ปีการศึกษา 2541

การวัดความเร็วของมอเตอร์เหนี่ยวนำโดยใช้โรเตอร์สล็อทฮาร์โมนิกส์

Speed Measurement of Induction Motor Using Rotor Slot Harmonic

โดย

นายจิระ	แซ่เตียว	รหัส 38014077
นายเฉลิมพล	ทองอุทัย	รหัส 38014095
นายฐิติพงศ์	สมัครพงศ์	รหัส 38014134
นายณรงค์วิทย์	ธรรมวรากุล	รหัส 38014137

อาจารย์ที่ปรึกษา

รศ.ดร. วิริยะ พิเชฐจำเริญ

ดร. วิจิตร กิณเรศ

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การวัดความเร็วของมอเตอร์เหนี่ยวนำโดยใช้โรเตอร์สล็อทฮาร์โมนิกส์

ผู้จัดทำ

1. นายจิระ แซ่เตียว
2. นายเฉลิมพล ทองอุทัย
3. นายฐิติพงศ์ สมัครพงศ์
4. นายณรงค์วิทย์ ธรรมวรากุล

.....อาจารย์ที่ปรึกษา

(รศ.ดร. วิริยะ พิเชฐจำเริญ)

.....อาจารย์ที่ปรึกษา

(ดร. วิจิตร กิณเรศ)

สารบัญ

การวัดความเร็วมอเตอร์เหนี่ยวนำโดยใช้โรเตอร์สล็อทฮาร์โมนิกส์

(Speed Measurement of Induction Motor Using Rotor Slot Harmonics)

นายจิระ แซ่เตียว
นายเฉลิมพล ทองอุทัย
นายฐิติพงศ์ สมัครพงศ์
นายณรงค์วิทย์ ธรรมวรากุล
รศ.ดร. วิริยะ พิเชฐจำเริญ อาจารย์ที่ปรึกษา
ดร. วิจิตร กิณเรศ อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

โครงงานนี้เป็นการศึกษาวิธีการวัดความเร็วมอเตอร์เหนี่ยวนำแนวทางใหม่โดยใช้เทคนิคการประมวลผลสัญญาณเชิงดิจิตอล (Digital Signal Processing) วิเคราะห์พฤติกรรมของโรเตอร์สล็อทฮาร์โมนิกส์ โดยแบ่งการดำเนินงานออกเป็น 2 ส่วน

1. *การประมวลผลแบบออฟไลน์* (Off Line) เป็นการศึกษาโดยใช้โปรแกรม MATLAB ในการวิเคราะห์หาตำแหน่งของโรเตอร์สล็อทฮาร์โมนิกส์เพื่อหาความเร็วรอบของมอเตอร์จากข้อมูลที่ได้มาจากดิจิตอลสตอเรจสโคป (Digital Storage Scope)
2. *การประมวลผลแบบเรียลไทม์* (Real Time) โดยในโครงงานนี้ได้ใช้บอร์ดประมวลผลทางดิจิตอล TMS320C50 DSK เพื่อใช้ในการวิเคราะห์และแปลงสเปกตรัมกระแสมอเตอร์ โดยในการทดลองจะต้องทำการลดทอนกระแสที่จะวัดด้วยเครื่องลดทอนกระแสก่อน และใช้คอมพิวเตอร์ในการโหลดโปรแกรมเข้าสู่บอร์ดTMS320C50 และแสดงผลลัพธ์ที่ได้จากการแปลงสเปกตรัมออกทางคอมพิวเตอร์

ในการดำเนินโครงงานทำให้ทราบถึงพฤติกรรม และเงื่อนไขต่างๆของโรเตอร์สล็อทฮาร์โมนิกส์ และได้ทำการทดลอง ปรับปรุงขั้นตอน วิธีการที่ใช้เพื่อให้ได้ค่าคลาดเคลื่อนน้อยที่สุด และนำไปสู่การพัฒนาเพื่อใช้งานจริงต่อไป

Speed Measurement of Induction Motor Using Rotor Slot Harmonics

Jira Saeteaw

Charlearmpol Tong-u-thai

Titipong Samakpong

Narongwit Thamwarakul

Assoc.Prof.Dr. Viriya Pichetchamroen Advisor

Dr. Vijit Kinnares

1998

Abstract

This project studies the measurement of the speed of induction motor using Digital Signal Processing techniques. The motor input current is processed by Fast Furier Transform and the result signal goes through spectral analyzing in order to deduce the motor speed. The project is divide into 2 part

1. Processing on Off Line that uses MATLAB program to analyze Rotor Slot Harmonics. The Data acquisition part consists of recording using a Digital Storage Scope.
2. Processing on Real Time that uses TMS320C50 board to analyze Rotor Slot Harmonics. Then transmits data to computer for spectrum display.

During the project implementation, more is learned about the behavior and conditions of Rotor Slot Harmonics. Also, additional test method improvements and different styles are tried in order to obtain minimum error so that methods can be developed into Real Time Mode in the future.

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

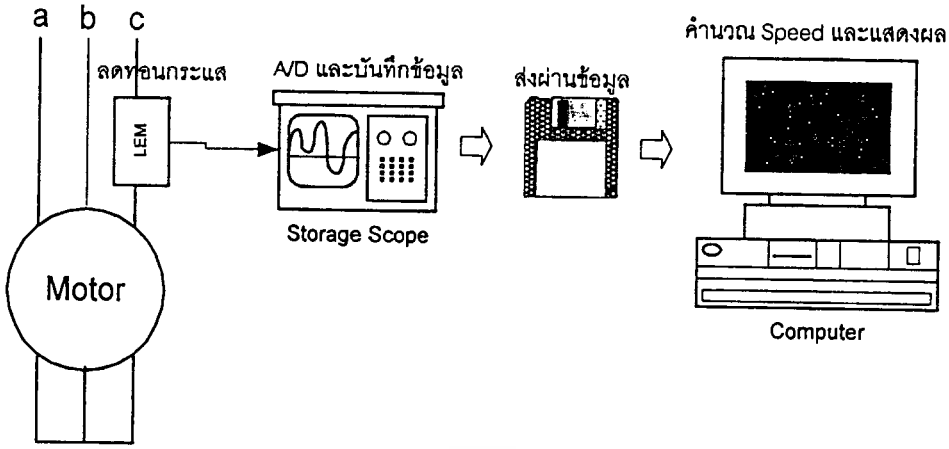
ในปัจจุบันนี้การควบคุมความเร็วและทอลค์มอเตอร์ให้มีประสิทธิภาพสูงสุด ได้ถูกพัฒนาขึ้นอย่างต่อเนื่อง ระบบขับมอเตอร์สมัยใหม่ต้องการความถูกต้องและการตอบสนองการปรับเปลี่ยนความเร็วได้รวดเร็ว ซึ่งในขณะนี้ระบบขับมอเตอร์ส่วนใหญ่จะใช้ทรานสดิวเซอร์ประเภท Optoelectrical หรือ Electromechanical เป็นตัววัดความเร็ว ซึ่งการใช้ทรานสดิวเซอร์ประเภทนี้มักต้องเผชิญกับปัญหา ความไม่เป็นเชิงเส้น ความถูกต้อง รีโซลูชั่น (Resolution) ต่ำ ขนาดของสัญญาณมีค่าน้อยเมื่อความเร็วต่ำๆ ราคาแพง ต้องทำการออกแบบจุดที่จะติดตั้ง ต้องมีการปรับแต่งก่อนใช้งาน การดูแลรักษา อายุการใช้งาน ฯลฯ

ดังนั้นในช่วง 2 ทศวรรษที่ผ่านมาได้มีการพยายามที่จะหาวิธี มาทดแทนการใช้ทรานสดิวเซอร์แบบเดิมๆ โดยการประยุกต์ใช้ปริมาณทางไฟฟ้าในตัวมอเตอร์ในการวัดความเร็ว เช่นใช้สัญญาณแรงดัน สัญญาณกระแสในมอเตอร์ และได้มีการทดลองและวิจัยหาวิธีการต่างๆ ซึ่งมีวิธีหนึ่งที่มีประสิทธิภาพคือ การใช้วิธีการวิเคราะห์โรเตอร์สล็อทฮาร์มอนิกส์ (Rotor Slot Harmonics) ซึ่งเกิดจากความไม่เป็นเชิงเส้นของสนามแม่เหล็กในช่องอากาศอันเนื่องมาจากความเป็นสล็อทของมอเตอร์ เมื่อมอเตอร์มีการหมุนจะเกิดการตัดกันของสนามแม่เหล็กเหนี่ยวนำให้เกิดสัญญาณโรเตอร์สล็อตฮาร์มอนิกส์ขึ้นในกระแสที่จ่ายให้กับมอเตอร์

โครงงานนี้เป็นการใช้หลักการแปลงฟาสท์ฟูเรียร์ (Fast Fourier Transform) แปลงรูปคลื่นของกระแสซึ่งอยู่ในโดเมนเวลาให้อยู่ในรูปของโดเมนความถี่ แล้วจึงวิเคราะห์ความถี่โรเตอร์สล็อทฮาร์โมนิกส์เพื่อหาค่าความเร็วรอบ ซึ่งลักษณะโดยรวมของโครงงานจะเริ่มจาก ใช้ตัวลดทอนกระแส (Current Transducer) แปลงค่ากระแสที่ป้อนเข้ามอเตอร์ให้เป็นแรงดันแล้วจึงแปลงค่าให้เป็นสัญญาณทางดิจิตอล แล้วจึงนำไปประมวลผล ฟาสท์ฟูเรียร์ นำสเปกตรัมที่ได้มาวิเคราะห์หาตำแหน่งของโรเตอร์สล็อตฮาร์มอนิกส์ เพื่อนำไปคำนวณหาความเร็วมอเตอร์ต่อไป

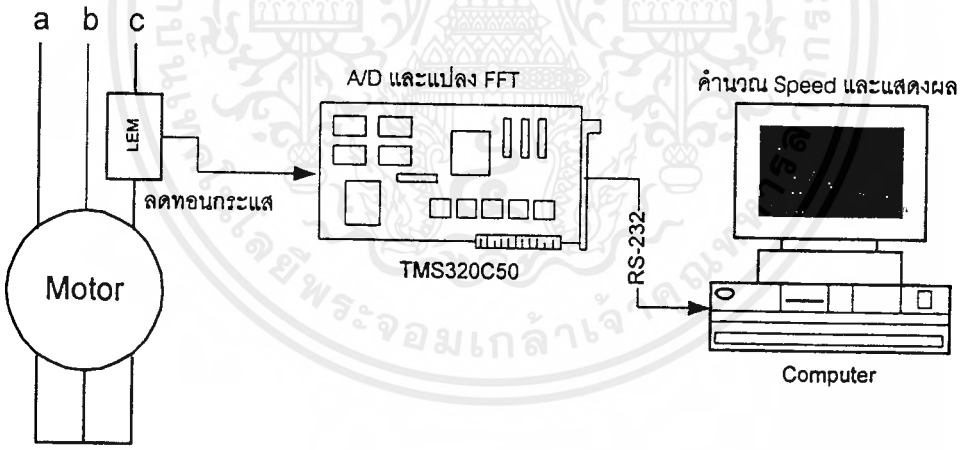
ซึ่งการศึกษาในโครงงานนี้แบ่งเป็น 2 ลักษณะคือ

1. การประมวลผลแบบออฟไลน์ (Off Line) ซึ่งทำโดยการใช้ ดิจิตอลสตอเรจสโคป รุ่น HP 54520C ทำการแปลงค่าสัญญาณที่ส่งมาจากหน่วยลดทอนกระแส ให้เป็นสัญญาณทางดิจิตอล บันทึกข้อมูลลงแผ่นดิสก์ แล้วใช้โปรแกรม MATLAB แปลงฟาสท์ฟูเรียร์ วิเคราะห์สเปกตรัม คำนวณความเร็ว และแสดงผลลัพธ์



รูปที่ 1.1 บล็อกไดอะแกรมแสดงการประมวลผลแบบออฟไลน์ (Off Line)

2. การประมวลผลแบบเรียลไทม์ (Real Time) ซึ่งได้ใช้บอร์ดประมวลผล DSP รุ่น TMS320C50 เป็นตัวรับสัญญาณจากหน่วยลดทอนกระแส แปลงให้เป็นสัญญาณดิจิตอล ทำการแปลง ฟาสท์ฟูเรียร์ แล้วส่งผลลัทธ์ไปแสดงออกทางคอมพิวเตอร์และวิเคราะห์สเปกตรัมเพื่อคำนวณหาความเร็ว



รูปที่ 1.2 บล็อกไดอะแกรมแสดงการประมวลผลแบบเรียลไทม์ (Real Time)

ซึ่งในโครงงานนี้ได้ทำการทดลองกับมอเตอร์หลายแบบคือ มอเตอร์เหนี่ยวนำชนิดโรเตอร์กรงกระรอก (Squirrel Cage) ขนาด 1 แรงม้า , ขนาด 2 แรงม้า , มอเตอร์เหนี่ยวนำชนิดโรเตอร์พันขดลวด (Wound Rotor) ขนาด 1 KW ภาระ(Load) ที่ใช้มี 2 แบบคือ โหลดทางกล และโหลดทางไฟฟ้า (ต่อผ่าน DC Generator) และการทดลองที่แรงดันไฟตก (Under Voltage) และแรงดันไฟเกิน (Over Voltage)

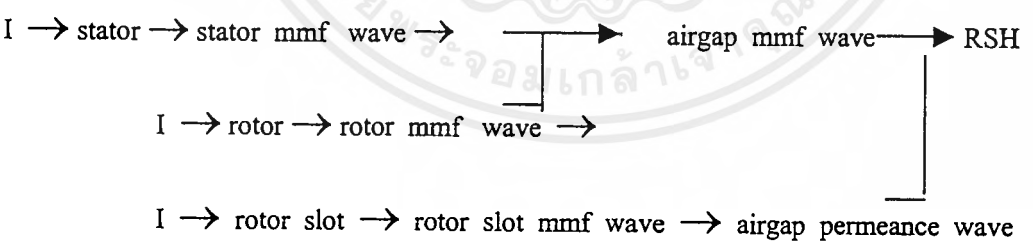
บทที่ 2

การใช้โรเตอร์สล็อทฮาร์โมนิกส์ในการหาค่าความเร็วของมอเตอร์เหนี่ยวนำ

2.1 การเกิดโรเตอร์สล็อทฮาร์โมนิกส์

2.1.1 การเกิดจากแหล่งจ่ายที่เป็นไซน์ (Sinusoidal)

ในมอเตอร์เหนี่ยวนำเมื่อเราจ่ายไฟฟ้ากระแสสลับ ที่มีลักษณะของรูปแบบสัญญาณ (Wave form) ของศักดาไฟฟ้าเป็นแบบไซน์ (Sinusoidal) จะทำให้เกิดสนามแม่เหล็กภายในช่องอากาศ (Airgap field) และเกิดแรงเคลื่อนแม่เหล็กภายในช่องอากาศ (Magneto Motive Force :Airgap mmf) ขึ้น และจะมีลักษณะการกระจายเป็นแบบไซน์ และเราพิจารณาให้ที่โรเตอร์มีลักษณะเป็นร่องสล็อท(slotting) ที่สเตเตอร์มีลักษณะที่เรียบ(smooth) การเกิดโรเตอร์สล็อตฮาร์โมนิกส์สามารถพิจารณาได้โดยย่อจากแผนภูมิด้านล่าง



เมื่อเราจ่ายศักดาไฟฟ้า 3 เฟสให้กับมอเตอร์จะเกิดแรงเคลื่อนแม่เหล็กไฟฟ้าในช่องอากาศขึ้น(airgap mmf wave) ซึ่งแรงเคลื่อนแม่เหล็กในช่องอากาศนี้เกิดมาจากผลของแรงเคลื่อนแม่เหล็กจากสเตเตอร์และแรงเคลื่อนแม่เหล็กที่โรเตอร์(stator mmf wave and rotor mmf wave) เนื่องจากการที่มีกระแสไหลในสเตเตอร์และโรเตอร์ ในขณะเดียวกันในช่องอากาศจะเกิดความเป็นตัวนำแม่เหล็กของช่องอากาศ(airgap permeance)ขึ้น ซึ่งการกระจายของความเป็นตัวนำแม่เหล็กภายในช่องอากาศนี้จะกระจายอย่างไม่เรียบ(variation)เนื่องจากผลของการที่โรเตอร์มีความเป็นสล็อทนั่นเอง

พิจารณาสมการของความหนาแน่นของฟลักซ์แม่เหล็กภายในช่องอากาศ

$$B_{ag}(\varphi_E, \theta_{rm}) = MMF(\varphi_E, \theta_{rm}) \cdot P_{ag}(\varphi_E, \theta_{rm})$$

เมื่อ B_{ag} คือ ความหนาแน่นของฟลักซ์แม่เหล็ก

MMF_{ag} คือ แรงเคลื่อนแม่เหล็กในช่องอากาศ

P_{ag} คือ ความเป็นตัวนำแม่เหล็กในช่องอากาศ

φ_s คือ ทิศทางเชิงมุมของสเตเตอร์(stator angular position)

θ_{rm} คือ ทิศทางทางกลของโรเตอร์(mechanical rotor position)

ดังนั้นเมื่อความเป็นตัวนำทางแม่เหล็กของอากาศไม่เรียบแล้วย่อมส่งผลให้ความหนาแน่นของฟลักซ์แม่เหล็กในช่องอากาศเกิดความผิดปกติขึ้นด้วย ทำให้ในช่องอากาศนี้จะเกิดฮาร์โมนิกส์ขึ้นมาซึ่งฮาร์โมนิกส์ที่เกิดขึ้นมาจะมีความสัมพันธ์กับความเร็วของมอเตอร์ ขนาดกระแส และจำนวนโรเตอร์ฮารโมนิกส์ที่เกิดก็คือ "โรเตอร์สล็อทฮาร์โมนิกส์" (Rotor slot harmonics) นั่นเอง

โรเตอร์สล็อทฮาร์โมนิกส์ที่เกิดขึ้นจะเหนี่ยวนำทำให้เกิดกระแสฮาร์มอนิกส์ไหลในขดลวดสเตเตอร์โดยที่ค่าของความถี่จะไม่มีการเปลี่ยนแปลงจากเดิม และมีรูปแบบของกระแสฮาร์มอนิกส์ที่ถูกเหนี่ยวนำในเฟส A เป็น

$$I_a(t) = \sum_{v=0}^{2} A_v \cos(\omega_v t - \psi_v)$$

โดยที่ A_v = ขนาด

ψ_v = เฟสขององค์ประกอบฮาร์มอนิกส์ที่ v^{th}

phase shift = $2\pi/3$,$4\pi/3$

ซึ่งถ้าเราพิจารณาองค์ประกอบฮาร์มอนิกส์ต่างๆในกระแสสเตเตอร์ (หรืออาจจะพิจารณาจากศักดาก็ได้)จะได้สมการความสัมพันธ์พื้นฐานของโรเตอร์สล็อทฮาร์โมนิกส์ เป็น

$$\omega_{sh} = \frac{Z}{P}\omega_r \pm \omega_0$$

โดยที่ ω_{sh} = ความถี่เชิงมุมของโรเตอร์สลิ่อทฮาร์โมนิกส์

ω_r = ความถี่เชิงมุมของโรเตอร์

ω_0 = ความถี่เชิงมุมของกระแสที่ป้อนเข้าที่สเตเตอร์

Z = จำนวนโรเตอร์สลิ่อท

P = จำนวนของคู่ขั้วแม่เหล็ก (Pole – Pair)

เราสามารถที่จะคำนวณค่าความเร็วของมอเตอร์เหนี่ยวนำได้จาก $\omega = 2\pi f$ จะได้สมการเป็น

$$f_{sh} = \frac{Z}{P} f_r \pm f_0 \qquad \text{เมื่อ } Z/P = 3N \pm 1 \text{ ; } N = 1,2, \quad \ldots\ldots 1.)$$

และนอกจากนี้ความสัมพันธ์ที่เราสามารถพบได้บ่อยๆและนำมาใช้ในโครงงานนี้ก็คือ

$$f_{sh} = \left[Z\frac{(1-s)}{P} \pm 1 \right] f_0 \qquad \ldots\ldots\ldots 2)$$

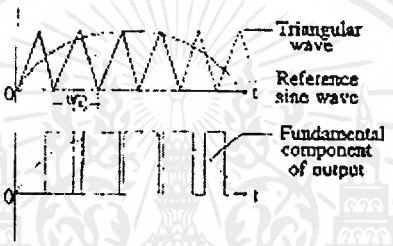
โดยที่ f_{sh} เป็นความถี่ของโรเตอร์สลิ่อทฮาร์โมนิกส์ (เฮิร์ท)

f_o เป็นความถี่ของแหล่งจ่ายศักดาไฟฟ้าที่สเตเตอร์

ในการวิเคราะห์ในข้างต้นจะเป็นเพียงการคิดความเป็นสลิ่อทเพียงข้างเดียวหรือการพิจารณาที่คิดเสมือนว่าสเตเตอร์ราบเรียบและมีเพียงโรเตอร์เท่านั้นที่มีลักษณะที่เป็นสลิ่อท ในทางปฏิบัติแล้วเราจะต้องคิดความเป็นสลิ่อทของทั้งสเตเตอร์และโรเตอร์ แต่ในกรณีเครื่องจักรกลไฟฟ้ามีจำนวนขั้วน้อย ความเป็น สลิ่อทของสเตเตอร์จะถือว่าไม่มีผลกับความนำแม่เหล็กในช่องอากาศ

2.1.2 การเกิดจากแหล่งจ่ายที่ไม่เป็นไซน์(Non – Sinusoidal)

ในการวิเคราะห์หัวข้อแรกเราพิจารณาว่าแหล่งจ่ายศักดาของมอเตอร์เหนี่ยวนำแบบ 3 เฟส เป็นแหล่งจ่ายที่มีความสมดุลย์ (Symmetrical) และมีความเป็นไซน์ แต่ในความเป็นจริงแล้ว ศักดาและกระแสที่ป้อนให้กับมอเตอร์นั้นจะไม่มีทางที่จะเป็นไซน์ที่แท้จริง(Pure sinusoidal)ได้เลย จะมีการผิดเพี้ยนของรูปสัญญาณ(distortion)อยู่ ตัวอย่างเช่นการใช้วงจรขับมอเตอร์(motor drive) ที่เป็นแบบพัลส์วิดมอดดูเลชัน (PWM) ที่มีส่วนประกอบของ inverter มาใช้งานกันอย่างกว้างขวาง inverter นี้สามารถจ่ายศักดาหรือกระแสที่เหมือนกันทุกประการในแต่ละเฟส แต่ศักดาและกระแสนี้จะไม่เป็นรูปแบบไซน์ที่สมบูรณ์ แสดงให้เห็นในรูปที่ 2.1



รูปที่ 2.1 แสดงกราฟของกระแสของ PWM ที่ไม่เป็นไซน์

การไม่เป็นไซน์ที่สมบูรณ์แบบจะทำให้เกิดองค์ประกอบของฮาร์มอนิกส์ที่มีความถี่เป็นจำนวนเท่าของความถี่หลักมูลและฮาร์มอนิกส์ทีมีความถีที่ไม่เป็นจำนวนเท่าของความถี่หลักมูล(Time and Space Harmonics) ขึ้นจำนวนมาก โดยปกติแล้วการใช้อินเวอร์ทเตอร์จะทำให้เกิดฮาร์มอนิกส์ในลำดับที่ α โดยที่ $\alpha = 6m \pm 1$ และ $m = 1,2,3,\ldots$ และจากการที่มีฮาร์โมนิกส์ที่เพิ่มขึ้นมานี้เองจะส่งผลทำให้ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์เลื่อนไปจากปกติ โดยจะมีความสัมพันธ์เป็น

$$f_{sh} = \frac{Z}{P} f_r \pm \alpha f_0 \qquad \text{.....3)}$$

และความสัมพันธ์

$$f_{sh} = \left[Z \frac{(1-s)}{P} \pm \alpha \right] f_0 \qquad \text{.....4)}$$

โดยที่ $\alpha = 6M \pm 1$ $\qquad M = 1, 2, 3, \ldots$

ขนาดของฮาร์มอนิกส์เหล่านี้จะเป็นสัดส่วนผกผันกับลำดับที่ของฮาร์มอนิกส์ อย่างไรก็ตาม ฮาร์มอนิกส์เหล่านี้สามารถถูกกำจัดได้ด้วยการใช้เทคนิคการ switching ที่เหมาะสม

สมการที่สามารถใช้ระบุความสัมพันธ์ของโรเตอร์สล็อตฮาร์โมนิกส์ได้ในทุกๆกรณี ไม่ว่าจะเป็นแหล่งกำเนิดที่เป็นไซน์หรือแหล่งกำเนิดที่ไม่เป็นไซน์ และเป็นสมการต้นแบบของทั้ง 4 สมการที่กล่าวมาแล้ว ได้แก่

$$f_{sh} = f_o\left((kZ + n_d)\left(\frac{1-s}{P}\right) + n_w \right) \qquad .....5)$$

โดยที่ k เป็นเลขจำนวนเต็ม , Z เป็นจำนวนสล็อทที่โรเตอร์ , P เป็นจำนวนคู่โพล , n_d เป็นค่าสัมประสิทธิ์ลำดับความบิดเบี้ยวของโรเตอร์ (rotor eccentric) , n_w = ค่าลำดับของฮาร์โมนิกส์ของแรงเคลื่อนแม่เหล็กในช่องอากาศ(airgap mmf harmonics order) ซึ่งจะมีค่าดังต่อไปนี้

k = 0 , 1 , 2 , 3 , . . . โดยปกติเป็น 1

n_d = 0 , $\pm$1 , . . . โดยปกติเป็น 0 หรือ 1

n_w = $\pm$1 , $\pm$3 , . . . โดยปกติเป็น 1 หรือ -1

ซึ่งค่าเหล่านี้จะมีค่าที่ต่างกันในมอเตอร์แต่ละตัว จะต้องมีการทดสอบมอเตอร์ที่จะนำมาวัดความเร็ว

2.2 พัฒนาการของการหาความเร็วโดยใช้ โรเตอร์สล็อทฮาร์โมนิกส์

การศึกษาการใช้โรเตอร์สล็อทฮาร์โมนิกส์มาช่วยในการหาความเร็วของมอเตอร์นั้น ได้เริ่มโดยศึกษาของ Ishida Hayashi และ K. Iwata ใช้เทคนิคในการหาโรเตอร์สล็อทฮาร์โมนิกส์ โดยการใช้วงจรกรองแบบอนาลอก [1]1 ต่อมา B. Williams I. Goodfellow และ T.C.Green ใช้วงจรกรองแบบ switched capacitor filter ได้ถูกประยุกต์รวมกับวิธีของ Ishida [2]1 และเทคนิค Phase Lock Loop ถูกนำมาใช้ในการหาโรเตอร์สล็อทฮาร์โมนิกส์ โดย D. Zinger, F. Profumo, T. Lipo , และ D.Novotny [3]1

ซึ่งเทคนิคที่กล่าวมาแล้วต่างก็มีข้อจำกัดในด้าน ความถูกต้อง, ความเป็นเชิงเส้น, Resolution, ขอบเขตของความเร็วมอเตอร์, ความเร็วในการตอบสนอง ซึ่งการแก้ปัญหาเหล่านี้จะเกิดกับการใช้การประมวลผลแบบอนาลอก ดังนั้นจึงได้มีการพัฒนาขึ้นเป็นการใช้วิธีการทางดิจิตอล คำนวณสเปกตรัมความถี่ โดยใช้การแปลง ฟาสท์ฟูเรียร์ ในการทดลองของ A, Ferrah, K.J.Bradley ในปี 1992 [4]1 ซึ่งประสบความสำเร็จอย่างมากในด้านความถูกต้องและความเป็นเชิงเส้นที่สูงขึ้น สามารถใช้งานได้ที่ช่วงความเร็วของมอเตอร์ที่กว้างขึ้น และที่เงื่อนไขของโหลดที่มากขึ้น แต่วิธีการแปลงฟาสท์ฟูเรียร์ ยังมีข้อจำกัดอยู่ที่ frequency resolution และความเร็วในการตอบสนอง(คำนวณ) คือถ้าเราต้อง frequency resolution ต่ำๆแล้ว(เพื่อความถูกต้องมากขึ้น)จะทำได้โดยการลดอัตราการ Sampling หรือ เพิ่มจำนวนข้อมูลที่สุ่มเข้ามาซึ่งทำให้ความเร็วในการตอบสนองช้า

การทำการประมาณ (Interpolation) สามารถช่วยปรับปรุง frequency resolution โดยที่เรารับข้อมูล

เข้ามาน้อยๆ แล้วทำการ Interpolate ทำให้ได้ข้อมูลมากขึ้นเป็นผลทำให้ frequency resolution มีค่าต่ำลงได้ ซึ่ง Brasco [5]1 ใช้ช่วยทำการ Vector controlled

เพื่อลดข้อจำกัดที่เกิดขึ้นโดยปกติเมื่อใช้ วิธีการแปลงฟาสท์ฟูเรียร์ เช่น Spectrum Leakage ,ความต้องการ resolution ต่ำๆที่ข้อมูลบันทึกมาสั้นๆ จึงได้มีการประยุกต์วิธีใหม่ในการประเมินสเปกตรัม วิธีเหล่านี้เป็นวิธีการทำนายเกี่ยวกับพารามิเตอร์ ซึ่งมันต้องการข้อมูลก่อน (A priori knowledge) เพื่อที่จะทำสัญญาณกระแสให้เหมาะกับรูปแบบ Stochastic แล้วใช้รูปแบบการทำนายเชิงเส้นในอันที่ซึ่ง ตัวอย่าง(Samples)ของสัญญาณล่วงหน้าถูกทำนายโดยการรวมกันเชิงเส้นของตัวอย่างก่อนหน้า กับ ตัวอย่างปัจจุบัน รูปแบบของพารามิเตอร์ ที่ได้จากการรวมกันจนได้ค่าที่ดีที่สุด จะทำให้เป็นสัญญาณสเปกตรัม วิธีการเก็บยอด (A peak picking algorithm) [4] จะนำมาใช้ในการหาตำแหน่งขององค์ประกอบที่สนใจในสเปกตรัมที่เกิดขึ้น และคำนวณหาความเร็ว

วิธีเกี่ยวกับพารามิเตอร์ ประกอบด้วย วิธี Batch หลายๆแบบ และวิธีการทำซ้ำ (recursive methods)

- วิธี Batch ประกอบด้วย

วิธี Maximum Entropy [8] , วิธี Covariance [9] , วิธีProny [10]

- วิธีการทำซ้ำ (Recursive)

เทคนิคการปรับการทำนาย(predictive adaptive techniques) โดยใช้ LMS (The least mean squares) [11] , วิธี FRLS (The fast recursive least-squares) [12] , ขั้นตอนดำเนินการของ Kalman [13] (Kalman Algorithm)

วิธีการต่างๆข้างต้นเป็นทางเลือกสำหรับแทนการแปลงฟาสท์ฟูเรียร์ มีประสิทธิภาพในการลดระยะเวลาการบันทึก ดังนั้นทำให้การหาความเร็วทำได้เร็วขึ้น ในเทคนิคทั่วไปเหล่านี้ จำเป็นต้องใช้วงจร กรองช่วงความถี่แบบดิจิตอล (Digital band pass filter) ในการกรอง เพื่อรับข้อมูลเพียงที่อยู่ในช่วงที่เราต้องการที่จะได้นำไปสร้าง Model และนำไปคำนวณหาลำดับ (order) ของ stochastic วิธีนี้ทำให้ลดเวลาในการคำนวณความเร็ว อย่างไรก็ตามวิธีการเหล่านี้อาจจะมีประสิทธิภาพน้อยกว่าวิธีการแปลงฟาสท์ฟูเรียร์

FRLS เป็นที่ยอมรับกันว่าเป็นวิธีการคำนวณแบบการทำซ้ำ (recursive) ที่เร็วที่สุด แต่อย่างไรก็ตามการหาความเร็วที่ภาวะ Steady – State วิธีการแปลงฟาสท์ฟูเรียร์ จะให้ความถูกต้องมากกว่า

การใช้การแปลงฟาสท์ฟูเรียร์ ร่วมกับ batch method ถูกใช้อย่างสมบูรณ์ในการที่จะแสดงผลในจุดที่มีประสิทธิภาพมากที่สุด ในขณะที่ใช้ค่าใช้จ่าย(เวลา)ที่น้อยที่สุด

แต่ในการวิจัยนั้นมิได้มุ่งไปที่เพียงวัดค่าความเร็วเพียงอย่างเดียว ยังสามารถประยุกต์เป็นวิธีการที่จะสามารถบอกจำนวนของโรเตอร์สล็อทโดยอัตโนมัติจากสเปกตรัมของกระแสเพื่อนำไปใช้ในอุปกรณ์จำพวกทำหน้าที่ได้ด้วยตัวเอง (Self Commissioning) เช่น AC Drive เพราะว่าจำนวนของโรเตอร์สล็อท เป็นพารามิเตอร์ ของเครื่องจักรกลไฟฟ้าที่ผู้ผลิตจะไม่บอกมา

2.3 วิธีการวิเคราะห์หาโรเตอร์สล็อทฮาร์โมนิกส์

ในรอบหลายปีที่ผ่านมานี้ ได้มีการเสนอเทคนิคและวิธีการในการคำนวนหาความเร็วของมอเตอร์เหนี่ยวนำ ทั้งที่เป็นเทคนิคทางอนาลอกและดิจิตอล ซึ่งในปัจจุบันนี้วิธีการส่วนใหญ่จะใช้เทคนิคทาง ดิจิตอลเนื่องจากช่วยเพิ่มค่าความถูกต้องให้กับการคำนวณ และในด้านของความรวดเร็วนั้นอุปกรณ์ที่เป็น ดิจิตอลได้มีการพัฒนาให้มีความรวดเร็วมากขึ้น.

