สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

เครื่องวิเคราะห์แถบความถี่ของสัญญาณโดยใช้ไมโครคอมพิวเตอร์

PC SPECTRUM ANALYZER

นาย ไพฑูรย์ ศรีนิล
นาย เอกชัย ไชยกูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

เลขหมู่.....
เลขทะเบียน 34105
วัน, เดือน, ปี - 5 ต.ค. 2542

เครื่องวิเคราะห์แถบความถี่ของสัญญาณโดยใช้ไมโครคอมพิวเตอร์

PC SPECTRUM ANALYZER

โดย

นาย ไพฑูรย์ ศรีนิล

นาย เอกชัย ไชยกูล

อาจารย์ที่ปรึกษา

อ. อภิเนตร อูนากูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องวิเคราะห์แถบความถี่ของสัญญาณโดยใช้ไมโครคอมพิวเตอร์

PC Spectrum Analyzer

ผู้จัดทำ

1. นาย ไพฑูรย์ ศรีนิล รหัสประจำตัว 39013245
2. นาย เอกชัย ไชยกูล รหัสประจำตัว 39013262

---------- อาจารย์ที่ปรึกษา

(อ. อภิเนตร อูนากูล)

เครื่องวิเคราะห์แถบความถี่โดยใช้เครื่องไมโครคอมพิวเตอร์

นาย ไพฑูรย์ ศรีนิล 39013245
นาย เอกชัย ไชยกูล 39013262
อ. อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์นี้ได้นำเสนอการนำเครื่องไมโครคอมพิวเตอร์มาวิเคราะห์แถบความถี่ของสัญญาณ โดยใช้อุปกรณ์ต่อร่วมในการทำงาน คือ ชุดพัฒนา DSP Start Kit ซึ่งเป็นชุดไมโครคอนโทรลเลอร์ที่มีโปรแซสเซอร์ TMS320C50 เป็นตัวประมวลผลกลาง ทำหน้าที่รับสัญญาณอนาล็อคที่ต้องการนำมาวิเคราะห์แปลงเป็นสัญญาณดิจิตอลแล้วนำมาวิเคราะห์หาแถบความถี่โดยโปรแกรม FFT จากนั้นส่งผลการวิเคราะห์ไปแสดงผลที่เครื่องไมโครคอมพิวเตอร์โดยการแสดงผลจะแสดงเป็นภาพกราฟิกเพื่อง่ายต่อการอ่านและแสดงผลเป็นตัวเลขเพื่อการอ่านที่แม่นยำถูกต้องตามค่าจริงที่วัดได้

PC SPECTRUM ANALYZER

Mr. Phaitoon Srinil
Mr. Ackachai Chaiyakool
Mr. Apinetr Unakul (Advisor)
1999

ABSTRACT

This project presents the design of a PC spectrum analyzer which uses a PC as a monitoring and displaying component and uses a TMS320C50 board as a data acquisition and analysis component. The PC Spectrum Analyzer can be used to analyze real-time sound spectrum with up to 4 KHz.

กิตติกรรมประกาศ

โครงงานนี้นับตั้งแต่เริ่มต้นจนกระทั่งสำเร็จเป็นโครงงานที่สมบูรณ์ส่วนหนึ่งเป็นเพราะท่านอาจารย์ที่ปรึกษา อ.อภิเนตร อูนากูล ที่ท่านได้สละเวลาอันมีค่ามาให้คำปรึกษาตลอดจนแนะนำและช่วยแก้ไขปัญหาต่างๆที่เกิดขึ้นในระหว่างการทำงาน

และต้องขอพระคุณท่านคุณพ่อ คุณแม่ และ พี่น้อง ที่ให้คำปลอบโยนและกำลังใจตลอดเมื่อยามท้อแท้และต้องการกำลังใจ ขอบคุณ จเด็จ สำหรับข้อมูลโปรแกรม พี่หลุยส์, พี่กุล และ บาว สำหรับขนมและกับข้าวทุกมื้อในเวลาทำงานและเพื่อนๆทุกคนที่ให้กำลังใจเสมอมา

ไพฑูรย์ ศรีนิล

เอกชัย ไชยกูล

สารบัญ

	หน้า
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญรูปประกอบ	VI
สารบัญตารางประกอบ	VII
บทที่1 บทนำ	1
บทที่2 ทฤษฎีและหลักการ	4
2.1 ทฤษฎีสเปคตรัมความถี่	4
ฟังก์ชั่นเป็นคาบ (Periodic Function)	4
อนุกรมฟูเรียร์ตรีโกณมิติ(Trigonometric Fourier Series)	5
อนุกรมฟูเรียร์เชิงซ้อน (Complex Form of Fourier Series)	7
สเปคตรัมความถี่เชิงซ้อน (Complex Frequency Spectrum)	9
สเปคตรัมเชิงกำลัง (Power Spectrum) ของฟังก์ชั่นคาบ	12
2.2 ทฤษฎีการสุ่มตัวอย่าง(Sampling)	15
2.3 ทฤษฎีการแปลงฟูร์เรียร์แบบเร็ว(Fast Fourier Transform)	17
ทฤษฎีดีสครีทฟูเรียร์ทรานฟอร์ม *(The Discrete Fourier Transform)*	17
ทฤษฎีฟาสต์ฟูเรียร์ทรานฟอร์ม *(FFT)* และวิธีการแบบบัตเตอร์ฟลาย	18
การคำนวณโดยใช้คอมพิวเตอร์	20
2.4 โปรแกรมแบบเรียลไทม์ (Real Time Programming)	22
2.5 สถาปัตยกรรม TMS320C50	23
การจัดตำแหน่งขาของ TMS320C50	26
การจัดหน่วยความจำของ TMS320C50	32
โหมดการอ้างถึงหน่วยความจำ (Memory Addressing Mode)	37
อุปกรณ์เสริม (Peripherals)	38
2.6 DSP Start Kit Board with TMS320C50	41
โครงสร้างของ TMS320C50 DSP Starter Kit	41
คุณสมบัติของ TLC32040 บนบอร์ด	42
การสร้างโปรแกรมเพื่อใช้กับ DSK	43
การเขียนโปรแกรมและคำสั่ง	44
วงจรอินเทอร์เฟสสัญญาณอนาล็อก TLC32040	47

บทที่ 3 การออกแบบและการสร้าง 56
3.1 การเชื่อมต่อทางฮาร์ดแวร์ 56
3.2 การออกแบบทางด้านซอร์ฟแวร์ 57
การคำนวณ FFT โดยใช้คอมพิวเตอร์ 57
การออกแบบซอร์ฟแวร์บนเครื่องไมโครคอมพิวเตอร์ 57
การออกแบบซอร์ฟแวร์บน DSK Board 60
3.3 เรียลไทม์สวิทซ์ 64
บทที่ 4 การทดลองและผลการทดลอง 66
การวัดความเร็วของโปรแกรมการคำนวณ FFT 66
การวัดความเร็วของโปรแกรมส่งข้อมูลมายังเครื่องไมโครคอมพิวเตอร์ 68
การทดสอบการวัดแถบความถี่ของรูปคลื่นไซน์โดยใช้ PC Spectrum Analyzer 70
การทดสอบผลการวิเคราะห์กับโปรแกรม MATLAB 72
บทที่ 5 สรุปผลและวิจารณ์ 74
บรรณานุกรม 76
ภาคผนวก 77

สารบัญรูปประกอบ

	หน้า
รูปที่ 1.1 การประมวลผลเฉพาะอย่าง	1
รูปที่ 1.2 การใช้งานไมโครโปรแซสเซอร์ประมวลผล	1
รูปที่ 1.3 ส่วนประกอบของ DSP	2
รูปที่ 1.4 การใช้งานชุดพัฒนา DSP Start Kit เป็นเครื่องสเปคตรัมอนาไลซ์เซอร์	3
รูปที่ 2.1.1 แสดงฟังก์ชั่นเป็นคาบ	4
รูปที่ 2.1.2 (ก) สัญญาณ(ฟังก์ชั่น)ไซน์และโคไซน์ที่มีคาบครบรอบเท่ากับ 2π	5
รูปที่ 2.1.2 (ข) สัญญาณไซน์ที่ฮาร์โมนิก n = 1,2 และ 3	5
รูปที่ 2.1.3 แสดงการแทนฟังก์ชั่นรูปสีเหลี่ยมด้วยฟังก์ชั่นไซน์และโคไซน์	6
รูปที่ 2.1.4 (ก) สัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น	10
รูปที่ 2.1.4 (ข)ไลน์สเปคตรัมของสัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น	10
รูปที่ 2.1.5 (ก) สัญญาณพัลส์รูปสี่เหลี่ยมพื้นผ้า	10
รูปที่ 2.1.5 (ข) สเปคตรัมเชิงแอมปริจูด	11
รูปที่ 2.1.6 สเปคตรัมเชิงแอมปริจูด	12
รูปที่ 2.1.7 (ก) สัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น	14
รูปที่ 2.1.7 (ข) สเปคตรัมเชิงกำลังของสัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น	14
รูปที่ 2.1.8 (ก) สัญญาณพัลส์รูปสี่เหลี่ยมพื้นผ้า	15
รูปที่ 2.1.8 (ข) สเปคตรัมเชิงกำลังของสัญญาณพัลส์รูปสี่เหลี่ยมพื้นผ้า	15
รูปที่ 2.2.1 การประมวลผลเชิงอนุมานโดยใช้ระบบการประมวลผลสัญญาณเชิงเลข	15
รูปที่ 2.2.2 แสดงตัวอย่างของความถี่ในการสุ่มสัญญาณ Ts = $1/(4f_O)$	16
รูปที่ 2.3.1 ดีสครีทฟูเรียร์ทรานฟอร์ม	17
รูปที่ 2.3.2 (ก) การหาค่า 8 จุดฟาสต์ฟูเรียร์ทรานฟอร์ม	19
รูปที่ 2.3.2 (ข) การหาค่า 8 จุดฟาสต์ฟูเรียร์ทรานฟอร์ม	20
รูปที่ 2.3.2 (ค) หลักการคำนวณโดยใช้บัตเตอร์ฟลายเดียว	20
รูปที่ 2.3.3 (ก) อินพุท 16 จุด ฟาสต์ฟูเรียร์ทรานฟอร์ม	21
รูปที่ 2.3.3 (ข) เอาท์พุทของ 16 จุด ฟาสท์ฟูเรียร์ทรานฟอร์ม	22
รูปที่ 2.4.1 (ก) โปรแกรมแบบอนุกรม(Serial Control)	22
รูปที่ 2.4.1 (ข) โปรแกรมแบบขนาน(Concurrent Control)	23
รูปที่ 2.5.1 บล็อกไดอะแกรมของ TMS320C50	25
รูปที่ 2.5.2 แสดงการจัดตำแหน่งของหน่วยความจำของ TMS320C50	32
รูปที่ 2.5.3 แสดงการอินเตอร์รัพต์แฟลก	39
รูปที่ 2.5.4 แสดงการต่อพอร์ตอนุกรมกับพอร์ตภายนอก	39

