	2
3	57207	29.070	0.305	3
4	56341	28.630	0.300	4
5	42766	21.732	0.228	5
6	187500	95.278	1.000	-
7	108660	55.216	0.580	-
8	71708	36.438	0.382	-
9	51753	26.298	0.276	6
10	78571	39.926	0.419	-
11	112112	56.970	0.598	-
12	108536	55.153	0.579	-
13	41880	21.281	0.223	7
14	56155	28.535	0.299	8
15	97734	49.664	0.521	-
16	86613	44.012	0.462	-
17	51988	26.418	0.277	9
18	61040	31.017	0.326	10
19	61226	31.112	0.327	11
20	48616	24.704	0.259	12
21	18750	9.528	0.100	13
22	86818	44.117	0.463	-
23	187434	95.245	1.000	-
24	54883	27.889	0.293	14
ความเสียหายที่เกิดจากการเฉือนทุเรียน				58.33%

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.32 เนื้อทุเรียนของระบบควบคุมแบบฟัซซี

4.5 วิเคราะห์การทดลอง

จากการทดลองจะสังเกตได้จากกราฟการทดลองจะเห็นได้ว่ากราฟของระบบควบคุมแบบฟัซซีมีแกว่งของกราฟมากกว่าระบบควบคุมแบบพีไอที่ความเร็วเท่ากัน จะเห็นได้ว่าระบบการควบคุมแบบพีไอมีความเสถียรมากกว่าระบบควบคุมแบบฟัซซี ซึ่งจะสังเกตได้จากรูปที่ 4.16, 4.17, 4.18, 4.19 และ 4.20

เมื่อนำระบบควบคุมไปติดตั้งกับเครื่องเฉือนเนื้อทุเรียนแล้วทำการทดลองเฉือนเนื้อทุเรียนจริง ทำให้รู้ว่าความเร็วของจานใบมีดที่เหมาะสมกับเครื่องไม่ควรเกิน 250 rpm จึงทำให้ได้เนื้อทุเรียนมีการแตกหักน้อยที่สุด

จากตารางที่4.1 และ 4.2 ทำให้ทราบว่าความเสียหายที่เกิดจากระบบควบคุมแบบพีไอจะมีความเสียหายน้อยกว่าระบบควบคุมแบบฟัซซี

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
สรุปผลการทดลองและข้อเสนอแนะ

5.1 สรุปผลการทดลอง

จากการทดลองเปรียบเทียบเครื่องเฉือนเนื้อทุเรียนแบบหมุนตามแนวแกนรัศมีระหว่างใช้ระบบควบคุมแบบพีไอกับระบบควบคุมแบบฟัซซี จะพบความแตกต่างของผลระหว่างระบบควบคุมแบบพีไอกับระบบควบคุมแบบฟัซซี ซึ่งสามารถเปรียบเทียบผลการทดลองได้จากลักษณะของเนื้อทุเรียนที่เป็นแผ่น และกราฟแสดงความเร็วรอบของมอเตอร์ คือ

1. ลักษณะของเนื้อทุเรียนที่ใช้การระบบควบคุมแบบพีไอ จะมีความบางและสม่ำเสมอกัน
2. ลักษณะของเนื้อทุเรียนที่ใช้การระบบควบคุมแบบฟัซซี จะมีความบางแต่ไม่สม่ำเสมอมีขาดเป็นบางแผ่น
3. จากกราฟความเร็วรอบจะเห็นได้ว่าระบบควบคุมแบบพีไอ ความเร็วจะลดลงได้ยากความเร็วคงที่ ซึ่งระบบควบคุมแบบฟัซซีจะลดลงได้ง่ายแต่จะพยายามรักษาความเร็วให้กลับมาเป็นเช่นเดิม

จึงเห็นได้ว่าระบบควบคุมแบบพีไอ จะเหมาะสมกับเครื่องเฉือนเนื้อทุเรียนมากกว่าระบบควบคุมแบบฟัซซี

5.2 ปัญหาและข้อเสนอแนะ

ปัญหา

1. เครื่องเฉือนเนื้อทุเรียนมีความสั่นสะเทือนมากทำให้การควบคุมมีความคลาดเคลื่อน
2. การศึกษาฟัซซีเนื่องจากไม่เคยศึกษามาก่อนจึงทำให้เกิดความล่าช้าในการทำงาน
3. มอเตอร์มีการหมุนที่ไม่สม่ำเสมอทำให้ผลที่วัดจากเอ็นโค้ดเดอร์คลาดเคลื่อน

ข้อเสนอแนะ

1. การควบคุมมอเตอร์มีหลายแบบควรเลือกการควบคุมให้เหมาะสมกับงานที่ปฏิบัติ
2. ควรสร้างโครงสร้างหรือมีความมั่นคงสูงเพื่อลดผลกระทบต่อระบบควบคุม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารอ้างอิง

[1] ประจิน พลังสันติกุล. **การเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ dsPIC30F ด้วยคอมไพเลอร์ MPLAB C**.

กรุงเทพมหานคร : บริษัท แอพซอฟต์เทค จำกัด. 2551.

[2] **"ตรรกศาสตร์คลุมเครือ."** [Online]. Available :

th.wikipedia.org/wiki/ตรรกศาสตร์คลุมเครือ. 2011.

[3] **"การควบคุมมอเตอร์ไฟฟ้ากระแสตรง."** [Online]. Available :

202.129.59.73/tn/motor10-52/motor3.htm. 2011.

[4] "ระบบกฎแบบฟัซซี." [Online]. Available :

202.44.34.134/teacher/FileDL/phayung206255323264.pdf.2011.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก

โปรแกรมควบคุมแบบพีไอ

```
//#include "p30f6014.h"
#include <p30f4011.h>
#include <stdio.h>
#include <math.h>
#include<uart.h>
//#include "adc10.h"
#include "lcd4011_8.h"
#include <spi.h>                    //  Module function for SPI module
#include <adc10.h>
#include<ports.h>
#include <adc10.h>
#define OUT_A 0                     // Macro for select Output A
#define OUT_B 1                     // Macro for select Output B
//#define FCY  29491200             // xtal = 7...Mhz; PLLx16 -> 20 MIPS
//#define FCY  7372800  // xtal = 7...Mhz; PLLx16 -> 20 MIPS


#define T1_CONFIG 0x8030
#define T2_CONFIG 0x8030


_FOSC(CSW_FSCM_OFF & XT_PLL16);
        // Disable Clock Switching,Enable Fail-Salf Clock
                                    // Clock Source = Primary XT + (PLL x 16)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
//_FOSC(CSW_FSCM_OFF & XT_PLL4);

_FWDT(WDT_OFF);
                        // Disable Watchdog

_FBORPOR(PBOR_OFF & PWRT_64 & MCLR_EN);                                    //
Disable Brown-Out ,Power ON = 64mS,Enable MCLR

_FGS(CODE_PROT_OFF);

signed int sv=0,its=0,adc_buff;

long ic=0,iii;

float Out=0,pv,pp,pi1=0,er,er1=0,erold=0,p,err;


unsigned int analog(unsigned char ch)

{
        unsigned int val;

        ADPCFG &= (!(ch+1));    // all PORTB = Digital; RB2 = analog

        ADCON1 = 0x0000;        // SAMP bit = 0 ends sampling ...

                                // and starts converting


        ADCHS = ch;             // Connect RB2/AN2 as CH0 input ..

                                // in this example RB2/AN2 is the input

        ADCSSL = 0;

        ADCON3 = 0x0002;        // Manual Sample, Tad = internal 2 Tcy

        ADCON2 = 0;

        ADCON1bits.ADON = 1; // turn ADC ON


        ADCON1bits.SAMP = 1; // start sampling ...

        delay_ms(40);                   // for 100 mS

        ADCON1bits.SAMP = 0;  // start Converting

        while (!ADCON1bits.DONE); // conversion done?
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น รวมถึงการนำไปเผยแพร่ต่อ ต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        val = ADCBUF0;              // yes then get ADC value
        return(val);
}

void _ISR _SPI1Interrupt(void)
{
        IFS0bits.SPI1IF = 0;                // Clear SPI1 interrupt flag
}

void spi_enable()
{
LATBbits.LATB6 = 0;       // Slave select control enable
}
//-------------------------------------------------------------------------------------------------
--------------//
//------------------------------ Function slave select Enable ------------------------------------
------//
//-------------------------------------------------------------------------------------------------
--------------//
void spi_disable()
{
LATBbits.LATB6 = 1;       // Slave select control disable
}
//-------------------------------------------------------------------------------------------------
--------------//
//------------------------------ Function Latch Output MCP4922 -----------------------------------
-----------//
//-------------------------------------------------------------------------------------------------
--------------//
void latch()
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
{
        int xx=0;
        LATBbits.LATB5 = 0;     // Latch Low
        for(xx=0;xx<10;xx++);   // Delay
        LATBbits.LATB5 = 1;     // Latch High
}
//-------------------------------------------------------------------------------------------------------------
--------------//
//------------------------------- Function write data to MPCP4922 ----------------------------------
-------------//
//-------------------------------------------------------------------------------------------------------------
--------------//
void MCP4922_write(unsigned int dat)
{
        spi_enable();                   // Start Frame data transfer
        WriteSPI1(dat>>8);              // Send data and MSB byte
        while(SPI1STATbits.SPITBF);     // Wait until data success
        WriteSPI1(dat & 0xO0FF);        // Send data LSB byte
        while(SPI1STATbits.SPITBF);     // Wait until data success
        spi_disable();                  // Stop Frame data transfer
        latch();                        // Latch Output
}
//-------------------------------------------------------------------------------------------------------------
--------------//
//--------------------------------- Function slave select Enable ---------------------------------------
------//
//-------------------------------------------------------------------------------------------------------------
--------------//
void dac_value(char channel,unsigned int value)
{
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        value &= 0x0FFF;          // Filter 12 bit data(0-4095 only)
        value |= 0x7000;          // Set config command as BUF=1,GA=1,SHDN=1
        //----------------- Set Output channel ----------------------//
        if(channel==OUT_A)
        {
                value &= 0x7FFF;          // Select output channel A
        }
        if(channel==OUT_B)
        {
                value |= 0x8000;          // Select output channel B
        }
        //-----------------------------------------------------//
        MCP4922_write(value); // Write data to MPCP4922
}
//-------------------------------------------------------------------------------------------
--------------//
//------------------------------ Function for configuration SPI1 ------------------------------
------//
//-------------------------------------------------------------------------------------------
--------------//
void spi_init()
{
        unsigned int SPICONValue;                 // Buffer for configuration for SPI1CON
        unsigned int SPISTATValue;                // Buffer for configuration for SPI1STAT


// TRISFbits.TRISF2=1;   // SDI1
// TRISFbits.TRISF3=0;   // SDO1
 //TRISFbits.TRISF6=0;  // SCK1
 //TRISBbits.TRISB2=0;  // SS1
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
CloseSPI1();            // Disable SPI1 module
spi_disable();          // Stop Frame data transfer
LATBbits.LATB5 = 1;     // Initial Latch High

SPICONValue  =  FRAME_ENABLE_OFF &          // Framed SPI support
Disable
                FRAME_SYNC_OUTPUT &                 //
Frame Sync Pulse direction control
                ENABLE_SDO_PIN &                    // Enable
SDO pin
                SPI_MODE16_OFF &                    // Disable
SPI 16 bit data mode(use 8 bit data mode)
                SPI_SMP_ON &                        // Enable
Sample phase
                SPI_CKE_ON &                        // Idle state
clock high
                SLAVE_ENABLE_OFF &                  // Slave
select(pin SS1) not use
                CLK_POL_ACTIVE_HIGH &               // Clock
polarity at high
                MASTER_ENABLE_ON &                  // SPI
master mode
                SEC_PRESCAL_1_1 &                   //
Secondary Prescale 1:1
                PRI_PRESCAL_1_1;                    // Primary
Prescale 1:1

SPISTATValue = SPI_ENABLE &                 // Enable SPI module
                SPI_IDLE_CON &                      //
SPI working on Idle mode
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                                        SPI_RX_OVFLOW_CLR;                  //
Automatic clear receive overflow flag


        OpenSPI1(SPICONValue,SPISTATValue );        // Execute configuration for SPI1

        EnableIntSPI1;                  // Enable SPI1 interrupt


}


void _ISR _INT0Interrupt(void)

{

IFS0bits.INT0IF = 0.0;

   //LATBbits.LATB0 = !LATBbits.LATB0;  // Toggle LED at RB0 pin

  // k=k-1.0;

ic++;

}


void _ISR __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)

{


//      dac_value(OUT_B,1000);   // I   // Write DAC value to MCP4922 output B


        iii=(ic*60*1000);

    pv=iii/3000;

    ic=0;


its++;

    if(its>=50)

    {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            er=(sv-pv);
           p=er;
        pp=(((er + er1)*0.005) + pi1);
        Out=(p*20)+(pp*8);
                er1=er;
                pi1=pp;
//Out=4000;
        if (Out>4095)
                {
                Out=4095;
                }
                if(Out<0)
                {
                Out=0;
                }
        its=0;
        dac_value(OUT_A,Out);
   }

LATFbits.LATF4=~LATFbits.LATF4;
        _T2IF = 0;
}

/* ===== Timer2 Initial Function ===== */
void timer2_init(void)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

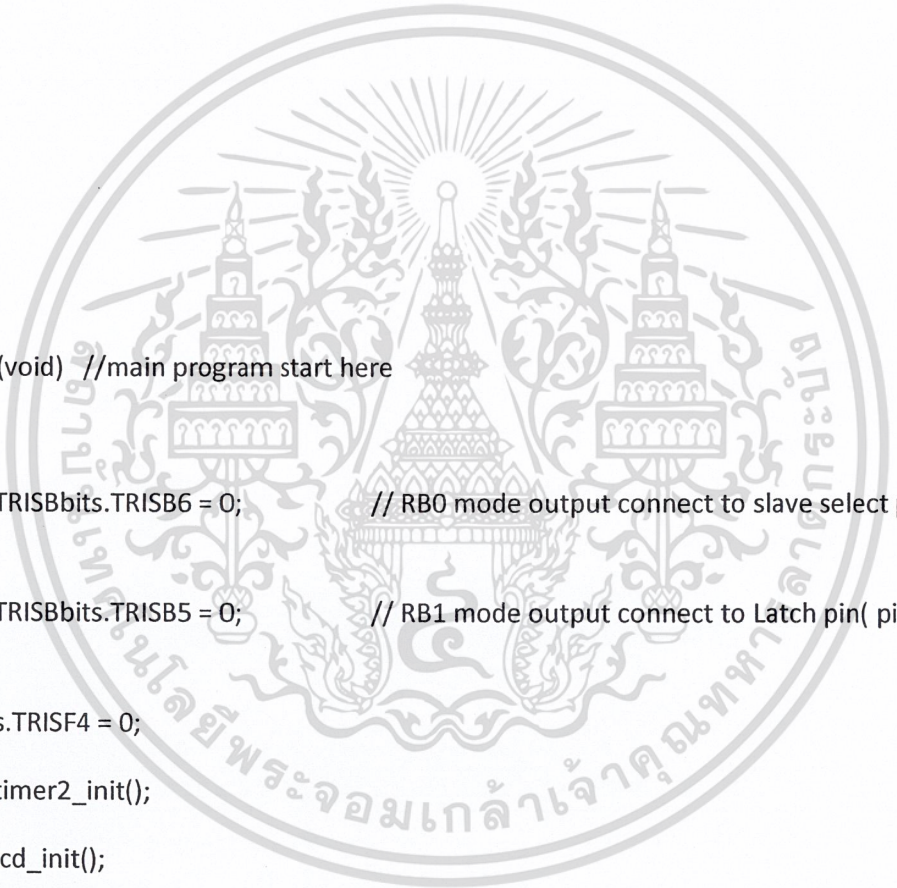
```
{
        _T2IE = 0;                          //Disable Timer2
        T2CON = T2_CONFIG;          //Timer1 config
        PR2 = 100;                          //Set period time 1ms
        _T2IP = 1;
        _T2IF = 0;                          //Clear Interrupt flag
        _T2IE = 1;                          //Enable Timer1
}
/* END Function */



void uart1_init()
{
  unsigned int baudvalue;
  unsigned int U1MODEvalue;
  unsigned int U1STAvalue;

  CloseUART1();
  ConfigIntUART1(UART_RX_INT_EN &
        UART_RX_INT_PR3 &
        UART_TX_INT_DIS); //&
        //UART_TX_INT_PR2 );


  baudvalue = 192;// 48;//25;     // Baudrate 9600 @PLL4
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
  U1MODEvalue = UART_EN &
        UART_IDLE_CON &
        UART_ALTRX_ALTTX &
        UART_DIS_WAKE &
        UART_DIS_LOOPBACK &
        UART_DIS_ABAUD &
        UART_NO_PAR_8BIT &
        UART_1STOPBIT ;

  U1STAvalue =  UART_INT_TX_BUF_EMPTY &
        UART_TX_PIN_NORMAL &
        UART_TX_ENABLE &
        UART_INT_RX_3_4_FUL &
        UART_ADR_DETECT_DIS &
        UART_RX_OVERRUN_CLEAR ;

  OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
}

//------------------Function Send UART----------------//
void uart1_puts(char string[])
{

  putsUART1 ((unsigned int *)string);
  while (BusyUART1());
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
void _ISR _U1TXInterrupt(void)
{
  IFS0bits.U1TXIF = 0;  // Clear TX interrupt flag
}
void _ISR _U1RXInterrupt(void)
{
  IFS0bits.U1RXIF = 0;
}
```



```
int main(void)  //main program start here
{
        TRISBbits.TRISB6 = 0;           // RB0 mode output connect to slave select pin(pin
CS )
        TRISBbits.TRISB5 = 0;           // RB1 mode output connect to Latch pin( pin
LATCH)
TRISFbits.TRISF4 = 0;
        timer2_init();
        lcd_init();
        spi_init();
uart1_init();
ConfigINT0(RISING_EDGE_INT & EXT_INT_ENABLE & EXT_INT_PRI_7);
lcd_clear();
        while(1)
        {
 adc_buff = analog(0);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น ห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
if (adc_buff>511)
{
 adc_buff=511;
}
sv=0.78*adc_buff;

//        lcd_puts(0x80," Speed Control");
//printf("test\n\r");
printf("%f\n\r",pv);
//printf("%d\n\r",a);
lcd_puts(0x80,"SV=    RPM");
inttolcd(0x83,sv);
//lcd_puts(0x89,"RPM");
//lcd_puts(0xC0,"    ");
lcd_puts(0xC0,"PV=    RPM");
inttolcd(0xC3,pv);
//lcd_puts(0xC9,"RPM");
//inttolcd(0x80,Out);
//delay_ms(1);
        }
        return 0;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ข

โปรแกรมควบคุมแบบฟัซซี

```
//#include "p30f6014.h"
#include <p30f4011.h>
#include <stdio.h>
#include <math.h>
#include<uart.h>
//#include "adc10.h"
#include "lcd4011_8.h"
#include <spi.h>                    //  Module function for SPI module
#include <adc10.h>
#include<ports.h>
#include <adc10.h>
#define OUT_A 0                     // Macro for select Output A
#define OUT_B 1                     // Macro for select Output B
//#define FCY  29491200             // xtal = 7...Mhz; PLLx16 -> 20 MIPS
//#define FCY  7372800  // xtal = 7...Mhz; PLLx16 -> 20 MIPS

#define T1_CONFIG 0x8030
#define T2_CONFIG 0x8030

_FOSC(CSW_FSCM_OFF & XT_PLL16);
                    // Disable Clock Switching,Enable Fail-Salf Clock
                                        // Clock Source = Primary XT + (PLL x 16)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
//_FOSC(CSW_FSCM_OFF & XT_PLL4);

_FWDT(WDT_OFF);
                        // Disable Watchdog

_FBORPOR(PBOR_OFF & PWRT_64 & MCLR_EN);                                    //
Disable Brown-Out ,Power ON = 64mS,Enable MCLR

_FGS(CODE_PROT_OFF);

signed int sv=0,its=0,adc_buff;

long ic=0,iii;

float Out=0,pv,pp,pi1=0,er,er1=0,erold=0,p,err;


unsigned int analog(unsigned char ch)

{
        unsigned int val;

        ADPCFG &= (!(ch+1));    // all PORTB = Digital; RB2 = analog

        ADCON1 = 0x0000;        // SAMP bit = 0 ends sampling ...
                                        // and starts converting


        ADCHS = ch;             // Connect RB2/AN2 as CH0 input ..
                                        // in this example RB2/AN2 is the input

        ADCSSL = 0;

        ADCON3 = 0x0002;        // Manual Sample, Tad = internal 2 Tcy

        ADCON2 = 0;

        ADCON1bits.ADON = 1; // turn ADC ON


        ADCON1bits.SAMP = 1; // start sampling ...

        delay_ms(40);                   // for 100 mS

        ADCON1bits.SAMP = 0;  // start Converting

        while (!ADCON1bits.DONE); // conversion done?
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        val = ADCBUF0;          // yes then get ADC value
        return(val);
}

void _ISR _SPI1Interrupt(void)
{
        IFS0bits.SPI1IF = 0;            // Clear SPI1 interrupt flag
}

void spi_enable()
{
LATBbits.LATB6 = 0;     // Slave select control enable
}
//-------------------------------------------------------------------------------------------------
-------------//
//------------------------------ Function slave select Enable ------------------------------------
------//
//-------------------------------------------------------------------------------------------------
-------------//
void spi_disable()
{
LATBbits.LATB6 = 1;     // Slave select control disable
}
//-------------------------------------------------------------------------------------------------
-------------//
//-------------------------------- Function Latch Output MCP4922 ----------------------------------
-----------//
//-------------------------------------------------------------------------------------------------
-----------//
void latch()
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
{
        int xx=0;
        LATBbits.LATB5 = 0;     // Latch Low
        for(xx=0;xx<10;xx++);   // Delay
        LATBbits.LATB5 = 1;     // Latch High
}
//------------------------------------------------------------------------------------------------------------
--------------//
//---------------------------------- Function write data to MPCP4922 ----------------------------------
-------------//
//------------------------------------------------------------------------------------------------------------
--------------//
void MCP4922_write(unsigned int dat)
{
        spi_enable();                   // Start Frame data transfer
        WriteSPI1(dat>>8);              // Send data and MSB byte
        while(SPI1STATbits.SPITBF);     // Wait until data success
        WriteSPI1(dat & 0xO0FF);        // Send data LSB byte
        while(SPI1STATbits.SPITBF);     // Wait until data success
        spi_disable();                  // Stop Frame data transfer
        latch();                        // Latch Output
}
//------------------------------------------------------------------------------------------------------------
--------------//
//---------------------------------- Function slave select Enable ----------------------------------------
------//
//------------------------------------------------------------------------------------------------------------
--------------//
void dac_value(char channel,unsigned int value)
{
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        value &= 0x0FFF;          // Filter 12 bit data(0-4095 only)
        value |= 0x7000;          // Set config command as BUF=1,GA=1,SHDN=1
        //----------------- Set Output channel ----------------------//
        if(channel==OUT_A)
        {
                value &= 0x7FFF;          // Select output channel A
        }
        if(channel==OUT_B)
        {
                value |= 0x8000;          // Select output channel B
        }
        //-----------------------------------------------------//
        MCP4922_write(value); // Write data to MPCP4922
}
//-------------------------------------------------------------------------------------------
-------------//
//------------------------------ Function for configuration SPI1 ------------------------------
------//
//-------------------------------------------------------------------------------------------
-------------//
void spi_init()
{
        unsigned int SPICONValue;                 // Buffer for configuration for SPI1CON
        unsigned int SPISTATValue;                // Buffer for configuration for SPI1STAT


// TRISFbits.TRISF2=1;   // SDI1
// TRISFbits.TRISF3=0;   // SDO1
 //TRISFbits.TRISF6=0;  // SCK1
 //TRISBbits.TRISB2=0;  // SS1
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        CloseSPI1();            // Disable SPI1 module
        spi_disable();          // Stop Frame data transfer
        LATBbits.LATB5 = 1;     // Initial Latch High

        SPICONValue   =         FRAME_ENABLE_OFF &          // Framed SPI support
Disable
                                FRAME_SYNC_OUTPUT &                     //
Frame Sync Pulse direction control
                                ENABLE_SDO_PIN &                // Enable
SDO pin
                                SPI_MODE16_OFF &                // Disable
SPI 16 bit data mode(use 8 bit data mode)
                                SPI_SMP_ON &                    // Enable
Sample phase
                                SPI_CKE_ON &                    // Idle state
clock high
                                SLAVE_ENABLE_OFF &              // Slave
select(pin SS1) not use
                                CLK_POL_ACTIVE_HIGH &           // Clock
polarity at high
                                MASTER_ENABLE_ON &              // SPI
master mode
                                SEC_PRESCAL_1_1 &               //
Secondary Prescale 1:1
                                PRI_PRESCAL_1_1;                // Primary
Prescale 1:1

        SPISTATValue = SPI_ENABLE &                     // Enable SPI module
                                SPI_IDLE_CON &                  //
SPI working on Idle mode
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                                        SPI_RX_OVFLOW_CLR;                  //
Automatic clear receive overflow flag


        OpenSPI1(SPICONValue,SPISTATValue );          // Execute configuration for SPI1

        EnableIntSPI1;                  // Enable SPI1 interrupt


}


void _ISR _INT0Interrupt(void)

{

IFS0bits.INT0IF = 0.0;

   //LATBbits.LATB0 = !LATBbits.LATB0;  // Toggle LED at RB0 pin

  // k=k-1.0;

ic++;

}


//void _ISR __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)

{


//        dac_value(OUT_B,1000);   // I   // Write DAC value to MCP4922 output B


        iii=(ic*60*1000);

    pv=iii/3000;

    ic=0;


its++;

    if(its>=50)

    {
```