ในช่วง 30 ปีที่ผ่านมา จะมีการใช้ทฤษฎีของ parametric method ในการช่วยปรับปรุงข้อเสียของการแปลงฟาสท์ฟูเรียร์ เช่น ในเรื่องของความยาวของการเก็บข้อมูล การใส่หน้าต่าง (windowing) และการประมาณค่าภายในช่วง (interpolation) ฯลฯ ซึ่งเทคนิควิธีการต่างๆ นั้นอาจจะสามารถแบ่งแยกได้ตามการใช้ข้อมูลในการประมวลผลได้เป็น 2 ประเภทใหญ่ๆ ได้แก่

- วิธีการคำนวณที่ใช้การรับข้อมูลเข้ามาเป็นชุดๆ (batch data)
- วิธีการคำนวณที่ใช้การรับข้อมูลเข้ามาแบบต่อเนื่อง (sequential data)

1. Batch Algorithm

วิธีการนี้ข้อมูลที่จะนำมาประมวลผลจะถูกสุ่มเข้ามา โดยมีการกำหนดขนาดของข้อมูลเอาไว้ก่อนหน้าแล้ว วิธีการเกี่ยวกับพารามิเตอร์ ที่ใช้ batch algorithm จะประกอบไปด้วยหลายวิธี เช่น

- Georizel algorithm (GAL)
- Fast Furrier Transform (FFT)
- Covariance method
- Prony method
- Maximum Entropy Method (MEM)

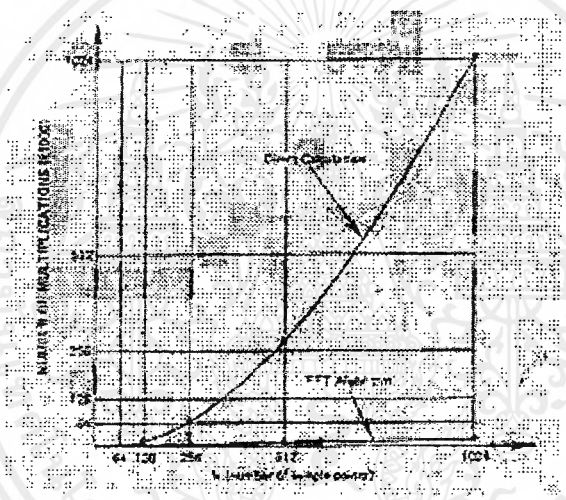
วิธีการ Covariance และ วิธี Prony เป็นวิธีที่ไม่นิยมมากนัก แต่ถ้าผู้อ่านสนใจอาจจะหาข้อมูลเพิ่มเติมได้ที่บทความ " Block time and frequency – domain modified covariance method of spectrum analysis" และหนังสือ " Advance of calculus application, Practice – Hall โดย F.B.Hildenberg ตามลำดับ

FFT กับ GAL ในรายงานฉบับนี้จะได้เสนอเกี่ยวกับ หลักการและข้อดีข้อเสียของ FFT และ GAL วิธีการแปลงฟาสท์ฟูเรียร์ (Fast Furrier Transform) จะเป็นการทำการแปลงฟูเรียร์แบบเต็มหน่วย (Discrete Furrier Transform) : DFT อย่างหนึ่ง แต่จะมีขั้นตอนการลดทอนจำนวนครั้งของการคิดให้น้อยลงกว่าการแปลงฟูเรียร์แบบโดยตรง ซึ่งจะมีทั้งการลดทอนทางเวลาและการลดทอนทางความถี่ แต่วิธี GAL จะเป็นการคำนวณ การแปลงฟูรียร์แบบเต็มหน่วย โดย

ตรงไม่มีการลดทอนขั้นตอน ทำให้ GAL ใช้เวลาในการคำนวณนานกว่าฟาสท์ฟูเรียร์ ความเร็วในการคำนวณสามารถเปรียบเทียบได้โดย สมมติให้มีข้อมูลที่ต้องการแปลง การแปลงฟูเรียร์แบบเต็มหน่วย เป็น N ตัว (ตัวอย่างการสุ่ม)

การแปลงฟูเรียร์แบบเต็มหน่วย โดย GAL ต้องคูณจำนวนเชิงซ้อน N^2 ครั้ง และการบวกจำนวนเชิงซ้อน N(N-1) ครั้ง

การแปลงฟูเรียร์แบบเต็มหน่วยโดยใช้การแปลงฟาสท์ฟูเรียร์แทน ต้องคูณจำนวนเชิงซ้อนเพียง $N/2(\log_2 N)$ ครั้ง และการบวกจำนวนเชิงซ้อนเพียง $N(\log_2 N)$ ครั้ง ความแตกต่างนี้จะมีมากขึ้นเรื่อยๆ เมื่อค่าของ N มีค่ามากขึ้น ดังรูปด้านล่าง



รูปที่ 2.2 แสดงการเปรียบเทียบจำนวนครั้งที่ต้องใช้ในการบวกและคูณจำนวนเชิงซ้อนของทั้ง 2 วิธี

แต่ถ้าที่ในเงื่อนไขที่จำนวนข้อมูลที่นำมาคิดมีน้อยนั้น วิธี GAL จะสามารถให้ประสิทธิภาพที่ดีกว่า ฟาสท์ฟูเรียร์ และเนื่องจากที่ทั้งสองวิธีมีพื้นฐานที่เหมือนกัน ดังนั้นจึงมีปัญหาที่เหมือนกันคือ- การเกิดการซ้อนทับกันของความถี่ (alasing) การเกิดการรั่วไหลของพลังงาน (leakage) ปรากฏการณ์ picket fence และ frequency resolution ไม่ดีพอ

ซึ่งปัญหาในบางกรณีก็สามารถปรับปรุงได้เช่น การเกิดการการซ้อนทับกันของความถี่เราสามารถแก้โดยการใช้อัตราการสุ่มสัญญาณให้มากกว่า 2 เท่าของสัญญาณที่พิจารณา และถ้าหากมีข้อมูลไม่เพียงพอ อาจใช้การ interpolate ข้อมูลขึ้นมาก็ได้ frequency resolution สามารถปรับปรุงได้โดยการเพิ่มจำนวนของข้อมูลที่ใช้ในการคำนวณ เพราะว่าการเพิ่มจำนวนข้อมูลเป็น

การลด frequency resolution ($F_{res} = F_{sam}/N$) จะทำให้มีความถูกต้องมากขึ้น หรืออาจจะลดความถี่ของการสุ่มสัญญาณ($F_{sampling}$) แต่จะไม่นิยมทำเพราะจะทำให้เกิดการ aliasing ได้ แต่การเพิ่มจำนวนของข้อมูลจะทำให้เสียเวลาในการประมวลผลนานขึ้น

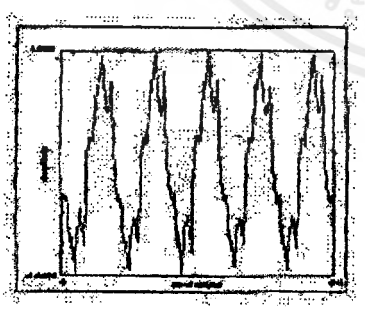
2. Adaptive Algorithm

เนื่องจาก โรเตอร์สลอตฮาร์มอนิกส์ จะมีการเปลี่ยนแปลงตามโหลด และ ความถี่ของ inverter ที่ใช้เป็นแหล่งจ่ายศักดาไฟฟ้า ดังนั้นสัญญาณไม่ควรถูกสมมุติให้หยุดนิ่งภายในกรอบ (Stationary in window) แต่วิธีของ FFT และ MEM จะสมมุติสัญญาณให้หยุดนิ่งอยู่ภายในกรอบของข้อมูล

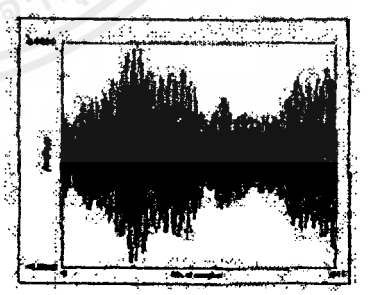
วิธีที่หนึ่งที่ใช้แทนการสมมุติให้สัญญาณอยู่นิ่งภายในกรอบ คือ การทำ moving data window แต่ไม่สามารถนำไปใช้ในวิธีฟาสท์ฟูเรียร์ได้ เนื่องจากทำได้ยากและไม่มีทางเป็นไปได้ ถ้าหากนำไปประยุกต์ใน MEM สามารถทำได้โดยง่าย เพราะเราสามารถที่จะใช้กรอบข้อมูลขนาดเล็กในวิธีของ MEM ได้ แต่เทคนิคนี้ยังมีข้อจำกัดที่เรายังสมมุติข้อมูลหยุดนิ่งภายในกรอบเล็กๆอยู่ดี

Adaptive Algorithm นี้เป็นเทคนิคที่พบมากในการหาค่าสัมประสิทธิ์ของ วิธีการคำนวณ Autoregressive โดยใช้หลัก Sample-by-sample. Adaptive Algorithm ที่ง่ายที่สุดคือ LMS และวิธีอื่นที่น่าสนใจ เช่น FRLS และ KF

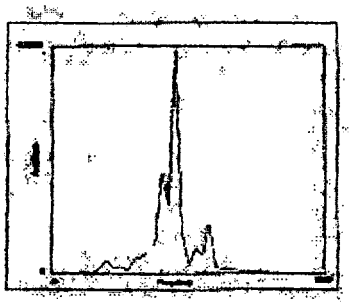
รูปแสดงการเปรียบเทียบสเปกตรัมจากวิธีต่างๆ



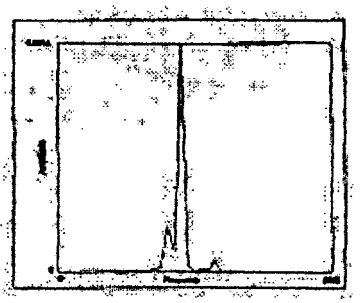
รูปที่ 2.4 สัญญาณกระแสที่สเตเตอร์ที่ความถี่ 50 Hz



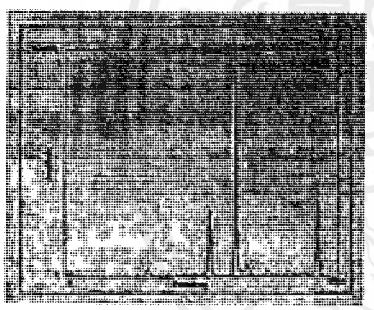
รูปที่ 2.5 สัญญาณเมื่อผ่านตัวกรองดิจิตอล



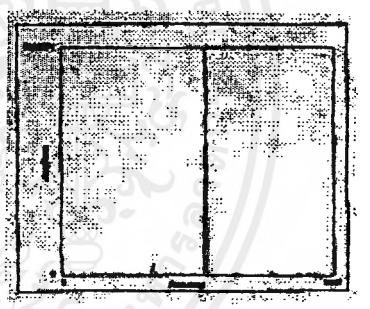
รูปที่ 2.6 สเปกตรัมของสัญญาณที่ผ่านการกรอง โดยใช้ FFT



รูปที่ 2.7 สเปกตรัมของสัญญาณที่ผ่านการกรอง โดยใช้ GAL



รูปที่ 2.8 สเปกตรัมของสัญญาณที่ผ่านการกรอง โดยใช้ MEM



รูปที่ 2.9 สเปกตรัมของสัญญาณที่ผ่านการกรอง โดยใช้ FRLS

ซึ่งในโครงงานนี้เลือกศึกษาการวิเคราะห์โรเตอร์สลอตฮาร์มอนิกส์โดยใช้ฟาสท์ฟูเรียร์ เพราะเป็นวิธีที่เหมาะสมกับพื้นฐานต่างๆที่มีอยู่ และสร้างความเข้าใจเบื้องต้นเกี่ยวกับพฤติกรรมของโรเตอร์สลอตฮาร์มอนิกส์

บทที่ 3
การแปลงฟาสท์ฟูเรียร์ (Fast Furier Transform)

3.1 บทนำ (Introduction)

โดยทั่วไปเราอาจกล่าวได้ว่าฟังก์ชันต่อเนื่อง (Continuos Function) ใดๆ ที่มีการซ้ำทุกๆช่วงเวลา T เราจะสามารถเขียนให้อยู่ในรูปผลบวกของคลื่นไซน์ (Sine wave) ที่มีความถี่หลักมูล(Fundamental) กับความถี่ฮาร์มอนิกส์ที่ลำดับสูงขึ้นไป ซึ่งเป็นจำนวนเท่าของความถี่หลักมูลได้เสมอ

ดังนั้นในการวิเคราะห์สัญญาณจะกล่าวถึงกระบวนการในการคำนวณขนาด (Magnitude)และมุมเฟส (Phase angle) ของแต่ละส่วนประกอบที่เป็นมูลฐานและในลำดับฮาร์มอนิกส์ ที่สูงขึ้น ผลของอนุกรมที่ได้เรียกว่า "อนุกรมฟูเรียร์"(Furier Series) ซึ่งแสดงความสัมพันธ์ในรูปโดเมนเวลา (Time domain) และโดเมนความถี่ (Frequency domain)

ในกรณีการวิเคราะห์สำหรับฟังก์ชั่นทั่วไป ซึ่งอยู่ในช่วง ∞ ถึง ∞- จะทำการวิเคราะห์โดยอาศัยการแปลงฟูเรียร์ (Furier Transform) ซึ่งเป็นการแปลงฟังก์ชั่นในโดเมนเวลาไปสู่โดเมนความถี่ และ อินเวอร์สการแปลงฟูเรียร์ (Inverse Furier Transform) ซึ่งเป็นการแปลงฟังก์ชั่นในโดเมนความถี่ไปสู่โดเมนเวลา ดังนั้นเราอาจกล่าวได้ว่า อนุกรมฟูเรียร์เป็นกรณีหนึ่งของการแปลงฟูเรียร์

อย่างไรก็ตามในทางปฏิบัติ ข้อมูลหรือค่าที่จะทำการวิเคราะห์เป็นค่าซึ่งได้จากการสุ่มค่า (Sampled data) ของฟังก์ชั่นในโดเมนเวลาซึ่งค่าที่ได้จะอยู่ในรูปของขนาดของฟังก์ชั่น ณ เวลาที่ทำการสุ่ม ซึ่งระยะห่างของการสุ่มแต่ละครั้งจะเป็นช่วงเวลาที่แน่นอน ในการแปลงฟูเรียร์ของค่าที่ได้จากการสุ่มเหล่านี้จะต้องใช้การแปลงที่เรียกว่า การแปลงฟูเรียร์เต็มหน่วย (Discrete Furier Transform: DFT) เนื่องจากการแปลงฟูเรียร์เต็มหน่วยนั้นใช้เวลาในการประมวลผลค่อนข้างมาก เราสามารถทำการแปลงฟูเรียร์เต็มหน่ายให้รวดเร็วขึ้นได้โดยการอาศัยวิธีการที่เรียกว่า การแปลงฟาสท์ฟูเรียร์ (Fast Furier Transform : FFT) ซึ่ง FFT เป็นวิธีการพื้นฐานซึ่งถือได้ว่ารวดเร็วและได้รับการยอมรับมากในการนำมาวิเคราะห์รูปแบบต่างๆ ดังนั้นในการวิเคราะห์สัญญาณที่จะกล่าวต่อไป จะเน้นในส่วนของ FFT

3.2 การวิเคราะห์อนุกรมฟูเรียร์ (Furier Series Analysis)

อนุกรมฟูเรียร์ของฟังก์ชั่นคาบ (Periodic function) *X(t)* ใดๆ อาจแสดงได้ดังนี้

$$X(f_n) = \sum_{n=-\infty}^{\infty} X(f_n) e^{-j2\pi f_n t} \qquad (3.1)$$

โดยที่ a_0 เป็นค่าเฉลี่ยของฟังก์ชัน *X(t)*

a_n และ b_n เป็นสัมประสิทธิ์ของอนุกรมของฟังก์ชัน *X(t)*

เมื่อ

$$a_0 = \frac{1}{T} \int_{-T/2}^{T/2} X(t)dt \qquad (3.2)$$

ซึ่งหมายถึงพื้นที่ใต้กราฟของ *X(t)* ในช่วง *T/2* ถึง *–T/2*

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} X(t)\cos(\frac{2\pi fnt}{T})dt \qquad \text{โดยที่ } n = 1 \to \infty \qquad (3.3)$$

และ

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} X(t)\cos(\frac{2\pi fnt}{T})dt \qquad \text{โดยที่ } n = 1 \to \infty \qquad (3.4)$$

นอกจากนั้นเราสามารถเขียนฟังก์ชั่นคาบใดๆให้อยู่ใน อนุกรมฟูเรียร์รูปเชิงซ้อน (Complex Form of the Furier series) ได้ในรูปต่อไปนี้

$$x(t) = A_0 + A_2 \sin(\alpha t + \phi) + A_2 \sin(2\alpha t + \phi) + \ldots \qquad (3.5)$$

โดยแต่ละลำดับของอนุกรมอาจสามารถเขียนได้ดังนี้

$$X(f_n) = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-j2\pi f_n t} dt \qquad \text{เมื่อ} \quad f_n = nf \qquad (3.6)$$

โดยที่ $e^{-j2\pi f_n t}$ เป็นเวกเตอร์หนึ่งหน่วย และ $x(t)$ ใดๆ เราสามารถเขียนในรูปเชิงซ้อนได้ดังนี้

$$x(t) = \sum_{n=-\infty}^{\infty} X(f_n) e^{-j2\pi f_n t} \qquad \text{และ} \qquad f_{-n} = -f_n \tag{3.7}$$

3.3 การแปลงฟูเรียร์ (Furier Transform)

ในการวิเคราะห์ฟูเรียร์นั้น สำหรับสัญญาณที่เป็นคาบ เราจะได้ว่าผลการวิเคราะห์จะอยู่ในรูปของอนุกรมของแต่ละความถี่ซึ่งเป็นจำนวนเท่าของความถี่หลักมูลเท่านั้น (Series of discrete frequency component) ถ้าเรานำมาประยุกต์ใช้กับสัญญาณซึ่งต่อเนื่องทั่วไปซึ่งไม่จำเป็นต้องเป็นสัญญาณที่เป็นคาบ โดยขยายการการอินทิเกรทในช่วงคาบ T ให้กว้างมาก ๆ หรือเป็นช่วง ∞ นั่นเอง ดังนั้นช่วงกว้างของแต่ละความถี่จึงเข้าใกล้ศูนย์ ดังนั้นเราจะได้ ฟังก์ชัน $X(f)$ เป็นฟังก์ชันต่อเนื่องหรือเราอาจเขียนใหม่ได้เป็น

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \tag{3.8}$$

โดยที่ $x(t)$ เป็นฟังก์ชันโดเมนเวลาซึ่งต่อเนื่อง และเรียก $X(f)$ ว่าเป็นการแปลงฟูเรียร์ของโดเมนเวลา $x(t)$ ในทางกลับกัน เราจะได้

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{-j2\pi f t} dt \tag{3.9}$$

เรียก $X(t)$ ว่าเป็นอินเวอร์สการแปลงฟูเรียร์ของฟังก์ชันโดเมนความถี่ $X(f)$ โดยค่า $X(f)$ จะอยู่ในรูปเชิงซ้อนซึ่งอาจเขียนได้ดังนี้

$$X(f) = Re\, X(f) + j\, Im\, X(f) \tag{3.10}$$

ส่วนจริงของ $X(f)$ หาได้จาก

$$\begin{aligned} Re\, X(f) &= 0.5\{X(f) + X(-f)\} \\ &= \int_{-\infty}^{\infty} x(t) \cos 2\pi f t dt \end{aligned} \tag{3.11}$$

และส่วนจินตภาพของ $X(f)$ หาได้จาก

$$Im\,X(f) = 0.5\{X(f)+X(-f)\}$$

$$= \int_{-\infty}^{\infty} x(t)\sin 2\pi ftdt \qquad (3.11)$$

ดังนั้นสเปกตรัมของแอมปลิจูด (Amplitude spectrum) ของแต่ละความถี่หาได้จาก

$$|X(f)| = \sqrt{(\mathrm{Re}\,X(f))^2 + (\mathrm{Im}\,X(f))^2} \qquad (3.15)$$

และสเปกตรัมของมุมเฟส (Phase spectrum) หาได้จาก

$$\phi(f) = \tan^{-1}\left(\frac{\mathrm{Im}\,X(f)}{\mathrm{Re}\,X(f)}\right) \qquad (3.16)$$

3.4 การแปลงฟูเรียร์แบบเต็มหน่วย (Discrete Furier Transform)

การแปลงฟูเรียร์แบบเต็มหน่วยเป็นการแปลงฟูเรียร์สำหรับข้อมูลในโดเมนเวลาที่ไม่ต่อเนื่อง(ค่าที่ผ่านการสุ่ม) ให้เป็นค่าในโดเมนความถี่ไม่ต่อเนื่องเช่นกัน ผลการแปลงฟูเรียร์ที่ได้จะประกอบด้วยส่วนประกอบย่อยรวมกันดังนี้

$$X(f_k) = \frac{1}{N}\sum_{n=-0}^{N-1} x(t_n)e^{-j2\pi kn/N} \qquad (3.15)$$

เรียกการแปลงฟูเรียร์ลักษณะนี้ว่า *การแปลงฟูเรียร์แบบเต็มหน่วย* (Discrete Furier Transform: DFT)

$$x(t_n) = \sum_{k=0}^{N-1} X(f_n)e^{-j2\pi kn/N} \qquad (3.16)$$

และเรียกการแปลงลักษณะนี้ว่า *อินเวอร์สการแปลงฟูเรียร์แบบเต็มหน่วย* (Inverse Discrete Furier Transform: IDFT)

คู่การแปลงฟูเรียร์ที่ได้จะอยู่ในรูปเต็มหน่วย (Discrete from)ซึ่งเป็นรูปแบบที่เหมาะสมในการนำมาคำนวณโดยการประมวลผลแบบดิจิตอลต่อไปจากสมการที่ 3.15 เราอาจเขียนผลการแปลงในรูปใหม่ได้ดังนี้

$$X(f_k) = \frac{1}{N}\sum_{n=0}^{N-1} x(t_n)W^{kn} \qquad (3.17)$$

โดยที่

$$W = e^{-j2\pi/N} \qquad (3.18)$$

3.5 การแปลงฟาสท์ฟูเรียร์ (Fast Fourier Transform : FFT)

โดยทั่วไปการคำนวณ DFT สำหรับสัญญาณเข้าที่ยาว N ลำดับนั้น คอมพิวเตอร์จะต้องทำการคูณจำนวนเชิงซ้อนถึง *NxN* ครั้ง และบวกจำนวนเชิงซ้อนอีก *N(N-1)* ครั้ง ซึ่งโดยทั่วไปคอมพิวเตอร์จะทำการบวกได้ง่ายและเร็วกว่าการคูณมากดังนั้นอาจกล่าวได้ว่า ความเร็วในการคำนวณ DFT ขึ้นอยู่กับจำนวนครั้งการคูณเป็นสำคัญ

เราจะกล่าวถึงวิธีการซึ่งสามารถลดจำนวนครั้งของการการคูณลงได้เหลือ Mog_2N ครั้ง หรือจำนวนครั้งลดไปถึง $N/(\log_2 N)$ เท่าซึ่งเรียกวิธีการนี้ว่าการแปลงฟาสท์ฟูเรียร์ (Fast Fourier Transform หรือ FFT)

3.5.1 การแปลงฟาสท์ฟูเรียร์แบบฐานสอง (Radix 2 FFT)

- *หลักการเบื้องต้นของ FFT*

เพื่อความสะดวกเราอาจละพจน์ *1/N* ในการคำนวณ DFT ดังนั้นการแปลงฟูเรียร์แบบเต็มหน่วยสำหรับ ลำดับ *x(m)* ที่ยาว *N* จุด ที่นิยามคือ

$$X(k) = \sum_{m=0}^{N-1} x(m).W^{mk} \qquad (3.20)$$

โดยที่ k_m = 0,1,...,N - 1 และจำนวนเชิงซ้อน W = $e^{-j2\pi/N}$ และ x(m) , X(k) เป็นสัญญาณในโดเมนเวลาและโดเมนความถี่ตามลำดับ

เราสามารถเขียนสมการ (3.23) ในรูปของสมการเมตริกซ์ได้เป็น

$$\{X\} = \{A\}\cdot\{x\} \qquad (3.21a)$$

โดยที่ $\{X\}$ และ $\{x\}$ เป็นเวกเตอร์แนวตั้ง (column vector) ที่ประกอบด้วยสมาชิก $X(k)$ และ $x(m)$ ตามลำดับจำนวน N ลำดับ และ $\{A\}$ เป็นเมตริกซ์จตุรัส (square matrix) ขนาด NxN ที่มีสมาชิกเป็นจำนวนเชิงซ้อน W^{mk} เช่นถ้าพิจารณากรณีที่ $N = 4$ เราสามารถเขียนแยกออกได้เป็น

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (3.21b)$$

แต่เนื่องจากคุณสมบัติความเป็นคาบของ W คือ

$$W^{mk} = W^{[mk \bmod(N)]} \qquad (3.22)$$

สมการ (3.21b) อาจเขียนได้เป็น

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (3.23)$$

คุณสมบัติความเป็นคาบทำให้เราต้องแยกตัวประกอบของเมตริกซ์ ออกเป็นเมตริกซ์ย่อยหลายเมตริกซ์คูณกัน และสมาชิกภายในเมตริกซ์ย่อยให้มีค่าเป็นศูนย์มากที่สุด วิธีการแยกตัวประกอบนี้จะไม่กระทำโดยตรงจาก แต่จะมีการสลับตำแหน่งหรือจัดกลุ่มของเมตริกซ์ด้วยวิธีการกลับบิต (bit reversal) และเมตริกซ์หลังจัดการสลับแถวแล้วนำมาแยกตัวประกอบอีกครั้ง

วิธีการแยกตัวประกอบอาจทำโดยวิธีการใช้ตัวเลขฐานสอง โดยการแทนดรรชนี k และ m ของสมการ (3.20) ด้วยเลขฐานสอง กรณีที่ ดรรชนี k และ m จะมีค่าได้เพียง 0,1,2, และ 3 เท่านั้น ดังนั้น

$$k = (k_1, k_0) \quad , \quad m = (m_1, m_0) \qquad (3.24)$$

โดยที่ k_1, k_0, m_0 และ m_0 เป็นเลขโดดที่มีค่าแค่ 0 และ 1 เท่านั้น ดังนี้

$$k = 2k_1, k_0 \quad , \quad m = 2m_1, m_0 \tag{3.25}$$

แทนค่า และ ลงในสมการ (3.20)

$$X(k_1, k_0) = x(m_1, m_0)\, W^{2m1+m0).(2k+k0)} \tag{3.26}$$

โดยคุณสมบัติความเป็นคาบของ W และ $W^{4m1k1}=1$ ดังนั้นสมการ (3.26) ได้ใหม่เป็น

$$X(k1,k0) = \sum_{m_0=0}^{1}\left\{\sum_{m_1=1}^{1} x(m_1,m_0)W^{(2m_1k_0)}\right\}W^{(2k_1+k_0)m_0} \tag{3.27a}$$
$$= \sum_{m_0=0}^{1}\{x(k_0,m_0)W^{(2k_1+k_0)m_0}$$

โดยสมมติให้ตัวแปร เป็นการคำนวณระหว่างกลาง ผลของสมการ (3.30a) เขียนเป็นเมตริกซ์ใหม่ ๆ ได้เป็น

$$\begin{bmatrix} x(0,0) \\ x(1,0) \\ x(0,1) \\ x(1,1) \end{bmatrix}^{(k_1,k_2)} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^0 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0,.0) \\ x(0,.1) \\ x(1,0) \\ x(1,1) \end{bmatrix}^{(m_1,m_2)} \tag{3.27b}$$

ซึ่งมีผลลัพธ์ระหว่างกลาง และ ผลลัพธ์ เป็น

$$\begin{bmatrix} x(0,0) \\ x(1,0) \\ x(0,1) \\ x(1,1) \end{bmatrix}^{(k_1,k_2)} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0,.0) \\ x(0,.1) \\ x(1,0) \\ x(1,1) \end{bmatrix}^{(m_1,m_2)} \tag{3.28a}$$

และค่า DFT ของสัญญาณเป็น

$$\begin{matrix}(k_1,k_2)\end{matrix}\begin{bmatrix}x(0,0)\\x(1,0)\\x(0,1)\\x(1,1)\end{bmatrix}=\begin{bmatrix}1 & W^0 & 0 & 0\\1 & W^2 & 0 & 0\\0 & 0 & 1 & W^1\\0 & 0 & 1 & W^0\end{bmatrix}\begin{matrix}(m_1,m_2)\end{matrix}\begin{bmatrix}x(0,.0)\\x(0,.1)\\x(1,0)\\x(1,1)\end{bmatrix} \qquad (3.28b)$$

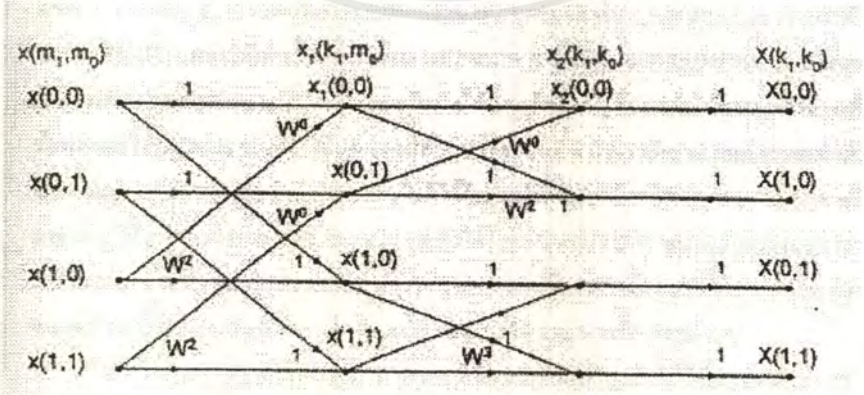
จะเห็นได้ว่าจากผลการแยกตัวประกอบของ $\{A\}$ ได้ว่า สมาชิกตามแนวนอนของตัวประกอบเมตริกซ์ จะมีเพียง 2 ตัวเท่านั้นที่มีค่าไม่เป็นศูนย์ และในสองตัวนี้จะมีสมาชิกหนึ่งตัวมีค่าเป็นศูนย์เสมอ ในขณะที่อีกตัวจะเป็นจำนวนเชิงซ้อน ซึ่งเราต้องการคูณจำนวนเชิงซ้อนเพียง $M\log 2N = 8$ ครั้ง และบวกจำนวนเชิงซ้อนอีก 8 ครั้ง ในขณะที่การคำนวณปกติใช้การคูณจำนวนเชิงซ้อน 16 ครั้ง และบวกจำนวนเชิงซ้อน 12 ครั้ง เรายังสามารถลดจำนวนการคูณลงได้อีก จากการคำนวณสัญญาณระหว่างกลาง $x_1(0,0)$ และ $x_1(0,1)$ ซึ่ง

$$x_1(0,0) = x(0,0) + W^0 .x(1,0) = x(0,0) + W^0 .x(1,0) \qquad (3.29)$$

และ $$x_1(1,0) = x(0,0) + W^2 .x(1,0) = x(0,0) - W^0 .x(1,0)$$

เนื่องจากคุณสมบัติของจำนวนเชิงซ้อน $W^2 = -W^0$ ทำให้การคำนวณ $x_1(0,1)$ ต้องการการคูณจำนวนเชิงซ้อนเพียงครั้งเดียวเท่านั้น ซึ่งทำได้โดยการคำนวณพจน์ $W^0 .x(1,0)$ ก่อนแล้วนำไปบวกและลบกับพจน์ x(0,0) เพื่อให้ได้ลำดับ $x_1(0,0)$ และ $x_1(1,0)$ ตามลำดับ

เราอาจแสดงวิธีการคำนวณ FFT โดยแสดงเป็นกราฟการไหล ดังรูป 3.1 โดยหัวลูกศรชี้ทิศทางการคำนวณ ส่วนอักษรกำกับเป็นค่าตัวคูณค่าของสัญญาณที่ต้นลูกศรนั้น และที่บัพ (Node) เป็นการรวมหรือบวกกันของสัญญาณ ส่วน $x_1(k_0,m_0)$ แทนการคำนวณระหว่างกลางและ $X(k_1,k_0)$ เป็นค่า DFT ของลำดับสัญญาณ



รูปที่ 3.1 กราฟการไหลแสดงถึงวิธีการคำนวณตามสมการ 3.31

- *ขั้นตอนการลดทอนทางเวลา*

วิธีการที่เสนอมาแล้วนั้น จะเป็นการจัดกลุ่มลำดับสัญญาณในโดเมนเวลา $x(m)$ ที่มีขนาด N จุด ออกเป็นสองอันดับสัญญาณที่มีความยาว N/2 จุดเท่ากัน โดยเรียกว่าลำดับสัญญาณคู่และลำดับสัญญาณคี่ โดยที่ลำดับสัญญาณคู่เกิดจากการเอาลำดับสัญญาณในตำแหน่งเลขคู่มาเรียงกัน ที่เหลือเป็นลำดับสัญญาณคี่ ดังนั้นถ้าเราให้ $x_E(m)$ เป็นลำดับคู่ และ ลำดับคี่เป็น $x_o(m)$ เพราะฉะนั้น

$$x_E(m) = x(2m) \quad ; \quad m = 0,1...(N/2) - 1 \tag{3.33}$$
$$x_o(m) = x(2m+1) \; ; \quad m = 0,1...(N/2) - 1$$

และถ้าเราให้ W_N แทน W ของลำดับ ที่ยาว N จุด ทำให้การคำนวณการแปลง DFT ของลำดับสัญญาณ $x(m)$ ที่ยาว N จุดเขียนได้ใหม่เป็น

$$X(k) = \underbrace{\sum_{m=0}^{N-1} x_\varepsilon(m)(W_N)^{loss}}_{\text{m=เลขคู่}} + \underbrace{\sum_{m=(}^{N-1} x_o(m)(W)^{loss}}_{\text{m=เลขคี่}} \tag{3.31}$$
$$= \sum_{n=0}^{(N/2)-1} x(2m)(W_N)^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1)(W_N)^{(2m+1)k}$$

โดยพจน์ $(W_N)^2 = W_{N/2}$ ซึ่งหมายถึงค่า W ของลำดับซึ่งยาว $N/2$ ดังนั้นจะเขียนใหม่เป็น

$$X(k) = \sum_{m=0}^{N/2-1} x_\varepsilon(m)(W_{N/2})^{loss} + (W_N)^k \sum_{m=0}^{N/2-1} x_0(m)(W_{N/2})^{loss} \tag{3.35}$$

$$X(k) = X_1(k) + (W_N)^k X_2(k)$$

โดยที่ $X_1(k)$ และ $X_2(k)$ ผลการแปลง DFT ขนาด $N/2$ จุดของลำดับ $x_E(m)$ และ $x_o(m)$ ตามลำดับ สมการที่ (3.35) แสดงให้เห็นว่าการคำนวณ DFT ขนาด N จุดนั้นสามารถแบ่งการคำนวณย่อยออกเป็นการคำนวณ DFT ขนาด $N/2$ จุด สองอันดับได้ และข้อสำคัญคือ จะทำให้การคูณจำนวนเชิงซ้อนลดลงไปประมาณ 50 เปอร์เซ็นต์ โดยหลักการเดียวกันนี้ทำให้เรายังสามารถแบ่งทอนลำดับ $x_E(m)$ และ $X_o(m)$ ออกเป็นลำดับคู่และลำดับคี่ได้อีก จนในที่สุดเหลือเพียงลำดับขนาด 2 จุด หรืออาจกล่าวได้ว่า การคำนวณ DFT ขนาด N จุด ทำได้โดยการแปลง DFT ขนาด 2 จุด จำนวน $N/2$ ภาคด้วยกัน ข้อสังเกตคือการซอยเพื่อแบ่งลำดับ $x(n)$ ออกเป็นทีละครึ่งจนเหลือการคำนวณ DFT ขนาด 2 จุด นี้ สำหรับสัญญาณ N ลำดับ จะทำการแบ่งได้ $\log_2 N$ ครั้ง (ดังรูป 3.3)

การนำ DFT ขนาด 2 จุด จำนวน $N/2$ ภาคนี้มาประกอบกันเพื่อให้ได้การคำนวณ DFT ขนาด N จุดนั้น จะต้องมีหลักเกณฑ์ในการทำเพื่อไม่ให้ค่าที่ได้ผิดพลาดไป ดังนั้นเราต้องทำการนิยามสมการ (3.32) สำหรับ $k>N/2$ ด้วยซึ่งทำได้โดยการเขียน

$$X(k) = X_1(k) + (W_N)^k X_2(k) \qquad ;0 \le k \le (N/2)-1 \qquad (3.33)$$

$$= X_1(k - N/2) + (W_N)^k X_2(k - n/2) \quad ;N/2 \le k \le N/2-1$$

พจน์ $(W_N)^k$ ในสมการ 3.32 เรียกว่าตัวประกอบการหมุน (twiddle factor) ซึ่งมีความสำคัญในการนำ DFT ขนาด 2 จุด หรือ DFT ขนาด $N/2$ จุดมาประกอบกันเป็น DFT ขนาด N จุดได้เหมือนเดิม และจากความสัมพันธ์ $(W_N)^{k-N/2} = -(W_N)^k$ เราจะได้

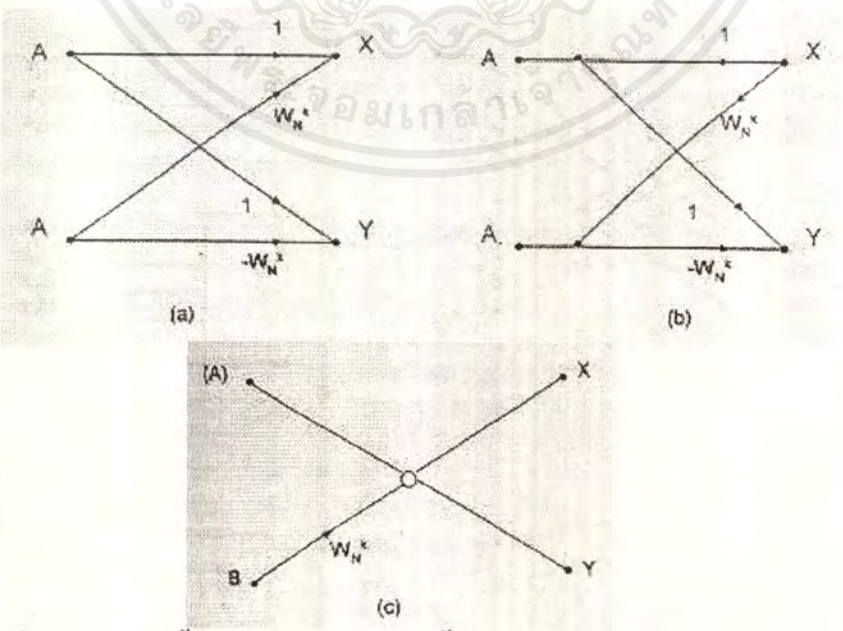
$$X(k) = X_1(k) + (W_N)^k X_2(k) \qquad ;0 \le k \le (N/2)-1 \qquad (3.34a)$$

$$= X_1(k - N/2) + (W_N)^{k-N/2} X_2(k - N/2) \;;N/2 \le k \le N/2-1 \qquad (3.34b)$$

ตามสมการ 3.37 ทำให้เราทราบว่าในการคำนวณหา DFT ของลำดับคู่หนึ่ง จะประกอบด้วยลำดับ $X(k)$ ในสมการ 3.34a และลำดับ $X(k)$ ในสมการ 3.34b ซึ่งจะห่างออกไปจากลำดับ $X(k)$ ในสมการ 3.34a ไป $N/2$ จุดนั้น สามารถคำนวณได้โดยใช้สูตรการคูณจำนวนเชิงเส้นเพียงครั้งเดียวเท่านั้น จากผลอันนี้เราจะนำไปสร้างหน่วยความจำที่มีชื่อว่าหน่วย *ความจำผีเสื้อ (butterfly unit)* โดยหน่วยคำนวณนี้ (อาจอยู่ในรูปแบบของวงจรหรือโปรแกรม) มีข้อมูลเข้าสองข้อมูลคือ A และ B และให้ข้อมูลออกเป็น X และ Y เป็น

