เครื่องปรับปรุงเพาเวอร์แฟคเตอร์ชนิดแอคทีฟ

Active Power Filter

โดย

นายนฤพนธ์ งามเงิน รหัส 43015308

นายวิธวัช สังข์เผือก รหัส 43015327

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

ภาควิชาระบบควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องปรับปรุงเพาเวอร์แฟคเตอร์ชนิดแอกทีฟ

ACTIVE POWER FILTER

ผู้จัดทำ

1. นายนฤพนธ์ งามเงิน รหัสประจำตัว 43015308
2. นายวิธวัช สังข์เผือก รหัสประจำตัว 43015327

อาจารย์ที่ปรึกษา

(ดร.นนทวัฒน์ จุลเดชะ)

หัวข้อปริญญานิพนธ์	เครื่องปรับปรุงเพาเวอร์แฟคเตอร์ชนิดแอคทีฟ	
นักศึกษา	วิธวัช สังข์เผือก	43015327
	นฤพนธ์ งามเงิน	43015308
อาจารย์ที่ปรึกษา	ดร. นนทวัฒน์ จุลเดชะ	
ระดับการศึกษา	วิศวกรรมศาสตรบัณฑิตภาควิชาระบบควบคุม	
ปีการศึกษา	2545	

บทคัดย่อ

ในปัจจุบันอุปกรณ์แอคทีฟเพาเวอร์ฟิลเตอร์ (Active power filter) นั้นยังไม่สามารถผลิตให้มีคุณภาพดีได้ในประเทศไทย ซึ่งแอคทีฟเพาเวอร์ฟิลเตอร์นั้นถือว่าเป็นอุปกรณ์ที่จำเป็น ที่ใช้ในการกำจัดฮาร์โมนิกส์และแก้เพาเวอร์แฟคเตอร์ในโรงงานอุตสาหกรรม

สำหรับปริญญานิพนธ์ฉบับนี้จะใช้ บอร์ด TMS320C31 ในการประมวลผลสัญญาณ ซึ่งใช้หลักการของดิจิตอลซิกแนลโปรเซสเซอร์ (Digital signal Processor) โดยการวิเคราะห์สัญญาณจะใช้วิธีฟาสต์ฟูเรียร์ทรานส์ฟอร์ม (Fast foucier transform) วิเคราะห์สัญญาณ สเปคตรัม(spectrum) ออกมาเพื่อหาขนาด (Magnitude) และ เฟส(phase) ของฮาร์โมนิกส์ต่างๆที่เกิดขึ้นมา

Thesis Title	: ACTIVE POWER FILTER	
Name	: Mr. Wittawat Sangpoak	43015327
	: Mr. Narupon Ngam-Ngern	43015308
Thesis Advisor	: Dr. Nonthawat Chunladacha	
Level of Study	: Engineer of Control System	
Academic Year	: 2002	

Abstract

In present Active power filter can't to good quality manufacture in Thailand. The Active power filter is an important eqiupment. It's use in to eliminate harmonics and to reduce power factor in factory industrial.

This project use card TMS320C31 for signal processing from a principle of Digital signal processing and a signal analysis use Fast fourier Transform method. Find spectrum for to seek magnitude and phase of varied harmonics.

กิตติกรรมประกาศ

โครงงานและปริญญานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงไปได้ด้วยดี เนื่องจากได้รับความช่วยเหลือและความร่วมมือจากหลายๆฝ่ายทั้งทางด้านวิชาการและกำลังใจ ทางผู้จัดทำขอขอบพระคุณมา ณ ที่นี้

ขอขอบพระคุณ อาจารย์ นนทวัฒน์ จุลเดชะ อาจารย์ที่ปรึกษาที่ได้ให้ความกรุณา และให้คำปรึกษาด้านความรู้ และจัดทำโครงงาน รวมทั้งการดูแลเอาใจใส่ และความห่วงใยในทุกๆเรื่อง

ขอขอบพระคุณ อาจารย์ ถาวร เบญจนราสุทธิ์ ที่ให้การดูแลและสอบถามข้อมูลอย่างสม่ำเสมอ

ขอขอบพระคุณ เพื่อนๆที่เป็นกำลังใจ และยังเอื้อเฟื้อสถานที่ในการทำเอกสารและทดลองอุปกรณ์

ที่สำคัญที่สุดขอขอบพระคุณ คุณพ่อ คุณแม่และพี่ๆทุกคนที่เป็นที่รักยิ่งที่ให้โอกาสในการสนับสนุนด้านการศึกษา และเป็นกำลังใจมาโดยตลอด

สารบัญ

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

ปัจจุบันเทคโนโลยี มีการพัฒนาอย่างรวดเร็วมาก การนำเทคโนโลยีทางอิเล็คทรอนิกส์มาใช้ในงานอุตสาหกรรมนั้นมีจำนวนมากขึ้นด้วย การนำอุปกรณ์อิเล็คทรอนิกส์กำลังไปใช้งานควบคุมเครื่องจักรกลเป็นที่นิยมอย่างกว้างขวาง เนื่องจากมีประสิทธิภาพค่อนข้างดี แต่ก็มิได้มีแต่ประโยชน์เท่านั้น เนื่องจากอุปกรณ์เหล่านี้ไม่มีความเป็นเชิงเส้นของกระแสและแรงดัน เพราะความสัมพันธ์ระหว่างค่าชั่วขณะของกระแสและแรงดันไม่คงที่ โหลดที่ไม่มีความเป็นเชิงเส้นและมีการเปลี่ยนแปลงของกระแสสูง การเปลี่ยนแปลงของกระแสอย่างฉับพลันเป็นผลทำให้เกิดฮาร์โมนิกส์หรือสัญญาณที่มีความถี่เป็นจำนวนเท่าของความถี่มูลฐานในระบบไฟฟ้า และความไม่เป็นเชิงเส้นจะทำให้กระแสเกิดการผิดเพี้ยนไป ดังนั้นโหลดที่ไม่เป็นเชิงเส้นจึงเป็นแหล่งกำเนิดฮาร์โมนิกส์ (Harmonics Source) ให้แก่ระบบ

อุปกรณ์ที่ไม่เป็นเชิงเส้นสามารถแบ่งได้เป็นสามประเภทดังนี้

1. อุปกรณ์ประเภทอิเล็กทรอนิกส์กำลัง (Power Electronics Device) เช่น เครื่องแปรผัน (Converter) สามารถใช้งานได้ เช่น ตัวเรียงกระแส (Rectifier) คอนเวอร์เตอร์กำลังผลิต (Static Power Converter) เครื่องควบคุมกำลังกระแสสลับใช้ไทริสเตอร์ (Thyristor A.C. power controlled) และ อินเวอร์เตอร์ (Invertor)

2. อุปกรณ์ประเภทแม่เหล็ก (Magnetizing device) อุปกรณ์จำพวกนี้ ได้แก่ หม้อแปลงมอเตอร์ เมื่อหม้อแปลงได้รับแรงดันมากเกินพิกัดทำให้กระแสสร้างสนามแม่เหล็ก (Magnetizing current) เพิ่มขึ้น ทำให้เกิดการอิ่มตัวของแกนเหล็ก

3. อุปกรณ์ประเภทอาร์ค (Arc Device) เช่น เตาหลอม (Arc Furnaces) จะกำเนิดฮาร์โมนิกส์จำนวนมากเพราะสาเหตุจากไม่เป็นเชิงเส้น

ในระบบที่มีการรบกวนของฮาร์โมนิกส์สูงจะทำให้เกิดความร้อนสูงในตัวหม้อแปลงและสายนิวทรัลในระบบสามเฟสสี่สายได้ ซึ่งกรณีทีมีการกระชากของกระแสสูงๆอาจทำให้เซอร์กิตเบรกเกอร์ตัดวงจรได้

แนวทางในการพิจารณาว่าเมื่อใดควรศึกษาถึงปัญหาฮาร์โมนิกส์

1.เมื่อระบบไฟฟ้าใช้คอนเวอร์เตอร์หรืออุปกรณ์ไฟฟ้าที่ผลิตฮาร์โมนิกส์ออกมาตั้งแต่ 20% ขึ้นไปของขนาดหม้อแปลง

2.เมื่ออุปกรณ์ป้องกันคาปาซิเตอร์ และ ตัวคาปาซิเตอร์ เสียหายบ่อย

3.ในการออกแบบระบบไฟฟ้าของโรงงานซึ่งมีโหลดประเภทกำเนิดฮาร์โมนิกส์และเพื่อการปรับปรุงค่าตัวประกอบกำลัง

4.ถ้าทางการไฟฟ้ามีกฎข้อบังคับจำกัดให้ระบบไฟฟ้าของโรงงานส่งฮาร์โมนิกส์เข้าระบบไฟฟ้าของการไฟฟ้าได้เกินค่าหนึ่ง

5.การขยายระบบไฟฟ้าของโรงงานซึ่งจะเพิ่งโหลดประเภทที่สามารถผลิตฮาร์โมนิกส์ได้เมื่อรู้แน่ว่าปัญหาที่เกิดขึ้นกับระบบไฟฟ้าเป็นผลมาจากฮาร์โมนิก เราสามารถหาระดับความถี่รีโซแนนซ์ได้

การแก้ปัญหาของการเกิดฮาร์โมนิกส์ในระบบไฟฟ้ากำลังที่สำคัญๆมีอยู่ 2 วิธีคือ พาสซีฟแอลซีฟิลเตอร์ (Passive LC Filter) และ แอกทีฟฟิลเตอร์ (Active filter) ซึ่งวงจรแอกทีฟฟิลเตอร์จัดเป็นวิธีที่ได้ประสิทธิภาพสูงสุด ถ้าหากได้รับการออกแบบและติดตั้งอย่างถูกวิธี สาเหตุดังนี้

1.การติดตั้งแอกทีฟฟิลเตอร์นอกจากจะกำจัดกระแสฮาร์โมนิกส์ได้สูงถึง80-90%แล้วยังสามารถปรับปรุงค่าเพาเวอร์แฟคเตอร์ให้อยู่ในเกณฑ์ที่ดีได้ในเวลาเดียวกันด้วย

2.สามารถติดตั้งแอกทีฟฟิลเตอร์ที่จุดเมนของระบบเพียงจุดเดียวทำให้การควบคุมเป็นไปอย่างอัตโนมัติ การดูแลรักษาทำได้ง่าย แทนที่จะติดตั้งตามจุดย่อยต่างๆในแต่ละอุปกรณ์ ซึ่งมักมีหลายจุดที่เป็นแหล่งกำเนิดฮาร์โมนิกส์

3.การใช้แอกทีฟฟิลเตอร์จะไม่มีข้อจำกัดกระแสโหลดหรือกระแสฮาร์โมนิกส์ว่ามากหรือน้อยเพียงใดกล่าวคือถ้ากระแสฮาร์โมนิกส์มากหลายร้อยหลายพันแอมป์ก็จะสามารถออกแบบแอกทีฟฟิลเตอร์ที่สามารถรับกระแสดังกล่าวได้ ด้วยเหตุนี้จึงสามารถรักษาคุณภาพของระบบไฟฟ้าที่ระดับแรงดันค่อนข้างคงที่มาก เมื่อเทียบกับการแก้ไขปัญหาฮาร์โมนิกส์ด้วยวิธีอื่นๆ

4.สามารถออกแบบติดตั้งได้ทั้งทางด้านแรงต่ำและด้านแรงสูง อีกทั้งมีข้อจำกัดน้อยมากกับโหลดในระบบ

วงจรแอกทีฟฟิลเตอร์ที่นำมาปรับปรุงค่าเพาเวอร์แฟคเตอร์ สามารถแบ่งออกเป็น 2 ประเภทใหญ่ๆได้ดังนี้

1.วงจรแอกทีฟฟิลเตอร์แบบฉีดกระแสฮาร์โมนิกส์ (Harmonic Current Injection)

2.วงจรแอกทีฟฟิลเตอร์แบบปรีเรกกูลเตอร์ (Preregulator)

บทที่ 2
ทฤษฎีและหลักการ

ในระบบไฟฟ้ากำลัง แรงดันของระบบ(Line Voltage) และกระแสของระบบ (Line Current) โดยปกติจะมีรูปคลื่นไซน์ (Sine wave) แต่เนื่องจากลักษณะของโหลดที่ต่ออยู่กับแหล่งจ่ายจะมีผลทำให้รูปคลื่นกระแสของระบบเปลี่ยนแปลงหรือผิดเพี้ยนไปจากเดิม ลักษณะของรูปคลื่นที่ผิดเพี้ยนไปจากเดิม ลักษณะของรูปคลื่นที่ผิดเพี้ยนก็จะแตกต่างกันออกไปตามลักษณะของโหลด ซึ่งรูปคลื่นเหล่านี้เกิดจากกระแสฮาร์โมนิกส์ (Harmonic Current)จึงต้องมีการชดเชยของกระแสโดยวงจรแอกทีฟเพาเวอร์ฟิลเตอร์ ซึ่งกระแสของระบบสามารถแสดงได้จากสมการดังนี้

$$i_l(t) = i_o(t) + i_p(t) + i_q(t) + i_h(t) \qquad (2.1)$$

$i_l(t)$ คือ กระแสโหลด (Load Current)

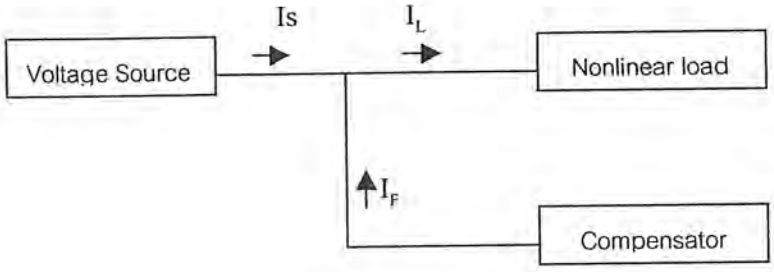
$i_o(t)$ คือ องค์ประกอบกระแสตรง (DC Component)

$i_p(t)$คือกระแสที่อินเฟสกับแรงดันของแหล่งจ่ายหรือกรแสแอกทีฟ (Active Current)

$i_q(t)$ คือ กระแสรีแอกทีฟ (Reactive Current)

$i_h(t)$ คือ กระแสฮาร์โมนิกส์ (Harmonic Current)

ในระบบทั่วไปในส่วนประกอบของกระแสโหลดจะมีเพียงส่วนประกอบเพียงอย่างเดียวที่แหล่งจ่ายควรจะจ่ายให้กับภาระคือ กระแสอินเฟสหรือกระแสแอกทีฟ ซึ่งนั่นหมายความว่า แหล่งจ่ายต้องการจะจ่ายแค่เพียงกระแสแอกทีฟให้แก่ภาระเท่านั้นกระแสที่นอกเหนือจากกระแสแอกทีฟคือส่วนประกอบของกระแสตรง กระแสรีแอกทีฟและกระแสฮาร์โมนิกส์ต่างๆจะถูกชดเชยออกนั่นคือวงจรชดเชยกระแสหรือวจรแอกทีฟฟิลเตอร์นั่นเอง



รูปที่ 2.1 ไดอะแกรมของวงจรแอกทีฟฟิลเตอร์

ความรู้เบื้องต้นเกี่ยวกับฮาร์โมนิกส์

1.ฮาร์โมนิกส์ (Harmonic)

ฮาร์โมนิกส์ คือ สัญญาณที่มีความถี่เป็นจำนวนเท่าของความถี่หลักมูล เช่น ความถี่มูลฐานคือ 50 Hz ฮาร์โมนิกส์คือ 100 , 200 Hz เป็นต้น ความถี่มูลฐานเรียกว่า ฮาร์โมนิกส์ที่ 1 ส่วนฮาร์โมนิกส์อื่นๆก็หาได้จากการเอาเลขจำนวนเต็มคูณกับความถี่มูลฐาน การวัดความเพี้ยนฮาร์โมนิกส์สามารถทำได้หลายวิธีที่แตกต่างกันออกไปแต่โดยส่วนมากใช้วัดความเพี้ยนฮาร์โมนิกส์รวม(Total Harmonic Distortion : THD) ที่ความถี่ฮาร์โมนิกส์ต่างๆกัน ปริมาณของขนาดจะแสดงด้วยจำนวนเปอร์เซนต์ของขนาดที่ความถี่หลักมูลและเครื่องมือวัดฮาร์โมนิกส์ทั้งหลาย จะแสดงผลของความเพี้ยนฮาร์โมนิกส์รวมและความเพี้ยนของฮาร์โมนิกส์แต่ละส่วนด้วยค่าเปอร์เซนต์ของขนาดที่ความถี่หลักมูลเช่นกัน

2. กระแสฮาร์โมนิกส์ (Harmonic Current)

ฮาร์โมนิกส์ที่อยู่ในรูปของกระแสในไลน์ในระบบไฟฟ้าเกิดขึ้นจากอุปกรณ์ที่มีคุณลักษณะไม่เป็นเชิงเส้น (Non-linear Devices) ซึ่งอาจเป็นโหลดหรือแหล่งกำเนิดก็ได้ ในปัจจุบันอุปกรณ์ไม่เป็นเชิงเส้นมีจำนวนมาก ทำให้ผลของฮาร์โมนิกส์ต่อระบบไฟฟ้ากำลังมีมากขึ้น

3. แรงดันฮาร์โมนิกส์ (Harmonic Voltage)

เกิดจากการที่กระแสฮาร์โมนิกส์ไหลผ่านค่ารีแอกแตนซ์ (Reactance) ของระบบทำให้เกิดความผิดเพี้ยนของรูปแรงดันอันเนื่องมาจากรีแอกแตนซ์มีค่าเปลี่ยนแปลงตามความถี่ ส่วนค่าความต้านทานไม่เปลี่ยนแปลง

กระแสฮาร์โมนิกส์ที่ไหลในระบบนั้นจะเป็นตัวทำให้เกิดความผิดเพี้ยนแรงดันฮาร์โมนิกส์ (Harmonic Voltage Distortion) ซึ่งเป็นไปตามสมการต่อไปนี้

$$V_n = I_n Z_n \qquad (2.2)$$

เมื่อ V_n = แรงดันฮาร์โมนิกส์

I_n = กระแสฮาร์โมนิกส์ในระบบ

Z_n = ค่าอิมพีแดนซ์ของระบบ

4. ตัวประกอบความเพี้ยน (Distortion Factor , DF)

ตัวประกอบฮาร์โมนิกส์ (Harmonic Factor , HF)

ความผิดเพี้ยนฮาร์โมนิกส์ทั้งหมด (Total Harmonic Distortion , THD)

ความหมายทั้งสามตัวนั้น IEEE ได้ให้ความหมายไว้เหมือนกัน กล่าวคือ ค่าที่บอกถึงปริมาณของฮาร์โมนิกส์ที่มีอยู่ทั้งหมดโดยเปรียบเทียบค่า rms ของส่วนประกอบความถี่หลักมูล IEEE 519-1992 Harmonic voltage และ Harmonic current

$$DF_V = HF_V = THD_V = \sqrt{\frac{\sum_{n=2}^{\infty} V_n^2}{V_1^2}} \times 100\%$$

$$= \frac{\sqrt{\sum_{n=2}^{\infty} V_n^2}}{V_1} \times 100\%$$

$$DF_I = HF_I = THD_I = \sqrt{\frac{\sum_{n=2}^{\infty} I_n^2}{I_1^2}} \times 100\%$$

$$= \frac{\sqrt{\sum_{n=2}^{\infty} I_n^2}}{I_1} \times 100\%$$

5. ตัวประกอบกำลัง (Power Factor)

หมายถึงอัตราส่วนของกำลังงานจริง (W หรือ kW) ต่อกำลังงานปรากฏ (VA หรือ kVA) อย่างไรก็ตามหากรูปคลื่นของแรงดันและกระแสมีลักษณะรูปคลื่นไซน์ เราอาจพูดถึงตัวประกอบกำลังในเทอมค่าโคไซน์ (cosine) ของมุมเฟสระหว่างแรงดันกับกระแสก็ได้

PF คือ อัตราส่วนกำลังงานจริงต่อกำลังงานปรากฏ

Displacement Power Factor (DPF) คือ อัตราส่วนของกำลังงานจริงจากรูปคลื่นความถี่หลักมูลต่อกำลังปรากฏของรูปคลื่นความถี่หลักมูล หรืออาจกล่าวได้ว่า DPF คือ ค่าโคไซน์ของมุมเฟสของแรงดันความถี่หลักมูลกับกระแสความถี่หลักมูล

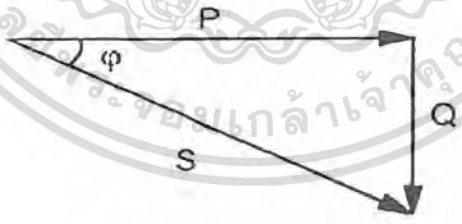
$$\text{DPF} = \text{Active} / \text{Aparent Power (Fund)} \qquad (2.3)$$

$$\text{Total PF} = \text{Active Power} / \text{Apparent Power (Total)} \qquad (2.4)$$

เมื่อ

พิกัดกำลังงานจริง (Active Power) = ผลคูณของแรงดันกับกระแส rms ที่ inphase กัน (Watt)
พิกัดกำลังงานที่ปรากฏ (Apparent Power) = ผลคูณของแรงดันกับกระแส rms (VA)
พิกัดกำลังที่ปรากฏ (Fund) ได้จาก rms ของแรงดันคูณกับ rms ของกระแสของ Fundamental
พิกัดกำลังที่ปรากฏ (Total) ได้จากการรวมเอาฮาร์โมนิกส์มาด้วยเช่น

$$V_1 \times (i_1^2 + i_2^2 + i_3^2 + \ldots)^{1/2} \qquad (2.5)$$



รูปที่ 2.2 : ค่าเพาเวอร์แฟคเตอร์ที่มีรูปคลื่นไซน์

$P = V_{RMS} . I_{RMS} \cos\theta$	In-Phase or Real power
$Q = V_{RMS} . I_{RMS} \sin\theta$	Reactive or Quadrative power
$S = V_{RMS} . I_{RMS}$	Total Apparent Power

บทที่ 3
การวิเคราะห์ฮาร์โมนิกส์ (Harmonics Analysis)

3.1 บทนำ (Introduction)

โดยทั่วไปเราอาจกล่าวได้ว่าฟังก์ชั่นต่อเนื่อง (Continous function) ใดๆ ซึ่งมีการซ้ำทุกๆช่วงเวลา T เราจะสามารถเขียนให้อยู่ในรูปของผลบวกของคลื่นไซน์ (Sine wave) ที่มีความถี่หลักมูล (Fundamental) กับความถี่ฮาร์โมนิกส์ที่ลำดับสูงขึ้นไป ซึ่งเป็นจำนวนเท่าของความถี่หลักมูลได้เสมอ

ดังนั้นในการวิเคราะห์ฮาร์โมนิกส์จะกล่าวถึง กระบวนการในการคำนวณขนาด (Magnitude) และมุมเฟส (Phase angle) ของแต่ละส่วนประกอบที่เป็นมูลฐานและในลำดับฮาร์โมนิกส์ที่สูงขึ้น ผลของอนุกรมที่ได้เรียกว่า อนุกรมฟูเรียร์ (Fourier Series) ซึ่งแสดงความสัมพันธ์ในรูปโดเมนเวลา (Time domain) และโดเมนความถี่ (Frequency domain)

ในกรณีการวิเคราะห์สำหรับฟังก์ชั่นเวลาทั่วไป ซึ่งอยู่ในช่วง ∞ ถึง $-\infty$ จะทำการวิเคราะห์โดยอาศัยการแปลงฟูเรียร์ (Fourier Transform) และ อินเวอร์สการแปลงฟูเรียร์ (Inverse Fourier Transform) ซึ่งการแปลงฟูเรียร์เป็นการแปลงฟังก์ชั่นในโดเมนความถี่ไปสู่โดเมนเวลา ดังนั้นอาจกล่าวได้ว่าอนุกรมฟูเรียร์เป็นกรณีหนึ่งของการแปลงฟูเรียร์

อย่างไรก็ตามในทางปฏิบัติ ข้อมูลหรือค่าที่จะทำการวิเคราะห์นั้น เป็นค่าซึ่งได้จากการสุ่มค่า (Sampled data) ของฟังก์ชั่นในโดเมนเวลา ซึ่งค่าที่ได้จะอยู่ในรูปของขนาดของฟังก์ชั่น ณ เวลาที่ทำการสุ่ม ซึ่งระยะห่างในการสุ่มแต่ละครั้ง จะเป็นช่วงเวลาที่แน่นอน ในการแปลงฟูเรียร์ของค่าที่ได้จากการสุ่มเหล่านี้จะต้องใช้การแปลงที่เรียกว่า การแปลงฟูเรียร์เต็มหน่วย (Discret Fourier Transform : DFT) แต่เราสามารถทำ DFT ให้รวดเร็วขึ้นโดยอาศัยวิธีการที่เรียกว่า การแปลงฟาส์ตฟูเรียร์ (Fast Fourier Transform : FFT) FFT เป็นวิธีการพื้นฐานซึ่งถือได้ว่ารวดเร็ว และได้รับการยอมรับมากที่สุด ในการนำมาวิเคราะห์สเปกตรัมและฮาร์โมนิกส์ของระบบ ดังนั้นในการวิเคราะห์ฮาร์โมนิกส์และการประยุกต์ใช้ต่อไปจะเน้นความสำคัญของ FFT

3.2 การวิเคราะห์อนุกรมฟูเรียร์ (Fourier Series Analysis)

อนุกรมฟูเรียร์ของฟังก์ชั่นคาบ (Periodic function) X(t) ใดๆ อาจแสดงได้ดังนี้

$$X(t) = a_0 + \left(a_n \cos\left(\frac{2\pi\pi n}{T}\right) + b_n \sin\left(\frac{2\pi\pi n}{T}\right)\right) \qquad (3.1)$$

โดยที่ a_0 เป็นค่าเฉลี่ยของฟังชั่น X(t)

a_n และ b_n เป็นสัมประสิทธิ์ของอนุกรมของฟังก์ชั่น X(t)

ทั้งนี้ a_0 , a_n และ b_n สามารถหาได้จากผลการอินทิเกรทต่อไปนี้

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\,dt \tag{3.2}$$

ซึ่งหมายถึงพื้นที่ใต้กราฟของ X(t) ในช่วง-T/2 ถึง T/2 นั่นเอง

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

และ

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} x(t)\sin(\frac{2\pi\pi n}{T})dt \qquad \text{โดยที่ } n = 1 \rightarrow \infty \tag{3.4}$$

3.1.1 อนุกรมฟูเรียร์ในรูปเชิงซ้อน (Complex Form of the Fourier Series)

โดยทั่วไปเราสามารถเขียนอนุกรมฟูเรียร์ของฟังก์ชั่นคาบ X(t) ใดๆได้ในรูปต่อไปนี้

$$X(t) = a_0 + A_1\sin(a)t + \phi_1) + A_2\sin(2a)t + \phi_1)+\ldots\ldots \tag{3.5}$$

โดยแต่ละลำดับของอนุกรมอาจสามารถเขียนได้ดังนี้

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

โดยที่ $e^{j2\pi fnt}$ เป็นเวกเตอร์ 1 หน่วย และ X(fn) เป็นแอมปลิจูดและมุมเฟสของเวกเตอร์ฮาร์โมนิกส์ใดๆ ดังนั้นสำหรับฟังก์ชันในโดเมนเวลา X(t) ใด เราสามารถเขียนในรูปเชิงซ้อนได้ดังนี้

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

3.3 การแปลงฟูเรียร์(Fourier transform)

ในการวิเคราะห์ฟูเรียร์นั้น สำหรับสัญญาณที่เป็นคาบ เราจะได้ว่าผลการวิเคราะห์จะอยู่ในรูปของอนุกรมของแต่ละความถี่ซึ่งเป็นจำนวนเท่าของความถี่หลักมูลเท่านั้น (Series of discrete frequency component)

ถ้าเรานำมาประยุกต์ใช้กับสัญญาณซึ่งต่อเนื่องทั่วไปซึ่งไม่จำเป็นต้องเป็นสัญญาณที่เป็นคาบ โดยขยายการอินทิเกรตในช่วยคาบ T ให้กว้างมากๆ หรือเป็นช่วง ∞ นั่นเอง ดังนั้นช่วงกว้างของแต่ละความถี่จึงเข้าใกล้ศูนย์ ดังนั้นเราจะได้ ฟังก์ชัน X(fn) เป็นฟังก์ชั่นต่อเนื่องหรือเราอาจเขียนใหม่ได้เป็น

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

โดยที่ X(t) เป็นฟังก์ชันโดเมนเวลาซึ่งต่อเนื่อง และเรียก X(f) ว่าเป็นการแปลงฟูเรียร์ของโดเมนเวลาX(t) ในทางกลับกันเราจะได้

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

เรียก X(t) ว่าเป็นอินเวอร์สการแปลงฟูเรียร์ของฟังก์ชั่น โดเมนความถี่ X(f)
โดยเรียกคู่ X(f) จะอยู่ในรูปเชิงซ้อนซึ่งอาจเขียนได้ดังนี้

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

ส่วนจริงของ X(f) หาได้จาก

$$\mathrm{Re}\, X(f) = 0.5[X(f) + X(-f)]$$

$$= \int_{-\infty}^{\infty} x(t) \cos 2\pi ft\, dt \qquad (3.11)$$

และส่วนจินตภาพของ X(f) หาได้จาก

$$\text{Im } X(f) = 0.5j[X(f) - X(-f)]$$
$$= -\int_{-\infty}^{\infty} x(t) \sin 2\pi ft \, dt \qquad (3.12)$$

ดังนั้นสเปกตรัมของแอมปลิจูด (Amplitude spectrum) ของแต่ละความถี่หาได้จาก

$$|X(f)| = [(\text{Re}X(f))^2 + (\text{Im } X(f))^2]^{1/2} \qquad (3.13)$$

และสเปกตรัมของมุมเฟส (Phase spectrum) หาได้จาก

$$\phi(f) = \tan^{-1}\left[\frac{I_m X(f)}{R_e X(f)}\right] \qquad (3.14)$$

3.4 การแปลงฟูเรียร์แบบเต็มหน่วย (Discrete Fourier Transform)

ในกรณีที่ต้องการให้ค่าในโดเมนความถี่เป็นฟังก์ชั่นที่ผ่านการสุ่มค่าเช่นเดียวกับในโดเมนเวลา ผลการแปลงฟูเรียร์ที่ได้จะประกอบด้วยส่วนประกอบย่อยรวมกันดังนี้

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

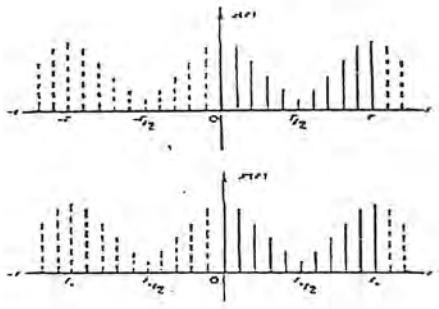
เรียกการแปลงฟูเรียร์ลักษณะนี้ว่า การแปลงฟูเรียร์แบบเต็มหน่วย (Discrete Furier Transform : DFT)

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

เรียกการแปลงลักษณะนี้ว่า อินเวอร์การแปลงฟูเรียร์แบบเต็มหน่วย (Inverse Discrete Furier Transform : IDFT)

สมมุติให้ฟังก์ชั่นทั้งในโดเมนเวลาและโดเมนความถี่เป็นฟังก์ชั่นคาบดังรูปที่ 3.1 ซึ่งมีอัตราการสุ่ม N ครั้งต่อคาบ

คู่การแปลงฟูเรียร์ที่ได้จะอยู่ในรูปเต็มหน่วย (Discrete form) ซึ่งเป็นรูปแบบที่เหมาะสมในการนำมาคำนวณโดยการประมวลผลแบบดิจิตอลต่อไป



รูปที่ 3.1 รูปแบบฟังก์ชั่นเต็มหน่วยทั้งในโดเมนเวลาและโดเมนความถี่

เราอาจเขียนผลการแปลงได้ใหม่ในรูปดังนี้

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

โดยที่ $$W = e^{-j2\pi/N} \tag{3.18}$$

ดังนั้นจากสมการดังกล่าวเราอาจเขียนในรูปเมตริกได้ดังนี้

$$\begin{bmatrix} X(f_0) \\ X(f_1) \\ \vdots \\ X(f_k) \\ \vdots \\ X(f_{n-1}) \end{bmatrix} = \frac{1}{N}\begin{bmatrix} 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & W & \cdots & W^k & \cdots & W^{N-1} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & W^k & \cdots & W^{k^2} & \cdots & W^{k(N-1)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & W^{n-1} & \cdots & W^{(N-1)k} & \cdots & W^{(N-1)^2} \end{bmatrix}\begin{bmatrix} x(t_o) \\ x(t_1) \\ \vdots \\ x(t_k) \\ \vdots \\ x(t_{n-1}) \end{bmatrix}$$

หรือเขียนในรูปอย่างง่ายได้เป็น

$$[X(f_k)] = \frac{1}{N}[W_{kn}][x(t_n)] \tag{3.19}$$

จะเห็นได้ว่าตามสมการข้างบนถ้าต้องการคำนวณค่าในโดเมนความถี่จากข้อมูลซึ่งได้จากการสุ่มค่าในโดเมนเวลาจำนวน N ค่านั้นจะต้องทำการคูณค่าเชิงซ้อนถึง N^2 ครั้ง

แต่ละสมาชิกในเมตริก [W^{kn}] นั้นแสดงเวกเตอร์หนึ่งหน่วยซึ่งหมุนตามเข็มนาฬิกามีมุมเฟส $2p\pi / N$ $(p=0,1,2 \ldots,(N-1))$

และ

$$W^{kn} = W^{[kn \bmod N]} \tag{3.20}$$

โดย kn mod N หมายถึงเศษจากการหาร kn ด้วย

ตัวอย่างเช่นในกรณีที่ N= 8 ดังนั้น

$$W = e^{-j2\pi/8} = \cos\frac{\pi}{4} - j\sin\frac{\pi}{4} \tag{3.21}$$

ดังนั้นจะได้

$$\begin{aligned} W^0 &= -W^4 = 1 \\ W^1 &= -W^5 = \left(\frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}\right) \\ W^2 &= -W^6 = -j \\ W^3 &= -W^7 = -1\left(\frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}\right) \end{aligned} \tag{3.22}$$

เราอาจมองว่าเวกเตอร์หนึ่งหน่วยแต่ละตัวหมุนไปโดยมีมุมเฟส $\pm 0^0$, $\pm 45^0$, $\pm 90^0$ และ $\pm 135^0$ ตามลำดับ จะสังเกตได้ว่า W มีค่าเท่ากับ 1 ซึ่งเป็นการหมุนที่ครบรอบ ดังนั้นสำหรับ kn ซึ่งมากกว่า 8 เราหาค่าเวกเตอร์หนึ่งหน่วยดังกล่าวได้โดยลบจำนวนที่เต็มรอบออกจาก kn ตัวอย่างเช่น k = 5 และ n = 6 ดังนั้น kn = 30 และจะได้ $W^{30} = W^{(3x8)+6} = W^6 = j$

สำหรับ N = 8 จะได้ [W^{kn}] ดังนี้

1	1	1	1	1	1	1	1
1	W	-j	W^3	-1	-W	j	$-W^3$
1	-j	-1	j	1	-j	-1	j
1	W^3	1	W	-1	$-W^3$	-j	-W
1	-1	-1	-1	1	-1	1	-1
1	-W	-j	$-W^3$	-1	W	j	W^3
1	j	-1	-j	1	j	-1	-j
1	$-W^3$	j	-W	-1	W^3	-j	W

ตารางที่ 3.1 แสดงค่า $[W^{kn}]$ กรณี N = 8

จากสมการเมตริกซ์จะเห็นชัดว่าส่วนดีซี (DC component : $X(f_0)$ ของสเปกตรัมความถี่ ได้จากการรวมค่าจากการสุ่มในโดเมนเวลาทุกค่า แล้วหารผลรวมดังกล่าวด้วยจำนวนข้อมูลที่ได้จากการสุ่มค่า ซึ่งหมายถึงค่าเฉลี่ยของข้อมูลจากการสุ่มในโดเมนเวลานั่นเอง

ในแถวถัดลงมาจากการสุ่มแต่ละค่าจะถูกถ่วงน้ำหนักโดยเวคเตอร์ที่มีมุมต่างกันซึ่งขึ้นกับลำดับของแถวนั้นๆ ดังนั้นสำหรับ $X(F_1)$ แต่ละเวคเตอร์ที่นำมาถ่วงน้ำหนักค่าจากการสุ่มจะมีมุมต่างกัน $-(1/N)2\pi$ และสำหรับ $X(F_2)$ แต่ละเวคเตอร์จะมีมุมต่างกัน $-(2/N)\,2\pi$

3.5 การแปลงฟาสต์ฟูเรียร์ (Fast Fourier Transform หรือ FFT)

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

เราอาจกล่าวถึงวิธีการซึ่งสามารถลดจำนวนครั้งของการคูณลงได้เหลือ $N \log_2 N$ ครั้ง หรือลดจำนวนครั้งลงไปถึง $N/(\log_2 N)$ เท่า ซึ่งเรียกวิธีการนี้ว่า การแปลงฟาสต์ฟูเรียร์ (Fast Fourier Transform หรือ FFT)

3.5.1 การแปลงฟาสต์ฟูเรียร์แบบฐานสอง (Radix 2 FFT)

3.5.1.1 หลักการเบื้องต้นของ FFT

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

$$X(k) = \sum_{m=0}^{N-1} x(m).W^{mk} \qquad (3.23)$$

โดยที่ k,m = 0,1,..., *N-1* และจำนวนเชิงซ้อน $W = \exp(-j2\pi/N)$ x (m) และ X(k) เป็นสัญญาณในโดเมนเวลาและโดเมนความถี่ตามลำดับ

เราสามารถเขียนสมการ (3.23) ในรูปของสมการเมตริกซ์ได้เป็น

$$\{X\} = \{A\}.\{x\} \qquad (3.24a)$$

โดยที่ {X} และ {x}เป็นเวคเตอร์แนวตั้ง (column vector) ที่ประกอบด้วยสมาชิก X(k) และ X(m) ตามลำดับจำนวน N ลำดับ และ {A} เป็นเมติกซ์จตุรัส (square matrix) ขนาด N x N ที่มีสมาชิกเป็นจำนวนเชิงซ้อน W_{mk} เช่นถ้าพิจารณาที่ N = 4 เราสามารถเขียนแยกออกได้เป็น

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (3.24b)$$

แต่เนื่องจากคุณสมบัติความเป็นคาบของ W คือ

$$W_{mk} = W[mk \bmod(n)] \qquad (3.25)$$

สมการ 3.24 b อาจเขียนได้เป็น

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (3.26)$$

คุณสมบัติความเป็นคาบทำให้เราต้องแยกตัวประกอบของเมตริกซ์ ออกเป็นเมตริกซ์ย่อยหลายเมตริกซ์คูณกัน และ สมาชิกภายในเมตริกซ์ย่อยให้มีค่าเป็นศูนย์มากที่สุด วิธีการแยกตัวประกอบนี้จะไม่กระทำโดยตรง แต่จะมีการสลับตำแหน่งหรือจัดกลุ่มของเมตริกซ์ด้วยวิธีการกลับบิต (bit reversal) และเมตริกซ์หลังจัดการสลับแถวแล้วนำมาแยกตัวประกอบอีกครั้ง

วิธีการแยกตัวประกอบอาจทำโดยวิธีการใช้ตัวเลขฐานสอง โดยการแทนดรรชนี k และ m ของสมการ (3.23) ด้วยเลขฐานสอง กรณีที่ดรรชนี k และ m จะมีค่าได้เพียง 0,1,2, และ 3 เท่านั้น ดังนั้น

$$k = (k_p\ k_o)\ ,\ m = (m_p\ m_o) \tag{3.27}$$

โดยที่ kp ko mp และ mo เป็นเลขโดดที่มีค่าแค่ 0 และ 1 เท่านั้น ดังนี้

$$k = 2k_p\ k_o\ ,\ \ m = (2m_p\ m_o) \tag{3.28}$$

แทนค่า และ ลงในสมการ (3.23)

$$X(k_p\ k_o) = x(m_1\ m_0)W^{(2m\quad 1 + m0).(2k + ko)} \tag{3.29}$$

โดยคุณสมบัติความเป็นคาบของ W และ $W^{4m1k1} = 1$ ดังนั้นในสมการ (3.29) ได้ใหม่เป็น

$$\begin{aligned} X(k_1,k_o) &= \sum_{m_o=0}^{1}\left\{\sum_{m_1=1}^{1} x(m_1,m_o)W^{2m_1k_o}\right\}W(2k_1+k_0)m_o \\ &= \left\{\sum_{m_1=1}^{1} x(k_o,m_o)\right\}W(2k_1+k_o)m_o \end{aligned} \tag{3.30a}$$