เอกสารนี้สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            er=(sv-pv);
                err=(er-erold)/0.01;
          // p=er;
         //pp=(((er + er1)*0.005) + pi1);
         //Out=(p*20)+(pp*8);
         //        er1=er;
          //        pi1=pp;
//Out=4000;

erold=er;



if((er>0)&&(err>=0))
{
Out=Out+10;
}
else if((er>0)&&(err<0))
{
Out=Out+10;
}
else if((er<0)&&(err>0))
{
Out=Out-10;
}
else if((er<0)&&(err<0))
{
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
Out=Out-10;
}

        if (Out>4095)
                {
                    Out=4095;
                    }
                if(Out<0)
                {
                    Out=0;
                    }
        its=0;
        dac_value(OUT_A,Out);
   }

LATFbits.LATF4=~LATFbits.LATF4;
        _T2IF = 0;
}
void _ISR __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)
{

//      dac_value(OUT_B,1000);   // I   // Write DAC value to MCP4922 output B
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        iii=(ic*60*1000);
    pv=iii/3000;
    ic=0;


its++;
    if(its>=50)
    {
            er=(sv-pv);
          p=er;
        pp=(((er + er1)*0.005) + pi1);
        Out=(p*20)+(pp*8);
                er1=er;
                pi1=pp;
//Out=4000;
        if (Out>4095)
                {
                    Out=4095;
                    }

                if(Out<0)
                {
                        Out=0;
                        }
        its=0;
        dac_value(OUT_A,Out);
  }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
LATFbits.LATF4=~LATFbits.LATF4;
        _T2IF = 0;
}

/* ===== Timer2 Initial Function ===== */
void timer2_init(void)
{
        _T2IE = 0;                          //Disable Timer2
        T2CON = T2_CONFIG;          //Timer1 config
        PR2 = 100;                          //Set period time 1ms
        _T2IP = 1;
        _T2IF = 0;                          //Clear Interrupt flag
        _T2IE = 1;                          //Enable Timer1
}
/* END Function */

void uart1_init()
{
  unsigned int baudvalue;
  unsigned int U1MODEvalue;
  unsigned int U1STAvalue;

  CloseUART1();
  ConfigIntUART1(UART_RX_INT_EN &
        UART_RX_INT_PR3 &
        UART_TX_INT_DIS); //&
        //UART_TX_INT_PR2 );
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
  baudvalue = 192;// 48;//25;     // Baudrate 9600 @PLL4


  U1MODEvalue = UART_EN &
        UART_IDLE_CON &
        UART_ALTRX_ALTTX &
        UART_DIS_WAKE &
         UART_DIS_LOOPBACK &
        UART_DIS_ABAUD &
        UART_NO_PAR_8BIT &
        UART_1STOPBIT ;


  U1STAvalue =  UART_INT_TX_BUF_EMPTY &
        UART_TX_PIN_NORMAL &
        UART_TX_ENABLE &
        UART_INT_RX_3_4_FUL &
        UART_ADR_DETECT_DIS &
        UART_RX_OVERRUN_CLEAR ;


  OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
}



//------------------Function Send UART----------------//

void uart1_puts(char string[])
{
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
  putsUART1 ((unsigned int *)string);
  while (BusyUART1());
}
void _ISR _U1TXInterrupt(void)
{
  IFS0bits.U1TXIF = 0;  // Clear TX interrupt flag
}
void _ISR _U1RXInterrupt(void)
{
  IFS0bits.U1RXIF = 0;
}