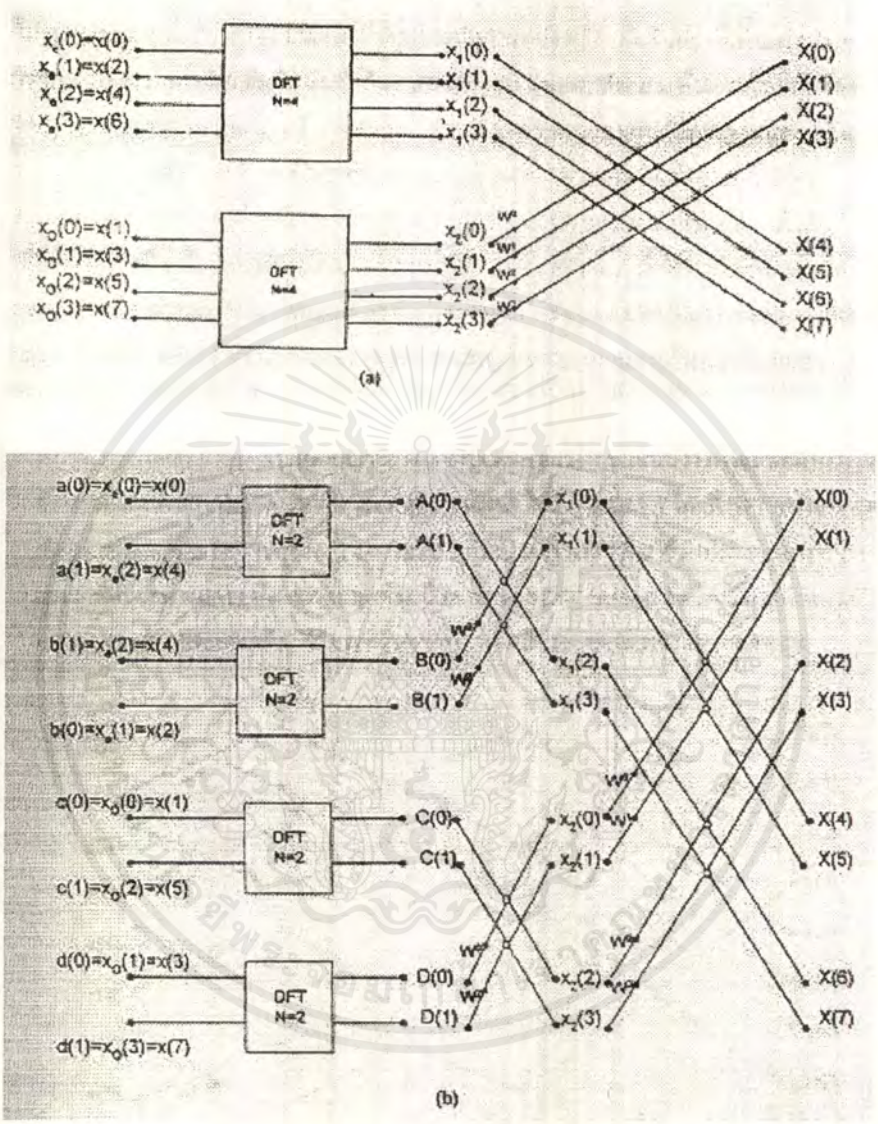
$$X = A + (W_N)^k.B$$

$$Y = A - (W_N)^k.B \qquad (3.38)$$



รูปที่ 3.2 หน่วยผีเสื้อของการคำนวณตามขั้นตอนวิธีลดทอนทางเวลา

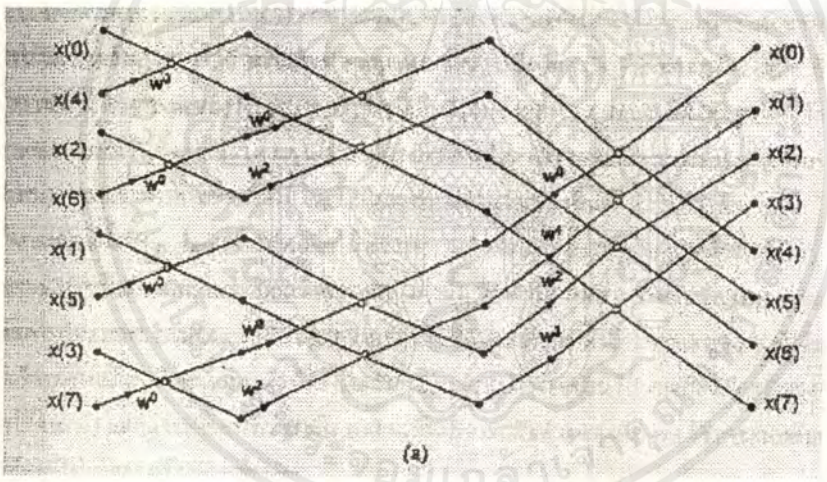
โดยที่การทำงานของหน่วยผีเสื้อ แทนได้ด้วยกราฟการไหล ดังแสดงไว้ในรูป 3.2(a) 3.2(b) หรือ 3.2(c) โดยรูปที่ 3.3 แสดงการคำนวณ DFT แบบ 8 จุด



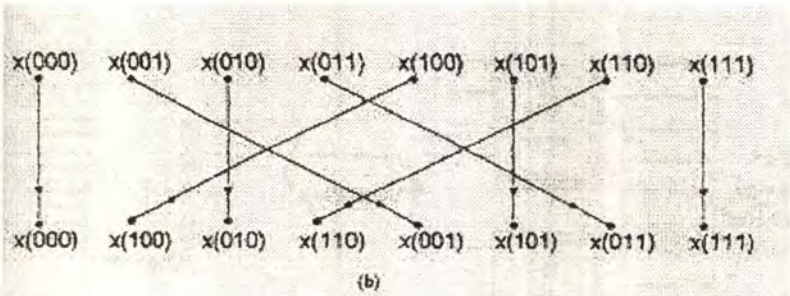
รูปที่ 3.3 (a) และ (b) แสดงขั้นตอนแบบ การลดทอนทางเวลา สำหรับ DFT แบบ 8 จุด

ตามรูป 3.4(a) ลำดับสัญญาณเข้า *x(n)* ไม่ได้ถูกจัดเรียงอย่างต่อเนื่อง หรือ ตามธรรมชาติแต่ไม่ถูก สลับตำแหน่งกันอย่างมีหลักเกณฑ์ คือการสลับตำแหน่ง หรือสลับอันดับ กันนี้จะเป็นไปตามวิธรการที่เรียกว่า การผันกลับบิต นั่นคือถ้าเราแทนดรรชนี n ของลำดับ *x(n)* ด้วยเลขฐานสองโดยที่จำนวนบิตต้องเพียงพอที่จะแทนค่า *N* ได้ เช่น ในกรณี *N=8* ที่ต้องแทนกันด้วยเลขฐานสอง 3 บิต จากนั้นการจัดลำดับ *x(n)* ใหม่จะได้จากการผันกลับบิตของเลขฐานสองที่แทนดรรชนี *n*

ดังรูป 3.4(b) คือ *x(001)* จะถูกแทนด้วย *x(100)* และ *x(110)* และ *x(110)* ถูกแทนด้วย *x(011x)* เป็นต้น เนื่องจากดรรชนี *n* เป็นดรรชนีในโดเมนเวลา และวิธีการของ FFT แบบนี้เป็นการลดทอนเวลาทางการคำนวณโดยการ สับ หรือ ตัดทอน ลำดับในโดเมนเวลา หรือ *x(n)* ออกเป็นกลุ่มย่อย โดยแต่ละกลุ่มประกอบด้วยลำดับ *x(n)* เพียงสองลำดับที่เป็น*ปมคู่*กัน การจัดกลุ่มนี้คล้ายกับเป็นการสุ่มตัวอย่างลำดับเดิมอีกครั้งหนึ่ง ด้วยอัตราการสุ่มตัวอย่างที่ต่ำกว่า และถ้าหากเราถือว่าแต่ละกลุ่มข้อมูลใหม่ที่จัดทำขึ้นมา ต่างเป็นลำดับ ข้อมูลชุดหนึ่งแล้ว ก็เท่ากับว่าเราได้ตัดทอนลำดับในโดเมนเวลาลงไปเป็นกลุ่มลำดับข้อมูลย่อยหลายลำดับ ดังนั้นจึงเรียกวิธีการแบบนี้ว่า การลดทอนทางเวลา ซึ่งแสดงแผนภาพลำดับวิธีการของ การลดทอนทางเวลา ได้ดังรูป 3.4.1.5 โดยที่ DFT ขนาด *N* จุด เดิมถูกแบ่งออกเป็น DFT ขนาด *N/2* จุดจำนวน 2 ภาคนำมารวมกันโดยใช้ตัวประกอบการหมุน และลำดับนี้จะทำจนกระทั่งผลลำดับสุดท้ายเป็นการแปลง DFT ขนาด 2 จุด โดยที่การนำเอา DFT ขนาด *N/2* จุดมาประกอบกัน มีวิธีการที่ขึ้นอยู่กับการจัดเรียงตัวของตัวประกอบการหมุน



รูปที่ 3.4(a) กราฟการไหลสัญญาณแสดงการคำนวณตามรูป 3.3



รูปที่ 3.4 (b) แสดงการสลับตำแหน่งของลำดับ x(n) ด้วยการผันกลับบิต

- *การลดทอนทางเวลาโดยวิธีการซ้ำที่*

คุณสมบัติที่เป็นข้อดีบางประการของการแปลงฟาสท์ฟูเรียร์ ตามรูป 3.4(a) โดยทั่วไปแล้วอาจกล่าวได้ว่าการคำนวณการแปลง DFT นั้นแท้จริงแล้วก็คือ การนำลำดับจำนวนเชิงซ้อนที่มีอยู่ N ลำดับ มาทำการแปลงเป็นจำนวนเชิงซ้อนอีกกลุ่มหนึ่งที่มีอยู่ N ลำดับเช่นกัน โดยการใช้ FFT การแปลงเช่นนี้จะกระทำ $\log_2 N$ ขั้นตอนด้วยกัน ดังนั้นตามรูป 3.4(a) ซึ่ง $N=8$ ควรต้องมีหน่วยความจำ ($\log_2 N$) หรือ 3 แถวลำดับ (array) ด้วยกันสำหรับเก็บข้อมูลที่ต้องใช้ในการคำนวณโดยที่แถวลำดับแรกไว้เก็บลำดับข้อมูล $x(n)$ สองแถวลำดับ $x_1(k)$ และแถวลำดับสุดท้ายสำหรับผลลัพธ์ $X(k)$ โดยการคำนวณจะประกอบด้วยหน่วยผีเสื้อ ซึ่งลักษณะการคำนวณของหน่วยผีเสื้อนี้ หากเรามีหน่วยความจำต่างหากไว้สำหรับเก็บค่าผลคูณของจำนวนเชิงซ้อน $(W_N)^k.B$ ผลลัพธ์ X กับ Y ที่คำนวณสามารถเก็บแทนที่ไว้ในหน่วยความจำที่เก็บลำดับข้อมูลเข้า A และ B ได้ (ดังรูป 3.2) โดยลักษณะการทำเช่นนี้จะเห็นว่า ตามรูป 3.4(a) ผลลัพธ์การคำนวณทางขวามือสามารถบรรจุแทนที่ในหน่วยความจำทางด้านซ้ายมือได้โดยไม่มีผลต่อการคำนวณส่วนอื่น ๆ ซึ่งเรียกว่าเป็น การคำนวณแบบซ้ำที่ (in place) ซึ่งมีข้อดีคือใช้หน่วยความจำเพียงหนึ่งแถวลำดับ หรือต้องการหน่วยความจำสำหรับเก็บจำนวนเชิงซ้อนเพียง $N+1$ ค่าเท่านั้น วิธีการนี้จะเหมาะสำหรับข้อมูลยาวมาก โดยการคำนวณจะไม่เปลืองเนื้อที่หน่วยความจำ ข้อดีอีกประการคือ ตัวประกอบหมุน $(W_N)^k$ นั้นจะถูกเรียงอย่างเป็นลำดับคือจากกำลังน้อยไปสู่กำลังมาก จึงทำให้การเขียนโปรแกรม หรือสร้างวงจรทำได้ง่ายกว่า แต่ก็มีข้อเสียตรงที่ว่าลำดับ $x(n)$ จะต้องมีการสลับตำแหน่งกันตาม วิธีการผันกลับบิต จึงต้องมีโปรแกรมหรือวงจรเพิ่มเติม

- *การแปลงฟาสท์ฟูเรียร์ชนิดลดทอนทางความถี่ (Decimation-in-Frequency :* DIF)

ลำดับการคำนวณ FFT ซึ่งใช้กันมากและประยุกต์ใช้ในโครงสร้างนี้คือ การลดทอนทางความถี่ ซึ่งมีหลักการคล้ายคลึงกับ การลดทอนทางเวลา โดยที่การลดทอนทางความถี่จะแบ่งลำดับโดเมนเวลา $x(m)$ ออกเป็นสองส่วนเท่า ๆ กัน โดยการแบ่งครึ่งซึ่งทำได้โดย ถ้าให้ $x_E(m)$ และ $x_o(m)$ แทนลำดับสัญญาณที่ได้จากการแบ่งครึ่งนี้

$$x_E(m) = x(m) \qquad ;m=0,1....,(N/2)-1$$
$$x_o(m) = x(m+N/2) \qquad ;m=0,1....,(N/2)-1 \qquad (3.36)$$

เพราะฉะนั้นการคำนวณ DFT ขนาด จุด ของ $x(n)$ สามารถเขียนแยกได้สองส่วนคือ

$$\begin{aligned} X(k) &= \sum_{m=0}^{(N/2)-1} x(m)(W_N)^{mk} + \sum_{m=N/2}^{N-1} x(m)(W_N)^{mk} \\ &= \sum_{m=0}^{(N/2)-1} x_E(m)(W_N)^{mk} + \sum_{m=N/2}^{(N/2)-1} x_0(m)(W_N)^{(m+N/2)k} \qquad (3.37) \\ X(k) &= \sum_{m=0}^{(N/2)-1} x_E(m)(W_N)^{mk} + (W)^{Nk/2} \sum_{m=N/2}^{(N/2)-1} x_0(m)(W_N)^{mk} \end{aligned}$$

เพราะว่าพจน์ $(W)^{NK/2} = e^{-j\pi k} = (-1)^k$ จึงเขียนได้เป็น

$$X(k) = \sum_{m=0}^{(N/2)-1} \left[x_E(m) + (-1)^k . x_0(m+N/2)\right](W_N)^{mk} \qquad (3.38)$$

และถ้าเราแยกดรรชนี k ออกเป็นเลขคู่และคี่ เพราะฉะนั้น X(2k) และ X(2k+1) จะแทน DFT จะแทนส่วนของเลขคู่และเลขคี่ตามลำดับหรือ

$$\begin{aligned} X(2k) &= \sum_{m=0}^{(N/2)-1} \left[x_E(m) + x_0(m)\right](W_N)^{2mk} \\ &= \sum_{m=0}^{(N/2)-1} f(m)(W_{N/2})^{mk} \qquad (3.42) \end{aligned}$$

และ

$$\begin{aligned} X(2k) &= \sum_{m=0}^{(N/2)-1} \left[x_E(m) + x_0(m)\right](W_N)^{2mk} \\ &= \sum_{m=0}^{(N/2)-1} f(m)(W_{N/2})^{mk} \qquad (3.40) \end{aligned}$$

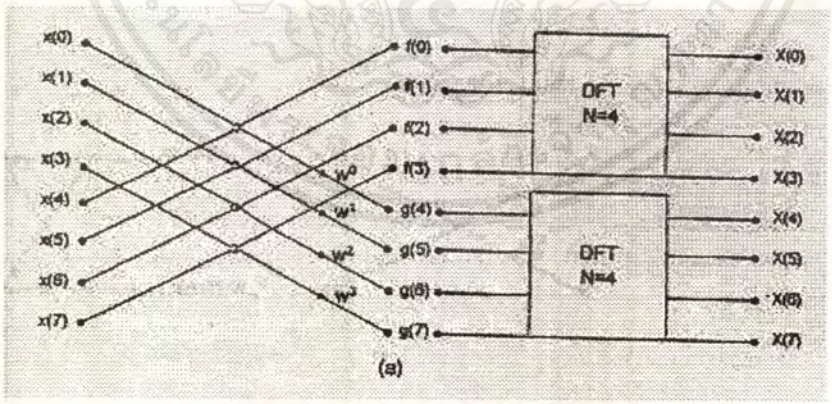
จากผลของ (3.39) และ (3.40) แสดงว่าการคำนวณ DFT ขนาด N จุด ด้วยวิธีนี้จะทำได้โดย ในเบื้องแรกจากลำดับ $x(m)$ เราทำการแบ่งครึ่งออกเป็น 2 ลำดับ ดังสมการ (3.39) แล้วนำมาสร้างเป็นลำดับอันใหม่ที่ยาว $N/2$ จุด 2 ลำดับ โดยสมมติให้ลำดับใหม่นี้ชื่อ $f(m)$ และ $g(m)$ ลำดับคู่นี้ สามารถสร้างได้โดยใช้สมการ

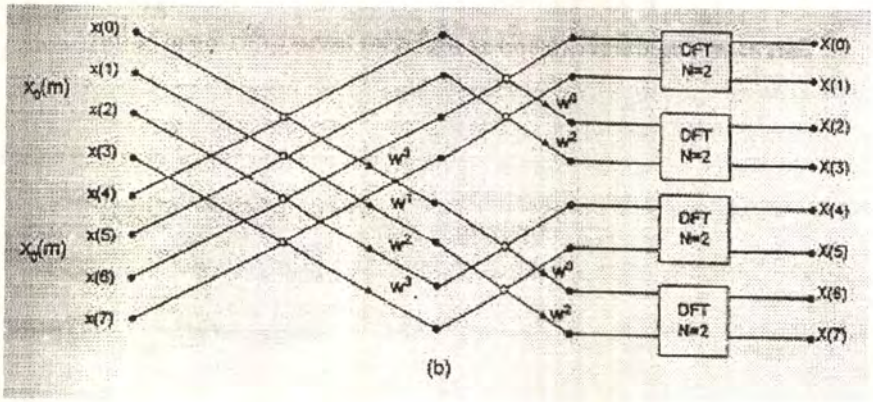
$$\begin{aligned} f(m) &= x_E(m) + x_o(m) \qquad ;m=0,1....,(M/2)-1 \\ g(m) &= \left[x_E(m) - x_o(m)\right].(W_N)^{mk}; \qquad (3.44) \end{aligned}$$

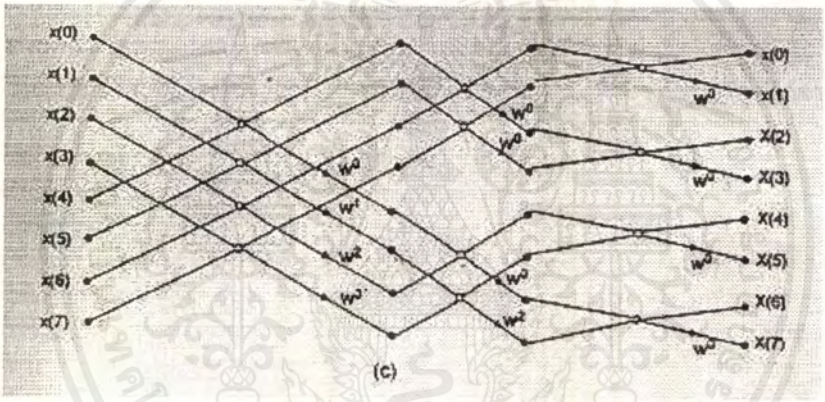
และจากลำดับยาว N/2 จุดในสมการ (3.44) เราก็อาจนำไปคำนวณหา DFT ขนาด N/2 จุด โดยใช้ (3.42) และ (3.43) เพราะฉะนั้น โดยรวมแล้วจะเห็นว่า การคำนวณ DFT ขนาด N จุด ได้ถูกแบ่งเป็นการคำนวณ DFT ขนาด N/2 จุดสองภาคด้วยกัน

จะเห็นว่าเหมือนกับในการลดทอนทางเวลา การแบ่งการคำนวณย่อยออกเป็น DFT ขนาด N/2 จุดนี้สามารถแบ่งย่อยออกไปได้เรื่อย ๆ จนในที่สุดเหลือการคำนวณขนาด 2 จุด ตัวอย่างที่แสดงการคำนวณโดยใช้กราฟการไหลกรณี N=8 แสดงไว้ในรูป 3.6 โดยในรูปแสดงการแบ่งลำดับย่อยออกตามลำดับนั้น จนเหลือการคำนวณ DFT ขนาด 2 จุด และการแบ่งย่อยทำได้ 3 ครั้ง หรือ $\log_2 8$ และจำนวนครั้งในการคูณจำนวนเชิงซ้อนจะประมาณ $N \log_2 N$ เช่นเดียวกัน

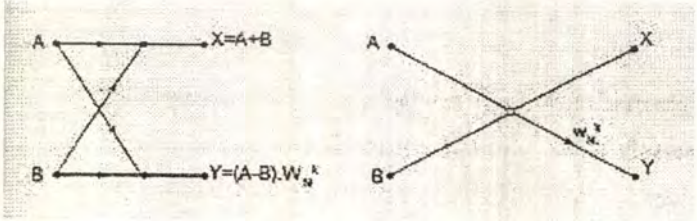
ข้อแตกต่างระหว่างสองวิธีขั้นตอนการคำนวณทั้งสองวิธีคือ ประการแรก การลดทอนทางเวลา $x(n)$ จะเรียงตามธรรมชาติ และ $X(k)$ เรียงตามธรรมชาติ ส่วนการลดทอนทางความถี่จะตรงข้ามคือ $x(n)$ จะเรียงตามธรรมชาติ และ $X(k)$ จะถูกเรียงสลับแบบผันกลับบิท ประการที่สอง หน่วยผีเสื้อของการลดทอนทางความถี่ต่างไปจากการลดทอนทางเวลาคือเพราะได้จากการเอาลำดับ $x_E(m)$ และ $x_o(m)$ มาบวกและลบกันก่อนแล้วจึงทำการคูณด้วยจำนวนเชิงซ้อน $(W_N)^k$ หน่วยคำนวณผีเสื้อของการลดทอนทางความถี่ แสดงดังรูป 3.6







รูปที่ 3.5 แสดงลำดับขั้นตอนวิธีการของ FFT ชนิดการลดทอนทางความถี่



รูปที่ 3.6 แสดงการคำนวณของ หน่วยผีเสื้อ ของขั้นตอนวิธีชนิดลดทอนทางความถี่

บทที่ 4
การวิเคราะห์สัญญาณโรเตอร์สล็อทฮาร์โมนิกส์ โดยใช้โปรแกรม MATLAB

4.1 บทนำ

เนื่องจากสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์มีพฤติกรรมที่เปลี่ยนแปลงตามความเร็วของมอเตอร์ ดังนั้นการศึกษาพฤติกรรม ลักษณะ องค์ประกอบของโรเตอร์สล็อทฮาร์โมนิกส์จึงอาจเป็นแนวทางหนึ่งในการใช้หาความเร็วของอินดัคชั่นมอเตอร์ได้ เนื่องจากลักษณะ พฤติกรรม องค์ประกอบโรเตอร์สล็อทฮาร์โมนิกส์ขึ้นอยู่กับปัจจัยและตัวแปรต่างมากมาย ดังนั้นในการศึกษาพฤติกรรมโรเตอร์สล็อทฮาร์โมนิกส์จำเป็นที่จะต้องทำการวิเคราะห์ในหลายๆ ด้าน เพื่อให้ทราบถึงพฤติกรรมที่แท้จริงของมัน ดังนั้นจึงมีความเหมาะสมที่จะใช้โปรแกรมMATLAB ด้วยเหตุผลดังต่อไปนี้

1. การประมวลผลของโปรแกรม MATLAB เป็นการประมวลผลสัญญาณเชิงดิจิตอล (Digital Signal Processing : DSP) ซึ่งช่วยให้ผลการทดลองมีความแม่นยำมากยิ่งขึ้น

2.โปรแกรม MATLAB มีฟังก์ชันต่างๆ ที่เกี่ยวกับการประมวลผลสัญญาณเชิงเลขมากพอสมควร ทำให้สะดวกในการวิเคราะห์การผลทดลอง

3. สามารถแก้ไขและปรับเปลี่ยน เพิ่มเติมวิธีการ แนวทางการทดลองได้ง่าย

4. สะดวกในการแสดงผล และวิเคราะห์การทดลอง เนื่องจากมีฟังก์ชันในการ แสดงผล และวิเคราะห์ อยู่มาก

ในรายงานฉบับนี้ใช้โปรแกรม MATLAB Version 5.1 ทำการทดลอง และวิเคราะห์ผล

เนื่องจากความถี่ของสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์มีการเปลี่ยนแปลงตามความเร็วของมอเตอร์ ดังนั้นในการหาค่าความเร็วของมอเตอร์ เราจะต้องทำการวิเคราะห์ให้ได้ว่าความถี่ของสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์ที่ความเร็วต่างๆ มีค่าเป็นเท่าใด โดยอาศัยการกระจายสเปกตรัม (Spectrum) ของค่าสัญญาณกระแสที่ได้จากการทดลองให้อยู่ในโดเมนความถี่ แล้วจึงทำการพิจารณาดูว่าสเปกตรัมตัวบ้างที่มีการเลื่อนตัวเมื่อความเร็วของมอเตอร์เปลี่ยนไป ซึ่งจะทำให้สามารถระบุค่าสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์ได้ว่ามีความถี่เป็นเท่าใด แล้วจึงทำการคำนวณหาค่าความเร็วมอเตอร์ต่อไป

4.2 รูปแบบของข้อมูลที่ได้จากการทดลอง

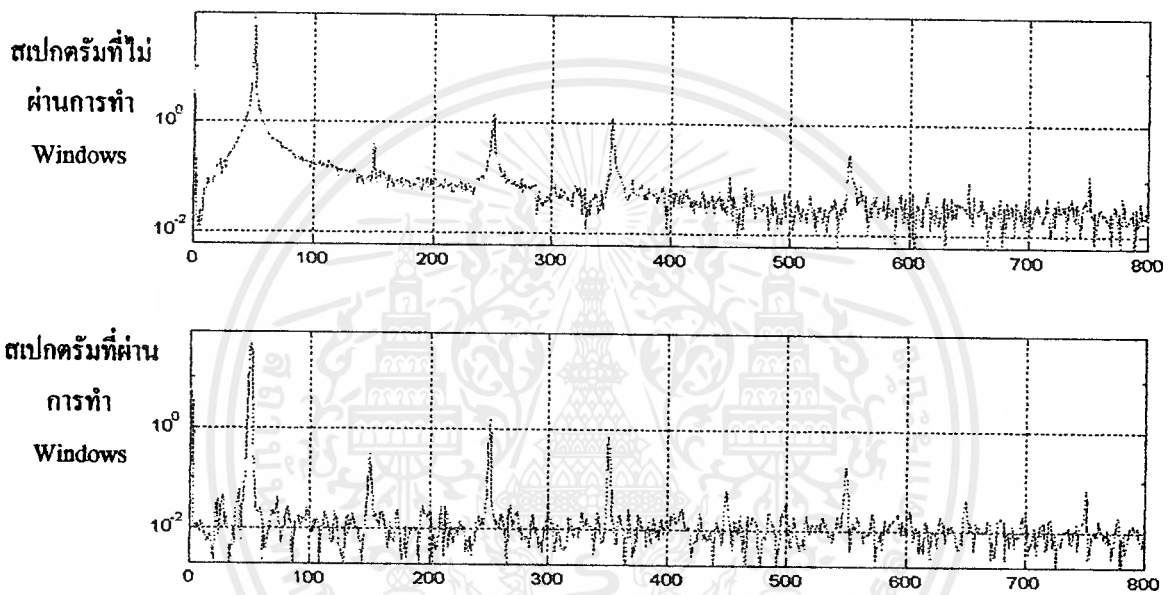
ในการการศึกษาพฤติกรรมโรเตอร์สลิ็อทฮาร์โมนิกส์โดยใช้โปรแกรม MATLAB นั้น จำเป็นที่จะต้องทำการประมวลผลสัญญาณเชิงดิจิตอล ดังนั้นเราจะต้องทำการแปลงสัญญาณกระแสที่ป้อนเข้ามอเตอร์ให้เป็นข้อมูลทางดิจิตอล โดยได้ใช้ Digital Storage Scope รุ่น HP 54520 Cเป็นตัวแปลง ซึ่งเราสามารถปรับเลือก ความยาวข้อมูล (Record Length) ความถี่ในการแซมปลิ้ง (Sampling Frequency) นอกจากนั้นยังสามารถในการกรองความถี่ต่ำผ่าน ซึ่งจะช่วยตัดสัญญาณรบกวนต่างๆ (Noise) บางส่วนออกไปได้บ้าง รูปแบบของข้อมูลที่บันทึกมาจะอยู่ในรูปแบบ Text File ที่มีรูปแบบดังรูป 4.1

Type: normal
Points: 8192
Count: 1
XInc: 9.99999974738E-05
XOrg: 1.12848499549080E-05
YData range: 8.06299E-02
YData center: -2.35300E-03
XRange: 5.00000E-02
YRange: 8.06299E-02
Date: 06 OCT 1998
Time: 90:56:55
Acq mode: real time
X Units: second
Y Units: Volt
Max Bandwidth: 500000000
Data:
2.43E-03
3.57E-03
4.62E-03
5.58E-03

Header

4.4 การทำวินโดวส์ (Windows) และการคอนโวลูชั่น(Convolution) ข้อมูล

จากข้อมูลทางดิจิตอลที่ได้มา หากเรานำมากระจายสเปกตรัมให้อยู่ในโดเมนความถี่โดยไม่ผ่านกระบวนการใด ๆ เลย จะทำให้ลักษณะการกระจายของสเปกตรัมอยู่ในรูปที่พิจารณาได้ยาก และซึ่งอาจทำให้เกิดความผิดพลาดในการวิเคราะห์ได้ เพื่อแก้ปัญหาดังกล่าวเราจึงต้องข้อมูลเล่านี้มาทำการจัดรูปแบบข้อมูลใหม่ โดยอาศัยการทำวินโดวส์และการคอนโวลูชั่น



รูปที่ 4.3 แสดงสเปกตรัมของข้อมูลไม่ผ่านการวินโดวส์ และผ่านการวินโดวส์

ซึ่งการทำวินโดวส์ก็คือ การทำให้ข้อมูลหรือสัญญาณมีผลตอบสนอง ลักษณะตามที่เราต้องการ โดยอาศัยการตัดปลาย หรือถ่วงน้ำหนักเพื่อให้ผลตอบสนองของอนุกรมเปลี่ยนแปลงในลักษณะที่เราต้องการ เปรียบเสมือนเรามีหน้าต่างรูปร่างต่างๆ เมื่อข้อมูลผ่านหน้าต่างนี้แล้วก็จะทำให้ผลตอบสนองของข้อมูลเปลี่ยนไป

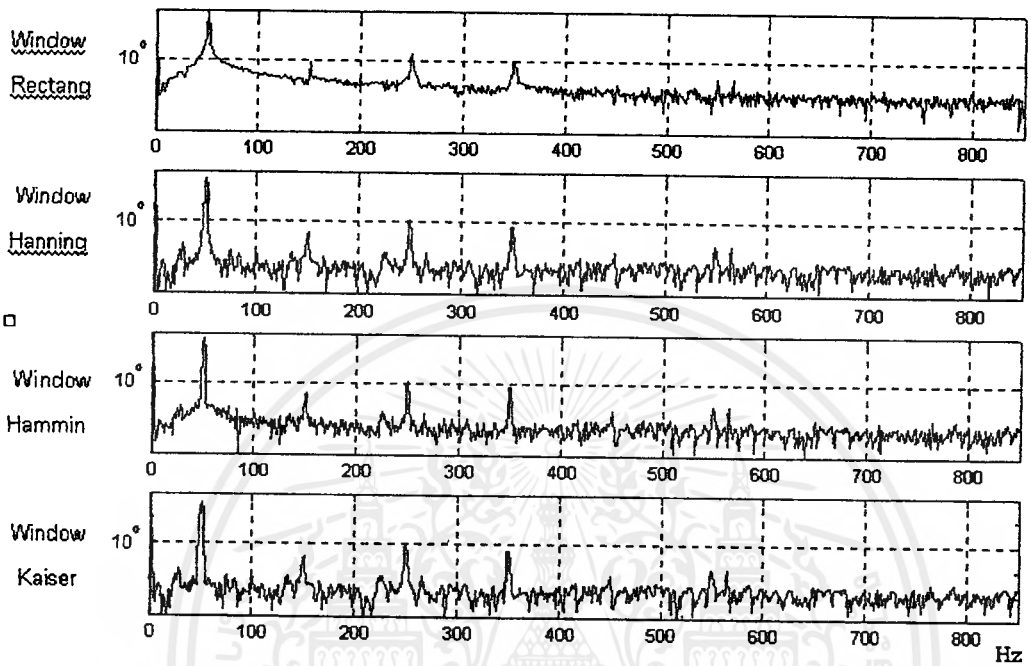
ถ้าให้ $h_d(n)$ คือลำดับการตอบสนองเชิงเวลาของข้อมูลก่อนทำการวินโดว์

$H_d(\omega)$ คือลำดับการตอบสนองเชิงความถี่ของข้อมูลก่อนทำการวินโดว์

$h(n)$ คือลำดับการตอบสนองเชิงเวลาของข้อมูลหลังทำการวินโดว์

$H(\omega)$ คือลำดับการตอบสนองเชิงความถี่ของข้อมูลหลังทำการวินโดว์

โดยส่วนใหญ่ในรายงานฉบับนี้จะใช้การทำวินโดวส์แบบ Hanning เนื่องจากวินโดวส์แบบนี้มีความแน่นอนทางความถี่สูง และไม่ลดขนาด (Magnitude) ของสัญญาณมากนัก



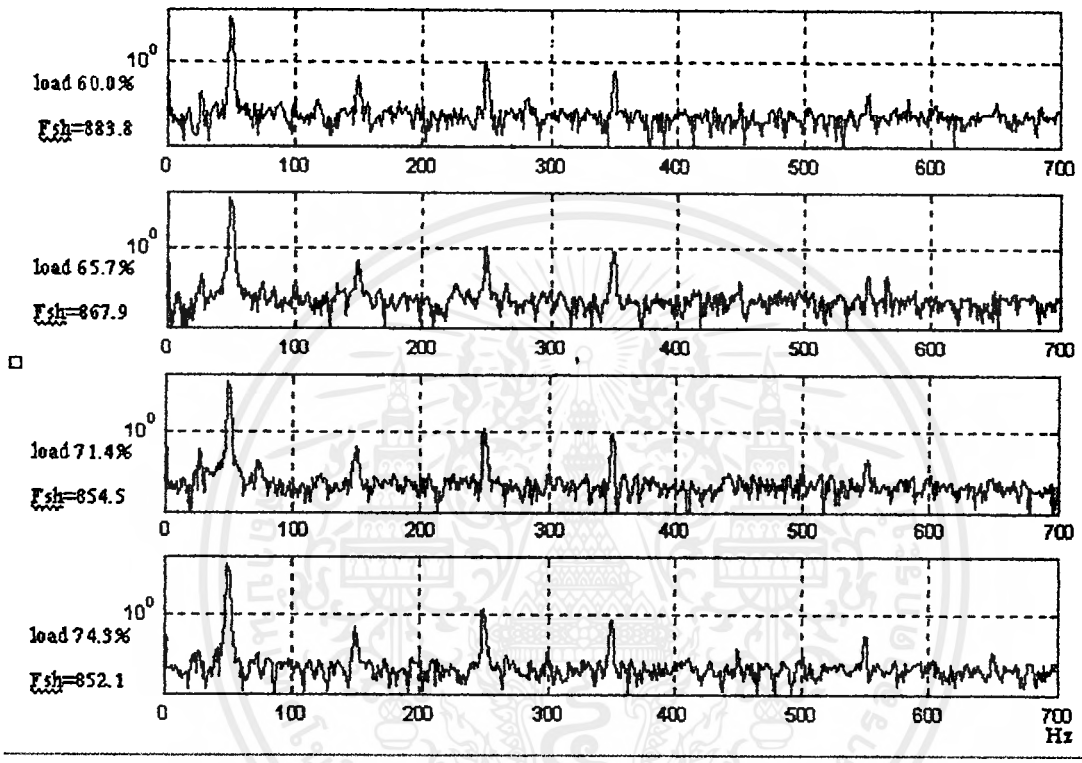
รูปที่ 4.4 แสดงสเปกตรัมของข้อมูลผ่านการวินโดวส์ แบบต่าง ๆ

4.5 การกระจายสเปกตรัมของข้อมูลโดยใช้ FFT ใน MATLAB

หลังจากที่ได้ทำการวินโดวส์ข้อมูล เพื่อปรับเปลี่ยนคุณสมบัติของข้อมูลแล้ว ก็จะนำข้อมูลที่ได้มากระจายสเปกตรัมโดยใช้การแปลงแบบ FFT ซึ่งใน MATLAB เองมีฟังก์ชันสำหรับการแปลง FFT อยู่แล้ว โดยข้อมูลที่จะมากระจายสเปกตรัมจะต้องมีจำนวนข้อมูลเป็น 2^M เมื่อ M เป็นจำนวนเต็มใดๆ มิฉะนั้น MATLAB จะทำการแปลงโดยใช้ DFT แทน

จากนั้นเราก็จะได้ผลลัพธ์จากการแปลง FFT ซึ่งเป็นลำดับสเปกตรัมของข้อมูล โดยจะอยู่ในรูปจำนวนเชิงซ้อน ที่มีขนาดความยาวข้อมูลเท่ากับความยาวข้อมูลก่อนทำการแปลง FFT จากผลลัพธ์ที่อยู่ในรูปจำนวนเชิงซ้อน เราจะต้องทำการคำนวณหาขนาด (Magnitude) ของผลลัพธ์ดังกล่าว เพื่อให้ได้สเปกตรัมของสัญญาณที่แท้จริง แล้วจึงนำขนาดของสัญญาณที่ได้มาสร้างกราฟเพื่อวิเคราะห์หาความถี่ของโรเตอร์สลื่อทฮาร์โมนิกส์ต่อไป