โดยสมมติให้ตัวแปร เป็นการคำนวณระหว่างกลาง ผลของสมการ (3.30 a) เขียนเป็นเมตริกซ์ใหม่ได้เป็น

$$\begin{matrix} (k_1,k_0) \\ \begin{bmatrix} x(0,0) \\ x(1,0) \\ x(0,1) \\ x(1,1) \end{bmatrix} \end{matrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^0 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{matrix} (m_1,m_0) \\ \begin{bmatrix} x(0,0) \\ x(0,1) \\ x(1,0) \\ x(1,1) \end{bmatrix} \end{matrix} \tag{3.30b}$$

ซึ่งมีผลลัพธ์ระหว่างกลาง และ ผลลัพธ์ เป็น

$$\begin{matrix} (k_1,k_0) \\ \begin{bmatrix} x(0,0) \\ x(0,1) \\ x(1,0) \\ x(1,1) \end{bmatrix} \end{matrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{matrix} (m_1,m_0) \\ \begin{bmatrix} x(0,0) \\ x(0,1) \\ x(1,0) \\ x(1,1) \end{bmatrix} \end{matrix} \tag{3.31a}$$

และค่า DFT ของสัญญาณเป็น

$$\begin{matrix} (k_0,k_0) & & & & & & (m_1,m_0) \end{matrix}$$

$$\begin{bmatrix} x(0,0) \\ x(0,1) \\ x(1,0) \\ x(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0,0) \\ x(1,0) \\ x(0,1) \\ x(1,1) \end{bmatrix} \qquad (3.31b)$$

จะเห็นได้ว่าจากผลการแยกตัวประกอบของ {A} ได้ว่า สมาชิกตามแนวนอนของตัวประกอบเมตริกซ์จะมีเพียง 2 ตัวเท่านั้นที่มีค่าเป็นศูนย์ และในสองตัวนี้จะมีสมาชิกหนึ่งตัวมีค่าเป็นศูนย์เสมอ ในขณะที่อีกตัวจะเป็นจำนวนเชิงซ้อน ซึ่งเราต้องการคูณจำนวนเชิงซ้อนเพียง Nlog 2 N = 8 ครั้ง และบวกจำนวนเชิงซ้อนอีก 8 ครั้ง ในขณะที่การคำนวณปกติใช้การคูณจำนวนเชิงซ้อน 16 ครั้งและบวกจำนวนเชิงซ้อน 12 ครั้ง

เรายังสามารถลดจำนวนการคูณได้อีก จากการคำนวณสัญญาณระหว่างกลาง x1 (0,0) และ x1 (0,1) ซึ่ง

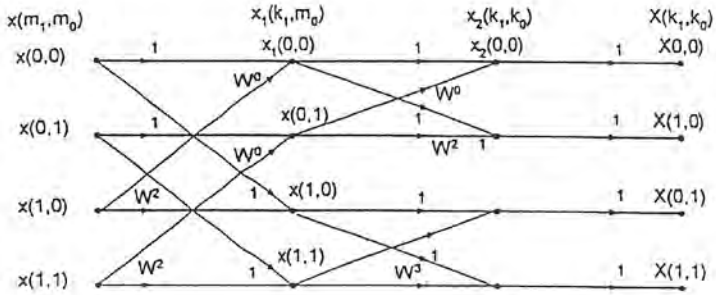
$$x_1(0,0) = x(0,0) + W_0 . x(1,0) = x(0,0) + W_0 . x(1,0)$$

และ (3.32)

$$x_1(1,0) = x(0,0) + W_0 . x(1,0) = x(0,0) + W_0 . x(1,0)$$

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 3.2 โดยหัวลูกศรชี้ทิศทางการคำนวณ ส่วนอักษรกำกับเป็นค่าตัวคูณค่าของสัญญาณที่ต้นลูกศรนั้น และที่บัพ หรือ ปม (Node) เป็นการรวมหรือบวกกันของสัญญาณ ส่วน $x_1(k_0,m_0)$ แทนการคำนวณระหว่างกลางและ $X(k_1,k_0)$ เป็นค่า DFT ของลำดับสัญญาณ



รูปที่ 3.2 กราฟการไหลแสดงถึงวิธีการคำนวณตามสมการ 3.31

3.5.1.2 ขั้นตอนการลดทอนทางเวลา

วิธีการที่เสนอมาแล้วนั้น จะเป็นการจัดกลุ่มลำดับสัญญาณในโดเมนเวลา x(m) ที่มีขนาด N จุด ออกเป็นสองอันดับสัญญาณที่มีความยาว N/2 จุดเท่ากัน โดยเรียกว่าลำดับสัญญาณคู่และลำดับสัญญาณคี่ โดยที่ลำดับสัญญาณคู่เกิดจากการเอาลำดับสัญญาณในตำแหน่งเลขคู่มาเรียงกัน ที่เหลือเป็นลำดับสัญญาณคี่ ดังนั้นถ้าเราให้ xE(m) เป็นลำดับคู่ และลำดับคี่เป็น x0(m) เพราะฉะนั้น

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

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

$$X(k) = \sum_{m=0}^{N-1} x_E(m)(W_n)^{km} + \sum_{m=0}^{N-1} x_0(m)(W)^{km}$$
$$m = \text{เลขคู่} \qquad m = \text{เลขคี่}$$
$$= \sum_{m=0}^{N/2-1} x(2m)(W_n)^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1)(W_N)^{(2m+1)k} \tag{3.34}$$

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

$$X(k) = \sum_{m=0}^{N/2-1} x_E(m)(W_{n/2})^{km} + (W_N)^k \sum_{m=0}^{N/2-1} x_0(m)(W_{N/2})^{km}$$

$$X(k) = X_1(k) + (W_n)^k X_2(k) \qquad (3.35)$$

โดยที่ $X_1(k)$ และ $X_2(k)$ ผลการแปลง DFT ขนาด N/2 จุดของลำดับ $X_E(m)$ และ $X_0(m)$ ตามลำดับ สมการที่(3.35) แสดงให้เห็นว่าการคำนวณ DFT ขนาด N จุดนั้นสามารถแบ่งการคำนวณย่อยออกเป็นการคำนวณ DFT ขนาด N/2 จุด สองอันดับได้ และข้อสำคัญคือ จำทำให้การคูณเชิงซ้อนลดลงไปประมาณ 50 เปอร์เซ็นต์ โดยหลักการนี้ทำให้เรายังสามารถแบ่งทอนลำดับ $x_E(m)$ และ $x_0(m)$ ออกเป็นลำดับคู่และลำดับคี่ได้อีก จนในที่สุดจะเหลือลำดับขนาด 2 จุด หรืออาจกล่าวได้ว่า การคำนวณ DFT ขนาด N จุด ทำได้โดยการแปลง DFT ขนาด 2 จุด จำนวน N/2 ภาคด้วยกัน ข้อสังเกตคือการซอยเพื่อแบ่งลำดับ x(n) ออกเป็นทีละครึ่งจนเหลือการคำนวณ DFT ขนาด 2 จุดนี้ สำหรับสัญญาณ N ลำดับ จะทำการแบ่งได้ $\log_2 N$ ครั้ง (ดังรูป 3.4)

การนำ DFT ขนาด 2 จุด จำนวน N/2 ภาคนี้มาประกอบกันเพื่อให้ได้การคำนวณ DFT ขนาด N จุดนั้น จะต้องมีหลักเกณฑ์ในการทำเพื่อไม่ให้ค่าที่ได้ผิดพลาดไป ดังนั้นเราต้องทำการนิยามสมการ (3.35) สำหรับ $k > N/2$ ด้วยซึ่งทำได้โดยการเขียน

$$\begin{aligned} X(k) &= X_1(k) + (W_n)^k X_2(k) \quad ; 0 \le k \le (N/2)-1 \\ &= X_1(k-N/2) + (W_n)^k X_2(k-N/2) \; ; N/2 \le k \le N/2-1 \end{aligned} \qquad (3.36)$$

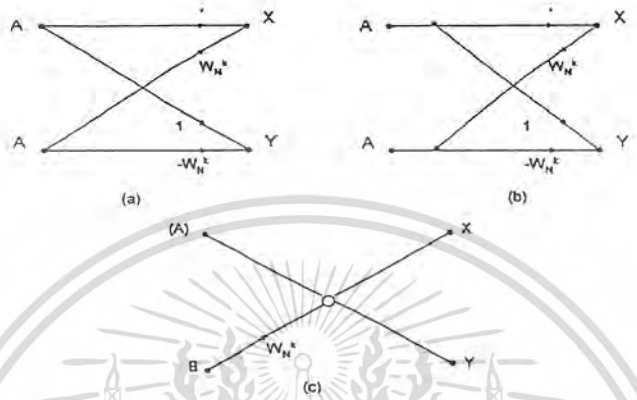
พจน์ $(W_N)^k$ ในสมการ 3.36 เรียกว่าตัวประกอบการหมุน (twiddle factor) ซึ่งมีความสำคัญในการนำ DFT ขนาด 2 จุด หรือ DFT ขนาด N/2 จุดมาประกอบกันเป็น DFT ขนาด N จุดได้เหมือนเดิมและจากความสัมพันธ์ $(W_N)^{K-N/2} = -(W_N)^k$ เราจะได้

$$X(k) = X_1(k) + (W_N)^k X_2(k) \quad ; 0 \le k \le (N/2)-1 \qquad (3.37a)$$

$$= X_1(k-N/2)-(W_N)^{k-N/2} X_2(k-N/2) \quad ; N/2 \le k \le N/2-1 \qquad (3.38b)$$

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

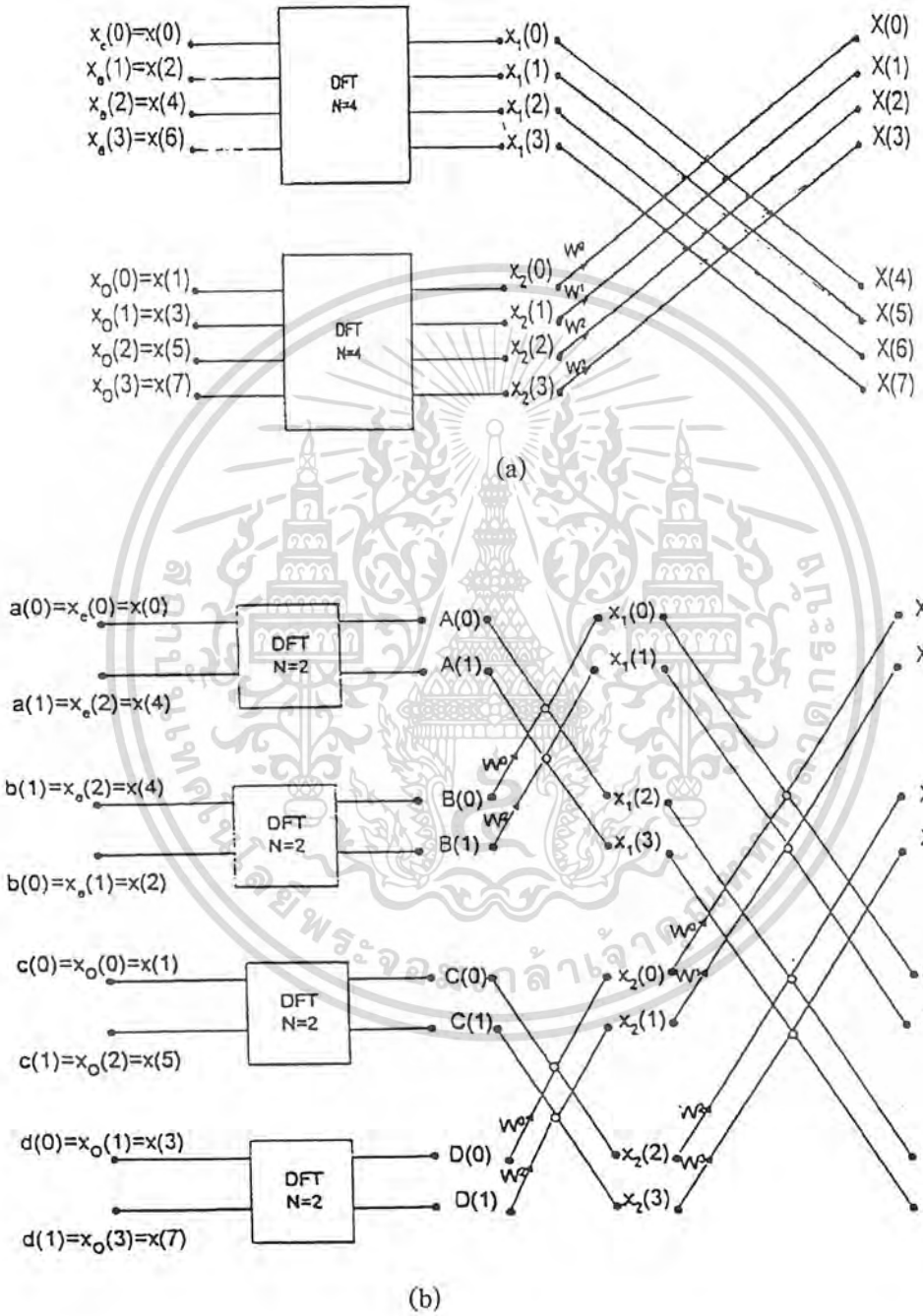
$$X = A + (W_N)^k.B$$
$$Y = A - (W_N)^k.B \qquad (3.38)$$



รูปที่ 3.3 หน่วยผีเสื้อของการคำนวณตามขั้นตอนวิธีลดทอนทางเวลา

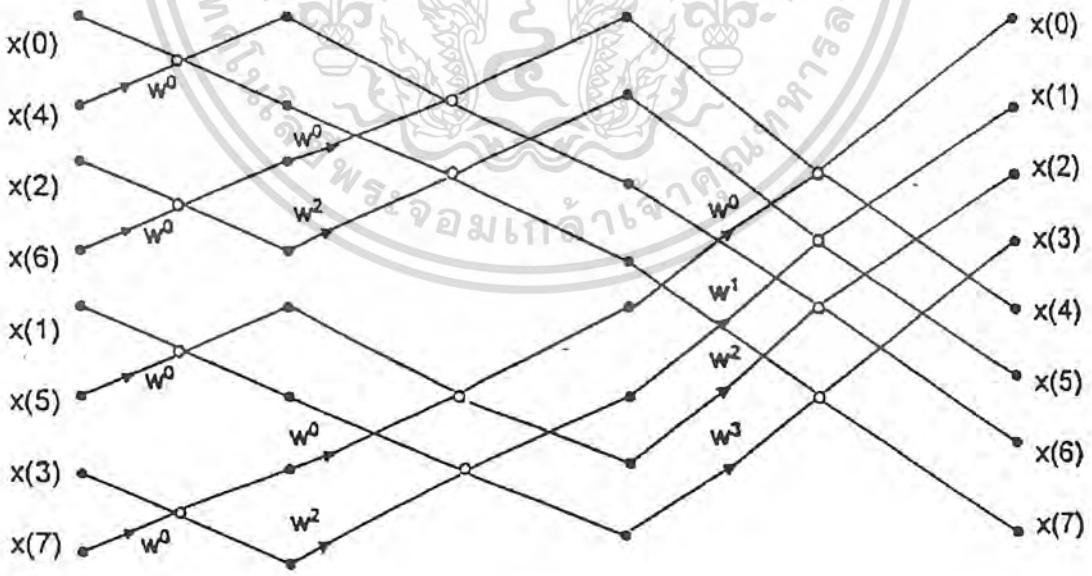
โดยที่การทำงานของหน่วยผีเสื้อ แทนได้ด้วยกราฟการไหล ดังแสดงไว้ในรูป 3.3(a)3.3(b) หรือ 3.3(c)

โดยในรูปที่ 3.4 แสดงการคำนวณ DFT แบบ 8 จุด

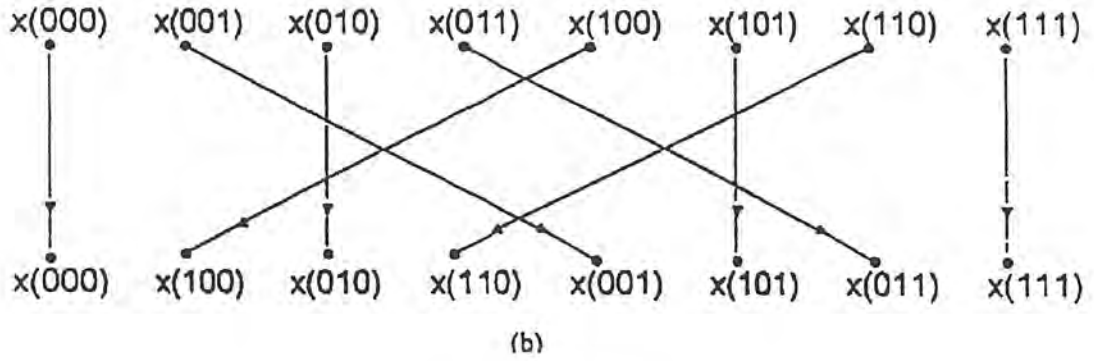


รูปที่ 3.4(a) และ (b) แสดงขั้นตอนแบบ การลดทอนทางเวลา สำหรับ DFT แบบ 8 จุด

ตามรูป 3.5(a) ลำดับสัญญาณเข้า x(n) ไม่ได้ถูกจัดเรียงอย่างต่อเนื่อง หรือ ตามธรรมชาติแต่ไม่ถูก สลับตำแหน่งกันอย่างมีหลักเกณฑ์ คือการสลับตำแหน่ง หรือสลับอันดับ กันนี้จะเป็นไปตามวิธีการที่เรียกว่า การผันกลับบิต นั่นคือถ้าเราแทนดรรชนี n ของลำดับ x(n) ด้วยเลขฐานสองโดยที่จำนวนบิตต้องเพียงพอที่จะแทนค่า N ได้ เช่น ในกรณี N=8 ที่ต้องแทนกันด้วยเลขฐานสอง 3 บิต จากนั้นการจัดลำดับ x(n) ใหม่จะได้จากการผันกลับบิตของเลขฐานสองที่แทนดรรชนี n ดังรูป 3.5(b) คือ x(001) จะถูกแทนด้วย x(100) และ x(110) ถูกแทนด้วย x(011x) เป็นต้น เนื่องจากดรรชนี n เป็นดรรชนีในโดเมนเวลา และวิธีการของ FFT แบบนี้เป็นการลดทอนเวลาทางการคำนวณโดยการ สับ หรือ ตัดทอน ลำดับในโดเมนเวลา หรือ x(n) ออกเป็นกลุ่มย่อยโดยแต่ละกลุ่มประกอบด้วยลำดับ x(n) เพียงสองลำดับที่เป็น ปมคู่กัน การจัดกลุ่มนี้คล้ายกับเป็นการสุ่มตัวอย่างลำดับเดิมอีกครั้งหนึ่ง ด้วยอัตราการสุ่มตัวอย่างที่ต่ำกว่า และถ้าหากเราถือว่าแต่ละกลุ่มข้อมูลใหม่ที่จัดทำขึ้นมา ต่างเป็นลำดับ ข้อมูลชุดหนึ่งแล้ว ก็เท่ากับว่าเราได้ตัดทอนลำดับในโดเมนเวลาลงไปเป็นกลุ่มลำดับข้อมูลย่อยหลายลำดับ ดังนั้นจึงเรียกวิธีการแบบนี้ว่า การลดทอนทางเวลาซึ่งแสดงแผนภาพตามลำดับวิธีการของ การลดทอนทางเวลา ได้ดังรูป 3.5 โดยที่ DFT ขนาด N จุด เดิมถูกแบ่งออกเป็น DFT ขนาด N/2 จุดจำนวน 2 ภาคนำมารวมกันโดยใช้ตัวประกอบการหมุนและลำดับนี้จะทำจนกระทั่งผลลำดับสุดท้ายเป็นการแปลง DFT ขนาด 2 จุด โดยที่การนำเอา DFT ขนาด N/2 จุดมาประกอบกัน มีวิธีการที่ขึ้นอยู่กับการจัดเรียงตัวประกอบการหมุน



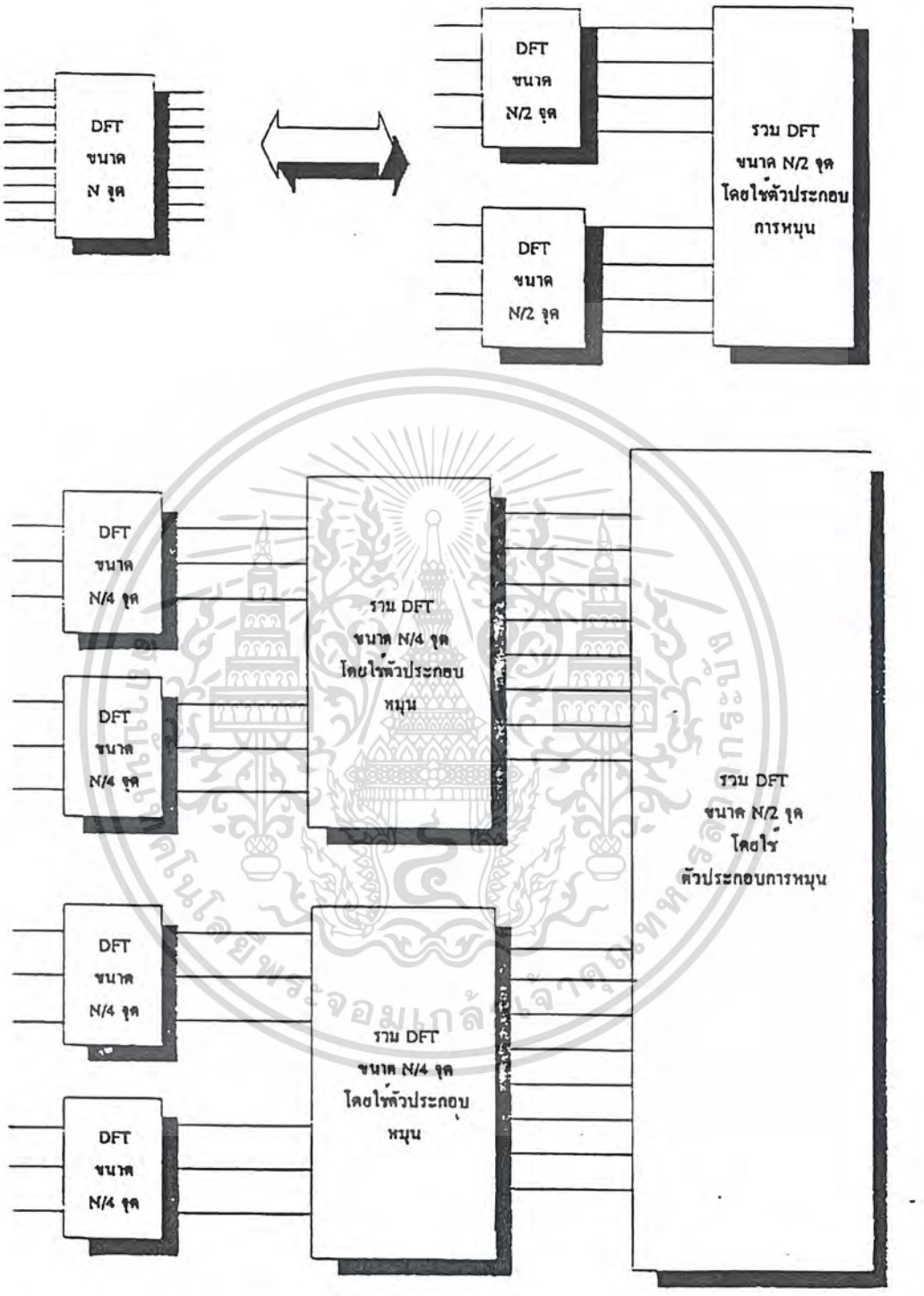
รูปที่ 3.5(a) กราฟการไหลสัญญาณแสดงการคำนวณตามรูป 3.4



รูปที่ 3.5(b) แสดงการสลับตำแหน่งของลำดับ x(n) ด้วยการผันกลับบิต

3.5.1.3 การลดทอนทางเวลาโดยวิธีการซ้ำที่ตามรูป

คุณสมบัติที่เป็นข้อดีบางประการของการแปลงฟาสต์ฟูเรียร์ 3.5(a) โดยทั่วไปแล้วอาจกล่าวได้ว่าการคำนวณการแปลง DFT นั้นแท้จริงแล้วก็คือ การนำลำดับจำนวนเชิงซ้อนที่มีอยู่ N ลำดับมาทำการแปลงเป็นจำนวนเชิงซ้อนอีกกลุ่มหนึ่งที่มี N ลำดับเช่นกัน โดยการใช้ FFT การแปลงเช่นนี้จะกระทำ log2N ขั้นตอนด้วยกัน ดังนั้นตามรูป 3.5(a) ซึ่ง N=8 ควรต้องมีหน่วยความจำ ($\log_2 N$) หรือ 3 แถวตามลำดับ (array) ด้วยกันสำหรับเก็บข้อมูลที่ต้องใช้ในการคำนวณ โดยที่แถวลำดับแรกไว้เก็บลำดับข้อมูล x(n) สองแถวลำดับ $x_1(k)$ และแถวสุดท้ายสำหรับผลลัพธ์ X(k) โดยการคำนวณจะประกอบด้วยหน่วยผีเสื้อ ซึ่งลักษณะการคำนวณของหน่วยผีเสื้อนี้ หากเรามีหน่วยความจำต่างหากไว้สำหรับเก็บค่าผลคูณของจำนวนเชิงซ้อน $(W_n)^k.B$ ผลลัพธ์ X กับ Y ที่คำนวณสามารถเก็บแทนที่ไว้ในหน่วยความจำที่เก็บลำดับข้อมูลเข้า A และ B ได้ (ดังรูปที่ 3.3) โดยลักษณะการทำเช่นนี้จะเห็นว่า ตามรูป 3.5(a) ผลลัพธ์การคำนวณทางขวามือสามารถบรรจุแทนที่ในหน่วยความจำทางด้านซ้ายมือได้โดยไม่มีผลต่อการคำนวณในส่วนอื่นๆซึ่งเรียกว่าเป็นการคำนวนแบบซ้ำที่ (in place) ซึ่งมีข้อดีคือใช้หน่วยความจำเพียงหนึ่งแถวลำดับ หรือต้องการหน่วยความจำสำหรับเก็บจำนวนเชิงซ้อนเพียง N+1 ค่าเท่านั้นวิธีการนี้จะเหมาะสำหรับข้อมูลยาวมากโดยการคำนวณจะไม่เปลืองเนื้อที่หน่วยความจำ ข้อดีอีกประการคือ ตัวประกอบหมุน $(W_n)^k$ นั้นจะถูกเรียงอย่างเป็นลำดับคือจากกำลังน้อยไปสู่กำลังมาก จึงทำให้การเขียนโปรแกรม หรือสร้างวงจรทำได้ง่ายกว่า แต่มีข้อเสียตรงที่ว่าลำดับ x(n) จะต้องมีการสลับตำแหน่งกันตาม วิธีการผันกลับบิตจึงต้องมีโปรแกรมหรือวงจรเพิ่มเติม



รูปที่ 3.6 ภาพรวมแสดงขั้นตอนวิธีการ DFT ขนาด N จุดแบบลดทอนเวลา

3.5.1.4 การแปลงฟาสต์ฟูเรียร์ชนิดลดทอนทางความถี่

(Decimation-in-Frequency หรือ DIF)

ลำดับการคำนวณ FFT ซึ่งใช้กันมากและประยุกต์ใช้ในโครงสร้างนี้คือ การลดทอนทางความถี่ ซึ่งมีหลักการคล้ายคลึงกับ การลดทอนทางเวลา โดยที่การลดทอนทางความถี่จะแบ่งลำดับโดเมนเวลา x(m) ออกเป็น 2 ส่วนเท่าๆกัน โดยการแบ่งครึ่งซึ่งทำได้โดย ถ้าให้ $x_E(m)$ และ $x_0(m)$ แทนลำดับสัญญาณที่ได้จากการแบ่งครึ่งนี้

$$\begin{aligned} x_E(m) &= x(m) && ; m = 0,1,\ldots,(N/2)-1 \\ x_0(m) &= x(m+N/2) && ; m = 0,1,\ldots,(N/2)-1 \end{aligned} \tag{3.39}$$

เพราะฉะนั้นการคำนวณ DFT ขนาด จุดของ x(n) สามารถเขียนแยกได้สองส่วนคือ

$$\begin{aligned} X(K) &= \sum_{m=0}^{N/2-1} x(m)(W_N)^{mk} + \sum_{m=N/2}^{N-1} x(m)(W_N)^{mk} \\ &= \sum_{m=0}^{N/2-1} x_E(m)(W_N)^{mk} + \sum_{m=0}^{N/2-1} x_0(m)(W_N)^{(m+N/2)k} \\ X(k) &= \sum_{m=0}^{N/2-1} x_E(m)(W_N)^{mk} + \sum_{m=0}^{N/2-1} x_0(m)(W_N)^{mk} \end{aligned} \tag{3.40}$$

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

$$X(k) = \sum_{m=0}^{N/2-1} [x_E(m) + (-1)^k . x_0(m + N/2](WN)^{mk} \tag{3.41}$$

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

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

และ

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

$$= \sum_{m=0}^{N/2-1} f(m)(W_{N/2})^{mk} \qquad (3.43)$$

จากผลของ (3.42) และ (3.43) แสดงว่าการคำนวณ DFT ขนาด N จุด ด้วยวิธีนี้จะทำได้โดย ในเบื้องแรกจากลำดับ x(m) เราทำการแบ่งครึ่งออกเป็น 2 ลำดับ ดังสมการ (3.39) แล้วนำมาสร้างลำดับอันใหม่ที่ยาว N/2 จุด 2 ลำดับ โดยสมมติให้ลำดับใหม่นี้ชื่อ f(m) และ g(m) ลำดับคู่นี้ สามารถสร้างได้โดยใช้สมการ

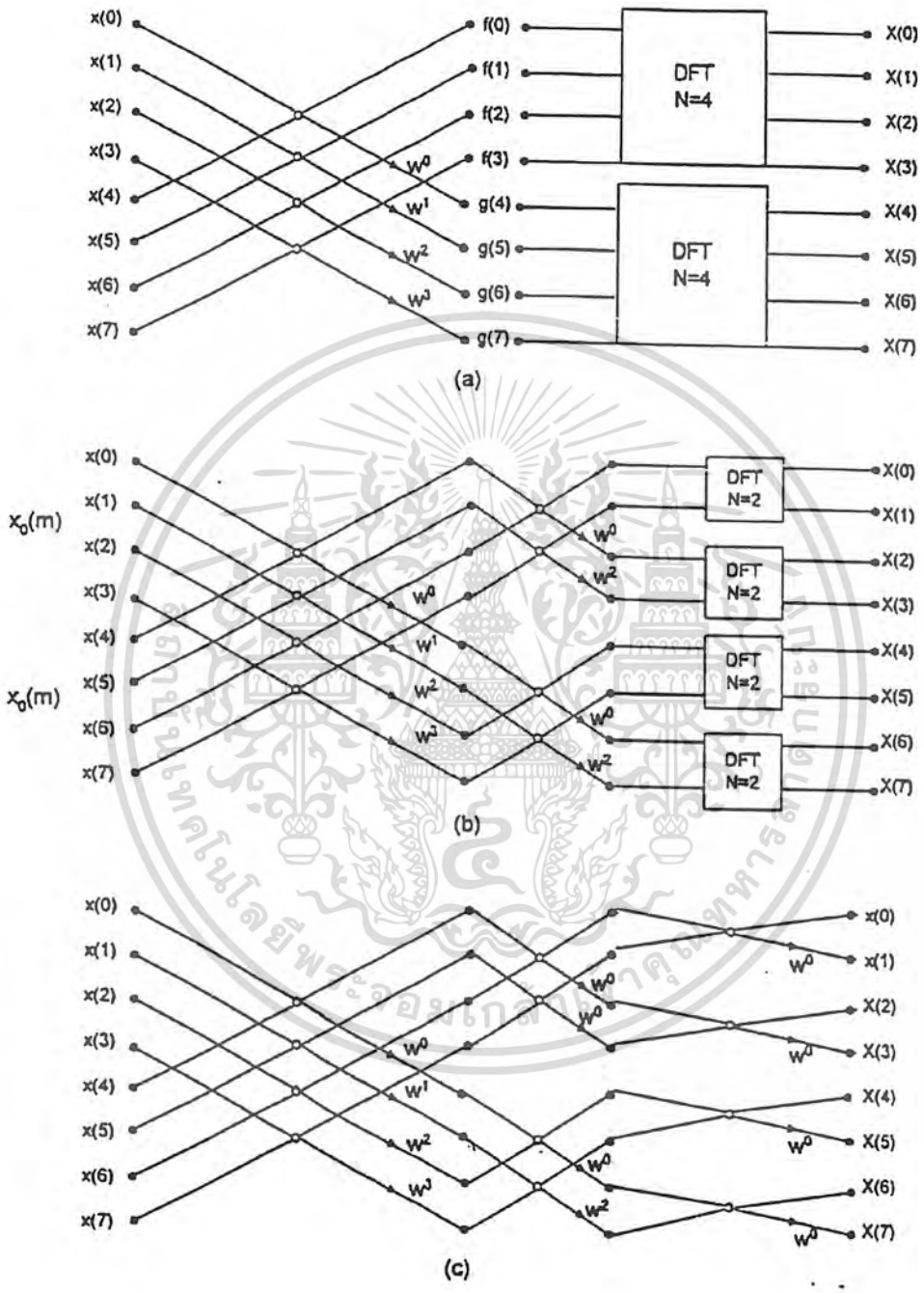
$$f(m) = x_E(m) + x_0(m) \qquad ; m = 0,1.....,(M/2)-1$$

$$g(m) = [x_E(m)-x_0(m)].(W_N)^{mk} \qquad (3.44)$$

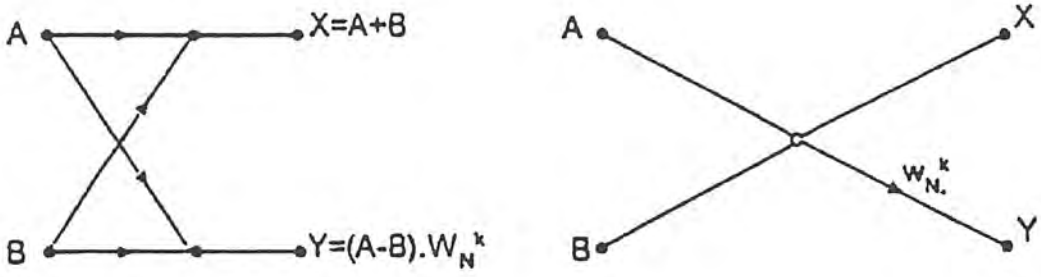
และจากลำดับยาว N/2 จุดในสมการ (3.44) เราก็อาจนำไปคำนวณหา DFT ขนาด N/2 จุด โดยใช้ (3.42) และ (3.43) เพราะฉะนั้น โดยรวมแล้วจะเห็นว่า การคำนวณ DFT ขนาด N จุดได้ถูกแบ่งเป็นการคำนวณ DFT ขนาด N/2 จุดสองภาคด้วยกัน

จะเห็นว่าเหมือนกับในการลดทอนทางเวลา การแบ่งการคำนวณย่อยออกเป็น DFT ขนาด N/2 จุดนี้สามารถแบ่งย่อยออกไปได้เรื่อยๆ จนในที่สุดเหลือการคำนวณขนาด 2 จุด ตัวอย่างที่แสดงการคำนวณโดยใช้กราฟการไหลกรณี N=8 แสดงไว้ในรูป 3.7 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 ส่วนการลดทอนทางความถี่จะตรงกันข้ามคือ x(n) จะเรียงตามธรรมชาติ และ X(k) จะถูกเรียงสลับแบบผันกลับบิท ประการที่สองหน่วยผีเสื้อของการลดทอนทางความถี่ต่างไปจากการลดทอนทางเวลาคือได้จากการเอาลำดับ $x_E(m)$ และ $x_0(m)$ มาบวกและลบกันก่อนแล้วจึงทำการคูณด้วยจำนวนเชิงซ้อน $(W_N)^k$ หน่วยคำนวณผีเสื้อของการลดทอนทางความถี่แสดงดังรูป 3.8



รูปที่ 3.7 แสดงลำดับขั้นวิธีการของ FFT ชนิดการลดทอนทางความถี่



รูปที่ 3.8 แสดงการคำนวณของหน่วยผีเสื้อ ของขั้นตอนวิธีชนิดลดทอนทางความถี่

บทที่ 4
สถาปัตยกรรมของ TMS320C31

ไอซีเบอร์ TMS320C31 ถูกสร้างและออกแบบมาเพื่อตอบสนองความต้องการทางด้านการประมวลผลบนพื้นฐานของคณิตศาสตร์ชั้นสูงและการแก้ปัญหาทาง hard ware และ soft ware

ประสิทธิภาพอันสูงของ TMS320C31 เกิดจากความไม่แน่นอนของความกว้างทางไดนามิกของส่วนคำนวณทางทศนิยม มีหน่วยความจำขนาดใหญ่ การทำงานแบบขนานและมี DMA Controller เพื่อการทำงานที่มีประสิทธิภาพมากยิ่งขึ้น

สถาปัตยกรรม TMS320C31 ประกอบด้วยส่วนประกอบที่สำคัญดังนี้

4.1 หน่วยประมวลผลกลาง (CPU)

หน่วยประมวลผลกลางจะประกอบไปด้วย

4.1.1 ตัวคูณ (Multiplier)

มีหน้าที่การคูณ และสามารถคูณจำนวนเต็ม 24 บิต(bit) และ จำนวนทศนิยม 32 บิต โดยใช้เวลาเพียง 1 รอบ ซึ่งการคำนวณจำนวนทศนิยมจะใช้เวลา 50 ns ต่อรอบ และสามารถประมวลผลแบบขนานได้ เพื่อที่จะเพิ่มความสามารถในการประมวลผลข้อมูล โดยใช้คำสั่งแบบขนานให้ทำการคูณและการทำงานของ หน่วยการคำนวณทางคณิตศาสตร์ (ALU) ภายใน 1 รอบ

เมื่อ ตัวคูณ ทำการทศนิยม อินพุทที่ใช้มีขนาด 32 บิต และผลลัพธ์ที่ได้จะมีขนาด 40 บิต แต่ถ้าตัวคูณ ทำการคูณจำนวนเต็มอินพุทจะมีขนาดเพียง 24 บิต และจะได้ผลลัพท์ขนาด 32 บิต