หน้า

รูปที่ 2.5.5 แสดงบล็อกไดอะแกรมการทำงานของพอร์ตอนุกรม 40
รูปที่ 2.5.6 แสดงบล็อกไดอะแกรมของไทม์เมอร์ 41
รูปที่ 2.6.1 บล็อกไดอะแกรมของ TMS320C5X DSK 42
รูปที่ 2.6.2 แสดงหน่วยความจำภายใน TMS320C50 DSK 42
รูปที่ 2.6.3 แสดงการเชื่อมต่อระหว่าง DSK และ PC ผ่านทาง RS232 43
รูปที่ 2.6.4 แสดงขั้นตอนการสร้างโปรแกรม 44
รูปที่ 2.6.5 แสดงการกำหนดค่าใน PMST 44
รูปที่ 2.6.6 แสดงฟังก์ชั่นบล็อกไดอะแกรม ของ TLC 32040 48
รูปที่ 2.6.7 แสดงการเชื่อมต่อระหว่าง TLC32040 กับ TMS320C50 53
รูปที่ 2.6.8 แสดงช่วงเวลาในการส่งและรับข้อมูลระหว่าง TLC32040 กับ TMS320C50 54
รูปที่ 3.1.1 แสดงการต่อสายสัญญาณ RS232 โดยผ่าน DB9/DB25 Connector 56
รูปที่ 3.2.1 หลักการคำนวณโดยใช้บัตเตอร์ฟลายเดียว 57
รูปที่ 3.2.2 การทำงานของโปรแกรมบนเครื่องไมโครคอมพิวเตอร์ 58
รูปที่ 3.2.3 แสดงโครงสร้างของไฟล์ DSK ที่ทำการอัพโหลด 59
รูปที่ 3.2.4 แสดงการแฮนด์เช็คระหว่างการรับข้อมูล 59
รูปที่ 3.2.5 การทำงานของโปรแกรมบน TMS320C50 60
รูปที่ 3.2.6 การทำงานของโปรแกรมบริการอินเทอร์รัพท์การรับข้อมูลอนุกรม 61
รูปที่ 3.2.7 แสดงไดอะแกรมการทำงานของการคำนวณฟาสท์ฟูเรียร์ 62
รูปที่ 3.2.8 แสดงการทำงานของฟังก์ชั่น To_Host 63

สารบัญตารางประกอบ

หน้า

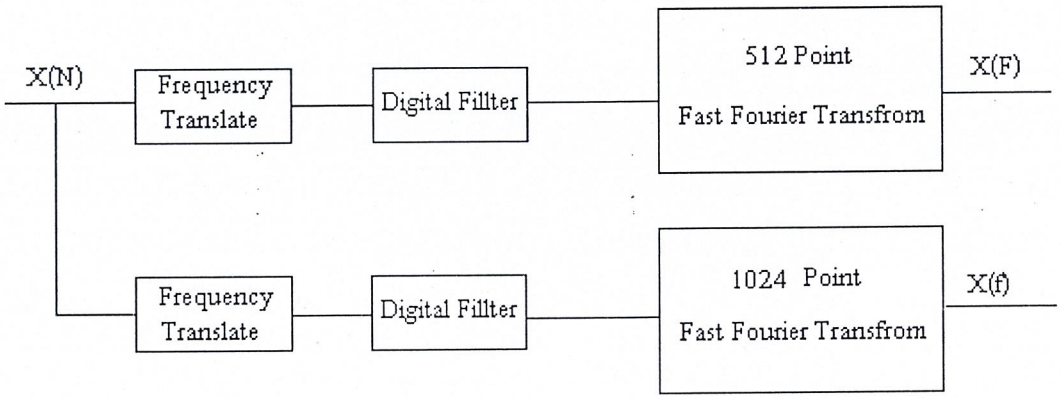
ตารางที่ 2.3.1 แสดงการเปรียบเทียบจำนวนการคำนวณระหว่างดีสครีทฟูเรียร์ทรานฟอร์ม และฟาสต์ฟูเรียร์ทรานฟอร์ม 21
ตารางที่ 2.5.1 แสดงตำแหน่งขาและหน้าที่การทำงานของ TMS320C50 26
ตารางที่ 2.5.2 แสดงการเซ็ตบิตรีจิสเตอร์ควบคุมหน่วยความจำเพื่อใช้หน่วยความจำข้อมูล 33
ตารางที่ 2.5.3 แสดง ตำแหน่งแอดเดรสข้อมูล เพจ 0 33
ตารางที่ 2.5.4 แสดงการกำหนดค่าสำหรับใช้หน่วยความจำโปรแกรม 36
ตารางที่ 2.5.5 แสดงตำแหน่งของอินเทอร์รัพต์เวกเตอร์ 36
ตารางที่ 2.5.6 แสดงขาสัญญาณของพอร์ตอนุกรม 39
ตารางที่ 2.5.7 แสดงรีจิสเตอร์พอร์ตอนุกรม 40
ตารางที่ 2.5.8 แสดงรีจิสเตอร์ควบคุมไทม์เมอร์ (TCR) 41
ตารางที่ 2.6.1 แสดงตำแหน่งขาและหน้าที่การทำงานของแต่ละขา 48
ตารางที่ 2.6.2 แสดงการกำหนดค่าใน D1 และ D0 ในการทำงานในโหมดต่างๆของ AIC 52
ตารางที่ 2.6.3 แสดงเงื่อนไขของการตอบสนอง AIC 54
ตารางที่ 3.3.1 แสดงเวลาที่แต่ละโปรแซสใช้ในการทำงานจนสำเร็จในรอบการทำงาน 64

บทที่ 1

บทนำ

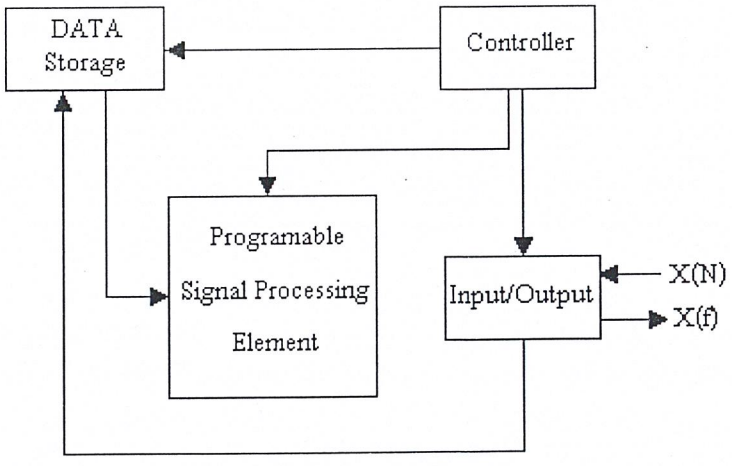
การประมวลผลสัญญาณเชิงเลข (Digital Signal Processing) ก็คือการเปลี่ยนรูปสัญญาณโดยการใช้การคำนวนทางคณิตศาสตร์โดยโปรแซสเซอร์ประมวลผลสัญญาณเชิงเลข ซึ่งจะเห็นได้ว่าเราสามารถนำสัญญาณนั้นใส่ลงไปในสมการคณิตศาสตร์ แล้วได้ผลลัพท์โดยตรงทีเดียว ในช่วง 10 ปีที่ผ่านมา ความก้าวหน้าทางด้านการประมวลผลสัญญาณเชิงเลขได้เกิดขึ้นเป็นอย่างมาก มีการค้นพบอัลกอริทึม (Algorithms) ใหม่ๆเพิ่มขึ้นเรื่อยๆและการพัฒนาเครื่องมือใหม่ๆ เพื่อช่วยในด้านการออกแบบตามอัลกอริทึมอีกด้วย ระบบประมวลผลสัญญาณเชิงเลขนี้มีอยู่ด้วยกัน 2 ระบบใหญ่ๆคือ

1.1 ระบบที่ออกแบบมาเพื่อใช้ในงานใดงานหนึ่งโดยเฉพาะดังแสดงในรูปที่ 1.1



รูปที่ 1.1 การประมวลผลเฉพาะอย่าง

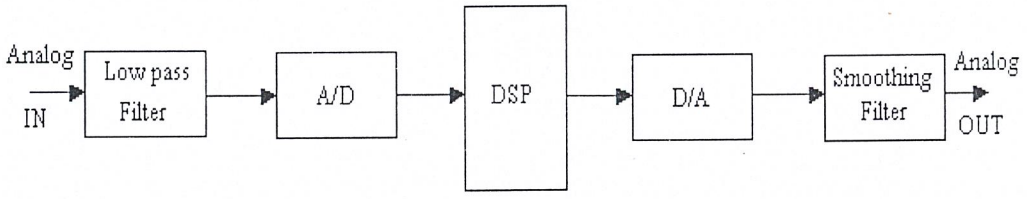
1.2 ระบบที่ใช้ไมโครโปรแซสเซอร์ ซึ่งสามารถทำงานเป็นอะไรก็ได้ขึ้นอยู่กับโปรแกรมไว้ ดังรูปที่ 1.2



รูปที่ 1.2 การใช้งานไมโครโปรแซสเซอร์ประมวลผล

ความก้าวหน้าทางอิเล็กทรอนิกส์ครั้งใหญ่เกิดขึ้น เมื่อได้มีการพัฒนาโปรแซสเซอร์ที่ทำหน้าที่ประมวลผลสัญญาณเชิงเลขแบบความเร็วสูงขึ้น ซึ่งในปัจจุบันได้มีการงานกันอย่างแพร่หลายในราคาที่ไม่แพงนักโปรแซสเซอร์ที่ทำหน้าที่ประมวลผลสัญญาณเชิงลบจำเป็นต้องมีความเร็วสูงจำเป็นต้องมีความเร็วสูงและถูกออกแบบมาโดยเฉพาะให้สามารถทำงานได้อย่างมีประสิทธิภาพโดยใช้ข้อดีของการประมวลผลแบบขนานและการใช้ชุดคำสั่ง ทั้งนี้เพื่อการประมวลผลแบบเวลาที่แท้จริง คือเมื่อเราใส่สัญญาณอินพุทเข้าไปก็จะได้สัญญาณเอาท์พุทออกมาโดยมีลายละเอียดที่ครบถ้วนและมีเวลาหน่วง (Delay)ที่น้อยมาก ซึ่งจะมีประโยชน์มากในการใช้งานมากกว่าการประมวลผลแบบที่ไม่ใช่เวลาจริง