ในการสร้างกราฟเพื่อแสดงขนาดแมกนิจูดของสัญญาณในโดเมนความถี่ จะต้องมีการปรับปรุงสเกล (Scale) ของแกนให้ตรงกับค่าความถี่ตามความเป็นจริงซึ่งจะสัมพันธ์กับขนาดความยาวของข้อมูลที่ใช้ประมวลผล และความถี่แซมปลิ้ง



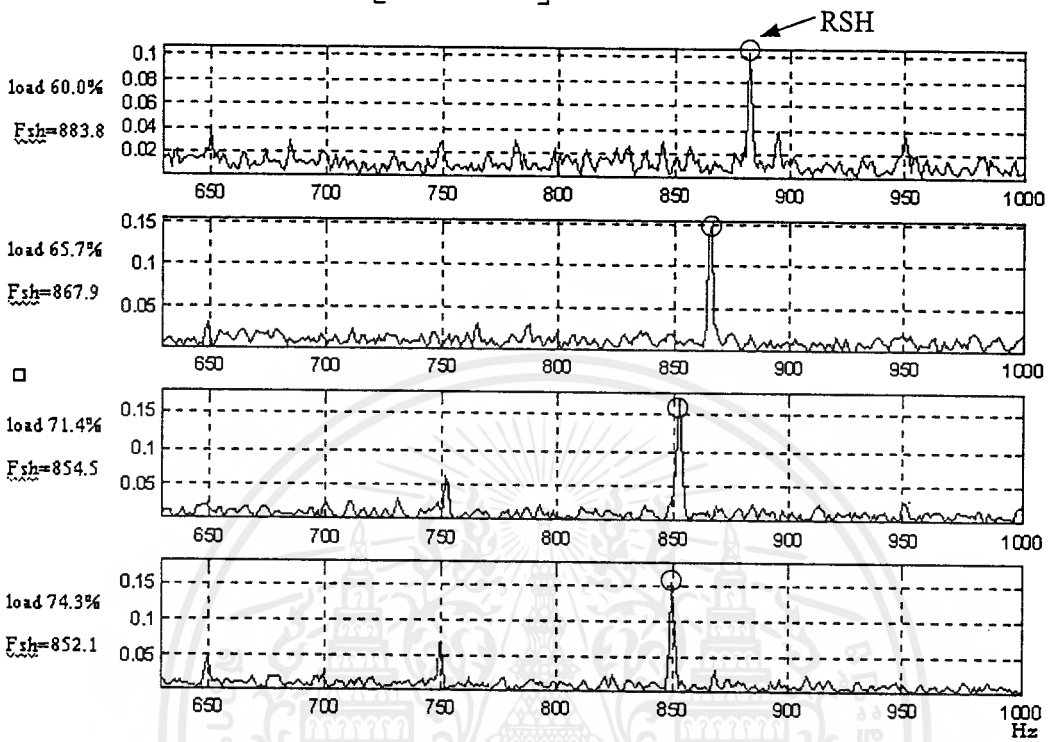
รูปที่ 4.5 แสดงสเปกตรัมของข้อมูลที่โหลดต่างๆ

4.6 กระบวนการที่ใช้ในการระบุตำแหน่งโรเตอร์สล็อทฮาร์โมนิกส์

เมื่อเราได้กราฟ และข้อมูลขนาดสเปกตรัมของข้อมูล ก็จะต้องทำการระบุให้ได้ว่าสเปกตรัมตัวใดคือโรเตอร์สล็อทฮาร์โมนิกส์มีความถี่เท่าใด จากรูปที่ 4.5 มีฮาร์โมนิกส์ต่างๆ เกิดขึ้นมากมายซึ่งมีทั้งใช้โรเตอร์สล็อทฮาร์โมนิกส์และฮาร์โมนิกส์อื่น ดังนั้นเราจะต้องทำการแยกเป็นโรเตอร์สล็อทฮาร์โมนิกส์โดยสังเกตว่าสเปคตรับตัวใดบ้างที่มีการเลื่อนตัวเมื่อโหลดเปลี่ยนแปลง

ในการทดลองได้ใช้ มอเตอร์เหนี่ยวนำแบบกรงกระรอก 1 HP 380/220v 1500 rpm 4 Pole 36 สล๊อต ซึ่งเป็นมอเตอร์ที่มีค่า Slot/pole เป็นจำนวนเท่าของ 3 ดังนั้นสูตรในการคำนวณหาค่าความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ดังนี้

$$f_{sh} = \left[Z\frac{(1-s)}{P} \pm 1 \right] f_0 \quad \text{เมื่อ Z/P เป็น 3N}$$

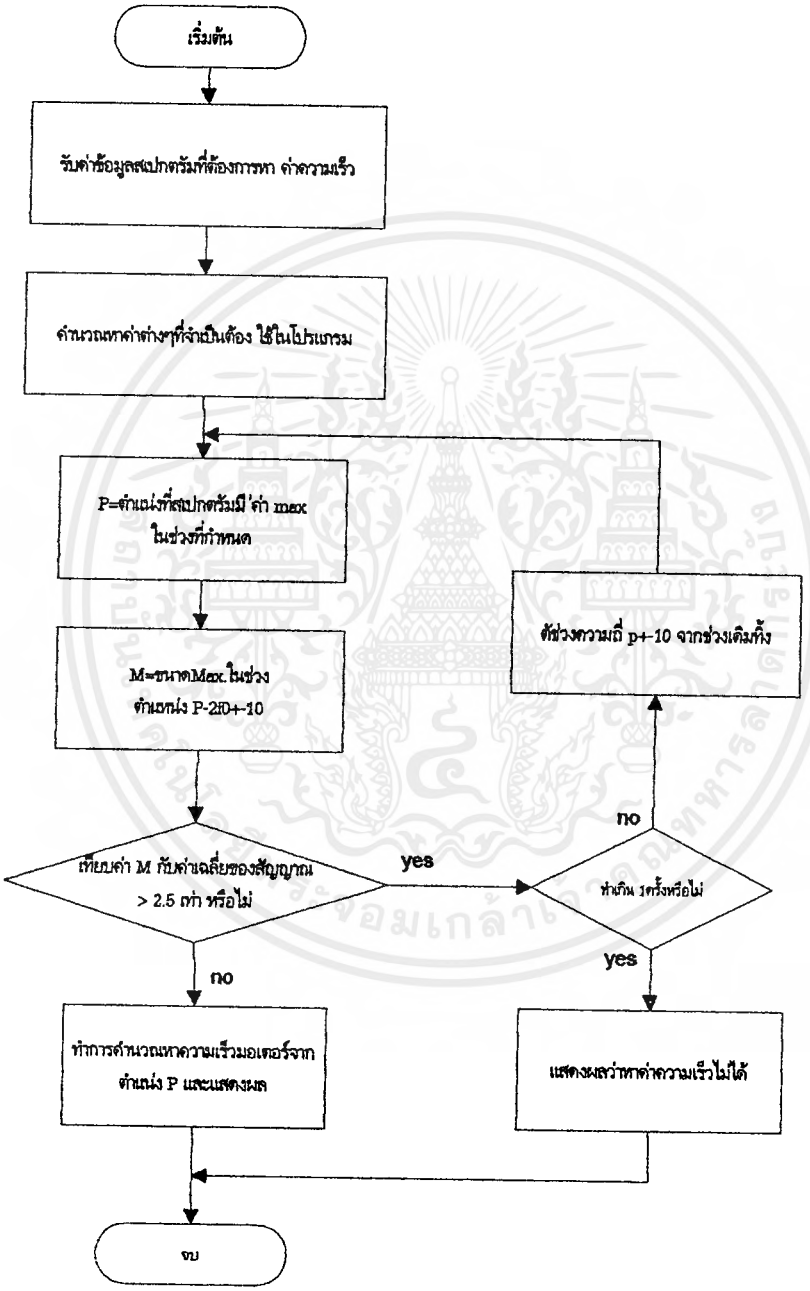


รูปที่ 4.6 แสดงสเปคตรัมของข้อมูลในช่วงความถี่ที่จะพิจารณาหาความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์

หากพิจารณาถึงช่วงความเร็วตั้งแต่ 1200 –1500 rpm ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ที่เกิดขึ้นจะมีค่าตั้งแต่ 750-950 Hz ดังนั้นในเขียนโปรแกรมสำหรับวิเคราะห์หาความเร็วของมอเตอร์ ก็จะการพิจารณาเฉพาะฮาร์โมนิกส์ที่เกิดในช่วงความถี่ดังกล่าวเท่านั้น และหากพิจารณาจากสูตรจะเห็นได้ว่าโรเตอร์สล็อทฮาร์โมนิกส์จะเกิดขึ้นเป็นคู่ โดยตำแหน่งที่เกิดจะห่างกันเท่ากับ 2 เท่าของความถี่ไฟฟ้าที่ป้อนให้กับมอเตอร์ และโรเตอร์สล็อทฮาร์โมนิกส์ที่มีความถี่สูงกว่ามักจะมีขนาดมากกว่า ตัวที่มีความถี่ต่ำกว่า ดังนั้นเพื่อเพิ่มความแน่นอนในการระบุความเร็วมอเตอร์จึงต้องมีการตรวจสอบเงื่อนไขดังกล่าวด้วย กระบวนการที่ใช้ในการหาค่าความเร็วมอเตอร์มีดังนี้

1. อ่านค่าข้อมูลการกระจายสเปกตรัมที่ต้องการจะหาค่าความเร็ว
2. ทำการคำนวณค่าเฉลี่ยของสเปกตรัม
3. ทำการพิจารณาหาตำแน่งที่ขนาดของสัญญาณมีค่ามากที่สุด ในช่วงความถี่ที่กำหนด
4. ตรวจสอบว่าสัญญาณที่หาได้นั้นมีคู่ของฮาร์โมนิกส์เกิดขึ้น ณ ตำแหน่งห่างจากเดิม $2f_0$ หรือไม่ โดยทำการหาค่าขนาดที่สูงที่สุดในช่วงความถี่ที่น้อยกว่า $2f_0 \pm 10$ Hz ว่ามีขนาดมากกว่า 2.5 เท่าหรือไม่

5. ถ้าใช่ก็แสดงว่ามีโอกาสเป็นโรเตอร์สล็อทฮาร์โมนิกส์สูง แล้วทำการคำนวณย้อนกลับหาความเร็วของมอเตอร์โดยใช้สมการข้างต้น
6. ถ้าไม่ใช่ที่เป็นให้ทำการคำนวณใหม่โดยตัดช่วงสัญญาณที่ได้ในข้อ 3 ทิ้งไปแล้วทำการคำนวณวนข้อ 3 –6 ใหม่ถ้ายังไม่เจออีกก็ให้แสดงผลว่าไม่เจอ



รูปที่ 4.7 แสดง Flow Chart ของโปรแกรมหาความเร็วมอเตอร์

บทที่ 5

โครงสร้างของ TMS320C50 DSP Starter Kit (50'DSK)

TMS320C50 เป็นโปรเซสเซอร์ที่ทำงานด้าน DSP (Digital Signal Processing) ซึ่งตระกูล TMS320 จะมีอยู่หลายเบอร์ เช่น TMS320C50, TMS320C51, TMS320C53 ซึ่งเป็นการนำเอาสถาปัตยกรรมของ C'25 มาปรับปรุงให้มากขึ้น ทั้งในด้านความเร็วและความสะดวกอื่นๆ

5.1 ลักษณะโดยทั่วไปของบอร์ด

1. มีแรมบนบอร์ด 9 K เวิร์ด
2. ทำงานได้รวดเร็วคือ 30-50 ns ต่อหนึ่งคำสั่ง
3. ใช้งานแทน 'C1X และ 'C2X ได้
4. มีแรม สำหรับ โปรแกรม / ข้อมูล ขนาด 9K x 16 บิต
5. มีรอมขนาด 2K x 16 บิต สำหรับการบูต
6. สามารถต่อหน่วยความจำข้างนอกได้ถึง 224K x 16 บิต ซึ่งประกอบด้วย หน่วยความจำสำหรับสำหรับเก็บโปรแกรม 64K หน่วยความจำเก็บข้อมูล 64K สำหรับ I/O และอื่นๆ อีก 64 K
7. มี ALU (Arithmetic Logic Unit) ACC (accumulator) และ ACCB (accumulator buffer) ขนาด 32 บิต
8. มี PLU (Parallel Logic Unit) ขนาด 16 บิต
9. มีคำสั่งในการคูณ 16 บิต ที่ทำงานใน 1 ไซเกิล
10. มีรีจิสเตอร์ถึง 8 ตัว ในการคำนวณและเก็บค่า
11. มีแสตก (Stack)
12. มีคำสั่งสำหรับการเลื่อนบิต (shift) ตั้งแต่ 0 – 16 บิต
13. อ้างแอดเดรสแบบเซอร์คูลาร์ (circular) โดยการใช้เซอร์คูลาบัฟเฟอร์ (circular buffer)
14. มีคำสั่งสำหรับการทำซ้ำโดยเฉพาะสำหรับการเคลื่อนย้ายบล็อก ภายในคำสั่งเดียว
15. มีคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูลและหน่วยความจำโปรแกรมมีพอร์ตอนุกรมที่รับส่งแบบ ฟูลดูเพลก (Full Duplex) สำหรับ 'C50 กับอุปกรณ์อื่นๆ
16. มี TDM (Time – division multiple) ของพอร์ตอนุกรม

17. สามารถกำหนดสัญญาณนาฬิกา โดยใช้ไทเมอร์ (Timer) , เคาน์เตอร์ (Counter)

18. มีซอฟแวร์ได้ทั้งหยุด (Stop) , เริ่ม (Start) , รีเซต (Reset)

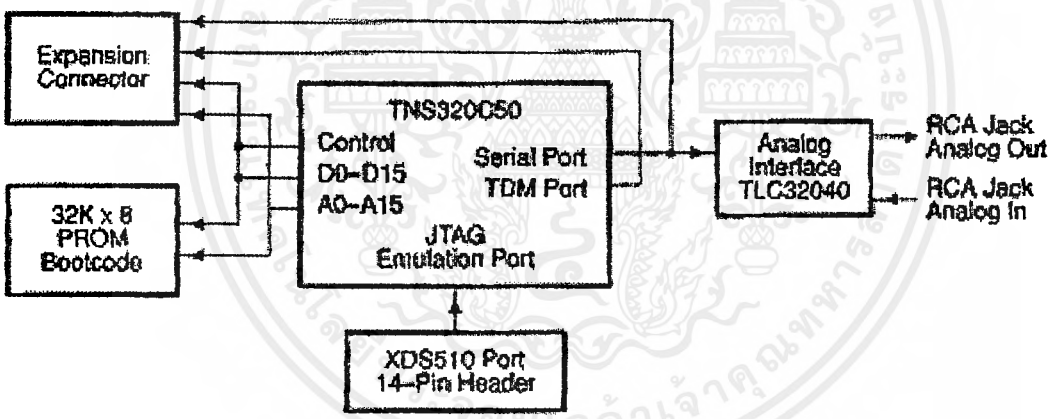
19. มีพอร์ต I/O ได้ถึง 64K และมี 16 ตำแหน่ง สำหรับการเข้าถึงหน่วยความจำ

20. มีการทำงานเป็นแบบกไปป์ไลน์

21. สามารถผลิตสัญญาณนาฬิกา โดยการหารสัญญาณนาฬิกาที่จ่ายให้กับซีพียู

ใช้เทคโนโลยีของ ซีมอส (CMOS) และใช้ไฟเลี้ยงเพียง 5V รูปที่ 2.1 แสดงบล็อกไดอะแกรมของบอร์ด ซึ่งประกอบไปด้วย โฮสอินเตอร์เฟส , อนาลอกอินเตอร์เฟส และ อีมูเลชั่น ทำให้สามารถติดต่อกับ พีซีได้โดยผ่านทาง RS232 นอกจากนี้ยังมี PROM ขนาด 32 Kbytes ที่ใช้เก็บ เคอร์แนลโปรแกรมไว้สำหรับการบูต

ส่วนของอนาลอกอินเตอร์เฟส (Analog Interface) ใช้ TLC32040 ซึ่งเป็นวงจรอินเตอร์เฟส สัญญาณอนาลอก (analog interface circuit : AIC) ที่มี RCA คอนเนคเตอร์ 2 ตัว สำหรับ อินพุท และ เอาท์พุท



รูปที่ 5.1 บล็อกไดอะแกรมของ TMS320C5x DSK

จากรูปที่ 5.1 การทำงานเริ่มต้นที่การบูตโหลดเดอร์ (Bootloader) ซึ่งบรรจุบนพีรอม บูตโหลดเดอร์ (PROM Bootloader) ขนาด 32 บิต เบอร์ 27PC256FM จะทำการบูต (Boot) บอร์ด เป็นการเซตค่าที่จำเป็นสำหรับ TMS ซึ่งใช้เป็นดีบักเกอร์ (debugger) ของ DSK

เมื่อเริ่มการรีเซต เริ่มต้นที่ตำแหน่ง 000H โดยขารีเซต หรือที่ $\overline{BR}$ จะเป็นสถานะต่ำ (Low) และทำการเรียกโปรแกรม เคอร์แนลโปรแกรม (Kernel Program) โดยใช้ขา $\overline{BIO}$ และ XF ของ TMS ในการติดต่อ กับ PC ทาง RS232

ในการติดต่อเมื่อ $\overline{BIO}$ เป็นสถานะต่ำ (Low) จะเป็นการแสดงว่าเริ่มติดต่อกับ RS232 ซึ่ง จะเป็นการกำหนดบิตเริ่มต้น สำหรับการคำนวณ โดนเริ่มต้นที่ 1 บิตเริ่มต้น + 7 บิต

ข้อมูล แล้วหารด้วย 8 จะได้อัตราบอร์ด (Baud Rate) ที่คำนวณจากคำสั่ง NOP ในโครงงานนี้กำหนดอัตราบอร์ดที่ 9600 bit/sec

5.1.1 วงจรส่วน CPU TMS320C50

ขา $MP/\overline{MC}$ ของ TMS320C50 ต่อลงกราวด์เพื่อให้อยู่ในโหมดไมโครคอมพิวเตอร์ในส่วนของอินเตอร์รัพต์ $\overline{INT1}-\overline{INT4}$ และ NMI จะแอคทิฟในสถานะต่ำ (low) ต้องมีตัวต้านทานต่อกับไฟเลี้ยงไว้ ส่วนของการติดต่อกับ RS232 จะต่อที่ $\overline{BIO}$ และ XF ซึ่งจะกล่าวอีกทีในการอธิบายส่วนของการติดต่อกับ PC ส่วนสัญญาณนาฬิกาของระบบจะใช้คริสตอลขนาด 40 MHz ต่อเข้าที่ CLKIN ในส่วนของ CLKOUT1 จะต่อไปยัง TLC32040

การติดต่อกับหน่วยความจำจะใช้แอดเดรสบัส A0 – A15 จะอ่านเขียนโดยกำหนดที่ขา R/W, RD, WE และ STRB

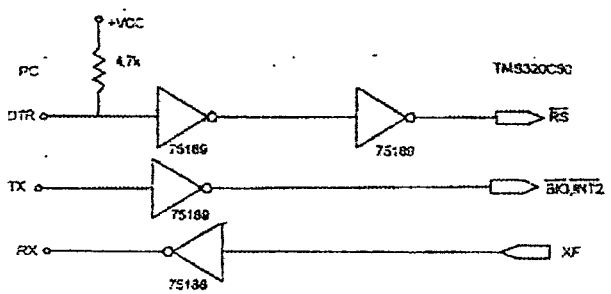
ในการติดต่อกับ TLC32040 เป็นการติดต่อแบบอนุกรมโดยมีขา DR สำหรับข้อมูลและ ขา DX สำหรับส่งข้อมูลโดยกำหนดสัญญาณที่ CLKR และ CLX ในการติดต่อ จะติดต่อเป็นแฟรม โดยมีการกำหนดเฟรมที่ FSR และ FSX กำหนดเฟรมในการรับและเฟรมในการส่งตามลำดับ

5.1.2 วงจรส่วนการติดต่อกับ RS232

การติดต่อกับ PC ผ่านทางพอร์ตอนุกรม ในโครงงานนี้จะใช้ COM2 มีขั้นตอนดังนี้ ข้อมูลจาก พีซี จะเข้าที่ขา TX ผ่าน UTA ซึ่งเป็นเบอร์ 75189 เพื่อเปลี่ยนสัญญาณ RS232 มาเป็นสัญญาณ TTL มาติดต่อกับ $\overline{INT2}$ และ $\overline{BIO}$ เข้าที่ TMS320C50 อีกที

ในการส่งข้อมูลจาก TMS ไปยัง PC จะผ่านที่ขา TX โดยมี IC เบอร์ 75188 เปลี่ยนสัญญาณ TTL เป็นสัญญาณ RS232 อีกที

ส่วนขา DRT จะใช้เป็นขาสำหรับเซตบอร์ด ซึ่งมี 75089X2 สำหรับการเปลี่ยนระดับสัญญาณ และการต่อเพิ่มเติมในส่วนของการติดต่อทาง RS232 จะมีการต่อดังรูปที่ 5.2



รูปที่ 5.2 แสดงการติดต่อกับ RS232

โดยที่

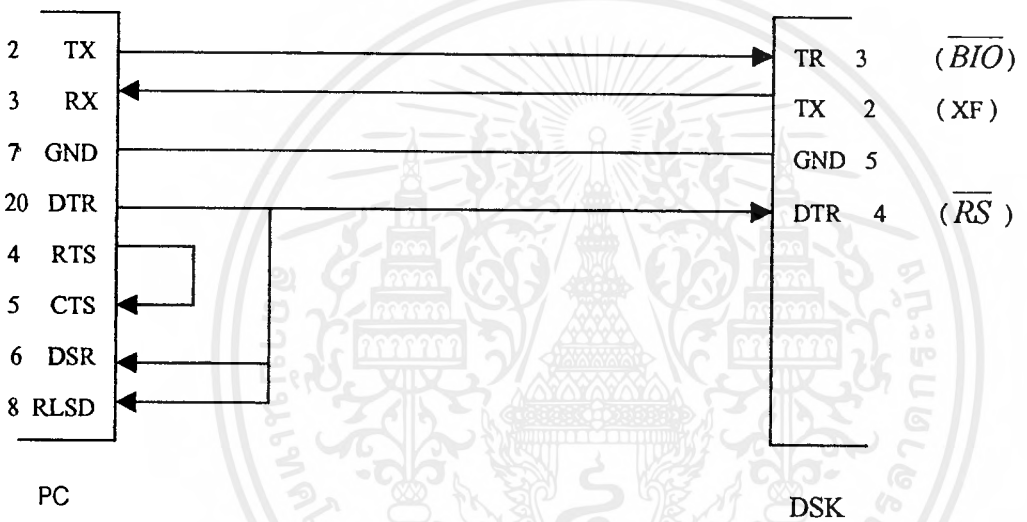
$\overline{CTS}$: clear to send

$\overline{DSR}$: data set ready

$\overline{RTS}$: reques to send

$\overline{DTR}$: data terminal ready

การรับสัญญาณจาก RS232 มาเป็น TTL และ จาก TTL มาเป็น RS232 จะใช้ไอซี 75189 และ 75188 ตามลำดับ โดยที่ปกติแล้ว ระดับ RS232 +12V จะเป็นลอจิก 0 และ -12V จะเป็นลอจิก 1 ซึ่งจะต่อดังรูปที่ 5.3



รูปที่ 5.3 สัญญาณในการติดต่อกับ RS232

5.1.3 วงจรส่วนที่ติดต่อกับ TLC32040

จากที่ได้กล่าวมาแล้ว การติดต่อระหว่าง TMS320C50 กับ TLC32040 จะผ่านทางพอร์ตอนุกรม โดยขาที่ใช้จะเป็น DX , DR , FSR , FSX และ CLK สัญญาณนาฬิกาของ TLC32040 จะได้มาจาก TMS320C50 ทาง TOUT

ในการติดต่อกับสัญญาณอนาลอกภายนอกจะเข้ามาทาง AIN+ หรือ AUXIN± เข้ามายัง TLC32040 ในการส่งข้อมูลอนาลอก จะส่งออกทาง OUT+ และ OUT-

5.1.4 วงจรที่ติดต่อกับหน่วยความจำ

เนื่องจากบอร์ดมี PROM เบอร์ 27PC256 ซึ่งเก็บบูตโหลดเดอร์ไว้ และส่วนของการติดต่อจะใช้บัสแอดเดรส A0 – A15 และบัสข้อมูล D0 – D7 ในการอ่าน $\overline{RD}$ และ $\overline{OE}$ โดยการเลือกชิปที่ $\overline{CE}$ ทาง $\overline{BR}$

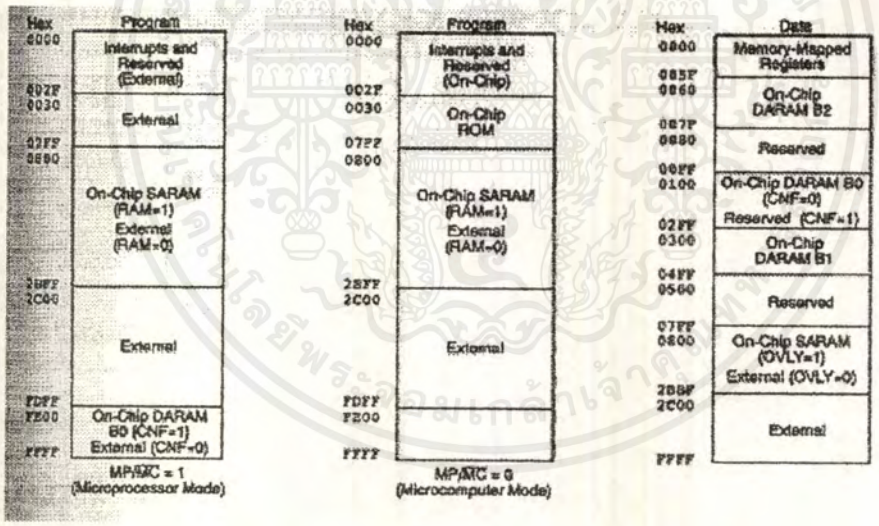
ในส่วนของ RAM ที่ใช้ จะใช้ภายใน TMS320C50 เอง ซึ่งมีขนาด 9K สำหรับหน่วยความจำโปรแกรม และ 9K สำหรับหน่วยความจำข้อมูลซึ่งจะอธิบายอีกครั้งในหัวข้อหน่วยความจำ

5.1.5วงจรส่วนไฟเลี้ยงและหัวต่อขยายบอร์ด

บอร์ดทำงานโดยใช้ไฟโดยขนาด ± 5 V โดยเรดติไฟ (rectifi) ไฟเข้ามา 9 V จากหม้อแปลง ในการทำให้แรงดันคงที่ จะใช้ IC LM7805 สำหรับไฟบวก และใช้ LM7905 สำหรับไฟลบ ส่วนการขยายบอร์ด หรือส่วนที่สามารถต่อกับภายนอกได้จะมีถึง 5 ส่วน (JP1 – JP5) ซึ่งจะเป็นการต่อกับขาของ TMS320C50 และ TLC32040 ทั้งหมด

5.2 การจัดหน่วยความจำของ TMS320C50

TMS320C50 จะมีการแบ่งหน่วยความจำเป็นส่วนของหน่วยความจำโปรแกรม และหน่วยความจำข้อมูล ตามรูป 5.4



รูปที่ 5.4 แสดงการเข้าถึงหน่วยความจำของ TMS320C50

5.2.1 หน่วยความจำข้อมูล (Data memory)

ในการออกแบบของ 'C5X จะใช้สถาปัตยกรรมแบบฮาร์วาร์ด (Havard) กล่าวคือมีการแยก หน่วยความจำข้อมูล และหน่วยความจำโปรแกรม ออกจากกัน ซึ่งจะมีผลดีต่อความเร็ว

หน่วยความจำข้อมูลของ 'C5X สามารถขยายได้ถึง 64K x 16 บิต ซึ่ง 'C51 มีแรมบนชิพถึง 9K ซึ่งแรมจะเป็นแบบ SARAM (single – access RAM) และ DARAM (dual – access RAM) ขนาด 1056 เวิร์ด บนชิพ ซึ่งมีข้อดีคือ

1. มีการปฏิบัติการได้เร็วไม่มีการรอ (wait state)
2. ทำงานโดยวิธีการไปป์ไลน์ ทำให้ทำงานได้รวดเร็ว
3. ช่วยลดค่าใช้จ่ายในการต่อหน่วยความจำภายนอก
4. ช่วยประหยัดไฟ ซึ่งถ้าใช้การต่อหน่วยความจำภายนอกจะสิ้นเปลืองกว่า

2.2.2 หน่วยความจำโปรแกรม (Program Memory)

ในการใช้งานของหน่วยความจำโปรแกรม ซึ่งสามารถขยายได้ถึง 64K ซึ่ง 'C5X มีรอม , SARAM และ DARAM ซึ่งมีความเร็วทีสูง โดยไม่มีการรอ (wait state)

ในการทำงานของซีพียู 'C50 สามารถใช้ร่วมกับรอมภายในขนาด 4K ซึ่งสามารถโปรแกรมมาจากโรงงาน มีความเร็วในการทำงานเต็มที่ ในการเลือกใช้จะต้องมีการกำหนดที่ขาของ $/MP/\overline{MC}$ หากขาดังกล่าวเป็นสถานะสูง ตำแหน่ง 4K แรกจะเป็นหน่วยความจำภายนอกชิพ แต่ถ้าเป็นสถานะต่ำ ตำแหน่ง 4K เวิร์ดแรกจะเป็นรอมภายในชิพ ดังรูปที่ 5.4

5.3 วงจรอินเตอร์เฟสสัญญาณอนาลอก TLC32040

5.3.1 ลักษณะที่สำคัญของ TLC320C40 มีดังนี้

- ใช้เทคโนโลยีการผลิต Advanced LinCMOS™ Silicon-Gate Process
- ความละเอียดของ ADC และ DAC เป็น 14 บิต
- สามารถเปลี่ยนอัตราการแซมปลิ้งของ ADC และ DAC ได้ถึง 19,200ครั้ง/วินาที
- มี Switch-Capacitor Antialiasing Input Filter และ Output-Reconstruction Filter
- มีพอร์ตอนุกรมสำหรับติดต่อโดยตรงกับ TMS3211, TMS320C17, TMS32020, และ TMS320C25 DSP
- สามารถปรับอัตราการแปลงของ ADC และ DAC ทั้งแบบซิงโครนัส หรืออะซิงโครนัสโดยใช้ซอฟท์แวร์ควบคุม

5.3.2 การทำงานของ TLC32040

- อินพุททางอนาลอก (analog input) สำหรับ อนาลอกอินพุทจะมี 2 กลุ่มคือ IN+ , IN- และ AUX IN+ , AUX IN- ซึ่งสามารถเลือกใช้กลุ่มใดกลุ่มหนึ่ง โดยจะใช้ในแบบ ดิฟเฟอเรนเชี

ยล (differential) หรือ ซิงเกิล-เอนด์ (single – ended) และค่าเกน สำหรับอินพุท IN+ , IN- , AUX IN+ และ AUX IN- สามารถจะใช้โปรแกรมตั้งค่าได้ (มี 3 ค่า คือ 1 2 หรือ 4) การเลือกกลุ่มอินพุทใด จะเลือกโดยใช้ซอฟท์แวร์ควบคุม

- A / D bandpass filter , A/D bandpass filter clocking และ A/D convertion timing

สำหรับ A/D แบนด์พาส ฟิลเตอร์ (A/D bandpass filter) เราสามารถที่จะเลือกใช้หรือไม่ใช้ก็ได้โดยใช้ซอฟท์แวร์ควบคุม ความถี่ของสัญญาณนาฬิกาควบคุมฟิลเตอร์ (filter clock) จะเป็นตัวกำหนดทรานเฟอร์ฟังก์ชั่น (transfer function) ของฟิลเตอร์ โดยจะคิดจากอัตราส่วนความถี่สัญญาณนาฬิกาควบคุมฟิลเตอร์ 28 kHz ที่ความถี่ต่ำ เริ่มมีลักษณะความถี่สูงผ่าน จะมีความถี่เป็น 300 Hz อัตราการแปลง D/A ก็หาได้จากความถี่ ที่ หาร 288 kHz ด้วย RX เคาน์เตอร์ B

- เอาท์พุททางอนาลอก (analog output)

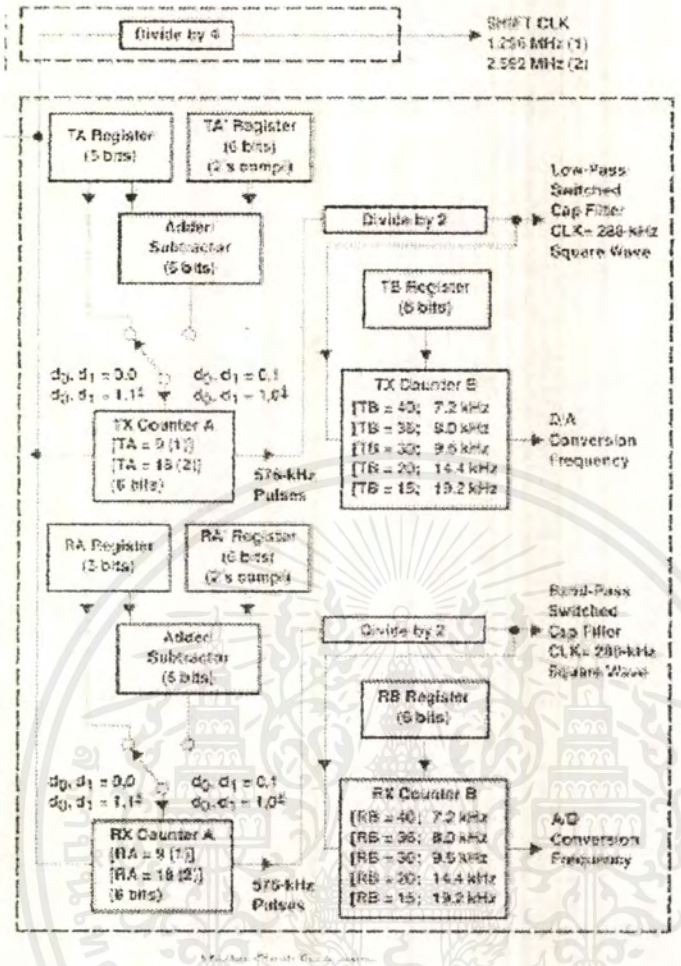
อนาลอกเอาท์พุทจะมีแอมปลิไฟล์ (power amplifier) มีเอาท์พุททั้งแบบ นอน-อินเวอร์ติง (non-inverting) และ แบบ อินเวอร์ติง (inverting) และเนื่องจากมีแอมปลิไฟล์ทำให้เอาท์พุท สามารถขับ ทรานฟอเมอร์ ชนิด ไฮบริดจ์ (hybridge transformer) หรือ โหลดอิมพีแดนซ์ต่ำได้ โดยใช้ทั้งแบบดิฟเฟอร์เรนเชียล หรือ ซิงเกิล – แอนด์

- วงจรกรองความถี่ต่ำของ A/D , วงจรกรองความถี่ แบนด์พาส ของ D/A

สัญญาณนาฬิการควบคุม วงรกรองความถี่ต่ำ และ อัตราการแปงสัญญาณดิจิตอลเป็นอนาลอก (D/A lowpass filter , D/A lowpass filter clocking , D/A conversion timing) เช่น Hz เดียวกับ A/D ฟิลเตอร์โดยทรานเฟอร์ฟังก์ชั่นของฟิลเตอร์ถูกกำหนดจากอัตราส่วนกับความถี่ 288 Hz และ อัตราการแปลง D/A ก็หาได้จากความถี่ 288 หารด้วย TX เคาน์เตอร์ B

- การต่อกลับ (Loopback)

การต่อกลับจะให้ผู้ใช้ตรวจสอบวงจร โดย OUT+ และ OUT- จะต่อเข้าภายในกับ IN+ และ IN- ดังนั้นบิต DAC(d15 – d2) จะถูกส่งไปยังขา DX และเปรียบเทียบกับบิต ADC ที่รับมาจากขา DR ซึ่งโดยปกติจะมีค่าที่เท่ากัน (แต่ในทางปฏิบัติ อาจไม่เท่ากันก็ได้) ในการตรวจสอบ ถ้าใช้ขา IN+ และ IN- สัญญาณภายนอกที่ต่อกับ IN+ และ IN- จะไม่มีผล แต่ถ้าใช้ AUX IN+ , AUX IN- สัญญาณภายนอกจะถูกรวมกับ OUT+ และ OUT- สำหรับการควบคุมการต่อกลับ จะทำโดยการตั้งค่าในรีจิสเตอร์ควบคุม



รูปที่ 5.5 แสดงไทม์เมอร์ภายใน

5.4 อุปกรณ์เสริม (Peripherals)