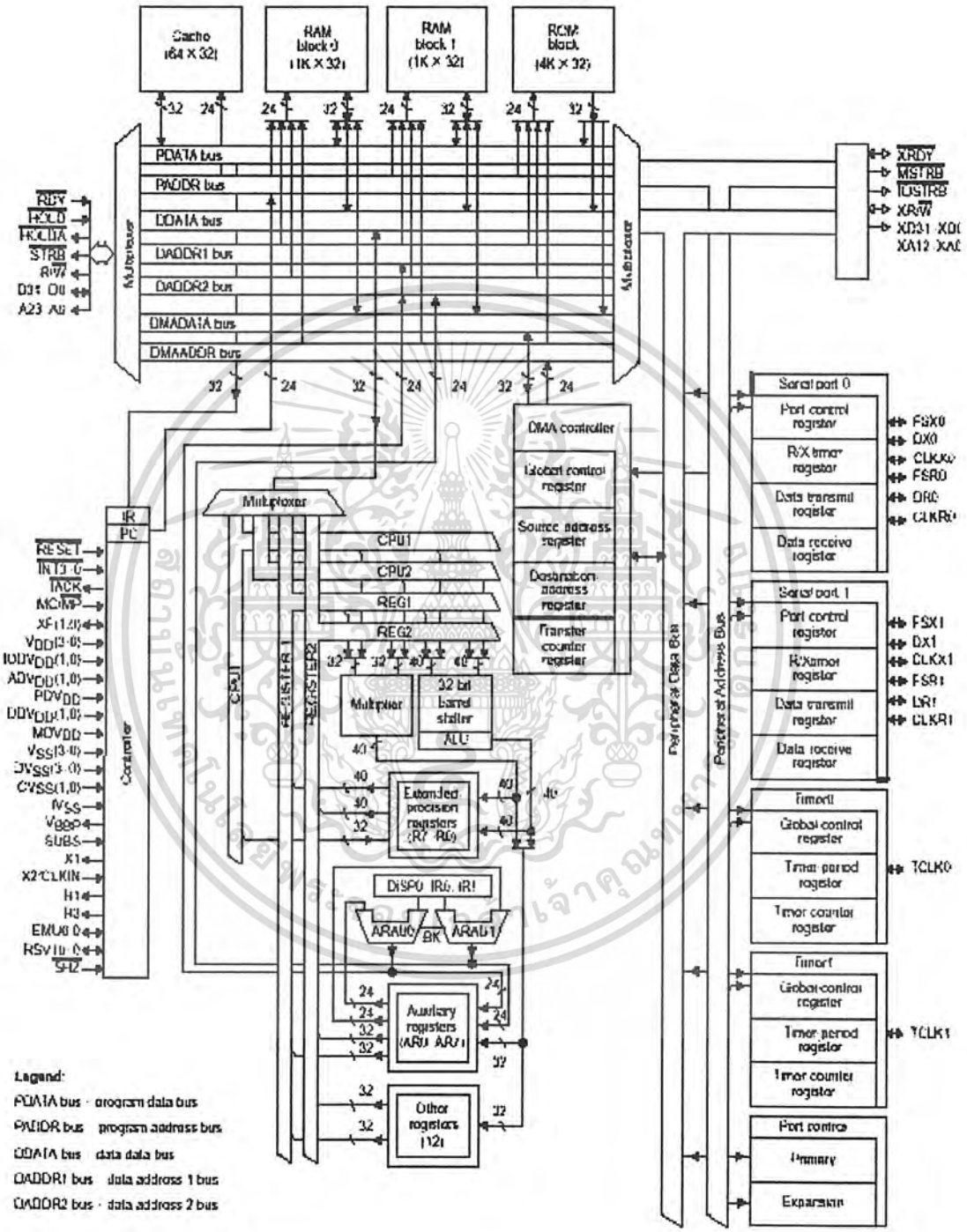
4.1.2 หน่วยการคำนวณทางคณิตศาสตร์ (Arithmetic Logic Unit (ALU))

ใน 1 รอบการทำงานของหน่วยการคำนวณทางคณิตศาสตร์ จะประมวลผลจำนวนเต็มได้ 32 บิต และ ทศนิยมได้ 40 บิต โดยผลลัพธ์จากหน่วยการคำนวณทางคณิตศาสตร์ ยังคงมีขนาด 32 บิต ถ้าเป็นจำนวนเต็ม และ 40 บิต ถ้าเป็นทศนิยม barrel shifter จะมีหน้าที่ในการเลื่อนข้อมูล 32 บิต ทั้งซ้ายและขวาใน 1 รอบของการทำงาน

บัสภายใน (CPU1/CPU2 และ REG1/REG2) จะเป็นตัวพาข้อมูลที่จำนำมาประมวลผล 2 ตัวจากหน่วยความจำ และอีก 2 ตัวจากรีจิสเตอร์ ดังนั้นจึงสามารถทำการคูณบวกและลบ แบบขนานได้ใน 1 รอบการทำงาน

4.1.3 หน่วยช่วยคำนวณทางคณิตศาสตร์ (Auxiliary Register Arithmetic Unit (ARAUs))

หน่วยช่วยคำนวณทางคณิตศาสตร์ เป็นรีจิสเตอร์ที่ช่วยการทำงานแบบขนานกับตัวคูณ และ หน่วยคำนวณทางคณิตศาสตร์ โดยหน่วยช่วยคำนวณทางคณิตศาสตร์ จะมี 2 ตัว คือ ARAU0 และ ARAU1 หน่วยช่วยคำนวณทางคณิตศาสตร์จะสามารถเก็บแอดเดรสได้ 2 แอดเดรสใน 1 รอบการทำงาน เพื่อช่วยในการอ้างตำแหน่งแบบต่างๆ



รูปที่ 4.1 แสดงสถาปัตยกรรมของไอซีเบอร์ TMS320C3X

4.1.4 CPU Register File

ใน TMS320C31 จะรีจิสเตอร์อยู่ 28 ตัว ซึ่งตรงกับ CPU โดยรีจิสเตอร์ทั้งหลายเหล่านี้จะทำงานกับ ตัวคูณ และ หน่วยคำนวณทางคณิตศาสตร์ และยังสามารถใช้เพื่อวัตถุประสงค์อื่นอีกด้วยอย่างไรก็ตามรีจิสเตอร์เหล่านี้ยังมีหน้าที่พิเศษบางอย่างอีกด้วย เช่น ใช้สำหรับการทำงานเกี่ยวกับจำนวนเลขทศนิยม และรีจิสเตอร์ช่วย (Auxiliary Register) ทั้ง 8 ตัวยังใช้การอ้างแอดเดรสแบบทางอ้อม (indirect addressing register) ด้วยรีจิสเตอร์ที่เหลือจะทำหน้าที่เกี่ยวกับระบบต่างๆเช่น การอ้างแอดเดรส การจัดการเกี่ยวกับสแตก (stack) การบอกสถานะของโปรเซสเซอร์ การอินเตอร์รัพท์และการเคลื่อนย้ายข้อมูลเป็นบล็อก

- extend – preision register (R0-R1): สามารถใช้เก็บจำนวนเต็ม 32 บิตแบบทศนิยม 40 บิต โดยจะเก็บลงในบิตที่ 0-39 แต่ถ้าจำนวนเต็มแบบมีเครื่องหมายก็จะทำการเก็บบิตที่ 0-31 ส่วนบิตที่ 32-39 จะไม่ใช้

- auxiliary register (AR0 – AR7) : จะถูกใช้โดยหน่วยประมวลผลหลักและถูกควบคุมโดยหน่วยช่วยคำนวณทางคณิตศาสตร์ทั้ง 2 ตัว หน้าที่แรกของ AR ยังใช้ให้ส่วนที่เก็บตัวนับในการวนลูปหรือจะใช้เพื่องานอื่นๆ ที่เกี่ยวกับตัวคูณ และ หน่วยคำนวณทางคณิตศาสตร์

- data page pointer (DP) : ส่วนรีจิสเตอร์ขนาด 32 บิต โดยที่ 8 บิตแรกจะถูกใช้ในการอ้างแอดเดรสแบบโดยตรง (direct addressing mode) ซึ่งใช้เป็นตัวชี้ (Pointer) ไปยังหน้า(page) ต่างๆของข้อมูล

- index register (IR0,IR1) : จะถูกใช้โดยหน่วยช่วยคำนวณทางคณิตศาสตร์ในการเก็บค่าดัชนีในการชี้แอดเดรส (indexed addressing mode)

- block size register (BK) : มีขนาด 32 บิตซึ่ง หน่วยช่วยคำนวณทางคณิตศาสตร์จะใช้ในการอ้างแอดเดรสแบบเก็บ circular addressing เพื่อเป็นการเก็บค่าของขนาดของบล็อกข้อมูล

- system stack pointer (SP) : เป็นรีจิสเตอร์ขนาด 32 บิตซึ่งเป็นค่าแอดเดรสของยอดของแสตก โดยแสตก (SP) จะชี้ไปยังค่าสุดท้ายที่เก็บอยู่บนสแตก การ push จะกระทำก่อนที่จะเพิ่มค่าแสตก และการ pop หลังการลดค่าของสแตก คำสั่งที่จะทำงานกับแสตกได้คือ interrupts, traps , returns,push และ pop

- status register (ST) : เป็นรีจิสเตอร์ที่เก็บค่าข้อมูลต่างๆที่แสดงสถานะของ CPU เช่นค่าของ flag ซึ่งจะแสดงสถานะของผลลัพธ์ที่ได้จากการคำนวณทางลอจิกด้วย

- CPU/DMA interrupt enable register (IE) : เป็นรีจิสเตอร์ขนาด 32 บิต ใช้สำหรับกำหนดขนาด enable interrupt ของ CPU โดยการเอ็นนาเบิ้ลของ CPU จะใช้บิตที่ 0-10 และการเอ็นนาเบิ้ลของ DMA จะใช้บิตที่ 16-26 ค่า “1” จะเป็นการเอ็นนาเบิ้ลและ “0” จะเป็นการดีสเอเบิ้ล

- I/O Flag register (IOF) : เป็นรีจิสเตอร์ในการควบคุมขา XF0 และ XF1

- repeat counter (RC) : เป็นรีจิสเตอร์ขนาด 32 บิต ใช้เก็บค่าของจำนวนเท่าหรือจำนวนครั้งของบล็อคข้อมูลซึ่งถูกกระทำซ้ำ เมื่อโปรเซสเซอร์ทำงานในโหมดของการกระทำข้อมูลซ้ำ (repeat mode)

- repeat start address register (RS) : รีจิสเตอร์ขนาด 32 บิต จะถูกใช้ในการเก็บแอดเดรสเริ่มต้นของบล็อคข้อมูลที่จะถูกกระทำซ้ำ

- repeat end address register (RE) : ซึ่งเป็นรีจิสเตอร์ขนาด 32 บิต จะถูกเก็บแอดเดรสสุดท้ายของบล็อคข้อมูลซึ่งถูกกระทำซ้ำ

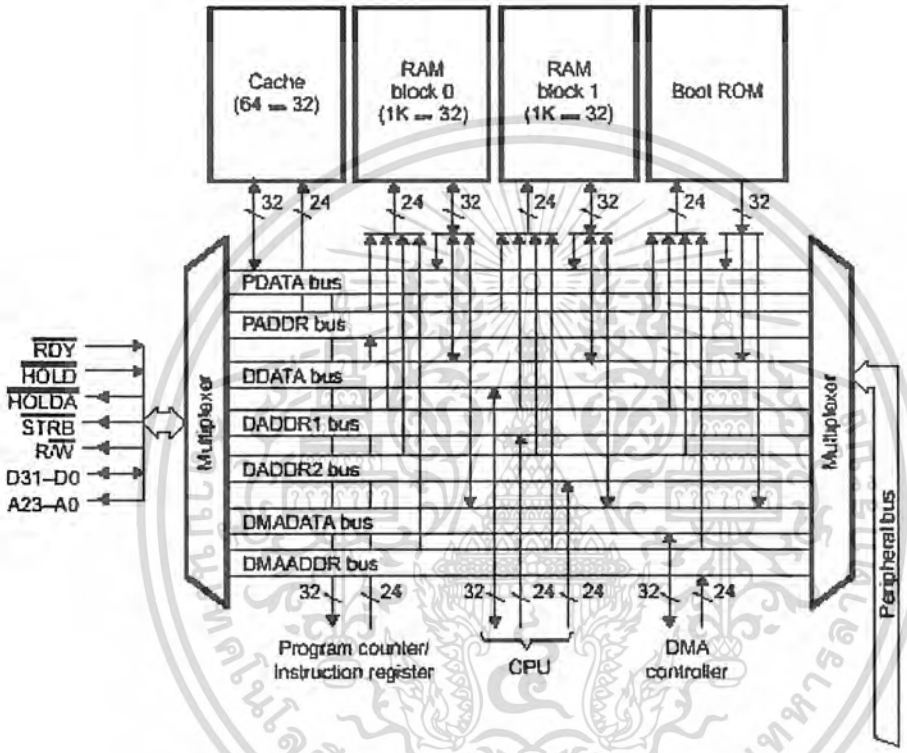
- program counter (PC) : เป็นรีจิสเตอร์ขนาด 32 บิต ซึ่งเก็บแอดเดรสคำสั่งที่จะถูกนำมาอ่านคำสั่งถัดไป

Register Symbol	Register Machine Value (hex)	Assigned Function Name
R0	00	Extended-precision register 0
R1	01	Extended-precision register 1
R2	02	Extended-precision register 2
R3	03	Extended-precision register 3
R4	04	Extended-precision register 4
R5	05	Extended-precision register 5
R6	06	Extended-precision register 6
R7	07	Extended-precision register 7
AR0	08	Auxiliary register 0
AR1	09	Auxiliary register 1
AR2	0A	Auxiliary register 2
AR3	0B	Auxiliary register 3
AR4	0C	Auxiliary register 4
AR5	0D	Auxiliary register 5
AR6	0E	Auxiliary register 6
AR7	0F	Auxiliary register 7
DP	10	Data-page pointer
IR0	11	Index register 0
IR1	12	Index register 1
BK	13	Block-size register
SP	14	System-stack pointer
ST	15	Status register
IE	16	CPU/DMA interrupt-enable
IF	17	CPU interrupt flags
IOF	18	I/O flags
RS	19	Repeat start-address
RE	1A	Repeat end-address
RC	1B	Repeat counter

ตารางที่ 4.1 ชื่อและหน้าที่ต่างๆของรีจิสเตอร์ใน CPU

4.2 หน่วยความจำ (Memory Organization)

หน่วยความจำของ TMS320C31 จะมีขนาด 16M และมีขนาดของ 1 word เท่ากับ 32 บิต โดยหน่วยความจำนี้จะใช้กับโปรแกรมข้อมูล และการอินพุท เอาท์พุท ดังนั้นค่าของสัมประสิทธิ์โปรแกรมหรือข้อมูลจะถูกเก็บได้ทั้ง RAM และ ROM



รูปที่ 4.2 แสดงหน่วยความจำของไอซีเบอร์ TMS320C31

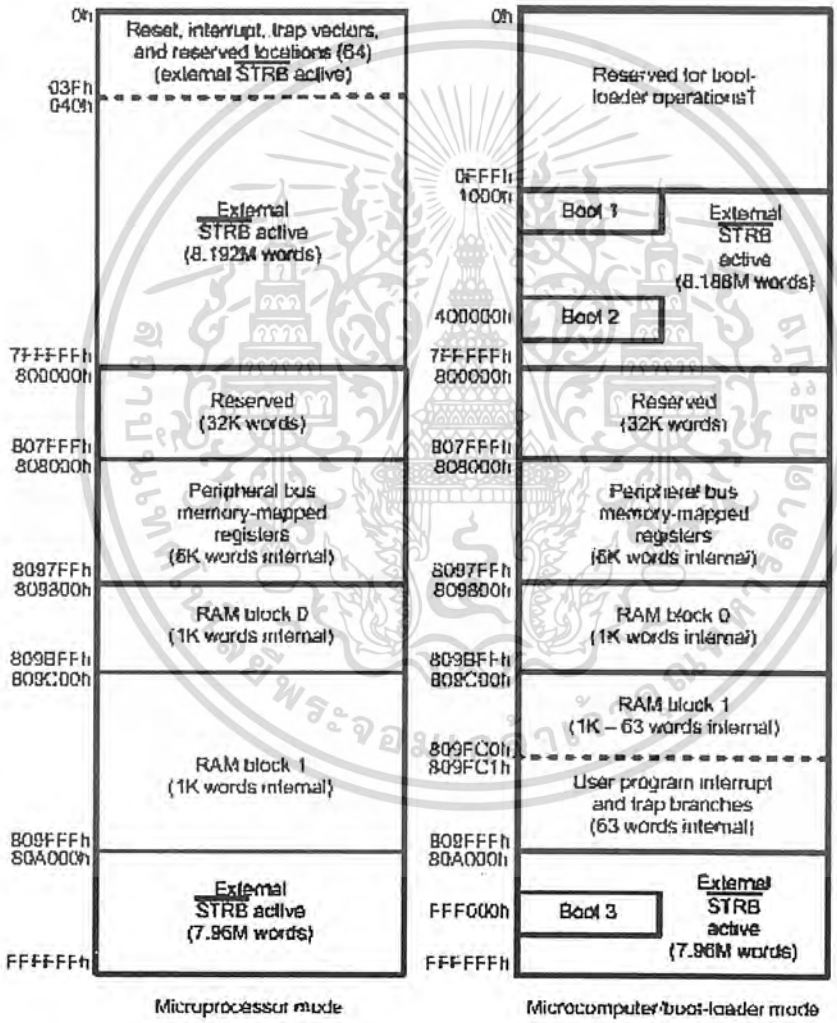
4.2.1 RAM,ROM

รูปที่ 4.2 จะแสดงการจัดหน่วยความจำภายใน TMS320C31 โดยแรมจะแบ่งออกเป็นบล็อก 0 และบล็อก 1 ซึ่งจะมีบล็อคละ 1K x 32 และ ROM จะมีขนาด 4K x 32 ทั้ง RAM และ ROM สามารถถูกเข้าใช้ได้โดย CPU2 ครั้งใน 1 รอบของการทำงานได้ ดังนั้น การที่มีบัสของโปรแกรมแยกกันและบัสของ DMA แยกกัน ทำให้การอ่านโปรแกรม การอ่านและการเขียนข้อมูลหรือ การทำงานของ DMA ที่สามารถทำงานแบบขนานได้

4.2.2 ตารางหน่วยความจำ(Memory Map)

ตารางหน่วยความจำจะขึ้นอยู่ว่าให้จะให้โปรเซสเซอร์ประมวลผลในโหมดของโปรเซสเซอร์ หรือ ไมโครคอมพิวเตอร์ แต่ในโครงงานนี้จะให้โปรเซสเซอร์ประมวลผลในโหมดของไมโครโปรเซสเซอร์ แสดงดังรูปที่ 4.3 แอดเดรสที่ 800000h จนถึง 801FFFh จะถูกใช้สำหรับ expansion bus

ซึ่งถูกเข้าถึงโดยการให้สัญญาณ /MSTRB แอดทีฟแอดเดรสที่ 80200h จนถึง 803FFFh จะไม่ใช้แอดเดรสที่ 804000h ถึง 805FFFh จะถูกใช้สำหรับ expansion bus ซึ่งจะถูกทำให้เข้าถึงโดยการให้สัญญาณ /IOSTRB แอดทีฟแอดเดรส 806000h ถึง 807FFFh จะไม่ใช้ทุกๆรีจิสเตอร์จะอยู่ที่ แอดเดรส 808000h ถึง 8097FFh RAM บล็อก 0 จะอยู่ที่ 809800h ถึง 809BFFh และบล็อคที่ 1 จะอยู่ที่ 809C00h ถึง 809FFFh แอดเดรส 80A000h จนถึง 0FFFFFFh จะถูกใช้โดยอุปกรณ์ภายนอก (เมื่อสัญญาณ /STRB แอดทีฟ)



รูปที่ 4.3 แสดงการจัดหน่วยความจำของไอซี TMS320C31

4.2.3 โหมดการอ้างอิงหน่วยความจำ (Memory Address Modes)

TMS32C31 จะแบ่งโหมดของการอ้างแอดเดรสออกเป็น 5 กลุ่ม โดยมีโหมดของการอ้างแอดเดรส 6 แบบ ดังนี้

1.General addressing mode

- Register : ค่าที่นำมาคำนวณจะเก็บที่ CPU รีจิสเตอร์
- Short immediate : ค่าที่นำมาคำนวณจะเป็นค่าที่โหลดโดยตรงขนาด 16 บิตจากคำสั่ง
- Direct : ค่าที่นำมาคำนวณจะถูกเก็บอยู่ในตำแหน่ง ซึ่งมีแอดเดรสขนาด 24 บิตแสดงอยู่ในคำสั่ง
- Indirect: รีจิสเตอร์จะเก็บแอดเดรสของข้อมูล

2. Three-operand addressing mode:

- Register
- Indirect

3. Parallel addressing mode :

- Register : ข้อมูลจะอยู่ใน extended – precision register
- Indirect

4. Long – immediate addressing mode :

- Long immediate : ข้อมูลจะเป็นค่าที่โหลดโดยตรงขนาด 24 บิต จากคำสั่ง

5. Conditional branch addressing mode:

- Register
- PC – relative : ค่าที่มีเครื่องหมายขนาด 16 บิต จะถูกบวกค่าใน PC

4.3 Internal Bus Operation

ข้อดีที่สำคัญอีกประการหนึ่งของ TMS320C31 คือการที่มีบัสภายในและสามารถประมวลผลแบบขนานได้ บัสโปรแกรมที่แยกกัน (PADDR และ PDATA) บัสข้อมูลที่แยกกัน (DADDR1,DADDR2 และ DDATA)และบัส DMA ที่แยกกัน (DMAADDR และ DMADATA) ทำให้เราสามารถทำการประมวลผลแบบขนานได้ บัสเหล่านี้สามารถต่อได้กับหน่วยความจำภายใน หน่วยความจำภายนอก หรือ อุปกรณ์ภายนอกอื่นๆ รูปที่ 4.2 จะแสดงการต่อบัสต่างๆเหล่านี้

PC จะถูกต่ออยู่กับ PADDR (ขนาด 24 บิต) IR จะต่ออยู่กับ PDATA (ขนาด 32 บิต) ซึ่งบัสเหล่านี้ สามารถอ่านคำสั่ง ทุกๆรอบของการทำงาน

DADDR1 และ DADDR2 (ขนาด 24 บิต) และ DDATA (ขนาด 32 บิต) จะใช้ติดต่อกับหน่วยความจำภายในทุกๆรอบของการทำงาน DDATA จะใช้ในการส่งข้อมูลไปยัง CPU โดยผ่าน บัส CPUI

และ CPU2 บัส CPU1 และ CPU2 สามารถส่งข้อมูลจากหน่วยความจำ 2 ข้อมูลไปยังตัวคูณ (Multiplier) หน่วยคำนวณทางคณิตศาสตร์ (ALU) และรีจิสเตอร์ในทุกๆรอบของการทำงาน นอกจากนี้ภายใน CPU มีบัสรีจิสเตอร์ REG1 และ REG2 ซึ่งใช้ในการหาข้อมูล 2 ค่าจากรีจิสเตอร์ไปคูณ และหน่วยการคำนวณทางคณิตศาสตร์ ในทุกๆรอบของการทำงาน รูปที่ 4.2 แสดงบัสภายในต่างๆ DMA Controller จะใช้ DMAADDR (ขนาด 24 บิต) และ DMADATA (ขนาด 32 บิต) โดยบัสเหล่านี้ทำให้ DMA สามารถติดต่อกับหน่วยความจำแบบขนานได้

4.4 External Bus Operation

หน่วยความจำและอุปกรณ์ภายนอกสามารถอินเตอร์เฟสกลับ TMS320C31 ได้ 2 ทางคือทาง Primary Bus และ Expansion Bus การอินเตอร์เฟสกับหน่วยความจำหรืออุปกรณ์ภายนอกที่มีความเร็วต่ำกว่า TMS320C31 จะต้องเพิ่ม Wait State เข้าไป โดยอาศัยการควบคุมของ Memory Mapped Control Register และสัญญาณจากภายนอก

Primary Bus มีขนาดของบัสข้อมูล (data bus) 32 บิต,บัสข้อมูล (Address bus) 34 บิต และสัญญาณควบคุม (control signal) อีก 1 ชุด โดยบัสทั้ง 2 สามารถถูกควบคุมได้โดยการใช้โปรแกรม (software) สั่งงานและสัญญาณควบคุมจากภายนอก

Peripheral Address	
808060h	Expansion-bus control ('C30 only)
808061h	Reserved
808062h	Reserved
808063h	Reserved
808064h	Primary-bus control ('C30, 'C31)
808065h	Reserved
808066h	Reserved
808067h	Reserved
808068h	Reserved
808069h	Reserved
80806Ah	Reserved
80806Bh	Reserved
80806Ch	Reserved
80806Dh	Reserved
80806Fh	Reserved

รูปที่ 4.4 memory – mapped external interface control register

อุปกรณ์ภายนอกสามารถเข้าถึงได้โดยการใช้สัญญาณ /STRB,/MSTRB และ /IOSTRB โดยสัญญาณ /STRB จะใช้เมื่อต้องการ primary bus ส่วน expansion bus จะสามารถใช้งานได้โดย 2 แบบ คือ

- การเข้าใช้ memory สามารถทำได้โดยให้ /MSTRB เป็น 0 โดย /MSTRB จะมีวงรอบการทำงานเหมือนกับ /STRB

- อุปกรณ์ภายนอกสามารถเข้าใช้ได้โดยให้สัญญาณ /IOSTRB เป็น 0

primary bus และ expansion bus จะมี control register ดังรูปที่ 4.5

31–16	15–13	12	11	10	9	8	7	6	5	4	3	2	1	0
xx	xx	BNKCMP					WTCNT			SWW		HIZ	NOHOLD	HOLDST
		R/W					R/W			R/W		R/W	R/W	R

Notes: 1) xx = reserved bit, read as 0
2) R = read, W = write

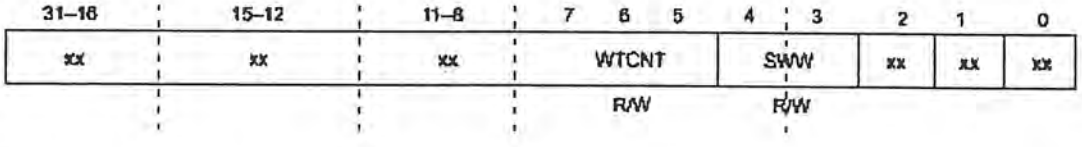
รูปที่ 4.5 primary bus control register

Abbreviation	Reset Value	Name	Description
HOLDST	0	Hold status bit	This bit signals whether the port is being held (HOLDST = 1) or is not being held (HOLDST = 0). This status bit is valid whether the port has been held through hardware or software.
NOHOLD	0	Port hold signal	NOHOLD allows or disallows the port to be held by an external HOLD signal. When NOHOLD = 1, the 'C3x takes over the external bus and controls it, regardless of serviced or pending requests by external devices. No hold acknowledge (HOLDA) is asserted when a HOLD signal is received. It is asserted if an internal hold is generated (HIZ = 1).
HIZ	0	Internal hold	When set (HIZ = 1), the port is put in hold mode. This is equivalent to the external HOLD signal. By forcing a high-impedance condition, the 'C3x can relinquish the external-memory port through software. HOLDA goes low when the port is placed in the high-impedance state.
SWW	11	Software wait mode	In conjunction with WTCNT, this 2-bit field defines the mode of wait-state generation (See Table 9-5.)
WTCNT	111	Software wait mode	This three-bit field specifies the number of cycles to use when in software wait mode for the generation of internal wait states. The range is 0 (WTCNT = 0 0 0) to 7 (WTCNT = 1 1 1) H1/H3 cycles (See Section 9.4.)
BNKCMP	11000	Bank compare	This 5-bit field specifies the number of MSBs of the address to be used to define the bank size. (See Table 9-6.)

ตารางที่ 4.2 แสดงหน้าที่ต่างๆของ primary bus control register

control register ของ primary bus จะมีขนาด 32 บิต ดังรูปที่ 4.5 และตารางที่ 4.2

control register ของ expansion bus จะมีขนาด 32 บิต ดังรูปที่ 4.6 และตารางที่ 4.3



Notes: 1) xx = reserved bit, read as 0
2) R = read, W = write

รูปที่ 4.6 expansion bus control register

Abbreviation	Reset Value	Name	Description
SWW	11	Software wait mode	In conjunction with the WTCNT, 2-bit field defines the mode of wait-state generation. (See Table 9-5.)
WTCNT	111	Software wait mode	This 3-bit field specifies the number of cycles to use when in software wait mode for the generation of internal wait state. The range is 0 (WTCNT = 0 0 0) to 7 (WTCNT = 1) H1/H3 cycles. (See Section 9.4.)

ตารางที่ 4.3 แสดงหน้าที่ต่างๆของ expansion bus control register

address จะเปลี่ยนที่ขอบขาลงของ H1 ทุกๆวงรอบของสัญญาณบนสายบัสจะขึ้นอยู่กับสัญญาณนาฬิกา H1 โดย 1 รอบของ H1 เริ่มนับจากขอบขาลงจนถึงขอบขาลงของลูกถัดไป สำหรับการทำงานแบบความเร็วเต็มที่ (คือไม่ใช้ wait state) ในการเขียนข้อมูลจะใช้เวลา 2 รอบของ H1 และการอ่านข้อมูลจะใช้ 1 รอบของ H1 แต่ถ้าการอ่านข้อมูลกระทำต่อคำสั่งการเขียนข้อมูล การอ่านข้อมูลจะใช้เวลา 2 รอบของ H1 ซึ่งหลักการนี้จะใช้ได้ทั้งกับ primary bus และ expansion bus (เมื่อใช้ /MSTRB)

ถ้าการเขียนข้อมูลเป็นการกระทำภายในโดยไม่มีการอินเตอร์เฟสกับอุปกรณ์ภายนอก (คือ การใช้จาก CPU และ DMA) วงรอบของการเขียนข้อมูลจะใช้เพียง 1 รอบของ H1 เท่านั้น

สัญญาณ /MSTRB และ /STRB จะแอคทีฟที่สถานะ 0 สำหรับการอ่านและการเขียนข้อมูลในขณะที่สภาวะก่อนของการแอคทีฟ /MSTRB และ /STRB ในการเขียนข้อมูลจะมี transion cycle ของ H1 ในระหว่าง transion cycle จะเกิด

1. /MSTRB และ /STRB จะเป็นสถานะ high
2. (X)R/W จะเปลี่ยนสถานะที่ขอบขาขึ้นของ H1
3. ถ้าเป็นสถานะของการขียนข้อมูล address จะเปลี่ยนสถานะที่ขอบขาขึ้นของ H1 และถ้าเป็นสถานะของการอ่านข้อมูล

4.5 คำสั่งของไอซีเบอร์ TMS320C31

คำสั่งของไอซีเบอร์ TMS320C31 เป็นคำสั่งของภาษาแอสแซมบลี้ ซึ่งเป็นคำสั่งเฉพาะของไอซีตระกูล TMS320C3x ซึ่งประกอบไปด้วยคำสั่งแบบธรรมดากับคำสั่งแบบขนาน ที่จะทำให้มีการประมวลผลที่เร็วขึ้น ซึ่งคำสั่งต่างๆจะแสดงดังตารางที่ 4.4

Mnemonic	Description	Operation
ABSF	Absolute value of a floating-point number	|*src*| → R*n*
ABSI	Absolute value of an integer	|*src*| → Dreg
ADDC	Add integers with carry	*src* + Dreg + C → Dreg
ADDC3	Add integers with carry (3-operand)	*src*1 + *src*2 + C → Dreg
ADDF	Add floating-point values	*src* + R*n* → R*n*
ADDF3	Add floating-point values (3-operand)	*src*1 + *src*2 → R*n*
ADDI	Add integers	*src* + Dreg → Dreg
ADDI3	Add integers (3 operand)	*src*1 + *src*2 + → Dreg
AND	Bitwise-logical AND	Dreg AND *src* → Dreg
AND3	Bitwise-logical AND (3-operand)	*src*1 AND *src*2 → Dreg
ANDN	Bitwise-logical AND with complement	Dreg AND $\overline{src}$ → Dreg
ANDN3	Bitwise-logical ANDN (3-operand)	*src*1 AND $\overline{src2}$ → Dreg
ASH	Arithmetic shift	If *count* ≥ 0: (Shifted Dreg left by *count*) → Dreg Else: (Shifted Dreg right by |*count*|) → Dreg
ASH3	Arithmetic shift (3-operand)	If *count* ≥ 0: (Shifted *src* left by *count*) → Dreg Else: (Shifted *src* right by |*count*|) → Dreg

Legend:				
	AR*n*	auxiliary register *n* (AR7–AR0)	RE	repeat interrupt register
	C	carry bit	RM	repeat mode bit
	C*src*	conditional-branch addressing modes	R*n*	register address (R7–R0)
	count	shift value (general addressing modes)	RS	repeat start register
	cond	condition code	SP	stack pointer
	Daddr	destination memory address	Sreg	register address (any register)
	Dreg	register address (any register)	ST	status register
	GIE	global interrupt enable register	*src*	general addressing modes
	N	any trap vector 0–27	*src*1	3-operand addressing modes
	PC	program counter	*src*2	3-operand addressing modes
	RC	repeat counter register	TOS	top of stack

ตารางที่ 4.4 แสดงคำสั่งของไอซี TMS320C31

Mnemonic	Description	Operation
B*cond*	Branch conditionally (standard)	If *cond* = true: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC → PC Else, PC + 1 → PC
B*cond*D	Branch conditionally (delayed)	If *cond* = true: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC + 3 → PC Else, PC + 1 → PC
BR	Branch unconditionally (standard)	Value → PC
BRD	Branch unconditionally (delayed)	Value → PC
CALL	Call subroutine	PC + 1 → TOS Value → PC
CALL*cond*	Call subroutine conditionally	If *cond* = true: PC + 1 → TOS If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC → PC Else, PC + 1 → PC
CMPF	Compare floating-point values	Set flags on R*n* – *src*
CMPF3	Compare floating-point values (3-operand)	Set flags on *src*1 – *src*2
CMPI	Compare integers	Set flags on Dreg – *src*
CMPI3	Compare integers (3-operand)	Set flags on *src*1 – *src*2

Legend:					
	AR*n*	auxiliary register *n* (AR7–AR0)	RE	repeat interrupt register
	C	carry bit	RM	repeat mode bit
	C*src*	conditional-branch addressing modes	R*n*	register address (R7–R0)
	count	shift value (general addressing modes)	RS	repeat start register
	cond	condition code	SP	stack pointer
	Daddr	destination memory address	Sreg	register address (any register)
	Dreg	register address (any register)	ST	status register
	GIE	global interrupt enable register	*src*	general addressing modes
	N	any trap vector 0–27	*src*1	3-operand addressing modes
	PC	program counter	*src*2	3-operand addressing modes
	RC	repeat counter register	TOS	top of stack

ตารางที่ 4.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
DB*cond*	Decrement and branch conditionally (standard)	AR*n* – 1 → AR*n* If *cond* = true and AR*n* ≥ 0: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC + 1 → PC Else, PC + 1 → PC
DB*cond*D	Decrement and branch conditionally (delayed)	AR*n* – 1 → AR*n* If *cond* = true and AR*n* ≥ 0: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC + 3 → PC Else, PC + 1 → PC
FIX	Convert floating-point value to integer	Fix (*src*) → Dreg
FLOAT	Convert integer to floating-point value	Float(*src*) → R*n*
IACK	Interrupt acknowledge	Dummy read of *src* IACK toggled low, then high
IDLE	Idle until interrupt	PC + 1 → PC Idle until next interrupt
IDLE2	Low-power idle	Idle until next interrupt stopping internal clocks
LDE	Load floating-point exponent	*src*(exponent) → R*n*(exponent)
LDF	Load floating-point value	*src* → R*n*
LDF*cond*	Load floating-point value conditionally	If *cond* = true, *src* → R*n* Else, R*n* is not changed
LDFI	Load floating-point value, interlocked	Signal interlocked operation *src* → R*n*
LDI	Load integer	*src* → Dreg

Legend:

ARn	auxiliary register *n* (AR7–AR0)	RE	repeat interrupt register
C	carry bit	RM	repeat mode bit
C*src*	conditional-branch addressing modes	R*n*	register address (R7–R0)
count	shift value (general addressing modes)	RS	repeat start register
cond	condition code	SP	stack pointer
Daddr	destination memory address	Sreg	register address (any register)
Dreg	register address (any register)	ST	status register
GIE	global interrupt enable register	*src*	general addressing modes
N	any trap vector 0–27	*src*1	3-operand addressing modes
PC	program counter	*src*2	3-operand addressing modes
RC	repeat counter register	TOS	top of stack

ตารางที่ 4.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
NOP	No operation	Modify AR*n* if specified
NORM	Normalize floating-point value	Normalize (*src*) → R*n*
NOT	Bitwise-logical complement	$\overline{src}$ → Dreg
OR	Bitwise-logical OR	Dreg OR *src* → Dreg
OR3	Bitwise-logical OR (3-operand)	*src*1 OR *src*2 → Dreg
POP	Pop integer from stack	*SP−− → Dreg
POPF	Pop floating-point value from stack	*SP−− → R*n*
PUSH	Push integer on stack	Sreg → *++ SP
PUSHF	Push floating-point value on stack	R*n* → *++ SP
RETI*cond*	Return from interrupt conditionally	If *cond* = true or missing: *SP−− → PC 1 → ST (GIE) Else, continue
RETS*cond*	Return from subroutine conditionally	If *cond* = true or missing: *SP−− → PC Else, continue
RND	Round floating-point value	Round (*src*) → R*n*
ROL	Rotate left	Dreg rotated left 1 bit → Dreg
ROLC	Rotate left through carry	Dreg rotated left 1 bit through carry → Dreg
ROR	Rotate right	Dreg rotated right 1 bit → Dreg
RORC	Rotate right through carry	Dreg rotated right 1 bit through carry → Dreg

Legend:	AR*n*	auxiliary register *n* (AR7–AR0)	RE	repeat interrupt register
	C	carry bit	RM	repeat mode bit
	C*src*	conditional-branch addressing modes	R*n*	register address (R7–R0)
	count	shift value (general addressing modes)	RS	repeat start register
	cond	condition code	SP	stack pointer
	Daddr	destination memory address	Sreg	register address (any register)
	Dreg	register address (any register)	ST	status register
	GIE	global interrupt enable register	*src*	general addressing modes
	N	any trap vector 0–27	*src*1	3-operand addressing modes
	PC	program counter	*src*2	3-operand addressing modes
	RC	repeat counter register	TOS	top of stack

ตารางที่ 4.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
RPTB	Repeat block of instructions	*src* → RE
		1 → ST (RM)
		Next PC → RS
RPTS	Repeat single instruction	*src* → RC
		1 → ST (RM)
		Next PC → RS
		Next PC → RE
SIGI	Signal, interlocked	Signal interlocked operation
		Wait for interlock acknowledge
		Clear interlock
STF	Store floating-point value	R*n* → Daddr
STFI	Store floating-point value, interlocked	R*n* → Daddr
		Signal end of interlocked operation
STI	Store integer	Sreg → Daddr
STII	Store integer, interlocked	Sreg → Daddr
		Signal end of interlocked operation
SUBB	Subtract integers with borrow	Dreg – *src* – C → Dreg
SUBB3	Subtract integers with borrow (3-operand)	*src*1 – *src*2 – C → Dreg
SUBC	Subtract integers conditionally	If Dreg – *src* ≥ 0:
		[(Dreg – *src*) << 1] OR 1 → Dreg
		Else, Dreg << 1 → Dreg
SUBF	Subtract floating-point values	R*n* – *src* → R*n*
SUBF3	Subtract floating-point values (3-operand)	*src*1 – *src*2 → R*n*

Legend:	AR*n*	auxiliary register *n* (AR7–AR0)		RE	repeat interrupt register
	C	carry bit		RM	repeat mode bit
	C*src*	conditional-branch addressing modes		R*n*	register address (R7–R0)
	count	shift value (general addressing modes)		RS	repeat start register
	cond	condition code		SP	stack pointer
	Daddr	destination memory address		Sreg	register address (any register)
	Dreg	register address (any register)		ST	status register
	GIE	global interrupt enable register		*src*	general addressing modes
	N	any trap vector 0–27		*src*1	3-operand addressing modes
	PC	program counter		*src*2	3-operand addressing modes
	RC	repeat counter register		TOS	top of stack

ตารางที่ 4.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
SUBI	Subtract integers	Dreg – *src* → Dreg
SUBI3	Subtract integers (3-operand)	*src*1 – *src*2 → Dreg
SUBRB	Subtract reverse integer with borrow	*src* – Dreg – C → Dreg
SUBRF	Subtract reverse floating-point value	*src* – R*n* → R*n*
SUBRI	Subtract reverse integer	*src* – Dreg → Dreg
SWI	Software interrupt	Perform emulator interrupt sequence
TRAP*cond*	Trap conditionally	If *cond* = true or missing: Next PC → *++SP Trap vector N → PC 0 → ST (GIE) Else, continue
TSTB	Test bit fields	Dreg AND *src*
TSTB3	Test bit fields (3-operand)	*src*1 AND *src*2
XOR	Bitwise-exclusive OR	Dreg XOR *src* → Dreg
XOR3	Bitwise-exclusive OR (3-operand)	*src*1 XOR *src*2 → Dreg

Legend:					
	AR*n*	auxiliary register *n* (AR7–AR0)	RE	repeat interrupt register
	C	carry bit	RM	repeat mode bit
	C*src*	conditional-branch addressing modes	R*n*	register address (R7–R0)
	count	shift value (general addressing modes)	RS	repeat start register
	cond	condition code	SP	stack pointer
	Daddr	destination memory address	*S*reg	register address (any register)
	Dreg	register address (any register)	ST	status register
	GIE	global interrupt enable register	*src*	general addressing modes
	N	any trap vector 0–27	*src*1	3-operand addressing modes
	PC	program counter	*src*2	3-operand addressing modes
	RC	repeat counter register	TOS	top of stack