ด้วยความสามารถของการประมวลผลสัญญาณเชิงเลข จึงนำไปใช้งานในแบบต่างๆ ซึ่งให้ผลลัพธ์ดีกว่าวงจรอนาล็อค เช่น ความน่าเชื่อ ความยืดหยุ่น การทดแทนได้ง่าย ขนาดเล็กกว่า สามารถดีบัก-แก้ไขการทำงานในส่วนที่ผิดพลาดได้ด้วยโปรแกรมทำให้ง่ายต่อการสร้าง ด้วยเหตุผลต่างๆ เหล่านี้และปัจจุบันได้มีการคิดค้นอัลกอริทึมใหม่ที่มีประสิทธิภาพในการแก้ขึ้นอีกมากมายจึงทำให้การใช้งานโปรแซสเซอร์ทำหน้าที่ในการประมวลผลสัญญาณเชิงเลขนั้นน่าสนใจเป็นอย่างสูงในปัจจุบัน ถ้าหากเปรียบเทียบการออกแบบกับการใช้งานอิเล็กทรอนิกส์แล้วในทางอิเล็กทรอนิกส์โดยส่วนใหญ่จะเกี่ยวข้องกับสัญญาณไฟฟ้า เช่น การกรองสิ่งที่ไม่ต้องการออกจากอินพุท การแยกข้อมูลออกมา หรือไม่ก็เป็นการสร้างรูปคลื่นที่ต้องการขึ้นมาหรือเปลี่ยนลักษณะทางแอมปริจูดของสัญญาณซึ่งในอดีตการกระทำเหล่านี้จะสร้างโดยวงจรอนาล็อคที่ออกแบบมาเพื่อใช้งานเฉพาะในงานนั้นๆ นอกจากนั้นวงจรอนาล็อคยังไม่เหมาะสมที่จะสร้างเป็นวงจรขนาดใหญ่ (VLSI) อีกด้วย คุณสมบัติของวงจรอนาล็อคนั้นขึ้นอยู่กับคุณสมบัติของอุปกรณ์ที่ใช้ เช่น R, L ที่มีค่าที่ไม่แน่นอน ที่สำคัญจะมีการรั่วไหลที่แปรตามเวลาและยังขึ้นอยู่กับโวลท์เตทและอุณหภูมิของการทำงานอีกด้วยและที่สำคัญที่สุดเป็นการยากที่สุดที่จะดีบักวงจรเมื่อเกิดข้อผิดพลาดขึ้นในการทำงานของระบบ ซึ่งข้อยุ่งยากทั้งหมดที่ได้กล่าวมาสามารถแก้ไขได้โดยการใช้ DSP เข้ามาทดแทนการออกแบบวงจรอนาล็อค ส่วนประกอบของ DSP แสดงในรูปที่ 1.3

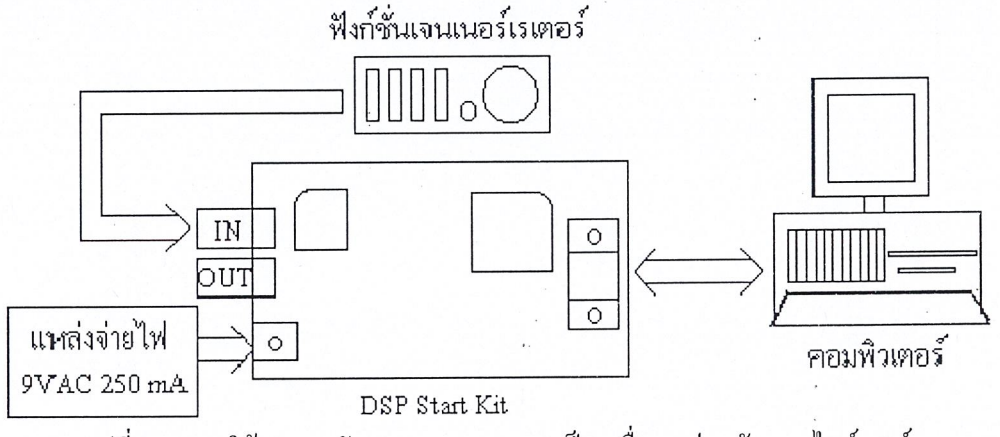


รูปที่ 1.3 ส่วนประกอบของ DSP

สัญญาณอนาล็อคอินพุทจะถูกป้อนเข้าสู่วงจรกรองความถี่ต่ำผ่านเพื่อจะจำกัดช่วงความถี่ของสัญญาณอินพุท จากนั้นก็ส่งไปยังวงจร อนาล็อค/ดิจิตอลคอนเวอร์เตอร์(Analog to Digital Converter:A/D) แล้วส่งให้ระบบประมวลผลเชิงเลขต่อไป ระบบประมวลผลเชิงเลขก็จะนำเอาข้อมูลมาจัดการตามอัลกอริทึม จากนำสัญญาณที่ทำการประมวลผลแล้วส่งไปยังส่วน ดิจิตอล/อนาล็อคคอนเวอร์เตอร์ (Digital to Analog Converter) เพื่อเปลี่ยนสัญญาณกลับไปเป็นสัญญาณอนาล็อคเช่นเดิม ส่วนวงจรกรองสัญญาณให้เรียบ(Smoothing Filter) ทำหน้าที่กรองความถี่สูงที่ไม่ต้องการออกไป แต่ระบบ DSP ก็มีข้อเสียอยู่เหมือนกัน เช่น การเกิดควันไตซ์เซชั่น (Quantization), การเกิดน๊อย(Noise), ความไม่เรียบของ

สัญญาณ เป็นต้น ซึ่งตัวแปรที่สำคัญอีกตัวหนึ่งที่จะแยกระบบ DSP ต่างๆออกตามการใช้งานคือ ค่าอัตราการสุ่มข้อมูล (Sampling Rate) ยิ่งอัตราการสุ่มข้อมูลยิ่งมากยิ่งต้องมีระบบประมวลผลที่เร็วยิ่งขึ้นด้วย อัตราการสุ่มข้อมูลที่สูงก็จะสามารถใช้งานในย่านความถี่สูงขึ้นได้โดยคุณภาพของสัญญาณข้อมูลยังคงยอมรับได้

ในอดีตการที่จะสร้างเครื่องสเปคตรัมอนาไลซ์เซอร์ขึ้นใช้งานสักเครื่องเป็นเรื่องที่ยิ่งใหญ่และยุ่งยากเอามากๆเลยทีเดียวเพราะว่าจะต้องออกแบบและสร้างด้วยวงจรอนาล็อคหรือแม้แต่ใช้ DSP ก็ตาม เป็นเพราะว่าเป็นเรื่องยากเอามากๆที่จะหาตัวชิพ DSP , คู่มือ , ซอร์ฟแวร์แอสแซมเบอร์และดีบักเกอร์ ฯลฯ แต่ในปัจจุบันไม่เป็นเรื่องยากอีกแล้ว เมื่อทางบริษัท เท็กซัสอินสตรูเมนท์ เจ้าของชิพ DSP ตระกูล TMS320Cxx ได้จัดทำชุดพัฒนาสมบูรณ์แบบ มีตัวชิพ , คู่มือและอุปกรณ์ต่างๆที่จำเป็นทั้งหมดพร้อมทั้งซอร์ฟแวร์และโปรแกรมตัวอย่างทั้งหมดจัดจำหน่ายในราคาที่ถูกพื่อต้องการให้มีการใช้งานอย่างแพร่หลายเพื่อการศึกษาและนักพัฒนาทั่วไป ซึ่งในโครงงานนี้ได้ใช้ชุดพัฒนา DSP Start Kit นำมาพัฒนาเป็นเครื่องสเปคตรัมอนาไลซ์เซอร์โดยมีการทำงานดังแสดงในรูปที่ 1.4



รูปที่ 1.4 การใช้งานชุดพัฒนา DSP Start Kit เป็นเครื่องสเปคตรัมอนาไลซ์เซอร์

จากรูปที่ 1.4 ชุดพัฒนา DSP Start Kit เป็นตัวสุ่มสัญญาณอนาล็อคจากเครื่องฟังก์ชั่นเจนเนอร์เตอร์หรือเป็นสัญญาณต่างๆที่ต้องการจะนำมาวิเคราะห์แถบความถี่ นำค่าที่สุ่มมาได้เก็บลงในหน่วยความจำบนตัวชิพ TMS320C50 แล้วให้โปรแกรมทำการประมวลผลแก้ปัญหาทางคณิตศาสตร์(FFT)เพื่อหาค่าแถบความถี่ จากนั้นส่งผลค่าคำตอบที่ได้ส่งไปแสดงผลที่จอมอร์นิเตอร์บนเครื่องไมโครโปรแซสเซอร์ โดยตัวชุดพัฒนา DSP Start Kit จะต้องการไฟเลี้ยง 9VAC 250 mA

บทที่ 2

ทฤษฎีและหลักการ

2.1 ทฤษฎีสเปคตรัมแถบความถี่

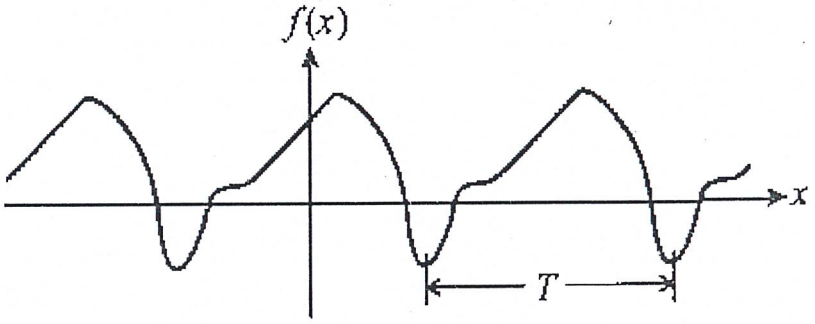
สัญญาณ (Signal) และ/หรือ ฟังก์ชั่น(Function) ต่างๆที่เกิดขึ้นในปัญหาทางกายภาพ(Physical problem) และปัญหาทางวิศวกรรม(Engineering problem)มักจะเป็นฟังก์ชั่นเป็นคาบ(Periodic function) ซึ่งฟังก์ชั่นเหล่านี้สามารถแทนให้อยู่ในเทอมหรือพจน์ของฟังก์ชั่นคาบของสัญญาณไซน์และโคไซน์ ดังแสดงในรูปที่ 2.3 ทั้งนี้แล้วสัญญาณทั้งสองจะมีความสำคัญมากเพราะเป็นสัญญาณพื้นฐานสำหรับใช้ในการพิจารณาสัญญาณอื่นๆและทั้งนี้ในการแทนแต่ละเทอมของอนุกรม(Series) ด้วยฟังก์ชั่นไซน์และโคไซน์เกิดขึ้นโดยแนวความคิดของนักฟิสิกส์และนักคณิตศาสตร์ชาวฝรั่งเศสชื่อ Joseph Fourier ดังนั้นเพื่อเป็นเกียรติแก่เขาจึงเรียกฟังก์ชั่นดังกล่าวนี้ว่า อนุกรมฟูเรียร์ (Fourier series) ซึ่งอนุกรมฟูเรียร์เป็นเหมือนเครื่องมืออันหนึ่งที่นำมาใช้แก้ปัญหาต่างๆ เช่น แก้ปัญหาเชิงอนุพันธ์สามัญ (Ordinary differential equation) และสมการเชิงอนุพันธ์ย่อย(Partial differential equation)