TMS320C50 มีการอินเตอร์เฟสกับภายนอกได้ถึง 8 วิธี ซึ่งประกอบด้วย

1. อินเตอร์รัพต์
2. พอร์ตอนุกรม (serial port)
3. พอร์ตอนุกรมแบบแบ่งเวลา (TDM serial port)
4. ไทม์เมอร์ (Timer)
5. รีจิสเตอร์ใช้โปรแกรมสภาวะการรอ (software – programable wait state)
6. พอร์ตติดต่อภายนอก (I/O port)
7. รีจิสเตอร์หารสัญญาณนาฬิกา (devide – by – one clock)

8. XF และ BIO

ซึ่งทั้งหมดจะควบคุมผ่านรีจิสเตอร์ควบคุม โดยการใช้วิธีการอ้างหน่วยความจำ ในการใช้งานจะเป็นการใช้ โดยการส่งลงที่แอดเดรสของหน่วยความจำโปรแกรม และข้อมูล ซึ่งเราก็จะกล่าวถึงเฉพาะหัวข้อที่เกี่ยวกับโครงงานเท่านั้น

5.4.1อินเตอร์รัพต์ (Interrupt)

TMS320C50 มีอินเตอร์รัพต์ทั้งหมด 16 อินเตอร์รัพต์ (16 INT – INT1) แต่จะไม่ใช้พร้อมกันทั้งหมด โดยปกติจะใช้ 9

การรีเซต (Reset) จะเป็นอินเตอร์รัพต์แบบ นอน มาร์กอะเบิล จากภายนอก nonmarskable external interrupt) ซึ่งเมื่อเกิดขึ้นจะมีการรีเซตดังนี้

1. CNF = 0 หมายถึงการใช้หน่วยความจำข้อมูลบนชิพ
2. PC = 0000H
3. INTM บิต = 1 และ IFR = 0 ทำให้อินเตอร์รัพต์ทุกตัวไม่ทำงาน
4. บิตสถานะ (status bit) จะเป็นดังนี้

$0 \rightarrow$ VO $0 \rightarrow$ XF $1 \rightarrow$ SXM $0 \rightarrow$ PM $1 \rightarrow$ HM

$0 \rightarrow$ BRAI $0 \rightarrow$ TRM $0 \rightarrow$ NDX $0 \rightarrow$ CENM1 $0 \rightarrow$ CENM2

$0 \rightarrow$ IPTR $0 \rightarrow$ OVLY $0 \rightarrow$ AVIS $0 \rightarrow$ RAM $0 \rightarrow$ BIG

$0 \rightarrow$ CNF $1 \rightarrow$ INTM MP/MC (PIN) $\rightarrow$ (MC/MP) $1 \rightarrow$ C

5. GREG = 0000000
6. RPTC = 00
7. IACK จะแสดงการเก็บสถานะการรีเซต

ในการควบคุมการอินเตอร์รัพต์จะมี IFR (Interrupt Flag Register) แสดงภาวะการอินเตอร์รัพต์ และ IMR (Interrupt Mask Register) แสดงการ Mask ของอินเตอร์รัพต์ ดังรูปที่ 5.7

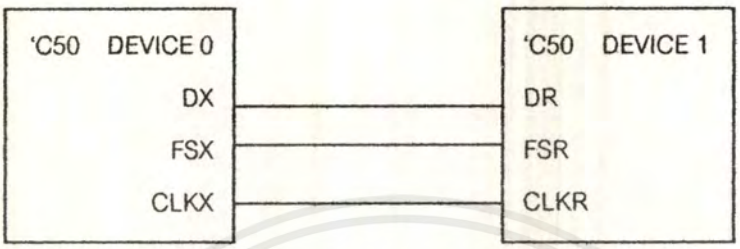
15 9	8	7	6	5	4	3	2	1	0
Reserved	$\overline{INT4}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{INT3}$	$\overline{INT2}$	$\overline{INT1}$

รูปที่ 5.6 รีจิสเตอร์อินเตอร์รัพต์แฟลก

ในการกำหนดให้ใช้อินเตอร์รัพต์จะต้องทำการเคลียร์ที่ INTM บิตด้วย

5.4.2 พอร์ตอนุกรม (Serial port)

Serial port ของ TMS320C50 จะเป็นแบบฟูล – ดูเพล็กซ์ คือ สามารถรับส่งภายในเวลาเดียวกัน ทำให้สามารถติดต่ออุปกรณ์ภายนอกอื่นๆได้ ไม่ว่าจะเป็น CODEC A/D หรือระบบอื่นๆ ในการต่อขาต่างๆ จะเป็นตามรูปที่ 5.8



รูปที่ 5.7 แสดงการต่อพอร์ตอนุกรมกับพอร์ตภายนอก

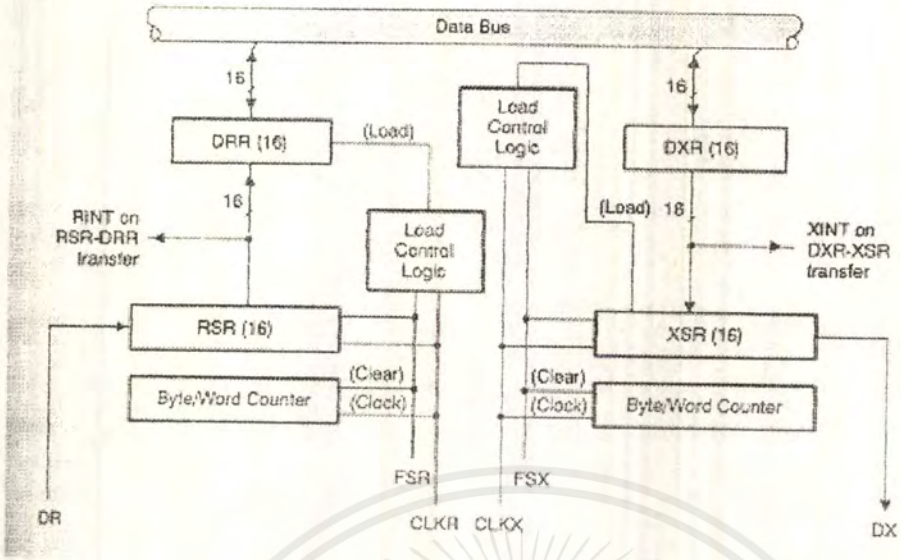
ตารางที่ 5.1 ขาสัญญาณของพอร์ตอนุกรม

ขาสัญญาณ	ความหมาย
CLKX	สัญญาณคล็อกที่ใช้ในการส่ง
CLKR	สัญญาณคล็อกที่ใช้ในการรับ
DX	สัญญาณข้อมูลอนุกรมในการส่ง
DR	สัญญาณข้อมูลอนุกรมในการรับ
FSX	สัญญาณการซิงโครไนซ์เฟรมการส่ง
FSR	สัญญาณการซิงโครไนซ์เฟรมการรับ

และในการควบคุมพอร์ตอนุกรมจะมีรีจิสเตอร์ที่จำเป็นตามตารางที่ 5.2

ตารางที่ 5.2 รีจิสเตอร์พอร์ตอนุกรม

รีจิสเตอร์	ความหมาย
SPC	รีจิสเตอร์ควบคุมพอร์ตอนุกรม
DXR	รีจิสเตอร์ส่งข้อมูล
DRR	รีจิสเตอร์รับข้อมูล
XSR	รีจิสเตอร์เลื่อนการส่ง
RSR	รีจิสเตอร์เลื่อนการรับ



รูปที่ 5.8 บล็อกไดอะแกรมการทำงานของพอร์ตอนุกรม

ในการส่งข้อมูลทำได้โดยการเขียนข้อมูลลงใน DXR (Data transmit register) จะเป็นตัวเลื่อนบิตตามสัญญาณนาฬิกาจนครบ ไปตามขา DR เข้ามาที่ RSR (Receive Shift Register) เมื่อครบแล้วจะทำการอินเตอร์รัพต์บอกให้ทราบ แล้วสามารถรับข้อมูลโดยการอ่านข้อมูลจาก DRR (Data Receive Register)

5.4.3 ไทม์เมอร์ (Timer)

ในการใช้ไทม์เมอร์ของ TMS320C50 ซึ่งจะสามารถตั้งคาบเวลาของสัญญาณนาฬิกาได้ โดยสัญญาณที่จะออกที่ขา Tout ของ TMS320C50

ซึ่งการคำนวณคาบเวลาจะเป็นไปตามสูตร

$$T = \frac{1}{t_c \times (TDDR+1) \times (PRD+1)}$$

ซึ่ง t_c เป็นคาบเวลาของสัญญาณที่จ่ายให้ที่ขาของ CLKOUT1

TDDR (Timer Divide Down Ratio)

PRD (Period Register) เป็นรีจิสเตอร์ที่สามารถใส่ค่าคาบเวลาได้

ในการทำงานจะต้องมีการใส่ค่าต่างๆลงบน TCR (Timer Control Register) ดังตารางที่ 5.3

ตารางที่ 5.3 รีจิสเตอร์ควบคุมไทม์เมอร์ (TCR)

15-12	11	10	9 -6	5	4	3-0
Reserve	SOFT	FREE	PSC	TRB	TSS	TDDR

5.4.4 การเขียนโปรแกรมและคำสั่ง

- การกำหนดสถานะเริ่มต้นสำหรับ TMS320C50

ทุกครั้งที่มีการใช้งาน TMS320C50 จำเป็นต้องมีการกำหนดสถานะเริ่มต้น (Initialize) ตัวซีพียูก่อนเสมอซึ่งต้องทำการเตรียมสภาวะของ โปรเซสเซอร์ให้พร้อม โดยปกติจะทำไว้เมื่อเกิดการรีเซตของ TMS320C50 จะเกิดขึ้นเมื่อขา $\overline{RS}$ อยู่ในสถานะต่ำ ซึ่งจะทำให้ IPTR (Interrupt pointer) ซึ่งอยู่ใน PMST (processor mode status register) เกิดการเคลียร์ ทำให้เกิดการ map เวคเตอร์ไปที่เพจ 0 ในการเกิดอินเตอร์รัพต์ นั้นเมื่อรีเซตจะทำให้ disable และในการกำหนดสถานะเริ่มต้นสำหรับโปรเซสเซอร์นั้นจะประกอบด้วย

- การ Map หน่วยความจำ และรีจิสเตอร์ควบคุม
- การเซตโครงสร้างของอินเตอร์รัพต์
- โหมดควบคุมต่างๆ (OVM , SXM , PM , AVIS , NDX , TRM)
- การควบคุมหน่วยความจำ (RAM , OVLY , CNF)
- กำหนดรีจิสเตอร์ช่วย (Auxiliary register) และตัวชี้รีจิสเตอร์ช่วย
- กำหนดตัวชี้เพจข้อมูล (Data page pointer)

การกำหนดค่าใน PMST (Processor mode status register)

ตารางที่ 5.4 การกำหนดค่าใน PMST

15 14 13 12 11	10	9	8	7	6	5	4	3	2	1	0
IPTR	0	0	0	AVIS	0	OVLY	RAM	$MP/\overline{MV}$	NDX	IRM	BRAT

ค่า IPTR ที่เรากำหนดที่ IPTR = 10000B หรือเป็นเพจที่ 1 ซึ่งตรงกับโปรแกรมตำแหน่ง 0800h ส่วนการกำหนดใช้ RAM ภายใน จะต้องใช้ RAM 9K ซึ่งอยู่ในชิพ กำหนดแรม OVLY เป็น 1 RAM ที่ใช้จะเป็น SARAM โดยที่หน่วยความจำโปรแกรมอยู่ที่ 0800h – 2BFFh

ในส่วนของการกำหนดอินเตอร์รัพต์เวกเตอร์ (Interrupt vector) มีการกำหนดดังนี้

- Reset 0800h การรีเซตภายนอก
- $\overline{INT1}$ 0802h สัญญาณอินเตอร์รัพต์จากผู้ใช้ภายนอก หมายเลข 1
- $\overline{INT2}$ 0804h สัญญาณอินเตอร์รัพต์จากผู้ใช้ภายนอก หมายเลข 2
- $\overline{INT3}$ 0806h สัญญาณอินเตอร์รัพต์จากผู้ใช้ภายนอก หมายเลข 3
- TINT 0808h สัญญาณอินเตอร์รัพต์ไทมเมอร์ภายใน

- RINT 080Ah สัญญาณอินเตอร์รัพต์สำหรับการรับค่าจากพอร์ตอนุกรม
- XINT 080Ch สัญญาณอินเตอร์รัพต์สำหรับการส่งค่าให้พอร์ตอนุกรม
- $\overline{INT4}$ 0812h สัญญาณการอินเตอร์รัพต์จากผู้ใช้ภายนอก
- TRAP 0822h คำสั่งซอฟต์แวร์ แทรป (Software trap instruction)
- NMI 0824h นอนมาร์กอะเบิลอินเตอร์รัพต์ (Nonmaskable Interrupt)

และในโครงงานนี้เรากำหนด interrupt จาก IMR (Interrupt mask register) ซึ่งจะกำหนดดังนี้

ตารางที่ 5.5 การกำหนดค่าใน IMR

15....9	8	7	6	5	4	3	2	1	0
Reserved	$\overline{INT4}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{INT3}$	$\overline{INT2}$	$\overline{INT1}$

ซึ่งก่อนการใช้อินเตอร์รัพต์จะต้องไม่ mask อินเตอร์รัพต์ ตัวนั้นๆ โดยใช้คำสั่ง clrc INTM ส่วน IMM จะแอคทิฟที่ "1"

<u>ตัวอย่าง</u>

```
setc    INTM          ; กำหนด INT1 , XINT
SPLK    #031h,IMR     ; RINT แอคทิฟ
Clrc    INTM
```

ในโครงงานนี้ใช้ $\overline{INT2}$, RINT , XINT แอคทิฟ

- การเขียนโปรแกรมทางด้านคณิตศาสตร์และลอจิก

TMS320C50 มี PLU (Parallel Logic Unit) ซึ่งจะเป็นการกระทำทางลอจิก (Logic) ที่มีความเร็วมาก ในส่วนนี้มีรีจิสเตอร์ซึ่งใช้ในการประมวลผล เช่น สเกลลิงชิฟเตอร์ (scaling shifter) ตัวคูณแบบขนาน 16×16 บิต อริธเมติก ลอจิก ยูนิต (arithmatic logic unit : ALU) 32 บิต แอคคิวมูเลเตอร์ (accumlator :ACC) 32 เป็นต้น

ซึ่งมีประสิทธภาพในการประมวลผลทางด้าน คณิตศาสตร์ และลอจิก ที่จำเป็นในการใช้งานทางด้านการประมวลผล ทางดิจิตอล

บทที่ 6
หน่วยลดทอนสัญญาณ

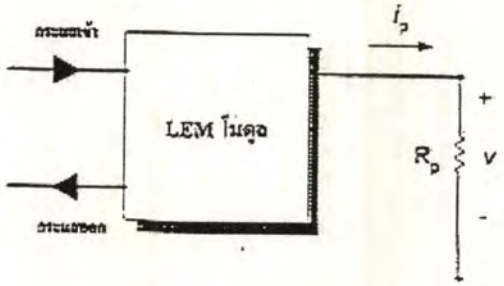
6.1 บทนำ

ข้อสำคัญประการหนึ่งในการวัดสัญญาณฮาร์โมนิกส์คือเราไม่สามารถวัดขนาดและรูปร่างสัญญาณเข้ามาคำนวณขนาดขององค์ประกอบฮาร์โมนิกส์ได้โดยตรง เนื่องจากจะต้องนำรูปคลื่นกระแสมาทำการคำนวณโดยอาศัย การประมวลผลเชิงตัวเลข (DSP ; Digital Signal Processing) โดยการใช้โปรแกรม MATLAB ซึ่งก่อนทำการประมวลผลลักษณะรูปคลื่นซึ่งเป็นสัญญาณอนาลอก (Analog) จะถูกเปลี่ยนเป็นสัญญาณดิจิตอลโดยใช้สโคปรุ่น HP 54520C ซึ่งเป็นตัวแปลงสัญญาณอนาลอกเป็นดิจิตอล จากนั้นทำการบันทึกข้อมูลลงแผ่นดิสก์ ในการใช้หน่วยลดทอนจะต้องทำการออกแบบวงจร ซึ่งได้ออกแบบไว้เพื่อให้สามารถใช้ได้กับบอร์ด TMS320C26 เนื่องจากมีความคิดที่จะใช้บอร์ดดังกล่าวในโครงงานที่ 2 สัญญาณ อนาลอกนี้จะเป็นสัญญาณที่อยู่ในช่วง ± 3 โวลท์ ดังนั้น จึงจะต้องลดทอนสัญญาณเข้าอยู่ในรูปของสัญญาณแรงดันไม่เกิน ± 3 โวลท์ โดยที่หลังการลดทอนสัญญาณลักษณะรูปคลื่น (wave form) ต้องมีผิดเพี้ยน (distortion) น้อยที่สุด

โดยความจำเป็นอันนี้จึงเลือกใช้ LEM โมดูลซึ่งเป็นตัวแปลงกระแสโดยอาศัยหลักการของปรากฏการของฮอลล์ (Hall effect) โดยการทำงานจะคล้ายกับหม้อแปลงกระแสโดยทั่วไป คือให้กระแสออกทางด้านทุติยภูมิแต่มีความผิดเพี้ยนของสัญญาณน้อยกว่า

6.2 การแปลงสัญญาณกระแสให้เป็นสัญญาณแรงดัน

กระแสที่ออกจาก LEM โมดูลจะถูกแปลงเป็นสัญญาณแรงดันค่าไม่เกิน ± 3 โวลท์ โดยให้กระแสผ่านความต้านทานค่าคงที่ค่าหนึ่งแล้วนำแรงดันตกคร่อมมาใช้



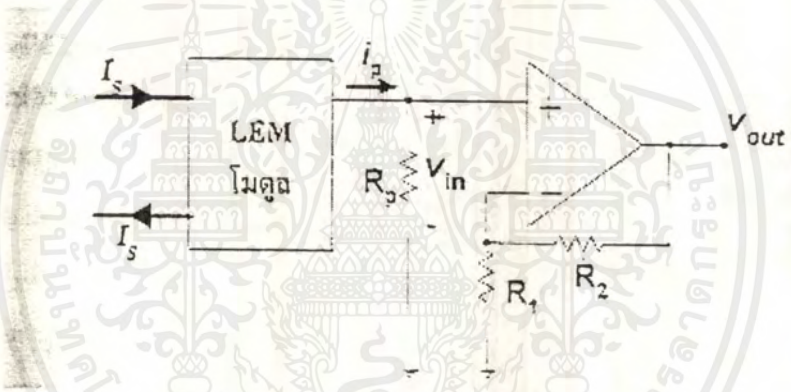
รูปที่ 6.1 แสดงการแปลงสัญญาณกระแสจาก LEM โมดูลให้เป็นสัญญาณแรงดัน

ซึ่งแรงดันตกคร่อม R_p หาได้จาก

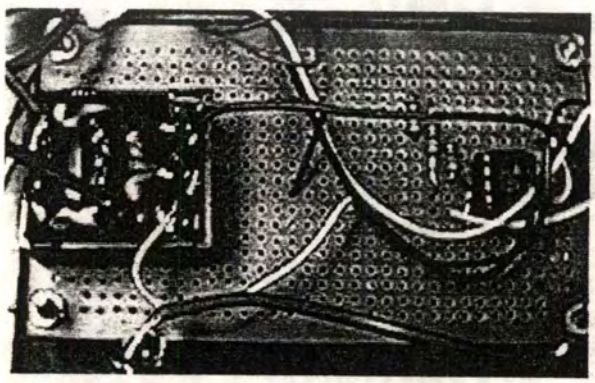
$$v = i_p R_p \qquad 6.1$$

แต่อย่างไรก็ตามการนำแรงดันตกคร่อม R_p ป้อนการ์ด DSP โดยตรงนั้นอาจมีผลให้การวัดผิดเพี้ยนได้เนื่องจากการดึงกระแสของการ์ด DSP (ถึงแม้จะมีค่าน้อยแต่สามารถหลีกเลี่ยงได้)

ดังนั้นจะใช้วงจร นอน-อินเวิร์ทติง (non-inverting) มาขั้นกลางโดยใช้ออปแอมป์ LF351 ซึ่งเป็น ออปแอมป์ที่ใช้งานได้ในช่วงความถี่ได้สูง (4 Mhz) ซึ่งใช้ได้ในย่านการใช้งานสำหรับสัญญาณในการวัดสัญญาณฮาร์โมนิกส์ไม่เกินลำดับที่ 35 คือ ที่ความถี่ 35×50=1750 Hz หรือ 1.75 Khz



รูปที่ 6.2 แสดงการใช้วงจร นอน อินเวิร์ทติง (non-inverting) มาขั้นกลางระหว่างสัญญาณแรงดันจาก LEM โมดูล



รูปที่ 6.3 ภาพแสดง LEM โมดูลที่ต่อเข้ากับนอน-อินเวิร์ทติง

จากวงจรรูปที่ 6.2 จะได้

$$v_{out} = (1 + \frac{R_2}{R_1})v_{in} \qquad 6.2$$

โดยเลือกใช้ $R_1 = R_2 = 5\ K\Omega$ เราจะได้ $V_{out} = 2.V_{in}$ และเนื่องจาก V_{out} เป็นสัญญาณที่จะต้องป้อนเข้าการ์ด DSP ซึ่งจำกัดขนาดแรงดันสูงสุดไม่เกิน ± 3 โวลท์ ดังนั้น V_{in} ต้องมีขนาดไม่เกิน ± 1.5 โวลท์ เนื่องจาก LEM โมดูลที่ใช้เป็นแบบ LA25 NP ซึ่งทนกระแสได้ 25 แอมแปร์ ดังนั้นจะทนกระแสสูงสุดได้ $25 \times \sqrt{2} = 35.4$ แอมแปร์ หรือประมาณ 36 แอมแปร์

โดยเลือกใช้อัตราส่วน (turn ratio) เป็น 1000/1 ดังนั้น

$$I_p = \frac{36}{1000} A$$

$$= 36 \quad \text{mA}$$

จากสมการ (6.1) จะสามารถคำนวณ R_p ได้ แต่ เนื่องจาก $V=V_{in}$ ดังนั้น

$$R_p = \frac{v}{i_p} \Omega \qquad 6.3$$

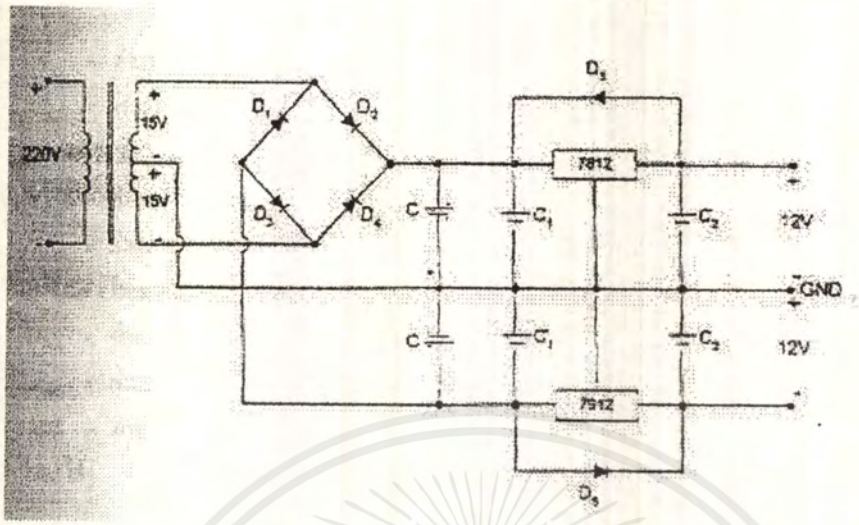
หาค่า $R_p = 1.5/36 \times 10^3\ \Omega$

$= 41.7\ \Omega$

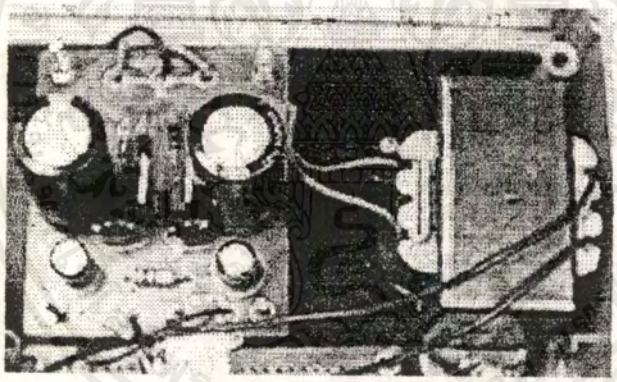
ดังนั้นเลือกใช้ $R_p = 40\ \Omega$

6.3 วงจรสร้างไฟเลี้ยงออปแอมป์ และ LEM โมดูล ± 12 โวลท์

โดยการจ่ายไฟสลับ 220 โวลท์ ผ่านหม้อแปลง ผ่านวงจรบริดจ์เรกติฟายแบบเต็มคลื่น ผ่านฟิลเตอร์ แล้วทำให้เรียบด้วยไอซีเรกกูเลตดังรูปที่ 6.3



รูปที่ 6.4 วงจรจ่ายไฟเลี้ยง ±12 โวลท์



รูปที่ 6.5 แสดงภาพภาคจ่ายไฟของเครื่องลดทอนสัญญาณ

ใช้ไอซีเรกกูเลต MC7812 และ MC7912

6.3.1 ขนาดของหม้อแปลง

คำนวณ V_{ac} ที่ใช้จาก

$$V_{ac} = \left(\frac{V_o + V_{reg} + V_{rect} + V_{ripple}}{0.92}\right) x \frac{V_{norm}}{V_{\min}} x \frac{1}{2} \qquad 6.4$$

เมื่อ V_o = ค่าแรงดันไฟตรงที่ผ่านการเรกกูเลตแล้ว

V_{reg} = แรงดันที่ตกคร่อมไอซีเรกกูเลต ซึ่งมีค่าประมาณ 3 โวลท์ สำหรับไอซีเรกกูเลตเบอร์ 78XX และ 79XX

V_{rect} = ค่าแรงดันคัทอิน (cutin voltage) ที่ตกคร่อมไดโอด ซึ่งประมาณ 1.1 โวลท์

V_{ripple} = ค่าแรงดันที่ไม่เรียบสูงสุด (peak ripple voltage) ที่ผ่านตัวเก็บประจุฟิลเตอร์ (C filter) ซึ่งมีค่าประมาณน้อยกว่า 1 โวลท์

V_{Nom} = ค่าแรงดันไฟสลับที่จ่ายให้วงจรในภาวะปกติ ใช้ 220 โวลท์

V_{min} = ค่าแรงดันไฟสลับที่จ่ายวงจรในภาวะผิดปกติ เช่น ไฟตก เป็น 200 โวลท์

0.92 = ค่าประสิทธิภาพการเรกติฟาย (Rectifier efficiency)

ดังนั้นจะได้

$$V_{ac} = \left(\frac{12+3+1.1+0.5}{0.92}\right) x \frac{220}{200} x \frac{1}{2}$$

$$= 4.0\ \text{V}$$

ใช้ V_{ac} 15 โวลท์ในการใช้งาน

เนื่องจากหม้อแปลงที่ใช้เป็นแบบมีแท็ปกลาง (center tap) ดังนั้นพิกัดหม้อแปลงที่ใช้คือ 220/30 โวลท์ กระแสไฟตรง I_{dc} ที่ต้องใช้ = กระแสจ่ายออปแอมป์ + กระแสจ่าย LEM โมดูล

$$= 1.8\ \text{mA} + 25\ \text{mA}$$

$$= 26.8\ \text{mA}$$

เนื่องจากเป็นวงจรบริดจ์เรกติฟายแบบเต็มคลื่นดังนั้น

$$I_{ac} = 1.65 I_{dc} \qquad 6.5$$

ดังนั้นจะคำนวณ $I_{ac} = 1.65 \times 26.8 = 44.2$ mA

เผื่อค่าไว้เป็น $I_{ac} = 100$ mA

ดังนั้นขนาดหม้อแปลงเป็น $0.1 \times 30 = 3$ VA

ดังนั้นใช้หม้อแปลงขนาดใหญ่กว่า 3 VA

6.3.2 ขนาดตัวเก็บประจุ

ค่า C_1 และ C_2 สำหรับ 7812 และ 9712 ดูได้จากเอกสารประกอบ (data sheet) ใช้ C1 = 0.22 μF และ C2 = 0.1 μF

สำหรับค่าตัวเก็บประจุสำหรับฟิลเตอร์คำนวณได้จาก

$$C = \frac{I_{dc}}{2fV_{r(p-p)}} \qquad 6.6$$

เมื่อ f = 50 Hz

หรืออาจคำนวณได้จาก

$$C = \frac{I_{dc}}{2\sqrt{3}\ fR_L r} \qquad 6.7$$

เมื่อ f = 100 Hz r = ค่าเปอร์เซ็นต์ความไม่เรียบ (percent ripple)

และ $$R_L = \frac{V_{dc}}{I_{dc}} \qquad 6.8$$

ดังนั้นคำนวณหาค่า C ที่ใช้ได้อย่างน้อยเท่ากับ

$$\frac{1}{2\sqrt{3}x100x(\frac{12}{0.027})x0.01} = 650\mu F$$

ในที่นี้ใช้ C = 2000 μF

6.3.3 ขนาดไดโอดที่ใช้

ในวงจรเรกติฟายแบบเต็มคลื่นไดโอดต้องทนแรงดันต้านกลับสูงสุด (PIV:Peak Invert Voltage) ได้อย่างน้อย 2 เท่าของค่าสูงสุดของแรงดันด้านทุติยภูมิเทียบกับแท็ปกลาง คือ PIV > $2V_m$ แต่เพื่อความปลอดภัยใช้ อย่างน้อย 3 เท่า PIV > $3V_m$

เนื่องจากแรงดันเทียบแท็ปกลางด้านทุติยภูมิเป็น 15 โวลท์ ดังนั้น

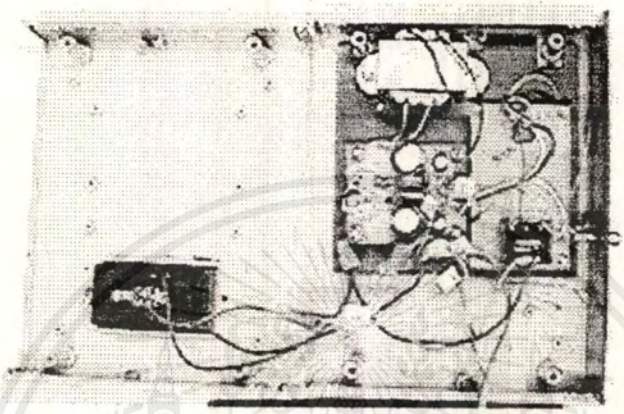
$$V_m = 5 \times \sqrt{2} = 21.2 \text{ โวลท์}$$

นั่นคือ

$$PIV > 3 \times 21.2 = 63.4 \text{ โวลท์}$$

ดังนั้นเลือกใช้ไดโอดเบอร์ 1N4002 ซึ่งมี PIV = 100 โวลท์

ไดโอด D_5 และ D_6 ใช้สำหรับป้องกันการคายประจุจาก C_2 และ C_3 ผ่านไอซีเรกกูเลต 7812 และ 7912 เบอร์ของ D_5 และ D_6 ดูได้จากเอกสารประกอบ (data sheet) ซึ่งได้ $D_5 = D_6$ เป็น 1N4001



รูปที่ 6.6 แสดงภายในของเครื่องลดทอนสัญญาณ

รูปที่ 6.7 แสดงภายนอกของเครื่องลดทอนสัญญาณ

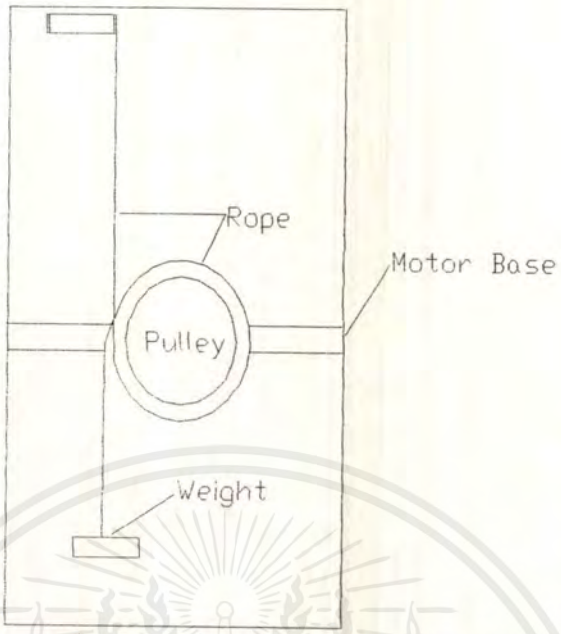
บทที่ 7
การทดลองและผลการทดลองในเวลาออฟไลน์ (Off Line)

ในการศึกษาการวัดความเร็วของมอเตอร์เหนี่ยวนำ โดยการนำตำแหน่งของโรเตอร์สล็อทฮาร์โมนิกส์มาใช้คำนวณในโครงงานนี้ ได้มีการศึกษาถึงพฤติกรรมของโรเตอร์สล็อทฮาร์โมนิกส์ในแง่ต่างๆกัน เพื่อที่จะสามารถทำนายตำแหน่งที่โรเตอร์สล็อทฮาร์โมนิกส์น่าจะเกิดขึ้นในมอเตอร์ตัวที่ต่างๆกันได้ด้วย และสามารถกำหนดช่วงขอบเขตของการเกิดโรเตอร์สล็อทฮาร์โมนิกส์ได้ ดังนั้นการทดลองจึงต้องมีการออกแบบเพื่อศึกษาพฤติกรรมของโรเตอร์สล็อทฮาร์โมนิกส์เมื่อมีการเปลี่ยนค่าพารามิเตอร์ต่างๆ เช่น การเปลี่ยนแปลงค่าโหลด ที่แรงดันไฟเกิน (Over Voltage) ที่แรงดันไฟตก (Under Voltage) และที่ตัวมอเตอร์ที่ต่างกัน ว่ามีผลอย่างไรต่อโรเตอร์สล็อทฮาร์โมนิกส์ ในการศึกษาของโครงงานนี้ได้ใช้การวิเคราะห์สเปกตรัมโดยใช้โปรแกรม Matlab ในการแปลงฟาสฟูเรียร์ ในโครงงานนี้จึงได้มีการศึกษาในหัวข้อต่างๆ ดังต่อไปนี้

1. การทดสอบหาความสัมพันธ์ระหว่างการเคลื่อนที่ของโรเตอร์สล็อทฮาร์โมนิกส์กับกระแสของโหลด
2. การทดสอบหาความเร็วของมอเตอร์เหนี่ยวนำชนิดโรเตอร์กรงกระรอก (Squirrel Cage) ขนาด 1 แรงม้า โดยใช้โหลดทางกล
3. การทดสอบหาความเร็วรอบมอเตอร์เหนี่ยวนำชนิดโรเตอร์กรงกระรอก ขนาด 1 แรงม้า , 2 แรงม้า และมอเตอร์เหนี่ยวนำชนิดโรเตอร์พันขดลวด (Wound Rotor)
4. การศึกษาผลของแรงดันไฟฟ้าเกินและแรงดันไฟฟ้าตกว่ามีผลต่อพฤติกรรมของโรเตอร์สล็อทฮาร์โมนิกส์หรือไม่อย่างไร

7.1 การทดสอบหาความสัมพันธ์ระหว่างการเคลื่อนที่ของสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์ กับกระแสโหลดของมอเตอร์เหนี่ยวนำขนาด 1 แรงม้า

การทดลองทำโดยการป้อนค่าศักดาไฟฟ้าที่พิกัดให้กับมอเตอร์เหนี่ยวนำขนาด 1 แรงม้า แล้วค่อยๆทำการเพิ่มโหลดโดยการใส่น้ำหนักถ่วงเพิ่มที่มู่เล่ โดยมีการจัดอุปกรณ์ ดังรูปที่ 7.1



รูปที่ 7.1 แสดงการต่ออุปกรณ์การทดลอง

การทดลองจะเริ่มต้นบันทึกผลตั้งแต่ NO – load แล้วจึงเพิ่มน้ำหนักที่ถ่วงมอเตอร์เข้าไปเป็นลำดับดังนี้ 3 , 6 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16 , 17 กิโลกรัม บันทึกค่ากระแสจากแอมป์มิเตอร์ , บันทึกค่าความเร็วของมอเตอร์จากทาโคมิเตอร์ , บันทึกข้อมูลลง disk เป็น file . txt แบบ verbose ข้อควรระวังสำหรับการทดลองก็คือต้องบันทึกความเร็วของมอเตอร์ในขณะที่saveข้อมูลลงdisk หลังจากที่ได้ข้อมูลมาจะทำการหาสเปกตรัมของข้อมูลโดยโปรแกรมMatlab แล้วก็พิจารณาว่ามีฮาร์โมนิกส์ตัวใดที่มีความถี่สัมพันธ์ตามสมการ

$$f_{sh} = \left[\frac{Z}{P}(1-s) \pm 1\right] f_0$$

ผลที่เกิดขึ้นคือที่ค่าโหลดต่างๆกันฮาร์โมนิกส์ที่เป็นไทม์ฮาร์โมนิกส์จะอยู่กับที่ แต่จะมีฮาร์โมนิกส์อยู่ 1 คู่ที่เคลื่อนที่และมีความถี่เป็นดังสมการด้านบน ดังนั้นเราจึงสามารถระบุได้ว่าฮาร์โมนิกส์ตัวนั้นเป็นโรเตอร์สล็อทฮาร์โมนิกส์และมีพฤติกรรมการเคลื่อนที่ดังตารางด้านล่าง