ตารางที่ 4.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

บทที่5

การเชื่อมต่อกันระหว่างคอมพิวเตอร์และ TMS320C31

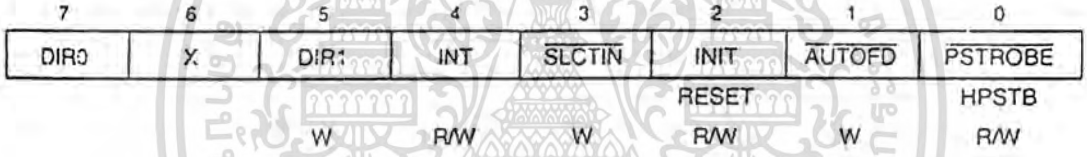
การทำงานของโปรแกรมนั้นจะต้องมีการทำงานร่วมกันระหว่างโปรแกรม 2 ส่วน คือ ส่วนรับค่าสัญญาณแล้วมาคำนวณ และ อีกส่วนหนึ่งก็คือส่วนที่จะทำการแสดงผล

ซึ่งจะประกอบไปด้วย

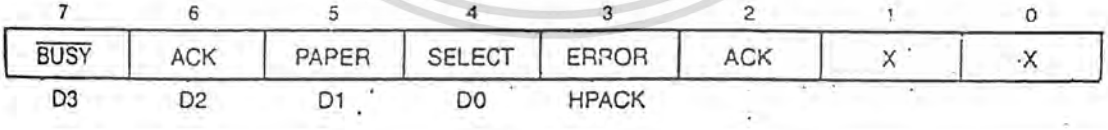
1. โปรแกรมในส่วนของ TMS320C31 ซึ่งจะทำหน้าที่ รับค่าจากสัญญาณและคำนวณค่า

2. โปรแกรมในส่วนของคอมพิวเตอร์ซึ่งจะทำหน้าที่รับค่าจาก TMS320C31 มาทำการแสดงผล

ซึ่งทั้งสองส่วนนี้จะต้องทำการส่งข้อมูลระหว่างกันเกือบตลอดเวลา การเชื่อมต่อกันระหว่าง 2 ตัวนี้จะเชื่อมต่อกันโดยใช้พอร์ทขนานของ TMS320C31 โดยจะเขียนหรืออ่านข้อมูลจากรีจิสเตอร์ควบคุมพอร์ทขนานและรีจิสเตอร์สถานะของคอมพิวเตอร์ซึ่งจะควบคุมการทำงานด้วยโปรแกรมบนคอมพิวเตอร์



รูปที่ 5.1 แสดงบิทของรีจิสเตอร์ควบคุมพอร์ทขนาน



รูปที่ 5.2 แสดงบิทของรีจิสเตอร์สถานะของพอร์ตขนาน

การส่งข้อมูลจากคอมพิวเตอร์ไป TMS320C31 สามารถใช้งานได้หลายแบบดังนี้

1. คอมพิวเตอร์เขียนข้อมูลลงในพื้นที่ในการอินพุทและเอาท์พุทของพอร์ทขนาน

2.คอมพิวเตอร์ให้สัญญาณ HPSTB เป็น 0 และรอสัญญาณเพื่ออนุญาตให้มีการอินเตอร์รัพโดยจะส่งมาทาง INT2 ซึ่งสัญญาณนี้จะใช้สำหรับการกำหนดค่าเริ่มต้นของระบบ นอกนั้นการทำงานของสัญญาณ INT2 จะไม่นำมาใช้ในการทำงานอื่นๆ

3.TMS320C31 เริ่มมีสภาวะการคอยที่ตำแหน่ง 0 x FFF000 ซึ่งจะสามารถถอดรหัสได้เป็นตำแหน่ง ของ HPACK ซึ่ง คอมพิวเตอร์จะใช้ สัญญาณ HPACK เพื่อที่ทำให้ TMS320C31 ทำการหยุดรอรับข้อมูล

4. คอมพิวเตอร์จะทำการส่ง HPSTB เพื่อบอกให้ TMS320C31 พร้อมที่จะเริ่มทำงานได้

การส่งข้อมูลจาก TMS320C31 ไปยัง คอมพิวเตอร์ สามารถใช้งานได้หลายแบบดังนี้

1. คอมพิวเตอร์จะทำการรอรับค่า สัญญาณ HPACK เนื่องจาก TMS320C31 ไม่เข้าใจการสั่งการทำงานของคอมพิวเตอร์

2. TMS320C31 เริ่มมีสภาวะการคอยที่ตำแหน่ง 0xFFF000 ซึ่งจะสามารถถอดรหัสได้เป็นตำแหน่ง ของ HPACK ซึ่ง คอมพิวเตอร์จะใช้ สัญญาณ HPACK เพื่อที่ทำให้ TMS320C31 ทำการพร้อมที่จะส่งข้อมูล

3. ขณะที่คอมพิวเตอร์รับสัญญาณ HPIA คอมพิวเตอร์จะให้สัญญาณ PSTROBE เป็น 0 และจะอ่านค่า 4 บิท ซึ่งจะผ่านทางพอร์ทขนาน

4. คอมพิวเตอร์จะทำการส่ง HPSTB เพื่อบอกว่าได้อ่านค่าจาก TMS320C31 เสร็จสิ้นการทำงานแล้ว

5.1 ขั้นตอนในการทำงานของโปรแกรม TMS320C31

เริ่มต้นจะกำหนดค่าบนเมมโมรี่ บน DSK และเซตค่าต่างๆเพื่อทำการรับค่า หลังจากนั้นจะทำการรับค่าและเช็คว่าจะเริ่มรับค่าที่จุดใดและจะเริ่มรับค่าไปเท่ากับจำนวนที่กำหนดไว้ แล้วนำค่าที่ได้ไปทำการคำนวณค่าตามทฤษฎี FFT แล้วนำค่าที่ได้ไปคำนวณหาขนาดและมุมเฟสของสเปกตรัมและนำไปจัดลำดับการเก็บค่าใหม่ รอจนกว่า COMPUTER จะรับค่าจาก DSK ทั้งหมดก่อนแล้วเช็คค่าว่า COMPUTER สั่งให้มีการเปลี่ยนแปลงค่าหรือไม่ถ้ามีการเปลี่ยนแปลงค่าให้กับเมมโมรี่ แต่ถ้าไม่มีก็ให้รับค่าต่อไปได้เลยโดยรูปที่ 5.1 จะแสดงขั้นตอนการทำงานของโปรแกรมทั้งหมดของ TMS320C31

5.2 ขั้นตอนในการทำงานของโปรแกรมบน คอมพิวเตอร์

เริ่มต้นจะเช็คว่ามีโปรแกรมที่ต้องการหรือไม่แล้วจึง load โปรแกรมลงไปยัง DSK แล้วทำการสั่งให้ ตัว DSK ทำงาน แล้วจะทำการวาดกราฟเพื่อรอค่าจาก DSK หลังจากนั้นจะทำการรับค่า KEY ที่กดว่ามีการกดอะไรบ้าง และถ้ามีการกด Q ก็จะออกจากโปรแกรมทันที แต่ถ้าไม่ก็จะทำงานต่อ โดยจะไปตรวจสอบว่า DSK ทำงานเสร็จหรือยังเมื่อเสร็จแล้วก็จะทำการรับค่าจาก DSK แล้วนำค่าที่ได้มา เขียนกราฟขนาด และมุมเฟสของสเปกตรัมแล้วจึงวนกลับไปรอรับค่าจาก DSK ต่อไป

บทที่ 6
ผลการทดลอง

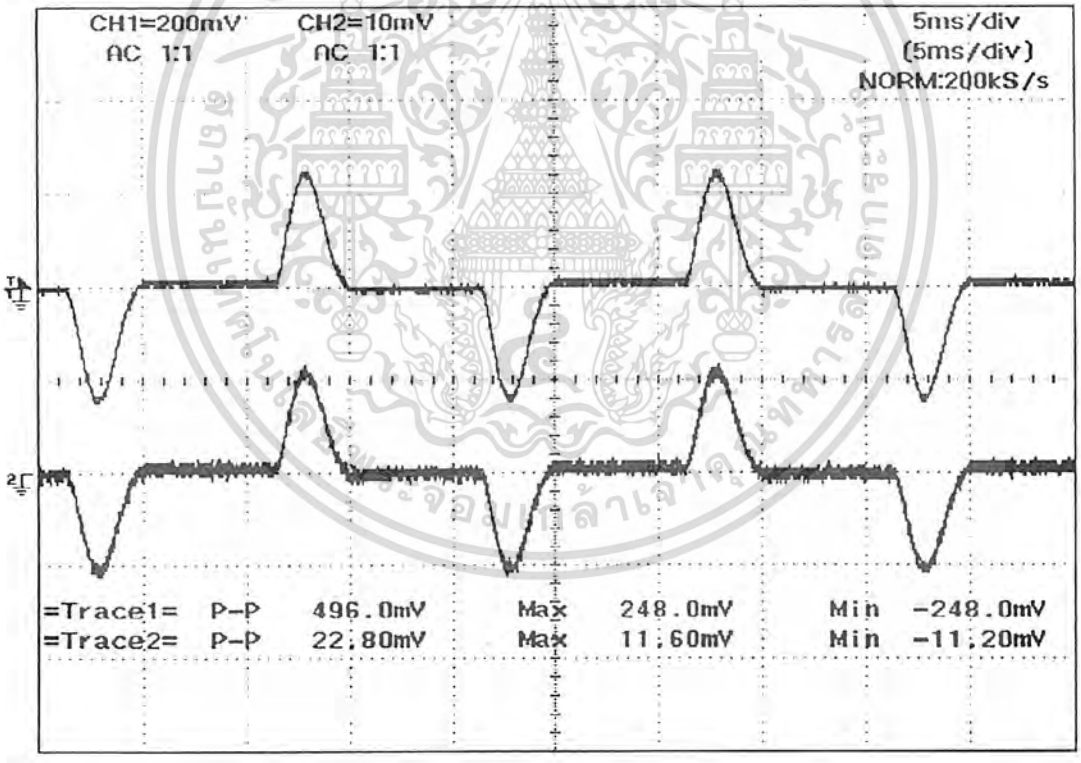
ผลการทดลองของ ACTIVE POWER FILTER แบ่งออกเป็น 2 ส่วนด้วยกันได้แก่

1. ส่วนตรวจจับกระแส
2. ส่วนการวิเคราะห์รูปคลื่นกระแส

6.1 ส่วนตรวจจับกระแส

ผลการทดลองจะศึกษาเรื่องของการพิจารณารูปคลื่นสัญญาณของกระแสที่โหลดดึงไปใช้งานว่ามีรูปร่างอย่างไร

6.1.1 ทดสอบป้อนกระแสผ่าน Current sensor พิจารณารูปสัญญาณของกระแส



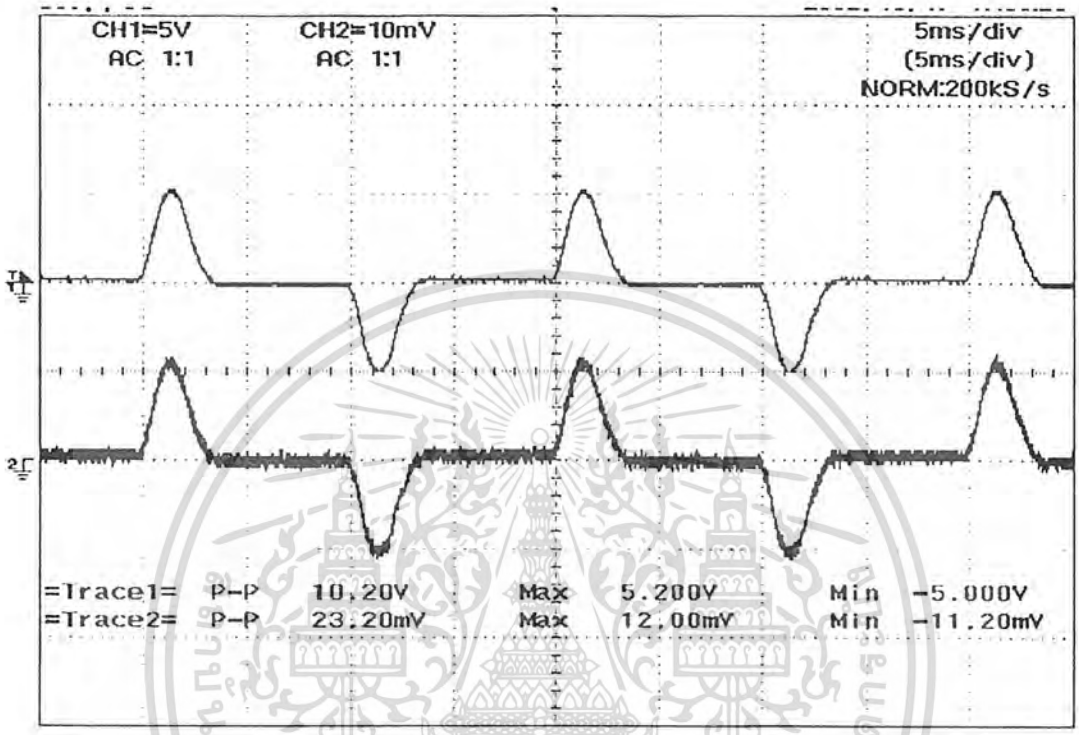
รูปที่ 6.1สัญญาณ Current sensor กับ Current probe

Ch1:สัญญาณกระแสที่ได้จาก Current sensor

Ch2:สัญญาณกระแสที่ได้จาก Current Probe

จากรูปแสดงสัญญาณกระแสที่ได้จาก Current sensor เทียบกับสัญญาณกระแสที่ได้จาก Current Probe

6.1.2 ทดสอบป้อนกระแสผ่านCurrent sensor โดยต่อผ่านวงจร Noninverting Amplifier พิจารณารูปสัญญาณของกระแส



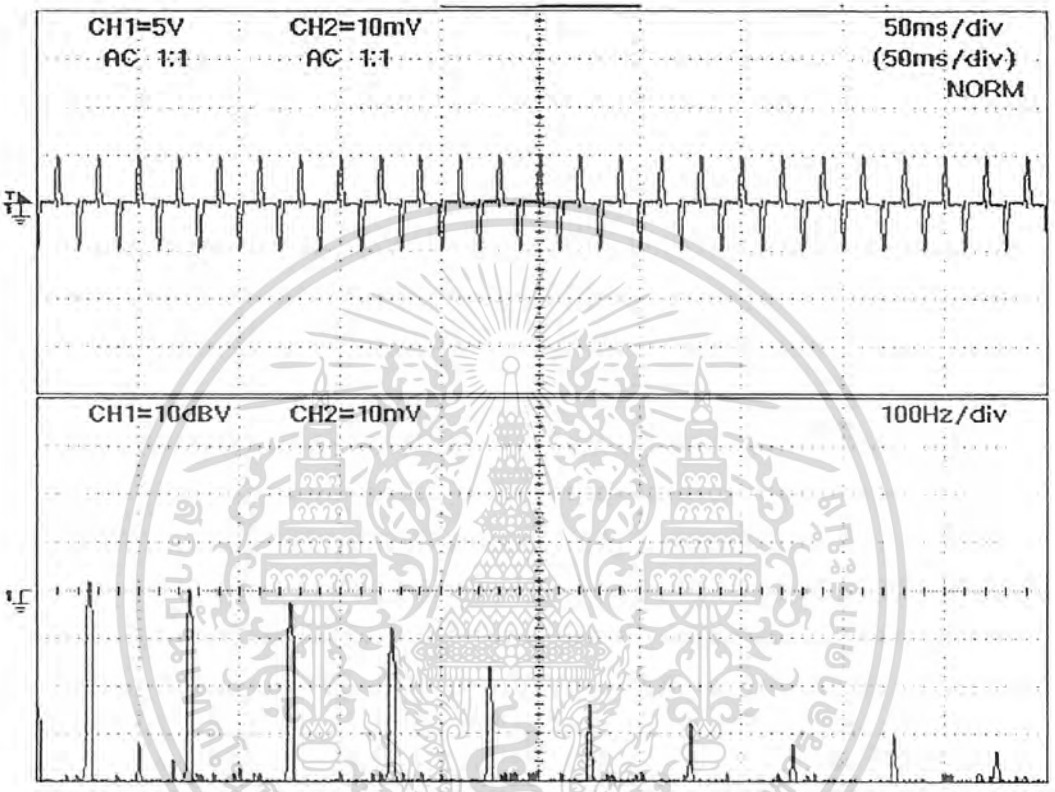
รูปที่6.2 สัญญาณ Current sensor กับ Current probe ผ่านวงจร noninvertting

Ch1:สัญญาณกระแสที่ได้จาก Current sensor

Ch2:สัญญาณกระแสที่ได้จาก Current Probe

จากรูปแสดงสัญญาณกระแสที่ได้จาก Current sensor เทียบกับสัญญาณกระแสที่ได้จาก Current Probe โดยสัญญาณกระแสที่ได้จาก Current Sensor ผ่านวงจร Noninverting Amplifier เพื่อขยายสัญญาณกระแสที่จะนำไปใช้ประมวลผลโดย DSP Board

6.1.3 นำสัญญาณกระแสจาก Current sensor มาทำ FFT (Fast Fourier Transform) เพื่อแสดงให้เห็นขนาดของสัญญาณ



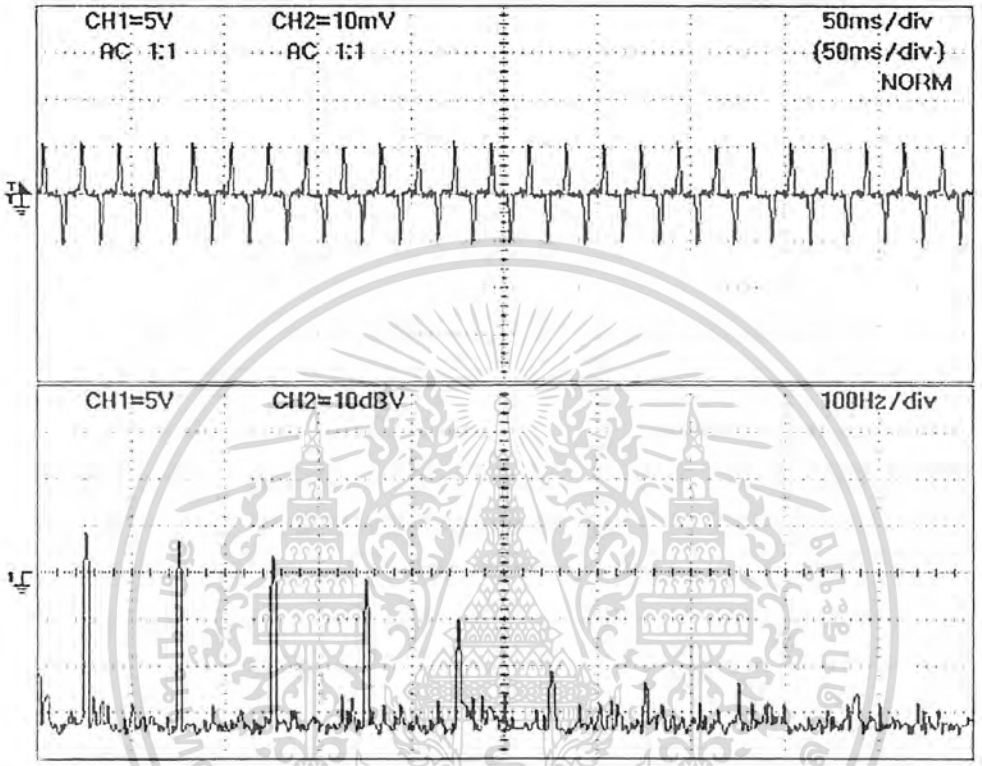
รูปที่ 6.3 สัญญาณ FFT จาก Current sensor

รูปบน:สัญญาณกระแสที่ได้จาก Current sensor

รูปล่าง:สัญญาณกระแสที่ได้จาก Current sensor ทำ FFT

จากรูปแสดงสัญญาณกระแสหลังจากทำ FFT จะเห็นว่ากระแสจะประกอบด้วยความถี่มูลฐาน (Fundamental frequency) รวมกับความถี่ที่ฮาร์มอนิกส์ต่างๆ

6.1.4 นำสัญญาณกระแสจาก Current Probe มาทำ FFT (Fast Fourier Transform) เพื่อแสดงให้เห็นขนาดของสัญญาณ



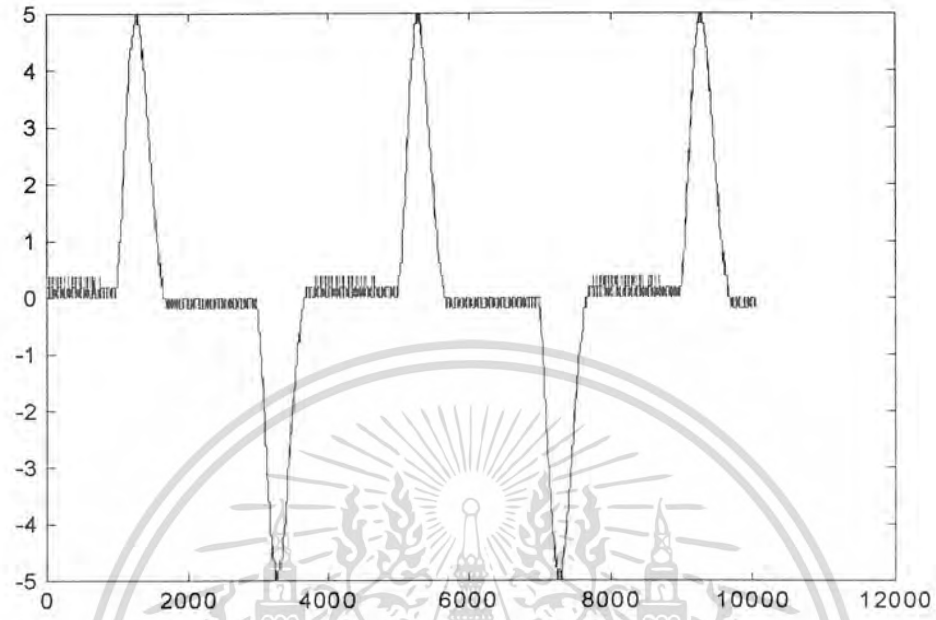
รูปที่ 6.4 สัญญาณ FFT จาก Current Probe

รูปบน:สัญญาณกระแสที่ได้จาก Current Probe

รูปล่าง:สัญญาณกระแสที่ได้จาก Current Probe ทำ FFT

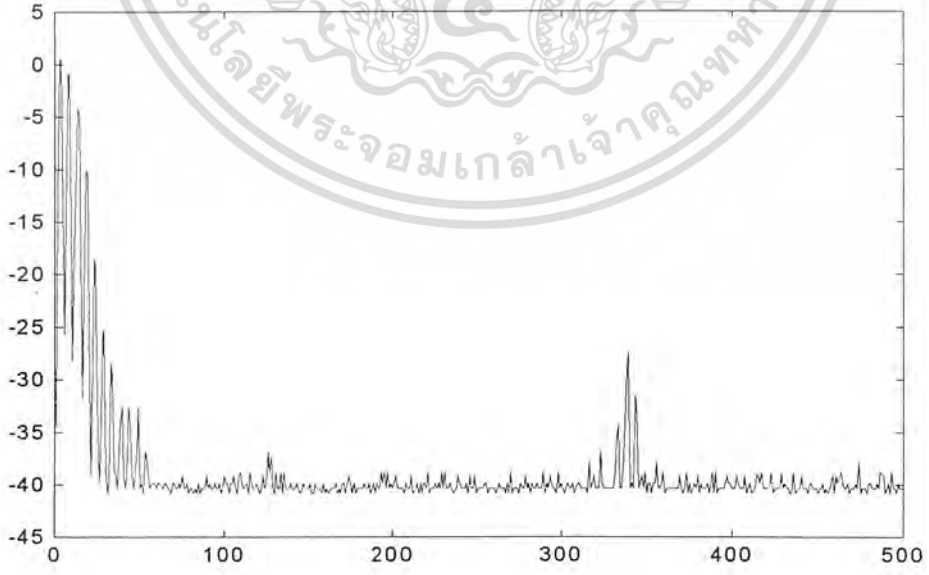
จากรูปแสดงสัญญาณกระแสหลังจากทำ FFT จะเห็นว่ากระแสจะประกอบด้วยความถี่มูลฐาน (Fundamental frequency) รวมกับความถี่ที่ฮาร์มอนิกส์ต่างๆ

6.1.5 นำค่ากระแสที่ได้จาก Storage scope มา plot ค่าโดยใช้โปรแกรม Matlab



รูปที่ 6.5ก รูปคลื่นกระแส plot ในโปรแกรม Mathlab

6.1.6 นำค่าที่ได้จากรูปมา Plot FFT โดยใช้โปรแกรม Matlab



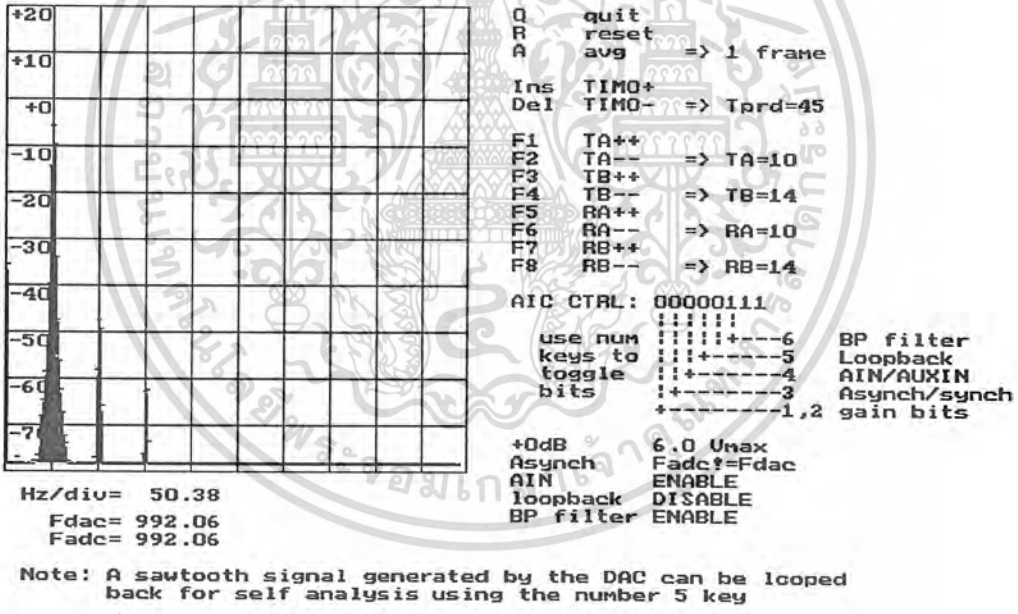
รูปที่ 6.5ข การวิเคราะห์ FFT จากโปรแกรม Mathlab

6.2 ส่วนการวิเคราะห์รูปคลื่นกระแส

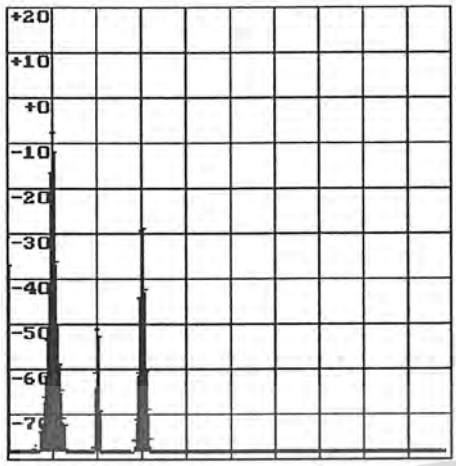
เป็นการนำสัญญาณกระแสที่ได้จาก Current Sensor มาประมวลผลโดย DSP Board โดยทั่วไปเราทราบว่าสัญญาณต่างๆจะประกอบไปด้วยสัญญาณที่มีความถี่ต่างๆมาประกอบกันเราจึงนำสัญญาณเหล่านั้นมาผ่านกระบวนการวิเคราะห์ฟาสต์ฟูเรียร์ (Fast Fourier Transform) แล้วแสดงผลออกมาเป็นแถบความถี่หรือสเปกตรัมของสัญญาณ

จาก TMS320C31 ได้ทำการเขียนโปรแกรมภาษาแอซแซมบลี้ของเครื่องในการติดต่อกับตัวเปลี่ยนสัญญาณอนาลอกเป็นดิจิตอล TLC32040 AIC เพื่อทำการสุ่มข้อมูล (Sampling) จากสัญญาณที่เราต้องการดูแถบความถี่หรือสเปกตรัมของสัญญาณ และนำค่าที่ได้จากการสุ่มข้อมูลนี้มาทำการประมวลผลโดยการวิเคราะห์ฟาสต์ฟูเรียร์ แล้วให้คอมพิวเตอร์นำค่าที่ได้ไปแสดงผลที่หน้าจอคอมพิวเตอร์

6.2.1 ทดลองป้อนสัญญาณรูปต่างๆให้กับ DSP Board ซึ่งค่าที่แสดงที่หน้าจอจะแสดงสเปกตรัมของสัญญาณที่ป้อนเข้ามาให้ตัวBoardโดยแสดงได้ดังนี้



รูปที่ 6.6 สเปกตรัมของสัญญาณคลื่นรูปซายน์ที่ความถี่ 50 Hz

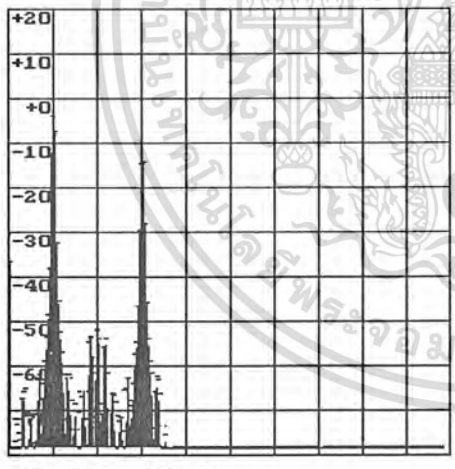



```
Hz/div=  50.38
  Fdac= 992.06
  Fadc= 992.06
```

```
Q    quit
R    reset
A    avg      => 1 frame

Ins  TIMO+
Del  TIMO-    => Tprd=45

F1   TA++
F2   TA--     => TA=10
F3   TB++
F4   TB--     => TB=14
F5   RA++
F6   RA--     => RA=10
F7   RB++
F8   RB--     => RB=14

AIC CTRL: 00000111
          |||||||
  use num |||||+---6   BP filter
  keys to ||||+----5   Loopback
  toggle  |||+-----4   AIN/AUXIN
  bits    ||+------3   Asynch/synch
          ++-------1,2 gain bits

+0dB      6.0 Vmax
Asynch    Fadc!=Fdac
AIN       ENABLE
loopback  DISABLE
BP filter ENABLE
```

```
Note: A sawtooth signal generated by the DAC can be looped
      back for self analysis using the number 5 key
```

รูปที่ 6.7 สเปกตรัมของสัญญาณคลื่นรูปฟันเลื่อยที่ความถี่ 50 Hz



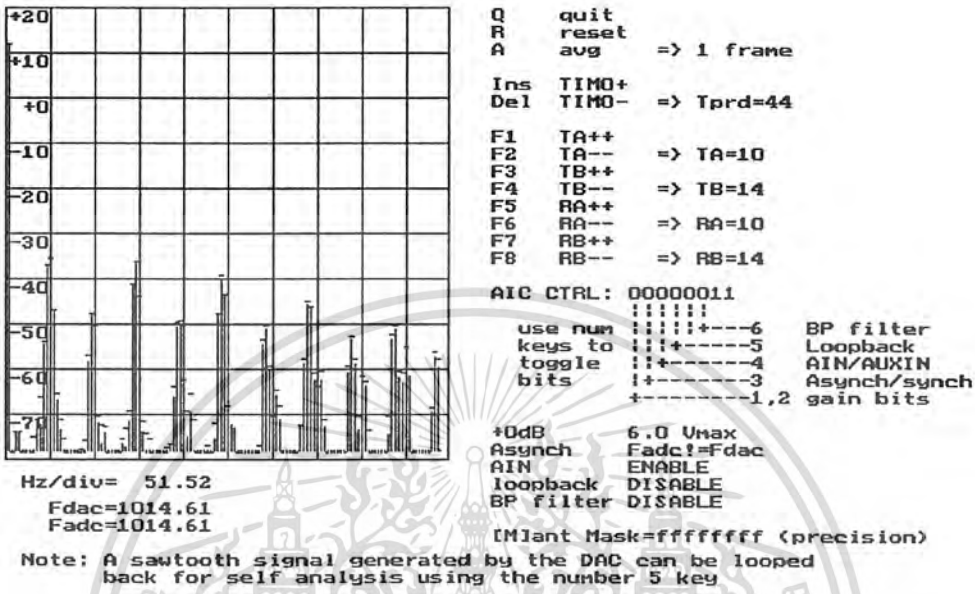

```
Hz/div=  50.38
  Fdac= 992.06
  Fadc= 992.06
```

```
Q    quit
R    reset
A    avg      => 1 frame

Ins  TIMO+
Del  TIMO-    => Tprd=45

F1   TA++
F2   TA--     => TA=10
F3   TB++
F4   TB--     => TB=14
F5   RA++
F6   RA--     => RA=10
F7   RB++
F8   RB--     => RB=14

AIC CTRL: 00000111
          |||||||
  use num |||||+---6   BP filter
  keys to ||||+----5   Loopback
  toggle  |||+-----4   AIN/AUXIN
  bits    ||+------3   Asynch/synch
          ++-------1,2 gain bits

+0dB      6.0 Vmax
Asynch    Fadc!=Fdac
AIN       ENABLE
loopback  DISABLE
BP filter ENABLE
```

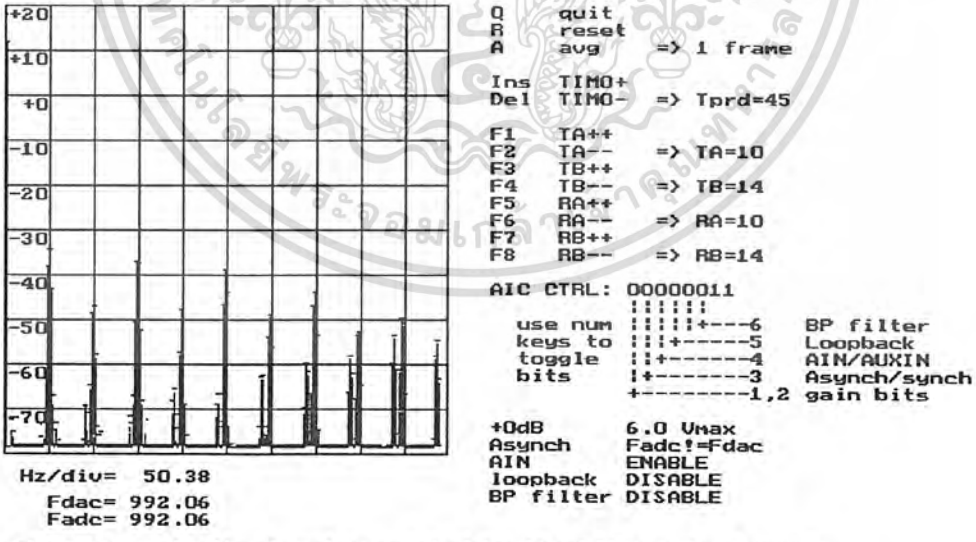
```
Note: A sawtooth signal generated by the DAC can be looped
      back for self analysis using the number 5 key
```

รูปที่ 6.8 สเปกตรัมของสัญญาณคลื่นรูปสี่เหลี่ยมที่ความถี่ 50 Hz

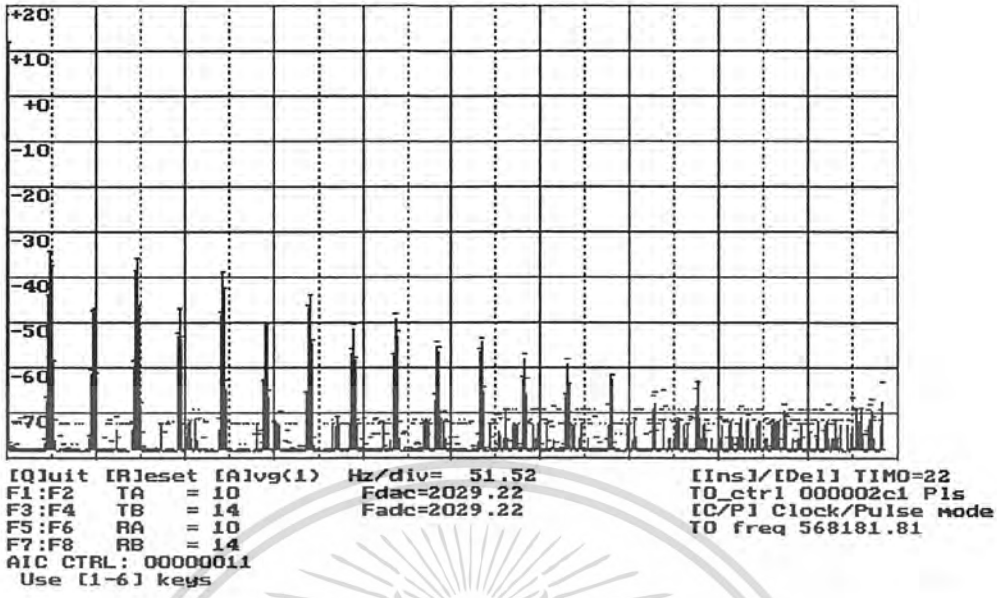
6.2.2 สัญญาณที่ได้จาก Current Sensor มาป้อนให้กับ DSP Board โดยการแซมปลิ้งที่จำนวนจุดต่างๆดังต่อไปนี้