int main(void)  //main program start here
{
        TRISBbits.TRISB6 = 0;           // RB0 mode output connect to slave select pin(pin
CS )
        TRISBbits.TRISB5 = 0;           // RB1 mode output connect to Latch pin( pin
LATCH)
TRISFbits.TRISF4 = 0;
        timer2_init();
        lcd_init();
        spi_init();
uart1_init();
ConfigINT0(RISING_EDGE_INT & EXT_INT_ENABLE & EXT_INT_PRI_7);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเอกสารทุกครั้งที่มีการนำไปใช้

```
lcd_clear();
        while(1)
        {
adc_buff = analog(0);
if (adc_buff>511)
{
 adc_buff=511;
}
sv=0.78*adc_buff;

//        lcd_puts(0x80," Speed Control");
//printf("test\n\r");
printf("%f\n\r",pv);
//printf("%d\n\r",a);
lcd_puts(0x80,"SV=    RPM");
inttolcd(0x83,sv);
//lcd_puts(0x89,"RPM");
//lcd_puts(0xC0,"    ");
lcd_puts(0xC0,"PV=    RPM");
inttolcd(0xC3,pv);
//lcd_puts(0xC9,"RPM");
//inttolcd(0x80,Out);
//delay_ms(1);
        }
        return 0;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ค

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

dsPIC30F4011/4012
Data Sheet

High Performance
Digital Signal Controllers

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Migratable Memory, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2005, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper. 11/12/04

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MICROCHIP

dsPIC30F4011/4012

dsPIC30F4011/4012 Enhanced Flash 16-bit Digital Signal Controller

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

High Performance Modified RISC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set architecture with flexible addressing modes
- 84 base instructions
- 24-bit wide instructions, 16-bit wide data path
- 48 Kbytes on-chip Flash program space (16K Instruction words)
- 2 Kbytes of on-chip data RAM
- 1 Kbytes of non-volatile data EEPROM
- Up to 30 MIPs operation:
  - DC to 40 MHz external clock input
  - 4 MHz-10 MHz oscillator input with PLL active (4x, 8x, 16x)
- 30 interrupt sources
  - 3 external interrupt sources
  - 8 user selectable priority levels for each interrupt source
  - 4 processor trap sources
- 16 x 16-bit working register array

DSP Engine Features:

- Dual data fetch
- Accumulator write back for DSP operations
- Modulo and Bit-Reversed Addressing modes
- Two, 40-bit wide accumulators with optional saturation logic
- 17-bit x 17-bit single cycle hardware fractional/ integer multiplier
- All DSP instructions single cycle
- ± 16-bit single cycle shift

Peripheral Features:

- High current sink/source I/O pins: 25 mA/25 mA
- Timer module with programmable prescaler:
  - Five 16-bit timers/counters; optionally pair 16-bit timers into 32-bit timer modules
- 16-bit Capture input functions
- 16-bit Compare/PWM output functions
- 3-wire SPI™ modules (supports 4 Frame modes)
- I^2C™ module supports Multi-Master/Slave mode and 7-bit/10-bit addressing
- 2 UART modules with FIFO Buffers
- 1 CAN modules, 2.0B compliant

Motor Control PWM Module Features:

- 6 PWM output channels
  - Complementary or Independent Output modes
  - Edge and Center Aligned modes
- 3 duty cycle generators
- Dedicated time base
- Programmable output polarity
- Dead-time control for Complementary mode
- Manual output control
- Trigger for A/D conversions

Quadrature Encoder Interface Module Features:

- Phase A, Phase B and Index Pulse input
- 16-bit up/down position counter
- Count direction status
- Position Measurement (x2 and x4) mode
- Programmable digital noise filters on inputs
- Alternate 16-bit Timer/Counter mode
- Interrupt on position counter rollover/underflow

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Analog Features:

- 10-bit Analog-to-Digital Converter (A/D) with 4 S/H Inputs:
  - 500 Ksps conversion rate
  - 9 input channels
  - Conversion available during Sleep and Idle
- Programmable Brown-out Detection and Reset generation

Special Microcontroller Features:

- Enhanced Flash program memory:
  - 10,000 erase/write cycle (min.) for industrial temperature range, 100K (typical)
- Data EEPROM memory:
  - 100,000 erase/write cycle (min.) for industrial temperature range, 1M (typical)
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Flexible Watchdog Timer (WDT) with on-chip low power RC oscillator for reliable operation
- Fail-Safe clock monitor operation detects clock failure and switches to on-chip low power RC oscillator
- Programmable code protection
- In-Circuit Serial Programming™ (ICSP™)
- Selectable Power Management modes
  - Sleep, Idle and Alternate Clock modes

CMOS Technology:

- Low power, high speed Flash technology
- Wide operating voltage range (2.5V to 5.5V)
- Industrial and Extended temperature ranges
- Low power consumption

dsPIC30F Motor Control and Power Conversion Family*

Device	Pins	Program Mem. Bytes/ Instructions	SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Moto Control PWM	A/D 10-bit 500 Ksps	Quad Enc	UART	SPI™	I²C™	CAN
dsPIC30F2010	28	12K/4K	512	1024	3	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F3010	28	24K/8K	1024	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F4012	28	48K/16K	2048	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	1
dsPIC30F3011	40/44	24K/8K	1024	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	-
dsPIC30F4011	40/44	48K/16K	2048	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66K/22K	2048	1024	5	4	4	8 ch	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144K/48K	8192	4096	5	8	8	8 ch	16 ch	Yes	2	2	1	2

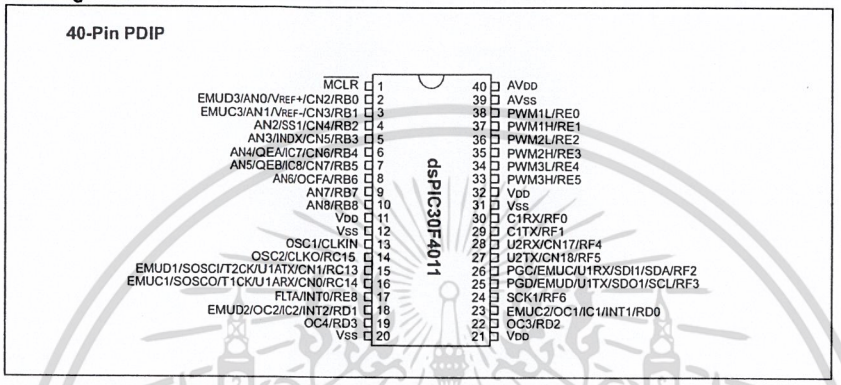
* This table provides a summary of the dsPIC30F6010 peripheral features. Other available devices in the dsPIC30F Motor Control and Power Conversion Family are shown for feature comparison.

 © 2005 Microchip Technology Inc.

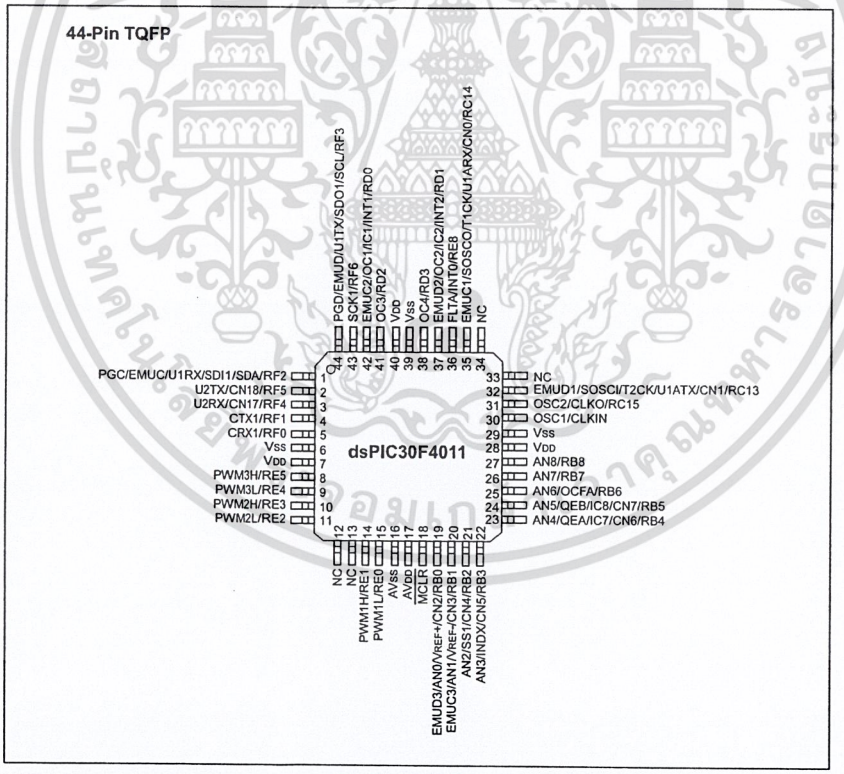
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Pin Diagrams

40-Pin PDIP



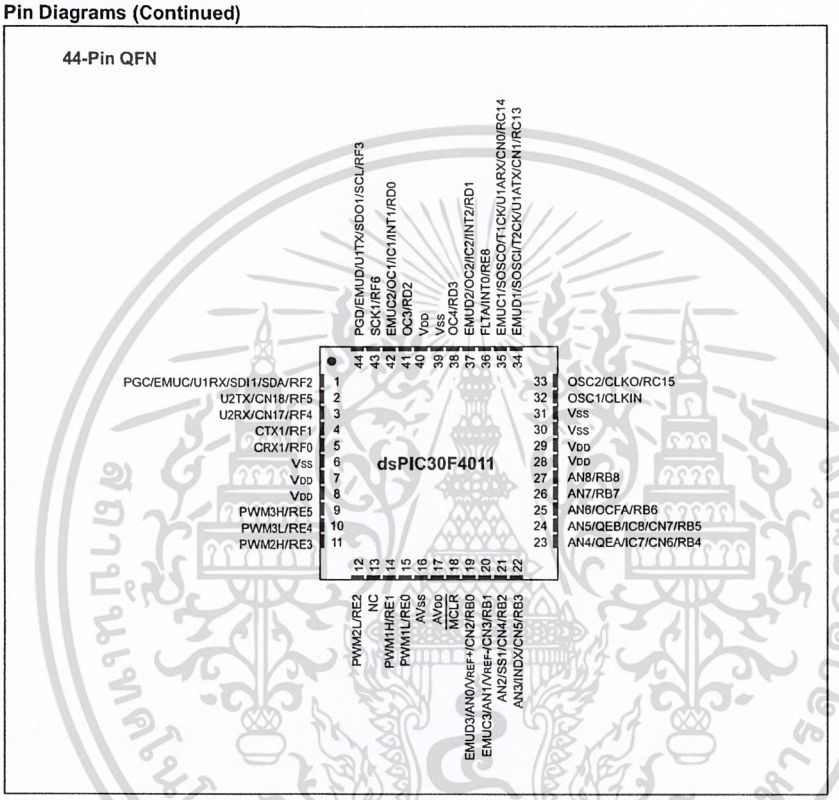
44-Pin TQFP



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Pin Diagrams (Continued)

44-Pin QFN

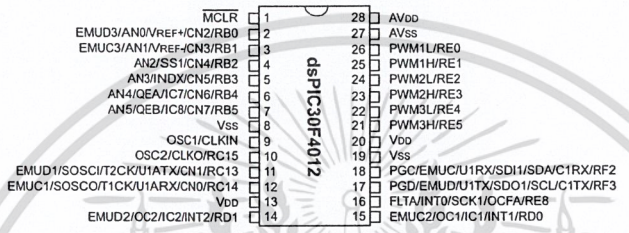


 © 2005 Microchip Technology Inc.

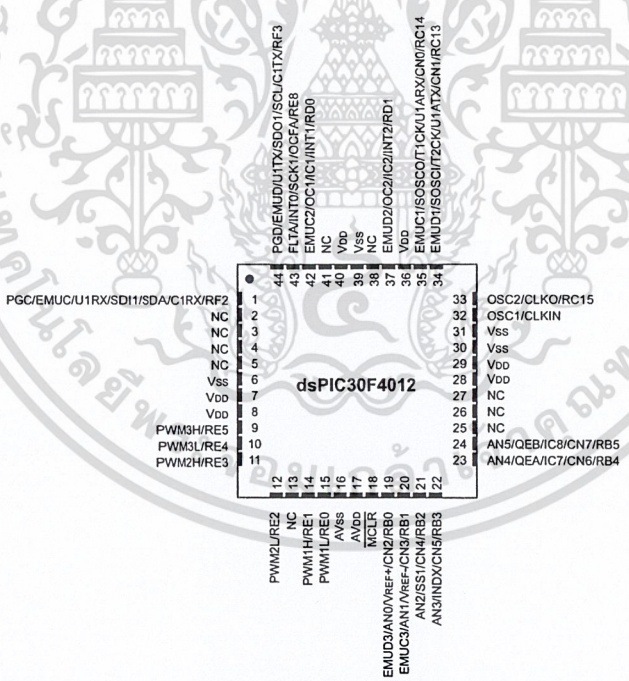
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Pin Diagrams (Continued)

28-Pin SPDIP
28-Pin SOIC



44-Pin QFN



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table of Contents

1.0 Device Overview 7
2.0 CPU Architecture Overview 15
3.0 Memory Organization 23
4.0 Address Generator Units 35
5.0 Interrupts 41
6.0 Flash Program Memory 47
7.0 Data EEPROM Memory 53
8.0 I/O Ports 57
9.0 Timer1 Module 63
10.0 Timer2/3 Module 67
11.0 Timer4/5 Module 73
12.0 Input Capture Module 77
13.0 Output Compare Module 81
14.0 Quadrature Encoder Interface (QEI) Module 85
15.0 Motor Control PWM Module 91
16.0 SPI™ Module 101
17.0 I2C™ Module 105
18.0 Universal Asynchronous Receiver Transmitter (UART) Module 113
19.0 CAN Module 121
20.0 10-bit High Speed Analog-to-Digital Converter (A/D) Module 131
21.0 System Integration 139
22.0 Instruction Set Summary 153
23.0 Development Support 161
24.0 Electrical Characteristics 167
25.0 Packaging Information 209
The Microchip Web Site 223
Customer Change Notification Service 223
Customer Support 223
Reader Response 224
Product Identification System 225

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)

When contacting a sales office, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com** to receive the most current information on all of our products.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

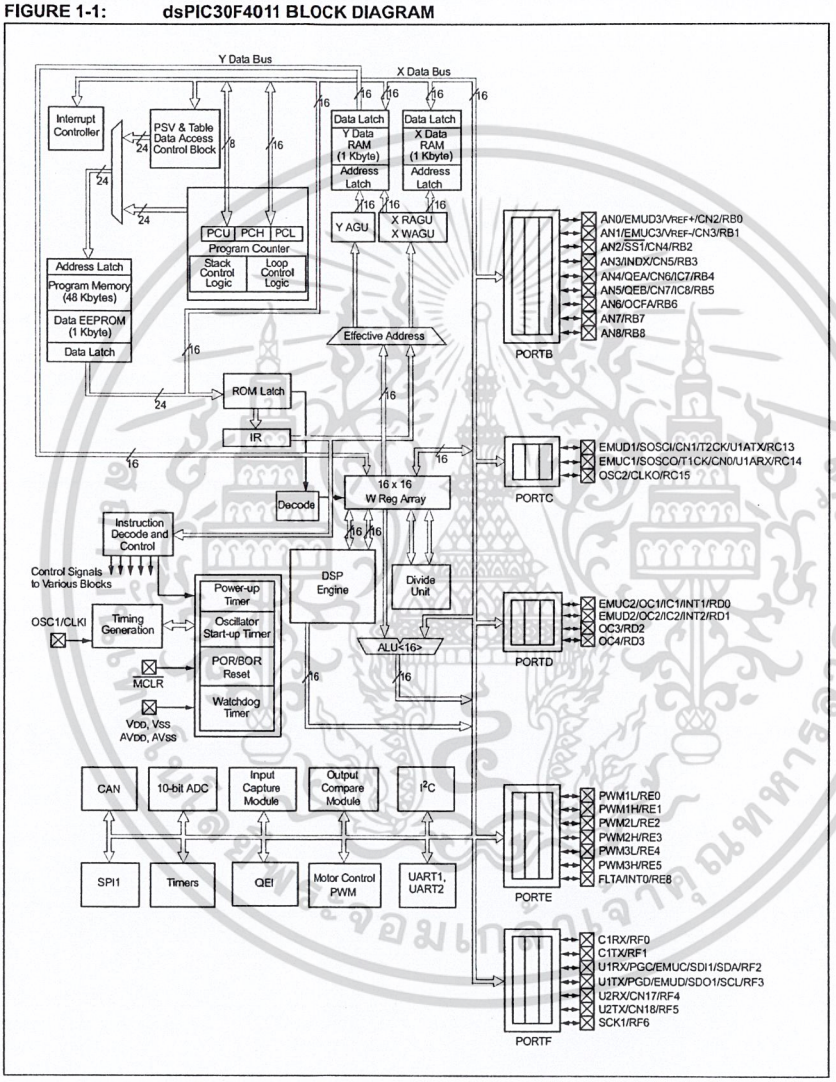
1.0 DEVICE OVERVIEW

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

This document contains device specific information for the dsPIC30F4011/4012 device. The dsPIC30F devices contain extensive Digital Signal Processor (DSP) functionality within a high performance 16-bit microcontroller (MCU) architecture. Figure 1-1 and Figure 1-2 show device block diagrams for the dsPIC30F4011 and dsPIC30F4012 device.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

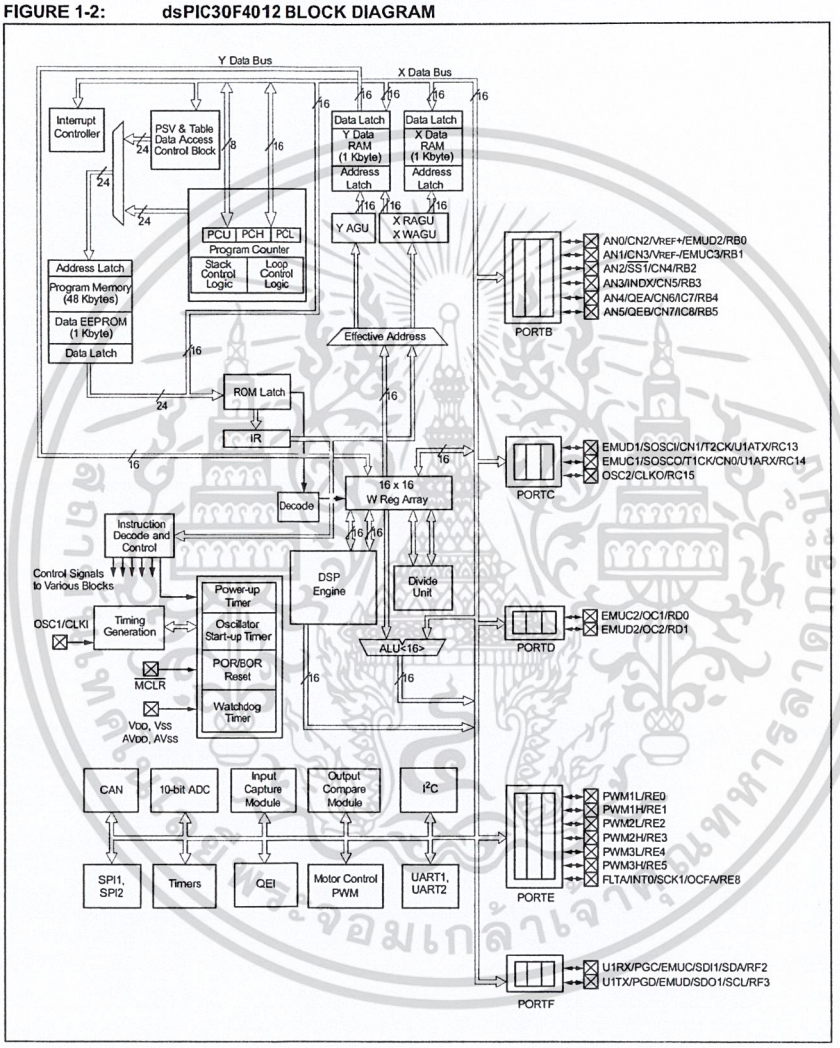
FIGURE 1-1: dsPIC30F4011 BLOCK DIAGRAM



 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 1-2: dsPIC30F4012 BLOCK DIAGRAM



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 1-1 provides a brief description of the device I/O pinout and the functions that are multiplexed to a port pin. Multiple functions may exist on one port pin. When multiplexing occurs, the peripheral module's functional requirements may force an override of the data direction of the port pin.

TABLE 1-1: dsPIC30F4011 I/O PIN DESCRIPTIONS

Pin Name	Pin Type	Buffer Type	Description
AN0-AN8	I	Analog	Analog input channels. AN0 and AN1 are also used for device programming data and clock inputs, respectively.
AVDD	P	P	Positive supply for analog module.
AVSS	P	P	Ground reference for analog module.
CLKI CLKO	I O	ST/CMOS —	External clock source input. Always associated with OSC1 pin function. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes. Always associated with OSC2 pin function.
CN0-CN7 CN17-CN18	I	ST	Input change notification inputs. Can be software programmed for internal weak pull-ups on all inputs.
C1RX C1TX	I O	ST —	CAN1 bus receive pin. CAN1 bus transmit pin.
EMUD EMUC EMUD1 EMUC1 EMUD2 EMUC2 EMUD3 EMUC3	I/O I/O I/O I/O I/O I/O I/O I/O	ST ST ST ST ST ST ST ST	ICD Primary Communication Channel data input/output pin. ICD Primary Communication Channel clock input/output pin. ICD Secondary Communication Channel data input/output pin. ICD Secondary Communication Channel clock input/output pin. ICD Tertiary Communication Channel data input/output pin. ICD Tertiary Communication Channel clock input/output pin. ICD Quaternary Communication Channel data input/output pin. ICD Quaternary Communication Channel clock input/output pin.
IC1, IC2, IC7, IC8	I	ST	Capture inputs 1, 2, 7 and 8.
INDX QEA QEB	I I I	ST ST ST	Quadrature Encoder Index Pulse input. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode.
INT0 INT1 INT2	I I I	ST ST ST	External interrupt 0. External interrupt 1. External interrupt 2.
FLTA PWM1L PWM1H PWM2L PWM2H PWM3L PWM3H	I O O O O O O	ST — — — — — —	PWM Fault A input. PWM 1 Low output. PWM 1 High output. PWM 2 Low output. PWM 2 High output. PWM 3 Low output. PWM 3 High output.
$\overline{MCLR}$	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low Reset to the device.
OCFA OC1-OC4	I O	ST —	Compare Fault A input (for Compare channels 1, 2, 3 and 4). Compare outputs 1 through 4.

Legend: CMOS = CMOS compatible input or output; Analog = Analog input; ST = Schmitt Trigger input with CMOS levels; O = Output; I = Input; P = Power

© 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 1-1: dsPIC30F4011 I/O PIN DESCRIPTIONS (CONTINUED)

Pin Name	Pin Type	Buffer Type	Description
OSC1 OSC2	I I/O	ST/CMOS —	Oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes.
PGD PGC	I/O I	ST ST	In-Circuit Serial Programming data input/output pin. In-Circuit Serial Programming clock input pin.
RB0-RB8	I/O	ST	PORTB is a bidirectional I/O port.
8RC13-RC15	8I/O	8ST	PORTC is a bidirectional I/O port.
RD0-RD3	I/O	ST	PORTD is a bidirectional I/O port.
RE0-RE5, RE8	I/O	ST	PORTE is a bidirectional I/O port.
RF0-RF6	I/O	ST	PORTF is a bidirectional I/O port.
SCK1 SDI1 SDO1 SS1	I/O I O I	ST ST — ST	Synchronous serial clock input/output for SPI™ 1. SPI 1 Data In. SPI 1 Data Out. SPI 1 Slave Synchronization.
SCL SDA	I/O I/O	ST ST	Synchronous serial clock input/output for I^2C. Synchronous serial data input/output for I^2C.
SOSCO SOSCI	O I	— ST/CMOS	32 kHz low power oscillator crystal output. 32 kHz low power oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise.
T1CK T2CK	I I	ST ST	Timer1 external clock input. Timer2 external clock input.
U1RX U1TX U1ARX U1ATX U2RX U2TX	I O I O I O	ST — ST — ST —	UART1 Receive. UART1 Transmit. UART1 Alternate Receive. UART1 Alternate Transmit. UART2 Receive. UART2 Transmit.
VDD	P	—	Positive supply for logic and I/O pins.
VSS	P	—	Ground reference for logic and I/O pins.
VREF+	I	Analog	Analog Voltage Reference (High) input.
VREF-	I	Analog	Analog Voltage Reference (Low) input.

Legend: CMOS = CMOS compatible input or output Analog = Analog input
ST = Schmitt Trigger input with CMOS levels O = Output
I = Input P = Power

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 1-2 provides a brief description of the device I/O pinout and the functions that are multiplexed to a port pin. Multiple functions may exist on one port pin. When multiplexing occurs, the peripheral module's functional requirements may force an override of the data direction of the port pin.

TABLE 1-2: dsPIC30F4012 I/O PIN DESCRIPTIONS

Pin Name	Pin Type	Buffer Type	Description
AN0-AN5	I	Analog	Analog input channels. AN0 and AN1 are also used for device programming data and clock inputs, respectively.
AVDD	P	P	Positive supply for analog module.
AVSS	P	P	Ground reference for analog module.
CLKI CLKO	I O	ST/CMOS —	External clock source input. Always associated with OSC1 pin function. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes. Always associated with OSC2 pin function.
CN0-CN7	I	ST	Input change notification inputs. Can be software programmed for internal weak pull-ups on all inputs.
C1RX C1TX	I O	ST —	CAN1 bus receive pin. CAN1 bus transmit pin.
EMUD EMUC EMUD1 EMUC1 EMUD2 EMUC2 EMUD3 EMUC3	I/O I/O I/O I/O I/O I/O I/O I/O	ST ST ST ST ST ST ST ST	ICD Primary Communication Channel data input/output pin. ICD Primary Communication Channel clock input/output pin. ICD Secondary Communication Channel data input/output pin. ICD Secondary Communication Channel clock input/output pin. ICD Tertiary Communication Channel data input/output pin. ICD Tertiary Communication Channel clock input/output pin. ICD Quaternary Communication Channel data input/output pin. ICD Quaternary Communication Channel clock input/output pin.
IC1, IC2, IC7, IC8	I	ST	Capture inputs 1, 2, 7 and 8.
INDX QEA QEB	I I I	ST ST ST	Quadrature Encoder Index Pulse input. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode. Quadrature Encoder Phase A input in QEI mode. Auxiliary Timer External Clock/Gate input in Timer mode.
INT0 INT1 INT2	I I I	ST ST ST	External interrupt 0. External interrupt 1. External interrupt 2.
FLTA PWM1L PWM1H PWM2L PWM2H PWM3L PWM3H	I O O O O O O	ST — — — — — —	PWM Fault A input. PWM 1 Low output. PWM 1 High output. PWM 2 Low output. PWM 2 High output. PWM 3 Low output. PWM 3 High output.
$\overline{MCLR}$	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low Reset to the device.
OCFA OC1, OC2	I O	ST —	Compare Fault A input (for Compare channels 1, 2, 3 and 4). Compare outputs 1 and 2.

Legend: CMOS = CMOS compatible input or output; Analog = Analog input
ST = Schmitt Trigger input with CMOS levels; O = Output
I = Input; P = Power

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 1-2: dsPIC30F4012 I/O PIN DESCRIPTIONS (CONTINUED)

Pin Name	Pin Type	Buffer Type	Description
OSC1 OSC2	I I/O	ST/CMOS —	Oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. Optionally functions as CLKO in RC and EC modes.
PGD PGC	I/O I	ST ST	In-Circuit Serial Programming data input/output pin. In-Circuit Serial Programming clock input pin.
RB0-RB5	I/O	ST	PORTB is a bidirectional I/O port.
RC13-RC15	8I/O	8ST	PORTC is a bidirectional I/O port.
RD0-RD1	I/O	ST	PORTD is a bidirectional I/O port.
RE0-RE5, RE8	I/O	ST	PORTE is a bidirectional I/O port.
RF2-RF3	I/O	ST	PORTF is a bidirectional I/O port.
SCK1 SDI1 SDO1	I/O I O	ST ST —	Synchronous serial clock input/output for SPI1. SPI1 Data In. SPI1 Data Out.
SCL SDA	I/O I/O	ST ST	Synchronous serial clock input/output for I^2C. Synchronous serial data input/output for I^2C.
SOSCO SOSCI	O I	— ST/CMOS	32 kHz low power oscillator crystal output. 32 kHz low power oscillator crystal input. ST buffer when configured in RC mode; CMOS otherwise.
T1CK T2CK	I I	ST ST	Timer1 external clock input. Timer2 external clock input.
U1RX U1TX U1ARX U1ATX	I O I O	ST — ST —	UART1 Receive. UART1 Transmit. UART1 Alternate Receive. UART1 Alternate Transmit.
VDD	P	—	Positive supply for logic and I/O pins.
VSS	P	—	Ground reference for logic and I/O pins.
VREF+	I	Analog	Analog Voltage Reference (High) input.
VREF-	I	Analog	Analog Voltage Reference (Low) input.

Legend: CMOS = CMOS compatible input or output Analog = Analog input
ST = Schmitt Trigger input with CMOS levels O = Output
I = Input P = Power

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

NOTES:

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.0 CPU ARCHITECTURE OVERVIEW

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

This document provides a summary of the dsPIC30F4011/4012 CPU and peripheral function. For a complete description of this functionality, please refer to the *dsPIC30F Family Reference Manual* (DS70046).

2.1 Core Overview

The core has a 24-bit instruction word. The Program Counter (PC) is 23 bits wide with the Least Significant (LS) bit always clear (see Section 3.1), and the Most Significant (MS) bit is ignored during normal program execution, except for certain specialized instructions. Thus, the PC can address up to 4M instruction words of user program space. An instruction pre-fetch mechanism is used to help maintain throughput. Program loop constructs, free from loop count management overhead, are supported using the DO and REPEAT instructions, both of which are interruptible at any point.

The working register array consists of 16x16-bit registers, each of which can act as data, address or offset registers. One working register (W15) operates as a software stack pointer for interrupts and calls.

The data space is 64 Kbytes (32K words) and is split into two blocks, referred to as X and Y data memory. Each block has its own independent Address Generation Unit (AGU). Most instructions operate solely through the X memory AGU, which provides the appearance of a single unified data space. The Multiply-Accumulate (MAC) class of dual source DSP instructions operate through both the X and Y AGUs, splitting the data address space into two parts (see Section 3.2). The X and Y data space boundary is device specific and cannot be altered by the user. Each data word consists of 2 bytes, and most instructions can address data either as words or bytes.

There are two methods of accessing data stored in program memory:

- The upper 32 Kbytes of data space memory can be mapped into the lower half (user space) of program space at any 16K program word boundary, defined by the 8-bit Program Space Visibility Page (PSVPAG) register. This lets any instruction access program space as if it were data space, with a limitation that the access requires an additional cycle. Moreover, only the lower 16 bits of each instruction word can be accessed using this method.
- SWWLinear indirect access of 32K word pages within program space is also possible using any working register, via table read and write instructions. Table read and write instructions can be used to access all 24 bits of an instruction word.

Overhead-free circular buffers (modulo addressing) are supported in both X and Y address spaces. This is primarily intended to remove the loop overhead for DSP algorithms.

The X AGU also supports bit-reversed addressing on destination effective addresses, to greatly simplify input or output data reordering for radix-2 FFT algorithms. Refer to Section 4.0 for details on modulo and bit-reversed addressing.

The core supports Inherent (no operand), Relative, Literal, Memory Direct, Register Direct, Register Indirect, Register Offset and Literal Offset Addressing modes. Instructions are associated with predefined Addressing modes, depending upon their functional requirements.

For most instructions, the core is capable of executing a data (or program data) memory read, a working register (data) read, a data memory write and a program (instruction) memory read per instruction cycle. As a result, 3-operand instructions are supported, allowing C = A+B operations to be executed in a single cycle.

A DSP engine has been included to significantly enhance the core arithmetic capability and throughput. It features a high speed 17-bit by 17-bit multiplier, a 40-bit ALU, two 40-bit saturating accumulators and a 40-bit bi-directional barrel shifter. Data in the accumulator or any working register can be shifted up to 16 bits right or 16 bits left in a single cycle. The DSP instructions operate seamlessly with all other instructions and have been designed for optimal real-time performance. The MAC class of instructions can concurrently fetch two data operands from memory, while multiplying two W registers. To enable this concurrent fetching of data operands, the data space has been split for these instructions and linear for all others. This has been achieved in a transparent and flexible manner, by dedicating certain working registers to each address space for the MAC class of instructions.

The core does not support a multi-stage instruction pipeline. However, a single stage instruction pre-fetch mechanism is used, which accesses and partially decodes instructions a cycle ahead of execution, in order to maximize available execution time. Most instructions execute in a single cycle, with certain exceptions.

The core features a vectored exception processing structure for traps and interrupts, with 62 independent vectors. The exceptions consist of up to 8 traps (of which 4 are reserved) and 54 interrupts. Each interrupt is prioritized based on a user assigned priority between 1 and 7 (1 being the lowest priority and 7 being the highest) in conjunction with a predetermined 'natural order'. Traps have fixed priorities, ranging from 8 to 15.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2 Programmer's Model

The programmer's model is shown in Figure 2-1 and consists of 16x16-bit working registers (W0 through W15), 2x40-bit accumulators (AccA and AccB), STATUS register (SR), Data Table Page register (TBLPAG), Program Space Visibility Page register (PSVPAG), DO and REPEAT registers (DOSTART, DOEND, DCOUNT and RCOUNT), and Program Counter (PC). The working registers can act as data, address or offset registers. All registers are memory mapped. W0 acts as the W register for file register addressing.

Some of these registers have a shadow register associated with each of them, as shown in Figure 2-1. The shadow register is used as a temporary holding register and can transfer its contents to or from its host register upon the occurrence of an event. None of the shadow registers are accessible directly. The following rules apply for transfer of registers into and out of shadows.

- `PUSH.S` and `POP.S`
  W0, W1, W2, W3, SR (DC, N, OV, Z and C bits only) are transferred.
- `DO` instruction
  DOSTART, DOEND, DCOUNT shadows are pushed on loop start, and popped on loop end.

When a byte operation is performed on a working register, only the Least Significant Byte of the target register is affected. However, a benefit of memory mapped working registers is that both the Least and Most Significant Bytes can be manipulated through byte wide data memory space accesses.

2.2.1 SOFTWARE STACK POINTER/ FRAME POINTER

The dsPIC® devices contain a software stack. W15 is the dedicated software stack pointer (SP), and will be automatically modified by exception processing and subroutine calls and returns. However, W15 can be referenced by any instruction in the same manner as all other W registers. This simplifies the reading, writing and manipulation of the stack pointer (e.g., creating stack frames).

Note: In order to protect against misaligned stack accesses, W15<0> is always clear.

W15 is initialized to 0x0800 during a Reset. The user may reprogram the SP during initialization to any location within data space.

W14 has been dedicated as a stack frame pointer as defined by the `LNK` and `ULNK` instructions. However, W14 can be referenced by any instruction in the same manner as all other W registers.

2.2.2 STATUS REGISTER

The dsPIC core has a 16-bit Status Register (SR), the LS Byte of which is referred to as the SR Low Byte (SRL) and the MS Byte as the SR High Byte (SRH). See Figure 2-1 for SR layout.

SRL contains all the MCU ALU operation status flags (including the Z bit), as well as the CPU Interrupt Priority Level status bits, IPL<2:0>, and the REPEAT active status bit, RA. During exception processing, SRL is concatenated with the MS Byte of the PC to form a complete word value which is then stacked.

The upper byte of the SR register contains the DSP Adder/Subtractor status bits, the DO Loop Active bit (DA) and the Digit Carry (DC) status bit.

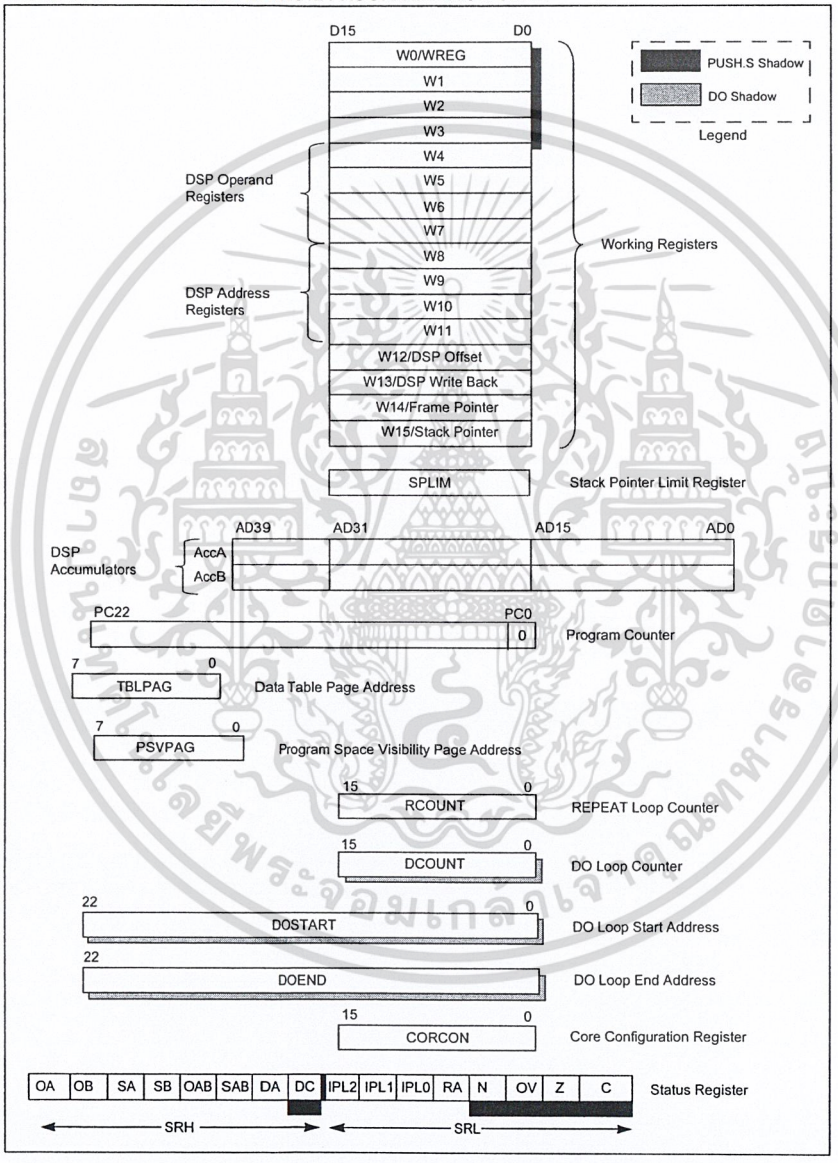
2.2.3 PROGRAM COUNTER

The Program Counter is 23 bits wide. Bit 0 is always clear. Therefore, the PC can address up to 4M instruction words.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 2-1: dsPIC30F4011/4012 PROGRAMMER'S MODEL



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3 Divide Support

The dsPIC devices feature a 16/16-bit signed fractional divide operation, as well as 32/16-bit and 16/16-bit signed and unsigned integer divide operations, in the form of single instruction iterative divides. The following instructions and data sizes are supported:

1. DIVF – 16/16 signed fractional divide
2. DIV.sd – 32/16 signed divide
3. DIV.ud – 32/16 unsigned divide
4. DIV.sw – 16/16 signed divide
5. DIV.uw – 16/16 unsigned divide

The divide instructions must be executed within a REPEAT loop. Any other form of execution (e.g. a series of discrete divide instructions) will not function correctly because the instruction flow depends on RCOUNT. The divide instruction does not automatically set up the RCOUNT value, and it must, therefore, be explicitly and correctly specified in the REPEAT instruction, as shown in Table 2-1 (REPEAT will execute the target instruction {operand value+1} times). The REPEAT loop count must be set up for 18 iterations of the DIV/DIVF instruction. Thus, a complete divide operation requires 19 cycles.

Note: The Divide flow is interruptible. However, the user needs to save the context as appropriate.

TABLE 2-1: DIVIDE INSTRUCTIONS

Instruction	Function
DIVF	Signed fractional divide: Wm/Wn → W0; Rem → W1
DIV.sd	Signed divide: (Wm+1:Wm)/Wn → W0; Rem → W1
DIV.sw (or DIV.s)	Signed divide: Wm/Wn → W0; Rem → W1
DIV.ud	Unsigned divide: (Wm+1:Wm)/Wn → W0; Rem → W1
DIV.uw (or DIV.u)	Unsigned divide: Wm/Wn → W0; Rem → W1

2.4 DSP Engine

The DSP engine consists of a high speed 17-bit x 17-bit multiplier, a barrel shifter, and a 40-bit adder/ Subtractor (with two target accumulators, round and saturation logic).

The dsPIC30F devices have a single instruction flow which can execute either DSP or MCU instructions. Many of the hardware resources are shared between the DSP and MCU instructions. For example, the instruction set has both DSP and MCU Multiply instructions which use the same hardware multiplier.

The DSP engine also has the capability to perform inherent accumulator-to-accumulator operations, which require no additional data. These instructions are ADD, SUB and NEG.

The DSP engine has various options selected through various bits in the CPU Core Configuration Register (CORCON), as listed below:

1. Fractional or integer DSP multiply (IF).
2. Signed or unsigned DSP multiply (US).
3. Conventional or convergent rounding (RND).
4. Automatic saturation on/off for AccA (SATA).
5. Automatic saturation on/off for AccB (SATB).
6. Automatic saturation on/off for writes to data memory (SATDW).
7. Accumulator Saturation mode selection (ACCSAT).

Note: For CORCON layout, see Table 4-2.

A block diagram of the DSP engine is shown in Figure 2-2.

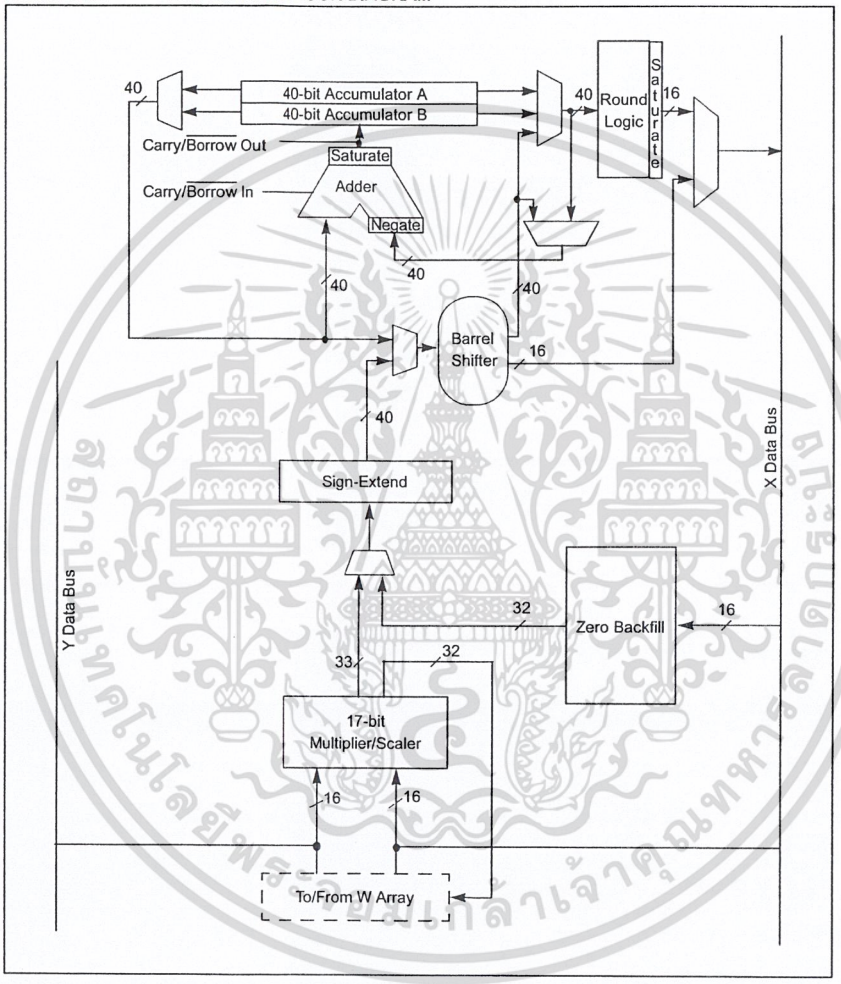
TABLE 2-2: DSP INSTRUCTION SUMMARY

Instruction	Algebraic Operation
CLR	A = 0
ED	A = $(x - y)^2$
EDAC	A = A + $(x - y)^2$
MAC	A = A + (x * y)
MOVSAC	No change in A
MPY	A = x * y
MPY.N	A = – x * y
MSC	A = A – x * y

© 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 2-2: DSP ENGINE BLOCK DIAGRAM



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.1 MULTIPLIER

The 17x17-bit multiplier is capable of signed or unsigned operation and can multiplex its output using a scaler to support either 1.31 fractional (Q31) or 32-bit integer results. Unsigned operands are zero-extended into the 17th bit of the multiplier input value. Signed operands are sign-extended into the 17th bit of the multiplier input value. The output of the 17x17-bit multiplier/scaler is a 33-bit value, which is sign-extended to 40 bits. Integer data is inherently represented as a signed two's complement value, where the MSB is defined as a sign bit. Generally speaking, the range of an N-bit two's complement integer is -2^{N-1} to $2^{N-1} - 1$. For a 16-bit integer, the data range is -32768 (0x8000) to 32767 (0x7FFF), including 0. For a 32-bit integer, the data range is -2,147,483,648 (0x8000 0000) to 2,147,483,645 (0x7FFF FFFF).

When the multiplier is configured for fractional multiplication, the data is represented as a two's complement fraction, where the MSB is defined as a sign bit and the radix point is implied to lie just after the sign bit (QX format). The range of an N-bit two's complement fraction with this implied radix point is -1.0 to $(1-2^{1-N})$. For a 16-bit fraction, the Q15 data range is -1.0 (0x8000) to 0.999969482 (0x7FFF), including 0 and has a precision of 3.01518×10^{-5}. In fractional mode, a 16x16 multiply operation generates a 1.31 product, which has a precision of 4.65661×10^{-10}.

The same multiplier is used to support the MCU multiply instructions, which include integer 16-bit signed, unsigned and mixed sign multiplies.

The MUL instruction may be directed to use byte or word sized operands. Byte operands will direct a 16-bit result, and word operands will direct a 32-bit result to the specified register(s) in the W array.

2.4.2 DATA ACCUMULATORS AND ADDER/SUBTRACTOR

The data accumulator consists of a 40-bit adder/subtractor with automatic sign extension logic. It can select one of two accumulators (A or B) as its pre-accumulation source and post-accumulation destination. For the ADD and LAC instructions, the data to be accumulated or loaded can be optionally scaled via the barrel shifter, prior to accumulation.

2.4.2.1 Adder/Subtractor, Overflow and Saturation

The adder/subtractor is a 40-bit adder with an optional zero input into one side and either true or complement data into the other input. In the case of addition, the carry/borrow input is active high and the other input is true data (not complemented), whereas in the case of subtraction, the carry/borrow input is active low and the other input is complemented. The adder/subtractor generates overflow status bits SA/SB and OA/OB, which are latched and reflected in the status register.

- Overflow from bit 39: this is a catastrophic overflow in which the sign of the accumulator is destroyed.
- Overflow into guard bits 32 through 39: this is a recoverable overflow. This bit is set whenever all the guard bits are not identical to each other.

The adder has an additional saturation block which controls accumulator data saturation, if selected. It uses the result of the adder, the overflow status bits described above, and the SATA/B (CORCON<7:6>) and ACCSAT (CORCON<4>) mode control bits to determine when and to what value to saturate.

Six status register bits have been provided to support saturation and overflow; they are:

1. OA:
   AccA overflowed into guard bits
2. OB:
   AccB overflowed into guard bits
3. SA:
   AccA saturated (bit 31 overflow and saturation)
   or
   AccA overflowed into guard bits and saturated (bit 39 overflow and saturation)
4. SB:
   AccB saturated (bit 31 overflow and saturation)
   or
   AccB overflowed into guard bits and saturated (bit 39 overflow and saturation)
5. OAB:
   Logical OR of OA and OB
6. SAB:
   Logical OR of SA and SB

The OA and OB bits are modified each time data passes through the adder/Subtractor. When set, they indicate that the most recent operation has overflowed into the accumulator guard bits (bits 32 through 39). The OA and OB bits can also optionally generate an arithmetic warning trap when set and the corresponding overflow trap flag enable bit (OVATEN, OVBTEN) in the INTCON1 register (refer to Section 5.0) is set. This allows the user to take immediate action, for example, to correct system gain.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

The SA and SB bits are modified each time data passes through the adder/subtractor, but can only be cleared by the user. When set, they indicate that the accumulator has overflowed its maximum range (bit 31 for 32-bit saturation, or bit 39 for 40-bit saturation) and will be saturated (if saturation is enabled). When saturation is not enabled, SA and SB default to bit 39 overflow and thus indicate that a catastrophic overflow has occurred. If the COVTE bit in the INTCON1 register is set, SA and SB bits will generate an arithmetic warning trap when saturation is disabled.

The overflow and saturation status bits can optionally be viewed in the Status Register (SR) as the logical OR of OA and OB (in bit OAB) and the logical OR of SA and SB (in bit SAB). This allows programmers to check one bit in the Status Register to determine if either accumulator has overflowed, or one bit to determine if either accumulator has saturated. This would be useful for complex number arithmetic which typically uses both the accumulators.

The device supports three Saturation and Overflow modes.

1. Bit 39 Overflow and Saturation:
   When bit 39 overflow and saturation occurs, the saturation logic loads the maximally positive 9.31 (0x7FFFFFFFFF) or maximally negative 9.31 value (0x8000000000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user. This is referred to as 'super saturation' and provides protection against erroneous data or unexpected algorithm problems (e.g., gain calculations).
2. Bit 31 Overflow and Saturation:
   When bit 31 overflow and saturation occurs, the saturation logic then loads the maximally positive 1.31 value (0x007FFFFFFF) or maximally negative 1.31 value (0x0080000000) into the target accumulator. The SA or SB bit is set and remains set until cleared by the user. When this Saturation mode is in effect, the guard bits are not used (so the OA, OB or OAB bits are never set).
3. Bit 39 Catastrophic Overflow
   The bit 39 overflow status bit from the adder is used to set the SA or SB bit, which remain set until cleared by the user. No saturation operation is performed and the accumulator is allowed to overflow (destroying its sign). If the COVTE bit in the INTCON1 register is set, a catastrophic overflow can initiate a trap exception.

2.4.2.2 Accumulator 'Write Back'

The MAC class of instructions (with the exception of MPY, MPY.N, ED and EDAC) can optionally write a rounded version of the high word (bits 31 through 16) of the accumulator that is not targeted by the instruction into data space memory. The write is performed across the X bus into combined X and Y address space. The following addressing modes are supported:

1. W13, Register Direct:
   The rounded contents of the non-target accumulator are written into W13 as a 1.15 fraction.
2. [W13]+=2, Register Indirect with Post-Increment:
   The rounded contents of the non-target accumulator are written into the address pointed to by W13 as a 1.15 fraction. W13 is then incremented by 2 (for a word write).

2.4.2.3 Round Logic

The round logic is a combinational block, which performs a conventional (biased) or convergent (unbiased) round function during an accumulator write (store). The Round mode is determined by the state of the RND bit in the CORCON register. It generates a 16-bit, 1.15 data value which is passed to the data space write saturation logic. If rounding is not indicated by the instruction, a truncated 1.15 data value is stored and the LS Word is simply discarded.

Conventional rounding takes bit 15 of the accumulator, zero-extends it and adds it to the ACCxH word (bits 16 through 31 of the accumulator). If the ACCxL word (bits 0 through 15 of the accumulator) is between 0x8000 and 0xFFFF (0x8000 included), ACCxH is incremented. If ACCxL is between 0x0000 and 0x7FFF, ACCxH is left unchanged. A consequence of this algorithm is that over a succession of random rounding operations, the value will tend to be biased slightly positive.

Convergent (or unbiased) rounding operates in the same manner as conventional rounding, except when ACCxL equals 0x8000. If this is the case, the LS bit (bit 16 of the accumulator) of ACCxH is examined. If it is '1', ACCxH is incremented. If it is '0', ACCxH is not modified. Assuming that bit 16 is effectively random in nature, this scheme will remove any rounding bias that may accumulate.

The SAC and SAC.R instructions store either a truncated (SAC) or rounded (SAC.R) version of the contents of the target accumulator to data memory, via the X bus (subject to data saturation, see Section 2.4.2.4). Note that for the MAC class of instructions, the accumulator write back operation will function in the same manner, addressing combined MCU (X and Y) data space though the X bus. For this class of instructions, the data is always subject to rounding.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4.2.4 Data Space Write Saturation

In addition to adder/subtractor saturation, writes to data space may also be saturated, but without affecting the contents of the source accumulator. The data space write saturation logic block accepts a 16-bit, 1.15 fractional value from the round logic block as its input, together with overflow status from the original source (accumulator) and the 16-bit round adder. These are combined and used to select the appropriate 1.15 fractional value as output to write to data space memory.

If the SATDW bit in the CORCON register is set, data (after rounding or truncation) is tested for overflow and adjusted accordingly. For input data greater than 0x007FFF, data written to memory is forced to the maximum positive 1.15 value, 0x7FFF. For input data less than 0xFF8000, data written to memory is forced to the maximum negative 1.15 value, 0x8000. The MS bit of the source (bit 39) is used to determine the sign of the operand being tested.

If the SATDW bit in the CORCON register is not set, the input data is always passed through unmodified under all conditions.

2.4.3 BARREL SHIFTER

The barrel shifter is capable of performing up to 16-bit arithmetic or logic right shifts, or up to 16-bit left shifts in a single cycle. The source can be either of the two DSP accumulators or the X bus (to support multi-bit shifts of register or memory data).

The shifter requires a signed binary value to determine both the magnitude (number of bits) and direction of the shift operation. A positive value will shift the operand right. A negative value will shift the operand left. A value of 0 will not modify the operand.

The barrel shifter is 40 bits wide, thereby obtaining a 40-bit result for DSP shift operations and a 16-bit result for MCU shift operations. Data from the X bus is presented to the barrel shifter between bit positions 16 to 31 for right shifts, and bit positions 0 to 15 for left shifts.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.0 MEMORY ORGANIZATION

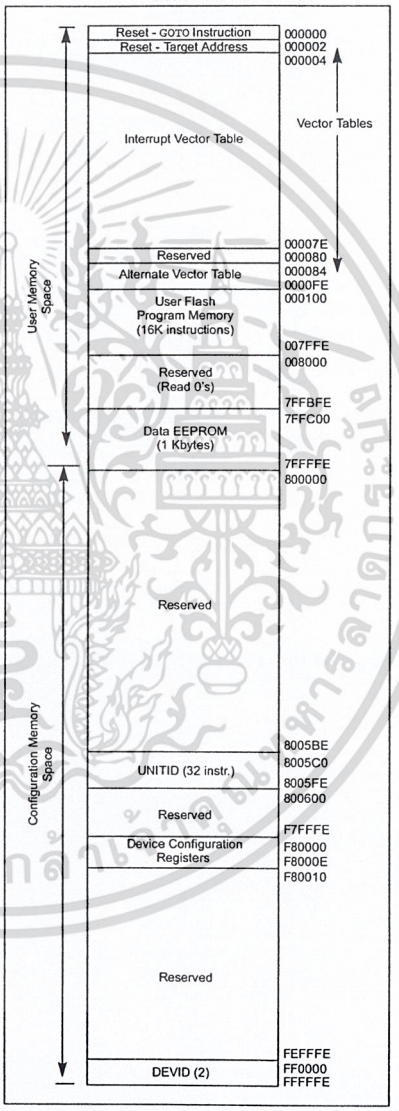
Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

3.1 Program Address Space

The program address space is 4M instruction words. It is addressable by the 23-bit PC, table instruction Effective Address (EA), or data space EA, when program space is mapped into data space, as defined by Table 3-1. Note that the program space address is incremented by two between successive program words, in order to provide compatibility with data space addressing.

User program space access is restricted to the lower 4M instruction word address range (0x000000 to 0x7FFFFE), for all accesses other than `TBLRD`/`TBLWT`, which use TBLPAG<7> to determine user or configuration space access. In Table 3-1, Read/Write instructions, bit 23 allows access to the Device ID, the User ID and the configuration bits. Otherwise, bit 23 is always clear.

FIGURE 3-1: PROGRAM SPACE MEMORY MAP FOR dsPIC30F4011/4012

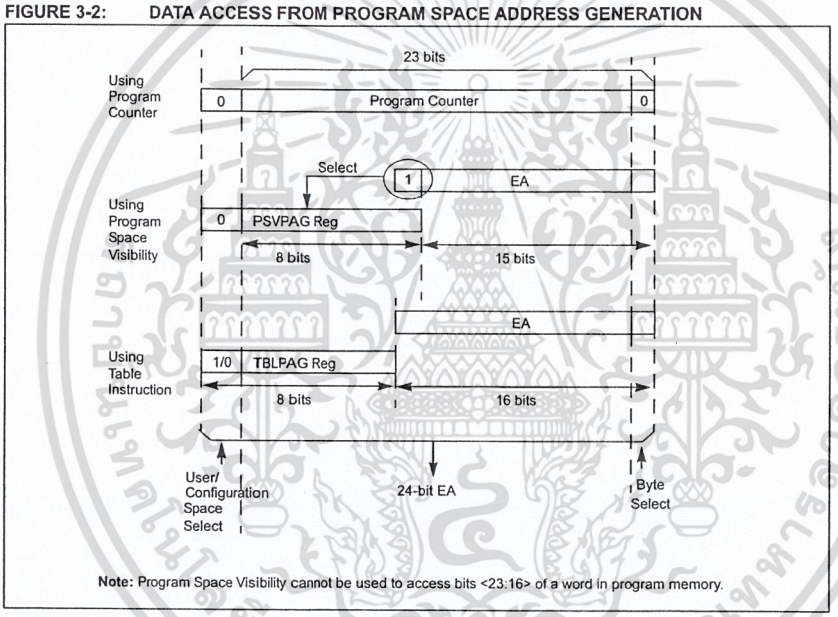


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 3-1: PROGRAM SPACE ADDRESS CONSTRUCTION

Access Type	Access Space	Program Space Address				
		<23>	<22:16>	<15>	<14:1>	<0>
Instruction Access	User	0	PC<22:1>			0
TBLRD/TBLWT	User (TBLPAG<7> = 0)	TBLPAG<7:0>		Data EA <15:0>		
TBLRD/TBLWT	Configuration (TBLPAG<7> = 1)	TBLPAG<7:0>		Data EA <15:0>		
Program Space Visibility	User	0	PSVPAG<7:0>		Data EA <14:0>	

FIGURE 3-2: DATA ACCESS FROM PROGRAM SPACE ADDRESS GENERATION



Note: Program Space Visibility cannot be used to access bits <23:16> of a word in program memory.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.1.1 DATA ACCESS FROM PROGRAM MEMORY USING TABLE INSTRUCTIONS

This architecture fetches 24-bit wide program memory. Consequently, instructions are always aligned. However, as the architecture is modified Harvard, data can also be present in program space.

There are two methods by which program space can be accessed; via special table instructions, or through the remapping of a 16K word program space page into the upper half of data space (see Section 3.1.2). The TBLRDL and TBLWTL instructions offer a direct method of reading or writing the LS Word of any address within program space, without going through data space. The TBLRDH and TBLWTH instructions are the only method whereby the upper 8 bits of a program space word can be accessed as data.

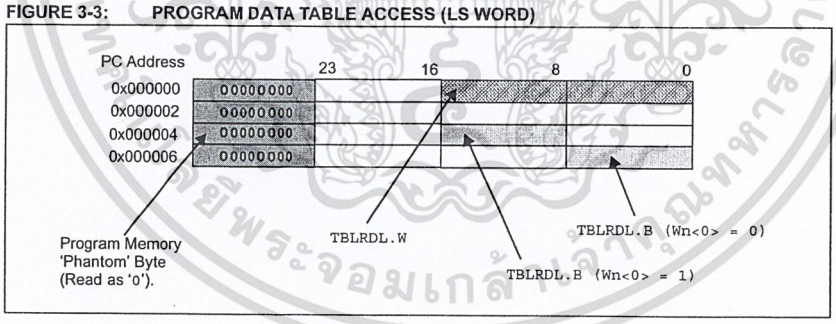
The PC is incremented by two for each successive 24-bit program word. This allows program memory addresses to directly map to data space addresses. Program memory can thus be regarded as two 16-bit word wide address spaces, residing side by side, each with the same address range. TBLRDL and TBLWTL access the space which contains the LS Data Word, and TBLRDH and TBLWTH access the space which contains the MS Data Byte.

Figure 3-2 shows how the EA is created for table operations and data space accesses (PSV = 1). Here, P<23:0> refers to a program space word, whereas D<15:0> refers to a data space word.

A set of Table Instructions are provided to move byte or word sized data to and from program space.

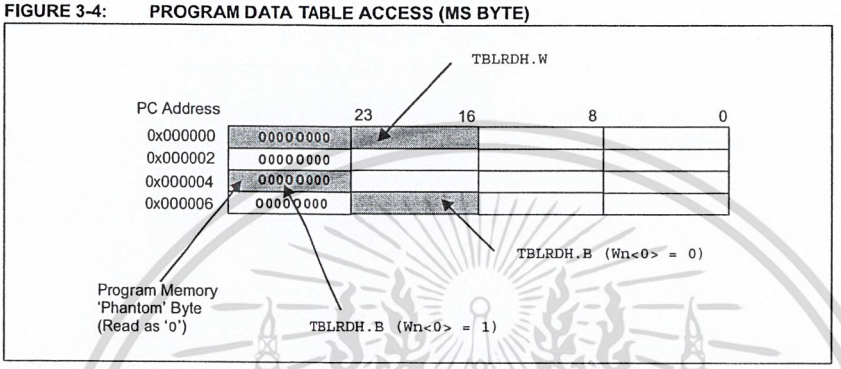
1. **TBLRDL:** Table Read Low
   Word: Read the LS Word of the program address;
   P<15:0> maps to D<15:0>.
   Byte: Read one of the LS Bytes of the program address;
   P<7:0> maps to the destination byte when byte select = 0;
   P<15:8> maps to the destination byte when byte select = 1.
2. **TBLWTL:** Table Write Low (refer to Section 6.0 for details on Flash Programming).
3. **TBLRDH:** Table Read High
   Word: Read the MS Word of the program address;
   P<23:16> maps to D<7:0>; D<15:8> always be = 0.
   Byte: Read one of the MS Bytes of the program address;
   P<23:16> maps to the destination byte when byte select = 0;
   The destination byte will always be = 0 when byte select = 1.
4. **TBLWTH:** Table Write High (refer to Section 6.0 for details on Flash Programming).

FIGURE 3-3: PROGRAM DATA TABLE ACCESS (LS WORD)



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 3-4: PROGRAM DATA TABLE ACCESS (MS BYTE)



3.1.2 DATA ACCESS FROM PROGRAM MEMORY USING PROGRAM SPACE VISIBILITY

The upper 32 Kbytes of data space may optionally be mapped into any 16K word program space page. This provides transparent access of stored constant data from X data space, without the need to use special instructions (i.e., `TBLRDL/H`, `TBLWTL/H` instructions).

Program space access through the data space occurs if the MS bit of the data space EA is set and program space visibility is enabled, by setting the PSV bit in the Core Control register (CORCON). The functions of CORCON are discussed in Section 2.4, DSP Engine.

Data accesses to this area add an additional cycle to the instruction being executed, since two program memory fetches are required.

Note that the upper half of addressable data space is always part of the X data space. Therefore, when a DSP operation uses program space mapping to access this memory region, Y data space should typically contain state (variable) data for DSP operations, whereas X data space should typically contain coefficient (constant) data.

Although each data space address, 0x8000 and higher, maps directly into a corresponding program memory address (see Figure 3-5), only the lower 16-bits of the 24-bit program word are used to contain the data. The upper 8 bits should be programmed to force an illegal instruction to maintain machine robustness. Refer to the *dsPIC30F Programmer's Reference Manual* (DS70030) for details on instruction encoding.

Note that by incrementing the PC by 2 for each program memory word, the LS 15 bits of data space addresses directly map to the LS 15 bits in the corresponding program space addresses. The remaining bits are provided by the Program Space Visibility Page register, PSVPAG<7:0>, as shown in Figure 3-5.

Note:	PSV access is temporarily disabled during Table Reads/Writes.

For instructions that use PSV which are executed outside a REPEAT loop:

- The following instructions will require one instruction cycle in addition to the specified execution time:
  - `MAC` class of instructions with data operand pre-fetch
  - `MOV` instructions
  - `MOV.D` instructions
- All other instructions will require two instruction cycles in addition to the specified execution time of the instruction.

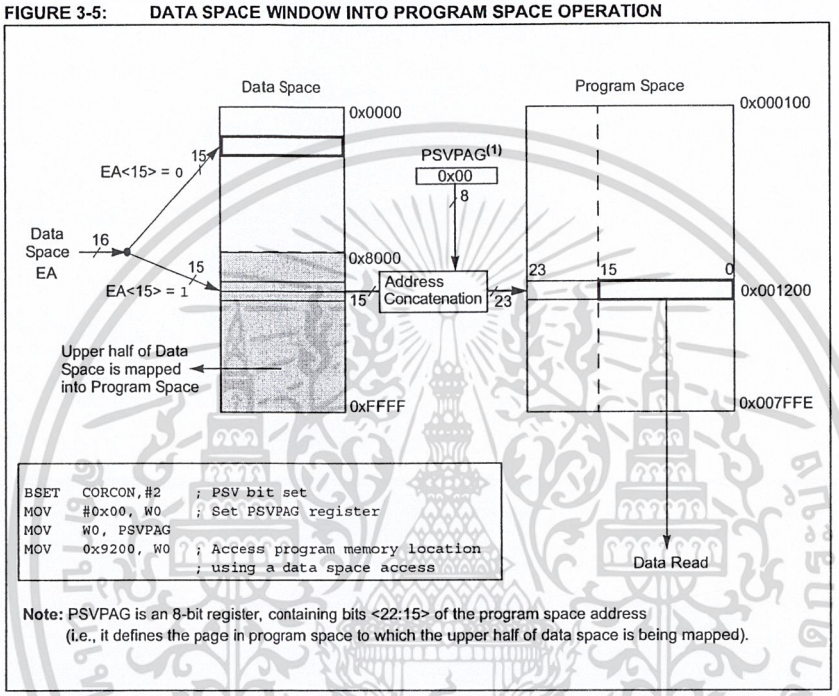
For instructions that use PSV which are executed inside a REPEAT loop:

- The following instances will require two instruction cycles in addition to the specified execution time of the instruction:
  - Execution in the first iteration
  - Execution in the last iteration
  - Execution prior to exiting the loop due to an interrupt
  - Execution upon re-entering the loop after an interrupt is serviced
- Any other iteration of the REPEAT loop will allow the instruction, accessing data using PSV, to execute in a single cycle.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 3-5: DATA SPACE WINDOW INTO PROGRAM SPACE OPERATION



Note: PSVPAG is an 8-bit register, containing bits <22:15> of the program space address (i.e., it defines the page in program space to which the upper half of data space is being mapped).

3.2 Data Address Space

The core has two data spaces. The data spaces can be considered either separate (for some DSP instructions), or as one unified linear address range (for MCU instructions). The data spaces are accessed using two Address Generation Units (AGUs) and separate data paths.

3.2.1 DATA SPACE MEMORY MAP

The data space memory is split into two blocks, X and Y data space. A key element of this architecture is that Y space is a subset of X space, and is fully contained within X space. In order to provide an apparent linear addressing space, X and Y spaces have contiguous addresses.

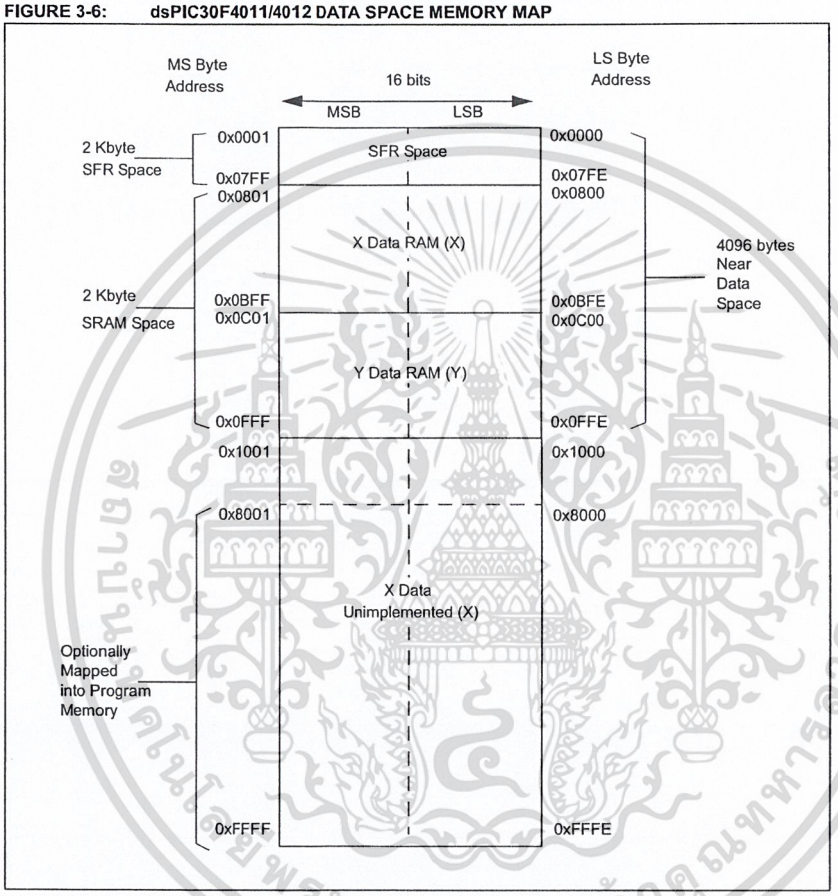
When executing any instruction other than one of the MAC class of instructions, the X block consists of the 64 Kbyte data address space (including all Y addresses). When executing one of the MAC class of instructions, the X block consists of the 64 Kbyte data address space excluding the Y address block (for data reads only). In other words, all other instructions regard the entire data memory as one composite address space. The MAC class instructions extract the Y address space from data space and address it using EAs sourced from W10 and W11. The remaining X data space is addressed using W8 and W9. Both address spaces are concurrently accessed only with the MAC class instructions.

A data space memory map is shown in Figure 3-6.

Figure 3-7 shows a graphical summary of how X and Y data spaces are accessed for MCU and DSP instructions.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

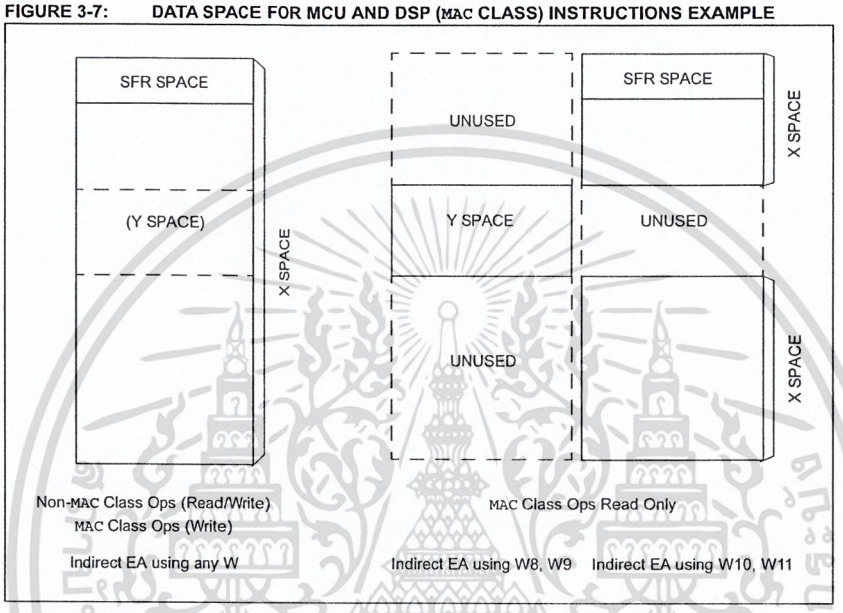
FIGURE 3-6: dsPIC30F4011/4012 DATA SPACE MEMORY MAP



 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 3-7: DATA SPACE FOR MCU AND DSP (MAC CLASS) INSTRUCTIONS EXAMPLE



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2.2 DATA SPACES

The X data space is used by all instructions and supports all addressing modes. There are separate read and write data buses. The X read data bus is the return data path for all instructions that view data space as combined X and Y address space. It is also the X address space data path for the dual operand read instructions (MAC class). The X write data bus is the only write path to data space for all instructions.