2.1.1 ฟังก์ชั่นเป็นคาบ (Periodic Function)

ฟังก์ชั่นเป็นคาบ $f(x)$ เรียกว่าฟังก์ชั่นที่มีคาบครบรอบ T ถ้า T เป็นค่าจริงใดและไม่เท่ากับศูนย์ ดังนั้น

$$f(x+T) = f(x) \qquad \text{ตลอดทุกค่า x} \qquad (1)$$

ซึ่ง T คือคาบครบรอบของฟังก์ชั่น $f(x)$ เช่น sin x และ cos x ที่มีคาบครบรอบเท่ากับ 2π

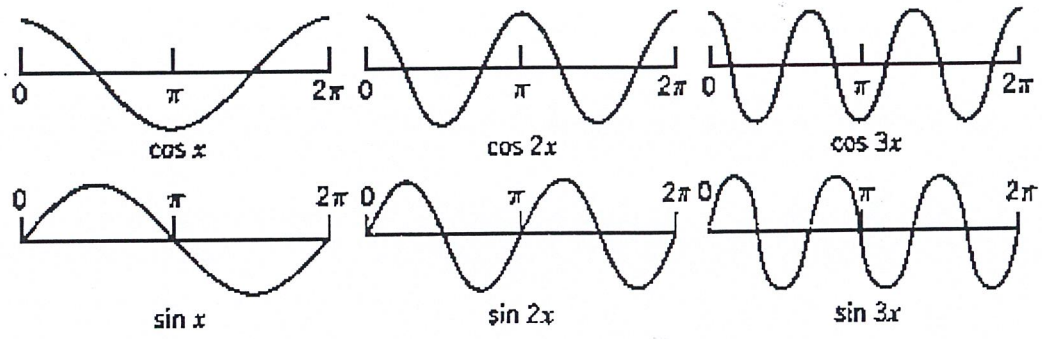


รูปที่ 2.1.1 แสดงฟังก์ชั่นเป็นคาบ

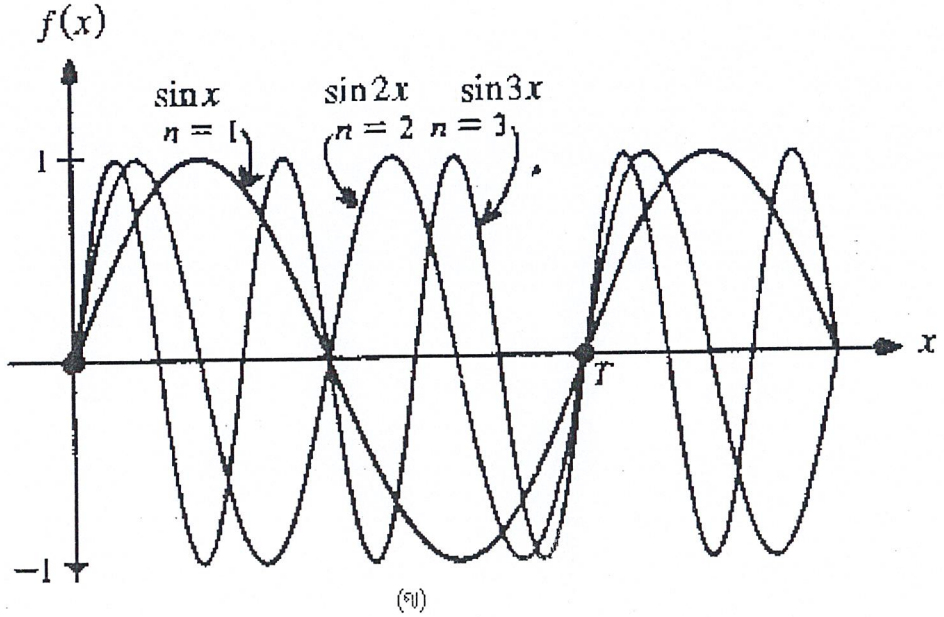
สมการที่(1) และรูปที่ 2.1.1 จะเห็นได้ว่าสัญญาณที่มีคาบครบรอบเท่า T จะเกิดขึ้นซ้ำกันในทุกๆค่าของ T ดังนั้นถ้า n เป็นตัวเลขจำนวนเต็มสามารถกำหนด :

$$f(x+nT) = f(x) \qquad \text{ตลอดทุกค่า x} \qquad (2)$$

ดังนั้นถ้า nT = 2T, 3T, 4T, ... เป็นคาบของฟังชั่น ซึ่งสัญญาณจะเกิดขึ้นซ้ำกันตามจำนวนรอบหรือค่าของ n



(ก)



(ข)

รูปที่ 2.1.2 (ก) สัญญาณ(ฟังก์ชั่น)ไซน์และโคไซน์ที่มีคาบครบรอบเท่ากับ 2π

(ข) สัญญาณไซน์ที่ฮาร์โมนิก $n = 1,2$ และ 3

2.1.2 อนุกรมฟูเรียร์ตรีโกณมิติ(Trigonometric Fourier Series)

จากในหัวข้อที่ 2.1.1 เราได้ทราบว่า $\sin nx$ และ $\cos nx$ มีค่าต่างๆตลอดช่วงทั้งหมด (x_0, x_0+T) ซึ่งเมื่อ $n = 0$ ค่า $\sin nx = 0$ แต่ $\cos nx = 1$ เพราะฉนั้นถ้า $f(x)$ เป็นฟังก์ชั่นใดๆจึงสามารถแทนให้อยู่ในเทอมของฟังก์ชั่นทั้งหมดที่เกิดขึ้นตลอดช่วง (x0,x0+T) ได้ดังนี้คือ

$$f(x) = a_0 + a_1\cos x + a_2\cos 2x + \ldots + a_n\cos nx + \ldots + b_1\sin x + b_2\sin 2x + \ldots + b_n\sin nx + \ldots \qquad (1)$$

เมื่อ $(x_0<x<x_0+T)$

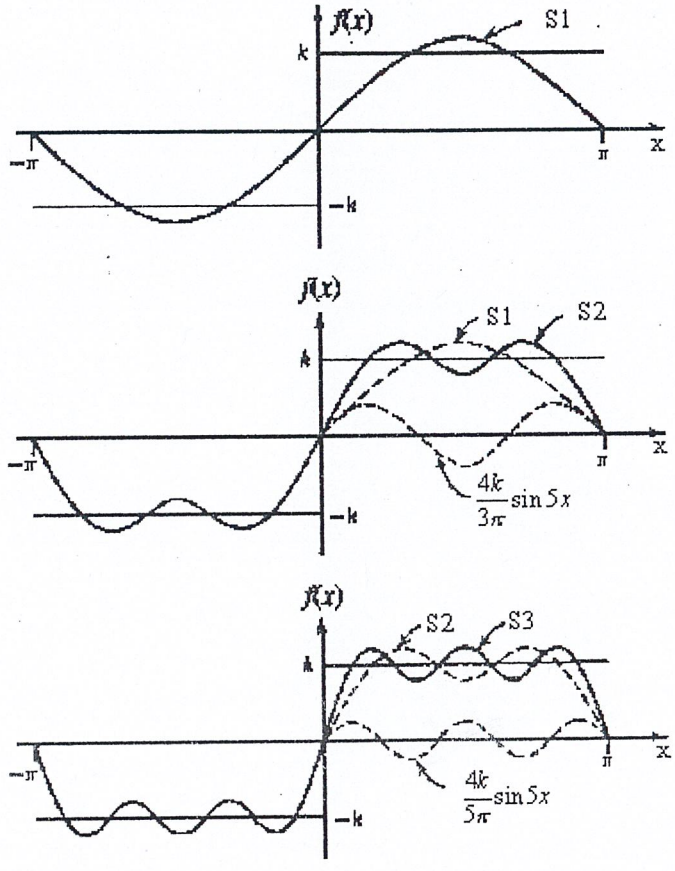
สมการที่ (3) สามารถเขียนให้อยู่ในรูปผลบวกได้ดังนี้คือ

$$f(x)=a_0+\sum_{n=1}^{\infty}(a_0+a_n\cos nx+b_n\sin nx) \tag{2}$$

$$a_0=\frac{1}{2\pi}\int_{-\pi}^{\pi}f(x)dx \tag{3}$$

$$a_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)\cos nxdx \tag{4}$$

$$b_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)\sin nxdx \tag{5}$$



รูปที่ 2.1.3 แสดงการแทนฟังก์ชั่นรูปสี่เหลี่ยมด้วยฟังก์ชั่นไซน์และโคไซน์

อนุกรมฟูเรียร์นำไปใช้ในการแก้ปัญหาในทางวิศวกรรม โดยทั่วไปทฤษฎีของอนุกรมฟูเรียร์จะค่อนข้างยุ่งยากซับซ้อน แต่จะง่ายต่อการนำมาใช้ ซึ่งอนุกรมฟูเรียร์นำไปใช้ได้ทั่วๆไปกว้างกว่าอนุกรมเทเลอร์ (Talor series) ทั้งนี้เพราะบางครั้งในทางปฏิบัติจะเกี่ยวข้องกับฟังก์ชั่นไม่เป็นคาบ(Nonperiodic function) และ/หรือฟังชั่นไม่ต่อเนื่อง(Discontinue function) ซึ่งอนุกรมฟูเรียร์สามารถนำไปใช้ได้ทั้งสองกรณี

2.1.3 อนุกรมฟูเรียร์เชิงซ้อน (Complex Form of Fourier Series)

ในการนำอนุกรมฟูเรียร์มาใช้งานส่วนใหญ่นั้นจะเป็นการง่ายและสะดวกเมื่อแทนอนุกรมฟูเรียร์ให้อยู่ในเทอมของเอกซ์โพเนนเชียลเชิงซ้อน(complex exponential) ดังนั้นจึงอาศัยการพิจารณาจากอนุกรมฟูเรียร์ที่มีคาบครบรอบ $f(t)$ ใดๆ จากสมการที่ (2) ในหัวข้อที่ 2.1.2 จะได้

$$f(t)=\frac{a_0}{2}+\sum_{n=1}^{\infty}[a_0\cos(n\omega_0 t)+b_0\sin(n\omega_0 t)] \tag{1}$$