รูปที่ 6.9 สเปกตรัมของสัญญาณจาก Current Sensor ทำการแซมปลิ้งที่256จุด

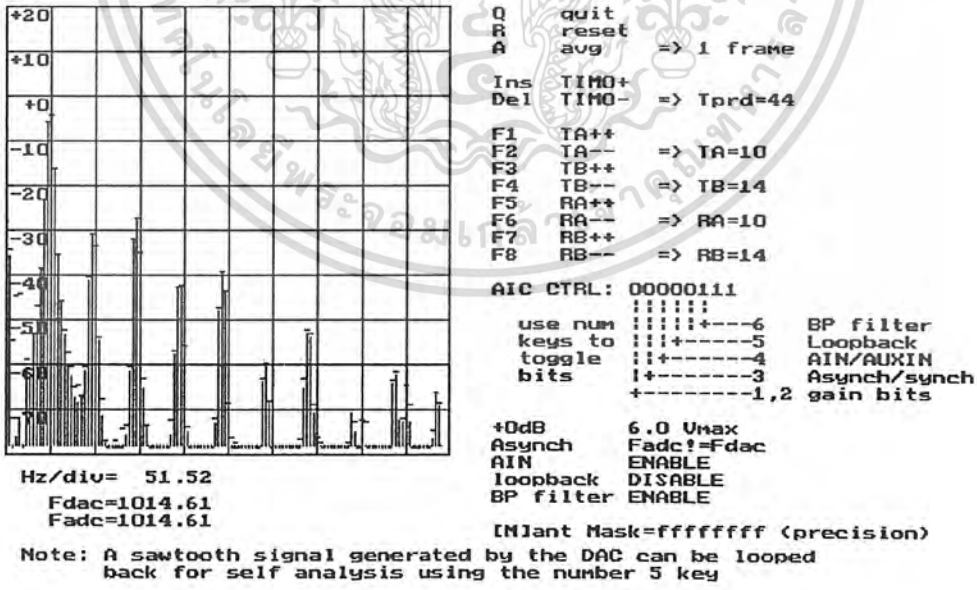


รูปที่ 6.10 สเปกตรัมของสัญญาณจาก Current Sensor ทำการแซมปลิ้งที่512จุด

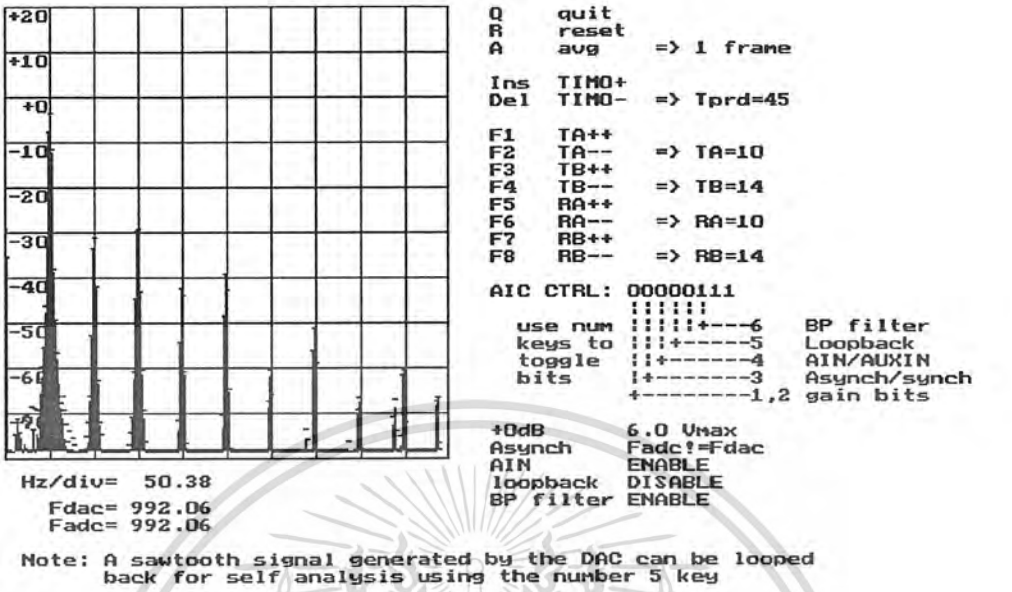


รูปที่ 6.11 สเปกตรัมของสัญญาณจาก Current Sensor ทำการแซมปลิ้งที่1024จุด

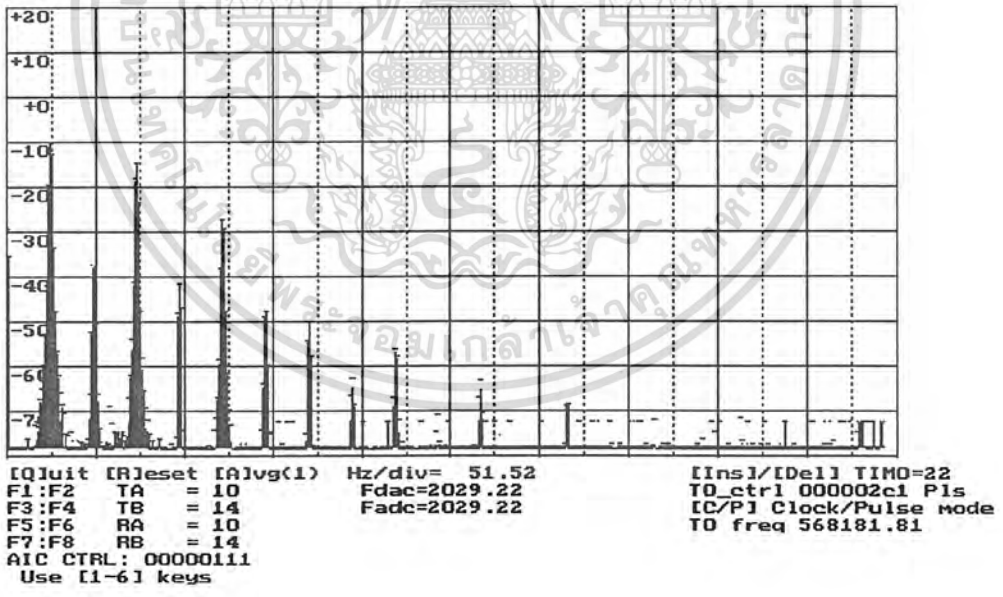
6.2.3 สัญญาณที่ได้จาก Current Sensor มาป้อนให้กับ DSP Board โดยการแซมปลิ้งที่จำนวนจุดต่างๆเมื่อผ่าน Band Pass Filter



รูปที่ 6.12 สเปกตรัมของสัญญาณที่ผ่าน Band Pass Filter ทำการแซมปลิ้งที่256จุด



รูปที่ 6.13 สเปกตรัมของสัญญาณที่ผ่าน Band Pass Filter ทำการแซมปลิ้งที่512จุด



รูปที่ 6.14 สเปกตรัมของสัญญาณที่ผ่าน Band Pass Filter ทำการแซมปลิ้งที่1024จุด

6.3 สรุปผลการทดลอง

จากการทดลองที่ได้ทดลองไปทั้งหมดทำให้สามารถหาข้อสรุปและปัญหาที่ได้พบดังนี้

1. ปัญหาของตัว Current Sensor โดยปัญหาของตัว Current Sensor นั้นมีอยู่เรื่องเดียวคือเรื่องของการวัดค่ากระแสที่ได้ โดยตัว Current Sensor นั้นจะมี offset ของตัวมันเองอยู่ซึ่งตัวที่ใช้นี้จะมี offset เท่ากับ Vcc/2

แนวทางการแก้ปัญหา เราทำได้โดยจ่ายไฟเข้าทั้งค่าบวกและลบจะทำให้ offset ของตัว Current Sensor

2. สัญญาณที่ได้จาก Current Sensor มีค่าน้อยมาก

แนวทางการแก้ปัญหา ทำได้โดยต่อวงจร Noninverting Amplifier เพื่อขยายสัญญาณให้เพิ่มขึ้น

3. ปัญหาเกี่ยวกับตัว Board ที่จะนำมาใช้ในการประมวลผลของสัญญาณนั้นมีปัญหาบางประการระหว่างที่ทำการทดลอง ทำให้ Board ตัวเก่าที่เคยใช้งานนั้นพังทำให้ต้องทำการเปลี่ยน Board ประมวลผลใหม่

แนวทางการแก้ปัญหา ทำได้โดยเปลี่ยน Board ประมวลผลใหม่โดยต้องเปลี่ยนยี่ห้อและรุ่นใหม่ เพราะ Board ประมวลผลตัวเก่านั้นต้องสั่งซื้อจากต่างประเทศทำให้ไม่สามารถที่จะรอได้อีกทั้งยังไม่สามารถหาร้านที่จะสั่งซื้อได้ด้วย

4. ภาษาที่ใช้ในการเขียนโปรแกรมไม่เหมือนกัน

แนวทางการแก้ปัญหา คือต้องเริ่มศึกษาภาษาที่ใช้ในการเขียนโปรแกรมใหม่หมดเพราะ Board ประมวลผลตัวใหม่ใช้ ภาษา Assembly ของ Board ในการเขียนโปรแกรม และใช้ ภาษาC++ ในการเขียนโปรแกรมแสดงผลหน้า computer

6.4 ข้อดีของ Active Power Filter

1. การใช้ Active Power Filter จะไม่มีข้อจำกัดด้านกระแสไฟฟ้าโหลดหรือกระแสฮาร์มอนิกส์ว่ามากน้อยเพียงใด กล่าวคือ กระแสฮาร์มอนิกส์จะมีค่าหลายร้อยแอมป์ก็สามารถออกแบบให้รองรับกระแสไฟฟ้าดังกล่าวได้

2. สามารถออกแบบติดตั้งได้ทั้งแรงดันสูงและแรงดันต่ำ

3. มีข้อจำกัดเกี่ยวกับโหลดของระบบน้อยมาก

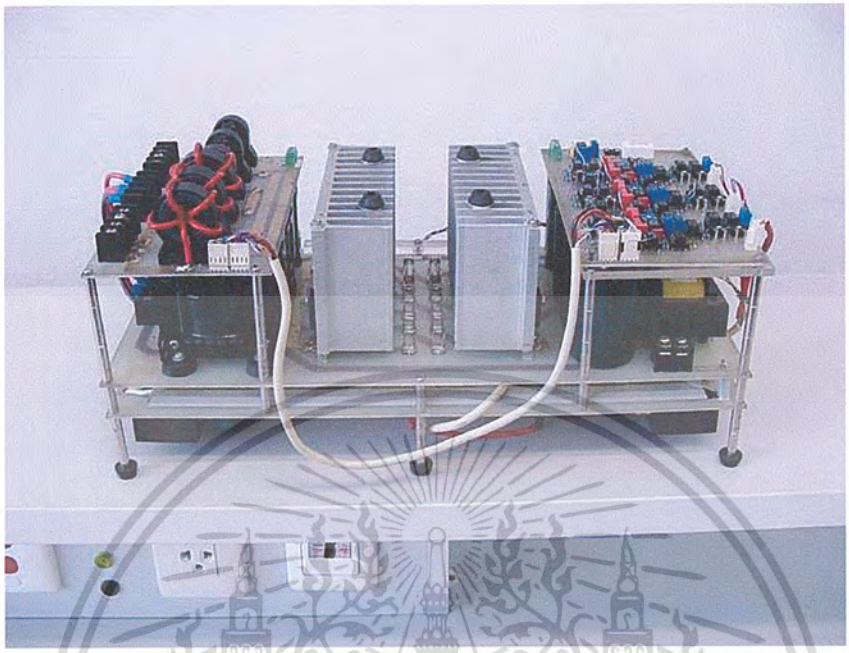
4. สามารถปรับปรุงค่า Power Factor ให้อยู่ในเกณฑ์ที่ดีได้

5. ติดตั้งที่จุดเมนของระบบเพียงจุดเดียวทำให้ง่ายต่อการควบคุมและดูแลรักษา

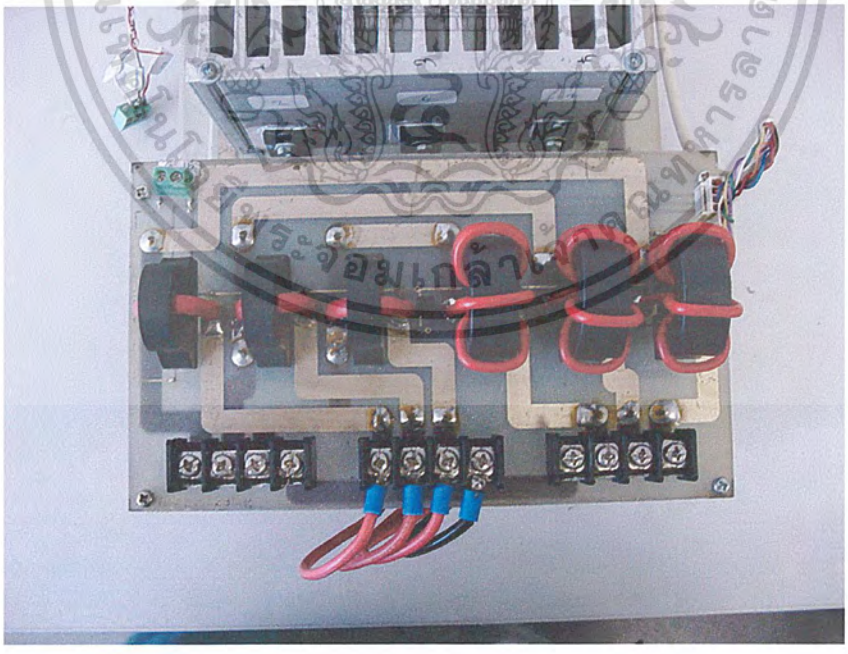
ข้อเสียของ Active Power Filter

1. ค่าเริ่มต้นและค่าใช้จ่ายในการดำเนินงานและติดตั้งค่อนข้างแพงเมื่อเทียบกับแบบอื่น

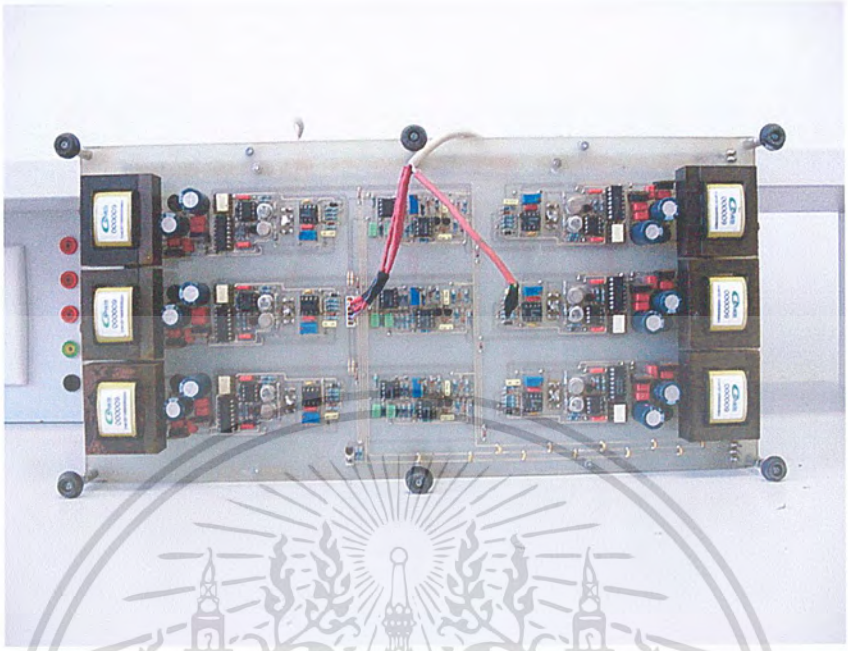
6.5 รูป Active Power Filter 3 เฟส 4 สายในส่วนต่างๆ



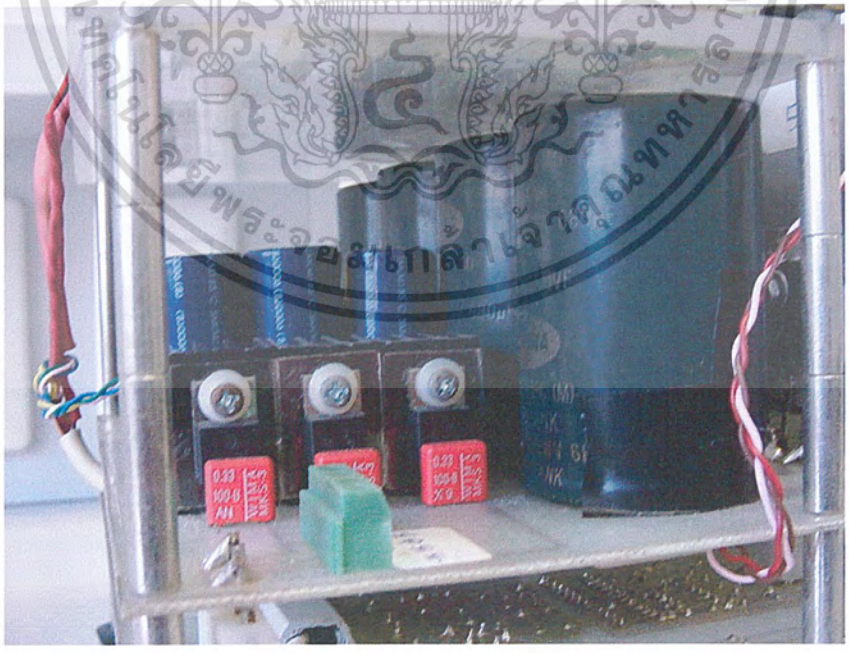
รูป 6.15 Active Power Filter



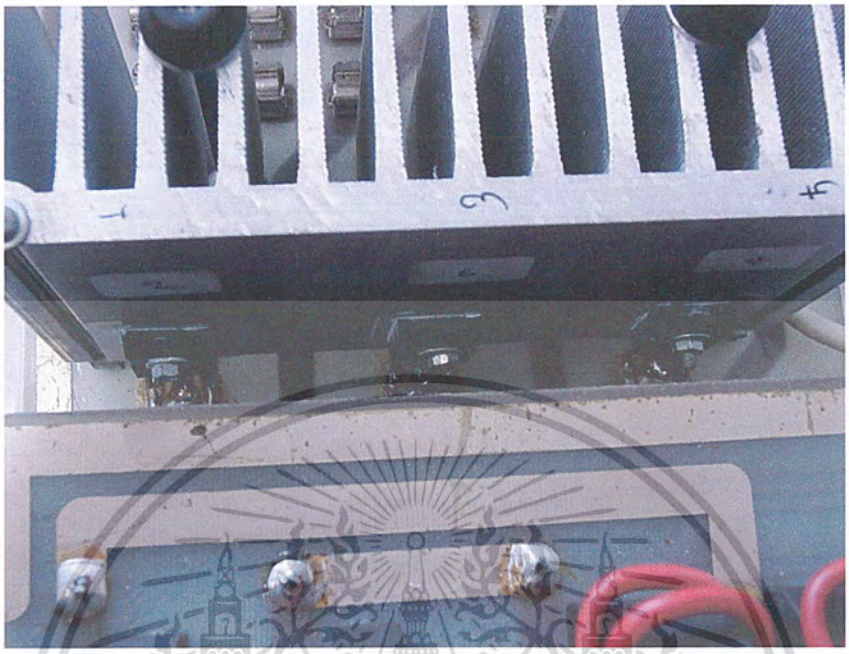
รูป 6.16 ส่วนตรวจจับสัญญาณกระแส (Current Sensor)



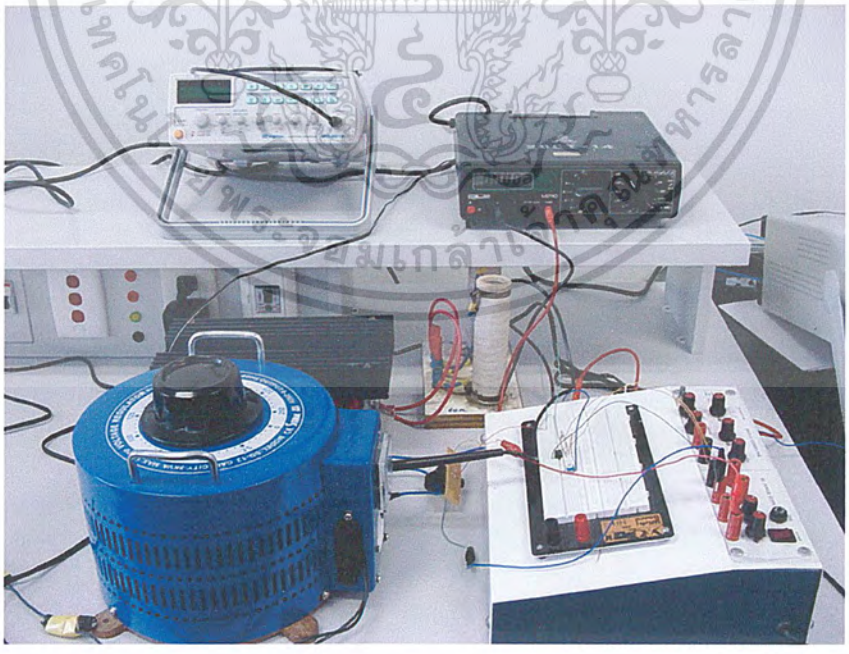
รูป 6.17 ส่วนขับเกท



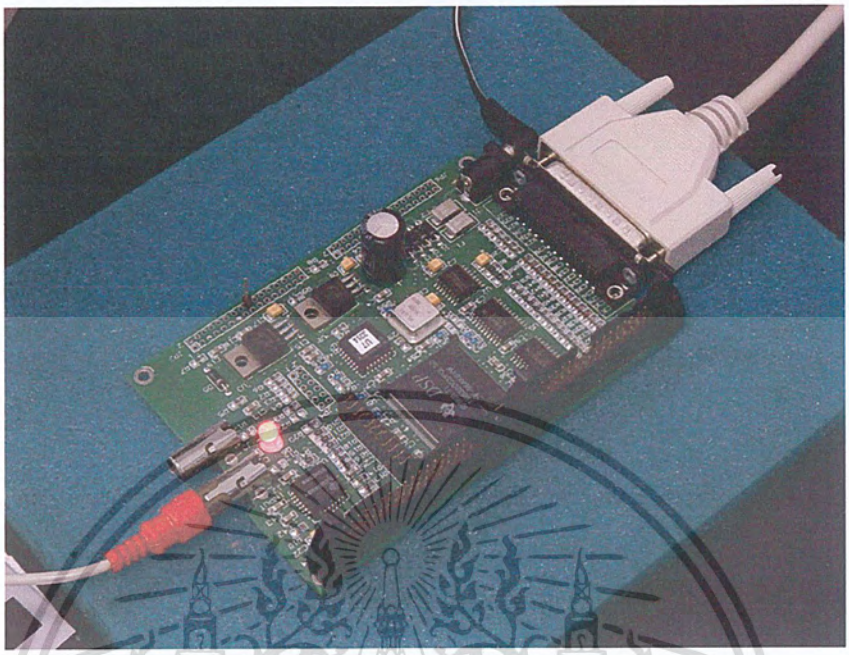
รูป 6.18 ส่วน DC busbar



รูป 6.19 IGBT inverter



รูป 6.20 อุปกรณ์ที่ใช้ทดลองในการวัดสัญญาณกระแส



รูป 6.21 DSP Board TMS320C31

รูป 6.22 ตัวตรวจจับกระแส (Current Sensor)

เอกสารอ้างอิง

1. TMS320C3x DSP User's Guide , Texas Instrument Incorparated ,USA

2. TMS320C3x DSP starter kit User's Guide , Texas Instrument Incorporated , USA

3. RULPH CHASSAING , Digital Signal Processing Laboratory Experiments Using C and the TMS32C31 DSK , JOHN WILEY & SONS ,INC ,USA

4. Vinay K.Ingle , John G.Proakis , Digital Signal Processing Using MATLAB , Bookware Companion Series , Canada

5. เอก ไชยสวัสดิ์ ,สัญญาณและระบบ Signal and System ,มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี , กรุงเทพ

ภาคผนวก ก

C3XMMRS.ASM

```
DMA_ctrl  .set 0x808000;  DMA cntl
DMA_srce  .set 0x808004;  DMA srce address
DMA_dest  .set 0x808006;  DMA dest address
DMA_xfr   .set 0x808008;  DMA xfer counter
T0_ctrl   .set 0x808020;  TIM0 gl control
T0_count  .set 0x808024;  TIM0 count
T0_prd    .set 0x808028;  TIM0 prd
T1_ctrl   .set 0x808030;  TIM1 gl control
T1_count  .set 0x808034;  TIM1 count
T1_prd    .set 0x808038;  TIM1 prd
S0_gctrl  .set 0x808040;  SP 0 global control
S0_xctrl  .set 0x808042;  SP 0 FSX/DX/CLKX port ctl
S0_rctrl  .set 0x808043;  SP 0 FSR/DR/CLKR port ctl
S0_tctrl  .set 0x808044;  SP 0 R/X timer control
S0_tcount .set 0x808045;  SP 0 R/X timer counter
S0_tprd   .set 0x808046;  SP 0 R/X timer period
S0_xdata  .set 0x808048;  SP 0 Data transmit
S0_rdata  .set 0x80804C;  SP 0 Data receive
S1_gctrl  .set 0x808050;  SP 1 global control
S1_xctrl  .set 0x808052;  SP 1 FSX/DX/CLKX port ctl
S1_rctrl  .set 0x808053;  SP 1 FSR/DR/CLKR port ctl
S1_tctrl  .set 0x808054;  SP 1 R/X timer control
S1_tcount .set 0x808055;  SP 1 R/X timer counter
S1_tprd   .set 0x808056;  SP 1 R/X timer period
S1_xdata  .set 0x808058;  SP 1 Data transmit
S1_rdata  .set 0x80805C;  SP 1 Data receive
e_buscon  .set 0x808060;  Exp bus control
p_buscon  .set 0x808064;  Pri bus control
JJUMP     .set    0x809ff4    ;<- Base address
```

```
JXWRIT  .set  0x809ff5  ;
JXREAD  .set  0x809ff6  ;
JXCTXT  .set  0x809ff7  ;
JXRUNF  .set  0x809ff8  ;
JXSTEP  .set  0x809ff9  ;
JXHALT  .set  0x809ffA  ;
JW_HOST .set  0x809ffB  ;
JR_HOST .set  0x809ffC  ;
JSPARE  .set  0x809ffd  ;
```

FFT256.ASM

```
RAM0    .set   0x809800
RAM1    .set   0x809C00
;-------------------------------------
TA      .set   10    ; AIC Default startup matches host side app
TB      .set   14    ;
RA      .set   10    ;
RB      .set   14    ;
        .include "C3XMMRS.ASM"
N       .set   256
N2      .set   N/2
PI      .set   3.1415926
PI2     .set   2*PI
PI2N    .set   2.0*PI/N
PIN     .set   PI/N
;=============================================
; Create the Twiddle, FFT and I/O buffer arrays
;=============================================
        .start "TWIDDLES",0X809800 ;
        .sect  "TWIDDLES"          ;
TR                          ; 0x809800
        .loop  N2           ;
        .float  cos(br($-TR,N)*PIN);
        .endloop            ;
TI                          ;
        .loop  N2           ; 0x809880
        .float -1*sin(br($-TI,N)*PIN);
        .endloop            ;
```

```
BF      .set   TI+N2            ; 0x809900
DR      .set   BF+N             ; 0x809A00
DI      .set   DR+N             ; 0x809B00
;=============================================================
        .start "FFTCODE",0x809C00  ; Start of RAM1
        .sect  "FFTCODE"        ;
;=============================================================
_STOP    .set  1
_START   .set  2
;
MSG_BOX  .word _STOP            ; 0x809C00
TLVL     .word 1000             ; 0x809C01
A_REG    .word (TA<<9)+(RA<<2)+0 ; 0x809C02
B_REG    .word (TB<<9)+(RB<<2)+2 ; 0x809C03
C_REG    .word 00000011b        ; 0x809C04  +/- 1.5 V
EDGESEL  .word 1                ; 0x809C05
SIZE     .word N                ; 0x809C06
MASK     .word 0xFFFFFFFF       ; 0x809C07  Mask for mantissa bits
A_REGOLD .word 0
B_REGOLD .word 0
C_REGOLD .word 0
;
S0_gctrl_val .word 0x0E970300 ; Runtime value XINT/RINT enabled
S0_xctrl_val .word 0x00000111 ;
S0_rctrl_val .word 0x00000111 ;
;
FFTSIZE  .word N              ; 256 point FFT
TR_ADDR  .word TR               ; 128 SIN values
TI_ADDR  .word TI             ; 128 COS values
DR_ADDR  .word DR               ; 128 REAL data point
DI_ADDR  .word DI             ; 128 IMAG data point
BF_ADDR  .word BF               ; Array of 256  Real buffer data
```

```
TEMP    .word 0
;-------------------------------
main    ldi  0x30,IE
        ldi  @S0_rdata,R0   ; Clear SP under/overflow
        ldi  0,R0           ;
        sti  R0,@S0_xdata   ;
        ldi  @S0_rdata,R0   ;
        ldi  0,R0           ;
        sti  R0,@S0_xdata   ;
        sti  R0,@RAMP

        ldi  25,RC          ; Preload some ADC data to flush out
        rptb preload        ; the AIC after putting it to sleep
preload call GETADC         ;
        ;---------------------
        ldi  @DR_ADDR,AR0   ;
        ldi  @DI_ADDR,AR1   ;
        ldi  @SIZE,RC       ; Now get samples
        subi 1,RC           ; N+1 repeats
        rptb samples        ;
        ;---------------------
        call GETADC         ;
        float R0,R0         ;
        stf  R0,*AR0++      ; store to data array
        ldf  0,R0           ;
samples stf  R0,*AR1++      ;
        ;---------------------
        ldi  0,R0           ; Put 0 into the DXR when it is not
        sti  R0,@S0_xdata   ; going to be used for awhile
;=============================================================

; Perform FFT
;-------------------------------
```

```
FFT:    ldi  @FFTSIZE,IR0   ;
        ldi  @FFTSIZE,IR1   ;
        lsh  -1,IR0
New_Stg
        ldi  @FFTSIZE,RC    ;
        ldi  @DR_ADDR,AR0   ;
        ldi  @DI_ADDR,AR1   ;
        ldi  @TR_ADDR,AR2   ; Tw Base never modified
        ldi  @TI_ADDR,AR3   ;
        lsh  -1,RC          ; Perform FFTSIZE/2 butterfly loops
        subi 1,RC           ;
        lsh  -1,IR0         ;
        lsh  -1,IR1         ;
        ldi  IR1,R0         ;
        bz   FFT_END        ;
        ;---------------------
Blk_Top rptb B_Fly          ; Start by getting all 6 Butterfly inputs
        ldf  *+AR0(IR1) ,R0 ; Bt real
    ||  ldf  *AR0       ,R1 ; Tp real
        ldf  *+AR1(IR1) ,R2 ; Bt imag
    ||  ldf  *AR1       ,R3 ; Tp imag
        ldf  *AR2++(IR0)B,R4 ; TW real - R4
    ||  ldf  *AR3++(IR0)B,R5 ; TW imag - R5

        and  @MASK,R0
        and  @MASK,R1
        and  @MASK,R2
        and  @MASK,R3
        and  @MASK,R4
        and  @MASK,R5

        addf3 R0,R1,R6      ; Top sum REAL
```

```
;  stf  R6,*AR0       ;
;  addf3 R2,R3,R7     ; Top sum IMAG
   addf3 *+AR1(IR1),R3,R7; Top sum IMAG
||  stf  R6,*AR0

   subf3 R0,R1,R6     ; R6=d_REAL  (R1 free)
   stf  R7,*AR1       ;

   subf3 R2,R3,R7     ; R7=d_IMAG  (R3 free)

   mpyf3 R6,R4,R1     ; R1 = R*TR = REAL_1
   mpyf3 R7,R5,R3     ; R3 = I*TI = REAL_2
   subf  R3,R1        ;
   stf  R1,*+AR0(IR1)  ; Store bottom real
   nop  *++AR0
   mpyf3 R6,R5,R1     ; R1 = R*TI = IMAG_1
   mpyf3 R7,R4,R3     ; R3 = I*TR = IMAG_2
   addf  R3,R1        ;
   stf  R1,*+AR1(IR1)  ; Store bottom real
   nop  *++AR1
   ;----------------------
   ; Identify EOB by twiddle wraparound
   ;----------------------
ident   ldi  @TR_ADDR,R7    ; At the end of a block the bit-reversed
   subi  AR2,R7        ; addressing of the twiddles will cause
   ldiz  IR1,R7        ; TR_ADDR==AR2.  If true, increment the R/I
   ldinz 0,R7          ; data pointers to start at the next block.
   addi  R7,AR0        ;
B_Fly   addi  R7,AR1        ; Loop till butterflys finished
   b    New_Stg       ; When RC=0, set up for new stage
;=============================================================
```

```
; When the FFT is complete a convolution with the response of the desired
; window function is used to clean up (filter) the otherwise non-windowed
; input samples.  In this case a raised cosine window is used since the
; filter coefficients are easily calculated as -0.5,1.0,-0.5.
;
;=================================================================
=
FFT_END  ldi  @BF_ADDR,AR0   ; OUTPUT ptr
         ldi  @DR_ADDR,AR1   ; REAL ptr
         ldi  @DI_ADDR,AR4   ; IMAG ptr
         ldi  @SIZE,IR0      ; IR0=SIZE/2 for bit-reverse access
         lsh  -1,IR0         ;
         ldi  IR0,R0         ;
         subi 1,R0           ; To unroll the response for DC a bit
         or   IR0,R0         ; reversed add of the twos compliment of the
         addi R0,AR1         ; index is used
         addi R0,AR4         ;
         sti  R0 ,@BRINDX_N  ; The BR twos compliment and normal index
         sti  IR0,@BRINDX_P  ; are both kept for easy backrolling
         ;--------------------
         ldi  AR1,AR2        ; AR1 = REAL[N-1] (one sample before DC)
         ldi  AR4,AR5        ; AR4 = IMAG[N-1]
         ;=================================================
         ldi  @SIZE,RC       ;
         lsh  -2,RC          ; Pack 4 results per word
         rptb WINDOW         ;
         ;---------------------
         ldi  0,R7           ; Call to Log_Mag returns with filtered
         call Log_Mag        ; R^2 + I^2 packed into 8 MSBs
         lsh  -24,R0         ; shift and pack 4 samples
         or   R0,R7          ;
         call Log_Mag        ; again
```

```
        lsh  -16,R0        ;
        or   R0,R7         ;
        call Log_Mag       ; again
        lsh  -8 ,R0        ;
        or   R0,R7         ;
        call Log_Mag       ; again
        lsh  0 ,R0         ;
        or   R0,R7         ;
WINDOW  sti  R7,*AR0++     ; store the packed data
        ;--------------------
        ldi  0x4,IE        ; Interlock with host only uses INT2
        ldi  _START,R0     ;
NO_START cmpi @MSG_BOX,R0  ; Restart when START message is received
        bnz  NO_START      ;
        ldi  _STOP,R0      ; Set MSG box to STOP
        sti  R0,@MSG_BOX   ;
        ;--------------------
        ldi  0,R2          ;
        ldi  @A_REG  ,R0   ; Reload any new runtime parameters
        cmpi @A_REGOLD,R0  ;
        ldinz 1,R2         ;
        sti  R0,@A_REGOLD  ;
        ;-----------
        ldi  @B_REG  ,R0   ;
        cmpi @B_REGOLD,R0  ;
        ldinz 1,R2         ;
        sti  R0,@B_REGOLD  ;
        ;-----------
        ldi  @C_REG  ,R0   ;
        cmpi @C_REGOLD,R0  ;
        ldinz 1,R2         ;
        sti  R0,@C_REGOLD  ;
```

```
        ;- - - - - - - - - - -
        cmpi  0,R2          ;
        bz    main          ;
        call  AIC_INIT      ; Restart with new AIC setup
        b     main          ; Do it all over again!
;=============================================================================
=
; Log_Mag: This function assumes that AR1,AR2,AR3 and IR0 have been
;         preinitialized to access the 'post convulationaly windowed'
;         FFT array.
;=============================================================================
=
BRINDX_N .word  0           ; BR index values used to move through data
BRINDX_P .word  0           ;
VU_scale .float 1/(N*128.0)     ; scale factor for FFT data growth
;- - - - - - - - - - - - - - - -
Log_Mag  push R2            ; 26 pixels/10 dB
        push  R4            ; 0.75000 V = 0 dBm
        push  R5            ;
        pushf R2            ;
        pushf R4            ;
        pushf R5            ;
        ldf   -0.500,R5     ;
        ;- - - - - - - - - - - -
        mpyf3 R5,*AR1++(IR0)B,R0;
        addf    *AR1++(IR0)B,R0;
        mpyf3 R5,*AR1       ,R2;
        addf  R0,R2         ;
        mpyf  @VU_scale,R2      ; Unscale data growth and fixed pt ADC
        mpyf  R2,R2         ; REAL^2
        ;- - - - - - - - - - - -
        mpyf3 R5,*AR4++(IR0)B,R0;
```

```
addf   *AR4++(IR0)B,R0;
mpyf3 R5,*AR4      ,R4;
addf  R0,R4          ;
mpyf  @VU_scale,R4    ;
mpyf  R4,R4          ; IMAG^2
;------------
ldi   @BRINDX_N,IR0    ; BR roll-back of pointers one sample
nop   *AR1++(IR0)B     ;
nop   *AR4++(IR0)B     ;
ldi   @BRINDX_P,IR0    ; Normal indexing
;------------
addf  R4,R2          ; REAL^2 + IMAG^2
;------------
lsh   1,R2          ; Quick log using float equivelency
pushf R2            ; See Designer Notebook Page DNP-22
pop   R0            ;
;------------
ash   -21,R0         ; Pack quick log number into 8 MSBs
cmpi  -128,R0        ; preserving three mantissa bits
ldile -128,R0        ; and clipping the result
cmpi   127,R0        ;
ldige  127,R0        ;
lsh    24,R0         ; result is in 8 MSBs, 24 lsbs are 0
;------------
popf  R5            ;
popf  R4            ;
popf  R2            ;
pop   R5            ;
pop   R4            ;
pop   R2            ;
rets              ;
;*************************************************************************
```

```
RAMP    .word 0
FLAGS   .word 0

GETADC  ldi  0x30,IE      ; Come here and wait for ADC interrupt
        IDLE              ; confirmation to save power and code space
        ldi  @FLAGS,R0    ;
        tstb 0x20,R0      ;
        bz   $-3          ;
        andn 0x20,R0      ;
        sti  R0,@FLAGS    ;
        ldi  @S0_rdata,R0 ; Return sign extended ADC value
        lsh  16,R0        ;
        ash  -16,R0       ;
        rets

ADC     push ST           ; On interrupt, set a software flag to
        push R0           ; let the CPU know that the RINT has been
        ldi  @S0_rdata,R0 ;
        ldi  @FLAGS,R0    ;
        or   0x20,R0      ;
        sti  R0,@FLAGS    ;
        pop  R0           ;
        pop  ST           ;
        reti              ;

DAC     push ST           ;
        push R1           ;
        ldi  @RAMP,R1     ; Send RAMP signal out DAC for loopback test
        subi 1024,R1      ;
        lsh  17,R1
        ash  -17,R1
        andn 3,R1         ;
```

```
        sti  R1,@RAMP       ;
        sti  R1,@S0_xdata   ; loopback ADC->DAC
        pop  R1
        pop  ST
        reti
;---------------------------------
prog_AIC push R1            ;
        push IE             ;
        ldi  0x10,IE        ;
        andn 0x30,IF        ;
        ldi  @S0_xdata,R1   ; Use original DXR data during 2 ndy
        sti  R1,@S0_xdata   ;
        idle
        ldi  @S0_xdata,R1   ; Use original DXR data during 2 ndy
        or   3,R1           ; Request 2 ndy XMIT
        sti  R1,@S0_xdata   ;
        idle                ;
        sti  R0,@S0_xdata   ; Send register value
        idle                ;
        andn 3,R1           ;
        sti  R1,@S0_xdata   ; Leave with original safe value in DXR
        pop  IE             ;
        pop  R1             ;
        rets                ;
;=============================================================;
; This section of code is called by the initialization ;
; code as well as by the main program loop.  It is    ;
; therfor assembled into the regular program RAM      ;
;=============================================================;
AIC_INIT push R0            ;
        LDI  0x10,IE        ; Enable XINT interrupt
        andn 0x34,IF        ;
```

```
AIC_reset
        ldi  0,R0          ;
        sti  R0,@S0_xdata  ;
        RPTS 0x040         ;
        LDI  2,IOF         ; XF0=0 resets AIC
        rpts 0x40          ;
        LDI  6,IOF         ; XF0=1 runs AIC
        ldi  @S0_rdata,R0
        ldi  0,R0
        sti  R0,@S0_xdata
        ;-----------------------------
        ldi  @C_REG,R0     ; Setup control register
        call prog_AIC      ;
        ldi  0xfffc ,R0    ; Program the AIC to be real slow
        call prog_AIC      ;
        ldi  0xfffc|2,R0   ;
        call prog_AIC      ;
        ldi  @B_REG,R0     ; Bump up the Fs to final rate
        call prog_AIC      ; (smallest divisor should be last)
        ldi  @A_REG,R0     ;
        call prog_AIC      ;
        pop  R0            ;
        ldi  0,R0          ; Put a safe 0 in DXR
        sti  R0,@S0_xdata  ;
        ldi  @S0_rdata,R0  ; Clear receive underrun
        rets               ;
;*******************************************************;
; Startup stub...                                       ;
;                                                       ;
; The following section of code is used only once for   ;
; initialization and can be safely overwritten by       ;
; assembling it into the stack or volatile data         ;
```

```
; storage.                                    ;
;*************************************************;
        .start  "STUB",BF    ; Place this code in the data buffer
        .sect   "STUB"       ; area as this is the first to go
        .entry  ST_STUB
ST_STUB  ldp  T0_ctrl        ; Use kernel data page and stack
        ldi  0,R0            ; Halt TIM0 & TIM1
        sti  R0,@T0_ctrl     ;
        sti  R0,@T1_ctrl     ;
        sti  R0,@T0_count    ; Set counts to 0
        sti  R0,@T1_count    ;
        ldi  1,R0            ; Set periods to 1
        sti  R0,@T0_prd      ;
        sti  R0,@T1_prd      ;
        ldi  0x2C1,R0        ; Restart both timers
        sti  R0,@T0_ctrl     ;
        sti  R0,@T1_ctrl     ;
        ;---------------------
        ldi  @S0_xctrl_val,R0;
        sti  R0,@S0_xctrl    ; transmit control
        ldi  @S0_rctrl_val,R0;
        sti  R0,@S0_rctrl    ; receive control
        ldi  0,R0            ;
        sti  R0,@S0_xdata    ; DXR data value
        ldi  @S0_gctrl_val,R0; Setup serial port
        sti  R0,@S0_gctrl    ; global control
        ;---------------------
        call AIC_INIT        ; Initialize the AIC
        ldi  0x30,IE         ; Service both RINT/XINT
        ldi  @S0_rdata,R0    ;
        b    main            ;
;*************************************************;
```

```
; Install the XINT/RINT ISR handler directly into   ;
; the vector RAM location it will be used in        ;
;************************************************;
        .start  "SP0VECTS",0x809FC5
        .sect   "SP0VECTS"
        B    DAC        ; XINT0
        B    ADC        ; RINT0
;***************************************************************;
; The folowing section is for C3x devices which are configured;
; in the u-processor mode (eg EVM or XDS510) and therefor have;
; RAM instead of bootrom at the vector table address.  The    ;
; DSK loader will also write to this address but the bootrom  ;
; is obviously not modifiable...                              ;
;***************************************************************;
        .start  "ROMVECTS",0x000000
        .sect   "ROMVECTS"
        .word   0x809FC0  ; RESET
        .word   0x809FC1  ; INT0
        .word   0x809FC2  ; INT1
        .word   0x809FC3  ; INT2
        .word   0x809FC4  ; INT3
        .word   0x809FC5  ; XINT0
        .word   0x809FC6  ; RINT0
        .word   0x809FC7  ; XINT1  NA
        .word   0x809FC8  ; RINT1  NA
        .word   0x809FC9  ; TINT0
        .word   0x809FCA  ; TINT1
        .word   0x809FCB  ; DINT
```