The X data space also supports Modulo Addressing for all instructions, subject to Addressing mode restrictions. Bit-Reversed Addressing is only supported for writes to X data space.

The Y data space is used in concert with the X data space by the MAC class of instructions (CLR, ED, EDAC, MAC, MOVSAC, MPY, MPY.N and MSC) to provide two concurrent data read paths. No writes occur across the Y bus. This class of instructions dedicates two W register pointers, W10 and W11, to always address Y data space, independent of X data space, whereas W8 and W9 always address X data space. Note that during accumulator write back, the data address space is considered a combination of X and Y data spaces, so the write occurs across the X bus. Consequently, the write can be to any address in the entire data space.

The Y data space can only be used for the data prefetch operation associated with the MAC class of instructions. It also supports Modulo Addressing for automated circular buffers. Of course, all other instructions can access the Y data address space through the X data path, as part of the composite linear space.

The boundary between the X and Y data spaces is defined as shown in Figure 3-6 and is not user programmable. Should an EA point to data outside its own assigned address space, or to a location outside physical memory, an all-zero word/byte will be returned. For example, although Y address space is visible by all non-MAC instructions using any Addressing mode, an attempt by a MAC instruction to fetch data from that space, using W8 or W9 (X space pointers), will return 0x0000.

TABLE 3-2: EFFECT OF INVALID MEMORY ACCESSES

Attempted Operation	Data Returned
EA = an unimplemented address	0x0000
W8 or W9 used to access Y data space in a MAC instruction	0x0000
W10 or W11 used to access X data space in a MAC instruction	0x0000

All effective addresses are 16 bits wide and point to bytes within the data space. Therefore, the data space address range is 64 Kbytes or 32K words.

3.2.3 DATA SPACE WIDTH

The core data width is 16-bits. All internal registers are organized as 16-bit wide words. Data space memory is organized in byte addressable, 16-bit wide blocks.

3.2.4 DATA ALIGNMENT

To help maintain backward compatibility with PICmicro® devices and improve data space memory usage efficiency, the dsPIC30F instruction set supports both word and byte operations. Data is aligned in data memory and registers as words, but all data space EAs resolve to bytes. Data byte reads will read the complete word, which contains the byte, using the LS bit of any EA to determine which byte to select. The selected byte is placed onto the LS Byte of the X data path (no byte accesses are possible from the Y data path as the MAC class of instruction can only fetch words). That is, data memory and registers are organized as two parallel byte wide entities with shared (word) address decode, but separate write lines. Data byte writes only write to the corresponding side of the array or register which matches the byte address.

As a consequence of this byte accessibility, all effective address calculations (including those generated by the DSP operations, which are restricted to word sized data) are internally scaled to step through word aligned memory. For example, the core would recognize that Post-Modified Register Indirect Addressing mode, [Ws++], will result in a value of Ws+1 for byte operations and Ws+2 for word operations.

All word accesses must be aligned to an even address. Mis-aligned word data fetches are not supported, so care must be taken when mixing byte and word operations, or translating from 8-bit MCU code. Should a misaligned read or write be attempted, an Address Error trap will be generated. If the error occurred on a read, the instruction underway is completed, whereas if it occurred on a write, the instruction will be executed but the write will not occur. In either case, a trap will then be executed, allowing the system and/or user to examine the machine state prior to execution of the address fault.

FIGURE 3-8: DATA ALIGNMENT

	MS Byte 15	LS Byte 8 7 0	
0001	Byte 1	Byte 0	0000
0003	Byte 3	Byte 2	0002
0005	Byte 5	Byte 4	0004

© 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

All byte loads into any W register are loaded into the LS Byte. The MSB is not modified.

A sign-extend (SE) instruction is provided to allow users to translate 8-bit signed data to 16-bit signed values. Alternatively, for 16-bit unsigned data, users can clear the MSB of any W register by executing a zero-extend (ZE) instruction on the appropriate address.

Although most instructions are capable of operating on word or byte data sizes, it should be noted that some instructions, including the DSP instructions, operate only on words.

3.2.5 NEAR DATA SPACE

An 8 Kbyte 'near' data space is reserved in X address memory space between 0x0000 and 0x1FFF, which is directly addressable via a 13-bit absolute address field within all memory direct instructions. The remaining X address space and all of the Y address space is addressable indirectly. Additionally, the whole of X data space is addressable using MOV instructions, which support memory direct addressing with a 16-bit address field.

3.2.6 SOFTWARE STACK

The dsPIC device contains a software stack. W15 is used as the Stack Pointer.

The stack pointer always points to the first available free word and grows from lower addresses towards higher addresses. It pre-decrements for stack pops and post-increments for stack pushes, as shown in Figure 3-9. Note that for a PC push during any CALL instruction, the MSB of the PC is zero-extended before the push, ensuring that the MSB is always clear.

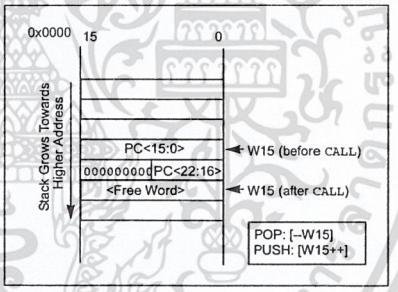
Note:	A PC push during exception processing will concatenate the SRL register to the MSB of the PC prior to the push.

There is a Stack Pointer Limit register (SPLIM) associated with the stack pointer. SPLIM is uninitialized at Reset. As is the case for the stack pointer, SPLIM<0> is forced to '0', because all stack operations must be word aligned. Whenever an effective address (EA) is generated using W15 as a source or destination pointer, the address thus generated is compared with the value in SPLIM. If the contents of the Stack Pointer (W15) and the SPLIM register are equal and a push operation is performed, a Stack Error Trap will not occur. The Stack Error Trap will occur on a subsequent push operation. Thus, for example, if it is desirable to cause a Stack Error Trap when the stack grows beyond address 0x2000 in RAM, initialize the SPLIM with the value, 0x1FFE.

Similarly, a Stack Pointer Underflow (Stack Error) trap is generated when the stack pointer address is found to be less than 0x0800, thus preventing the stack from interfering with the Special Function Register (SFR) space.

A write to the SPLIM register should not be immediately followed by an indirect read operation using W15.

FIGURE 3-9: CALL STACK FRAME



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 3-3: CORE REGISTER MAP

SFR Name	Address (Home)	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
W0	0000	W0 / WREG																0000 0000 0000 0000
W1	0002	W1																0000 0000 0000 0000
W2	0004	W2																0000 0000 0000 0000
W3	0006	W3																0000 0000 0000 0000
W4	0008	W4																0000 0000 0000 0000
W5	000A	W5																0000 0000 0000 0000
W6	000C	W6																0000 0000 0000 0000
W7	000E	W7																0000 0000 0000 0000
W8	0010	W8																0000 0000 0000 0000
W9	0012	W9																0000 0000 0000 0000
W10	0014	W10																0000 0000 0000 0000
W11	0016	W11																0000 0000 0000 0000
W12	0018	W12																0000 0000 0000 0000
W13	001A	W13																0000 0000 0000 0000
W14	001C	W14																0000 0000 0000 0000
W15	001E	W15																0000 1000 0000 0000
SPLIM	0020	SPLIM																0000 0000 0000 0000
ACCAL	0022	ACCAL																0000 0000 0000 0000
ACCAH	0024	ACCAH																0000 0000 0000 0000
ACCAU	0026	Sign-Extension (ACCA<39>)									ACCAU							0000 0000 0000 0000
ACCBL	0028	ACCBL																0000 0000 0000 0000
ACCBH	002A	ACCBH																0000 0000 0000 0000
ACCBU	002C	Sign-Extension (ACCB<39>)									ACCBU							0000 0000 0000 0000
PCL	002E	PCL																0000 0000 0000 0000
PCH	0030	—	—	—	—	—	—	—	—	—	PCH							0000 0000 0000 0000
TBLPAG	0032	—	—	—	—	—	—	—	—	TBLPAG								0000 0000 0000 0000
PSVPAG	0034	—	—	—	—	—	—	—	—	PSVPAG								0000 0000 0000 0000
RCOUNT	0036	RCOUNT																uuuu uuuu uuuu uuuu
DCOUNT	0038	DCOUNT																uuuu uuuu uuuu uuuu
DOSTARTL	003A	DOSTARTL															0	uuuu uuuu uuuu uuu0
DOSTARTH	003C	—	—	—	—	—	—	—	—	—	DOSTARTH							0000 0000 0uuu uuuu
DOENDL	003E	DOENDL															0	uuuu uuuu uuuu uuu0
DOENDH	0040	—	—	—	—	—	—	—	—	—	DOENDH							0000 0000 0uuu uuuu
SR	0042	OA	OB	SA	SB	OAB	SAB	DA	DC	IPL2	IPL1	IPL0	RA	N	OV	Z	C	0000 0000 0000 0000
CORCON	0044	—	—	—	US	EDT	DL2	DL1	DL0	SATA	SATB	SATDW	ACCSAT	IPL3	PSV	RND	IF	0000 0000 0010 0000
MODCON	0046	XMODEN	YMODEN	—	—	BWM<3:0>				YWM<3:0>				XWM<3:0>				0000 0000 0000 0000

Legend: u = uninitialized bit

Preliminary

© 2005 Microchip Technology Inc.

TABLE 3-3: CORE REGISTER MAP (CONTINUED)

SFR Name	Address (Home)	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
XMODSRT	0048	XS<15:1>															0	uuuu uuuu uuuu uuu0
XMODEND	004A	XE<15:1>															1	uuuu uuuu uuuu uuu1
YMODSRT	004C	YS<15:1>															0	uuuu uuuu uuuu uuu0
YMODEND	004E	YE<15:1>															1	uuuu uuuu uuuu uuu1
XBREV	0050	BREN	XB<14:0>															uuuu uuuu uuuu uuuu
DISICNT	0052	—	—	DISICNT<13:0>														0000 0000 0000 0000

Legend: u = uninitialized bit

Note: Refer to *dsPIC30F Family Reference Manual* (DS70046) for descriptions of register bit fields.

© 2005 Microchip Technology Inc.

NOTES:

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.0 ADDRESS GENERATOR UNITS

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

The dsPIC core contains two independent address generator units: the X AGU and Y AGU. The Y AGU supports word sized data reads for the DSP MAC class of instructions only. The dsPIC AGUs support three types of data addressing:

- Linear Addressing
- Modulo (Circular) Addressing
- Bit-Reversed Addressing

Linear and Modulo Data Addressing modes can be applied to data space or program space. Bit-Reversed addressing is only applicable to data space addresses.

4.1 Instruction Addressing Modes

The addressing modes in Table 4-1 form the basis of the addressing modes optimized to support the specific features of individual instructions. The addressing modes provided in the MAC class of instructions are somewhat different from those in the other instruction types.

4.1.1 FILE REGISTER INSTRUCTIONS

Most file register instructions use a 13-bit address field (f) to directly address data present in the first 8192 bytes of data memory (near data space). Most file register instructions employ a working register W0, which is denoted as WREG in these instructions. The destination is typically either the same file register, or WREG (with the exception of the MUL instruction), which writes the result to a register or register pair. The MOV instruction allows additional flexibility and can access the entire data space during file register operation.

4.1.2 MCU INSTRUCTIONS

The three-operand MCU instructions are of the form:

Operand 3 = Operand 1 <function> Operand 2

where Operand 1 is always a working register (i.e., the addressing mode can only be register direct), which is referred to as Wb. Operand 2 can be a W register, fetched from data memory, or a 5-bit literal. The result location can be either a W register or an address location. The following addressing modes are supported by MCU instructions:

- Register Direct
- Register Indirect
- Register Indirect Post-modified
- Register Indirect Pre-modified
- 5-bit or 10-bit Literal

Note: Not all instructions support all the addressing modes given above. Individual instructions may support different subsets of these addressing modes.

TABLE 4-1: FUNDAMENTAL ADDRESSING MODES SUPPORTED

Addressing Mode	Description
File Register Direct	The address of the file register is specified explicitly.
Register Direct	The contents of a register are accessed directly.
Register Indirect	The contents of Wn forms the EA.
Register Indirect Post-modified	The contents of Wn forms the EA. Wn is post-modified (incremented or decremented) by a constant value.
Register Indirect Pre-modified	Wn is pre-modified (incremented or decremented) by a signed constant value to form the EA.
Register Indirect with Register Offset	The sum of Wn and Wb forms the EA.
Register Indirect with Literal Offset	The sum of Wn and a literal forms the EA.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.1.3 MOVE AND ACCUMULATOR INSTRUCTIONS

Move instructions and the DSP Accumulator class of instructions provide a greater degree of addressing flexibility than other instructions. In addition to the addressing modes supported by most MCU instructions, Move and Accumulator instructions also support Register Indirect with Register Offset Addressing mode, also referred to as Register Indexed mode.

Note: For the MOV instructions, the addressing mode specified in the instruction can differ for the source and destination EA. However, the 4-bit Wb (Register Offset) field is shared between both source and destination (but typically only used by one).

In summary, the following addressing modes are supported by Move and Accumulator instructions:

- Register Direct
- Register Indirect
- Register Indirect Post-modified
- Register Indirect Pre-modified
- Register Indirect with Register Offset (Indexed)
- Register Indirect with Literal Offset
- 8-bit Literal
- 16-bit Literal

Note: Not all instructions support all the addressing modes given above. Individual instructions may support different subsets of these addressing modes.

4.1.4 MAC INSTRUCTIONS

The dual source operand DSP instructions (CLR, ED, EDAC, MAC, MPY, MPY.N, MOVSAC and MSC), also referred to as MAC instructions, utilize a simplified set of addressing modes to allow the user to effectively manipulate the data pointers through register indirect tables.

The two source operand pre-fetch registers must be a member of the set {W8, W9, W10, W11}. For data reads, W8 and W9 will always be directed to the X RAGU and W10 and W11 will always be directed to the Y AGU. The effective addresses generated (before and after modification) must, therefore, be valid addresses within X data space for W8 and W9 and Y data space for W10 and W11.

Note: Register Indirect with Register Offset Addressing is only available for W9 (in X space) and W11 (in Y space).

In summary, the following addressing modes are supported by the MAC class of instructions:

- Register Indirect
- Register Indirect Post-modified by 2
- Register Indirect Post-modified by 4
- Register Indirect Post-modified by 6
- Register Indirect with Register Offset (Indexed)

4.1.5 OTHER INSTRUCTIONS

Besides the various addressing modes outlined above, some instructions use literal constants of various sizes. For example, BRA (branch) instructions use 16-bit signed literals to specify the branch destination directly, whereas the DISI instruction uses a 14-bit unsigned literal field. In some instructions, such as ADD Acc, the source of an operand or result is implied by the opcode itself. Certain operations, such as NOP, do not have any operands.

4.2 Modulo Addressing

Modulo addressing is a method of providing an automated means to support circular data buffers using hardware. The objective is to remove the need for software to perform data address boundary checks when executing tightly looped code, as is typical in many DSP algorithms.

Modulo addressing can operate in either data or program space (since the data pointer mechanism is essentially the same for both). One circular buffer can be supported in each of the X (which also provides the pointers into Program space) and Y data spaces. Modulo addressing can operate on any W register pointer. However, it is not advisable to use W14 or W15 for Modulo addressing, since these two registers are used as the Stack Frame Pointer and Stack Pointer, respectively.

In general, any particular circular buffer can only be configured to operate in one direction, as there are certain restrictions on the buffer start address (for incrementing buffers) or end address (for decrementing buffers) based upon the direction of the buffer.

The only exception to the usage restrictions is for buffers which have a power-of-2 length. As these buffers satisfy the start and end address criteria, they may operate in a Bi-directional mode, (i.e., address boundary checks will be performed on both the lower and upper address boundaries).

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.2.1 START AND END ADDRESS

The Modulo addressing scheme requires that a starting and an end address be specified and loaded into the 16-bit modulo buffer address registers: XMODSRT, XMODEND, YMODSRT and YMODEND (see Table 3-3)..

Note:	Y-space modulo addressing EA calculations assume word-sized data (LS bit of every EA is always clear).

The length of a circular buffer is not directly specified. It is determined by the difference between the corresponding start and end addresses. The maximum possible length of the circular buffer is 32K words (64 Kbytes).

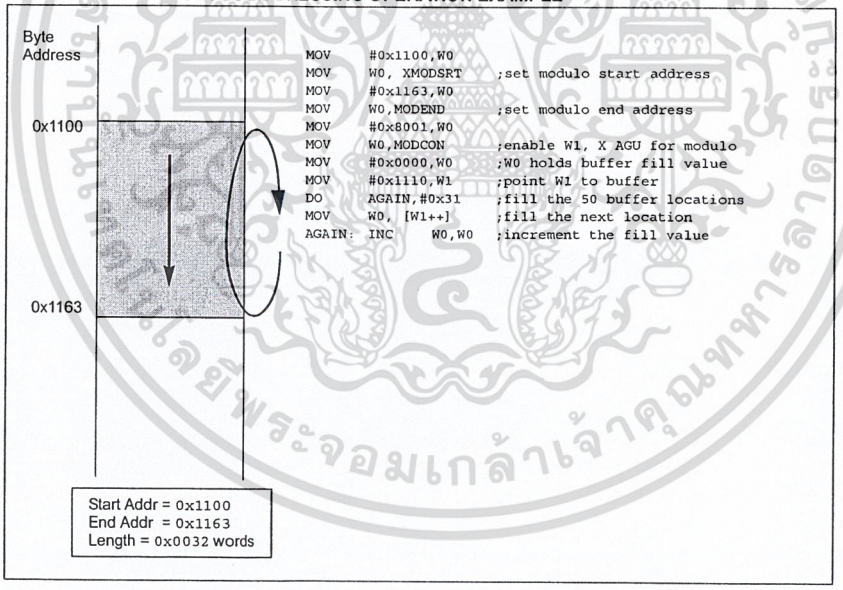
4.2.2 W ADDRESS REGISTER SELECTION

The Modulo and Bit-Reversed Addressing Control register MODCON<15:0> contains enable flags as well as a W register field to specify the W address registers. The XWM and YWM fields select which registers will operate with modulo addressing. If XWM = 15, X RAGU and X WAGU modulo addressing are disabled. Similarly, if YWM = 15, Y AGU modulo addressing is disabled.

The X Address Space Pointer W register (XWM) to which modulo addressing is to be applied, is stored in MODCON<3:0> (see Table 3-3). Modulo addressing is enabled for X data space when XWM is set to any value other than 15 and the XMODEN bit is set at MODCON<15>.

The Y Address Space Pointer W register (YWM) to which modulo addressing is to be applied, is stored in MODCON<7:4>. Modulo addressing is enabled for Y data space when YWM is set to any value other than 15 and the YMODEN bit is set at MODCON<14>.

FIGURE 4-1: MODULO ADDRESSING OPERATION EXAMPLE



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.2.3 MODULO ADDRESSING APPLICABILITY

Modulo addressing can be applied to the effective address (EA) calculation associated with any W register. It is important to realize that the address boundaries check for addresses less than or greater than the upper (for incrementing buffers) and lower (for decrementing buffers) boundary addresses (not just equal to). Address changes may, therefore, jump beyond boundaries and still be adjusted correctly.

Note: The modulo corrected effective address is written back to the register only when Pre-Modify or Post-Modify Addressing mode is used to compute the Effective Address. When an address offset (e.g., [W7+W2]) is used, modulo address correction is performed, but the contents of the register remains unchanged.

4.3 Bit-Reversed Addressing

Bit-Reversed addressing is intended to simplify data reordering for radix-2 FFT algorithms. It is supported by the X AGU for data writes only.

The modifier, which may be a constant value or register contents, is regarded as having its bit order reversed. The address source and destination are kept in normal order. Thus, the only operand requiring reversal is the modifier.

4.3.1 BIT-REVERSED ADDRESSING IMPLEMENTATION

Bit-Reversed addressing is enabled when:

1. BWM (W register selection) in the MODCON register is any value other than 15 (the stack can not be accessed using bit-reversed addressing) **and**
2. the BREN bit is set in the XBREV register **and**
3. the addressing mode used is Register Indirect with Pre-Increment or Post-Increment.

If the length of a bit-reversed buffer is $M = 2^N$ bytes, then the last 'N' bits of the data buffer start address must be zeros.

XB<14:0> is the bit-reversed address modifier or 'pivot point' which is typically a constant. In the case of an FFT computation, its value is equal to half of the FFT data buffer size.

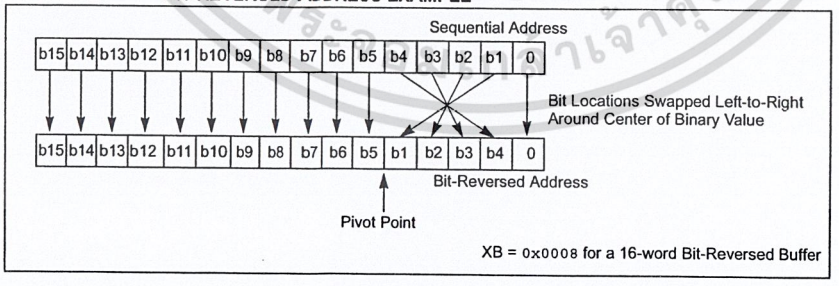
Note: All Bit-Reversed EA calculations assume word sized data (LS bit of every EA is always clear). The XB value is scaled accordingly to generate compatible (byte) addresses.

When enabled, bit-reversed addressing will only be executed for register indirect with pre-increment or post-increment addressing and word sized data writes. It will not function for any other addressing mode or for byte-sized data, and normal addresses will be generated instead. When bit-reversed addressing is active, the W address pointer will always be added to the address modifier (XB) and the offset associated with the Register Indirect Addressing mode will be ignored. In addition, as word sized data is a requirement, the LS bit of the EA is ignored (and always clear).

Note: Modulo addressing and bit-reversed addressing should not be enabled together. In the event that the user attempts to do this, bit reversed addressing will assume priority when active for the X WAGU, and X WAGU modulo addressing will be disabled. However, modulo addressing will continue to function in the X RAGU.

If bit-reversed addressing has already been enabled by setting the BREN (XBREV<15>) bit, then a write to the XBREV register should not be immediately followed by an indirect read operation using the W register that has been designated as the bit-reversed pointer.

FIGURE 4-2: BIT-REVERSED ADDRESS EXAMPLE



 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 4-2: BIT-REVERSED ADDRESS SEQUENCE (16-ENTRY)

Normal Address					Bit-Reversed Address				
A3	A2	A1	A0	Decimal	A3	A2	A1	A0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	8
0	0	1	0	2	0	1	0	0	4
0	0	1	1	3	1	1	0	0	12
0	1	0	0	4	0	0	1	0	2
0	1	0	1	5	1	0	1	0	10
0	1	1	0	6	0	1	1	0	6
0	1	1	1	7	1	1	1	0	14
1	0	0	0	8	0	0	0	1	1
1	0	0	1	9	1	0	0	1	9
1	0	1	0	10	0	1	0	1	5
1	0	1	1	11	1	1	0	1	13
1	1	0	0	12	0	0	1	1	3
1	1	0	1	13	1	0	1	1	11
1	1	1	0	14	0	1	1	1	7
1	1	1	1	15	1	1	1	1	15

TABLE 4-3: BIT-REVERSED ADDRESS MODIFIER VALUES FOR XBREV REGISTER

Buffer Size (Words)	XB<14:0> Bit-Reversed Address Modifier Value*
32768	0x4000
16384	0x2000
8192	0x1000
4096	0x0800
2048	0x0400
1024	0x0200
512	0x0100
256	0x0080
128	0x0040
64	0x0020
32	0x0010
16	0x0008
8	0x0004
4	0x0002
2	0x0001

*Modifier values for buffer sizes greater than 1024 words will exceed the available data memory on the dsPIC30F4011/4012 devices.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

NOTES:

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.0 INTERRUPTS

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

The dsPIC30F4011/4012 has 30 interrupt sources and 4 processor exceptions (traps), which must be arbitrated based on a priority scheme.

The CPU is responsible for reading the Interrupt Vector Table (IVT) and transferring the address contained in the interrupt vector to the program counter. The interrupt vector is transferred from the program data bus into the program counter, via a 24-bit wide multiplexer on the input of the program counter.

The Interrupt Vector Table (IVT) and Alternate Interrupt Vector Table (AIVT) are placed near the beginning of program memory (0x000004). The IVT and AIVT are shown in Figure 5-1.

The interrupt controller is responsible for preprocessing the interrupts and processor exceptions, prior to their being presented to the processor core. The peripheral interrupts and traps are enabled, prioritized and controlled using centralized special function registers:

- IFS0<15:0>, IFS1<15:0>, IFS2<15:0>
  All interrupt request flags are maintained in these three registers. The flags are set by their respective peripherals or external signals, and they are cleared via software.
- IEC0<15:0>, IEC1<15:0>, IEC2<15:0>
  All Interrupt Enable Control bits are maintained in these three registers. These control bits are used to individually enable interrupts from the peripherals or external signals.
- IPC0<15:0>... IPC11<7:0>
  The user assignable priority level associated with each of these interrupts is held centrally in these twelve registers.
- IPL<3:0> The current CPU priority level is explicitly stored in the IPL bits. IPL<3> is present in the CORCON register, whereas IPL<2:0> are present in the status register (SR) in the processor core.
- INTCON1<15:0>, INTCON2<15:0>
  Global interrupt control functions are derived from these two registers. INTCON1 contains the control and status flags for the processor exceptions. The INTCON2 register controls the external interrupt request signal behavior and the use of the alternate vector table.

Note: Interrupt Flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding Enable bit. User software should ensure the appropriate Interrupt Flag bits are clear prior to enabling an interrupt.

All interrupt sources can be user assigned to one of 7 priority levels, 1 through 7, via the IPCx registers. Each interrupt source is associated with an interrupt vector, as shown in Table 5-1. Levels 7 and 1 represent the highest and lowest maskable priorities, respectively.

Note: Assigning a priority level of 0 to an interrupt source is equivalent to disabling that interrupt.

If the NSTDIS bit (INTCON1<15>) is set, nesting of interrupts is prevented. Thus, if an interrupt is currently being serviced, processing of a new interrupt is prevented, even if the new interrupt is of higher priority than the one currently being serviced.

Note: The IPL bits become read-only whenever the NSTDIS bit has been set to '1'.

Certain interrupts have specialized control bits for features like edge or level triggered interrupts, interrupt-on-change, etc. Control of these features remains within the peripheral module which generates the interrupt.

The `DISI` instruction can be used to disable the processing of interrupts of priorities 6 and lower for a certain number of instructions, during which the DISI bit (INTCON2<14>) remains set.

When an interrupt is serviced, the PC is loaded with the address stored in the vector location in Program Memory that corresponds to the interrupt. There are 63 different vectors within the IVT (refer to Figure 5-2). These vectors are contained in locations 0x000004 through 0x0000FE of program memory (refer to Figure 5-2). These locations contain 24-bit addresses, and in order to preserve robustness, an address error trap will take place should the PC attempt to fetch any of these words during normal execution. This prevents execution of random data as a result of accidentally decrementing a PC into vector space, accidentally mapping a data space address into vector space, or the PC rolling over to 0x000000 after reaching the end of implemented program memory space. Execution of a `GOTO` instruction to this vector space will also generate an address error trap.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.1 Interrupt Priority

The user assignable Interrupt Priority (IP<2:0>) bits for each individual interrupt source are located in the LS 3-bits of each nibble, within the IPCx register(s). Bit 3 of each nibble is not used and is read as a '0'. These bits define the priority level assigned to a particular interrupt by the user.

Note: The user selectable priority levels start at 0, as the lowest priority, and level 7, as the highest priority.

Since more than one interrupt request source may be assigned to a specific user specified priority level, a means is provided to assign priority within a given level. This method is called "Natural Order Priority".

Natural Order Priority is determined by the position of an interrupt in the vector table, and only affects interrupt operation when multiple interrupts with the same user-assigned priority become pending at the same time.

Table 5-1 lists the interrupt numbers and interrupt sources for the dsPIC devices and their associated vector numbers.

Note 1: The natural order priority scheme has 0 as the highest priority and 53 as the lowest priority.

2: The natural order priority number is the same as the INT number.

The ability for the user to assign every interrupt to one of seven priority levels implies that the user can assign a very high overall priority level to an interrupt with a low natural order priority. For example, the PLVD (Low Voltage Detect) can be given a priority of 7. The INT0 (external interrupt 0) may be assigned to priority level 1, thus giving it a very low effective priority.

TABLE 5-1: INTERRUPT VECTOR TABLE

INT Number	Vector Number	Interrupt Source
Highest Natural Order Priority		
0	8	INT0 - External Interrupt 0
1	9	IC1 - Input Capture 1
2	10	OC1 - Output Compare 1
3	11	T1 - Timer 1
4	12	IC2 - Input Capture 2
5	13	OC2 - Output Compare 2
6	14	T2 - Timer 2
7	15	T3 - Timer 3
8	16	SPI1
9	17	U1RX - UART1 Receiver
10	18	U1TX - UART1 Transmitter
11	19	ADC - ADC Convert Done
12	20	NVM - NVM Write Complete
13	21	SI2C - I^2C Slave Interrupt
14	22	MI2C - I^2C Master Interrupt
15	23	Input Change Interrupt
16	24	INT1 - External Interrupt 1
17	25	IC7 - Input Capture 7
18	26	IC8 - Input Capture 8
19	27	OC3 - Output Compare 3
20	28	OC4 - Output Compare 4
21	29	T4 - Timer 4
22	30	T5 - Timer 5
23	31	INT2 - External Interrupt 2
24	32	U2RX - UART2 Receiver
25	33	U2TX - UART2 Transmitter
26	34	Reserved
27	35	C1 - Combined IRQ for CAN1
28	36	Reserved
29	37	Reserved
30	38	Reserved
31	39	Reserved
32	40	Reserved
33	41	Reserved
34	42	Reserved
35	43	Reserved
36	44	Reserved
37	45	Reserved
38	46	Reserved
39	47	PWM - PWM Period Match
40	48	QEI - QEI Interrupt
41	49	Reserved
42	50	Reserved
43	51	FLTA - PWM Fault A
44	52	Reserved
45-53	53-61	Reserved
Lowest Natural Order Priority		

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2 Reset Sequence

A Reset is not a true exception, because the interrupt controller is not involved in the Reset process. The processor initializes its registers in response to a Reset, which forces the PC to zero. The processor then begins program execution at location 0x000000. A GOTO instruction is stored in the first program memory location, immediately followed by the address target for the GOTO instruction. The processor executes the GOTO to the specified address and then begins operation at the specified target (start) address.

5.2.1 RESET SOURCES

There are 6 sources of error which will cause a device reset.

- Watchdog Time-out:
  The watchdog has timed out, indicating that the processor is no longer executing the correct flow of code.
- Uninitialized W Register Trap:
  An attempt to use an uninitialized W register as an address pointer will cause a Reset.
- Illegal Instruction Trap:
  Attempted execution of any unused opcodes will result in an illegal instruction trap. Note that a fetch of an illegal instruction does not result in an illegal instruction trap if that instruction is flushed prior to execution due to a flow change.
- Brown-out Reset (BOR):
  A momentary dip in the power supply to the device has been detected, which may result in malfunction.
- Trap Lockout:
  Occurrence of multiple Trap conditions simultaneously will cause a Reset.

5.3 Traps

Traps can be considered as non-maskable interrupts indicating a software or hardware error, which adhere to a predefined priority as shown in Figure 5-1. They are intended to provide the user a means to correct erroneous operation during debug and when operating within the application.

Note: If the user does not intend to take corrective action in the event of a trap error condition, these vectors must be loaded with the address of a default handler that simply contains the RESET instruction. If, on the other hand, one of the vectors containing an invalid address is called, an address error trap is generated.

Note that many of these trap conditions can only be detected when they occur. Consequently, the questionable instruction is allowed to complete prior to trap exception processing. If the user chooses to recover from the error, the result of the erroneous action that caused the trap may have to be corrected.

There are 8 fixed priority levels for traps: Level 8 through Level 15, which implies that the IPL3 is always set during processing of a trap.

If the user is not currently executing a trap, and he sets the IPL<3:0> bits to a value of '0111' (Level 7), then all interrupts are disabled, but traps can still be processed.

5.3.1 TRAP SOURCES

The following traps are provided with increasing priority. However, since all traps can be nested, priority has little effect.

Math Error Trap:

The Math Error trap executes under the following three circumstances:

1. Should an attempt be made to divide by zero, the divide operation will be aborted on a cycle boundary and the trap taken.
2. If enabled, a Math Error trap will be taken when an arithmetic operation on either accumulator A or B causes an overflow from bit 31 and the Accumulator Guard bits are not utilized.
3. If enabled, a Math Error trap will be taken when an arithmetic operation on either accumulator A or B causes a catastrophic overflow from bit 39 and all saturation is disabled.
4. If the shift amount specified in a shift instruction is greater than the maximum allowed shift amount, a trap will occur.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Address Error Trap:

This trap is initiated when any of the following circumstances occurs:

1. A misaligned data word access is attempted.
2. A data fetch from our unimplemented data memory location is attempted.
3. A data access of an unimplemented program memory location is attempted.
4. An instruction fetch from vector space is attempted.

Note:	In the MAC class of instructions, wherein the data space is split into X and Y data space, unimplemented X space includes all of Y space, and unimplemented Y space includes all of X space.

5. Execution of a "BRA #literal" instruction or a "GOTO #literal" instruction, where literal is an unimplemented program memory address.
6. Executing instructions after modifying the PC to point to unimplemented program memory addresses. The PC may be modified by loading a value into the stack and executing a RETURN instruction.

Stack Error Trap:

This trap is initiated under the following conditions:

1. The stack pointer is loaded with a value which is greater than the (user programmable) limit value written into the SPLIM register (stack overflow).
2. The stack pointer is loaded with a value which is less than 0x0800 (simple stack underflow).

Oscillator Fail Trap:

This trap is initiated if the external oscillator fails and operation becomes reliant on an internal RC backup.

5.3.2 HARD AND SOFT TRAPS

It is possible that multiple traps can become active within the same cycle (e.g., a misaligned word stack write to an overflowed address). In such a case, the fixed priority shown in Figure 5-2 is implemented, which may require the user to check if other traps are pending, in order to completely correct the fault.

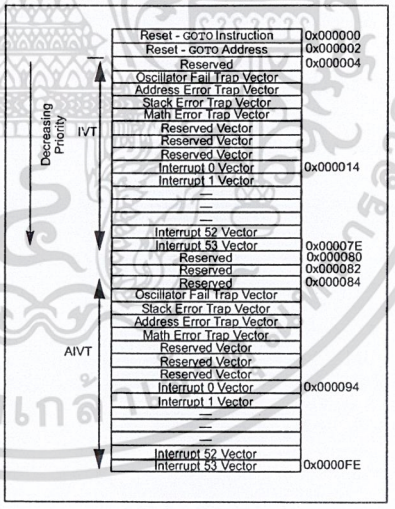
'Soft' traps include exceptions of priority level 8 through level 11, inclusive. The arithmetic error trap (level 11) falls into this category of traps.

'Hard' traps include exceptions of priority level 12 through level 15, inclusive. The address error (level 12), stack error (level 13) and oscillator error (level 14) traps fall into this category.

Each hard trap that occurs must be acknowledged before code execution of any type may continue. If a lower priority hard trap occurs while a higher priority trap is pending, acknowledged, or is being processed, a hard trap conflict will occur.

The device is automatically Reset in a hard trap conflict condition. The TRAPR status bit (RCON<15>) is set when the Reset occurs, so that the condition may be detected in software.

FIGURE 5-1: TRAP VECTORS



 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

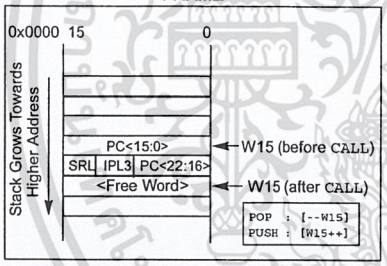
5.4 Interrupt Sequence

All interrupt event flags are sampled in the beginning of each instruction cycle by the IFSx registers. A pending interrupt request (IRQ) is indicated by the flag bit being equal to a '1' in an IFSx register. The IRQ will cause an interrupt to occur if the corresponding bit in the interrupt enable (IECx) register is set. For the remainder of the instruction cycle, the priorities of all pending interrupt requests are evaluated.

If there is a pending IRQ with a priority level greater than the current processor priority level in the IPL bits, the processor will be interrupted.

The processor then stacks the current program counter and the low byte of the processor status register (SRL), as shown in Figure 5-2. The low byte of the status register contains the processor priority level at the time, prior to the beginning of the interrupt cycle. The processor then loads the priority level for this interrupt into the status register. This action will disable all lower priority interrupts until the completion of the Interrupt Service Routine.

FIGURE 5-2: INTERRUPT STACK FRAME



Note 1: The user can always lower the priority level by writing a new value into SR. The Interrupt Service Routine must clear the interrupt flag bits in the IFSx register before lowering the processor interrupt priority, in order to avoid recursive interrupts.

2: The IPL3 bit (CORCON<3>) is always clear when interrupts are being processed. It is set only during execution of traps.

The RETFIE (Return from Interrupt) instruction will unstack the program counter and status registers to return the processor to its state prior to the interrupt sequence.

5.5 Alternate Vector Table

In Program Memory, the Interrupt Vector Table (IVT) is followed by the Alternate Interrupt Vector Table (AIVT), as shown in Figure 5-1. Access to the Alternate Vector Table is provided by the ALTIVT bit in the INTCON2 register. If the ALTIVT bit is set, all interrupt and exception processes will use the alternate vectors instead of the default vectors. The alternate vectors are organized in the same manner as the default vectors. The AIVT supports emulation and debugging efforts by providing a means to switch between an application and a support environment, without requiring the interrupt vectors to be reprogrammed. This feature also enables switching between applications for evaluation of different software algorithms at run time.

If the AIVT is not required, the program memory allocated to the AIVT may be used for other purposes. AIVT is not a protected section and may be freely programmed by the user.

5.6 Fast Context Saving

A context saving option is available using shadow registers. Shadow registers are provided for the DC, N, OV, Z and C bits in SR, and the registers W0 through W3. The shadows are only one level deep. The shadow registers are accessible using the PUSH.S and POP.S instructions only.

When the processor vectors to an interrupt, the PUSH.S instruction can be used to store the current value of the aforementioned registers into their respective shadow registers.

If an ISR of a certain priority uses the PUSH.S and POP.S instructions for fast context saving, then a higher priority ISR should not include the same instructions. Users must save the key registers in software during a lower priority interrupt, if the higher priority ISR uses fast context saving.

5.7 External Interrupt Requests

The interrupt controller supports five external interrupt request signals, INT0-INT4. These inputs are edge sensitive; they require a low-to-high or a high-to-low transition to generate an interrupt request. The INTCON2 register has five bits, INT0EP-INT4EP, that select the polarity of the edge detection circuitry.

5.8 Wake-up from Sleep and Idle

The interrupt controller may be used to wake up the processor from either Sleep or Idle modes, if Sleep or Idle mode is active when the interrupt is generated.

If an enabled interrupt request of sufficient priority is received by the interrupt controller, then the standard interrupt request is presented to the processor. At the same time, the processor will wake-up from Sleep or Idle and begin execution of the Interrupt Service Routine (ISR) needed to process the interrupt request.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 5-2: INTERRUPT CONTROLLER REGISTER MAP

SFR Name	ADR	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
INTCON1	0080	NSTDIS	—	—	—	—	OVATE	OVBTE	COVTE	—	—	—	MATHERR	ADDRERR	STKERR	OSCFAIL	—	0000 0000 0000 0000
INTCON2	0082	ALTIVT	—	—	—	—	—	—	—	—	—	—	—	—	INT2EP	INT1EP	INT0EP	0000 0000 0000 0000
IFS0	0084	CNIF	MI2CIF	SI2CIF	NVMIF	ADIF	U1TXIF	U1RXIF	SPI1IF	T3IF	T2IF	OC2IF	IC2IF	T1IF	OC1IF	IC1IF	INT0IF	0000 0000 0000 0000
IFS1	0086	—	—	—	—	C1IF	—	U2TXIF	U2RXIF	INT2IF	T5IF	T4IF	OC4IF	OC3IF	IC8IF	IC7IF	INT1IF	0000 0000 0000 0000
IFS2	0088	—	—	—	—	FLTAIF	—	—	QEIIF	PWMIF	—	—	—	—	—	—	—	0000 0000 0000 0000
IEC0	008C	CNIE	MI2CIE	SI2CIE	NVMIE	ADIE	U1TXIE	U1RXIE	SPI1IE	T3IE	T2IE	OC2IE	IC2IE	T1IE	OC1IE	IC1IE	INT0IE	0000 0000 0000 0000
IEC1	008E	—	—	—	—	C1IE	—	U2TXIE	U2RXIE	INT2IE	T5IE	T4IE	OC4IE	OC3IE	IC8IE	IC7IE	INT1IE	0000 0000 0000 0000
IEC2	0090	—	—	—	—	FLTAIE	—	—	QEIIE	PWMIE	—	—	—	—	—	—	—	0000 0000 0000 0000
IPC0	0094	—	T1IP<2:0>			—	OC1IP<2:0>			—	IC1IP<2:0>			—	INT0IP<2:0>			0100 0100 0100 0100
IPC1	0096	—	T31P<2:0>			—	T2IP<2:0>			—	OC2IP<2:0>			—	IC2IP<2:0>			0100 0100 0100 0100
IPC2	0098	—	ADIP<2:0>			—	U1TXIP<2:0>			—	U1RXIP<2:0>			—	SPI1IP<2:0>			0100 0100 0100 0100
IPC3	009A	—	CNIP<2:0>			—	MI2CIP<2:0>			—	SI2CIP<2:0>			—	NVMIP<2:0>			0100 0100 0100 0100
IPC4	009C	—	OC3IP<2:0>			—	IC8IP<2:0>			—	IC7IP<2:0>			—	INT1IP<2:0>			0100 0100 0100 0100
IPC5	009E	—	INT2IP<2:0>			—	T5IP<2:0>			—	T4IP<2:0>			—	OC4IP<2:0>			0100 0100 0100 0100
IPC6	00A0	—	C1IP<2:0>			—	—	—	—	—	U2TXIP<2:0>			—	U2RXIP<2:0>			0100 0000 0100 0100
IPC7	00A2	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
IPC8	00A4	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
IPC9	00A6	—	PWMIP<2:0>			—	—	—	—	—	—	—	—	—	—	—	—	0100 0000 0100 0100
IPC10	00A8	—	FLTAIP<2:0>			—	—	—	—	—	—	—	—	—	QEIIP<2:0>			0100 0000 0000 0100
IPC11	00AA	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000

Legend: u = uninitialized bit

Note: Refer to *dsPIC30F Family Reference Manual* (DS70046) for descriptions of register bit fields.

© 2005 Microchip Technology Inc.

6.0 FLASH PROGRAM MEMORY

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

The dsPIC30F family of devices contains internal program Flash memory for executing user code. There are two methods by which the user can program this memory:

1. In-Circuit Serial Programming™ (ICSP™)
2. Run Time Self-Programming (RTSP)

6.1 In-Circuit Serial Programming (ICSP)

dsPIC30F devices can be serially programmed while in the end application circuit. This is simply done with two lines for Programming Clock and Programming Data (which are named PGC and PGD respectively), and three other lines for Power (VDD), Ground (VSS) and Master Clear (MCLR). this allows customers to manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product. This also allows the most recent firmware or a custom firmware to be programmed.

6.2 Run Time Self-Programming (RTSP)

RTSP is accomplished using TBLRD (table read) and TBLWT (table write) instructions.

With RTSP, the user may erase program memory, 32 instructions (96 bytes) at a time and can write program memory data, 32 instructions (96 bytes) at a time.

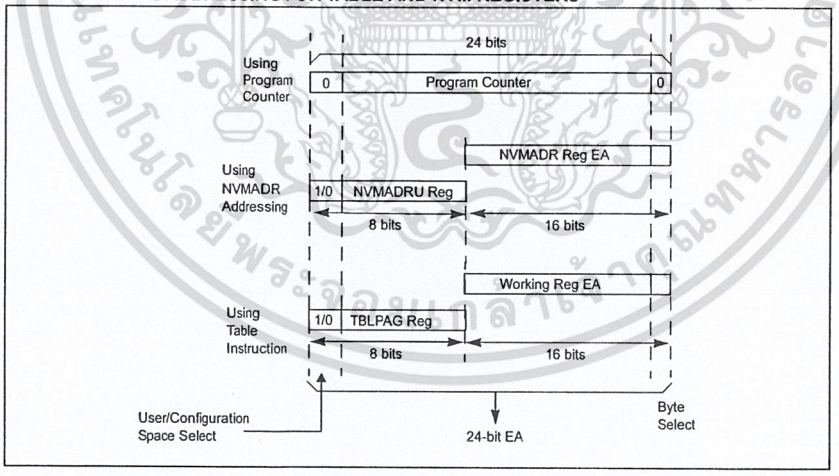
6.3 Table Instruction Operation Summary

The TBLRDL and the TBLWTL instructions are used to read or write to bits <15:0> of program memory. TBLRDL and TBLWTL can access program memory in Word or Byte mode.

The TBLRDH and TBLWTH instructions are used to read or write to bits<23:16> of program memory. TBLRDH and TBLWTH can access program memory in Word or Byte mode.

A 24-bit program memory address is formed using bits<7:0> of the TBLPAG register and the effective address (EA) from a W register specified in the table instruction, as shown in Figure 6-1.

FIGURE 6-1: ADDRESSING FOR TABLE AND NVM REGISTERS



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.4 RTSP Operation

The dsPIC30F Flash program memory is organized into rows and panels. Each row consists of 32 instructions, or 96 bytes. Each panel consists of 128 rows, or 4K x 24 instructions. RTSP allows the user to erase one row (32 instructions) at a time and to program 32 instructions at one time.

Each panel of program memory contains write latches that hold 32 instructions of programming data. Prior to the actual programming operation, the write data must be loaded into the panel write latches. The data to be programmed into the panel is loaded in sequential order into the write latches; instruction 0, instruction 1, etc. The addresses loaded must always be from an even group of 32 boundary.

The basic sequence for RTSP programming is to set up a table pointer, then do a series of TBLWT instructions to load the write latches. Programming is performed by setting the special bits in the NVMCON register. 32 TBLWTL and 32 TBLWTH instructions are required to load the 32 instructions.

All of the table write operations are single word writes (2 instruction cycles), because only the table latches are written.

After the latches are written, a programming operation needs to be initiated to program the data.

The Flash Program Memory is readable, writable and erasable during normal operation over the entire VDD range.

6.5 RTSP Control Registers

The four SFRs used to read and write the program Flash memory are:

- NVMCON
- NVMADR
- NVMADRU
- NVMKEY

6.5.1 NVMCON REGISTER

The NVMCON register controls which blocks are to be erased, which memory type is to be programmed, and start of the programming cycle.

6.5.2 NVMADR REGISTER

The NVMADR register is used to hold the lower two bytes of the effective address. The NVMADR register captures the EA<15:0> of the last table instruction that has been executed and selects the row to write.

6.5.3 NVMADRU REGISTER

The NVMADRU register is used to hold the upper byte of the effective address. The NVMADRU register captures the EA<23:16> of the last table instruction that has been executed.

6.5.4 NVMKEY REGISTER

NVMKEY is a write-only register that is used for write protection. To start a programming or an erase sequence, the user must consecutively write 0x55 and 0xAA to the NVMKEY register. Refer to Section 6.6 for further details.

Note:	The user can also directly write to the NVMADR and NVMADRU registers to specify a program memory address for erasing or programming.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.6 Programming Operations

A complete programming sequence is necessary for programming or erasing the internal Flash in RTSP mode. A programming operation is nominally 2 msec in duration and the processor stalls (waits) until the operation is finished. Setting the WR bit (NVMCON<15>) starts the operation, and the WR bit is automatically cleared when the operation is finished.

6.6.1 PROGRAMMING ALGORITHM FOR PROGRAM FLASH

The user can erase or program one row of program Flash memory at a time. The general process is:

1. Read one row of program Flash (32 instruction words) and store into data RAM as a data "image".
2. Update the data image with the desired new data.
3. Erase program Flash row.
   a) Setup NVMCON register for multi-word, program Flash, erase, and set WREN bit.
   b) Write address of row to be erased into NVMADRU/NVMDR.
   c) Write '55' to NVMKEY.
   d) Write 'AA' to NVMKEY.
   e) Set the WR bit. This will begin erase cycle.
   f) CPU will stall for the duration of the erase cycle.
   g) The WR bit is cleared when erase cycle ends.
4. Write 32 instruction words of data from data RAM "image" into the program Flash write latches.
5. Program 32 instruction words into program Flash.
   a) Setup NVMCON register for multi-word, program Flash, program, and set WREN bit.
   b) Write '55' to NVMKEY.
   c) Write 'AA' to NVMKEY.
   d) Set the WR bit. This will begin program cycle.
   e) CPU will stall for duration of the program cycle.
   f) The WR bit is cleared by the hardware when program cycle ends.
6. Repeat steps 1 through 5 as needed to program desired amount of program Flash memory.

6.6.2 ERASING A ROW OF PROGRAM MEMORY

Example 6-1 shows a code sequence that can be used to erase a row (32 instructions) of program memory.

EXAMPLE 6-1: ERASING A ROW OF PROGRAM MEMORY