เพื่อไม่ให้เข้าใจสับสนในการพิจรณาต่อไปนี้จึงใช้แทน ω ในสมการที่ (1) ด้วย $\omega_0=2\pi/T$ ซึ่งในสมการที่(1) เทอมของโคไซน์และไซน์สามารถเขียนในรูปของเอกซ์โพเนนเชียลได้จาก

$$\cos(n\omega_{0t})=\frac{1}{2}(e^{jn\omega_0 t}+e^{-jn\omega_0 t}) \tag{2}$$

และ

$$\sin nx(n\omega_0 t)=\frac{1}{2}(e^{jn\omega_0 t}-e^{-jn\omega_0 t}) \tag{3}$$

แทนสมการที่(2)และ(3) ในสมการที่ (4) จะได้

$$f(t)=\frac{a_0}{2}+\sum_{n=1}^{\infty}[a_n\frac{1}{2}(e^{jn\omega_0 t}+e^{-jn\omega_0 t})+b_n\frac{1}{2j}(e^{jn\omega_0 t}-e^{-jn\omega_0 t})] \tag{4}$$

$$f(t)=\frac{a_0}{2}+\sum_{n=1}^{\infty}[\frac{1}{2}(a_n-jb_n)e^{jn\omega_0 t}+\frac{1}{2}(a_n+jb_n)e^{-jn\omega_0 t}] \tag{5}$$

ให้

$$F_0=\frac{a_0}{2},\ F_n=\frac{1}{2}(a_n-jb_n),\ F_{-n}=\frac{1}{2}(a_n+jb_n) \tag{6}$$

แทนสมการที่(6)ในสมการที่(5)

$$f(t)=F_0+\sum_{n=1}^{\infty}(F_n e^{jn\omega_0 t}+F_{-n}e^{-jn\omega_0 t})$$

$$f(t)=F_0+\sum_{n=1}^{\infty}F_n e^{jn\omega_0 t}+\sum_{n=-1}^{-\infty}F_n e^{jn\omega_0 t})$$

$$f(t)=\sum_{n=-\infty}^{\infty}F_n e^{jn\omega_0 t} \tag{7}$$

สมการที่(7) เรียกว่า รูปแบบเชิงซ้อน(complex form) ของอนุกรมฟูเรียร์ $f(t)$ หรือเรียกว่าอนุกรมเชิงซ้อน (complex Fourier series) ของ $f(t)$ สัมประสิทธิ์ F_n สามารถคำนวณได้ง่ายๆในเทอมของ a_n และ b_n และจากสมการที่(3)ในหัวข้อที่ 2.1.2 คือ

$$F_0=\frac{a_0}{2}=\frac{1}{2}\bullet\frac{2}{T}\int_{-T/2}^{T/2}f(t)dt$$

$$F_0 = \frac{a_0}{2} = \frac{1}{T}\int_{-T/2}^{T/2} f(t)dt \qquad (8)$$

จากสมการที่(6)และสมการที่(4),(5) ในหัวข้อที่ 2.1.2 จะได้

$$F_n = \frac{1}{2}(a_n - jb_n)$$
$$= \frac{1}{2}[\frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos(n\omega_0 t)dt - j\frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin(n\omega_0 t)dt]$$
$$= \frac{1}{T}\{\int_{-T/2}^{T/2} f(t)[\cos(n\omega_0 t) - j\sin(n\omega_0 t)]dt\}$$
$$F_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-jn\omega_0 t}dt \qquad (9)$$

และ

$$F_{-n} = \frac{1}{2}(a_n + jb_n) = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-jn\omega_0 t}dt \qquad (10)$$

ถ้า เป็นฟังก์ชั่นจำนวนจริง ดังนั้น

$$F_{-n} = F_n^* \qquad (11)$$

ซึ่ง * แสดงถึงสังคยุคเชิงซ้อน (complex conjugate)

จากสมการที่(8),(9),(10) สามารถเขียนรวมอยู่ในสูตรเดียวได้คือ

$$F_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-jn\omega_0 t}dt \qquad ;n = 0,\pm1,\pm2,... \qquad (13)$$

เนื่องจาก $f(t)e^{-jn\omega_0 t}$ เป็นคาบครบรอบด้วยคาบ T จากการพิจารณาที่ผ่านมาสามารถกำหนดได้ว่า

$$F_n = \frac{1}{T}\int_0^T f(t)e^{-jn\omega_0 t}dt \qquad (13)$$

กำหนดให้

$$F_n = |F_n|e^{j\theta_n}, F_{-n} = F_n^* = |F_n|e^{-j\theta_n} \qquad (14)$$

ซึ่ง

$$|F_n| = \frac{1}{2}(a_n^2 + b_n^2)^{1/2} \qquad (15)$$

$$\theta_n = \tan^{-1}(-\frac{b_n}{a_n}) \qquad \text{สำหรับทุกค่าของ n ยกเว้นที่ n=0} \qquad (16)$$

ในกรณี F_0 เป็นค่าจริงนั้น

$$F_0 = \frac{a_0}{2} \qquad (17)$$

2.1.4 สเปคตรัมความถี่เชิงซ้อน (Complex Frequency Spectrum)

การกระจายอนุกรมฟูเรียร์ของฟังก์ชั่นคาบดังกล่าวข้างต้นนั้น ฟังชั่นจะอยู่ในเทอมของส่วนประกอบความถี่ต่างๆ ฟังก์ชั่นคาบที่มีคาบเท่ากับ T จะมีส่วนประกอบของความถี่เชิงมุม ω_0 , $2\omega_0$, $3\omega_0$,.:., $n\omega_0$ ฯลฯ ดังนั้นฟังก์ชั่นคาบ $f(t)$ ก็จะอยู่ภายในสเปคตรัมของความถี่เหล่านั้น ถ้าเรากำหนด $f(t)$ ใดๆขึ้นมาเราก็สามารถหาสเปคตรัมของมันได้ ในทางกลับกัน ถ้าเราทราบสเปคตรัมเราก็สามารถหาฟังก์ชั่นคาบนี้ได้ เพราะฉนั้นจึงสามารถกำหนดฟังก์ชั่นคาบได้ 2 วิธี คือ *ไทม์โดเมน*(time domain) และ *ฟรีเควนซี่โดเมน*(frequency domain)

ไทม์โดเมน (Time domain) คือ $f(t)$ แสดงอยู่ในรูปฟังก์ชั่นของเวลา

ฟรีเควนซี่โดเมน (Frequency domain) คือ $f(t)$ จะกำหนดได้จากส่วนประกอบของความถี่ต่างๆ ทั้งนี้สเปคตรัมจะเกิขึ้นที่ ที่ $\omega = \omega_0, 2\omega_0, 3\omega_0, \ldots, n\omega_0$ ดังนั้นสเปคตรัมจะเป็นเคิร์ปที่ไม่ต่อเนื่อง ซึ่งจะเกิดขึ้นตามค่าดิสครีต(discrete value) ของ ω ดังนั้นจึงเรียกว่า สเปคตรัมแบบดิสครีต(discrete spectrum) หรือ สเปคตรัมแบบเส้น(line spectrum) เราอาจจะแสดงสเปคตรัมได้ด้วยเส้นกราฟโดยกำหนดให้แนวนอนเป็นค่า ω และแนวตั้งเป็นแอมปริจูด(amplitude) หรือระดับของสัญญาณที่เป็นสัดส่วนของส่วนประกอบต่างๆ

เราสามารถใช้อนุกรมเอกซ์โพเนลเชียลหรืออนุกรมตรีโกณแสดงสเปคตรัมได้การใช้รูปแบบเอกซ์โพเนนเชียงจะแสดงความหมายได้ดีกว่าและใช้ได้ทั่วๆไป ดังพิจารณาต่อไปนี้ ซึ่งสำหรับฟังก์ชั่นคาบที่มีคาบเท่ากับ T อนุกรมเอกซ์โพเนนเชียลจากสมการที่(7)ในหัวข้อที่ 2.1.3 คือ

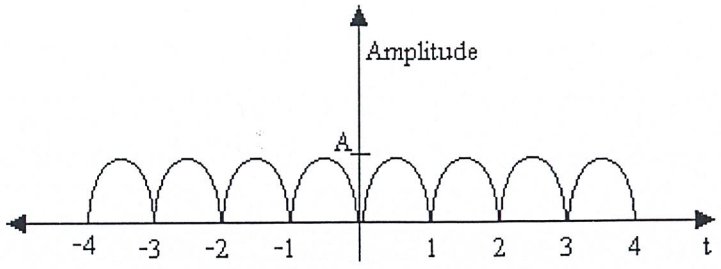
$$f(t) = \sum_{n=-\infty}^{\infty} F_n e^{jn\omega_0 t} \qquad (1)$$

$$f(t) = F_0 + F_1 e^{j\omega_0 t} + F_2 e^{j\omega_0 t} + \ldots + F_n e^{j\omega_0 t} + \ldots + F_{-1} e^{-j\omega_0 t} + F_{-2} e^{-j\omega_0 t} + \ldots + F_{-n} e^{-j\omega_0 t} + \ldots$$

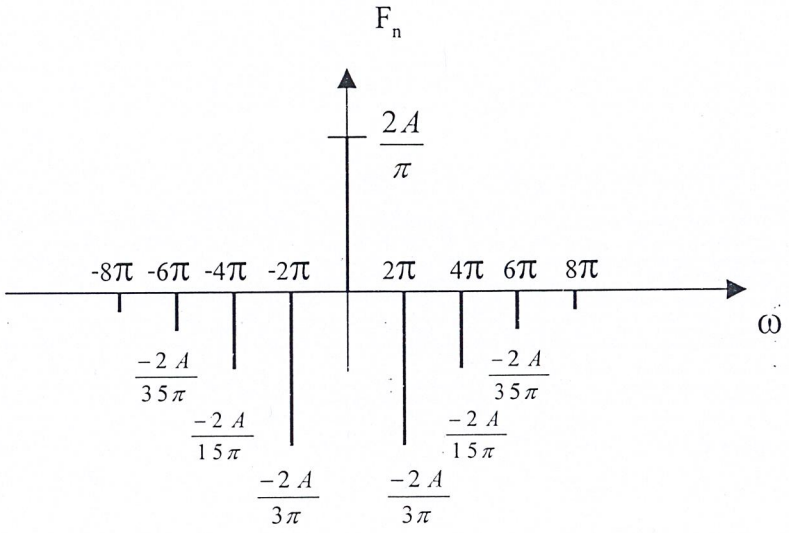
ซึ่งสมการที่(1) จะประกอบด้วยความถี่ต่างๆคือ 0, ω_0 , - ω_0 , 2 ω_0 , -2 ω_0 ,..., $n\omega_0$,- $n\omega_0$
และแอมปริจูดคือ F_0, F_1, F_{-1}, F_2, F_{-2},...,F_n, F_{-n},...