ตารางที่ 7.1 แสดงตำแหน่งของโรเตอร์สล็อทฮาร์โมนิกส์เมื่อโหลดเปลี่ยนแปลงจนถึงค่าพิกัด

กระแสโหลด (%)	37.14	44.29	54.29	60.00	65.71	71.43	74.29	80.00	85.71	91.43	100.0
ตำแหน่งของ RSH(Hz)	933.8	916.7	893.6	883.8	867.9	854.5	852.1	828.8	811.8	784.9	761.7

สรุป จะเห็นได้ว่าโรเตอร์สล็อทฮาร์โมนิกส์ จะมีความถี่ลดลงเมื่อกระแสโหลดมีค่าเพิ่มขึ้น

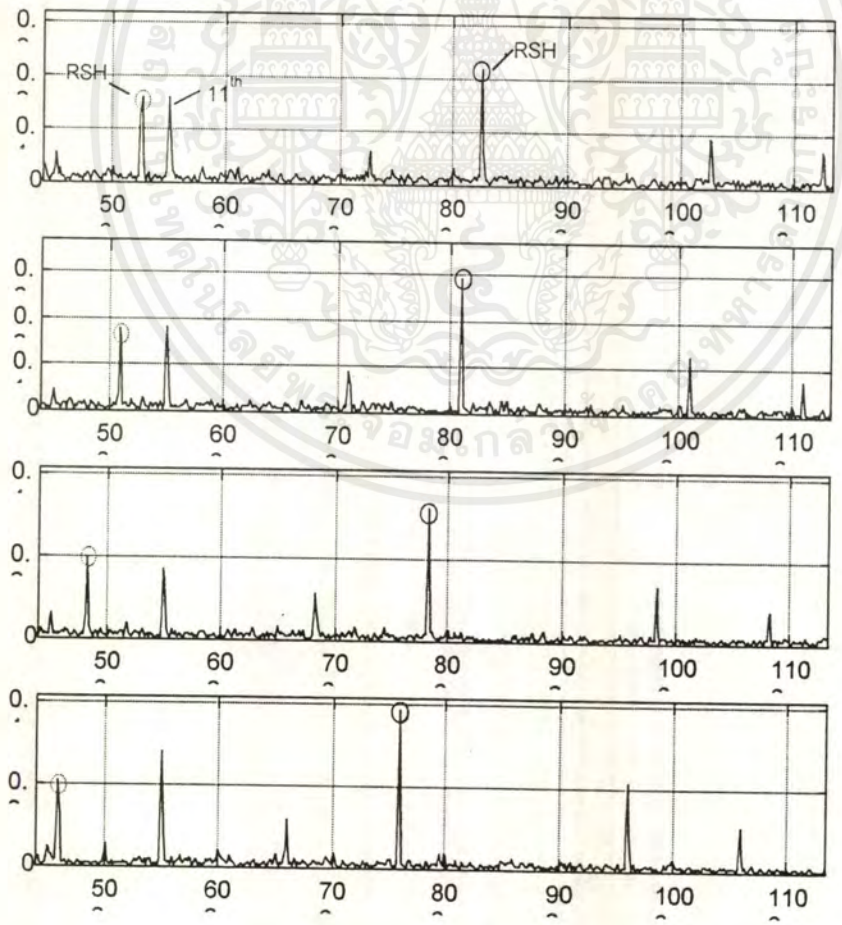
7.2 การหาความเร็วรอบของมอเตอร์เหนี่ยวนำชนิดโรเตอร์กรงกระรอก (Squirrel Cage)

ความเร็วที่ได้จากการคำนวณสามารถหาได้โดยการนำข้อมูลมาผ่านฮานนิ่งวินโดว์ แล้วนำมาแปลง FFT จะได้สเปกตรัมออกมาจะเป็นกราฟในโดเมนความถี่ ตรวจสอบว่ามีฮาร์โมนิกส์ตัวใด ที่อยู่ในช่วงที่น่าจะเกิดโรเตอร์สลิ็อทฮาร์โมนิกส์ ได้แล้วพิจารณาค่าฮาร์โมนิกส์ที่มีค่าแอมพลิจูดสูงสุดในย่านนั้นเมื่อพบแล้วก็ตรวจสอบว่าฮาร์โมนิกส์ดังกล่าวเป็นไทม์ฮาร์โมนิกส์ หรือไม่ถ้าไม่เป็นก็แสดงว่าเป็นโรเตอร์สลิ็อทฮาร์โมนิกส์ แล้วนำค่าความถี่ของโรเตอร์ฮาร์โมนิกส์ ; f_{sh} แทนค่าในสูตร

$$f_{sh} = \left[\frac{Z}{P}(1-s) \pm 1\right] f_0$$

7.2.1 มอเตอร์เหนี่ยวนำกรงกระรอกขนาด 1 แรงม้าใช้โหลดทางกล

ทดลองโดยการนำมูเล่มาต่อกับแกนเพลามอเตอร์กรงกระรอกขนาด 1 แรงม้าแล้วเพิ่มโหลดโดยการใส่เหล็กถ่วงที่มู่เล่ ผลตัวอย่างการแปลงสเปกตรัมข้อมูลแสดงในรูปที่ 7.2



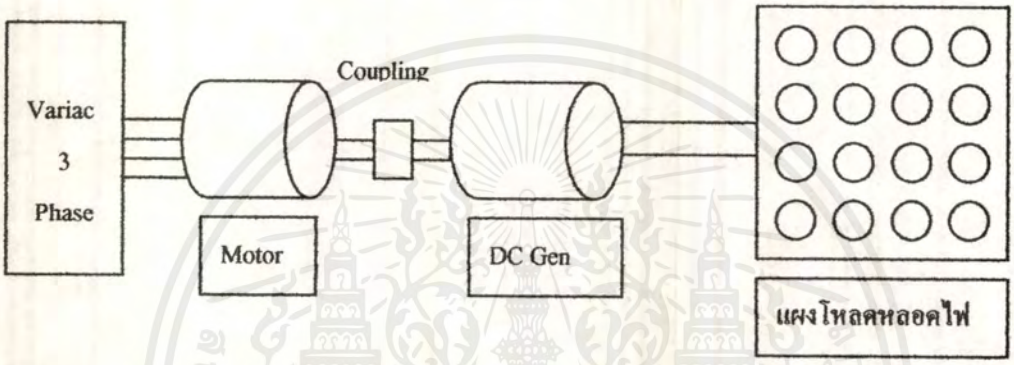
รูปที่ 7.2 แสดงตัวอย่างสเปกตรัมกระแสมอเตอร์ที่โหลด 80 , 85.7 , 91.4 , 100%

ตารางที่7.2 แสดงค่าความความเร็วที่ได้จากการคำนวณและค่าความเร็วจริงที่เปอร์เซนต์พิกัดโหลดต่างๆ

น้ำหนัก (kg)	% load	ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ ; f_{sh} (Hz)	ความเร็ว ที่ได้จากการคำนวณ (RPM)	ความเร็วจริงจากทาโคมิเตอร์ (RPM)	ค่าความคลาดเคลื่อน (% Error)
No-load	-	-	-	1499	-
3	37.14	933.8	1473	1474	0.07
6	44.29	916.7	1444.58	1450	0.37
9	54.29	893.6	1405.9	1418	0.85
10	60.00	883.8	1389.65	1405	1.09
11	65.71	867.9	1363.2	1384	1.50
12	71.43	854.5	1340.82	1367	1.77
13	74.29	852.1	1336.75	1355	1.81
14	80.00	832.8	1303.2	1328	2.03
15	85.71	811.8	1269.6	1303	2.49
16	91.43	784.9	1224.8	1274	2.79
17	100	761.7	1186.2	1223	3.01

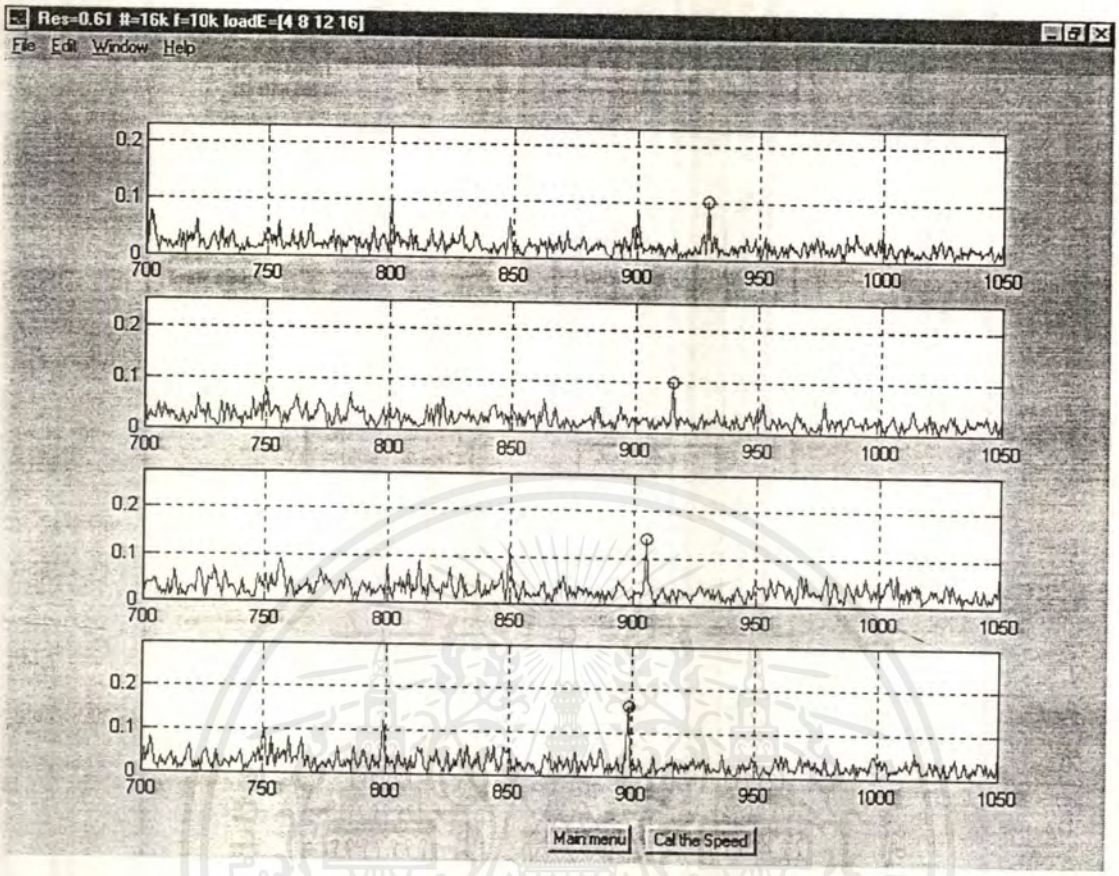
7.2.2 มอเตอร์เหนี่ยวนำกรงกระรอกขนาด 1 แรงม้าใช้โหลดทางไฟฟ้า

การทดลองใช้โหลดทางไฟฟ้านี้ ทดลองโดยการนำมอเตอร์กระแสตรงมาต่อพ่วงกัน (coupling motor) เข้ากับมอเตอร์เหนี่ยวนำ จากนั้นจ่ายไฟจากแหล่งจ่าย 3 เฟส (Variac 3 phase) เข้าที่มอเตอร์เหนี่ยวนำกรงกระรอก 1 แรงม้า ใช้มอเตอร์กระแสตรงทำหน้าที่เป็นโหลดให้กับมอเตอร์เหนี่ยวนำ จ่ายไฟให้กับโหลดหลอดไฟ



รูปที่ 7.3 แสดงการต่ออุปกรณ์เพื่อทดลองโหลดทางไฟฟ้า

จากบันทึกผลการทดลองและทำการประมวลผลโดยการใช้โปรแกรม Matlab แล้วจะได้ผลการแปลงสเปกตรัมดังรูปที่ 7.4 สามารถเห็นการเคลื่อนที่ของโรเตอร์สล็อทฮาร์โมนิกส์ได้เปรียบเทียบกับไทม์ฮาร์โมนิกส์ ที่ไม่มีการเคลื่อนที่ และตารางที่ 7.3 แสดงผลการคำนวณความเร็วจากโปรแกรม Matlab



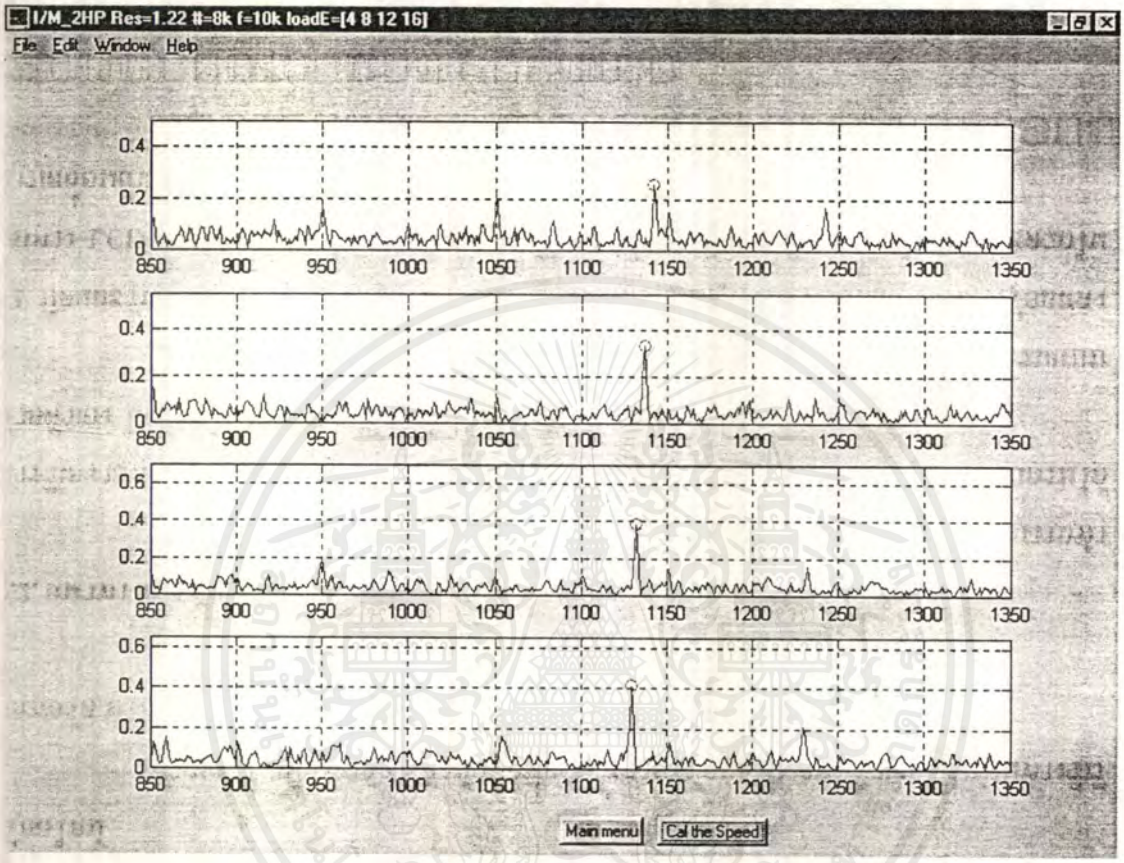
รูปที่ 7.4 แสดงสเปกตรัมของกระแสมอเตอร์เหนี่ยวนำ 1 แรงม้า เมื่อใส่โหลดทางไฟฟ้า

ตารางที่ 7.3 แสดงความเร็วที่ได้จากการคำนวณ และค่าความผิดพลาดที่เกิดขึ้น จากการใช้โหลดทางไฟฟ้ากับมอเตอร์กรงกระรอกขนาด 1 แรงม้า

% Load	จำนวนหลอดไฟ	ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ (Hz)	ความเร็วที่ได้จากการใช้โปรแกรม Matlab (RPM)	ความเร็วที่ได้จากการใช้ทาโคมิเตอร์	ค่าความคลาดเคลื่อน %
37.1	4	929.5654	1465.94	1472	0.41
44.29	8	916.1377	1443.56	1451	0.512
48.62	12	905.7617	1426.27	1438	0.81
50.64	16	899.6582	1416.1	1430	0.97

7.2.3 มอเตอร์เหนี่ยวนำขนาด 2 แรงม้าใช้โหลดทางไฟฟ้า

การทดลองจะเหมือนกันกับการทดลองในหัวข้อ 7.2.2 แต่เปลี่ยนจากมอเตอร์กรงกระรอก 1 แรงม้าเป็น 2 แรงม้า ซึ่งจะได้ผลการทดลองเป็นดังที่แสดงในรูปที่ 7.5 และตารางที่ 7.4



รูปที่ 7.5 แสดงสเปกตรัมของกระแสมอเตอร์เหนี่ยวนำ 2 แรงม้า เมื่อใช้โหลดทางไฟฟ้า

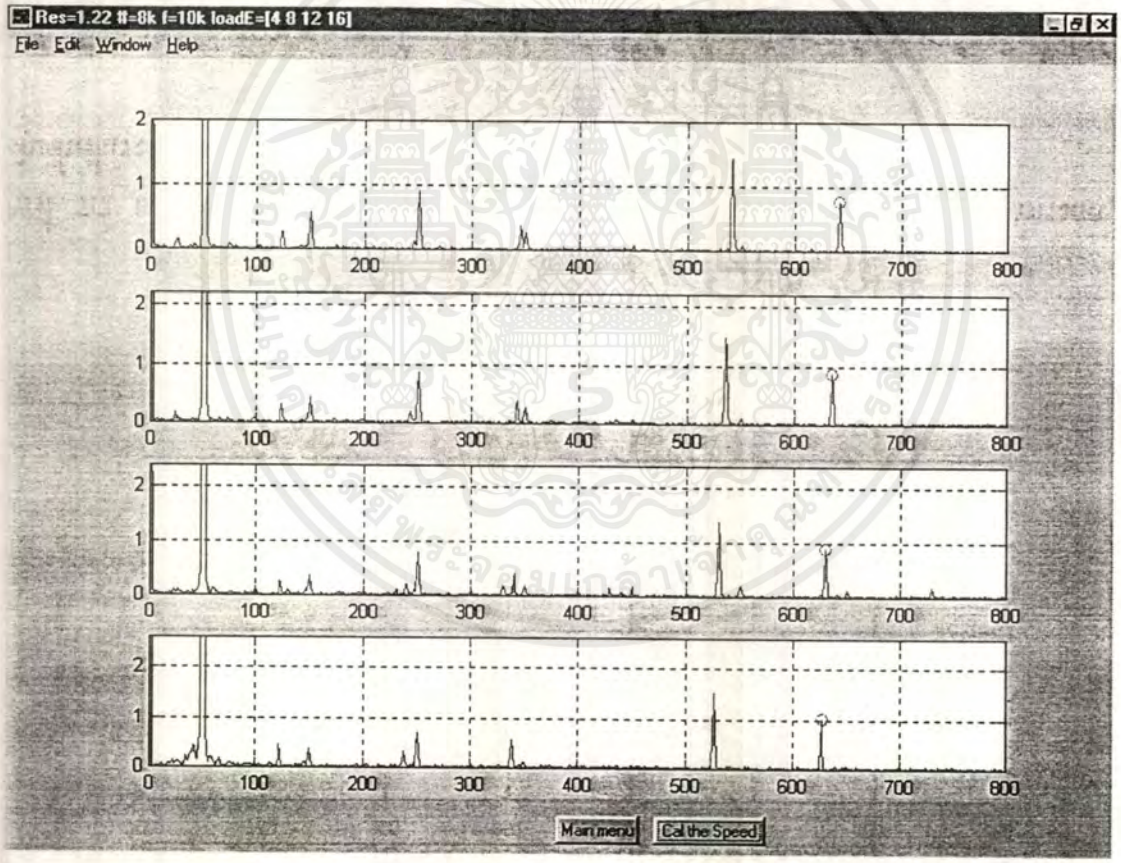
ซึ่งจะสังเกตได้ว่าโรเตอร์สล็อทฮาร์โมนิกส์ที่เราสามารถที่จะเห็นได้ชัด จะต่างกับที่เราเห็นได้จากมอเตอร์ขนาด 1 แรงม้า เนื่องจากมอเตอร์คนละตัวกันจะมีค่าพารามิเตอร์ที่ต่างกัน โดยที่มอเตอร์ขนาด 2 แรงม้านี้โรเตอร์สล็อทฮาร์โมนิกส์ที่สามารถเห็นได้ชัดคือที่เงื่อนไข $n_w = 5$, $k = 1$, $n_d = 0$ จากสมการ

$$f_{sh} = f_o\left((kZ + n_d)\left(\frac{1-s}{P}\right) + n_w\right)$$

<u>ตารางที่7.4</u> แสดงค่าความคลาดเคลื่อนของความเร็วที่ได้จากการคำนวณกับความเร็วจริงที่เปอร์เซนต์พิกัดโหลดต่างๆ ของมอเตอร์ 2 แรงม้าโหลดทางไฟฟ้า

จำนวนหลอด (หลอด)	ความถี่ของโรเตอร์สล็อตฮาร์โมนิกส์ ; f_{sh} (Hz)	ความเร็ว ที่ได้จากการคำนวณ (RPM)	ความเร็วจริงจากทาโคมิเตอร์ (RPM)	ค่าความคลาดเคลื่อน (% Error)
33.87	1142.6	1487.63	1488	0.02
36.29	1137.7	1479.49	1481	0.10
38.30	1132.8	1471.35	1475	0.25
40.32	1130.4	1467.29	1472	0.32

7.3 การหาความเร็วรอบของมอเตอร์เหนี่ยวชนิดโรเตอร์พันขดลวด (Wound Rotor)



รูปที่ 7.6 แสดงสเปกตรัมของกระแสมอเตอร์เหนี่ยวนำชนิดของโรเตอร์พันขดลวด เมื่อใช้โหลดทางไฟฟ้า

และจากการสังเกตเราจะพบว่าโรเตอร์สล็อทฮาร์โมนิกส์ที่เกิดขึ้นจะเกิดที่ความถี่ ในช่วง 500-700 Hz เนื่องจากค่าพารามิเตอร์ของมอเตอร์ที่เปลี่ยนไปโดยที่ $k = 1$, $n_d = 0$, $n_w = 1$ จากสมการ

$$f_{sh} = f_o\left(\left(kZ + n_d\right)\left(\frac{1-s}{P}\right) + n_w\right)$$

ตารางที่7.5 แสดงค่าความคลาดเคลื่อนของความเร็วที่ได้จากการคำนวณกับความเร็วจริงที่ค่าโหลด

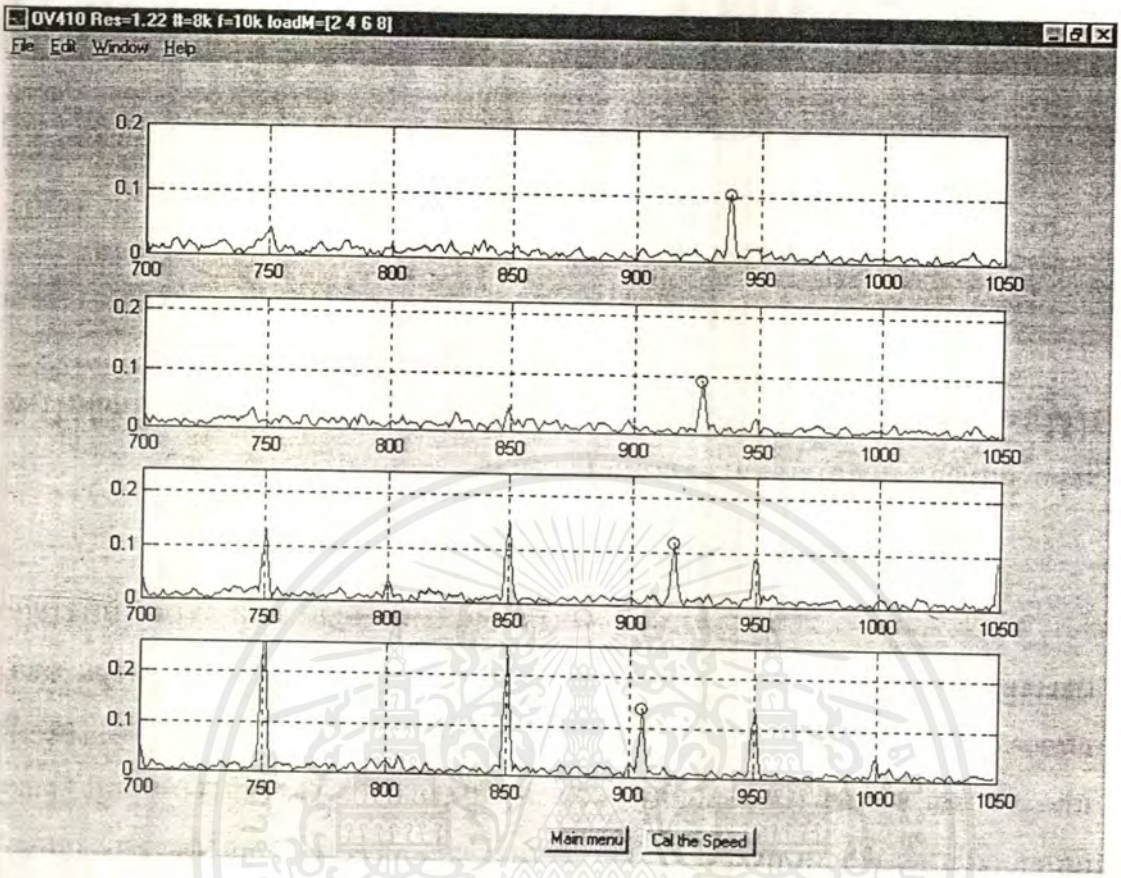
จำนวนหลอด (หลอด)	ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ ; f_{sh} (Hz)	ความเร็วที่ได้จากการคำนวณ (RPM)	ความเร็วจริงจากทาโคมิเตอร์ (RPM)	ค่าความคลาดเคลื่อน (% Error)
4	642.0898	1480.22	1478	0.15
8	635.9863	1464.97	1462	0.20
12	631.1035	1452.76	1450	0.19
16	627.4414	1443.60	1440	0.25

7.4 การหาความเร็วรอบของมอเตอร์เหนี่ยวนำเมื่อแรงดันไฟตกหรือแรงดันไฟเกิน

เนื่องจากในระบบไฟฟ้าจริงๆแล้วจะมีเหตุการณ์ของแรงดันไฟฟ้าที่ไม่ปกติอยู่เสมอ เช่น การเกิดแรงดันไฟตกและแรงดันไฟเกิน ดังนั้นเราจึงต้องมีการศึกษาถึงผลของแรงดันไฟที่ผิดปกติที่ว่าจะส่งผลกระทบอย่างไรต่อโรเตอร์สล็อทฮาร์โมนิกส์ ซึ่งเราจะทำการศึกษาที่แรงดันไฟเกินที่ 410 โวลท์ และแรงดันไฟตกที่ 350 โวลท์ ในการทดลองจะใช้มอเตอร์กรงกระรอกขนาด 1 แรงม้า และทำการใส่ทางกล เนื่องจากมอเตอร์ 1 แรงที่ใช้ในการทดสอบสามารถทดสอบได้ง่าย

7.4.1ที่แรงดันไฟเกิน 410 V

ทำการจ่ายศักดาเข้ามอเตอร์เหนี่ยวนำกรงกระรอก 1 แรงม้าด้วยแรงดัน 410 โวลท์ แล้วทำการบันทึกผล และประมวลผลโดยการใช้โปรแกรม Matlab ซึ่งผลจากการแปลงสเปกตรัมจะได้ดังรูปที่ 7.7 จะเห็นการเคลื่อนที่ของโรเตอร์สล็อทฮาร์โมนิกส์ และตารางที่ 7.6 แสดงค่าความเร็วที่ได้จากการคำนวณและค่าความคลาดเคลื่อนเมื่อเปรียบเทียบกับความเร็วที่ได้จากการใช้ทาโคมิเตอร์

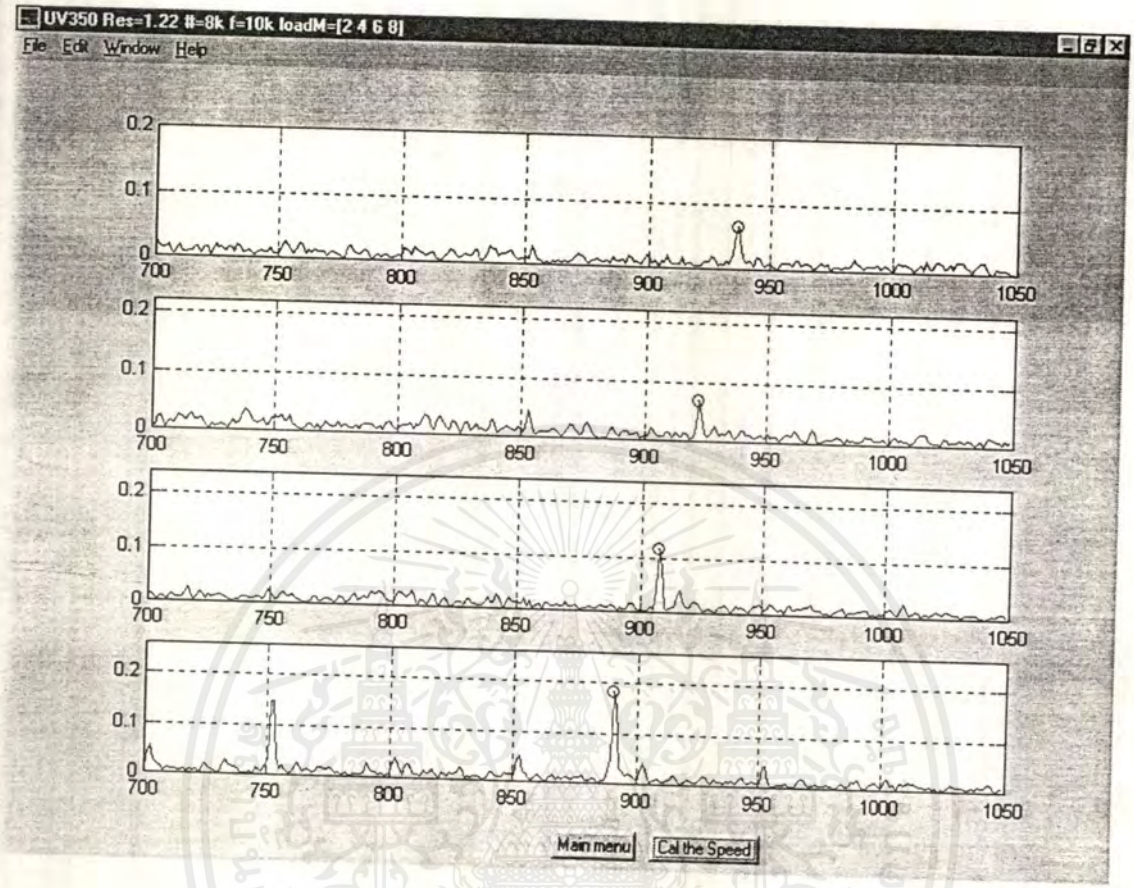


รูปที่ 7.7 แสดงสเปกตรัมของกระแสมอเตอร์เหนี่ยวนำชนิดโรเตอร์กรงกระรอกเมื่อโหลดทางกลเพิ่มขึ้นที่แรงดันไฟฟ้าเกิน 410 V

ตารางที่7.6แสดงค่าความคลาดเคลื่อนของความเร็วที่ได้จากการคำนวณกับความเร็วจริงที่เปอร์เซนต์พิกัดโหลดต่างๆเมื่อแรงดันไฟฟ้าเกิน

% load	น้ำหนักถ่วง (Kg)	ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ ; f_{sh} (Hz)	ความเร็ว ที่ได้จากการคำนวณ (RPM)	ความเร็วจริงจาก ทาโกมิเตอร์ (RPM)	ค่าความคลาดเคลื่อน (% Error)
35.3	2	938.72	1481.2	1481	0.01
38.6	4	927.73	1462.89	1467	0.28
43.2	6	917.97	1446.61	1453	0.44
49.3	8	905.76	1426.27	1436	0.67

7.4.2 ที่แรงดันไฟตก 350 V



รูปที่ 7.8 แสดงสเปกตรัมของกระแสมอเตอร์เหนี่ยวนำชนิดโรเตอร์กรงกระรอกเมื่อโหลดทางกลเพิ่มขึ้นที่แรงดันไฟฟ้าตก 350 V

ตารางที่7.7แสดงค่าความคลาดเคลื่อนของความเร็วที่ได้จากการคำนวณกับความเร็วจริงที่เปอร์เซนต์พิกัดโหลดต่างๆเมื่อแรงดันไฟฟ้าตก

% load	น้ำหนักถ่วง (Kg)	ความถี่ของโรเตอร์สล็อทฮาร์โมนิกส์ ; f_{sh} (Hz)	ความเร็ว ที่ได้จากการคำนวณ (RPM)	ความเร็วจริงจาก ทาโคมิเตอร์ (RPM)	ค่าความคลาดเคลื่อน (% Error)
38.5	2	937.50	1479.17	1481	0.12
40.1	4	922.85	1454.75	1461	0.42
45.2	6	908.20	1430.34	1440	0.67
51.9	8	891.11	1401.83	1417	1.07

7.5 วิเคราะห์และสรุปผลการทดลอง

1. จากผลการทดลองจะเห็นได้ว่าเมื่อโหลดมีค่ามากขึ้น สัญญาณของโรเตอร์สล็อทฮาร์โมนิกส์จะมีความถี่ลดลง และขนาดของโรเตอร์สล็อทฮาร์โมนิกส์ก็มีค่าเพิ่มมากขึ้น

2. การใช้โหลดทางกลมีผลเสียคือการเกิดความร้อนเมื่อใส่ค่าโหลดสูงๆกระแสไม่นิ่ง

3.การใช้โหลดทางไฟฟ้าจะทำให้เกิดการรบกวนของสัญญาณที่เกิดจากการสปาร์คของแปรงถ่าน ทำให้มีค่าความคลาดเคลื่อนมากกว่าการใช้โหลดทางกลที่เปอร์เซนต์สลิปเดียวกัน และจำเป็นต้องมีการใช้ข้อมูลจำนวนมากช่วยในการประมวลผล

4. ค่าความคลาดเคลื่อนการคำนวณความเร็วจะมีค่าสูงขึ้นเมื่อโหลดมีค่ามากขึ้น

5. การใช้มอเตอร์เหนี่ยวนำชนิดพันขดลวด จะได้ผลการทดลองที่มีความคลาดเคลื่อนน้อยกว่า โรเตอร์สล็อทฮาร์โมนิกส์มีขนาดสูงสามารถตรวจจับได้ง่ายเนื่องจาก มีความเป็นร่องสล็อทที่ชัดเจนกว่ามอเตอร์แบบกรงกระรอก

6. มอเตอร์แต่ละตัวมีค่าพารามิเตอร์ที่ต่างกันออกไป ดังนั้นในการใช้สูตรระบุความสัมพันธ์จะต้องมีการทดสอบหาความสัมพันธ์ที่ถูกต้องเสียก่อน เพราะว่าโรเตอร์สล็อทฮาร์โมนิกส์ของมอเตอร์แต่ละตัวที่สามารถตรวจจับได้จะเกิดที่ความถี่ไม่เท่ากันเนื่องจากมีค่าพารามิเตอร์ที่ต่างกัน เช่น มอเตอร์ 1 แรงม้าเกิดโรเตอร์สล็อทฮาร์โมนิกส์ที่ช่วง 700-950 Hz แต่ถ้าเป็นมอเตอร์ 2 แรงม้าจะสามารถพบได้ในช่วงความถี่ ประมาณ 1100 Hz

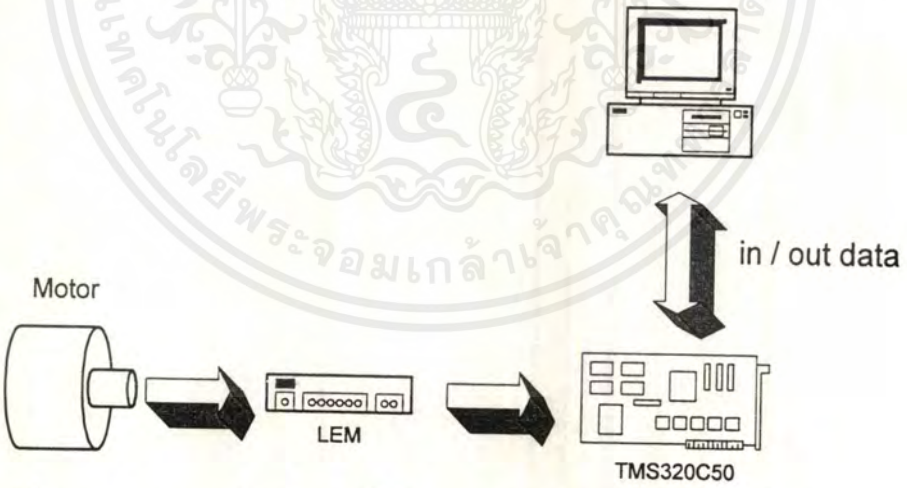
7.การเกิดความผิดปกติของแรงดันไฟฟ้าทั้งไฟตกและไฟเกินจะทำให้เกิดค่าความคลาดเคลื่อนที่สูงมากขึ้น แต่ตัวโรเตอร์สล็อทฮาร์โมนิกส์ยังสามารถเห็นได้ชัดเจนและมีพฤติกรรมต่างๆ เหมือนเดิม