```
; Setup NVMCON for erase operation, multi word write
; program memory selected, and writes enabled
        MOV     #0x4041,W0                  ;
        MOV     W0,NVMCON                   ; Init NVMCON SFR
; Init pointer to row to be ERASED
        MOV     #tblpage(PROG_ADDR),W0      ;
        MOV     W0,NVMADRU                  ; Initialize PM Page Boundary SFR
        MOV     #tbloffset(PROG_ADDR),W0    ; Intialize in-page EA[15:0] pointer
        MOV     W0, NVMADR                  ; Intialize NVMADR SFR
        DISI    #5                          ; Block all interrupts with priority <7
                                            ; for next 5 instructions
        MOV     #0x55,W0                    ;
        MOV     W0,NVMKEY                   ; Write the 0x55 key
        MOV     #0xAA,W1                    ;
        MOV     W1,NVMKEY                   ; Write the 0xAA key
        BSET    NVMCON,#WR                  ; Start the erase sequence
        NOP                                 ; Insert two NOPs after the erase
        NOP                                 ; command is asserted
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.6.3 LOADING WRITE LATCHES

Example 6-2 shows a sequence of instructions that can be used to load the 96 bytes of write latches. 32 TBLWTL and 32 TBLWTH instructions are needed to load the write latches selected by the table pointer.