โดยทั่วไปแล้ว F_n เป็นปริมาณเชิงซ้อนจึงสามารถนำไปอธิบายสเปคตรัมเชิงแอมปริจูดของฟังก์ชั่นคาบ $f(t)$ จะได้จากการกำหนดขนาดของ F_n ในอนุกรมสมการที่(1) ตามแกน y กับความถี่เชิงมุม (angular frequency) ตามแกน x และสเปคตรัมเชิงเฟส(phase spectrum) ของ $f(t)$ จะได้จากการกำหนดมุมเฟส(phase angle) θ_n ของ F_n ตามแกน y กับค่าความถี่เชิงมุมตามแกน x

โดยทั่วไปแล้วสเปคตรัมจะเกิดขึ้นที่ $\omega = 0, \pm 2\pi, \pm 4\pi, , \pm 6\pi, \ldots$ และขนาด(magnitude) จะเป็นไปตามค่า $2A/\pi$, $-2A/3\pi$, $-2A/15\pi$, $-2A/35\pi$,... จาก $f(t)$ ซึ่งแอมปริจูดทั้งหมดเป็นค่าจริงจึงสามารถนำไปกำหนดได้เพียงสเปคตรัมเดียวเท่านั้น ซึ่งสเปคตรัมความถี่(frequency spectrum) ได้แสดงไว้ในรูปที่ 2.1.4



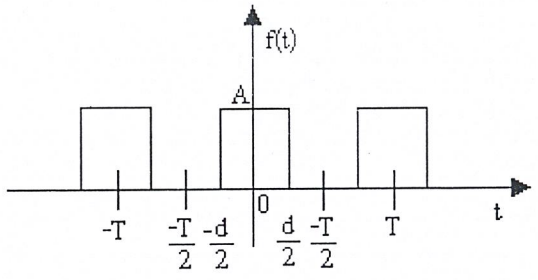
(ก)



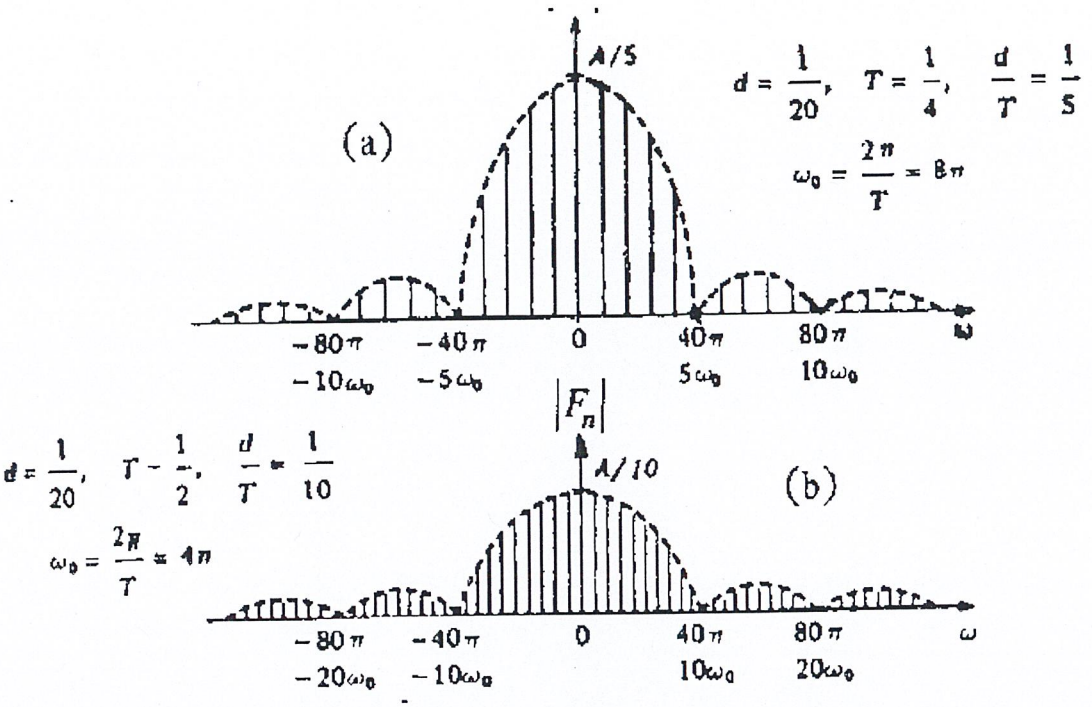
(ข)

รูปที่ 2.1.4 (ก) สัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น

(ข)ไลน์สเปคตรัมของสัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น



(ก)

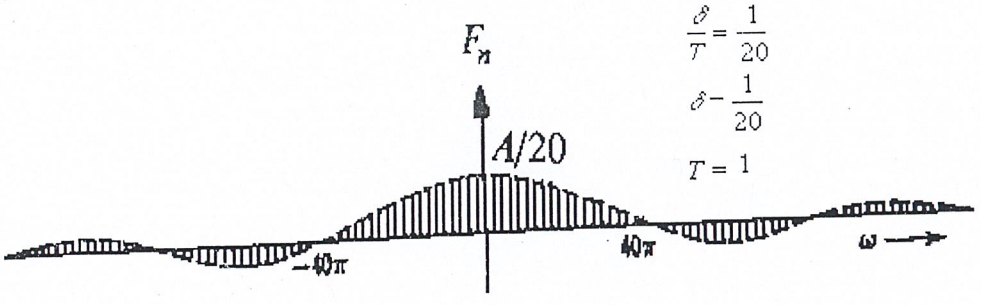


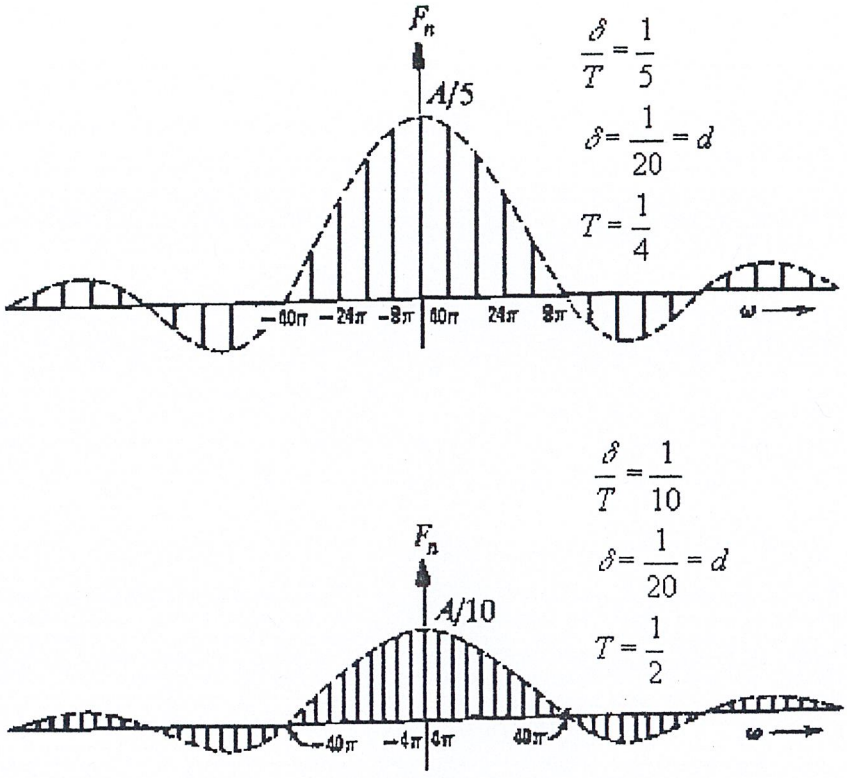
(ข)

รูปที่ 2.1.5 (ก) สัญญาณพัลส์รูปสี่เหลี่ยมพื้นผ้า

(ข) สเปคตรัมเชิงแอมปริจูด

จากรูปที่ 2.1.5(ข) จะเห็นได้ว่าเมื่อเวลาามากขึ้นสเปคตรัมจะหนาแน่นมากขึ้น(แน่นขึ้น) แต่ขนาดหรือแอมปริจูดจะลดลง แต่รูปร่างยังคงเหมือนเดิม เปรียบเทียบให้ชัดเจนยิ่งขึ้นดังรูปที่ 2.1.6





(ข)

รูปที่ 2.1.6 สเปคตรัมเชิงแอมปริจูด

2.1.5 สเปคตรัมเชิงกำลัง (Power Spectrum) ของฟังก์ชั่นคาบ

สมมุติว่าฟังก์ชั่นคาบ $f(t)$ คือแรงดันหรือกระแส ดังนั้นอินทิกรัลคือ

$$\frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt \qquad (1)$$

ซึ่งอินทิกรัล (1) แสดงถึงกำลังที่ได้จาก $f(t)$ ที่ป้อนให้กับความต้านทาน 1 โอห์ม และอินทิกรัลนี้เรียกว่ากำลังของสัญญาณคาบ $f(t)$ สเปคตรัมเชิงความถี่ของ $f(t)$ จะแสดงถึงแอมปริจูดของส่วนประกอบความถี่ต่างๆของ $f(t)$ ซึ่งจะมีกำลังรวมอยู่ในแต่ละส่วนประกอบ ดังนั้นดังที่กล่าวไปแล้วในส่วนของสเปคตรัมเชิงแอมปริจูด ซึ่งสามารถแทนอยู่ในรูปของความถี่ต่างๆเราจึงสามารถแทนสเปคตรัมเชิงกำลังที่เป็นไปตามกำลังที่รวมอยู่กับส่วนประกอบของความถี่ต่างๆใน $f(t)$ ได้

พิจารณาอนุกรมฟูเรียร์ในรูปเอกซ์โพเนนเชียลจากสมการที่(7)ในหัวข้อที่ 2.1.3 คือ

$$f(t)=\sum_{n=-\infty}^{\infty} F_n e^{jn\omega_0 t} \qquad (2)$$

แทนสมการ(2)ลงในสมการ(1) ดังนั้น

$$\frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\sum_{n=\infty-}^{\infty} F_n e^{jn\omega_0 t}\,dt \qquad (3)$$

เครื่องหมายอินทิเกรตและผลบวกทางด้านขวามือของสมการที่(3)สามารถสลับที่กันได้ ดังนั้น

$$\frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt = \frac{1}{T}\int_{-T/2}^{T/2} F_n \sum_{n=\infty-}^{\infty} f(t) e^{jn\omega_0 t}\,dt \qquad (4)$$

ซึ่งจะเห็นว่าอินทิกรัลทางด้านขวามือในสมการที่(14)มีค่าเท่ากับ TF_{-n} ดังนั้น

$$\frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt = \sum_{n=-\infty}^{\infty} F_n F_{-n} \qquad (5)$$

จากสมการที่(11)ในหัวข้อ 2.1.3 จะได้

$$\frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt = \sum_{n=-\infty}^{\infty} |F_n|^2 \qquad (6)$$

$$= F_0^2 + |F_1|^2 + |F_2|^2 + |F_n|^2 + \ldots + |F_{-1}|^2 + |F_{-2}|^2 + |F_{-n}|^2 + \ldots \qquad (7)$$