FFT512.ASM

```
TA      .set  10              ; Use AIC startup == host side app
TB      .set  14
RA      .set  10
RB      .set  14
        .include "C3XMMRS.ASM"
N       .set  512
N2      .set  N/2
PI      .set  3.1415926
PI2     .set  2*PI
PI2N    .set  2.0*PI/N
PIN     .set  PI/N
;=============================================================
; Create the Twiddle, FFT and I/O buffer arrays
;=============================================================
        .start  "TWIDDLES",0X809800 ;
        .sect   "TWIDDLES"          ;
TR                          ; 0x809800
        .loop   N2          ;
        .float  cos(br($-TR,N)*PIN);
        .endloop            ;
TI                          ;
        .loop   N2          ; 0x809900
        .float -1*sin(br($-TI,N)*PIN);
        .endloop            ;

BF      .set    TWIDDLES+512        ; 0x809A00
DR      .set    TWIDDLES+512        ; 0x809A00
DI      .set    TWIDDLES+1024       ; 0x809C00
```

```
;=============================================================
        .start  "FFTCODE",0x809E00  ; Start of RAM1
        .sect   "FFTCODE"           ;
;=============================================================
_STOP    .set  1
_START   .set  2
;
MSG_BOX  .word _STOP            ; 0x809E00
SPARE1   .word 0                ; 0x809E01
A_REG    .word (TA<<9)+(RA<<2)+0 ; 0x809E02
B_REG    .word (TB<<9)+(RB<<2)+2 ; 0x809E03
C_REG    .word 00000011b        ; 0x809E04 +/- 1.5 V
SPARE2   .word 0                ; 0x809E05
SIZE     .word N                ; 0x809E06
LOAD     .word 0xFFFFFFFF       ; 0x809E07 Mask for mantissa bits
;
;0_gctrl_val .word 0x0E973300 ; CLKR/X active low
S0_gctrl_val .word 0x0E970300 ; CLKR/X active high, use for higher speed DSP
S0_xctrl_val .word 0x00000111 ;
S0_rctrl_val .word 0x00000111 ;
;
FFTSIZE  .word N              ; 512 point FFT
TR_ADDR  .word TR             ; N/2 SIN values
TI_ADDR  .word TI             ; N/2 COS values
DR_ADDR  .word DR             ; N/2 REAL data point
DI_ADDR  .word DI             ; N/2 IMAG data point
BF_ADDR  .word BF             ; Array of Real buffer data
;-----------------------------
main     ldi  0x30,IE
         ldi  @S0_rdata,R0    ; Clear SP under/overflow
         ldi  0,R0            ;
```

```
        sti  R0,@S0_xdata   ;
        ldi  @S0_rdata,R0   ;
        ldi  0,R0           ;
        sti  R0,@S0_xdata   ;
        sti  R0,@RAMP       ;
        call GETADC         ; Flush first ADC value (might be trash)
        ;--------------------
        ldi  @DR_ADDR,AR0   ;
        ldi  @DI_ADDR,AR1   ;
        ldi  @SIZE,RC       ; Now get samples
        subi 1,RC           ; RC+1 repeats
        rptb samples        ;
        ;--------------------
        call GETADC         ;
        float R0,R0         ;
        stf  R0,*AR0++      ; store to data array
        ldf  0,R0           ;
samples stf  R0,*AR1++      ;
        ;-------------------- Put 0 into the DXR when it is not
        sti  R0,@S0_xdata   ; going to be used for awhile

        andn 0x30,IE        ; Turn off ADC interrupts
;=============================================================

; Perform FFT
; NOTE: This FFT is written for readability and size.
;     Other FFT programs which are substantialy faster
;     are given as examples in the users guide and
;     as downloadable programs from the BBS/FTP site.
;-------------------------------------------------------

FFT:    ldi  @FFTSIZE,IR0   ;
        ldi  @FFTSIZE,IR1   ;
        lsh  -1,IR0         ; IR0 = IR1/2 ... used to BR step twiddles
```

```
        ldi  @TR_ADDR,AR2    ; The twiddle base is modified using
        ldi  @TI_ADDR,AR3    ; bit-reversal and wraps around (circular)
New_Stg
        ldi  @DR_ADDR,AR0    ;
        ldi  @DI_ADDR,AR1    ;
        lsh  -1,IR1          ; decimate IR0 and IR1 (divide by two)
        lsh  -1,IR0          ;  Note: this does not set flags
        ldi  IR1,R0          ; Check if index is zero (end of FFT)
        bzd  FFT_END         ;
        ldi  @FFTSIZE,RC     ; Loop for FFTSIZE/2 butterflies
        lsh  -1,RC           ;  Note: Replace 3 lines with one load...
        subi 1,RC            ;
        ;---------------------
Blk_Top  rptb B_Fly          ; Start by getting all 6 Butterfly inputs
        ldf  *+AR0(IR1) ,R0 ; Bt real
   ||   ldf  *AR0       ,R1 ; Tp real
        ldf  *+AR1(IR1) ,R2 ; Bt imag
   ||   ldf  *AR1       ,R3 ; Tp imag
        ldf  *AR2++(IR0)B,R4 ; TW real - R4
   ||   ldf  *AR3++(IR0)B,R5 ; TW imag - R5

        addf3 R0,R1,R6       ; Top sum REAL
        addf3 *+AR1(IR1),R3,R7; Top sum IMAG
   ||   stf  R6,*AR0
  ;-------------------------------------------------------
  ;     The following line almost works except that the
  ;     operand ordering of the subf3 is not correct
  ;
  ;     subf3 *+AR0(IR1),R1,R6; R6=d_REAL  (R1 free)
  ; ||  stf  R7,*AR1         ;
    ;-------------------------------------------------------
```

```
    subf3 R0,R1,R6      ; R6=d_REAL  (R1 free)
    stf  R7,*AR1        ;

    subf3 R2,R3,R7      ; R7=d_IMAG  (R3 free)
    mpyf3 R6,R4,R1      ; R1 = R*TR = REAL_1
    mpyf3 R7,R5,R3      ; R3 = I*TI = REAL_2

    subf  R3,R1         ; R1 = R*TR - I*TI

    ldf  *++AR0,R1      ;
||  stf  R1,*+AR0(IR1)  ; Store bottom real

    mpyf3 R6,R5,R1      ; R1 = R*TI = IMAG_1
    mpyf3 R7,R4,R3      ; R3 = I*TR = IMAG_2
    addf  R3,R1         ;
    ldf  *++AR1,R1      ;
||  stf  R1,*+AR1(IR1)  ; Store bottom real
    ;---------------------------------------
    ; Identify EOB by twiddle wraparound
    ;---------------------------------------
    ldi  IR1,R7         ; save the index (test/load will corrupt it)
    cmpi @TR_ADDR,AR2   ; At E.O.B. Real Tw == start of Imag Tw
    ldinz 0,IR1         ;
    ldi  *AR0++(IR1),R0 ; Point R/I data pointers to next block
||  ldi  *AR1++(IR1),R0 ;
B_Fly   ldi  R7,IR1     ; Restore the index
    b    New_Stg        ; Exit is from top of routine
;=============================================================================
=

; When the FFT is complete a convolution with the response of the desired
; window function is used to clean up (filter) the otherwise non-windowed
; input samples.  In this case a raised cosine window is used since the
```

```
; filter coefficients are easily calculated as -0.5,1.0,-0.5.
;
;=============================================================================
=
FFT_END  ldi  @DI_ADDR,AR0   ; OUTPUT ptr
         ldi  @DR_ADDR,AR1   ; REAL ptr
         ldi  @DI_ADDR,AR4   ; IMAG ptr
         ldi  @SIZE,IR0      ; IR0=SIZE/2 for bit-reverse access
         lsh  -1,IR0         ;
         ldi  IR0,R0         ;
         subi 1,R0           ; To unroll the response for DC a bit
         or   IR0,R0         ; reversed add of the twos compliment of the
         addi R0,AR1         ; index is used
         addi R0,AR4         ;
         ;-------------------------------------
         ; AR0 = OUTPUT BUFFER = IMAG []
         ; AR1 = REAL[N-1] (unrolled to point to one sample before DC)
         ; AR4 = IMAG[N-1]
         ;=====================================================
         ldi  @SIZE,RC       ;
         lsh  -2,RC          ; Pack 4 results per word
         subi 1,RC
         rptb WINDOW         ;
         ;---------------------
         ldi  0,R7           ; Log_Mag returns with windowed R^2 + I^2
         call Log_Mag        ; packed into 8 MSBs of R0 which is then
         lsh  -24,R0         ; shifted and packed into 4 samples per word
         or   R0,R7          ;
         call Log_Mag        ;
         lsh  -16,R0         ;
         or   R0,R7          ;
         call Log_Mag        ;
```

```
        lsh  -8 ,R0        ;
        or   R0,R7         ;
        call Log_Mag       ; no shift this time
        or   R0,R7         ;
        lsh  -2,IR0        ;
        sti  R7,*AR0++(IR0)B ; store packed data in free IMAG[] slots
WINDOW  lsh  2,IR0         ;
        ;---------------------
BR_DATA ldi  @DI_ADDR,AR0  ; Convert Bit-Reverse IMAG[] -> Linear REAL[]
        ldi  @DR_ADDR,AR1  ;
        lsh  -2,IR0        ;
        ldi  *AR0++(IR0)B,R0 ;
        rpts 63            ;
        ldi  *AR0++(IR0)B,R0 ;
     || sti  R0,*AR1++     ;
        ;---------------------
        ldi  0x4,IE        ; Interlock with host only uses INT2
        ldi  _START,R0     ;
NO_START cmpi @MSG_BOX,R0  ; Restart when START message is received
        bnz  NO_START      ;
        ldi  _STOP,R0      ; Set MSG box to STOP
        sti  R0,@MSG_BOX   ;
        ;---------------------
        ldi  @LOAD,R2      ; Check to see if the host requested an
        bz   main          ; AIC reinitialization
        ldi  0,R2
        sti  R2,@LOAD
        call AIC_INIT      ; Restart with new AIC setup
        b    main          ; Do it all over again!
;=============================================================================
=
; Log_Mag: This function assumes that AR1,AR3 and IR0 have been
```

```
;       preinitialized to access the 'post convulationaly windowed'
;       FFT array.
;=============================================================
=
VU_scale .float 1/(N*128.0)    ; scale factor for FFT data growth
;-----------------
Log_Mag  ldf  -0.500,R5        ; 26 pixels/10 dB
      ;------------ 0.75000 V = 0 dBm
      mpyf3 R5,*AR1++(IR0)B,R0;
      push  AR1                ; save AR4 since this is the exit value
      addf   *AR1++(IR0)B,R0;
      mpyf3 R5,*AR1      ,R2;
      addf  R0,R2              ;
      mpyf  @VU_scale,R2       ; Unscale data growth and fixed pt ADC
      mpyf  R2,R2              ; REAL^2
      pop   AR1                ; restore ARx pointer
      ;------------
      mpyf3 R5,*AR4++(IR0)B,R0;
      push  AR4                ; save AR1 since this is the exit value
      addf   *AR4++(IR0)B,R0;
      mpyf3 R5,*AR4      ,R4;
      addf  R0,R4              ;
      mpyf  @VU_scale,R4       ;
      mpyf  R4,R4              ; IMAG^2
      pop   AR4                ; restore ARx pointer
      ;------------
      addf  R4,R2              ; REAL^2 + IMAG^2
      ;------------
      lsh   1,R2               ; Quick log using float equivelency
      pushf R2                 ; See Designer Notebook Page DNP-22
      pop   R0                 ;
      ;------------
```

```
        ash   -21,R0          ; Pack quick log number into 8 MSBs
        cmpi  -128,R0         ; preserving three mantissa bits
        ldile -128,R0         ; and clipping the result
        cmpi   127,R0         ;
        ldige  127,R0         ;
        lsh    24,R0          ; result is in 8 MSBs, 24 lsbs are 0
        ;- - - - - - - - - - -
        rets                  ;
;*************************************************************************
RAMP    .word 0
FLAGS   .word 0

GETADC  ldi  0x30,IE          ; Come here and wait for ADC interrupt
        IDLE                  ; confirmation to save power and code space
        ldi  @FLAGS,R0        ;
        tstb 0x20,R0          ;
        bz   $-3              ;
        andn 0x20,R0          ;
        sti  R0,@FLAGS        ;
        ldi  @S0_rdata,R0     ; Return sign extended ADC value
        lsh  16,R0            ;
        ash  -16,R0           ;
        rets

ADC     push ST               ; On interrupt, set a software flag to
        push R0               ; let the CPU know that the RINT has been
        ldi  @S0_rdata,R0     ;
        ldi  @FLAGS,R0        ;
        or   0x20,R0          ;
        sti  R0,@FLAGS        ;
        pop  R0               ;
        pop  ST               ;
```

```
        reti            ;

DAC     push ST         ;
        push R1         ;
        ldi  @RAMP,R1   ; Send RAMP signal out DAC for loopback test
        subi 1024,R1    ;
        lsh  17,R1
        ash  -17,R1
        andn 3,R1       ;
        sti  R1,@RAMP   ;
        sti  R1,@S0_xdata   ; loopback ADC->DAC
        pop  R1
        pop  ST
        reti
;-------------------------------
prog_AIC push R1        ;
        push IE         ;
        ldi  0x10,IE    ;
        andn 0x30,IF    ;
        ldi  @S0_xdata,R1   ; Use original DXR data during 2 ndy
        sti  R1,@S0_xdata   ;
        idle
        ldi  @S0_xdata,R1   ; Use original DXR data during 2 ndy
        or   3,R1       ; Request 2 ndy XMIT
        sti  R1,@S0_xdata   ;
        idle            ;
        sti  R0,@S0_xdata   ; Send register value
        idle            ;
        andn 3,R1       ;
        sti  R1,@S0_xdata   ; Leave with original safe value in DXR
        pop  IE         ;
        pop  R1         ;
```

```
        rets            ;
;===============================================================;
; This section of code is called by the initialization ;
; code as well as by the main program loop.  It is    ;
; therfor assembled into the regular program RAM      ;
;===============================================================;
AIC_INIT  push  R0          ;
        LDI   0x10,IE       ; Enable XINT interrupt
        andn  0x34,IF       ;
AIC_reset
        ldi   0,R0          ;
        sti   R0,@S0_xdata  ;
        RPTS  0x040         ;
        LDI   2,IOF         ; XF0=0 resets AIC
        rpts  0x40          ;
        LDI   6,IOF         ; XF0=1 runs AIC
        ldi   @S0_rdata,R0
        ldi   0,R0
        sti   R0,@S0_xdata
        ;-----------------------------
        ldi   @C_REG,R0     ; Setup control register
        call  prog_AIC      ;
        ldi   0xfffc ,R0    ; Program the AIC to be real slow
        call  prog_AIC      ;
        ldi   0xfffc|2,R0   ;
        call  prog_AIC      ;
        ldi   @B_REG,R0     ; Bump up the Fs to final rate
        call  prog_AIC      ; (smallest divisor should be last)
        ldi   @A_REG,R0     ;
        call  prog_AIC      ;
        pop   R0            ;
        ldi   0,R0          ; Put a safe 0 in DXR
```

```
        sti  R0,@S0_xdata   ;
        ldi  @S0_rdata,R0   ; Clear receive underrun
        rets                ;
;*****************************************************;
; Startup stub...                                     ;
;                                                     ;
; The following section of code is used only once for ;
; initialization and can be safely overwritten by     ;
; assembling it into the stack or volatile data       ;
; storage.                                            ;
;*****************************************************;
        .start  "STUB",BF   ; Place this code in the data buffer
        .sect   "STUB"      ; area as this is the first to go
        .entry  ST_STUB
ST_STUB ldp  T0_ctrl        ; Use kernel data page and stack
        ldi  0,R0           ; Halt TIM0 & TIM1
        sti  R0,@T0_ctrl    ;
        sti  R0,@T1_ctrl    ;
        sti  R0,@T0_count   ; Set counts to 0
        sti  R0,@T1_count   ;
        ldi  1,R0           ; Set periods to 1
        sti  R0,@T0_prd     ;
        sti  R0,@T1_prd     ;
        ldi  0x2C1,R0       ; Restart both timers
        sti  R0,@T0_ctrl    ;
        sti  R0,@T1_ctrl    ;
        ;---------------------
        ldi  @S0_xctrl_val,R0;
        sti  R0,@S0_xctrl   ; transmit control
        ldi  @S0_rctrl_val,R0;
        sti  R0,@S0_rctrl   ; receive control
        ldi  0,R0           ;
```

```
        sti   R0,@S0_xdata    ; DXR data value
        ldi   @S0_gctrl_val,R0; Setup serial port
        sti   R0,@S0_gctrl    ; global control
        ;---------------------
        call  AIC_INIT        ; Initialize the AIC
        ldi   0x30,IE         ; Service both RINT/XINT
        ldi   @S0_rdata,R0    ;
        b     main            ;
;*************************************************;
; Install the XINT/RINT ISR handler directly into  ;
; the vector RAM location it will be used in       ;
;*************************************************;
        .start  "SP0VECTS",0x809FC5
        .sect   "SP0VECTS"
        B     DAC       ; XINT0
        B     ADC       ; RINT0
;*****************************************************;
; The folowing section is for C3x devices which are configured;
; in the u-processor mode (eg EVM or XDS510) and there for have;
; RAM instead of bootrom at the vector table address.  The    ;
; DSK loader will also write to this address but the bootrom  ;
; is obviously not modifiable...                              ;
;*****************************************************;
        .start  "ROMVECTS",0x000000
        .sect   "ROMVECTS"
        .word   0x809FC0  ; RESET
        .word   0x809FC1  ; INT0
        .word   0x809FC2  ; INT1
        .word   0x809FC3  ; INT2
        .word   0x809FC4  ; INT3
        .word   0x809FC5  ; XINT0
        .word   0x809FC6  ; RINT0
```

```
.word   0x809FC7  ; XINT1  NA
.word   0x809FC8  ; RINT1  NA
.word   0x809FC9  ; TINT0
.word   0x809FCA  ; TINT1
.word   0x809FCB  ; DINT
```

```
        .include "C3XMMRS.ASM"
TA      .set  12                ; Use AIC startup == host side app
TB      .set  14
RA      .set  12
RB      .set  14
N       .set  1024
N2      .set  N/2
N4      .set  N/4
PI      .set  3.141592654
PIN     .set  PI/N
;=============================================================
; Create the Twiddle, FFT and I/O buffer arrays
;=============================================================
DATA_ARRAY .set  0x809800
TWID_ARRAY .set  0x809C00

        .start  "TWIDDLES",TWID_ARRAY
        .sect   "TWIDDLES"
;-------------------------------------------
; Set F16 to 1 for a [8:1: 7] float format
; -or-      0 for a [4:1:11] float format
;-------------------------------------------
F16     .set    0  ;<- Change this value
;-------------------------------------------
 .if F16

TW
TWSCALE .set    1.0
Cnt     .sdef   0.0
        .loop   N4
        .pfloat16 cos(Cnt*PIN),-sin(Cnt*PIN)
```

```
Cnt     .sdef    br($-TW,N)
        .endloop

        .loop    N4
        .pfloat16 cos(Cnt*PIN),-sin(Cnt*PIN)
Cnt     .sdef    br($-TW,N)
        .endloop

  .else
;-------------------------------------------------------
; 1) An sdef definition can eliminate some calculations,
;   speeding up DSK3A
; 2) Each code loop is limited to 256x to prevent accidentally
;   overwriting the entire hard drive space
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;       .loop    N4
;       .psfloat  TWSCALE*cos(Cnt),-TWSCALE*sin(Cnt)
;Cnt     .sdef    br($-TW,N)*PIN
;       .endloop
;-------------------------------------------------------
TW
TWSCALE .set     256.0

Cnt     .sdef    0.0
        .loop    N4      ; Loop is limited to 256 times...
        .psfloat  TWSCALE*cos(br($-TW,N)*PIN),-TWSCALE*sin(br($-TW,N)*PIN)
        .endloop
        .loop    N4      ; loop next 256 times
        .psfloat  TWSCALE*cos(br($-TW,N)*PIN),-TWSCALE*sin(br($-TW,N)*PIN)
        .endloop
  .endif
        .start  "FFTCODE",0x809E00  ; 256 words before comm kernel
```

```
        .sect  "FFTCODE"        ;
;=============================================================
MSG_BOX  .word  0               ; 0x809E00
LOAD     .word  1               ; 0x809E01
A_REG    .word  (TA<<9)+(RA<<2)+0 ; 0x809E02
B_REG    .word  (TB<<9)+(RB<<2)+2 ; 0x809E03
C_REG    .word  00000011b       ; 0x809E04  +/- 1.5 V
FFTSIZE  .word  N               ; 0x809E05
FFTSIZE2 .word  N/2             ; 0x809E05
;
TW_ADDR  .word TWID_ARRAY       ; IMAG/REAL twiddle values
DR_ADDR  .word DATA_ARRAY       ; REAL/IMAG data
DR_ADDR2 .word DATA_ARRAY+256   ; swap array used for inplace bit reversal


;-------------------------------------
; pfloat16 pack/unpack
;  pack F6:F7 into F6
;  unpack R4 into F4:F5
;-------------------------------------
 .if F16
Min_sf   .float 1.0             ;
DMASK    .word 0xFF000000       ; chop down to 8 bit mantissa
Pack     lsh  -24,R6            ; clear 16 lsbs
         lsh  24,R6             ;
         pushf R7               ; move to integer field
         pop  R7                ;
         lsh  -8,R7             ; move to correct position
         or   R7,R6             ; concatenate
         pushf R6
         pop  R6
         rets                   ;
;-------------------------------------
```

```
unpack   ldi   R4,R5        ;
         push  R4           ; Convert to 32 bit float
         popf  R4           ;
         lsh   16,R5        ;
         push  R5           ;
         popf  R5           ;
         and   @DMASK,R4    ;
         and   @DMASK,R5    ;
         rets               ;
 .else
;------------------------------------ 11111000
; pack F6:F7 into pfloat16 R6
;------------------------------------

TMASK    .word 0xFFF00000    ; chop down to 12 bit mantissa
Max_sf   .word 0x077FF000    ; max magnitude for 4:1:11 format
Min_sf   .word 0xF8000001    ; min magnitude for 4:1:11 format
;
Clip_sf  cmpf  0,R7
         ldfge 1,R5
         ldflt -1,R5
         absf  R7,R7
         cmpf  @Max_sf,R7
         ldfge @Max_sf,R7
         cmpf  @Min_sf,R7
         ldflt @Min_sf,R7
         mpyf  R5,R7
         rets
         ;------------
Pack     pushf R5
         pushf R7
         ldf   R6,R7
```

```
        call  Clip_sf
        ldf   R7,R6
        popf  R7
        call  Clip_sf
        pushf R6            ; clear 16 lsbs
        pop   R6            ;
        lsh   -12,R6        ; move to upper 16 bits (lsbs = 0)
        lsh   16,R6         ;
        pushf R7            ;
        pop   R7            ; move to lower 16 bits
        lsh   4,R7          ;
        lsh   -16,R7        ; concatenate
        or    R7,R6         ;
        popf  R5            ;
        rets                ;
        ;-----------
unpack  ldi   R4,R5
        ash   -4,R4         ; Sign extend the exponent
        push  R4            ; Convert to 32 bit float
        popf  R4            ;
        lsh   16,R5         ;
        ash   -4,R5         ;
        push  R5            ;
        popf  R5            ;
        and   @TMASK,R4
        and   @TMASK,R5
        rets                ;
    .endif
;-------------------------------------
TSCALE  .float 1.0/TWSCALE
main    ldi   0x30,IE       ;
        ldi   0,R0          ; Clear flags,DXR etc...
```

```
        sti  R0,@MSG_BOX    ;
        sti  R0,@LOAD       ;
        sti  R0,@S0_xdata   ;
        sti  R0,@S0_xdata   ;
        sti  R0,@FLAGS      ;
        idle                ; Flush first ADC value (might be trash)
        ;--------------------
        ldi  @DR_ADDR,AR0   ;
        ldi  @FFTSIZE,RC    ; Now get samples
        subi 1,RC           ; RC+1 repeats
        rptb samples        ;
        ;--------------------
        call GETADC         ; Get ADC value
        float R0,R6         ; convert to float
        mpyf @Min_sf,R6
        ldf  0,R7           ; pack ADC:0.0 and store in R:I data
        call Pack           ;
samples sti  R6,*AR0++      ;
        ;--------------------
        ldi  0,R0           ; Put 0 into the DXR when it is not
        sti  R0,@S0_xdata   ; going to be used for awhile
        ldi  0,IE           ; Turn off all (ADC) interrupts
;=============================================================================
; Perform FFT
;
; NOTE: This FFT is written with the intent of fitting the
;       maximum size possible on chip.  To do this the FFT relies
;       on data packing and unpacking routines which substantialy
;       slow down the FFT.  However since the host cannot typicly
;       transfer and display the results as fast as the DSP the
;       impact is not much.
;-----------------------------------------------------------------------------
```

```
FFT:    ldi  @FFTSIZE2,IR0   ;
        ldi  @FFTSIZE ,IR1   ; IR0 = IR1/2 ... used to BR step twiddles
        ldi  @TW_ADDR,AR2    ; The twiddle base is modified using
                             ; bit-reversal and wraps around (circular)
New_Stg ldi  @DR_ADDR,AR0    ;
        lsh  -1,IR1          ; decimate IR0 and IR1 (divide by two)
        lsh  -1,IR0          ; Note: this does not set flags
        ldi  IR1,R0          ; Check if index is zero (end of FFT)
        bzd  FFT_END         ;
        ldi  @FFTSIZE2,RC    ; Loop for FFTSIZE/2 butterflies
        subi 1,RC            ;
        ;---------------------
Blk_Top rptb B_Fly           ; Start by getting all 6 Butterfly inputs
        ;
        ldi  *AR0,R4         ; Top R/I pair
        call unpack          ; Unpack REAL/IMAG data to R2,R3
        ldf  R4,R1           ;
        ldf  R5,R3           ;
        ;
        ldi  *+AR0(IR1),R4   ; Bottom R/I pair
        call unpack          ; Unpack REAL/IMAG data to R2,R3
        ldf  R4,R0           ;
        ldf  R5,R2           ;
        ;
        ldi  *AR2++(IR0)B,R4 ; Load and unpack twiddles
        call unpack          ; NOTE: Since R4:R5 pair is used by FFT
        mpyf @TSCALE,R4
        mpyf @TSCALE,R5

        ; ldf  R4,R4         ;    R4:R5 copy is not required
        ; ldf  R5,R5         ;
```

```
    addf3 R0,R1,R6      ; Top sum REAL
    addf3 R2,R3,R7      ; Top sum IMAG
.if F16
.else
    mpyf  0.5,R6
    mpyf  0.5,R7
.endif
    call  Pack          ; Pack R6:R7
    sti   R6,*AR0       ; store Top REAL/IMAG

    subf3 R0,R1,R6      ; R6 = Rt-Rb        (R1 free)
    subf3 R2,R3,R7      ; R7 = It-Ib        (R3 free)
    mpyf3 R6,R4,R1      ; R1 = R*TR
    mpyf3 R7,R5,R3      ; R3 = I*TI
    subf  R3,R1         ; R1 = R1-R3 = R*TR-I*TI
    ldf   R1,R0
    ;
    mpyf3 R6,R5,R1      ; R1 = (Rt-Rb)*TI
    mpyf3 R7,R4,R3      ; R3 = (It-Ib)*TR
    addf3 R3,R1,R7      ; R1 = (Rt-Rb)*TI + (It-Ib)*TR
    ;
    ldf   R0,R6         ;
   ; ldf   R1,R7        ;
.if F16
.else
    mpyf  0.5,R6
    mpyf  0.5,R7
.endif
    call  Pack          ;
    sti   R6,*+AR0(IR1) ; Store Bottom REAL/IMAG

    ldi   *++AR0,R0     ; Point to next data address
```

```
        ;---------------------------------------
        ; Identify EOB by twiddle wraparound
        ;---------------------------------------
        ldi  IR1,R7          ; save the index (test/load will corrupt it)
        cmpi @TW_ADDR,AR2    ; At end of block the twiddle pointers will
        ldinz 0,IR1          ; have completed the bit-rev circular address
        ldi  *AR0++(IR1),R0  ; Point R/I data pointers to next block
B_Fly   ldi  R7,IR1          ; Restore the index
        b    New_Stg         ; Exit is from top of routine
;=============================================================================
=
; When the FFT is complete a convolution with the response of the desired
; window function is used to filter (clean up) the non-windowed FFT. A
; raised cosine window is used since the coefficients are -0.5,1.0,-0.5.
; After filtering, the R^2 + I^2 magnitude is calculated and packed into
; 4 log scaled bytes for display on the host
;
;  AR0 = OUTPUT BUFFER = IMAG []
;  AR1 = REAL[N-1] (unrolled to point to one sample before DC)
;  AR4 = IMAG[N-1]
;  IR0 = FFTsize/2
;=============================================================================
=
FFT_END
        ldi  @DR_ADDR,AR0    ; OUTPUT ptr
        ldi  @DR_ADDR,AR4    ; REAL/IMAG ptr
        ldi  @FFTSIZE2,IR0   ; IR0=SIZE/2 for bit-reverse access
        ldi  IR0,R0          ;
        subi 1,R0            ; To unroll the response for DC a bit
        or   IR0,R0          ; reversed add of the twos compliment of the
        addi R0,AR4          ; index is used
Log_Mag ldi  @FFTSIZE,RC     ;
```

```
    lsh  -3,RC        ; Cvrt 1/2 data array at 4 points/loop
    subi 1,RC         ;
    rptb WINDOW       ;
    ;- - - - - - - - - - -
    ldi  3,AR7        ; AR7 for 4 X loop counter
Loop_LM
    ldi  *AR4++(IR0)B,R4 ; Load three packed R/I pairs into Regs
    call unpack
    ldf  R4,R0
    ldf  R5,R1

    push AR4          ; Save AR4 of mid value for next loop

    ldi  *AR4++(IR0)B,R4 ;
    call unpack
    ldf  R4,R2
    ldf  R5,R3

    ldi  *AR4++(IR0)B,R4 ;
    call unpack
;   ldf  R4,R4
;   ldf  R5,R5

    pop  AR4          ;

    addf R0,R4        ; Perform convolution window on REAL data
    mpyf -0.5,R4      ;
    addf R2,R4        ;
    mpyf @VU_scale,R4 ; Scale FFT data growth
    mpyf R4,R4        ; REAL^2

    addf R1,R5        ; Perform convolution window on IMAG data
```

```
        mpyf -0.5,R5        ;
        addf R3,R5          ;
        mpyf @VU_scale,R5   ;
        mpyf R5,R5          ; IMAG^2
        ;------------
        addf3 R4,R5,R2      ; REAL^2 + IMAG^2
        cmpf @MaxL,R2       ; Clip exponent magnitude to 2^16 & 2^-16
        ldfgt @MaxL,R2      ;
        cmpf @MinL,R2       ;
        ldflt @MinL,R2      ;
        lsh 1,R2            ; Quick log using float equivelency
        pushf R2            ; See Designer Notebook Page DNP-22
        pop R0              ;
        ;------------
        lsh -8,R7           ; OR into result
        lsh -21,R0          ;
        lsh 24,R0           ;
        or R0,R7            ;
        dbu AR7,Loop_LM     ; Loop until four samples are packed
        lsh -2,IR0          ;
        sti R7,*AR0++(IR0)B ; store packed data in free IMAG[] slots
WINDOW  lsh 2,IR0           ;
        ;--------------------
        lsh -2,IR0          ;

BR_DATA ldi @DR_ADDR,AR0    ; Convert Bit-Reverse IMAG[] -> Linear REAL[]
        ldi @DR_ADDR2,AR1   ;
      ; addi 256,AR1        ;
        ldi *AR0++(IR0)B,R0 ;
        rpts 127            ;
        ldi *AR0++(IR0)B,R0 ;
     || sti R0,*AR1++       ;
```

```
      ldi  @DR_ADDR2,AR0  ; Move data back to bottom of data array
      ldi  @DR_ADDR,AR1   ;
     ; addi  256,AR0       ;


      ldi  *AR0++,R0      ;
      rpts  127           ;
      ldi  *AR0++,R0      ;
   || sti  R0,*AR1++      ;
      ;--------------------
      ldi  0x4,IE         ; Interlock with host only uses INT2
NO_START ldi  @MSG_BOX,R0   ; Restart when START message is received
      bz   NO_START       ;
      ldi  @LOAD,R2       ; Check to see if the host requested an
      bz   main           ; AIC reinitialization
      call  AIC_INIT      ; Restart with new AIC setup
      b    main           ; Do it all over again!
;*****************************************************************************
MaxL  .float  65535.0       ; Clip to 2^16 and 2^-16
MinL  .float  1/65536.0     ;

 .if  F16
VU_scale  .float  1/(N*128.0)  ; FFT data growth factor and -128 exp offset
 .else
VU_scale  .float  1.0       ; -128 exp offset
 .endif
FLAGS    .word 0            ;
;-----------------
GETADC   ldi  0x30,IE       ; Come here and wait for ADC interrupt
         IDLE               ; confirmation to save power and code space
         ldi  @FLAGS,R0     ;
```

```
      bz   GETADC       ;
      ldi  0,R0         ;
      sti  R0,@FLAGS    ;
      ldi  @S0_rdata,R0 ; Return sign extended ADC value
      lsh  16,R0        ;
      ash  -15,R0       ;
      rets              ;
      ;-----------
ADC   push ST           ; On interrupt, set a software flag to
      push R0           ; let the CPU know that the RINT has been
      ldi  @S0_rdata,R0 ;
      ldi  1,R0         ;
      sti  R0,@FLAGS    ;
      b    DACRET
      ;-----------
DAC   push ST           ;
      push R0           ;
      ldi  0,R0         ;
      sti  R0,@S0_xdata ; RAMP sent to DAC
DACRET pop R0           ;
      pop  ST           ;
      reti              ;
;-------------------------------
prog_AIC andn 0x30,IF   ;
      ldi  0,R1         ;
      sti  R1,@S0_xdata ;
      idle              ;
      ldi  3,R1         ; Request 2 ndy XMIT
      sti  R1,@S0_xdata ;
      idle              ;
      sti  R0,@S0_xdata ; Send register value
      idle              ;
```

```
      andn  3,R1          ;
      sti   R1,@S0_xdata  ; Leave with original safe value in DXR
      rets              ;
;=============================================================;
; This section of code is called by the initialization ;
; code as well as by the main program loop.  It is    ;
; therfor assembled into the regular program RAM      ;
;=============================================================;
AIC_INIT  LDI   0x10,IE       ; Enable XINT interrupt
      andn  0x34,IF       ;
AIC_reset
      ldi   0,R0          ;
      sti   R0,@S0_xdata  ;
      RPTS  0x040         ;
      LDI   2,IOF         ; XF0=0 resets AIC
      rpts  0x40          ;
      LDI   6,IOF         ; XF0=1 runs AIC
      ldi   @S0_rdata,R0
      ldi   0,R0
      sti   R0,@S0_xdata
      ;-----------------------------
      ldi   0xfffc ,R0    ; Program the AIC to be real slow
      call  prog_AIC      ;
      ldi   0xfffc|2,R0   ;
      call  prog_AIC      ;
      ldi   @C_REG,R0     ; Setup control register
      call  prog_AIC      ;
      ldi   @B_REG,R0     ; Bump up the Fs to final rate
      call  prog_AIC      ; (smallest divisor should be last)
      ldi   @A_REG,R0     ;
      call  prog_AIC      ;
      ldi   0,R0          ; Put a safe 0 in DXR
```

```
        sti  R0,@S0_xdata   ;
        ldi  @S0_rdata,R0   ; Clear receive underrun
        rets                ;
;*************************************************;
; Startup stub...                                 ;
;                                                 ;
; The following section of code is used only once for ;
; initialization and can be safely overwritten by    ;
; assembling it into the stack or volatile data      ;
; storage.                                           ;
;*************************************************;
        .start  "STUB",DATA_ARRAY
        .sect   "STUB"      ; Place this code in the data buffer
        .entry  ST_STUB     ; area as this is the first to go
ST_STUB ldp  T0_ctrl        ; Use kernel data page and stack
        ldi  1,R0           ; Set periods to 1
        sti  R0,@T0_prd     ;
        sti  R0,@T1_prd     ;
        ldi  0,R0           ; Halt TIM0 & TIM1
        sti  R0,@T0_ctrl    ;
        sti  R0,@T1_ctrl    ;
        sti  R0,@T0_count   ; Set counts to 0
        sti  R0,@T1_count   ;

                    ;               1 2 3 4 5
        ldi  0x2C1,R0       ; Pulse md clock     12, 6, 4, 3, 1.4 Mhz
;       ldi  0x3C1,R0       ; Clock md (bi-quinry) 12, 6, 3, 2, 1  Mhz
                    ;               0 1 2 3 4


        sti  R0,@T0_ctrl    ;
        sti  R0,@T1_ctrl    ;
        ;--------------------
```

```
      ldi  @S0_xctrl_val,R0;
      sti  R0,@S0_xctrl   ; transmit control
      ldi  @S0_rctrl_val,R0;
      sti  R0,@S0_rctrl   ; receive control
      ldi  0,R0           ;
      sti  R0,@S0_xdata   ; DXR data value
      ldi  @S0_gctrl_val,R0; Setup serial port
      sti  R0,@S0_gctrl   ; global control
      ;---------------------
      call AIC_INIT       ; Initialize the AIC
      ldi  0x30,IE        ; Service both RINT/XINT
      ldi  @S0_rdata,R0   ;
      b    main           ;
;0_gctrl_val .word 0x0E973300 ; CLKR/X active low
S0_gctrl_val .word 0x0E970300 ; CLKR/X active high, use for higher speed DSP
S0_xctrl_val .word 0x00000111 ;
S0_rctrl_val .word 0x00000111 ;

;*****************************************************;
; Install the XINT/RINT ISR handler directly into     ;
; the vector RAM location it will be used in          ;
;*****************************************************;
      .start  "SP0VECTS",0x809FC5
      .sect   "SP0VECTS"
      B    DAC        ; XINT0
      B    ADC        ; RINT0
```