EXAMPLE 6-2: LOADING WRITE LATCHES

```
; Set up a pointer to the first program memory location to be written
; program memory selected, and writes enabled
        MOV     #0x0000,W0                      ;
        MOV     W0,TBLPAG                       ; Initialize PM Page Boundary SFR
        MOV     #0x6000,W0                      ; An example program memory address
; Perform the TBLWT instructions to write the latches
; 0th_program_word
        MOV     #LOW_WORD_0,W2                  ;
        MOV     #HIGH_BYTE_0,W3                 ;
        TBLWTL  W2,[W0]                         ; Write PM low word into program latch
        TBLWTH  W3,[W0++]                       ; Write PM high byte into program latch
; 1st_program_word
        MOV     #LOW_WORD_1,W2                  ;
        MOV     #HIGH_BYTE_1,W3                 ;
        TBLWTL  W2,[W0]                         ; Write PM low word into program latch
        TBLWTH  W3,[W0++]                       ; Write PM high byte into program latch
;  2nd_program_word
        MOV     #LOW_WORD_2,W2                  ;
        MOV     #HIGH_BYTE_2,W3                 ;
        TBLWTL  W2,[W0]                         ; Write PM low word into program latch
        TBLWTH  W3,[W0++]                       ; Write PM high byte into program latch
        •
        •
        •
; 31st_program_word
        MOV     #LOW_WORD_31,W2                 ;
        MOV     #HIGH_BYTE_31,W3                ;
        TBLWTL  W2,[W0]                         ; Write PM low word into program latch
        TBLWTH  W3,[W0++]                       ; Write PM high byte into program latch
```