สมการที่(16)จะเป็นเงื่อนไขตามทฤษฎีของ ปาร์เซอวาล (Parseval) ซึ่งนำไปประยุกต์ใช้กับอนุกรมฟูเรียร์ในรูปเอกซ์โพเนนเชียล และสมการที่(17)แสดงให้เห็นว่ามีกำลังรวมอยู่ในแต่ละส่วนประกอบของความถี่และขนาดของกำลังที่บรรจุอยู่ ส่วนประกอบของความถี่ $n\omega_0$ คือ $|F_n|^2$ และในส่วนของ $-n\omega_0$ คือ $|F_{-n}|^2$ แต่เนื่องจาก

$$F_{-n} = F_n^*$$

จะได้

$$|F_{-n}|^2 = |F_n|^2 \qquad (8)$$

ดังนั้นกำลังที่รวมอยู่ในส่วนประกอบของความถี่ $n\omega_0$ จะเป็นเช่นเดียวกับที่ความถี่ $-n\omega_0$ เนื่องจากเราทราบว่าสัญญาณ $e^{jn\omega_0 t}$ และ $e^{-jn\omega_0 t}$ ทั้งสองแสดงถึงสัญญาณของความถี่เดียวกันคือ $n\omega_0$ เพราะฉะนั้นกำลังทั้งหมดที่รวมอยู่กับความถี่ $n\omega_0$ ในสัญญาณจะมีค่าเท่ากับ $|F_n|^2$ ซึ่งจะแสดงให้เห็นดังข้างล่างนี้ และส่วนประกอบทั้งสองของ $F_n e^{jn\omega_0 t}$ จะให้ฟังก์ชั่นจำนวนจริง(real function)ของเวลา $f(t)$ กล่าวคือ

$$f_n(t) = F_n e^{jn\omega_0 t} + F_{-n} e^{-jn\omega_0 t}$$

$$F_n = |F_n| e^{j\theta_n}$$

$$F_{-n} = |F_n| e^{-j\theta_n}$$

เพราะฉะนั้น

$$f_n(t) = |F_n| e^{j(n\omega_0 t + \theta_n)} + |F_n| e^{-j(n\omega_0 t + \theta_n)}$$

$$= 2|F_n|\cos(n\omega_0 t + \theta_n)$$

$$= 2\,|\,F_n\,|\,\cos(n\omega_0 t + \theta_n) \qquad (9)$$

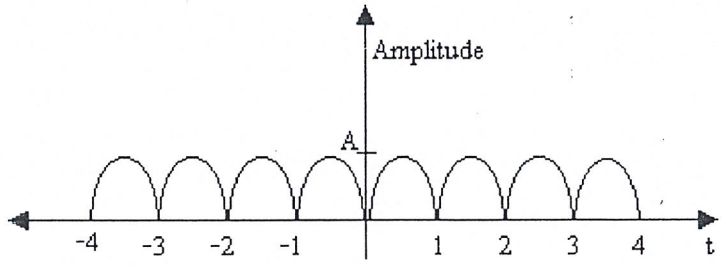
กำลังที่บรรจุในสัญญาณใดๆที่มีแอมปริจูด A คือ $A^2/2$ ดังนั้นกำลังที่บรรจุอยู่ใน $2\,|\,F_n\,|\,\cos(n\omega_0 t + \theta_n)$ คือ $2\,|\,F_n\,|^2$ เราสามารถรวมกำลังนี้ครึ่งหนึ่งกับส่วนประกอบ** เช่นเดียวกันนี้ กำลังที่รวมกับ *** ก็คือ $|\,F_{-n}\,|^2$ เพราะฉะนั้นสมการที่(7)อาจจะแสดงความหมายได้ว่า กำลังในสัญญาณคาบ(periodic degnal) จะมีค่าเท่ากับผลรวมของกำลังในแต่ละส่วนประกอบ

เราสามารถกำหนดสเปคตรัมกำลังที่รวมอยู่กับแต่ละส่วนประกอบของความถี่ได้ ซึ่งเห็นได้อย่างชัดเจนว่าสเปคตรัมจะเกิดขึ้นเป็นช่วงๆไม่ต่อเนื่อง(discrete) หรือไม่ติดต่อกัน ทั้งนี้จะเกิดตามค่าความถี่ต่างๆคือที่ $\omega = 0, \pm\omega_0, \pm 2\omega_0, \ldots$ฯลฯและจะสมมาตรกันทั้งซ้ายและขาวกับแนวตั้ง

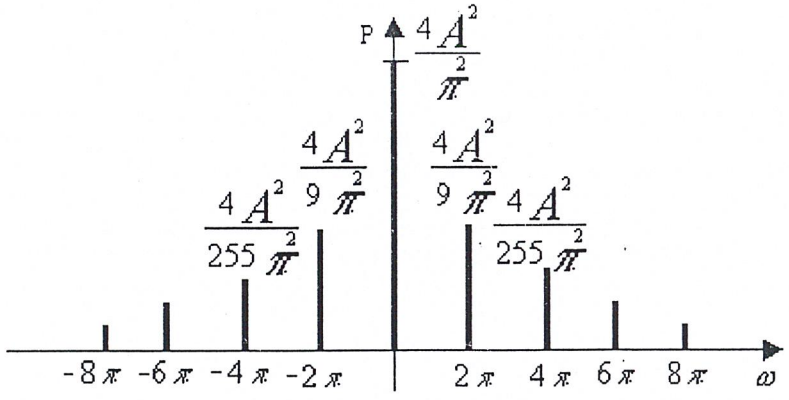
ดังนั้นถ้าเราให้ P เป็นกำลังของสัญญาณคาบ $f(t)$ จะได้

$$P = \frac{1}{T}\int_{-T/2}^{T/2} f^2(t)\,dt$$

$$P = \sum_{n=-\infty}^{\infty} |F_n|^2$$



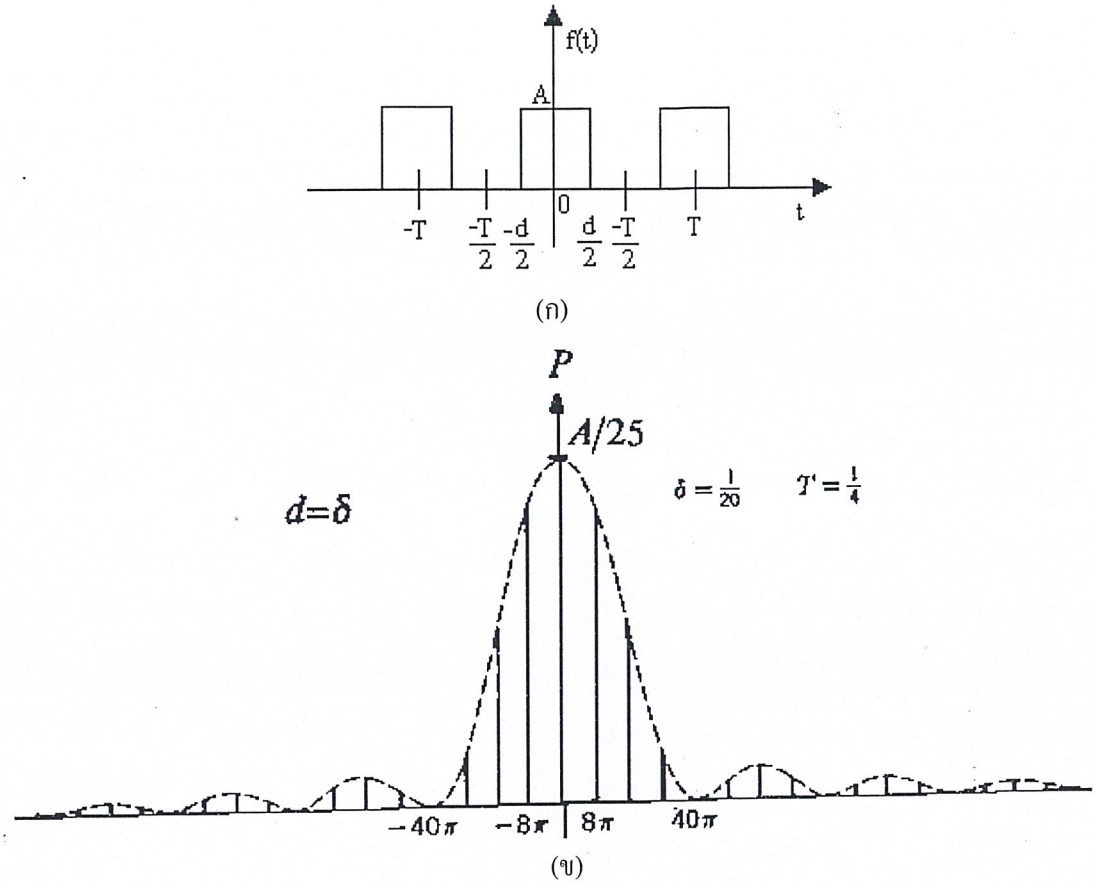
(ก)



(ข)

รูปที่ 2.1.7 (ก) สัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น

(ข) สเปคตรัมเชิงกำลังของสัญญาณเร็คติฟาร์ยไซน์เต็มคลื่น

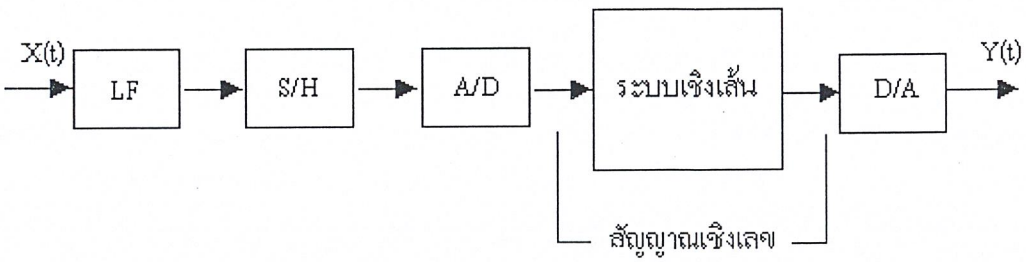


รูปที่ 2.1.8 (ก) สัญญาณพัลส์รูปสี่เหลี่ยมพื้นผ้า

(ข) สเปคตรัมเชิงกำลังของสัญญาณพัลส์รูปสี่เหลี่ยมพื้นผ้า

2.2 ทฤษฎีการสุ่มตัวอย่าง (Sampling)