8. การวัดสัญญาณของโรเตอร์สล็อทฮาร์โมนิกส์นั้นจะต้องเลือกค่าอัตราการสุ่มข้อมูลและค่าจำนวนข้อมูลที่เหมาะสม เพื่อที่จะทำให้ได้ค่าความถี่รีโซลูชั่น (Frequency Resolution) ที่ดีที่สุด มีค่าความถูกต้องมากที่สุดแต่ต้องไม่ใช้ข้อมูลมากเกินไปจนทำให้เสียเวลาการประมวลผลนาน และความถี่การสุ่มข้อมูลต้องไม่น้อยจนทำให้เกิดการซ้อนทับของความถี่(aliasing) และค่าความถี่รีโซลูชั่นที่ดีที่สุดจากการทดสอบคือที่ประมาณ 0.31 Hz โดยใช้ข้อมูล 8192 ข้อมูล ที่อัตราการสุ่มข้อมูลเป็น 10 Ksa/S

บทที่ 8
การทดลอง และผลการทดลองที่เวลาจริง (Real Time)

เนื่องจากในการศึกษาการวัดความเร็วของมอเตอร์เหนี่ยวนำโดยการใช้ตำแหน่งของโรเตอร์สล็อทฮาร์โมนิกส์มาคำนวณได้กล่าวถึงในบทที่ 7 นั้นเป็นเพียงการศึกษาแบบออฟไลน์ (Off Line) เท่านั้นไม่สามารถที่จะนำมาใช้จริงได้ ในบทนี้จะกล่าวถึงการนำวิธีการวัดความเร็วมอเตอร์ที่เวลาจริง (Real Time) ซึ่งสามารถพัฒนาไปสู่การใช้งานจริงได้

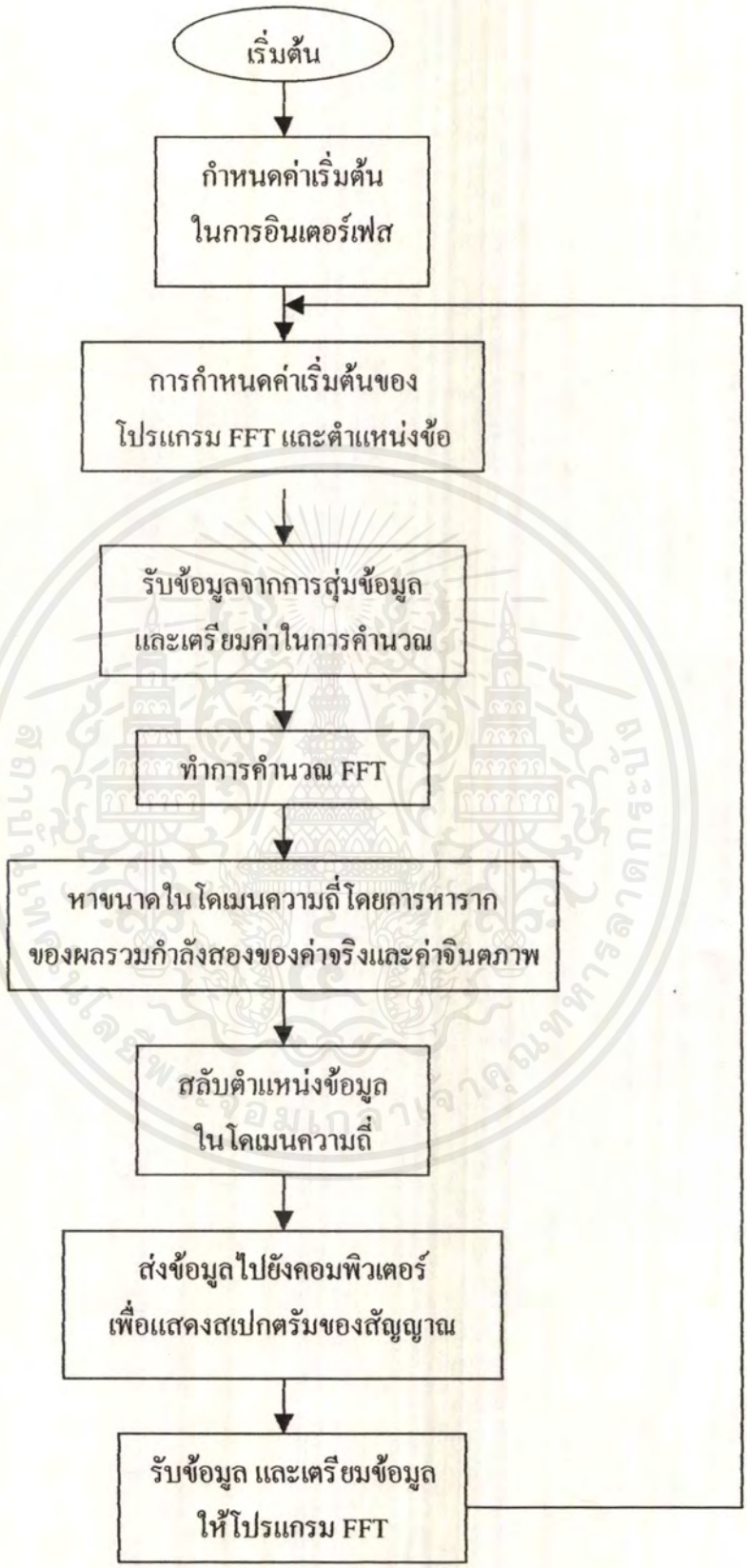
ในการทดลองนี้จะใช้อุปกรณ์ LEM ทำการลดทอนกระแสแล้วทำการเปลี่ยนสัญญาณกระแสให้เป็นสัญญาณแรงดันไม่เกิน $\pm$ 3 V เพื่อนำไปประมวลผลโดยบอร์ด TMS320C50 DSK ซึ่งเป็นบอร์ดที่ใช้การประมวลผลเชิงตัวเลข (DSP: Digital Signal Processing) ของบริษัท Texas Instrument โดยจะทำการสุ่มข้อมูลเข้ามาเป็นช่วง แล้วทำการแปลงข้อมูลจากโดเมนเวลาไปยังโดเมนความถี่ (FFT) จากนั้นจะทำการส่งข้อมูลที่แปลงเป็นโดเมนความถี่แล้วไปยังเครื่องคอมพิวเตอร์ทำการแสดงผลกราฟฟิกออกทางหน้าจอคอมพิวเตอร์



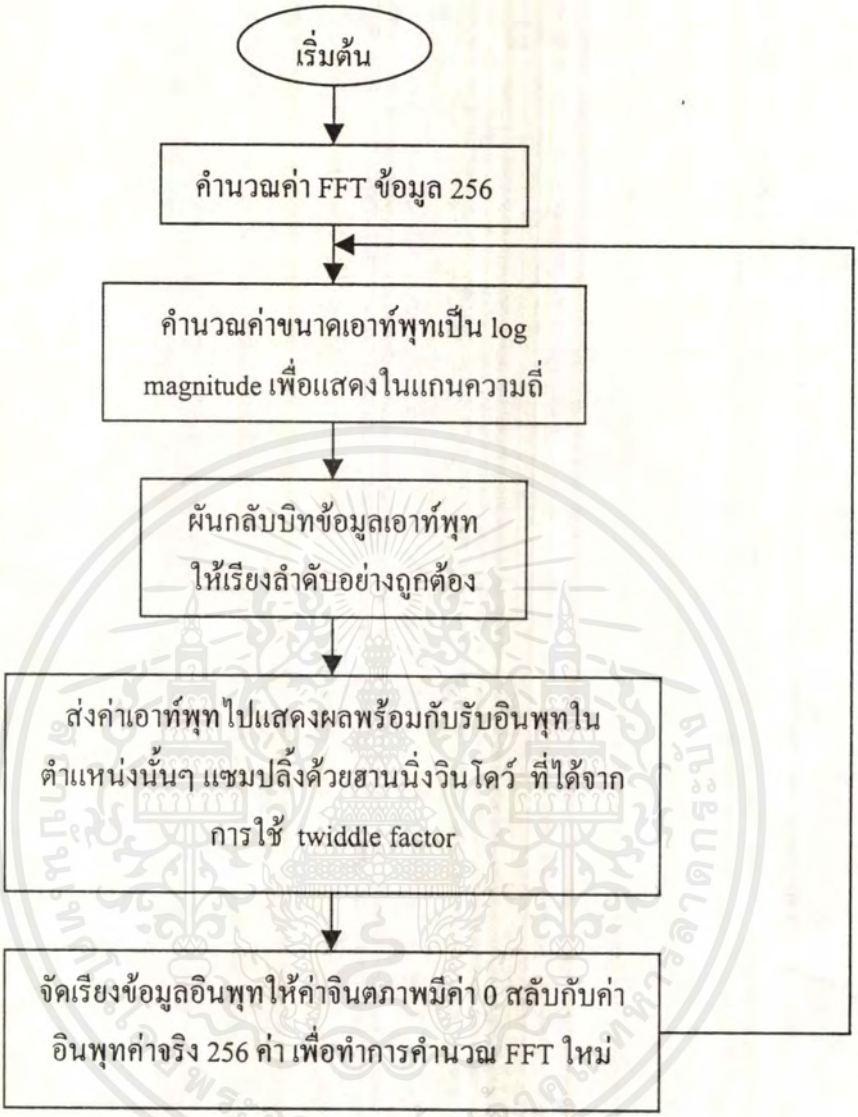
รูปที่ 8.1 รูปแสดงการต่ออุปกรณ์ที่ใช้ในการศึกษาที่เวลาจริง (Real Time)

8.1 การแปลงสเปกตรัมจากโดเมนเวลาไปยังโดเมนความถี่ (FFT)

การแปลงสเปกตรัมจะอาศัยการแปลงฟาสท์ฟูเรียร์ทรานฟอร์มแบบลดทอนทางเวลา (Decimation In Frequency Fast Fourier Transform) โดยใช้บอร์ด TMS320C50 DSK เป็นตัวประมวลผลสัญญาณเชิงตัวเลข โดยการเขียนโปรแกรมจะใช้ภาษาของตัวบอร์ดเอง สามารถแสดงเป็นโฟล์วชาร์ตได้ดังนี้



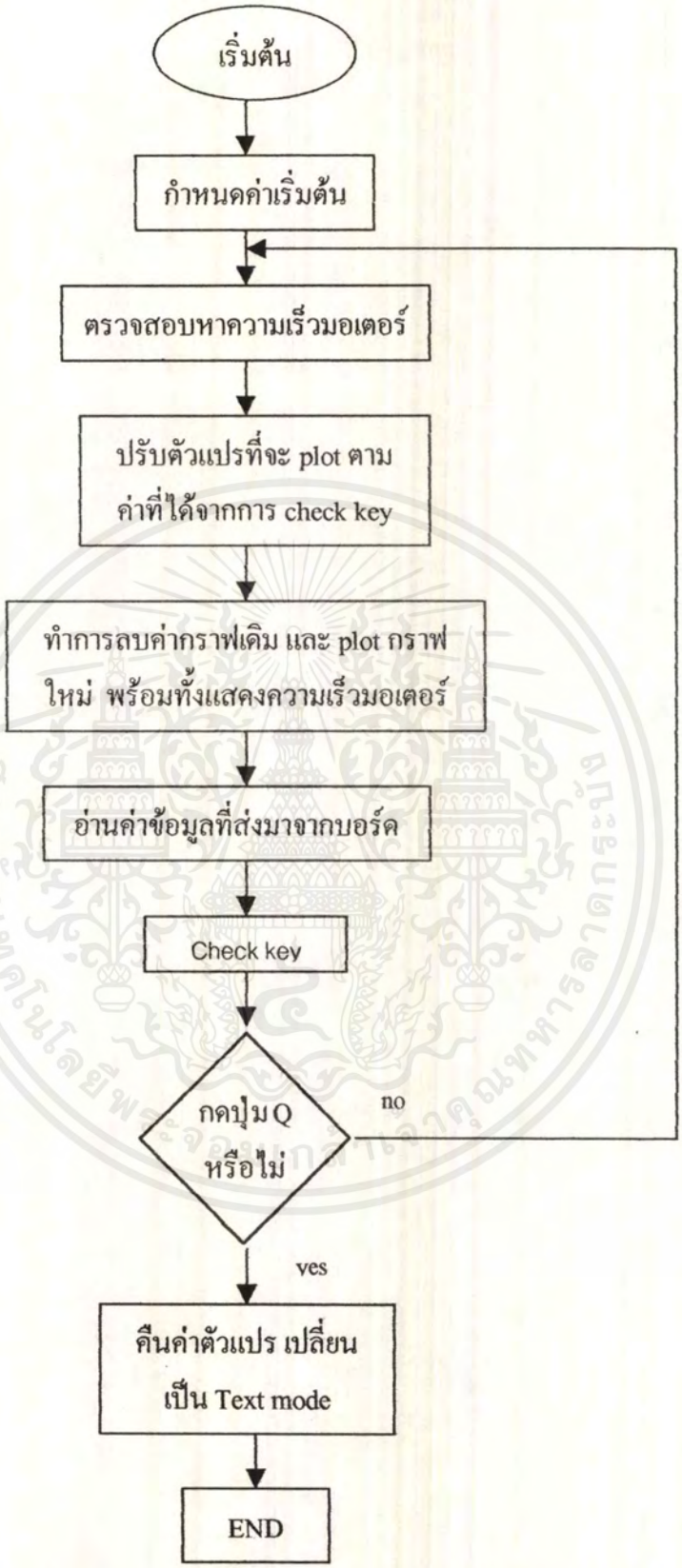
รูปที่ 8.2 แผนภาพแสดงการทำงานของโปรแกรม RSH.asm



รูปที่ 8.3 แผนภาพการแสดงการคำนวณฟาสท์ฟูเรียร์ทรานฟอร์ม

8.2 การแสดงผลการแปลงสเปกตรัมออกทางหน้าจอคอมพิวเตอร์

ในการแสดงผลทำโดยรับข้อมูลที่ผ่านการแปลงฟาสท์ฟูเรียร์แล้ว จากบอร์ด TMS320C50 DSK จากทางพอร์ทขนาน RS232 จากนั้นทำการแสดงผลข้อมูลที่รับมาโดยใช้ภาษา C++ เขียนโปรแกรมในการแสดงผลทางด้านกราฟฟิกซึ่งมีลำดับการทำงาน ดังนี้

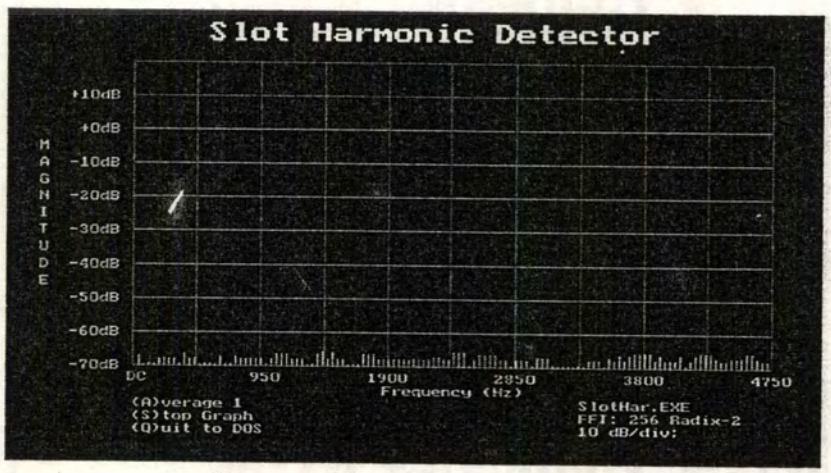


รูปที่ 8.4 โฟล์วชาร์ตแสดงผลทางด้านกราฟฟิกออกทางหน้าจอคอมพิวเตอร์

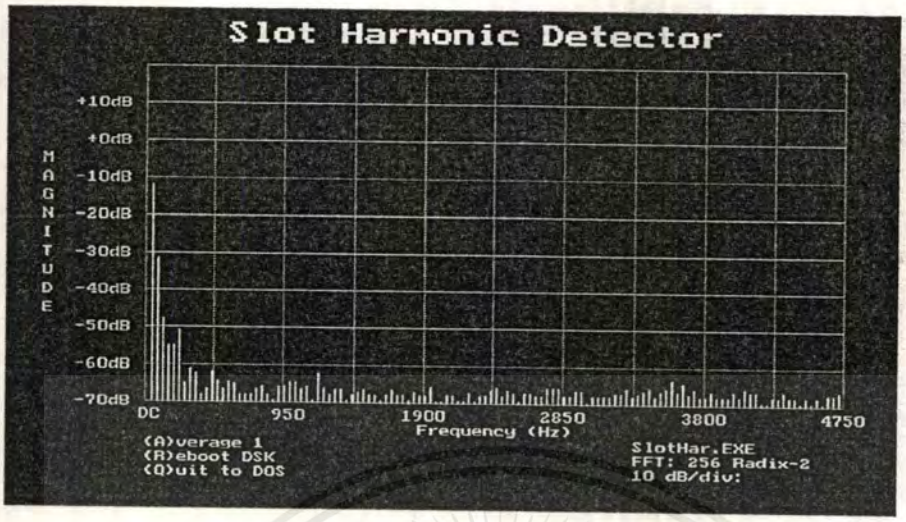
รูปที่ 8.5 โฟลว์ชาร์ตแสดงการ Initial ค่าในการแสดงผลทางด้านกราฟฟิก

8.3 ผลการทดลอง

ในการทดลองการศึกษาพฤติกรรมในลักษณะที่เวลาจริง จะทำการทดลองโดยการสุ่มข้อมูลเข้ามาที่อัตราการสุ่ม 10.286 ksa/s ใช้มอเตอร์ขนาด 1 แรงม้า ทำการเพิ่มโหลดทางกลผลการแปลงสเปกตรัม ดังนี้



รูปที่ 8.6 รูปแสดงสเปกตรัมของการใช้บอร์ด TMS32C50 ขณะยังไม่มีการวัดสัญญาณ



รูปที่ 8.7 รูปแสดงผลของการใช้บอร์ด TMS32C50 ขณะวัดสัญญาณจากมอเตอร์

8.4 การวิเคราะห์ และการสรุปผลการทดลอง

จากการทดลอง รูปสเปกตรัมที่ได้จากการประมวลผลโดยบอร์ด TMS320C50 นั้น สัญญาณที่เข้ามามีการเปลี่ยนแปลงตลอดเวลา(Real Time) ซึ่งได้ทำการจับภาพมาในช่วงเวลาหนึ่ง (holding) จากรูปที่ 8.6 และ 8.7 จะเห็นว่ากราฟที่ได้ไม่ละเอียดเพียงพอที่จะสามารถระบุได้ว่าฮาร์โมนิกส์ตัวใดเป็นโรเตอร์สล็อทฮาร์โมนิกส์ ทั้งนี้เนื่องจากจำนวนของข้อมูลที่ใช้ในการนำมาประมวลผลมีเพียงแค่ 256 ค่าเท่านั้นซึ่งจะทำให้กราฟที่ได้จากการแปลงแล้วจะมีความละเอียดไม่เพียงพอที่จะสามารถเห็นและจับโรเตอร์สล็อทฮาร์โมนิกส์ได้ แต่ในการประมวลผลแบบเวลาไม่ต่อเนื่อง(Off Line)โดยใช้โปรแกรม MATLAB จำนวนข้อมูลที่นำมาคำนวณจากดิจิตอลสตอเรจสโคปมีมากพอ เพราะฉะนั้นจึงสามารถจับสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์ได้ แต่ในการระบุตำแหน่งโรเตอร์สล็อทฮาร์โมนิกส์ได้ชัดเจนนั้น จะต้องใช้ข้อมูลอย่างน้อย 4K (4096 ค่า) ที่อัตราการแซมปลิ้ง 10 KSa/S

เพราะฉะนั้นในการพัฒนาให้เป็นการประมวลผลแบบเรียลไทม์ ที่สามารถตรวจสอบหาค่าความเร็วได้จริงนั้น จะต้องใช้บอร์ดที่มีแรมภายในมากขึ้นหรือต่อแรมภายนอกเพิ่มให้พอที่จะเก็บข้อมูลและทำการคำนวณ และต้องแก้ไขโปรแกรมภาษาแอสเซมบลีที่ใช้กับบอร์ดให้มีการใช้จำนวนข้อมูลมากขึ้นในการประมวลผล และเพื่อให้มีการหน่วงเวลา(Delay Time)ที่น้อยที่สุด ควรใช้บอร์ดที่มีความเร็วสูง

บทที่ 9
ปัญหา แนวทางการแก้ปัญหาและการพัฒนา

9.1 ปัญหา

9.1.1 ปัญหาในการศึกษาพฤติกรรมแบบออฟไลน์ (Off Line)

- การใช้โหลดทางกล ค่าความผิดพลาดที่ได้จากการทดลองจะมีค่าเพิ่มมากขึ้น เมื่อเราใส่โหลดเพิ่มขึ้น

สาเหตุ : เกิดจากความร้อนจากการเสียดสีระหว่าง มู่เล่และเชือกที่ใช้คล้องเพื่อถ่วงน้ำหนัก

แนวทางการแก้ไข : ควรหาวิธีการใส่โหลดเพิ่มด้วยวิธีการอื่นๆ เช่น การใช้เบรคทางกล (mechanic break)

- การใช้โหลดทางไฟฟ้า ข้อมูลที่ได้จากการทดลองหลังจากการแปลงฟาสต์ฟูเรียร์แล้ว ตรวจสอบตำแหน่งของโรเตอร์สล็อตฮาร์โมนิกส์ได้ยาก

สาเหตุ : เกิดจากการสปาร์คของแปรงถ่าน ทำให้เกิดการเหนี่ยวนำไปรบกวน ทำให้สัญญาณที่วัดได้ตรวจสอบหาสล็อทฮาร์โมนิกส์ได้ยาก

แนวทางการแก้ไข : เลือกใช้มอเตอร์กระแสตรง ที่มีการสปาร์คของแปลงถ่านน้อย ๆ และควรใช้สายไฟที่มีการชีล (shield) เพื่อป้องกันสัญญาณรบกวนอื่นที่อาจเข้ามารบกวน

- ค่าความถี่มีความผิดเพี้ยนจากความเป็นจริง

สาเหตุ : จำนวนข้อมูลที่นำมาใช้มีค่าไม่เพียงพอ หรือบางครั้งอาจมาจากการที่มีอัตราการสุ่มสัญญาณที่ไม่เพียงพอ หรืออาจเกิดจากการแปลงสมการ Matlab ที่แกนด้านความถี่ไม่ถูกต้อง

แนวทางการแก้ไข : ทดลองหาค่าจำนวนข้อมูลและอัตราการสุ่มข้อมูลที่เหมาะสม เพื่อให้ได้ค่าความถูกต้องที่สูงที่สุดแต่ต้องไม่ใช้เวลาจนมากเกินไปด้วย

- การคับปลิ้ง (Coupling) ระหว่างมอเตอร์มีปัญญาเรื่องการต่อเชื่อมไม่ดีพอ ทำให้ระหว่างการทดลองมอเตอร์จะสั่นมาก

สาเหตุ : เนื่องจากมอเตอร์มีขนาด และความสูงไม่เท่ากัน

แนวทางการแก้ไข : ปรับปรุงการคับปลิ้ง และทำฐานรองมอเตอร์ให้ดีขึ้น

9.1.2 ปัญหาในการศึกษาพฤติกรรมแบบเรียลไทม์ (Real time)

- ไม่สามารถตรวจสอบหาสัญญาณสล็อทฮาร์โมนิกส์และวัดค่าความเร็วมอเตอร์ได้

สาเหตุ : เนื่องจากอัตราการสุ่มข้อมูลไม่เหมาะสม และจำนวนข้อมูลที่ใช้ในการประมวลผลน้อยเกินไป

แนวทางการแก้ไข : เปลี่ยนอัตราการสุ่มข้อมูล และเพิ่มจำนวนข้อมูลที่ใช้ในการประมวลผลมากขึ้น

- ไม่สามารถแปลงฟาสฟูเรียร์ กับจำนวนข้อมูลมาก ๆ ได้

สาเหตุ : เนื่องจากหน่วยความจำข้อมูลของบอร์ด TMS320C50 มีประมาณ 9k เวิร์ด ซึ่งในการแปลง FFT ต้องใช้พื้นที่ในการเก็บค่าทวิตเดิลแฟคเตอร์ (Twiddle Factor) ค่าอินพุท ค่าผลลัพท์ที่ได้ ซึ่งมีทั้งค่าจริงและค่าจินตภาพ ซึ่งทำให้ไม่สามารถแปลง FFT กับจำนวนข้อมูลมากได้

แนวทางการแก้ไข : ทำการต่อเพิ่มหน่วยความจำภายนอก เพื่อให้การประมวลผลมากขึ้น ซึ่งจะทำให้เห็นสัญญาณสล็อทฮาร์โมนิกส์ได้ชัดเจนขึ้น

- หากต้องการเพิ่มจำนวนข้อมูลที่จะแสดงผลจะทำให้ข้อมูลที่ส่งมาจะไม่เป็นเรียลไทม์

สาเหตุ : เนื่องจากการส่งข้อมูลที่เพื่อแสดงผลข้อมูลทางคอมพิวเตอร์ส่งผ่านทางซีเรียลพอร์ต ทำให้ความเร็วในการส่งไม่สูงมาก

แนวทางการแก้ไข : ส่งข้อมูลที่ประมวลผลได้ผ่านทางโปรโตคอลที่เร็วขึ้น

9.2. แนวทางการพัฒนา

1 เพิ่มจำนวนข้อมูลที่ใช้ประมวลผลสำหรับบอร์ดTMS320C50เพื่อที่จะสามารถทำการวิเคราะห์ได้และมีความถูกต้องมากขึ้นและควรปรับปรุงให้บอร์ดสามารถโหลดโปรแกรมได้ด้วยตัวเอง โดยการใช้ EPROM บันทึกโปรแกรมไว้

2 ควรปรับปรุงโปรแกรมที่ใช้ในการแสดงผลทางคอมพิวเตอร์ ให้เป็นโปรแกรมแบบ 32 บิตและรันบนวินโดวส์ได้ เพื่อเพิ่มประสิทธิภาพ และลดปัญหาในการอินเตอร์เฟส หรือแสดงผลทาง LCD Module แสดงค่าความเร็วมอเตอร์โดยไม่ต้องอาศัยคอมพิวเตอร์ เพื่อความสะดวกในการเคลื่อนย้าย

3 ใช้อุปกรณ์ที่มีความสามารถที่ดีขึ้นเช่น ใช้บอร์ดที่มีการประมวลผลได้เร็วขึ้น อัตราการสุ่มข้อมูลสูงขึ้น และมีคำสั่งให้ใช้ได้ง่ายมากขึ้น

4 หน่วยลดทอนสัญญาณควรปรับปรุงให้ใช้งานร่วมกับ CT ได้ โดย CT ที่ใช้ต้องสามารถตอบสนองความถี่สูงได้ดี

5 หาวิธีการใหม่ๆในการวิเคราะห์โรเตอร์สล็อทฮาร์โมนิกส์เช่น การทำซ้ำ(Auto Regressive)

ภาคผนวก ก
โปรแกรมภาษาแอสเซ็มบลีที่ใช้ในการ
วิเคราะห์ และคำนวณหาสัญญาณโรเตอร์สล็อทฮาร์โมนิกส์

```
;*********************************;

; RSH.ASM
;ADVISER      DR. VIJIT  KINNARES
;           Assoc.Prof.DR. VIRIYA
PICHETJUMRERN
;ADVISEE      JIRA      SAE-TEAW
;            CHALERMPOL  TONG U TAI

;            TITIPONG    SAMAKPONG
;            NARONGWIT   THAMWARAKUL

;*********************************;


;   SET     PARAMETER
ONEPASS .set    0                ;1=
no avg  0=Average on
YES     .set    1                ;
NO      .set    0                ;
;---------------------------------;
BITLEN2         .set    54h
;0078h   ;JUMP CMD
BITLEN          .set    0a8h
;079h    ;JUMP XMIT
SENDC           .set    08f0h
;================================
     .ds 0x60                    ;
AIC/FFT Setup data in upper 1/2 B2
;================================

;   Fclk = 10.368 MHz
Master clock input to the AIC
;   Fbit = Fclk/4
Serial port bit rate
;   Fs   = Fclk/(2*TA*TB)     ADC
sampling rate

TA              .word   12       ;
Fs=10.286 kHz
TB              .word   42       ;

;- - - - - - - - - - - - - - - - -
;*********************************
AIC_CMD         .word   0x0003   ;
0x72  0000 0000 0000 0011=0x0003
SIG_DEL         .word   0x0000   ;
0x73
STAT1           .word   0x0000   ;
0x74
ACCU_lo         .word   0x0000   ;
0x75
ACCU_hi         .word   0x0000   ;
0x76
REAL            .word   0x0000   ;
0x77
IMAG            .word   0x0000   ;
0x78
AUX0            .word   0x0000   ;
0x79
AUX1            .word   0x0000   ;
0x7A
FFT_S           .word   256      ;
FFT_S-1         .word   255      ;
FFT_S/2         .word   128      ;
FFT_S/2-1       .word   127      ;
scratch         .word   0
;---------------------------------
;   SET  MEMORYMAP  REGISTER
;---------------------------------
        .mmregs                  ;
        .ps     080ah            ;
        B       RINT
;RINT  Branch to receive interrupt
routine
        B       XINT
;XINT  XINT is only for timing, so
just return
                                 ;
;---------------------------------
;   START    PROGRAM
;---------------------------------
;      TMS320C50    INITIALIZATON
;*********************************

        .ps     0x0a00           ;
        .entry                   ;
start:                           ;
        SETC    INTM
        SETC    SXM
        SETC    OVM              ;
catch accumulator overflows
        LDP     #0               ;
All direct addressing is to MMRs
and B2
        SPLK    #0830h,PMST
        LACC    #0
        SAMM    CWSR             ;
Setsoftware wait stat eto 0
        SAMM    PDWSR            ;

;*********************************;
;initialize and reset serial
port
;*********************************;

        SPLK    #020h,IMR        ;
Using XINT syn TX & RX
                                 ;
initialize AIC and enable
interrupts
AIC_RS  SPLK    #01h,PRD         ;
To generate  10MHz from Tout
        SPLK    #20h,TCR         ;
for AIC maste rclock
        MAR     *,AR0
        LACC    #0008h           ;
Non continuous mode
        SACL    SPC              ;
FSX as input
        LACC    #00c8h           ;
16 bit words
        SACL    SPC
        LACC    #080h            ;
        SACH    DXR
        SACL    GREG
        LAR     AR0,#0FFFFh      ;
Reset AIC by pulsing BR\pin
```

```
        RPT     #500            ;
and taking it high after 500 cycles
        LACC    *,0,AR0         ;
(.25ms at 50ns )
        SACH    GREG
        SETC    SXM             ;
;-------------------------------
        LACC    AIC_CMD         ;
AIC control
        CALL    AIC_2nd         ;
call loader
        ;----------------------
        LACC    TB,2            ;
AIC timing B regs
        ADD     TB,9            ;
        ADD     #2              ;
        CALL    AIC_2nd         ;
call loader
        ;----------------------
        LACC    TA,2            ;
AIC timing A regs
        ADD     TA,9            ;
        CALL    AIC_2nd         ;
call loader
        ;----------------------
        LDP     #0
        LACC    #010h           ;
AIC RINT
        SACL    IMR             ;
where INT0 indicates EOC (End Of
Conv)
;*********************************
MAIN:   CALL    Init_S          ;
Load 512 alternating data points
        LAR     AR0,FFT_S       ;
Sigma FFT
        CALL    FFT             ;

      .if ONEPASS
        CALL    TO_HOST         ;
        B       MAIN
        ;----------------------
      .else
        CALL    Init_D          ;
        LAR     AR0,FFT_S       ;
Delta FFT
        CALL    FFT             ;
      .endif
;*********************************
AVG2:   LAR     AR5,#_B_base
        LAR     AR6,#_D_base
        LAR     AR7,#255
        MAR     *,AR5
AVGX:   LACC    *+,0,AR6
        ADD     *
        SFR
        SACL    *+
        MAR     *+,AR7
        BANZ    AVGX,*-,AR5
        CALL    TO_HOST         ;
        ;----------------------
        B       MAIN            ;

;*********************************
```

```
Init_S: LAR     AR6,#_D_base    ;
_D_base = A1[n],A2[n],A1[n+1],A2
[n+1],...
        LAR     AR7,#511        ;
        MAR     *,AR7           ;
enable buffer fill
        CLRC    INTM            ;
Fill FFT array with 512 samples
FILL    BANZ    FILL,*,AR7      ;
        SETC    INTM            ;
        LAR     AR5,#_B_base    ;
        LAR     AR6,#_D_base    ;
        LAR     AR7,#255        ;
        MAR     *,AR6           ;
FILLX   LACC    *+,0,AR5        ;
Load first 256 into buffer array
        SACL    *+,0,AR7        ;
        BANZ    FILLX,*-,AR6    ;
        ;----------------------
        LAR     AR7,#255        ;
        LAR     AR5,#_D_base    ;
        MAR     *,AR6           ;
FILLY   LACC    *+,16,AR5       ;
Then backfill last 256 into FFT
array
        SACH    *+              ;
REAL=X[n]
        SACL    *+,0,AR7        ;
IMAG=0
        BANZ    FILLY,*-,AR6    ;
        RET                     ;
;*********************************
Init_D: LAR     AR6,#_D_base    ;
        LAR     AR5,#_B_base    ;
Flip Flop data buffer and FFT array
        LAR     AR7,#255        ;
        MAR     *,AR5           ;
FILLD   LACL    *,AR6           ;
get x[n+256]
        LAR     AR0,*           ;
get X[n]
        SACL    *+              ;
REAL=X[n]
        SACH    *+,0,AR5        ;
IMAG=0
        SAR     AR0,*+,AR7      ;
save X[n] in data buffer
        BANZ    FILLD,*-,AR5    ;
        RET                     ;
*********************************
TO_HOST LACC    #80h
        CALL    BCXMIT          ;
Send frame synch (START)
        CALL    BRECV           ;
Wait for host start word (01Bh)
        SUB     #01Bh
        NOP                     ;
Insert NOPs to ensure condition was
ready
        NOP                     ;
while execute BCND
        BCND    TO_HOST,NEQ
        LAR     AR3,#_D_base
;AR3= FFT data array
```

```
        LAR     AR5,FFT_S/2-1
;AR5= loop counter
        MAR     *,AR3           ;
more2   LACC    *+
;scale log magnitude for host
        MAR     *+
;skipping imaginary
        RPT     #7              ;
        SFR                     ;
        CALL    XMIT            ;
        MAR     *,AR5           ;
        BANZ    more2,*-,AR3    ;
        RET
;*********************************
FFT:    MAR     *,AR0           ;
Calculate FFT parameters from AR0 =
Size
        SAR     AR0,FFT_S       ;
        MAR     *-              ;
        SAR     AR0,FFT_S-1     ;
        MAR     *+              ;
        MAR     *BR0+           ;
AR0 = AR0 >> 1;
        SAR     AR0,FFT_S/2     ;
        MAR     *-
        SAR     AR0,FFT_S/2-1   ;
        ;-----------------------
        LAR     AR0,FFT_S/2     ;
new_stg LAR     AR1,#_D_base    ;
AR1 is the TOP BFLY address
        LAR     AR2,#_D_base    ;
AR2 is the BOT BFLY address
        LAR     AR3,#_T_base+1  ;
AR3 is the TWiddle pointer
        LAR     AR4,FFT_S/2     ;
AR4 counts DFT blocks
        B       n_DFT2,*,AR1    ;
DFT:    MAR     *BR0+,AR5       ;
complete circular buffer for TW's
        LAR     AR5,#1          ;
set up DFT loop with *BR0+/BANZ
        MAR     *BR0+,AR1       ;
using 1 cuts *BR0+ loop in half!
        ;-------------------------
        ; AR1=Top AR2=Bottom
AR3=Twiddle
        ;-------------------------
BFLY:   LACC    *,14,AR2        ;
(imag1+imag2)/4
        ADD     *,14,AR1        ;
        SACH    *+,1,AR2
;store TOP imag
        SUB     *,15            ;
(imag1-imag2)/2
        SACH    *+,1,AR1
;store BOT imag
        LACC    *,14,AR2        ;
(real1+real2)/4
        ADD     *,14,AR1        ;
        SACH    *+,1,AR2
;store TOP real
        SUB     *,15            ;
(real1-real2)/2
        SACH    *,I,AR5
;store BOT real
        BANZ    OK,*BR0+,AR3    ;If
at DFT end quit early
        ;-----------------------
        MAR     *+,AR2
;clean up TW base (xxx0000+1)
        MAR     *+
;modify BOTom DATA pointer
        MAR     *0+             ;
        MAR     *0+,AR1         ;
n_DFT2: MAR     *0+
;modify the TOP pointer
        MAR     *0+,AR4         ;
        BANZ    DFT,*0-,AR3
;dec DFT block count AR4 by OFFset
        MAR     *,AR0           ;
        MAR     *BR0+           ;
        BANZ    new_stg,*       ;if
OFFset was 1, now cleared
        B       endFFT          ;
        ;-----------------------
OK      LT      *-,AR2
;TREG=TWR    *NOTE* Twiddles are
Q15
        MPY     *-
;PREG=REAL*TWR
        LTP     *+,AR3
;TREG=IMAG    ACCU=REAL*TWR
        MPY     *
;PREG=IMAG*TWI                   AR2=R
AR3=I
        LTS     *+,AR2
;TREG=TWI     ACCU=REAL*TWR-
IMAG*TWI
        MPY     *
;PREG=REAL*TWI
        SACH    *-,1,AR2    ;<<;
        LTP     *,AR3
;TREG=IMAG    ACCU=REAL*TWI
        MPY     *BR0+,AR2
;PREG=IMAG*TWR
        APAC                    ;
ACCU=IMAG*TWR+REAL*TWI
        SACH    *+,1,AR2    ;<<;
        B       BFLY,*+,AR1     ;
        ;-------------------------
endFFT: MAR     *,AR2
;Transform REAL & IMAG to log
magnitude
        LAR     AR2,#_D_base
;AR3=FFT data pointer
        LAR     AR3,FFT_S-1
;AR5=FFT loop counter
        LAR     AR0,FFT_S
        ;-------------------------;
        ; WINDOW: Performs post FFT
raised cosine windowing!        ;
        ; This is done by using the
frequency coefficients of the   ;
        ; window in a convolution
filter of the spectrum.         ;
        ;-------------------------;
```

```
       ;mar      *BR0+           ;
don't start at DC
more_MAG
        MAR      *BR0-           ;
-IMAG[-1]  1-COS(nwt/N)        + 1
        LACC     *BR0+,15        ;
IMAG[-0]  filter by post      |
        SUB      *BR0+,16        ;
+IMAG[+1]  convolution     <--+++-->
        ADD      *BR0-,15        ;
IMAG                           + + -.5
        SACH     IMAG            ;
        MAR      *+              ;
REAL
        MAR      *BR0-           ;
-REAL[-1]
        LACC     *BR0+,15        ;
REAL[-0]  X[-1] -2*X[0] + X[1]
        SUB      *BR0+,16        ;
+REAL[+1]
        ADD      *BR0-,15,AR1    ;
REAL
        SACH     REAL            ;
        SQRA     IMAG
;IMAG & REAL can be at most 0x7fff
Q15
        LTP      REAL
;MPY will result (at most) in max
positive
        MPY      REAL            ;
        APAC
;output is positive Q30
        ADD      #0x1
;Set up a floor value; log(0) not
legal!
        LAR      AR1,#17
;pre-scaling exponent shifts Y axis
        RPT      #31             ;
        NORM     *-              ;
        NOP
        NOP
        MAR      *,AR2           ;
        MAR      *BR0-           ;-
REAL;dump log(f) into oldest REAL
(odd addr)
        SACH     *,2
;clr explicit 1.0 and sign bit from
mant
        LACL     *
;load into ACCU_lo
        SAR      AR1,*
;then append exponent (AR1)
        ADD      *,16            ;
        RPT      #10
;jam result into ACCU_hi
        SFL                      ;If
needed, Use ADDH to saturate
overflow
     ;  sach     *               ;
     ;  addh     *               ;
        SACH     *               ;
        LACC     *               ;
        AND      #0xfffc,0       ;
        SACL     *BR0+           ;
REAL
        MAR      *-              ;
IMAG
        MAR      *BR0+,AR3
;+IMAG
        BANZ     more_MAG,*-,AR2
;keep going until all done
        ;-------------------------
BITREV: LAR      AR0,FFT_S
;Now perform Output bit reversal
        LAR      AR1,#_D_base    ;by
moving the magnitude, which
        LAR      AR2,#_D_base+1  ;is
in the REAL slots, into the
        LAR      AR3,FFT_S-1
;IMAG slots of the FFT data array
more_BR LACC     *+
;load the magnitude
        MAR      *+,AR1          ;
        SACL     *BR0+,0,AR3
;move it to an open IMAG slot
        BANZ     more_BR,*-,AR2
;more data to move?
        ;-----------------------
        RET                      ;
FFT is finished!!
**********************************
AIC_2nd SACH     DXR
        CLRC     INTM
        IDLE
        ADD      #6,15
;set ACCU_hi = 3 for secondary XMIT
        SACH     DXR             ;
        IDLE
;ACCU_hi requests 2nd XMIT
        SACL     DXR             ;
        IDLE
;ACCU_lo sets up registers
        SACL     DXR,2
;close command with LSB = 00
        IDLE
        RET                      ;
**********************************
; Transmitt a character
**********************************
;================================
================================
; Trasmit word routine
;================================
================================
xmtwrd  SACL     scratch
        RPT      #7
        SFR
        CALL     XMIT            ;
send highbyte
        LACC     scratch         ;
BCXMIT  BCND     $+4,bio         ;
expect a sync (0) signal from PC
        B        $-2
;--------------------------------
; xmtbyte follows now             ;
send lowbyte and return to calling
pgm
```

```
;
;  'xmtbyte' requires ARP->AR0 and
sets ACCU='mask for byte'
;  send startbit (0) +  databyte +
stopbits (2)
;---------------------------------
XMIT    CLRC    c               ;
startbit=0
        LAR     ar1,#8          ;
counter: 1 startbit+ 8 databits (+
2 stopbits)
nextbit1 BCND   snd0,nc         ;
if c=1 send 1 else send 0
snd1    SETC    xf              ;
send one
        B       snd
snd0    CLRC    xf
snd     RPT     #BITLEN         ;
send one bit
        MAR     *,ar1
        ROR                     ;
lsb(accu) -> carrybit
        BANZ    nextbit1,*-     ;
repeat for entire word (10 bits)
        SETC    xf
        RPT     #BITLEN
        NOP
        RPT     #BITLEN
        NOP
        RET
;=================================
; Read Routine
;=================================
BRECV:
wait    BCNDD   STOK,bio
;wait for start bit
        LAR     AR1,#7
        LACL    #0
        B       wait            ;
STOK    RPT     #BITLEN2
;BITLEN is scaled and
        NOP
        MAR     *,AR1
;number of bits - 1
WTBIT   SFR
        RPT     #BITLEN
;decremented by 8/3 for
        NOP
;BITLEN/2 wait
        BCND    ZEROBT,bio
        ADD     #80h
ZEROBT  BANZ    WTBIT,*-
;last bit ?
        RET
;ACC = read value

;*********************************
*********************************
; NOTE: This code is NOT written
with context restore!
;*********************************
*********************************
RINT:   SST     #1,STAT1
;Recover ARP from ARB by LST1
        MAR     *,AR7           ;
        BANZ    more_buf,*-,AR6 ;if
buffer is full RET w/o EINT
        LAR     AR7,#0          ;
        LST     #1,STAT1        ;
        RET                     ;
more_buf                        ;
        LACC    DRR             ;
        SACL    *+              ;
<<< store Ax[n], point to next
        ;-----------------------
FASTRET LST     #1,STAT1        ;
        RETE                    ;
XINT:   RETE
;=================================
        .listoff                ;
        .ds     01000h
;NOTE: Twiddles are relocated to
        .include "dsk_twid.asm" ;
0x400 (B2) using CONF 1
        .liston
        .end
```