Note: In Example 6-2, the contents of the upper byte of W3 has no effect.

6.6.4 INITIATING THE PROGRAMMING SEQUENCE

For protection, the write initiate sequence for NVMKEY must be used to allow any erase or program operation to proceed. After the programming command has been executed, the user must wait for the programming time until programming is complete. The two instructions following the start of the programming sequence should be NOPs.

EXAMPLE 6-3: INITIATING A PROGRAMMING SEQUENCE

```
        DISI    #5                      ; Block all interrupts with priority <7
                                        ; for next 5 instructions
        MOV     #0x55,W0
        MOV     W0,NVMKEY               ; Write the 0x55 key
        MOV     #0xAA,W1                ;
        MOV     W1,NVMKEY               ; Write the 0xAA key
        BSET    NVMCON,#WR              ; Start the erase sequence
        NOP                             ; Insert two NOPs after the erase
        NOP                             ; command is asserted
```

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 6-1: NVM REGISTER MAP

File Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All RESETS
NVMCON	0760	WR	WREN	WRERR	—	—	—	—	TWRI	—	PROGOP<6:0>							0000 0000 0000 0000
NVMADR	0762	NVMADR<15:0>																uuuu uuuu uuuu uuuu
NVMADRU	0764	—	—	—	—	—	—	—	—	—	NVMADR<22:16>							0000 0000 uuuu uuuu
NVMKEY	0766	—	—	—	—	—	—	—	—		KEY<7:0>							0000 0000 0000 0000

Legend: u = uninitialized bit

Note: Refer to *dsPIC30F Family Reference Manual* (DS70046) for descriptions of register bit fields.

NOTES:

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.0 DATA EEPROM MEMORY

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046). For more information on the device instruction set and programming, refer to the *dsPIC30F Programmer's Reference Manual* (DS70030).

The Data EEPROM Memory is readable and writable during normal operation over the entire VDD range. The data EEPROM memory is directly mapped in the program memory address space.

The four SFRs used to read and write the program Flash memory are used to access data EEPROM memory, as well. As described in Section 4.0, these registers are:

- NVMCON
- NVMADR
- NVMADRU
- NVMKEY

The EEPROM data memory allows read and write of single words and 16-word blocks. When interfacing to data memory, NVMADR, in conjunction with the NVMADRU register, is used to address the EEPROM location being accessed. TBLRDL and TBLWTL instructions are used to read and write data EEPROM. The dsPIC30F4011/4012 device has 1 Kbyte (512 words) of data EEPROM, with an address range from 0x7FFC00 to 0x7FFFFE.

A word write operation should be preceded by an erase of the corresponding memory location(s). The write typically requires 2 ms to complete, but the write time will vary with voltage and temperature.

A program or erase operation on the data EEPROM does not stop the instruction flow. The user is responsible for waiting for the appropriate duration of time before initiating another data EEPROM write/erase operation. Attempting to read the data EEPROM while a programming or erase operation is in progress results in unspecified data.

Control bit WR initiates write operations, similar to program Flash writes. This bit cannot be cleared, only set, in software. This bit is cleared in hardware at the completion of the write operation. The inability to clear the WR bit in software prevents the accidental or premature termination of a write operation.

The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a MCLR Reset, or a WDT Time-out Reset, during normal operation. In these situations, following Reset, the user can check the WRERR bit and rewrite the location. The address register NVMADR remains unchanged.

Note: Interrupt flag bit NVMIF in the IFS0 register is set when write is complete. It must be cleared in software.

7.1 Reading the Data EEPROM

A TBLRD instruction reads a word at the current program word address. This example uses W0 as a pointer to data EEPROM. The result is placed in register W4, as shown in Example 7-1.