DSK.H

```
#define XWRIT    1
#define XREAD    2
#define XCTXT    3
#define XRUNF    4
#define XSSTEP   5
#define XHALT    6
#define XW_HOST  7  /* Not really a host based command     */
#define XR_HOST  8  /* Not really a host based command     */
#define XSPARE   9  /* Need to add your own vector for this */

MSGS putmem  (ulong addr, ulong length, ulong *data);
MSGS getmem  (ulong addr, ulong length, ulong *data);
MSGS SSTEP_CPU    (void);
MSGS RUN_CPU      (void);
MSGS HALT_CPU     (void);
MSGS GET_DEBUG_CTXT(void);
extern int swap_en;
//
// Prototypes and declarations for DRIVER.CPP
//
extern uint  port  ;
extern uint  status;
extern uint  ctrl  ;
extern ulong WSHIFT ;
extern ulong WSCOUNT;
extern int   timeout;
extern int   test_flag;
extern char huge Help_Msg[];
extern char  LO_PWR;

void   whirly(int x,int y);
```

```
void  HPI_STRB(int val);
char  HPI_ACK(void);
MSGS  DSK_reset(void);
MSGS  Interlock(void);
MSGS  recv_long(ulong *);
MSGS  xmit_byte(char);
MSGS  xmit_long(ulong);
void  Pulse_Init(void);
//
// C31 DSK interface and parallel printer port signal definitions
// Bit definitions at printer control port (write)
//  B7    B6    B5   B4    B3      B2      B1      B0
// +-----------------+-----------+--------+--------+--------+
//| DIR |  x  | DIR | INT |/SLCTIN|  INIT  |/AUTOFEED| /STROBE |
// +-----------------+-----------+--------+--------+--------+
//                        RESET          HPSTB
//             W    R/W    W    R/W       W       R/W
//
// Bit definitions at printer status port (read)
//  B7       B6       B5       B4      B3    B2   B1   B0
// +---------+---------+---------+--------+-----------------+
//| /BUSY   | ACK    | PAPER   | SELECT |ERROR| ACK |  x  |  x |
// +---------+---------+---------+--------+-----------------+
// <----------------- D0 - D3 ---------------> HPACK
extern int _DIR;       // NOTE: Some laptops use these bits for shutdown!
#define PPSTRB_LO 0x5
#define PPSTRB_HI 0x4
#define RESET_LO 0x0   /* RESET is lo, PSTRB hi */
#define RESET_HI 0x4   /* RESET is hi, PSTRB hi */
#define outctrl(val)  outportb(ctrl,val)
#define instat        inportb(status)
#define inbyte        inportb(port)
```

```
#define innible      ((inportb(status) >> 4) ^ 0x8)
#define outbyte(val)  outportb(port,val)
#define inctrl       inportb(ctrl)
//
// The following enumeration matches the context save structure used
// in the communications kernel.  By downloading the context save area
// to PC memory (int the array CTXT[]) programs can have access to the
// DSP's registers.  By then modifying and reloading these values
// the user can modify the runtime parameters.
//
typedef enum { R0F, R1F, R2F, R3F, R4F, R5F, R6F, R7F, // R8i-11i  0- 7
        R0 , R1 , R2 , R3 , R4 , R5 , R6 , R7 , // R8F-11F  8-15
        AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7, //       16-24
        DP , IR0, IR1, BK , SP , ST , IE , IF , //       25-31
        IOF, RS ,  RE,  RC, PC , FREERUN, T1DIF, CNTXT //  32-38
       } TMS_REGS;

#define CTXTSIZE 39
extern ulong CTXT[];
//
// Prototypes and declarations for OBJECT.CPP
//
extern ulong ENTRY;
extern ulong PC_Appli;
extern ulong DEBUG_CTXT;
extern int  WINDOWS ;
extern int  timeout ;
extern int  BW_force;
extern int  Windows_Detected;
extern int  DSK3D;   // Global flag for low level code
extern int  No_MTask;
```

```
int  ports        (int base);
void enable_Mtask  (void);
void disable_Mtask (void);
void Release_TSlice(void);
MSGS Detect_Windows(void);
MSGS Scan_Command_line(char *app_name);
MSGS Display_manual   (char     *name);
MSGS View_Manual      (char     *name);
//
// Edit the hardware manual to have the name of the app and options
void Edit_Help_Msg(char *app_name, char *sw1, char *desc);
MSGS Init_Communication(int loops);
MSGS get_buswidth  (void);
MSGS set_buswidth  (void);
MSGS Disp_Menu(char *banner, char huge *msg,int,int,int,int);

#define FRAME2 "ษปอศผบ" /* Double bar frame  */
#define FRAME1 "ฺฟฤฎฌ" /* Single bar frame  */
#define FRAME0 "      " /* Plain text window */

extern float DSK3_rev; // Global version variable available to all apps

#endif
```

DSK_COFF.H

```
#include "typedefs.h"
#include "errormsg.h"
#define MAGIC_FL   0x93  /* C3x/C4x    */
#define MAGIC_FX   0x92  /* C1x/C2x/C5x */
#define MAXSECTIONS  64
extern  char  section_name[];
extern  char  DASM_file[];  // Current loaded DSK filename
extern  char  HLL_file[];   // High Level Language file (C source)
extern  ulong section_start;
extern  ulong section_offs;
extern  int   CoffVer;      // COFF version of last loaded COFF file
extern  char  *DSKEXT;
extern  char  *OUTEXT;

typedef enum
{
 LOAD,      // Loads a file to the DSK target
 BOOT,      // Bootloads a file to the DSK target
 FILE2HEX,  // Writes bootable hex file to file.hex
 BOOTHEX,   // Writes loadable hex file to file.hex
 LOADHEX,   // Load/Boot hex files. Line 1 sets mode
 DSK2COFF,  // Convert ASCII DSK file to Binary COFF file
 MEM2COFF,  // Convert mem block to COFF file
 MEM2HEX,   // Convert mem block to HEX file
 SLOAD      // Load only symbols from file
}TASK;

void     Symbol_Clear(int level);
MSGS Save_File(char *file,ulong start, ulong length,TASK task);
MSGS Load_File(char *fileptr, TASK task);
MSGS LOADCOFF (char *fileptr, TASK task);
```

```
MSGS LOADDSKA (char *fileptr, TASK task);
MSGS CVT2COFF (char *fileptr, ulong start, ulong length, TASK task);
MSGS Load_Hex_File(char *filename, TASK task);

typedef struct {
 ushort mgc;    // 2 coff magic number 0x93 (0x92 for fixed pt DSP)
 ushort sect;   // 2 number of sections in file
 ulong  date;   // 4 time and date stamp
 ulong  symb;   // 4 file pointer to symbol table
 ulong  nsym;   // 4 number of symbols in symbol table
 ushort optn;   // 2 length of optional header
 ushort flag;   // 2 Flags
}FILE_HDR0;      // 20 bytes long

typedef struct {
 ushort mgc;    // 2 coff magic number 0x93 (0x92 for fixed pt DSP)
 ushort sect;   // 2 number of sections in file
 ulong  date;   // 4 time and date stamp
 ulong  symb;   // 4 file pointer to symbol table
 ulong  nsym;   // 4 number of symbols in symbol table
 ushort optn;   // 2 length of optional header
 ushort flag;   // 2 Flags
 ushort mgc2;   // 2 COFF version 1,2,3,4... use this field
}FILE_HDR1;

typedef struct {
 ushort   mgc; // options magic number 0x108
 ushort   vers; // version stamp
 ulong  text_sz; // size (in words) of .text
 ulong  data_sz; // size (in words) of .data
 ulong   bss_sz; // size (in bits) of .bss
 ulong   entry; // entry point address
```

```
 ulong  text_ad;  // beginning address of .text
 ulong  data_ad;  // beginning address of .data
}OPTN_HDR;

typedef struct {   // +----------Value is zero if unitilized section
 char   name[8];  // | 8 Section name
 ulong  radd;     // | 4 runtime address
 ulong  vadd;     // | 4 virtual address (load address)
 ulong  size;     // | 4 section size
 ulong  fptr;     // 0 4 file pointer to raw data
 ulong  rent;     // 0 4 file pointer to relocation entries
 ulong  lent;     // 0 4 file pointer to line number entries
 ushort nrel;     // 0 2 number of relocation entries
 ushort nlin;     // 0 2 number of line number entries
 ushort flag;     //   2 flags
 uchar  resv;     //   1 reserved
 uchar  page;     //   1 memory page number
}SECT_HDR0;       //  40 bytes long

typedef struct {   // +----------Value is zero if unitilized section
 char   name[8];  // | 8 Section name
 ulong  radd;     // | 4 runtime address
 ulong  vadd;     // | 4 virtual address (load address)
 ulong  size;     // | 4 section size
 ulong  fptr;     // 0 4 file pointer to raw data
 ulong  rent;     // 0 4 file pointer to relocation entries
 ulong  lent;     // 0 4 file pointer to line number entries
 ulong  nrel;     // 0 4 number of relocation entries
 ulong  nlin;     // 0 4 number of line number entries
 ulong  flag;     //   4 flags
 short  resv;     //   2 reserved
 ushort page;     //   2 memory page number
```

```
}SECT_HDR1;      // 48 bytes long

typedef struct {
 char  sname[8]; // 8 char name, or ptr to strg tbl
 long  svalu;   // symbol value
 int   sectn;   // section number of symbol
 uint  stype;   // symbol type
 char  sclass;  // storage class
 char  s_aux;   // number of auxiliary entries (0 or 1)
}SYMB_TBL;
//-------------------------------------------------------
//typedef struct {
// long  sleng;   //
// uint  sreln;   //
// uint  sline;   //
// char  garb[10]; // not used
//}AUX_STBL;
//=============================================================
//======== Auxiliary Header definitions by Symbol Class ========
//=============================================================
 typedef struct {     // File Name format
  char  fname[14];  // File name
  char  nused[ 4];  // Not used
 }A_18;             //
 //- - - - - - - - - - - - - - - - - - - - - - - - -
 typedef struct {     // Section format
  long  sleng;      // length of section to follow
  uint  Nreloc;     // number of relocation entries
  uint  Nlines;     // number of line entries
  char  nused[10];  //
 }A_19;             //
 //- - - - - - - - - - - - - - - - - - - - - - - - -
```

```
typedef struct {     // Tag name format
 char  nuse1[6];   //
 uint  SzStrct;    //
 char  nuse2[4];   //
 long  NxtIndex;   //
 char  nuse3[2];   //
}A_20;             //
//----------------------------
typedef struct {     // End of Structure
 long  TagIndex;   //
 char  nuse1[2];   //
 uint  SzStrct;    //
 char  nuse3[10];  //
}A_21;             //
//----------------------------
typedef struct {     // Function format
 long  TagIndex;   // TAG index
 long  fnsize;     // Function size in bits
 long  fptr ;      // File pointer to line number
 long  index ;     // Index of next entry past this function
}A_22;             //
//----------------------------
typedef struct {     // Array format
 long  TagIndex;   //
 uint  LineNum;    //
 uint  SzArray;    //
 uint  Dim1;       //
 uint  Dim2;       //
 uint  Dim3;       //
 uint  Dim4;       //
 char  nused[2];   //
}A_23;             //
```

```
//---------------------------
typedef struct {     // End of blocks and functions
 char  nused1[4];  //
 uint  cline;      // C source line number
 char  nused2[12]; //
}A_24;             //
//---------------------------
typedef struct {     // Beginning of blocks and functions
 char  nused1[4];  //
 uint  cline;      // C source line number
 char  nused2[6];  //
 long  NxtIndex;   // Index to next entry past this block
 char  nused3[2];  //
}A_25;             //
//---------------------------
typedef struct {     // Beginning of blocks and functions
 long  TagIndex;   //
 char  nused1[2];  //
 uint  SzStrct;    // Size of structure or array
 char  nused2[10]; //
}A_26;             //
//=============================================================

enum SC_CLASS
{
 SC_NULL  =  0,
 SC_AUTO  =  1,
 SC_EXT   =  2,
 SC_STAT  =  3,
 SC_REG   =  4,
 SC_EXTDEF =  5,
 SC_LABEL  =  6,
```

```
SC_ULABEL = 7,
SC_MOS   = 8,
SC_ARG   = 9,
SC_STRTAG = 10,
SC_MOU   = 11,
SC_UNTAG = 12,
SC_TPDEF = 13,
SC_USTATIC= 14,
SC_ENTAG = 15,
SC_MOE   = 16,
SC_REGPARM= 17,
SC_FIELD = 18,
SC_UEXT  = 19,
SC_STATLAB= 10,
SC_EXTLAB = 21,
SC_BLOCK = 100,
SC_FCN   = 101,
SC_EOS   = 102,
SC_FILE  = 103,
SC_LINE  = 104
};
enum TYPES
{
 T_VOID=0,
 T_SCHAR ,
 T_CHAR  ,
 T_SHORT ,
 T_INT   ,
 T_LONG  ,
 T_FLOAT ,
 T_DOUBLE,
 T_STRUCT,
```

```
 T_UNION ,
 T_ENUM ,
 L_DOUBLE,
 T_UCHAR ,
 T_USHORT,
 T_UINT ,
 T_ULONG ,
 T_NULL=0x80
};
enum D_TYPES
{
 DT_NON=0,
 DT_PTR=1,
 DT_FCN=2,
 DT_ARY=3
};

typedef union
{
 struct
 {
  long DSP_addr; // DSP Physical address
  uint HLL_line; // Line number in HLL source
 }xref;
 struct
 {
  ulong sym_offs; // File pointer to a symbol entry
  uint  zero2  ; // 0
 }sptr;
}LINE_TBL;

#define HLL_limit 512    /* Limit for line entries */
```

```
extern LINE_TBL huge HLL[];
extern int lastHLL;

#endif
```

KEYDEF.H

```
#define _ESC  0x0100

#define _1    0x0200
#define _2    0x0300
#define _3    0x0400
#define _4    0x0500
#define _5    0x0600
#define _6    0x0700
#define _7    0x0800
#define _8    0x0900
#define _9    0x0A00
#define _0    0x0B00

#define _Hyp  0x0C00
#define _Eql  0x0D00
#define _Bs   0x0E00

#define _TAB  0x0f00
#define _Q    0x1000
#define _W    0x1100
#define _E    0x1200
#define _R    0x1300
#define _T    0x1400
#define _Y    0x1500
#define _U    0x1600
#define _I    0x1700
#define _O    0x1800
#define _P    0x1900

#define _Lbkt 0x1A00
#define _Rbkt 0x1B00
```

```
#define _Enter 0x1C00
//efine _???? 0x1D00

#define _A    0x1E00
#define _S    0x1F00
#define _D    0x2000
#define _F    0x2100
#define _G    0x2200
#define _H    0x2300
#define _J    0x2400
#define _K    0x2500
#define _L    0x2600

#define _Sqt  0x2700
#define _Qt   0x2800
//efine _?    0x2900
//efine _?    0x2A00
//efine _?    0x2B00

#define _Z    0x2C00
#define _X    0x2D00
#define _C    0x2E00
#define _V    0x2F00
#define _B    0x3000
#define _N    0x3100
#define _M    0x3200

#define _cma  0x3300
#define _prd  0x3400
#define _fsl  0x3500
//efine _?    0x3600
//efine _?    0x3700
```

```
//efine _?    0x3800
#define _Spc   0x3900
//efine _?    0x3A00
//efine _?    0x3B00

#define _F1    0x3B00
#define _F2    0x3C00
#define _F3    0x3D00
#define _F4    0x3E00
#define _F5    0x3F00
#define _F6    0x4000
#define _F7    0x4100
#define _F8    0x4200
#define _F9    0x4300
#define _F10   0x4400
//efine _F11   0x4500
//efine _F12   0x4600

#define _Hm    0x4700
#define _Up    0x4800
#define _Pup   0x4900
#define _mns   0x4A00
#define _Lt    0x4B00
#define _Cnt   0x4C00
#define _Rt    0x4D00
#define _pls   0x4E00
#define _End   0x4F00
#define _Dn    0x5000
#define _Pdn   0x5100
#define _Ins   0x5200
#define _Del   0x5300
```

```
#define _SF1  0x5400
#define _SF2  0x5500
#define _SF3  0x5600
#define _SF4  0x5700
#define _SF5  0x5800
#define _SF6  0x5900
#define _SF7  0x5A00
#define _SF8  0x5B00
#define _SF9  0x5C00
#define _SF10 0x5D00

#define _CF1  0x5D00
#define _CF2  0x5E00
#define _CF3  0x5F00
#define _CF4  0x6000
#define _CF5  0x6100
#define _CF6  0x6200
#define _CF7  0x6300
#define _CF8  0x6400
#define _CF9  0x6500
#define _CF10 0x6600

#define _AF1  0x6800
#define _AF2  0x6900
#define _AF3  0x6A00
#define _AF4  0x6B00
#define _AF5  0x6C00
#define _AF6  0x6D00
#define _AF7  0x6E00
#define _AF8  0x6F00
#define _AF9  0x7000
#define _AF10 0x7100
```

```
//#define _Alt  0x7500
// Key modifiers for bioskey(2)
#define _Rt_Sh  0x0001   /* Right Shift pressed */
#define _Lt_Sh  0x0002   /* Left Shift pressed  */
#define _Ctrl   0x0004   /* Ctrl pressed        */
#define _Alt    0x0008   /* Alt pressed         */
#define _Scrl   0x0010   /* Scroll Lock on      */
#define _Nmlck  0x0020   /* Num Lock on         */
#define _Cplck  0x0040   /* Caps on             */
#define _Insrt  0x0080   /* Insert on           */
#define _Lt_Ctrl 0x0100  /* Left Ctrl pressed   */
#define _Lt_Alt  0x0200  /* Left Alt pressed    */
#define _Rt_Ctrl 0x0400  /* Right Ctrl pressed  */
#define _Rt_Alt  0x0800  /* Right Alt pressed   */
#define _Scrllk  0x1000  /* Scroll Lock pressed */
#define _Numlck  0x2000  /* Num Lock pressed    */
#define _Caplck  0x4000  /* Caps Lock pressed   */
#define _Sysreq  0x8000  /* Sys Req pressed     */
#endif
```

FFT_256.CPP

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <graphics.h>
#include <bios.h>
#include "DSK.H"
#include "DSK_COFF.H"
#include "C3XMMRS.H"
#include "keydef.h"

#define MSG_BOX   0x809C00L
#define TRG_BOX   0x809C01L
#define A_BOX     0x809C02L
#define B_BOX     0x809C03L
#define C_BOX     0x809C04L
#define EDGESEL   0x809C05L
#define SAMPLES   0x809C06L
#define MASK      0x809C07L // Added logical AND of mantissa and mask
#define DATABLOCK 0x809900L
#define THx 40e-9          // DSP cycle time H1/H3 clock rate
#define Samples 256        // FFT size is 256
#define graph_vwport() setviewport( 50, 0, 563, 340,   1)
#define menu_vwport()  setviewport(334, 0, 639, 340,   1)
#define A_REG  ((TA <<9)+(RA <<2)+0) // A divisors.. set SCF rate
#define AP_REG ((TAP<<9)+(RAP<<2)+1) // TA RA prime registers (not used)
#define B_REG  ((TB <<9)+(RB <<2)+2) // B divisors
```

```
typedef enum messages
{
 STOP =1,
 START=2
}message;

char DSK_APP[]   ="FFT256.DSK";
char DSK_EXE_APP[]="FFT_256.EXE";
long  TLVL_V;
ulong T0_prdv=0x00000001L;
ulong ZERO   =0x00000000L;
ulong MASKVAL=0xFFFFFFFFL;

float Hz_per_div = 0.0;
float Fsr=1000.0, Fsx=1000.0;
int  TA =  10;    // DAC divisors
int  TB =  14;    //
int  RA =  10;    // ADC divisors
int  RB =  14;    //
int  TAP=  1;    // TA' and RA' are not used in this application
int  RAP=  1;    //
int C_REG=0x03;    // AIC control register bits

int  oldbuf[512];
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];
char buf_5[512];
char buf_6[512];
```

```
char buf_7[512];
char buf_8[512];

char  buf_pk_t[512];
int   buf_y_pk[512];
int   avg_on = 0;
void  init_graphics(void);
int   check_key (void);
void  binsprintf(char *s,int val);
//-------------------------------------------------
// draw_peak()
//-------------------------------------------------
#define pw 1   // controls peak indicator draw width
void draw_peak(void)
{
  int  x, y, *y_pk;
  char *pk_t, *ptr0, *tmp0;
  pk_t = buf_pk_t;
  tmp0 = buf_0;
  y_pk = buf_y_pk;
  ptr0 = tmp0;                    // Set buffer pointers
  setcolor(LIGHTRED);
  for(x=0;x<256;x+=2)
  {
   y = 128 - *ptr0++;             // present Y to display
   *pk_t += 1;                    // Inc times peak has been displayed
   if((*pk_t > 8) || (y < *y_pk))  // If vmag>last redraw new peak
   {
    line(x-pw,*y_pk,x+pw,*y_pk  ); // undraw old horizontal peak
   //line(x   ,*y_pk,x   ,*y_pk-4); // vertical version (not as interesting)
    if(y <= *y_pk)                // if vmag>last freshen peak hold
    {
```

```
      *y_pk = y;
      *pk_t = 0;
     }
     else
     {                    // else decay the peak
      if(y > (*y_pk+6)) *y_pk += 6;
      else          *y_pk += 1;
     }
     line(x-pw,*y_pk,x+pw, *y_pk  );  // draw new horizontal peak
    // line(x   ,*y_pk,x   , *y_pk-4);  // vertical
     }
     y_pk++;
     pk_t++;
    }
  }
//-------------------------------------------------------
// draw_vect() draws the vertical display bars for each
// frequency bin.  To make the display much faster only
// the part which changes is drawn using an XOR function.
//-------------------------------------------------------
void draw_vect()
{
 int  x, y, *old;
 char *ptr0, *tmp0;
 tmp0 = buf_0;
 ptr0 = tmp0;                    // Set buffer pointers
 setcolor(WHITE);
 old = oldbuf;
 setwritemode(1);                // Set line draw to XOR mode
 for(x=0;x<256;x+=2)
 {
  y = 128 - *ptr0++;             // present Y to display
```

```
// if(y == old_y) do nothing
    if(y  > *old) line(x,*old+1,x,y  );
    if(y  < *old) line(x,*old  ,x,y+1);
    *old++ = y;
  }
}
//-------------------------------------------------------------
// main()  After checking for any command line arguments main
// enters an outer 'forever loop' where the DSK communications
// channel is initialized, followed by loading the code and
// then initializing the graphics mode.  A second 'inner forever
// loop' is then entered where the application interacts with the
// host.  When a failure occurs inside the inner forever loop that
// loop is broken (see break; commands), causing the application
// to re-execute the outer loop, which causes the channel to be
// re-initialized followed by a code load and graphics initialize.
// This method allows the DSK to be disconnected and reconnected
// without the application bombing out to the command line.
//-------------------------------------------------------------
void main(void)
{
  int New_Params = 0, reset_flag = 0;
  ulong MSG=START, aic;
  MSGS err;
  char *ptr1, *ptr2, *ptr3, *ptr4;
  char *ptr5, *ptr6, *ptr7, *ptr8, *ptr0;
  char *tmp1, *tmp2, *tmp3, *tmp4;
  char *tmp5, *tmp6, *tmp7, *tmp8, *tmp0;
  int x;
  clrscr();
  Scan_Command_line(DSK_EXE_APP);
  clrscr();
```

```
//*********************************************//
// The outer loop initializes the DSK on entry  //
// or if the application fails                  //
//*********************************************//
for(;;)
{ for(;;)
 { if(Init_Communication(10000) == NO_ERR) break;
  if(kbhit()) exit(0);
 }
 HALT_CPU(); // Halt previously running apps code
 if((err=Load_File(DSK_APP,LOAD))!=NO_ERR)
 { printf("%s %s\n",DSK_APP,Error_Strg(err));
  exit(0);
 }
 RUN_CPU();
 init_graphics();
 //
 // Init buff ptrs, Temps are rotated
 //
 tmp0 = buf_0;
 tmp1 = buf_1; tmp2 = buf_2; tmp3 = buf_3; tmp4 = buf_4;
 tmp5 = buf_5; tmp6 = buf_6; tmp7 = buf_7; tmp8 = buf_8;
 ptr0 = tmp0;
 for(x=0;x<256;x++)              // Clear all buffersq
 {
  buf_1[x] = 0; buf_2[x] = 0; buf_3[x] = 0; buf_4[x] = 0;
  buf_5[x] = 0; buf_6[x] = 0; buf_7[x] = 0; buf_8[x] = 0;

  buf_0[x]   = 0;
  buf_y_pk[x] = 255;
  buf_pk_t[x] = 8;
  oldbuf[x]   = 255;
```

```
}
check_key();
draw_vect(); //
draw_peak(); // Draw peaks and vector displays using XOR write
//*************************************************************//
// The inner loop is repeated until a keyboard hit exits the //
// application or the application reports an error and needs //
// to be reinitialized                                       //
//*************************************************************//
//for(int loop=0;loop<1000;loop++) // 1K loop benchmark for entire app
for(;;)
{ // The inner loop draws the display using an XOR line draw
 // to overdraw existing vectors
 ptr0 = tmp0;                          // Set buffer pointers
 ptr1 = tmp1; ptr2 = tmp2; ptr3 = tmp3; ptr4 = tmp4;
 ptr5 = tmp5; ptr6 = tmp6; ptr7 = tmp7; ptr8 = tmp8;
 for(x=0;x<256;x+=2)
 { switch(avg_on)
  { case  1: *ptr0=(*ptr1+*ptr2)          >> 1; break; // avg 2
    case  2: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4) >> 2; break; // avg 4
    case  3: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4+          // avg 8
             *ptr5+*ptr6+*ptr7+*ptr8) >> 3; break;
    default: *ptr0= *ptr1;                 break; // avg 1
  }
  ptr0++;
  ptr1++; ptr2++; ptr3++; ptr4++;          // next data
  ptr5++; ptr6++; ptr7++; ptr8++;
 }
 draw_vect();
 draw_peak();
 // To prevent getmem timeouts an artificial strobe is used to
 // signal to the DSK that a host request is in progress
```

```
HPI_STRB(0);          // Drive HPSTB (INT2) low and wait
reset_flag = 0;
for(;;)               // for DSK to stop with full buffer
{ if(kbhit())
  {
   reset_flag = check_key();
   New_Params=1;
  }
  if(HPI_ACK())break;  // Note: break last to ensure keytrap!
  delay(1);
}
if(reset_flag) break;

ptr1 = tmp8;                  // Rotate the buffer pointers
tmp8 = tmp7; tmp7 = tmp6; tmp6 = tmp5; tmp5 = tmp4;
tmp4 = tmp3; tmp3 = tmp2; tmp2 = tmp1; tmp1 = ptr1;
//
// If a key was pressed, update the AIC setup
if(New_Params)
{
  if(putmem(MASK,          1,&MASKVAL     )!=NO_ERR) break;
  if(putmem(T0_prd ,       1,&T0_prdv)!=NO_ERR) break;
  if(putmem(T0_count,       1,     &ZERO)!=NO_ERR) break;
  if(putmem(T1_prd ,       1,&T0_prdv)!=NO_ERR) break;
  if(putmem(T1_count,       1,     &ZERO)!=NO_ERR) break;

// if(putmem(EDGESEL,1,&        Edge)!=NO_ERR) break;
  if(TB>TA)
  {
   aic = A_REG; if(putmem(A_BOX,1,&aic)!=NO_ERR) break;
   aic = B_REG; if(putmem(B_BOX,1,&aic)!=NO_ERR) break;
  }
```

```
      else
      {
       aic = B_REG; if(putmem(A_BOX,1,&aic)!=NO_ERR) break;
       aic = A_REG; if(putmem(B_BOX,1,&aic)!=NO_ERR) break;
      }
     aic = C_REG; if(putmem(C_BOX,1,&aic)!=NO_ERR) break;
     //closegraph();
     //exit(0);
    }
    New_Params = 0;
    if(getmem(DATABLOCK,Samples/8,(ulong *)ptr1)!=NO_ERR) break;//128 char
    putmem(MSG_BOX,1,&MSG);
   }
   closegraph();  // Shutdown graphics before re-initializing
   printf("%s: %s\n",DSK_APP, Error_Strg(err));
   printf("Communications are being reinitialized");
   delay(1000);
   DSK_reset();
  }
}
//---------------------------------------------------------------
// clip() is used to clip the upper and lower bounds of a number
//---------------------------------------------------------------
long inline clip(long x, int min, int max)
{
 if(x<min) return min;
 if(x>max) return max;
 return x;
}
//---------------------------------------------------------------
// check_key() is used to associate keystrokes with actions
//---------------------------------------------------------------
```

```
int check_key(void)
{
 int  key=0;
 int  Yt=0;
 char buf[80];
 static int old_key=0; // accelerate action if same key is hit again
 static int accel  =1;
 if(kbhit()) key = bioskey(0) & 0xFF00;
 if(old_key==key) accel = accel + 1;
 else             accel = accel / 4;
 accel = clip(accel,1,200); old_key = key;
 if(key)
 {
  switch(key)
  {
   case _Ins: T0_prdv -=accel; break;
   case _Del: T0_prdv +=accel; break;
   case _R  : return 1;  // Return reset (break) flag
   //
   // _M adjusts a mantissa mask which can be used to
   // analyze the effects of numeric precision
   //
   case _M  : MASKVAL = MASKVAL << 1;
              if(MASKVAL==0) MASKVAL = 0xFFFFFFFFL;
              break;

   case _Q  :closegraph();
             _setcursortype(_NORMALCURSOR);
             exit(0);
             break;
   case _F1 : TA++; break;
   case _F2 : TA--; break;
```

```
 case _F3 : TB++; break;
 case _F4 : TB--; break;
 case _F5 : RA++; break;
 case _F6 : RA--; break;
 case _F7 : RB++; break;
 case _F8 : RB--; break;
 case _1  : C_REG = C_REG ^ 0x80; break; // bit 7
 case _2  : C_REG = C_REG ^ 0x40; break; // bit 6
 case _3  : C_REG = C_REG ^ 0x20; break; // bit 5
 case _4  : C_REG = C_REG ^ 0x10; break; // bit 4
 case _5  : C_REG = C_REG ^ 0x08; break; // bit 3
 case _6  : C_REG = C_REG ^ 0x04; break; // bit 2
 case _A  :
        switch(avg_on)
        {
         case  1: avg_on=2; break;
         case  2: avg_on=3; break;
         case  3: avg_on=4; break;
         default: avg_on=1; break;
        }
        break;

  default : return 0;
 }
 TA      = clip(TA      ,  3,      31);
 TB      = clip(TB      , 12,      63);
 RA      = clip(RA      ,  3,      31);
 RB      = clip(RB      , 12,      63);
 T0_prdv = clip(T0_prdv,   1,      64);
}
setwritemode(0);                // turn off XOR write
menu_vwport();                  // Write to window in bright blue
```

```
clearviewport();
setcolor(11);
Yt=1;

#define C1 1
outtextxy(C1,Yt   ,  "Q   quit" );
outtextxy(C1,Yt+=10,  "R   reset" ); switch(avg_on)
{ case  1: sprintf(buf,"A   avg   => 2 frames");break;
 case  2: sprintf(buf,"A   avg   => 4 frames");break;
 case  3: sprintf(buf,"A   avg   => 8 frames");break;
 default: sprintf(buf,"A   avg   => 1 frame");break;
}
outtextxy(C1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"Ins  TIM0+"); outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"Del  TIM0-  => Tprd=%02ld",T0_prdv); outtextxy(C1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"F1   TA++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F2   TA--  => TA=%02d",TA);   outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F3   TB++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F4   TB--  => TB=%02d",TB);   outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F5   RA++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F6   RA--  => RA=%02d",RA);   outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F7   RB++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F8   RB--  => RB=%02d",RB);   outtextxy(C1,Yt+=10,buf);
//sprintf(buf,"          => Fdac=%7.2f",Fsx);outtextxy(C1,Yt+=10,buf);
//sprintf(buf,"          => Fadc=%7.2f",Fsr);outtextxy(C1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"AIC CTRL:  ");binsprintf(buf+10,C_REG);outtextxy(C1,Yt+=10,buf);
```

```
sprintf(buf,"        ||||||"                ); outtextxy(C1,Yt+=10,buf);
sprintf(buf," use num |||||+---6  BP filter"  ); outtextxy(C1,Yt+=10,buf);
sprintf(buf," keys to |||+-----5  Loopback"   ); outtextxy(C1,Yt+=10,buf);
sprintf(buf," toggle  ||+------4  AIN/AUXIN"  ); outtextxy(C1,Yt+=10,buf);
sprintf(buf," bits    |+-------3  Asynch/synch"); outtextxy(C1,Yt+=10,buf);
sprintf(buf,"         +--------1,2 gain bits"  ); outtextxy(C1,Yt+=10,buf);
 Yt+=10;
 switch(C_REG&0xC0)
 { case 0x00:  outtextxy(C1,Yt+=10,"+0dB     6.0 Vmax"); break;
  case 0x40:  outtextxy(C1,Yt+=10,"+6dB     3.0 Vmax"); break;
  case 0x80:  outtextxy(C1,Yt+=10,"+12dB    1.5 Vmax"); break;
  case 0xC0:  outtextxy(C1,Yt+=10,"+0dB     6.0 Vmax"); break;
 }
 if(C_REG&0x20)outtextxy(C1,Yt+=10,"Synch    Fadc==Fdac");
 else        outtextxy(C1,Yt+=10,"Asynch   Fadc!=Fdac");
 if(C_REG&0x10)outtextxy(C1,Yt+=10,"AUXIN    ENABLE");
 else        outtextxy(C1,Yt+=10,"AIN      ENABLE");
 if(C_REG&0x08)outtextxy(C1,Yt+=10,"loopback  ENABLE");
 else        outtextxy(C1,Yt+=10,"loopback  DISABLE");
 if(C_REG&0x04)outtextxy(C1,Yt+=10,"BP filter ENABLE");
 else        outtextxy(C1,Yt+=10,"BP filter DISABLE");


 Yt+=10;
 sprintf(buf,"[M]ant Mask=%08lx (precision)",MASKVAL);
 outtextxy(C1,Yt+=10,buf);
 //
 //
 graph_vwport();
 Fsx    = 1/(2*TA*TB*(2*THx * T0_prdv));
 if(C_REG & 0x20) Fsr = Fsx;
 else        Fsr = 1/(2*RA*RB*(2*THx * T0_prdv));
```

```
Hz_per_div = (520.0/512.0)*Fsr/(2.0*10);
//
//
setfillstyle(SOLID_FILL, BLACK);
bar(0,269,260,310);

sprintf(buf,  "Hz/div=%7.2f",Hz_per_div); outtextxy(10,270,buf);
sprintf(buf,  "  Fdac=%7.2f",Fsx); outtextxy(10,285,buf);
if(C_REG&0x20)
 sprintf(buf,"  Fadc=Fdac");
else
 sprintf(buf,"  Fadc=%7.2f",Fsr);
outtextxy(10,295,buf);

sprintf(buf,"Note: A sawtooth signal generated by the DAC can be looped");
outtextxy(10,315,buf);
sprintf(buf,"      back for self analysis using the number 5 key");
outtextxy(10,325,buf);
return 0;
}
//---------------------------------------------------------------
// init_graphics() initializes the host display for graphics
// output and then draws the lines and text for the display graph
//---------------------------------------------------------------
void init_graphics(void)
{
 int gdriver = EGA, gmode = EGAHI, errorcode, Y, X;
//
// register EGAVGA_driver which is the name of the driver in EGAVGA.OBJ
// EGAVGA.OBJ is created from EGAVGA.BGI using the BGIOBJ.EXE utility
// and is linked by either the link list or project file
//
```

```
// NOTE: This section of code can be ommitted if EGAVGA.BGI is
//       located in the applications startup directory
//
errorcode = registerbgidriver(EGAVGA_driver);
if (errorcode < 0)  // report any registration errors
{
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1); // terminate with an error code
}

  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();                       //
  graph_vwport();
  setcolor(GREEN);
  for(Y=0;Y<=260;Y+=26) line(0,Y,260,Y);  // draw reticle
  for(X=0;X<=260;X+=26) line(X,0,X,260);  //

 // setcolor(DARKGRAY);
  Y = 2;
  X = 2;
  outtextxy(X,Y    ,"+20");
  outtextxy(X,Y+=26,"+10");
  outtextxy(X,Y+=26," +0");
```