ภาคผนวก ช
โปรแกรมภาษาซี ที่ใช้ในการ
แสดงผลทางด้านกราฟฟิกออกทางหน้าจอคอมพิวเตอร์

```
/* ---------------------------------
---------------------------------
- */
/*    File: MAIN.CPP    Example
Main using the interface.h   */
/* ---------------------------------
---------------------------------
- */

#include "interface.H"
# define THR       0    //
Transmitter holding register
# define DHR       0    // Receiver
data       register
# define BRDL      0    // Baud rate
devisor, low byte
# define BRDH      1    // Baud rate
devisor, high byte
# define LCR       3    // Line
control register
# define MSR       4    // Modem
status register
# define LSR       5    // Line
status register
# define DATA_READY    1  //
defined in the LCR register
# define DATA_OVERRUN  2  //
defined in the LCR register
# define TRUE      1
# define FALSE     0
# define XMT_BUF_EMPTY 0x20
# define DD        0    // Upload
data from DSK -> PC
# define DP        1    // Upload
prog from DSK -> PC
# define LD        2    // Download
data from PC -> DSK
# define LP        3    // Download
prog from PC -> DSK
# define XG        5    // Execute
the program

FILE  *stream;         // global for
this file

const STRING programme ="Slot
Harmonic Detector";
const STRING pgmdate   ="1/2/99";
const STRING version   ="1.00";
const STRING copyright ="Copyright
(c) Rotor Slot Harmonic Group";
extern pcom;

// global variables
PARAMETER  prm;             //
parameter file and definition -->
getchar.cpp
char       INFO[300];
char       title[200];

char buf_0[512];
// Keep past data history for
char buf_1[512];
// time averaging of signals
char buf_2[512];
char buf_3[512];
char buf_4[512];

int avg_on = 0, restart = 0;
clock_t   duration;
char dskFile[]={"spec50.dsk"};

void ExePgm(void);       // eXcecute
Go, Address, Single step
INITMONITOR InitMonitor(void);
void init_graphics(void);
void check_key(void);
void setup_vals(void);
void detectSpeed(void);

//---------------------------------
---------------------------------
// Main program
//---------------------------------
---------------------------------
void main(int argc, char *argv[])
{
  char *ptr0, *ptr1, *ptr2, *ptr3,
*ptr4;
  char *tmp0, *tmp1, *tmp2, *tmp3,
*tmp4;
  int  x, y, old_y;
  int two_block_length;
  int k,c ;

  if(argc > 3){
     printf("Please enter the
command as 'SlotHar -cx'\n");
     printf("  where x is the
comport that DSK connect to.\n");
     printf("  (The default value
is 1 for comport 1)\n");
     exit(0);
  }
  else{
     strlwr(argv[1]);
     if(argv[1]){
      if(strstr(argv[1],"-c"))
        prm.com = argv[1][2]-'1';
      else{
        printf("Please enter the
command as 'SlotHar -cx'\n");
        printf("  where x is the
comport that DSK connect to.\n");
        printf("  (The default is
1 for comport 1)\n");
        exit(0);
      }
     }
  }
  clrscr();
  sprintf(title,"\n%s - Version %s
/ %s "

      "\n%s\n",programme,version,pg
mdate,copyright);
  printf(title);
```

```
prm.EntryPoint=prm.PGM_CNT=0x0a00;
// standard entry point
  prm.INVERSE   =YES;
// for standard boards like DSK

  prm.speed=57600;
// default !
  InitializeMonitor();   //
Establish communication at 57600
baud using old kernel

//---------------------------------
-------------------------------
// Load Slot Harmonic detect DSK
file to TMS320C50
//---------------------------------
-------------------------------

  printf("\nDownloading Slot
Harmonic Detector Program...\n");
  stream=fopen(dskFile,"rb");
  duration = LoadDsk();
  ExePgm();
re_start:
//---------------------------------
-------------------------------
// Reinitialize PC serial port to
115200 baud.
//---------------------------------
-------------------------------
  prm.speed=1152001;
  InitPort();
  init_graphics();
//---------------------------------
-------------------------------
// This module test the transfering
of data without handshaking
//---------------------------------
-------------------------------
  two_block_length=128;
  tmp0 = buf_0;
  tmp1 = buf_1; tmp2 = buf_2; tmp3
= buf_3; tmp4 = buf_4;

  ptr0 = tmp0;
  for(x=0;x<256;x+=2) *ptr0++ =
128;  // init 1st line

  for(          ;          ;      ) {
      ptr0 = tmp0;
// Set buffer data pointers
      ptr1 = tmp1; ptr2 = tmp2;
ptr3 = tmp3; ptr4 = tmp4;

      detectSpeed();
      for(x=0;x<512;x+=4) {
         old_y = 128 - *ptr0;
         switch(avg_on) {
           case  1: *ptr0 =
(*ptr1+*ptr2) >> 1;   // avg of 2
                    break;
           case  2: *ptr0 =
(*ptr1+*ptr2+*ptr3+*ptr4) >> 2; //
avg of 4
                    break;
           default: *ptr0 = *ptr1;
                    break;
// avg of 1
         }
         y = 128 - *ptr0;

// ..............plot graph
.................//

         if(y > old_y){
            old_y++;
            line(x+5,old_y,x+5,y);
         }
         if(y < old_y){
            y++;
            line(x+5,old_y,x+5,y);
         }
         ptr0++;
         ptr1++; ptr2++; ptr3++;
ptr4++;                  // next data
      }
      if(kbhit()) check_key();
      if(restart){
        avg_on=restart=0;
//      closegraph();
        goto re_start;
      }
      ptr1 = tmp4;
// Rotate the buffer pointers
      tmp4 = tmp3; tmp3 = tmp2;
tmp2 = tmp1; tmp1 = ptr1;

      for(k = 10000; k > 0; k--){
         WaitFor(XMT_BUF_EMPTY);
// Send a pulse to DSP then wait
for
         outportb(pcom,NULL);
// starting pulse 0x80
         delay(1);
         c = inportb(pcom);
         if( c == 0x80 ){
// reveive starting pulse 0x80 send
            WaitFor(XMT_BUF_EMPTY);
// 0x1b to DSP indicate "ready to
            outportb(pcom,0x1B);
// receive 128 data with interrupt
            break;
// turn-off
         }
      }
      if( k == 0 ){
         printf("Fail to get start
pulse\n");
         exit(0);
      }

// Turn off interrupt and reading
the data from DSP
      asm cli;
      WaitFor(XMT_BUF_EMPTY);
```

```
      outport(pcom,NULL);

for(k=0;k<two_block_length;k++)
                        *ptr1++ =
rcvdata(pcom);
      asm sti;
  }
}
void detectSpeed(void)
{
    for(int i=1;i<10;i++){
      if(*ptr0 >*ptr0+1){

      }
    }
}

void check_key(void)
{
  int key;
  key = (bioskey(0)) >> 8;
  switch(key)
// Key traps for various routines
  {
    case 0x1E: avg_on += 1;
// (A)veraging select
               if(avg_on >2)
                avg_on = 0;
break;
    case 0x1F: while (!kbhit()){};
    case 0x13: restart = 1;
// (R)eboot code
               break;
    case 0x10: closegraph();
// (Q)uit to DOS
               exit(0);
break;
    default :                  break;
  }
  setup_vals();
// redraw the setup values
  setviewport(100,40,600,274,1);
// Set window to show display
  setwritemode(1);
// display lines are XOR drawn
  setcolor(15);
// light gray
}

void InitializeMonitor(void)
{
 UINT error;
 int pass=FALSE;

 InitPort(); //initialise serial
com regiaters.

 pass=FALSE;
 BaudRateDetect(); //send 0x80 to
DSP to calculate BITLEN.
 delay(2);  // what is the right
time ??? was 1 up to version 0.98

 switch(error=InitMonitor())
 {
       case 0:
       case 1:
       case 2:   pass=FALSE;
                 break;
       case 3:   pass=TRUE;
                 break;
 }
 if(!pass){
    printf("\nCommunication not
successful\n");
    switch(error) {
      case 0: printf("\nNo
Response.\n");
              break;
      case 1: printf("\nTest
Error.\n");
              break;
      case 2: printf("\nNo
Escape.\n");
              break;
     default: break;
    }
    exit(1);
 }
}

void InitPort(void)
{
  UINT  COMADD[]={0x3f8, 0x2f8,
0x3e8, 0x2e8};
   int   BRD=1152001/prm.speed;
   pcom = COMADD[prm.com];
   UINT port_no=pcom;

  asm  mov  DX,word ptr port_no  //
  asm  add  DX,3                 //
  asm  mov  AL,087h              //
SET BAUD access
  asm  out  DX,AL                //
  asm  sub  DX,3                 //
  asm  mov  AX,word ptr BRD  //LO//
Set 19200 baud
  asm  out  DX,AL                //
  asm  add  DX,1                 //
  asm  mov  AL,byte ptr BRD  //HI//
  asm  xchg AL,AH                //
  asm  out  DX,AL                //
  asm  sub  DX,1                 //
  asm  add  DX,3                 //
  asm  mov  AL,07h               //
CLR BAUD access...  N-8-2
  asm  out  DX,AL                //
  asm  out  DX,AL                //
N-8-2
  asm  mov  DX,port_no
//0x3FC 0x2FC  modem control
register
  asm  add  DX,5
//check LINE_STATUS  0x2FD  0x3FD
(+5)
xempty0:                         //
  asm  in   AL,DX                //
```

```
   asm  and  AL,060h
  //wait for DXR & TXR to empty
  before RST!
   asm  cmp  AL,060h                //
   asm  jne  xempty0                //
  }

  //---------------------------------
  ---------------------------------
  INITMONITOR InitMonitor(void)
  {
    int  i=10000;
    int  ans=inportb(pcom+LSR);
    int  byte_return;

    while(!(ans & DATA_READY) && i){
  // data received?
         ans = inportb(pcom+LSR);
         i--;
    }
    if(i<1)
       return  0;
    if( inportb(pcom)!=KB_ESC)
        return 2;

     delay(1);
     UINT send=0xAA;                 //
  test for reponse with random
  pattern
     WaitFor(XMT_BUF_EMPTY);         //
  wait for buff empty
     outportb(pcom,send);            //
  send a byte
     WaitFor(DATA_READY);            //
  wait for data received
     delay(1);
     if ((byte_return=inportb
  (pcom))!=send) {
         printf("\nbyte recieve =
  %x\n",byte_return);
         return 1;
     }
     WaitFor(XMT_BUF_EMPTY);         //
  wait for buff empty
     outportb(pcom,(send=KB_ESC)); //
  send a byte
     WaitFor(DATA_READY);            //
  wait for data received
     delay(1);
     if ((byte_return=inportb
  (pcom))==send)
         return 3;                   //
  all tests are successful!
     return 1;
  }

  //---------------------------------
  ---------------------------------
  void reset50()
  {
    UINT  port_no=pcom;
    if(prm.INVERSE) {
  // do the inverse DTR for reset of
  c50
         asm  mov  DX,word ptr
   port_no//modem control register
   02FC 03FC
         asm  add  DX,4
   //
         asm  mov  AL,0xB
   //RTS=1, DTR=0
         asm  out  DX,AL
         delay(1);
         asm  mov  DX,word ptr
   port_no//modem control register
         asm  add  DX,4
   //
         asm  mov  AL,0xA
   //RTS=1, DTR=0
         asm  out  DX,AL
         delay(1);
         asm  mov  DX,word ptr
   port_no//modem control register
         asm  add  DX,4
   //
         asm  mov  AL,0xB
   //RTS=1, DTR=0
         asm  out  DX,AL
         delay(1);
     }
     else  {                              //
   else we have a hardware negate
   function from PC to c50
         asm  mov  DX,word ptr
   port_no//modem control register
         asm  add  DX,4
   //
         asm  mov  AL,0xA
   //RTS=1, DTR=0
         asm  out  DX,AL
         delay(1);
         asm  mov  DX,word ptr
   port_no//modem control register
         asm  add  DX,4
   //
         asm  mov  AL,0xB
   //RTS=1, DTR=0
         asm  out  DX,AL
         delay(1);
         asm  mov  DX,word ptr
   port_no//modem control register
         asm  add  DX,4
   //
         asm  mov  AL,0xA
   //RTS=1, DTR=0
         asm  out  DX,AL
         delay(1);
     }
  }

  //---------------------------------
  ---------------------------------
  void BaudRateDetect()
  {
   reset50();
   delay(12);          // v.02 ROM code
  need delay for boot loader
```

```
 if(prm.speed<57600){
    while(!(inportb(pcom+LSR) &
XMT_BUF_EMPTY));
    outportb(pcom,0x80);    //
write a byte to the com port
(320c50)
 }
 else
    outportb(pcom,0x80);    //
write a byte to the com port
(320c50)
}
void WaitFor(UINT what)
{
 int j=10000,i=j;

 while(!(inportb(pcom+LSR) & what)
&& i--);

 if(i<1){
    fprintf(stdout,"Handshake
error"
          "\r\nWait loop finished
in WaitFor() with start=%d",j);
    exit(1);
 }
 else return;
}

void sendbyte(UINT send)
{
  WaitFor(XMT_BUF_EMPTY);        //
wait for buff empty

  outportb(pcom,send);           //
send a byte

  WaitFor(DATA_READY);           //
wait for data received
  delay(1);
  UINT receive=inportb(pcom);
  if (receive!=send)
     fprintf(stdout,"Handshake
error"
              "\r\nSent byte in
'sendbyte()' is not correct!\r\n"
              "Sent    : 0x%02x
Received:
0x%02x\r\n",send,receive);
}

UINT sendword(UINT send)
{
 UINT   s1;

 WaitFor(XMT_BUF_EMPTY);
// wait for empty
 s1 = (send>>8)&0xff;
 outportb(pcom,s1);   // send high
byte of end address

 WaitFor(DATA_READY);
// wait for data rec
 delay(1);
 UINT receive=inportb(pcom);

 WaitFor(XMT_BUF_EMPTY);
 s1 = send&0xff;
 outportb(pcom,s1);        // send
low  byte of end address

 WaitFor(DATA_READY);
// wait for data rec
 delay(1);
 if((receive=(receive<<8)+inportb
(pcom))!=send)
    fprintf(stdout,"Handshake
error"
              "\r\nSent word in
sendword() is not correct!\r\n"
              "Sent    : %04xh
Received: %04xh\r\n",send,receive);

 return receive; // return sent
word for check if necessary
 }

//---------------------------------
---------------------------------
//--- Load a program/data file in
TI-Dsk Format: default ext. dsk   -
-
//---------------------------------
---------------------------------
double LoadDsk(void)
{
 char    linebuf[MAXLINE],
*pbuf=linebuf;
 UINT    line=0;        // line
counter of file
 UINT    data[MAXLINE]; // max 100.
words to download
 UINT    numdata=0;     // num of
words to download
 ULONG   sumdata=0;     // sum of
all downloaded words
 UINT    address;       // address
for     download
 BOOLE   program;       // type: if
program 'yes' , no if data 'no'
 double  d;
 clock_t start=clock();
 while(1) {
// for all lines

pbuf=fgets(linebuf,MAXLINE,stream);
     switch(*pbuf){
// analyse line
       case  ':':
       case  'K': numdata = 0;
break;          // Do something with
the name ?
       case  '9': address=GetDskAdd
(pbuf+1); // DSK for an address
                  switch(*(pbuf+5)){
                     case '7': break;
// Do nothing with checksum   ?
```

```
                    case 'M':
program=NO; break;
                    case 'B':
program=YES; break;
                    default :
fprintf(stdout,"\n\nerror 1: 'Wrong
Tag "

      "in Pos 6'!\n\n\rPlease check
whether"
                              " the
file is a valid *.dsk
file!\n\n\r");
                         return 0;
                  }
                  numdata=GetDskData
(pbuf+5,data);
                  sumdata +=numdata;
                  break;
      case  '1': address=GetDskAdd
(pbuf+1);          // Entry address
                  if(address==0)
                    break; // dska
generates address=0 if no entry
defined!


prm.PGM_CNT=prm.EntryPoint=address;
                  break;
      case NULL: fclose(stream);
                  fprintf
(stdout,"\nLoading of prog/data
finished");
                 d=clock()-start;
                 return d>0 ? d :
0.001;

      default  : fprintf
(stdout,"\n\nerror 2: 'Wrong Tag in
Pos 1' in LoadDsk()"

      "!\n\n\rPlease check whether"
                         " the
file is a valid *.dsk
file!\n\n\r");
                  return 0;
    }

    if(numdata){
// download line
      UINT i,cmd=(program==YES) ?
LP:LD;

      fprintf(stdout,"\r   %4d %5d
%8ld  %s",

++line,numdata,sumdata,program ?
"program":"data");
      sendcommand(cmd,address,numda
ta);
      for(i=0;i<numdata;i++)
sendword(data[i]);
    }
 }

}

//---------------------------------
---------------------------------
UINT GetDskAdd(STRING pbuf)
{
 UINT x=0,i=4;
 char c=*pbuf++;
 for(;i;i--,c=*pbuf++)
   {
    x =(x<<4) + c;
    if(isdigit(c))
      x -= '0';
    else
      x -= 'A'-10;
   }
 return x;
}
//---------------------------------
---------------------------------
UINT GetDskData(STRING pbuf,UINT
*data)
{
 UINT num=0,i;
 UCHAR c;

 while(*(pbuf++)!='7')
   {
    for(*data=0,i=4;i;i--)
      {
       c = *pbuf++;
       *data<<=4;
       if(isdigit(c))
        *data +=c-'0';
       else
        *data +=c-'A'+10;
      }
    data++;
    num++;
   }

 return num;
}
void ExePgm(void)
{
 sendbyte(XG); sendword
(prm.PGM_CNT);    // execute
 outportb(pcom,NULL);
// Startpulse / sync for
windows/OS2 ??????
 return;
}
void setup_vals(void)
{
  setwritemode(0);
// turn off XOR write
  setviewport(0,300,200,309,1);
// clear window
  clearviewport();
//
  setviewport(100,0,620,30,1);
// Write the title in yellow
  setcolor(14);
```

```
  settextstyle(0,0,2);
  outtextxy(60,10,"Slot Harmonic
Detector");
  setviewport(0,30,620,330,1);
// Write to window in bright blue
  setcolor(11);
//
  settextstyle(0,0,0);
  outtextxy(10,33, "     +10dB");
  outtextxy(10,59, "      +0dB");
  outtextxy(10,72, "  M       ");
  outtextxy(10,85, "  A  -10dB");
  outtextxy(10,98, "  G       ");
  outtextxy(10,111,"  N  -20dB");
  outtextxy(10,124,"  I       ");
  outtextxy(10,137,"  T  -30dB");
  outtextxy(10,150,"  U       ");
  outtextxy(10,163,"  D  -40dB");
  outtextxy(10,176,"  E       ");
  outtextxy(10,189,"     -50dB");
  outtextxy(10,215,"     -60dB");
  outtextxy(10,241,"     -70dB");
  outtextxy(450,270,"SlotHar.EXE
");   // Name of program
  outtextxy(450,280,"FFT: 256
Radix-2");   //
  outtextxy(450,290,"10 dB/div:
");   //
  outtextxy(96,250,"DC");
  outtextxy(185,250," 950");
  outtextxy(285,250,"1900");
  outtextxy(385,250,"2850");
  outtextxy(485,250,"3800");
  outtextxy(585,250,"4750");
  outtextxy(224,260,"
Frequency (Hz)          ");
  outtextxy(100,290,"(Q)uit to DOS
");   //
  outtextxy(100,280,"(S)top Graph
");   //
  switch(avg_on)
  {
    case  1: outtextxy(100,270,"(A)
verage 2");break; //
    case  2: outtextxy(100,270,"(A)
verage 4");break; //
    default: outtextxy(100,270,"(A)
verage 1");break; //
  }
}
void init_graphics(void)
{
  int gdriver = EGA, gmode = EGAHI,
errorcode, Y, X;
//  registerbgidriver
(EGAVGA_driver);     // Use with
BCC for no EGAVGA.BGI
  initgraph(&gdriver, &gmode, "");
// if possible open EGA mode
  errorcode = graphresult();
  if (errorcode != grOk)
  {
    printf("Graphics error: %s\n",
grapherrormsg(errorcode));
    printf("Press any key to
halt:");
    getch();
    exit(1);
  }
  clearviewport();
//
  setup_vals();
// dispaly the setup
  setviewport(100,40,600,274,1);
// Set window to show display
  setcolor(2);
// reticle is green
  for(Y=0;Y<=234;Y+=26) line
(0,Y,500,Y);  // draw reticle
  for(X=0;X<=500;X+=50) line
(X,0,X,234);  //
  setwritemode(1);
// display lines are XOR drawn
  setcolor(15);
// light gray
}
```

```
/* ---------------------------------
---------------------------------
- */
/* File: interface.CPP -> Host
Interface Library Source Code   Ver
1.00   */
/* ---------------------------------
---------------------------------
- */

#include "interface.H"

static FILE    *stream;
extern char appfile[];          /*
app filename with .DSK extension
*/

extern int  MSGRx=2,MSGRy=21;   /*
x&y coordinates of messenger window
*/
extern UINT pcom=0;             /*
commport address
*/
extern PARAMETER prm;           /*
you can use these in main
*/
extern STRUCT_IMR imr;          /*
*/
```

```
/* ---------------------------------
---------------------------------
- */

unsigned char rcvdata(int port_no)
{
  unsigned char c;
  asm  mov  DX,word ptr port_no //
RECV a character
  asm  add  DX,5                //
check LINE_STATUS  0x2FD  0x3FD
(+5)
  asm  mov  BX,4000             //
chkst:                          //
  asm  sub  bx,1                //
  asm  jz   timerr              //
  asm  in   AL,DX               //
  asm  and  AL,061h             //
  asm  cmp  AL,061h             //
  asm  jne  chkst               //
wait for DRR & RSR to fill
  asm  sub  DX,5                //
  asm  in   AL,DX               //
  asm  mov  byte ptr c,AL       //
  return(c);                    //
timerr:                         //
    return(-1);                 //
}
```

```
/* ---------------------------------
---------------------------------
-- */
/*  File 'interface.H'
/* ---------------------------------
---------------------------------
-- */
typedef unsigned int   UINT;
typedef unsigned char  UCHAR;
typedef          char* STRING;
typedef unsigned long  ULONG;
# include <alloc.h>
# include <conio.h>
# include <ctype.h>
# include <direct.h>
# include <dos.h>
# include <fcntl.h>
# include <io.h>
# include <malloc.h>
# include <math.h>
# include <graphics.h>
# include <setjmp.h>
# include <stdio.h>
# include <stdarg.h>
# include <stdlib.h>
# include <string.h>
# include <float.h>
# include <time.h>
# include <bios.h>

# define KB_ESC  0x01b

/*---UART definitions: register
etc.   ----------------------------
---  */

# define THR      0    /*
Transmitter holding register
*/
# define DHR      0    /* Receiver
data       register
*/
# define BRDL     0    /* Baud rate
devisor, low byte
*/
# define BRDH     1    /* Baud rate
devisor, high byte
*/
# define LCR      3    /* Line
control register
*/
# define MSR      4    /* Modem
status register
*/
# define LSR      5    /* Line
status register
*/
# define DATA_READY    1  /*
defined in the LCR register
*/
# define XMT_BUF_EMPTY 0x20

const int MAXLEN = 0x100,
          MAXLINE = 100;

                              /*
Addresses of 'C50 registers
*/
const int ACCU =0x60,      /*
Address of the Accumulator
*/
          ACCB =0x62,      /*
Address of Accumulator B
*/
          PREG =0x64,      /*
Address of Product Register
*/
          ST0 =0x66,       /*
Address of ST0
*/
          ST1 =0x67,       /*
Address of ST1
*/
          TREG =0x68,      /*
Address of the T register
*/
          STACK =0x69,     /*
Address of the hardware stack
*/
          ARX =0x71,       /*
Address of the Auxillary ergisters
*/
          TIMER =0x74;     /*
address of Timer register
*/

/* ----------enumarations --------
---------------------------------
*/

enum BOOLE {NO, YES};

enum c50COMMANDS { /* commands for
comm between PC and monitor kernel
*/
                   DD=0,DP,LD,LP,/*
these 4 are also defined in c50
*/
                   XCH_PGM,XG};  /*
these 2 are also defined in c50
*/
                              /*
XCH_PGM is supported on debugger
only */
```

```
                    /* These are used
for status reporting
*/
enum INITMONITOR
{GOOD,TESTERROR,NOESCAPE,NORESPONSE
};


/* -------------structures --------
---------------------------------
*/
struct STRUCT_IMR {
              UINT add, user, debug;
              BOOLE modified;
};

struct PARAMETER {
      UINT  PGM_CNT;        /*
Value to init the program counter
*/
          UINT  EntryPoint;
      /*    the next 3 parameter
define the comport       */
      BOOLE INVERSE;      /*
Inversion of the DTR line No/Yes
*/
      UINT  com;          /*
Communication Port number (1 or 2)
*/
      ULONG speed;        /*
Baudrate up to 115K baud
*/
};

/* ---------------------------------
---------------------------------
- */

extern struct STRUCT_IMR imr;
extern struct PARAMETER  prm;
/* ---------------------------------
---------------------------------
- */
void         BaudRateDetect(void);
UINT          getwordcom(void);
void          GetArguments(void);
UINT         GetDskAdd(STRING pbuf);
UINT         GetDskData(STRING
pbuf,UINT data[]);
BOOLE         FreeRun(UINT address);
void         InitPort(void);
void         InitializeMonitor
(void);
INITMONITOR InitMonitor(void);
void         InitRegister(BOOLE
all);
double       LoadDsk(void);
void         reset50(void);
void         sendbyte(UINT);
UINT         sendword(UINT);
void         WaitFor(UINT);
UCHAR         rcvdata(int);
void         MSGR(char msg_string
[50]);


/* ---------------------------------
---------------------------------
*/
inline void sendcommand(const
c50COMMANDS cmd,UINT
startaddress,UINT length)
{
 sendbyte(cmd); sendword
(startaddress); sendword(length-1);
}
/* ---------------------------------
---------------------------------
*/
inline void SendDataWord(UINT
address,UINT word)
{
 sendcommand(LD,address,1);
 sendword(word);
}
/* ---------------------------------
---------------------------------
*/
inline UINT GetPgmWord(UINT
address)
{
 sendcommand((c50COMMANDS) DP,
address, 1);
 return getwordcom();
}
/* ---------------------------------
---------------------------------
*/
```

กิตติกรรมประกาศ

ทางคณะผู้จัดทำขอขอบคุณ อาจารย์วิจิตร กิณเรศ และอาจารย์วิริยะ พิเชฐจำเริญ ซึ่งเป็นอาจารย์ที่ปรึกษา ที่ได้ให้คำปรึกษา คำแนะนำ และคอยช่วยเหลืออยู่ตลอดเวลา

ขอขอบคุณ เจ้าหน้าที่ ศูนย์วิจัยพัฒนาและบริการทางด้านวิศวกรรมชั้น 2 ตึก 12 ที่ทุกๆ คนคอยให้ความช่วยเหลือในการใช้บริการจากศูนย์ในการยืมอุปกรณ์การทำโครงงาน รวมทั้งห้อง เพื่อทำการทดสอบด้วย

ขอขอบคุณเพื่อนๆ ที่คอยช่วยเหลือในการขนอุปกรณ์เช่น คอมพิวเตอร์ แผงโหลด หลอดไฟ จากหอ หรือไม่ก็ จากภาควิชาไปยังตึก 12

ขอขอบคุณภาควิชาวิศวกรรมเครื่องกลในการที่ให้ยืมสถานที่ในการต่อแท่นมอเตอร์และติดมู่เล่

ขอขอบคุณอาจารย์ดุสิต ที่ได้ไว้ใจและให้ยืม Wound Rotor และ DC Motor เพื่อใช้ในการทดลอง

เอกสารอ้างอิง

[1] M.Ishida and K. Iwata, “A new slip frequency detector of an induction motor utilizing rotor slot harmonics,” IEEE trams. Ind. Applicat.,vol IA-20, pp, 575-582, May/June 1984.

[2] B. Williams, J. Foodfellow, and T. C. Grean, “Sensorless speed measurement of inverter driver squirrel cage induction motors,” inPorc. IEE 4ht Int Conf, Power Electronics and Variable Speed Drives,1990, pp 279-300.

[3] D.Zinger, F. Profumo, T. Lipo, and D, Novotny, :A direct field oriented controller for induction motor drives using tapped stator windings,” in Proc. IEEE Power Electronics Specialists Comf., 1988, pp, 855-861,

[4] A. Ferrah, K. J. Bradley ,and G. M. Asher, “An FFT-base novel approach to noninvasive speed measurement in induction motor drive,” IEEE Trans. Instrum. Meas., vol. 41, pp. 797-802, Dec. 1992.

[5] R, Blasco-Gaminez, “High performance sensorless vector control of induction motor drives,”Ph.D. disertation, Dep. Elext. Electron. End., Univ. Nottingham, U.K., 1996.

[6] Peter Vas , "Parameter Estimation , Condition Monitoring , and diagnosis electrical machine , Oxford : Clarendon Press , 1993

[7] สุธรรม ศรีเกษม , เมธินทร์ ทรงชัยสกุล , สง่า ศรีศุภปรีดา , " Matlab เพื่อการแก้ปัญหาทางวิศวกรรม " , สำนักพิมพ์มหาวิทยาลัยรังสิต , 2540

[8] วัลลภ สุรกำพล , ศ.ดร. , " การประมวลผลสัญญาณเชิงเลข " , กรุงเทพฯ , บริษัทกรุงเทพไดนาปริ้นท์ จำกัด , 2533

[9] วิรธร รักวนิชพงษ์ , วิโรจน์ ฉ่ำบุญรอด , วีรพันธ์ กุณฑโชติ , "เครื่องวัดสัญญาณฮาร์โมนิกส์โดยใช้ TMS320C50 DSK " , " ปริญญานิพนธ์ " , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2539

[10] สุรศักดิ์ รอดแก้ว , อุดม วงศ์ศิรินพคุณ , " อุปกรณ์วัดสัญญาณฮาร์มอนิกส์ " , "ปริญญานิพนธ์ , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

[11] Texas Instrument , "TMS320C5x DSP Starter Kit User 's Guide", Texas Instrument Incorporate , 1994

[12] Texas Instrument,"TMS320C5x User ' s Guide ", Texas Instrument Incorporate , 1994