EXAMPLE 7-1: DATA EEPROM READ

```
MOV     #LOW_ADDR_WORD,W0   ; Init Pointer
MOV     #HIGH_ADDR_WORD,W1
MOV     W1,TBLPAG
TBLRDL  [ W0 ], W4          ; read data EEPROM
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.2 Erasing Data EEPROM

7.2.1 ERASING A BLOCK OF DATA EEPROM

In order to erase a block of data EEPROM, the NVMADRU and NVMADR registers must initially point to the block of memory to be erased. Configure NVMCON for erasing a block of data EEPROM, and set the ERASE and WREN bits in NVMCON register. Setting the WR bit initiates the erase, as shown in Example 7-2.

EXAMPLE 7-2: DATA EEPROM BLOCK ERASE

```
; Select data EEPROM block, ERASE, WREN bits
    MOV     #0x4045,W0
    MOV     W0,NVMCON                       ; Initialize NVMCON SFR

; Start erase cycle by setting WR after writing key sequence
    DISI    #5                              ; Block all interrupts with priority <7
                                            ; for next 5 instructions
    MOV     #0x55,W0                        ;
    MOV     W0,NVMKEY                       ; Write the 0x55 key
    MOV     #0xAA,W1                        ;
    MOV     W1,NVMKEY                       ; Write the 0xAA key
    BSET    NVMCON,#WR                      ; Initiate erase sequence
    NOP
    NOP
; Erase cycle will complete in 2mS. CPU is not stalled for the Data Erase Cycle
; User can poll WR bit, use NVMIF or Timer IRQ to determine erasure complete
```

7.2.2 ERASING A WORD OF DATA EEPROM

The TBLPAG and NVMADR registers must point to the block. Select erase a block of data Flash, and set the ERASE and WREN bits in NVMCON register. Setting the WR bit initiates the erase, as shown in Example 7-3.

EXAMPLE 7-3: DATA EEPROM WORD ERASE

```
; Select data EEPROM word, ERASE, WREN bits
    MOV     #0x4044,W0
    MOV     W0,NVMCON

; Start erase cycle by setting WR after writing key sequence
    DISI    #5                              ; Block all interrupts with priority <7
                                            ; for next 5 instructions
    MOV     #0x55,W0                        ;
    MOV     W0,NVMKEY                       ; Write the 0x55 key
    MOV     #0xAA,W1                        ;
    MOV     W1,NVMKEY                       ; Write the 0xAA key
    BSET    NVMCON,#WR                      ; Initiate erase sequence
    NOP
    NOP
; Erase cycle will complete in 2mS. CPU is not stalled for the Data Erase Cycle
; User can poll WR bit, use NVMIF or Timer IRQ to determine erasure complete
```

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.3 Writing to the Data EEPROM

To write an EEPROM data location, the following sequence must be followed:

1. Erase data EEPROM word.
   a) Select word, data EEPROM, erase and set WREN bit in NVMCON register.
   b) Write address of word to be erased into NVMADRU/NVMADR.
   c) Enable NVM interrupt (optional).
   d) Write '55' to NVMKEY.
   e) Write 'AA' to NVMKEY.
   f) Set the WR bit. This will begin erase cycle.
   g) Either poll NVMIF bit or wait for NVMIF interrupt.
   h) The WR bit is cleared when the erase cycle ends.
2. Write data word into data EEPROM write latches.
3. Program 1 data word into data EEPROM.
   a) Select word, data EEPROM, program, and set WREN bit in NVMCON register.
   b) Enable NVM write done interrupt (optional).
   c) Write '55' to NVMKEY.
   d) Write 'AA' to NVMKEY.
   e) Set The WR bit. This will begin program cycle.
   f) Either poll NVMIF bit or wait for NVM interrupt.
   g) The WR bit is cleared when the write cycle ends.

The write will not initiate if the above sequence is not exactly followed (write 0x55 to NVMKEY, write 0xAA to NVMCON, then set WR bit) for each word. It is strongly recommended that interrupts be disabled during this code segment.

Additionally, the WREN bit in NVMCON must be set to enable writes. This mechanism prevents accidental writes to data EEPROM, due to unexpected code execution. The WREN bit should be kept clear at all times, except when updating the EEPROM. The WREN bit is not cleared by hardware.

After a write sequence has been initiated, clearing the WREN bit will not affect the current write cycle. The WR bit will be inhibited from being set unless the WREN bit is set. The WREN bit must be set on a previous instruction. Both WR and WREN cannot be set with the same instruction.

At the completion of the write cycle, the WR bit is cleared in hardware and the Non-Volatile Memory Write Complete Interrupt Flag bit (NVMIF) is set. The user may either enable this interrupt, or poll this bit. NVMIF must be cleared by software.

7.3.1 WRITING A WORD OF DATA EEPROM

Once the user has erased the word to be programmed, then a table write instruction is used to write one write latch, as shown in Example 7-4.

EXAMPLE 7-4: DATA EEPROM WORD WRITE

```
; Point to data memory
    MOV             #LOW_ADDR_WORD,W0                ; Init pointer
    MOV             #HIGH_ADDR_WORD,W1
    MOV             W1,TBLPAG
    MOV             #LOW(WORD),W2                    ; Get data
    TBLWTL          W2,[ W0]                         ; Write data
; The NVMADR captures last table access address
; Select data EEPROM for 1 word op
    MOV             #0x4004,W0
    MOV             W0,NVMCON

; Operate key to allow write operation
    DISI    #5                          ; Block all interrupts with priority <7
                                        ; for next 5 instructions
    MOV             #0x55,W0
    MOV             W0,NVMKEY                        ; Write the 0x55 key
    MOV             #0xAA,W1
    MOV             W1,NVMKEY                        ; Write the 0xAA key
    BSET            NVMCON,#WR                       ; Initiate program sequence
    NOP
    NOP
; Write cycle will complete in 2mS. CPU is not stalled for the Data Write Cycle
; User can poll WR bit, use NVMIF or Timer IRQ to determine write complete
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.3.2 WRITING A BLOCK OF DATA EEPROM

To write a block of data EEPROM, write to all sixteen latches first, then set the NVMCON register and program the block.

EXAMPLE 7-5: DATA EEPROM BLOCK WRITE

```
    MOV     #LOW_ADDR_WORD,W0   ; Init pointer
    MOV     #HIGH_ADDR_WORD,W1
    MOV     W1,TBLPAG
    MOV     #data1,W2           ; Get 1st data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data2,W2           ; Get 2nd data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data3,W2           ; Get 3rd data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data4,W2           ; Get 4th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data5,W2           ; Get 5th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data6,W2           ; Get 6th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data7,W2           ; Get 7th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data8,W2           ; Get 8th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data9,W2           ; Get 9th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data10,W2          ; Get 10th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data11,W2          ; Get 11th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data12,W2          ; Get 12th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data13,W2          ; Get 13th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data14,W2          ; Get 14th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data15,W2          ; Get 15th data
    TBLWTL  W2,[W0]++           ; write data
    MOV     #data16,W2          ; Get 16th data
    TBLWTL  W2,[W0]++           ; write data. The NVMADR captures last table access address.
    MOV     #0x400A,W0          ; Select data EEPROM for multi word op
    MOV     W0,NVMCON           ; Operate Key to allow program operation
    DISI    #5                  ; Block all interrupts with priority <7
                                ; for next 5 instructions
    MOV     #0x55,W0
    MOV     W0,NVMKEY           ; Write the 0x55 key
    MOV     #0xAA,W1
    MOV     W1,NVMKEY           ; Write the 0xAA key
    BSET    NVMCON,#WR          ; Start write cycle
    NOP
    NOP
```

7.4 Write Verify

Depending on the application, good programming practice may dictate that the value written to the memory should be verified against the original value. This should be used in applications where excessive writes can stress bits near the specification limit.

7.5 Protection Against Spurious Write

There are conditions when the device may not want to write to the data EEPROM memory. To protect against spurious EEPROM writes, various mechanisms have been built-in. On power-up, the WREN bit is cleared; also, the Power-up Timer prevents EEPROM write.

The write initiate sequence and the WREN bit together, help prevent an accidental write during brown-out, power glitch or software malfunction.

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.0 I/O PORTS

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *dsPIC30F Family Reference Manual* (DS70046).

All of the device pins (except VDD, VSS, MCLR and OSC1/CLKIN) are shared between the peripherals and the parallel I/O ports.

All I/O input ports feature Schmitt Trigger inputs for improved noise immunity.

8.1 Parallel I/O (PIO) Ports

When a peripheral is enabled and the peripheral is actively driving an associated pin, the use of the pin as a general purpose output pin is disabled. The I/O pin may be read, but the output driver for the Parallel Port bit will be disabled. If a peripheral is enabled, but the peripheral is not actively driving a pin, that pin may be driven by a port.

All port pins have three registers directly associated with the operation of the port pin. The data direction register (TRISx) determines whether the pin is an input or an output. If the Data Direction bit is a '1', then the pin is an input. All port pins are defined as inputs after a Reset. Reads from the latch (LATx), read the latch. Writes to the latch, write the latch (LATx). Reads from the port (PORTx), read the port pins, and writes to the port pins, write the latch (LATx).

Any bit and its associated data and control registers that are not valid for a particular device will be disabled. That means the corresponding LATx and TRISx registers and the port pin will read as zeros.

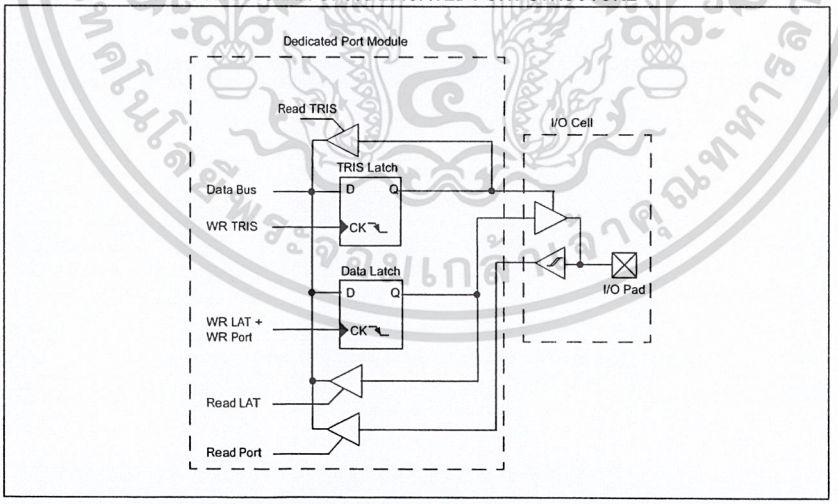
When a pin is shared with another peripheral or function that is defined as an input only, it is nevertheless regarded as a dedicated port because there is no other competing source of outputs. An example is the INT4 pin.

The format of the registers for PORTX are shown in Table 8-1.

The TRISX (Data Direction Control) register controls the direction of the pins. The LATX register supplies data to the outputs, and is readable/writable. Reading the PORTX register yields the state of the input pins, while writing the PORTX register modifies the contents of the LATX register.

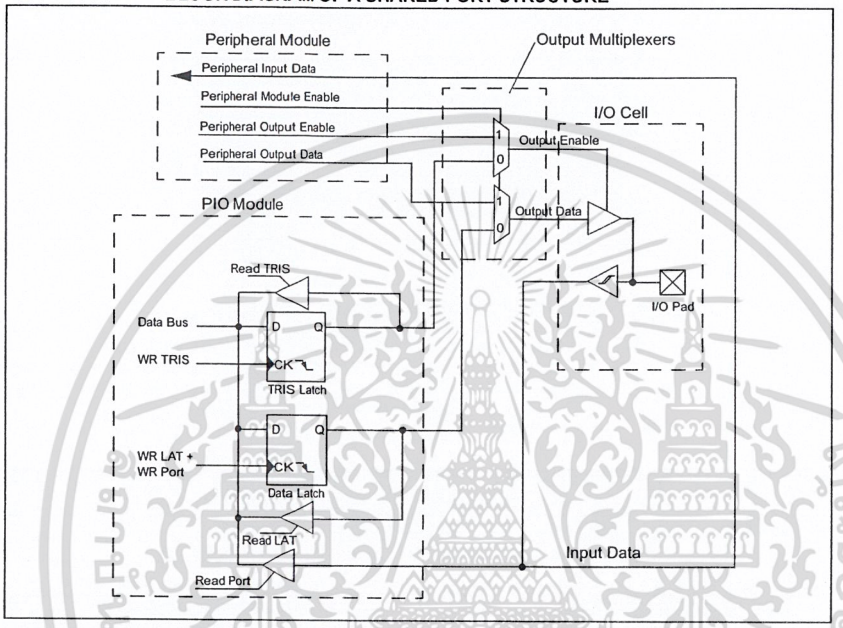
A parallel I/O (PIO) port that shares a pin with a peripheral is, in general, subservient to the peripheral. The peripheral's output buffer data and control signals are provided to a pair of multiplexers. The multiplexers select whether the peripheral or the associated port has ownership of the output data and control signals of the I/O pad cell. Figure 8-2 shows how ports are shared with other peripherals, and the associated I/O cell (pad) to which they are connected. Table 8-1 and shows the formats of the registers for the shared ports, PORTB through PORTG.

FIGURE 8-1: BLOCK DIAGRAM OF A DEDICATED PORT STRUCTURE



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

FIGURE 8-2: BLOCK DIAGRAM OF A SHARED PORT STRUCTURE



8.2 Configuring Analog Port Pins

The use of the ADPCFG and TRIS registers control the operation of the A/D port pins. The port pins that are desired as analog inputs must have their corresponding TRIS bit set (input). If the TRIS bit is cleared (output), the digital output level (V_{OH} or V_{OL}) will be converted.

When reading the PORT register, all pins configured as analog input channel will read as cleared (a low level).

Pins configured as digital inputs will not convert an analog input. Analog levels on any pin that is defined as a digital input (including the ANx pins), may cause the input buffer to consume current that exceeds the device specifications.

8.2.1 I/O PORT WRITE/READ TIMING

One instruction cycle is required between a port direction change or port write operation and a read operation of the same port. Typically this instruction would be a NOP.

EXAMPLE 8-1: PORT WRITE/READ EXAMPLE

```
MOV   0xPF00, W0  ; Configure PORTB<15:8>
                  ; as inputs
MOV   W0, TRISBB  ; and PORTB<7:0> as outputs
NOP               ; Delay 1 cycle
btss  PORTB, #13  ; Next Instruction
```

 © 2005 Microchip Technology Inc.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

TABLE 8-1: dsPIC30F4011 PORT REGISTER MAP

SFR Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
TRISB	02C6	—	—	—	—	—	—	—	TRISB8	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	0000 0001 1111 1111
PORTB	02C8	—	—	—	—	—	—	—	RB8	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	0000 0000 0000 0000
LATB	02CB	—	—	—	—	—	—	—	LATB8	LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0	0000 0000 0000 0000
TRISC	02CC	TRISC15	TRISC14	TRISC13	—	—	—	—	—	—	—	—	—	—	—	—	—	1110 0000 0000 0000
PORTC	02CE	RC15	RC14	RC13	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
LATC	02D0	LATC15	LATC14	LATC13	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
TRISD	02D2	—	—	—	—	—	—	—	—	—	—	—	—	TRISD3	TRISD2	TRISD1	TRISD0	0000 0000 0000 1111
PORTD	02D4	—	—	—	—	—	—	—	—	—	—	—	—	RD3	RD2	RD1	RD0	0000 0000 0000 0000
LATD	02D6	—	—	—	—	—	—	—	—	—	—	—	—	LATD3	LATD2	LATD1	LATD0	0000 0000 0000 0000
TRISE	02D8	—	—	—	—	—	—	—	TRISE8	—	—	TRISE5	TRISE4	TRISE3	TRISE2	TRISE1	TRISE0	0000 0001 0011 1111
PORTE	02DA	—	—	—	—	—	—	—	RE8	—	—	RE5	RE4	RE3	RE2	RE1	RE0	0000 0000 0000 0000
LATE	02DC	—	—	—	—	—	—	—	LATE8	—	—	LATE5	LATE4	LATE3	LATE2	LATE1	LATE0	0000 0000 0000 0000
TRISF	02EE	—	—	—	—	—	—	—	—	—	TRISF6	TRISF5	TRISF4	TRISF3	TRISF2	TRISF1	TRISF0	0000 0000 0111 1111
PORTF	02E0	—	—	—	—	—	—	—	—	—	RF6	RF5	RF4	RF3	RF2	RF1	RF0	0000 0000 0000 0000
LATF	02E2	—	—	—	—	—	—	—	—	—	LATF6	LATF5	LATF4	LATF3	LATF2	LATF1	LATF0	0000 0000 0000 0000

Legend: u = uninitialized bit

Note: Refer to *dsPIC30F Family Reference Manual* (DS70046) for descriptions of register bit fields.

© 2005 Microchip Technology Inc.

TABLE 8-2: dsPIC30F4012 PORT REGISTER MAP

SFR Name	Addr.	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
TRISB	02C6	—	—	—	—	—	—	—	—	—	—	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	0000 0000 0011 1111
PORTB	02C8	—	—	—	—	—	—	—	—	—	—	RB5	RB4	RB3	RB2	RB1	RB0	0000 0000 0000 0000
LATB	02CB	—	—	—	—	—	—	—	—	—	—	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0	0000 0000 0000 0000
TRISC	02CC	TRISC15	TRISC14	TRISC13	—	—	—	—	—	—	—	—	—	—	—	—	—	1110 0000 0000 0000
PORTC	02CE	RC15	RC14	RC13	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
LATC	02D0	LATC15	LATC14	LATC13	—	—	—	—	—	—	—	—	—	—	—	—	—	0000 0000 0000 0000
TRISD	02D2	—	—	—	—	—	—	—	—	—	—	—	—	—	—	TRISD1	TRISD0	0000 0000 0000 0011
PORTD	02D4	—	—	—	—	—	—	—	—	—	—	—	—	—	—	RD1	RD0	0000 0000 0000 0000
LATD	02D6	—	—	—	—	—	—	—	—	—	—	—	—	—	—	LATD1	LATD0	0000 0000 0000 0000
TRISE	02D8	—	—	—	—	—	—	—	TRISE8	—	—	TRISE5	TRISE4	TRISE3	TRISE2	TRISE1	TRISE0	0000 0001 0011 1111
PORTE	02DA	—	—	—	—	—	—	—	RE8	—	—	RE5	RE4	RE3	RE2	RE1	RE0	0000 0000 0000 0000
LATE	02DC	—	—	—	—	—	—	—	LATE8	—	—	LATE5	LATE4	LATE3	LATE2	LATE1	LATE0	0000 0000 0000 0000
TRISF	02EE	—	—	—	—	—	—	—	—	—	—	—	—	TRISF3	TRISF2	—	—	0000 0000 0000 1100
PORTF	02E0	—	—	—	—	—	—	—	—	—	—	—	—	RF3	RF2	—	—	0000 0000 0000 0000
LATF	02E2	—	—	—	—	—	—	—	—	—	—	—	—	LATF3	LATF2	—	—	0000 0000 0000 0000

Legend: u = uninitialized bit

Note: Refer to *dsPIC30F Family Reference Manual* (DS70046) for descriptions of register bit fields.

© 2005 Microchip Technology Inc.

8.3 Input Change Notification Module

The Input Change Notification module provides the dsPIC30F devices the ability to generate interrupt requests to the processor in response to a change-of-state on selected input pins. This module is capable of detecting input change-of-states even in Sleep mode, when the clocks are disabled. There are 10 external signals (CN0 through CN7, CN17 and CN18) that may be selected (enabled) for generating an interrupt request on a change-of-state.

Please refer to the Pin Diagrams for CN pin locations.

TABLE 8-3: INPUT CHANGE NOTIFICATION REGISTER MAP (BITS 7-0)

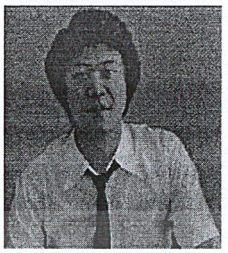
SFR Name	Addr.	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
CNEN1	00C0	CN7IE	CN6IE	CN5IE	CN4IE	CN3IE	CN2IE	CN1IE	CN0IE	0000 0000 0000 0000
CNEN2	00C2	—	—	—	—	—	CN18IE*	CN17IE*	—	0000 0000 0000 0000
CNPU1	00C4	CN7PUE	CN6PUE	CN5PUE	CN4PUE	CN3PUE	CN2PUE	CN1PUE	CN0PUE	0000 0000 0000 0000
CNPU2	00C6	—	—	—	—	—	CN18PUE*	CN17PUE*	—	0000 0000 0000 0000

Legend: u = uninitialized bit
*Not available on dsPIC30F4012

Note: Refer to *dsPIC30F Family Reference Manual* (DS70046) for descriptions of register bit fields.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประวัติผู้จัดทำ



นาย พร้อมพงษ์ ภาชนนท์

เกิดวันที่ 2 เมษายน พ.ศ. 2531

ประวัติการศึกษา : สำเร็จการศึกษาจากโรงเรียนพะเยาพิทยาคม อำเภอเมือง จังหวัดพะเยา

เข้าศึกษาที่คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมแมคคาทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปี พ.ศ. 2550

ที่อยู่ : 106 หมู่ 2 ตำบลแม่ใจ อำเภอแม่ใจ จังหวัดพะเยา 56130

เบอร์โทรศัพท์ : 08-5204-7213

อีเมล์ : pa_prompong@hotmail.com

ความสามารถทางคอมพิวเตอร์ : Microsoft office , PLC , Solidwork 2008 , Photoshop

ฝึกงานที่ Water doctor Co.,Ltd ระหว่างเดือนมีนาคม ถึง เดือนพฤษภาคม พ.ศ. 2553

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประวัติผู้จัดทำ

นาย ภักดิคุณ อยู่ดี

เกิดวันที่ 30 มีนาคม พ.ศ. 2532

ประวัติการศึกษา : สำเร็จการศึกษาจากโรงเรียนพะเยาพิทยาคม อำเภอเมือง จังหวัดพะเยา

เข้าศึกษาที่คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมแมคคาทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปี พ.ศ. 2550

ที่อยู่ : 69 หมู่ 13 ตำบลท่าวังทอง อำเอเมือง จังหวัดพะเยา 56000

เบอร์โทรศัพท์ : 08-2629-0597

อีเมล์ : pakdikoon@hotmail.com

ความสามารถทางคอมพิวเตอร์ : Microsoft office , PLC , Solidwork 2008 , Photoshop

ฝึกงานที่ Water doctor Co.,Ltd ระหว่างเดือนมีนาคม ถึง เดือนพฤษภาคม พ.ศ. 2553

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประวัติผู้จัดทำ

นาย สุรสิทธิ์ วงศ์เฉลิมรัตน์

เกิดวันที่ 16 สิงหาคม พ.ศ. 2531

ประวัติการศึกษา : สำเร็จการศึกษาจากโรงเรียนปิยะมหาราชาลัย อำเภอเมือง จังหวัดนครพนม

เข้าศึกษาที่คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมแมคคาทรอนิกส์ สถาบันเทคโนโลยีพระ

จอมเกล้าเจ้าคุณทหารลาดกระบัง ในปี พ.ศ. 2550

ที่อยู่ : 33/2 ถ.บำรุงเมือง อ.เมือง จ.นครพนม 48000

เบอร์โทรศัพท์ : 08-3676-9917

อีเมล์ : sabol_4@hotmail.com

ความสามารถทางคอมพิวเตอร์ : Microsoft office , PLC , Solidwork 2008 , Photoshop

ฝึกงานที่ Bombardier Transportation Signal (Thailand) Ltd. ระหว่างเดือนมีนาคม ถึง เดือนพฤษภาคม พ.ศ. 2553

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้