```
 outtextxy(X,Y+=26,"-10");
 outtextxy(X,Y+=26,"-20");
 outtextxy(X,Y+=26,"-30");
 outtextxy(X,Y+=26,"-40");
 outtextxy(X,Y+=26,"-50");
 outtextxy(X,Y+=26,"-60");
 outtextxy(X,Y+=26,"-70");
//outtextxy(X,Y+=26,"-80");

 setwritemode(1);                    // display lines are XOR drawn
 setcolor(15);                       // light gray
}
//-------------------------------------------------------------
// bunsprintf() prints an integer values 1,0 bit pattern to a
// string pointer which can then be printed to the display device
//-------------------------------------------------------------
void binsprintf(char *s,int val)
{
 char *p;
 unsigned int t;
 p = s;
 t = 0x80;  // scan 8 bits
 for(;t>0;t=t>>1)
 {
  if(val & t) *p++ = '1';
  else        *p++ = '0';
 }
 *p=0;
}
```

FFT_512.CPP

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <graphics.h>
#include <bios.h>
#include "DSK.H"
#include "DSK_COFF.H"
#include "C3XMMRS.H"
#include "keydef.h"

#define MSG_BOX    0x809E00L
#define TRG_BOX    0x809E01L
#define A_BOX      0x809E02L
#define B_BOX      0x809E03L
#define C_BOX      0x809E04L
#define EDGESEL    0x809E05L
#define SAMPLES    0x809E06L
#define LOAD_BOX   0x809E07L
#define Samples 512 // FFT size
#define THx 40e-9   // DSP cycle time H1/H3 clock rate
#define DATABLOCK (0x809800L + Samples)
#define graph_vwport() setviewport( 50, 0, 563, 340, 1)
#define menu_vwport()  setviewport(334, 0, 639, 340, 1)
#define A_REG  ((TA <<9)+(RA <<2)+0) // A divisors.. set SCF rate
#define AP_REG ((TAP<<9)+(RAP<<2)+1) // TA RA prime registers (not used)
#define B_REG  ((TB <<9)+(RB <<2)+2) // B divisors
```

```
char DSK_APP[]    ="FFT512.DSK";
char DSK_EXE_APP[]="FFT_512.EXE";

typedef enum messages
{
 STOP =1,
 START=2
}message;

long  TLVL_V;
ulong T0_prdv = 0x00000001L;
ulong ZERO    = 0x00000000L;

float Hz_per_div = 0.0;
float Fsr=1000.0, Fsx=1000.0;
int   TA =  10;    // DAC divisors
int   TB =  14;    //
int   RA =  10;    // ADC divisors
int   RB =  14;    //
int   TAP=  1;     // TA' and RA' are not used in this application
int   RAP=  1;     //
int C_REG=0x03;    // AIC control register bits

int  oldbuf[Samples];
char buf_0[Samples];              // Keep past data history for
char buf_1[Samples];              // time averaging of signals
char buf_2[Samples];
char buf_3[Samples];
char buf_4[Samples];
char buf_5[Samples];
char buf_6[Samples];
```

```
char buf_7[Samples];
char buf_8[Samples];
char buf_pk_t[Samples];
int  buf_y_pk[Samples];

void binsprintf(char *s,int val);
void init_graphics(void);
int  check_key(void);
void out_TEXT(void);
int  avg_on = 0;
//---------------------------------------------------------
// draw the red cross bars at the top of each data column
//---------------------------------------------------------
#define pw 1 // define peak indicator draw width
void draw_peak(void)
{
 int  x, y, *y_pk;
 char *pk_t, *ptr0, *tmp0;
 pk_t = buf_pk_t;
 tmp0 = buf_0;
 y_pk = buf_y_pk;
 ptr0 = tmp0;                    // Set buffer pointers
 setcolor(LIGHTRED);
 for(x=0;x<Samples/2;x++)
 {
  y = 128 - *ptr0++;              // present Y to display
  *pk_t += 1;                     // Inc times peak has been displayed
  if((*pk_t > 8) || (y < *y_pk))// If vmag>last redraw new peak
  {
   line(x-pw,*y_pk,x+pw,*y_pk  ); // undraw old hsorizontal peak
  //line(x   ,*y_pk,x   ,*y_pk-4); // vertical
   if(y <= *y_pk)                 // if vmag>last freshen peak hold
```

```
    {
     *y_pk = y;
     *pk_t = 0;
    }
    else
    {                       // else decay the peak
     if(y > (*y_pk+6))
      *y_pk += 6;
     else
      *y_pk += 1;
    }
    line(x-pw,*y_pk,x+pw, *y_pk  );  // draw new horizontal peak
   // line(x   ,*y_pk,x   , *y_pk-4);  // vertical
    }
    y_pk++;
    pk_t++;
  }
}
//-------------------------------------------------------
// draw the data data columns using an XOR line draw of
// only the pixels that change for maximum speed.
//-------------------------------------------------------
void draw_vect()
{
  int  x, y, *old;
  char *ptr0, *tmp0;
  tmp0 = buf_0;
  ptr0 = tmp0;                    // Set buffer pointers
  setcolor(WHITE);
  old = oldbuf;
  setwritemode(1);
  for(x=0;x<Samples/2;x++)
```

```
  {
   y = 128 - *ptr0++;              // present Y to display
//  if(y == old_y) do nothing
   if(y  > *old) line(x,*old+1,x,y  );
   if(y  < *old) line(x,*old  ,x,y+1);
   *old++ = y;
  }
}
//-------------------------------------------------------
// main(): Note the use of an outer/inner forever loop.
// If a communications error occurs in the inner loop the
// outer loop catches the error and re-initializes the DSK
//-------------------------------------------------------
void main(void)
{
 int New_Params = 0, reset_flag = 0;
 ulong MSG=START, aic;
 MSGS err;
 char *ptr1, *ptr2, *ptr3, *ptr4;
 char *ptr5, *ptr6, *ptr7, *ptr8, *ptr0;
 char *tmp1, *tmp2, *tmp3, *tmp4;
 char *tmp5, *tmp6, *tmp7, *tmp8, *tmp0;
 int x;
 clrscr();
 Scan_Command_line(DSK_EXE_APP);
 clrscr();
 //*********************************************//
 // The outer loop initializes the DSK on entry  //
 // or if the application fails                  //
 //*********************************************//
 for(;;)
 { for(;;)
```

```
{ if(Init_Communication(10000) == NO_ERR) break;
 if(kbhit()) exit(0);
}
HALT_CPU(); // Halt previously running apps code
if((err=Load_File(DSK_APP,LOAD))!=NO_ERR)
{ printf("%s %s\n",DSK_APP,Error_Strg(err));
 exit(0);
}
RUN_CPU();
init_graphics();
//
// Init buff ptrs, Temps are rotated
//
tmp0 = buf_0;
tmp1 = buf_1; tmp2 = buf_2; tmp3 = buf_3; tmp4 = buf_4;
tmp5 = buf_5; tmp6 = buf_6; tmp7 = buf_7; tmp8 = buf_8;
ptr0 = tmp0;
for(x=0;x<Samples;x++)                  // Clear all buffersq
{
 buf_1[x] = 0; buf_2[x] = 0; buf_3[x] = 0; buf_4[x] = 0;
 buf_5[x] = 0; buf_6[x] = 0; buf_7[x] = 0; buf_8[x] = 0;

 buf_0[x]   = 0;
 buf_y_pk[x] = 255;
 buf_pk_t[x] = 8;
 oldbuf[x]  = 255;
}
check_key();
draw_vect(); //
draw_peak(); // Draw peaks and vector displays using XOR write
//*********************************************************//
// The inner loop is repeated until a keyboard hit exits the //
```

```
// application or the application reports an error and needs //
// to be reinitialized                                       //
//***********************************************************//
for(;;)
{ // The inner loop draws the display using an XOR line draw
 // to overdraw existing vectors
 ptr0 = tmp0;                          // Set buffer pointers
 ptr1 = tmp1; ptr2 = tmp2; ptr3 = tmp3; ptr4 = tmp4;
 ptr5 = tmp5; ptr6 = tmp6; ptr7 = tmp7; ptr8 = tmp8;
 for(x=0;x<Samples/2;x+=1)
 { switch(avg_on)
  { case  1: *ptr0=(*ptr1+*ptr2)          >> 1; break; // avg 2
    case  2: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4) >> 2; break; // avg 4
    case  3: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4+          // avg 8
                *ptr5+*ptr6+*ptr7+*ptr8) >> 3; break;
    default: *ptr0= *ptr1;                 break; // avg 1
  }
  ptr0++;
  ptr1++; ptr2++; ptr3++; ptr4++;          // next data
  ptr5++; ptr6++; ptr7++; ptr8++;
 }
 draw_vect();
 draw_peak();
 // To prevent getmem timeouts an artificial strobe is used to
 // signal to the DSK that a host request is in progress
 HPI_STRB(0);          // Drive HPSTB (INT2) low and wait
 reset_flag = 0;
 for(;;)               // for DSK to stop with full buffer
 { if(kbhit())
   {
    reset_flag = check_key();
    New_Params=1;
```

```
 }
 if(HPI_ACK())break;  // Note: break last to ensure keytrap!
 delay(1);
}
if(reset_flag) break;

ptr1 = tmp8;                  // Rotate the buffer pointers
tmp8 = tmp7; tmp7 = tmp6; tmp6 = tmp5; tmp5 = tmp4;
tmp4 = tmp3; tmp3 = tmp2; tmp2 = tmp1; tmp1 = ptr1;
//
// If a key was pressed, update the AIC setup
if(New_Params)
{
 if(putmem(T0_prd ,      1,&T0_prdv)!=NO_ERR) break;
 if(putmem(T0_count,     1,     &ZERO)!=NO_ERR) break;
 if(putmem(T1_prd ,      1,&T0_prdv)!=NO_ERR) break;
 if(putmem(T1_count,     1,     &ZERO)!=NO_ERR) break;
 if(TB>TA)
 {
  aic = A_REG; if(putmem(A_BOX,1,&aic)!=NO_ERR) break;
  aic = B_REG; if(putmem(B_BOX,1,&aic)!=NO_ERR) break;
 }
 else
 {
  aic = B_REG; if(putmem(A_BOX,1,&aic)!=NO_ERR) break;
  aic = A_REG; if(putmem(B_BOX,1,&aic)!=NO_ERR) break;
 }
 aic = C_REG; if(putmem(C_BOX,1,&aic)!=NO_ERR) break;
 if(putmem(LOAD_BOX,1,&aic)!=NO_ERR) break; // Any nonzero reinits AIC
}
New_Params = 0;
if(getmem(DATABLOCK,Samples/8,(ulong *)ptr1)!=NO_ERR) break;//128 char
```

```
    putmem(MSG_BOX,1,&MSG);
   }
   closegraph();  // Shutdown graphics before re-initializing
   printf("%s: %s\n",DSK_APP, Error_Strg(err));
   printf("Communications are being reinitialized");
   delay(1000);
   DSK_reset();
  }
 }
//-------------------------------------------------------
// clip() clips a value to min and max
//-------------------------------------------------------
long inline clip(long x, int min, int max)
{
  if(x<min) return min;
  if(x>max) return max;
  return x;
}
//-------------------------------------------------------
// check_key() is the main keytrap routine.
//-------------------------------------------------------
int check_key(void)
{
  int key=0;
  char buf[80];
  int Yt=0;
  static int old_key=0;  // If the same key is hit repeatedly, speed up
  static int accel=1;

  if(kbhit()) key = bioskey(0) & 0xFF00;
  if(old_key==key) accel = accel + 1;
  else             accel = accel / 4;
```

```
accel = clip(accel,1,200); old_key = key;
if(key)
{
 switch(key)
 {
  case _Ins: T0_prdv -=accel; break;
  case _Del: T0_prdv +=accel; break;
  case _R  : return 1;  // Return reset (break) flag
  case _Q  :closegraph();
           _setcursortype(_NORMALCURSOR);
           exit(0);
           break;
  case _F1 : TA++; break;
  case _F2 : TA--; break;
  case _F3 : TB++; break;
  case _F4 : TB--; break;
  case _F5 : RA++; break;
  case _F6 : RA--; break;
  case _F7 : RB++; break;
  case _F8 : RB--; break;
  case _1  : C_REG = C_REG ^ 0x80; break; // bit 7
  case _2  : C_REG = C_REG ^ 0x40; break; // bit 6
  case _3  : C_REG = C_REG ^ 0x20; break; // bit 5
  case _4  : C_REG = C_REG ^ 0x10; break; // bit 4
  case _5  : C_REG = C_REG ^ 0x08; break; // bit 3
  case _6  : C_REG = C_REG ^ 0x04; break; // bit 2
  case _A  :
           switch(avg_on)
           {
            case  1: avg_on=2; break;
            case  2: avg_on=3; break;
            case  3: avg_on=4; break;
```

```
            default: avg_on=1; break;
          }
          break;

  default : return 0;
 }
 TA      = clip(TA      ,  3,      31);
 TB      = clip(TB      , 12,      63);
 RA      = clip(RA      ,  3,      31);
 RB      = clip(RB      , 12,      63);
 T0_prdv = clip(T0_prdv,   1,      64);
}
setwritemode(0);                  // turn off XOR write
menu_vwport();                    // Write to window in bright blue
clearviewport();
setcolor(11);
Yt=1;

#define C1 1
outtextxy(C1,Yt   ,  "Q   quit" );
outtextxy(C1,Yt+=10,  "R   reset" ); switch(avg_on)
{ case  1: sprintf(buf,"A   avg   => 2 frames");break;
  case  2: sprintf(buf,"A   avg   => 4 frames");break;
  case  3: sprintf(buf,"A   avg   => 8 frames");break;
  default: sprintf(buf,"A   avg   => 1 frame");break;
}
outtextxy(C1,Yt+=10,buf);

Yt+=10;
sprintf(buf,"Ins  TIM0+"); outtextxy(C1,Yt+=10,buf);
sprintf(buf,"Del  TIM0-  => Tprd=%02ld",T0_prdv); outtextxy(C1,Yt+=10,buf);
```

```
 Yt+=10;
 sprintf(buf,"F1  TA++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F2  TA--  => TA=%02d",TA);  outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F3  TB++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F4  TB--  => TB=%02d",TB);  outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F5  RA++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F6  RA--  => RA=%02d",RA);  outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F7  RB++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F8  RB--  => RB=%02d",RB);  outtextxy(C1,Yt+=10,buf);
//sprintf(buf,"         => Fdac=%7.2f",Fsx);outtextxy(C1,Yt+=10,buf);
//sprintf(buf,"         => Fadc=%7.2f",Fsr);outtextxy(C1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"AIC CTRL: ");binsprintf(buf+10,C_REG);outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"          |||||"                 ); outtextxy(C1,Yt+=10,buf);
 sprintf(buf," use num |||||+---6  BP filter"   ); outtextxy(C1,Yt+=10,buf);
 sprintf(buf," keys to |||+-----5  Loopback"    ); outtextxy(C1,Yt+=10,buf);
 sprintf(buf," toggle  ||+------4  AIN/AUXIN"   ); outtextxy(C1,Yt+=10,buf);
 sprintf(buf," bits    |+-------3  Asynch/synch"); outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"         +--------1,2 gain bits"  ); outtextxy(C1,Yt+=10,buf);
 Yt+=10;
 switch(C_REG&0xC0)
 { case 0x00: outtextxy(C1,Yt+=10,"+0dB    6.0 Vmax"); break;
   case 0x40: outtextxy(C1,Yt+=10,"+6dB    3.0 Vmax"); break;
   case 0x80: outtextxy(C1,Yt+=10,"+12dB   1.5 Vmax"); break;
   case 0xC0: outtextxy(C1,Yt+=10,"+0dB    6.0 Vmax"); break;
 }
 if(C_REG&0x20)outtextxy(C1,Yt+=10,"Synch   Fadc==Fdac");
 else          outtextxy(C1,Yt+=10,"Asynch  Fadc!=Fdac");
 if(C_REG&0x10)outtextxy(C1,Yt+=10,"AUXIN   ENABLE");
 else          outtextxy(C1,Yt+=10,"AIN     ENABLE");
 if(C_REG&0x08)outtextxy(C1,Yt+=10,"loopback ENABLE");
```

```
else        outtextxy(C1,Yt+=10,"loopback  DISABLE");
if(C_REG&0x04)outtextxy(C1,Yt+=10,"BP filter ENABLE");
else        outtextxy(C1,Yt+=10,"BP filter DISABLE");

graph_vwport();
Fsx    = 1/(2*TA*TB*(2*THx * T0_prdv));
if(C_REG & 0x20) Fsr = Fsx;
else          Fsr = 1/(2*RA*RB*(2*THx * T0_prdv));
Hz_per_div = (520.0/Samples)*Fsr/(2.0*10);
//
//
setfillstyle(SOLID_FILL, BLACK);
bar(0,269,260,310);

sprintf(buf,  "Hz/div=%7.2f",Hz_per_div); outtextxy(10,270,buf);
sprintf(buf,  "  Fdac=%7.2f",Fsx); outtextxy(10,285,buf);
if(C_REG&0x20)
 sprintf(buf,"  Fadc=Fdac");
else
 sprintf(buf,"  Fadc=%7.2f",Fsr);
outtextxy(10,295,buf);

sprintf(buf,"Note: A sawtooth signal generated by the DAC can be looped");
outtextxy(10,315,buf);
sprintf(buf,"      back for self analysis using the number 5 key");
outtextxy(10,325,buf);
return 0;
}
//---------------------------------------------------------------
// init_graphics() is responsible for initializing the graphics
// and then filling in the display lines and text
//---------------------------------------------------------------
```

```
void init_graphics(void)
{
 int gdriver = EGA, gmode = EGAHI, errorcode, Y, X;
//
// register EGAVGA_driver which is the name of the driver in EGAVGA.OBJ
// EGAVGA.OBJ is created from EGAVGA.BGI using the BGIOBJ.EXE utility
// and is linked by either the link list or project file
//
// NOTE: This section of code can be ommitted if EGAVGA.BGI is
//       located in the applications startup directory
//
errorcode = registerbgidriver(EGAVGA_driver);
if (errorcode < 0)  // report any registration errors
{
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1); // terminate with an error code
}

 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();                    //
 graph_vwport();
 setcolor(GREEN);
 for(Y=0;Y<=260;Y+=26) line(0,Y,260,Y);  // draw reticle
```

```
 for(X=0;X<=260;X+=26) line(X,0,X,260);  //


// setcolor(DARKGRAY);
 Y = 2;
 X = 2;
 outtextxy(X,Y    ,"+20");
 outtextxy(X,Y+=26,"+10");
 outtextxy(X,Y+=26," +0");
 outtextxy(X,Y+=26,"-10");
 outtextxy(X,Y+=26,"-20");
 outtextxy(X,Y+=26,"-30");
 outtextxy(X,Y+=26,"-40");
 outtextxy(X,Y+=26,"-50");
 outtextxy(X,Y+=26,"-60");
 outtextxy(X,Y+=26,"-70");
//outtextxy(X,Y+=26,"-80");


 setwritemode(1);              // display lines are XOR drawn
 setcolor(15);                 // light gray
}
//-------------------------------------------------
// binsprintf() creates a 1/0 binary print string
//-------------------------------------------------
void binsprintf(char *s,int val)
{
 char *p;
 unsigned int t;
 p = s;
 t = 0x80;  // scan 8 bits
 for(;t>0;t=t>>1)
 {
  if(val & t) *p++ = '1';
```

```
    else        *p++ = '0';
  }
  *p=0;
}
```

FFT_1024.CPP

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <graphics.h>
#include <bios.h>
#include "DSK.H"
#include "DSK_COFF.H"
#include "C3XMMRS.H"
#include "keydef.h"
void binsprintf(char *s,int val);
char DSK_APP[]    ="FFT1024.DSK";
char DSK_EXE_APP[]="FFT_1024.EXE";

#define MSG_BOX   0x809E00L
#define LOAD_BOX  0x809E01L
#define A_BOX     0x809E02L
#define B_BOX     0x809E03L
#define C_BOX     0x809E04L
#define SAMPLES   0x809E05L

#define Samples 1024 //512 //1024  // FFT size
#define DATABLOCK (0x809800L /*Samples*/)
#define THx 40e-9 // DSP cycle time H1/H3 clock rate

#define graph_vwport() setviewport( 0,  0, 639, 340,   1)
#define menu_vwport()  setviewport( 0,262, 639, 340,   1)
```

```
#define A_REG  ((TA <<9)+(RA <<2)+0) // A divisors.. set SCF rate
#define AP_REG ((TAP<<9)+(RAP<<2)+1) // TA RA prime registers (not used)
#define B_REG  ((TB <<9)+(RB <<2)+2) // B divisors

void setup_menu2(void);

typedef enum messages
{
 STOP =1,
 START=2
}message;
long TLVL_V;
ulong ClockMd = 0x2C1;
ulong T0_prdv =  1;
ulong ZERO=0;
//ulong CHOP=0xFFFFFFFFL;

//float AIC_gain = 0.25;
float Hz_per_div = 0.0;
float Fsr=1000.0, Fsx=1000.0;
int  TA =  10;    // DAC divisors
int  TB =  14;    //
int  RA =  10;    // ADC divisors
int  RB =  14;    //
int  TAP=  1;    // TA' and RA' are not used in this application
int  RAP=  1;    //
//int C_REG=0x83;    // control
int C_REG=0x03;    // control

void vline(int,int,int,int);

int PR0A,offst,PR0A_old,off2,PVGA,maxcol,maxrow;
```

```
int  oldbuf[Samples];
char buf_0[Samples];                    // Keep past data history for
char buf_1[Samples];                    // time averaging of signals
char buf_2[Samples];
char buf_3[Samples];
char buf_4[Samples];
char buf_5[Samples];
char buf_6[Samples];
char buf_7[Samples];
char buf_8[Samples];

char buf_pk_t[Samples];
int  buf_y_pk[Samples];

void init_graphics(void);
int  check_key(void);
void  out_TEXT(void);
//void setup_vals(void);
int avg_on = 0;

#define pw 1 // controls peak draw width
void draw_peak(void)
{
  int  x, y, *y_pk;
  char *pk_t, *ptr0, *tmp0;
  pk_t = buf_pk_t;
  tmp0 = buf_0;
  y_pk = buf_y_pk;
  ptr0 = tmp0;                          // Set buffer pointers
  setcolor(LIGHTRED);
  for(x=0;x<Samples/2;x++)
```

```
{
 y = 128 - *ptr0++;              // present Y to display
 *pk_t += 1;                     // Inc times peak has been displayed
 if((*pk_t > 8) || (y < *y_pk))// If vmag>last redraw new peak
 {
  line(x-pw,*y_pk,x+pw,*y_pk  ); // undraw old hsorizontal peak
 //line(x   ,*y_pk,x   ,*y_pk-4); // vertical
  if(y <= *y_pk)                 // if vmag>last freshen peak hold
  {
   *y_pk = y;
   *pk_t = 0;
  }
  else
  {                        // else decay the peak
   if(y > (*y_pk+6))
    *y_pk += 6;
   else
    *y_pk += 1;
  }
  line(x-pw,*y_pk,x+pw, *y_pk  );  // draw new horizontal peak
 // line(x   ,*y_pk,x   , *y_pk-4);  // vertical
  }
  y_pk++;
  pk_t++;
 }
}
void draw_vect(void)
{
 int  x, y, *old;
 char *ptr0, *tmp0;
 tmp0 = buf_0;
 ptr0 = tmp0;                    // Set buffer pointers
```

```
 setcolor(WHITE);
 old = oldbuf;
 setwritemode(1);
 for(x=0;x<Samples/2;x++)
 {
  y = 128 - *ptr0++;              // present Y to display
//  if(y == old_y) do nothing
  if(y  > *old) line(x,*old+1,x,y  );
  if(y  < *old) line(x,*old  ,x,y+1);
  *old++ = y;
 }
}

void main(void)
{
 FILE *dskfile;
 struct ftime ft1, ft2;
 int New_Params = 0, reset_flag = 0;
 ulong MSG=START, aic;
 long L0, L1;
 MSGS err;
 char *ptr1, *ptr2, *ptr3, *ptr4;
 char *ptr5, *ptr6, *ptr7, *ptr8, *ptr0;
 char *tmp1, *tmp2, *tmp3, *tmp4;
 char *tmp5, *tmp6, *tmp7, *tmp8, *tmp0;
 int x;
 clrscr();
 Scan_Command_line(DSK_EXE_APP);
 clrscr();
 //*********************************************//
 // The outer loop initializes the DSK on entry  //
 // or if the application fails                  //
```

```
//*********************************************//
for(;;)
{ for(;;)
  { if(Init_Communication(10000) == NO_ERR) break;
    if(kbhit()) exit(0);
  }
  HALT_CPU(); // Halt previously running apps code

/*  if((err=Load_File(DSK_APP,LOAD))!=NO_ERR)
  { printf("%s %s\n",DSK_APP,Error_Strg(err));
    exit(0);
  } */
  //
  // If DSK_APP.HEX is older than DSK_APP.DSK perform an update
  //
  dskfile = fopen(DSK_APP,"rb");
  getftime(fileno(dskfile), &ft1);
  fclose(dskfile);
  //
  // Now look for HEX file
  //
  ptr0 = strstr(DSK_APP,".");
  strcpy(ptr0,".HEX");
  if ((dskfile = fopen(DSK_APP,"rb")) != NULL)
  {
    getftime(fileno(dskfile), &ft2);
    //
    // The file time structure is 32 bits, so a typecasted
    // 32 bit compare avoids normalization of each field
    //
    L0 = *(long *)&ft1;
    L1 = *(long *)&ft2;
```

```
 if(L0>=L1) unlink(DSK_APP);
 fclose(dskfile);
}

if ((dskfile = fopen(DSK_APP,"rb")) == NULL)
{
 strcpy(ptr0,".DSK");
 if((err=Load_File(DSK_APP,FILE2HEX))!=NO_ERR)
 { printf("%s %s\n",DSK_APP,Error_Strg(err));
   exit(0);
 }
}
strcpy(ptr0,".DSK");
//
//
if((err=Load_File(DSK_APP,LOADHEX))!=NO_ERR)
{ printf("%s %s\n",DSK_APP,Error_Strg(err));
  exit(0);
}
//
//
RUN_CPU();
init_graphics();
//
// Init buff ptrs, Temps are rotated
//
 tmp0 = buf_0;
 tmp1 = buf_1; tmp2 = buf_2; tmp3 = buf_3; tmp4 = buf_4;
 tmp5 = buf_5; tmp6 = buf_6; tmp7 = buf_7; tmp8 = buf_8;
 ptr0 = tmp0;
 for(x=0;x<Samples/2;x++)                // Clear all buffersq
 {
```

```
 buf_1[x] = 0; buf_2[x] = 0; buf_3[x] = 0; buf_4[x] = 0;
 buf_5[x] = 0; buf_6[x] = 0; buf_7[x] = 0; buf_8[x] = 0;

 buf_0[x]   = 0;
 buf_y_pk[x] = 255;
 buf_pk_t[x] = 8;
 oldbuf[x]  = 255;
}
check_key();
draw_vect(); //
draw_peak(); // Draw peaks and vector displays using XOR write
//*************************************************************//
// The inner loop is repeated until a keyboard hit exits the //
// application or the application reports an error and needs //
// to be reinitialized                                       //
//*************************************************************//


//for(int loop=0;loop<1000;loop++)
 for(;;)
 { // The inner loop draws the display using an XOR line draw
  // to overdraw existing vectors
  ptr0 = tmp0;                          // Set buffer pointers
  ptr1 = tmp1; ptr2 = tmp2; ptr3 = tmp3; ptr4 = tmp4;
  ptr5 = tmp5; ptr6 = tmp6; ptr7 = tmp7; ptr8 = tmp8;
  for(x=0;x<(Samples/2);x++)
  { switch(avg_on)
   { case  1: *ptr0=(*ptr1+*ptr2)          >> 1; break; // avg 2
     case  2: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4) >> 2; break; // avg 4
     case  3: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4+          // avg 8
                *ptr5+*ptr6+*ptr7+*ptr8) >> 3; break;
     default: *ptr0= *ptr1;                 break; // avg 1
```

```
  }
  ptr0++;
  ptr1++; ptr2++; ptr3++; ptr4++;            // next data
  ptr5++; ptr6++; ptr7++; ptr8++;
 }
 draw_vect();
 draw_peak();
 // To prevent getmem timeouts an artificial strobe is used to
 // signal to the DSK that a host request is in progress
 HPI_STRB(0);          // Drive HPSTB (INT2) low and wait
 reset_flag = 0;
 for(;;)              // for DSK to stop with full buffer
 { if(kbhit())
   {
    reset_flag = check_key();
    New_Params=1;
   }
  if(HPI_ACK())break;  // Note: break last to ensure keytrap!
   delay(1);
 }
 if(reset_flag) break;

 ptr1 = tmp8;                  // Rotate the buffer pointers
 tmp8 = tmp7; tmp7 = tmp6; tmp6 = tmp5; tmp5 = tmp4;
 tmp4 = tmp3; tmp3 = tmp2; tmp2 = tmp1; tmp1 = ptr1;
 //
 // If a key was pressed, update the AIC setup
 if(New_Params)
 {
  if(putmem(T0_ctrl ,1,&ClockMd)!=NO_ERR) break;
  if(putmem(T0_prd  ,1,&T0_prdv)!=NO_ERR) break;
  if(putmem(T0_count,1,  &ZERO)!=NO_ERR) break;
```

```
    if(putmem(T1_prd ,1,&T0_prdv)!=NO_ERR) break;
    if(putmem(T1_count,1,  &ZERO)!=NO_ERR) break;
    if(TB>TA)
    {
     aic = A_REG; if(putmem(A_BOX,1,&aic)!=NO_ERR) break;
     aic = B_REG; if(putmem(B_BOX,1,&aic)!=NO_ERR) break;
    }
    else
    {
     aic = B_REG; if(putmem(A_BOX,1,&aic)!=NO_ERR) break;
     aic = A_REG; if(putmem(B_BOX,1,&aic)!=NO_ERR) break;
    }
    aic = C_REG; if(putmem(C_BOX,1,&aic)!=NO_ERR) break;
    if(putmem(LOAD_BOX,1,&aic)!=NO_ERR) break; // Any nonzero reinits AIC
   }
   New_Params = 0;
   if(getmem(DATABLOCK,Samples/8,(ulong *)ptr1)!=NO_ERR) break;//256 char
   /*----------------------------*/
   /* Scale contents of ptr1 here */
   /*----------------------------*/
   putmem(MSG_BOX,1,&MSG);
  }
  closegraph();  // Shutdown graphics before re-initializing
  printf("%s: %s\n",DSK_APP, Error_Strg(err));
  printf("Communications are being reinitialized");
  delay(1000);
  DSK_reset();
 }
}

int old_key=0;
int accel=1;
```

```
long inline clip(long x, int min, int max)
{
 if(x<min) return min;
 if(x>max) return max;
 return x;
}

int check_key(void)
{
 float Ftim0;
 int key=0;
 char buf[80];
 int Yt=0;

 if(kbhit()) key = bioskey(0) & 0xFF00;

 if(old_key==key) accel = accel + 1;
 else             accel = accel / 4;
 accel = clip(accel,1,200); old_key = key;
 if(key)
 {
  switch(key)
  {
   case _Ins: T0_prdv -=accel; break;
   case _Del: T0_prdv +=accel; break;
   case _R  : return 1;  // Return reset (break) flag
   case _Q  :closegraph();
             _setcursortype(_NORMALCURSOR);
             exit(0);
             break;
   case _C  : ClockMd =0x3C1; break;
```

```
 case _P  : ClockMd =0x2C1; break;
 case _F1 : TA++; break;
 case _F2 : TA--; break;
 case _F3 : TB++; break;
 case _F4 : TB--; break;
 case _F5 : RA++; break;
 case _F6 : RA--; break;
 case _F7 : RB++; break;
 case _F8 : RB--; break;
 case _1  : C_REG = C_REG ^ 0x80; break; // bit 7
 case _2  : C_REG = C_REG ^ 0x40; break; // bit 6
 case _3  : C_REG = C_REG ^ 0x20; break; // bit 5
 case _4  : C_REG = C_REG ^ 0x10; break; // bit 4
 case _5  : C_REG = C_REG ^ 0x08; break; // bit 3
 case _6  : C_REG = C_REG ^ 0x04; break; // bit 2
 case _A  :
        switch(avg_on)
        {
         case  1: avg_on=2; break;
         case  2: avg_on=3; break;
         case  3: avg_on=4; break;
         default: avg_on=1; break;
        }
        break;

 default : return 0;
}
TA      = clip(TA      ,  3,      31);
TB      = clip(TB      , 12,      63);
RA      = clip(RA      ,  3,      31);
RB      = clip(RB      , 12,      63);
T0_prdv = clip(T0_prdv,   0,      64);
```

```
}
setwritemode(0);                // turn off XOR write
menu_vwport();                  // Write to window in bright blue
clearviewport();
setcolor(11);

Yt=-5;
//
// Calculate runtime values
//
if(ClockMd & 0x100)
{
 if(T0_prdv ==0) Ftim0 = 1/(2*THx);
 else           Ftim0 = 1/(4*THx * T0_prdv);
}
else
{
 if(T0_prdv ==0) T0_prdv = 1;
 Ftim0 = 1/(2*THx * T0_prdv);
}

 Fsx    = Ftim0/(2*TA*TB);
//Fsx    = 1/(2*TA*TB*(2*THx * T0_prdv));
 if(C_REG & 0x20) Fsr = Fsx;
 else
 Fsr = Ftim0/(2*RA*RB);              //Fsr = 1/(2*RA*RB*(2*THx * T0_prdv));
 Hz_per_div = (520.0/Samples)*Fsr/(2.0*10);
 //
 // Display the runtime values
 //
 #define C1 1
 switch(avg_on)
```

```
{ case  1: sprintf(buf,"[Q]uit [R]eset [A]vg(2)");break;
 case  2: sprintf(buf,"[Q]uit [R]eset [A]vg(4)");break;
 case  3: sprintf(buf,"[Q]uit [R]eset [A]vg(8)");break;
 default: sprintf(buf,"[Q]uit [R]eset [A]vg(1)");break;
}                              outtextxy(C1,Yt+=10,buf);
sprintf(buf,"F1:F2  TA  = %02d" ,   TA); outtextxy(C1,Yt+=10,buf);
sprintf(buf,"F3:F4  TB  = %02d" ,   TB); outtextxy(C1,Yt+=10,buf);
sprintf(buf,"F5:F6  RA  = %02d" ,   RA); outtextxy(C1,Yt+=10,buf);
sprintf(buf,"F7:F8  RB  = %02d" ,   RB); outtextxy(C1,Yt+=10,buf);
sprintf(buf,"AIC CTRL:  ");
binsprintf(buf+10,C_REG);                outtextxy(C1,Yt+=10,buf);
sprintf(buf," Use [1-6] keys");          outtextxy(C1,Yt+=10,buf);
//
//
#define C2 200
Yt=-5;
sprintf(buf,  "Hz/div=%7.2f",Hz_per_div);   outtextxy(C2,Yt+=10,buf);
sprintf(buf,  " Fdac=%7.2f",Fsx);           outtextxy(C2,Yt+=10,buf);
if(C_REG&0x20) sprintf(buf," Fadc=Fdac");
else        sprintf(buf," Fadc=%7.2f",Fsr);outtextxy(C2,Yt+=10,buf);
#define C3 400
Yt=-5;
sprintf(buf,"[Ins]/[Del] TIM0=%02ld",T0_prdv); outtextxy(C3,Yt+=10,buf);
if(ClockMd & 0x100)
sprintf(buf,"T0_ctrl %08lx Clk",ClockMd);
else
sprintf(buf,"T0_ctrl %08lx Pls",ClockMd);   outtextxy(C3,Yt+=10,buf);
sprintf(buf,"[C/P] Clock/Pulse mode");     outtextxy(C3,Yt+=10,buf);
sprintf(buf,"T0 freq %7.2f",Ftim0);        outtextxy(C3,Yt+=10,buf);
```



```
graph_vwport();
```

```
 return 0;
}

void init_graphics(void)
{
 int gdriver = EGA, gmode = EGAHI, errorcode, Y, X;
//
// register EGAVGA_driver which is the name of the driver in EGAVGA.OBJ
// EGAVGA.OBJ is created from EGAVGA.BGI using the BGIOBJ.EXE utility
// and is linked by either the link list or project file
//
// NOTE: This section of code can be ommitted if EGAVGA.BGI is
//       located in the applications startup directory
//
errorcode = registerbgidriver(EGAVGA_driver);
if (errorcode < 0)  // report any registration errors
{
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1); // terminate with an error code
}

 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();                       //
```

```
graph_vwport();
setcolor(GREEN);

#define Xd 520
#define xd 52

for(Y=0;Y<=Xd;Y+=xd/2) line(0,Y,Xd,  Y); // draw reticle
setlinestyle(DOTTED_LINE, 1, 1);
for(X=0;X<=Xd;X+=xd/2) line(X,0, X, Xd);

setlinestyle(SOLID_LINE, 1, 1);
for(X=0;X<=Xd;X+=xd) line(X,0, X, Xd);

Y = 2;
X = 2;
outtextxy(X,Y    ,"+20");
outtextxy(X,Y+=26,"+10");
outtextxy(X,Y+=26," +0");
outtextxy(X,Y+=26,"-10");
outtextxy(X,Y+=26,"-20");
outtextxy(X,Y+=26,"-30");
outtextxy(X,Y+=26,"-40");
outtextxy(X,Y+=26,"-50");
outtextxy(X,Y+=26,"-60");
outtextxy(X,Y+=26,"-70");

setwritemode(1);                  // display lines are XOR drawn
setcolor(15);                     // light gray
}

void binsprintf(char *s,int val)
{
```

```
  char *p;
  unsigned int t;
  p = s;
  t = 0x80;  // scan 8 bits
  for(;t>0;t=t>>1)
  {
   if(val & t) *p++ = '1';
   else        *p++ = '0';
  }
  *p=0;
}
```

ภาคผนวก ข

Solid State Sensors
Linear Current Sensors

CS Series

FEATURES
- Linear output
- AC or DC current sensing
- Through-hole design
- Fast response time
- Output voltage isolation from input
- Minimum energy dissipation
- Maximum current limited only by conductor size
- Adjustable performance and built-in temperature compensation assures reliable operation
- Accurate, low cost sensing
- Operating temperature range –25 to 85° C
- Housing, PET polyester

LINEAR CURRENT SENSORS
MICRO SWITCH CS series linear current sensors incorporate our 91SS12-2 and SS94A1 linear output Hall effect transducer (LOHET™). The sensing element is assembled in a printed circuit board mountable housing. This housing is available in four configuration as shown in mounting dimension figures 1, 1a, 2 and 2a. Normal mounting is with 0.375 inch long 4-40 screw and square nut (not provided) inserted in the housing or a 6-20 self-tapping screw. The combination of the sensor, flux collector, and housing comprises the holder assembly. These sensors are ratiometric.

ORDER GUIDE — BOTTOM MOUNT WITH 9SS SENSOR, SOURCE OUTPUT

Catalog Listing	Mtg. Dim. Fig.	Supply Volt. (Volts DC)	Supply Current (mA Max.)	Sensed Current (Amps Peak)	Offset Volt. (Volts ± 10%)	Sensitivity mV-N* At 12 VDC		Offset Shift (%/°C)	Response Time (µ Sec.)
						Nominal	± TOL.		
CSLA1CD	1	9 to 15	19	67	Vcc/2	49.6	5.8	± .05	3
CSLA1CE	1	9 to 15	19	75	Vcc/2	39.4	4.4	± .05	3
CSLA1DE	2	9 to 15	19	75	Vcc/2	39.1	4.8	± .05	3
CSLA1CF	1	9 to 15	19	100	Vcc/2	29.7	2.7	± .05	3
CSLA1DG	2	9 to 15	19	120	Vcc/2	24.6	2.1	± .05	3
CSLA1CH	1	9 to 15	19	150	Vcc/2	19.6	1.8	± .05	3
CSLA1DJ	2	9 to 15	19	225	Vcc/2	13.2	1.2	± .05	3
CSLA1EJ	1a	9 to 15	19	225	Vcc/2	13.2	1.5	± .05	3
CSLA1DK	2	9 to 15	19	325	Vcc/2	9.1	1.7	± .05	3
CSLA1EK	1a	9 to 15	19	325	Vcc/2	9.4	1.3	± .05	3
CSLA1EL	1a	9 to 15	19	625	Vcc/2	5.6	1.3	± .05	3

BOTTOM MOUNT WITH SS9 SENSOR, SINK/SOURCE OUTPUT

Catalog Listing	Mtg. Dim. Fig.	Supply Volt. (Volts DC)	Supply Current (mA Max.)	Sensed Current (Amps Peak)	Offset Volt. (Volts ± 2%)	Sensitivity mV-N* At 8 VDC		Offset Shift (%/°C)	Response Time (µ Sec.)
						Nominal	± TOL.		
CSLA2CD	1	5 to 12	20	72	Vcc/2	32.7	3.0	± .02	3
CSLA2CE	1	5 to 12	20	92	Vcc/2	26.1	2.1	± .02	3
CSLA2DE	2	5 to 12	20	92	Vcc/2	25.6	2.2	± .02	3
CSLA2CF	1	5 to 12	20	125	Vcc/2	19.6	1.3	± .02	3
CSLA2DG	2	5 to 12	20	150	Vcc/2	16.2	1.1	± .02	3
CSLA2DJ	2	5 to 12	20	225	Vcc/2	9.7	0.6	± .020	3
CSLA2DH	2	5 to 12	20	235	Vcc/2	9.8	1.1	± .0125	3
CSLA2EJ	1a	5 to 12	20	310	Vcc/2	7.6	0.7	± .0125	3
CSLA2DK	2	5 to 12	20	400	Vcc/2	5.8	0.5	± .0125	3
CSLA2EL	1a	5 to 12	20	550	Vcc/2	4.3	0.4	± .0125	3
CSLA2EM	1a	5 to 12	20	765	Vcc/2	3.1	0.3	± .007	3
CSLA2EN	1a	5 to 12	20	950	Vcc/2	2.3	0.2	± .007	3

NOTE: When monitoring purely AC current with zero DC component, a capacitor can be inserted in series with the output of the current sensor. The capacitor will block out the effect of the temperature variation of the offset voltage which increases the accuracy of the device.

*N = number of turns

Solid State Sensors

CS Series

Linear Current Sensors

SIDE MOUNT WITH 9SS SENSOR, SOURCE OUTPUT

Catalog Listing	Mtg. Dim. Fig.	Supply Volt. (Volts DC)	Supply Current (mA Max.)	Current (Amps Peak)	Sensed Offset Volt. (Volts ± 10%)	Sensitivity mV-N* At 12 VDC Nominal	Sensitivity mV-N* At 12 VDC ± TOL	Offset Shift (%/°C)	Response Time (µ Sec.)
CSLA1GD	2a	9 to 15	19	50	Vcc/2	49.6	5.8	± .05	3
CSLA1GE	2a	9 to 15	19	75	Vcc/2	39.4	4.4	± .05	3
CSLA1GF	2a	9 to 15	19	100	Vcc/2	29.7	2.7	± .05	3

SIDE MOUNT WITH SS9 SENSOR, SINK/SOURCE OUTPUT

Catalog Listing	Mtg. Dim. Fig.	Supply Volt. (Volts DC)	Supply Current (mA Max.)	Sensed Current (Amps Peak)	Offset Volt. (Volts ± 2%)	Sensitivity mV-N* At 8 VDC Nominal	Sensitivity mV-N* At 8 VDC ± TOL	Offset Shift (%/°C)	Response Time (µ Sec.)
CSLA2GD	2a	5 to 12	20	72	Vcc/2	32.7	3.0	± .02	8
CSLA2GE	2a	5 to 12	20	92	Vcc/2	26.1	2.1	± .02	8
CSLA2GF	2a	5 to 12	20	125	Vcc/2	19.6	1.3	± .02	8
CSLA2GG	2a	5 to 12	20	150	Vcc/2	13.7	0.6	± .02	8

NOTE: When operating purely AC current with zero DC component, a capacitor can be inserted in series with the output of the current sensor. The capacitor will block out the effect of the temperature variation of the offset voltage which can decrease the accuracy of the device.

*N = number of turns.

MOUNTING DIMENSIONS (for reference only)

Figure 1

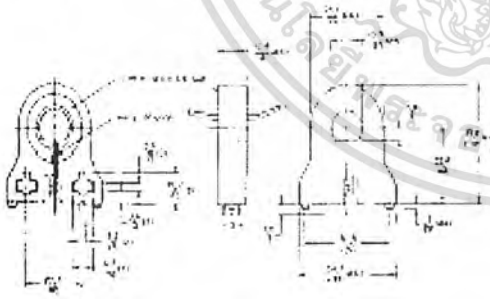


Figure 2

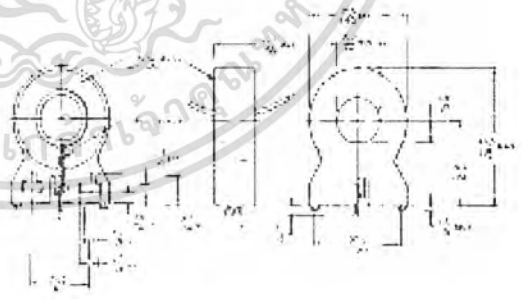


Figure 1a

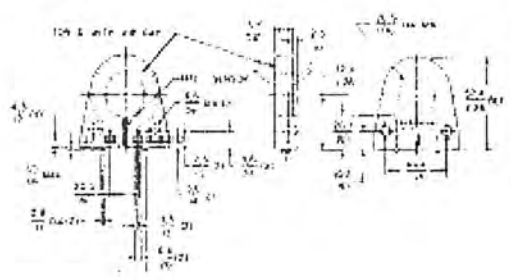


Figure 2a