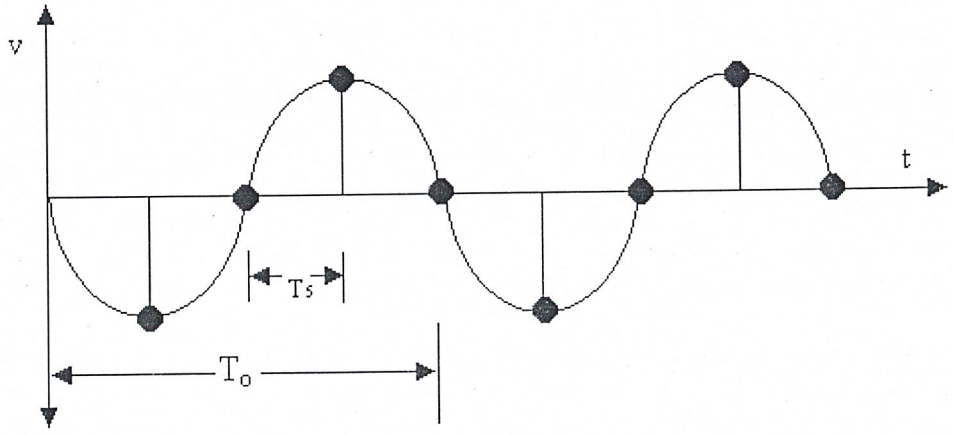
อาจกล่าวได้ว่าสัญญาณในธรรมชาติส่วนมาก เช่น สัญญาณเสียง สัญญาณการสั่นสะเทือนของพื้นโลก หรือการแปลค่าของอุณหภูมิ เหล่านี้เป็นไปในลักษณะต่อเนื่องกับพิสัยเวลา หรืออาจกล่าวได้ว่าเป็นสัญญาณเชิงอนุมาน การนำสัญญาณเหล่านี้ไปทำการประมวลผลในเชิงเลข หรือ การประมวลผมเชิงหน่วย (Discrete Signal Processing) ได้จำเป็นต้องใช้ระบบการประมวลผลตามรูปที่ 2.2.1



รูปที่ 2.2.1 การประมวลผลเชิงอนุมานโดยใช้ระบบการประมวลผลสัญญาณเชิงเลข

ตามรูป วงจร S/H เป็นวงจรสุ่มและคงค่าสัญญาณไว้เพื่อให้วงจรแปลงอนาล็อคเป็นดิจิตอลทำการแปลงเป็นตัวเลขอีกทีหนึ่ง สังเกตุว่าระบบการประมวลผลเชิงตัวเลขเป็นส่วนของระบบที่อยู่ระหว่างจุด A และ B โดยที่สัญญาณที่จุดนี้เป็นสัญญาณที่ถูกแปลงเป็นตัวเลขแล้ว อย่างไรก็ตามการแปลงสัญญาณเชิงอนุมานเป็นสัญญาณเชิงเลขนั้น ในเบื้องแรกต้องมีการสุ่มตัวอย่างก่อน ซึ่งความถี่ในการสุ่มตัวอย่างโดยที่ไม่ทำให้สัญญาณสูญเสียข้อมูลเดิมไป โดยทั่วไปแล้วเราอาจสุ่มตัวอย่างโดยใช้ค่าความถี่ $f_{SN} = 2f_o$ พอดี ค่าความถี่นี้มีชื่อเรียกว่า *ความถี่ไนซ์ควิสต์* (Nyquist Frequency) และคาบเวลา $Tn=1/(2f_o)$ เรียกว่า *ช่วงเวลาสุ่มตัวอย่างไนซ์ควิสต์* (Interval) ในทางปฏิบัติเพื่อหลีกเลี่ยงผลของปรากฏการณ์ที่ไม่เป็นเชิงเส้น (Non Linearlity) ที่อาจเกิดจากการสุ่มตัวอย่าง เรามักใช้ความถี่ในการสุ่มตัวอย่าง 2fS มากกว่าค่าความถี่ไนท์ควิสต์หรือ fSN ขึ้นไป ส่วนจะมีค่ามากกว่าเป็นเท่าใดนั้นขึ้นอยู่กับลักษณะของงานไม่ได้มีการกำหนดค่าที่แน่นอน

ตามทฤษฎีการสุ่มตัวอย่างของแชนนอนนั้นจะเห็นได้ว่าการสุ่มตัวอย่างได้ถูกต้องนั้นก็ต่อเมื่อรู้ค่าแถบความถี่ที่ปฏิบัติงานของสัญญาณหรือพูดอีกนัยหนึ่ง สัญญาณต้องรู้แถบความถี่ที่ปฏิบัติงานจึงจะสามารถทำการสุ่มตัวอย่างได้ ดังนั้นเพื่อให้มั่นใจว่าสัญญาณที่ทำการประมวลผล ถูกประมวลผลอย่างถูกต้อง ในภาคแรกของการประมลผลเชิงเต็มหน่วยและการประมวลผลเชิงเลข ดังแสดงในรูป2.2.2



รูปที่ 2.2.2 แสดงตัวอย่างของความถี่ในการสุ่มสัญญาณ $Ts = 1/(4f_o)$

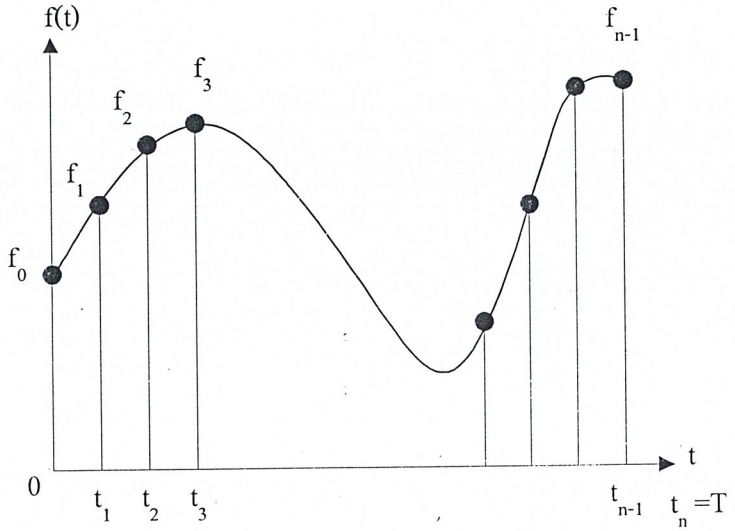
สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

2.3 ทฤษฎีฟาสท์ฟูเรียร์ทรานฟอร์ม *(Fast Fourier Transform)*

ทฤษฎีฟาสท์ฟูเรียร์ทรานฟอร์มเป็นการคำนวณฟูเรียร์เพื่อสามารถคำนวณได้เร็วขึ้นโดยใช้คอมพิวเตอร์เป็นตัวควบคุมการคำนวณทั้งหมด ในการที่จะทำความเข้าใจเกี่ยวกับฟาสท์ฟูเรียร์ทรานฟอร์มจะต้องศึกษาทฤษฎีดีสครีทฟูเรียร์เสียก่อน

2.3.1 ทฤษฎีดีสครีทฟูเรียร์ทรานฟอร์ม *(The Discrete Fourier Transform)*

ดีสครีทฟูเรียร์ทรานฟอร์มเป็นการแปลงฟูเรียร์แบบไม่ต่อเนื่อง โดยข้อมูลที่เราสุ่มเข้ามาจะต้องเริ่มจากค่า "0" จนถึงค่า "T" ดังรูปที่ 1 โดยให้ค่า N เป็นจำนวนการสุ่มข้อมูลทั้งหมด โดยมีค่า t_n เป็นเวลาในการสุ่มข้อมูล และค่า f_n เป็นค่าของฟังก์ชั่นต่อเนื่องตามการสุ่มของ t_n ใดๆ



รูปที่ 2.3.1 ดีสครีทฟูเรียร์ทรานฟอร์ม

โดยข้อมูลที่อยู่ที่ตำแหน่ง n = 0,1,2,3,……,N-1 สามารถเขียนสมการดีสครีทฟูเรียร์ทรานฟอร์มได้ดังนี้

$$F_k = \sum_{n=0}^{N-1} f_n e^{-jk\omega_{on}} \qquad \text{โดยค่า } k = 0 \text{ ถึง } N-1 \qquad (1)$$

ดังนั้นจะได้ค่าอินเวิสฟูเรียร์ทรานฟอร์มดังนี้

$$f_n = 1/N \sum_{k=0}^{N-1} F_k e^{jk\omega_{on}} \qquad \text{โดยค่า } n = 0 \text{ ถึง } N-1 \qquad (2)$$

34105

โดยค่า $\omega o = 2\pi/N$ และการแปลงค่าดีสครีทฟูเรียร์ทรานฟอร์มจากสมการ (1) ต้องทำการคำนวณเป็นจำนวน N^2 โดยข้อมูลต้องอยู่ในรูปของจำนวนเชิงซ้อนเพื่อให้คอมพิวเตอร์คำนวณได้ง่าย จากสมการ(1) สามารถใช้ทฤษฎีของอูเลอร์(Euler's) มาคำนวณได้จากสูตรดังนี้

$$e^{\pm ia} = \cos a \pm i \sin a$$

แทนสูตรของอูเลอร์ลงในสมการ (1) และ (2) จะได้

$$F_k = \sum_{n=0}^{N-1} [\, f_n \cos(k\omega_o n) - j\, f_n \sin(k\omega_o n) \,] \qquad (3)$$

$$f_n = 1/N \sum_{n=0}^{N-1} [\, F_k \cos(k\omega_o n) + j\, F_k \sin(k\omega_o n) \,] \qquad (4)$$

จากสมการ (3) และ (4) สามารถนำไปเขียนโปรแกรมเพื่อใช้ในการคำนวณหาค่าฟูเรียร์ได้ต่อไป

2.3.2 ทฤษฎีฟาสต์ฟูเรียร์ทรานฟอร์ม *(FFT)* และวิธีการแบบบัตเตอร์ฟลาย (Butterfly)

ฟาสต์ฟูเรียร์ทรานฟอร์ม(FFT)เป็นการพัฒนามาจากดีสครีทฟูเรียร์ทรานฟอร์ม(DFT)เพื่อให้มีการคำนวณเร็วขึ้น ซึ่งดิสครีทฟูเรียร์ทรานฟอร์มต้องการข้อมูลที่ใช้ในการคำนวณเป็นจำนวน N^2 แต่ฟาสต์ฟูเรียร์ทรานฟอร์มจะใช้ข้อมูลในการคำนวณประมาณ $N \log_2 N$

ฟาสต์ฟูเรียร์ทรานฟอร์มจะคำนวณเร็วกว่าดิสครีทฟูเรียร์ทรานฟอร์มประมาณ 10 เท่า โดยจะใช้วิธีการแบบบัตเตอร์ฟลายหรือวิธีการของ คูลี่ย์-ทูคีย์ (Cooley-Tukey algorithm)

โดยสมมุติให้ $N = 2^M$ โดย M เป็นค่าจริงใดๆ จะได้

$F_k = X(k)$, $f_n = x(n)$ จะได้สมการใหม่ดังนี้

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j(2\pi/N)\, nk} \qquad \text{โดย } k = 0,1,2,3,\ldots..,\ N-1 \qquad (5)$$

ถ้าให้ $e^{-j(2\pi/N)\, nk} = W^{nk}$ ได้สมการใหม่ดังนี้

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W^{nk} \qquad (6)$$

แบ่งสมการ (6) ได้เป็น 2 ส่วนดังนี้

$$X(k) = \sum_{n=0}^{N/2-1} x(2n)\, W_N^{nk} + \sum_{n=0}^{N/2-1} x(2n+1)\, W_N^{(2n+1)\, k} \qquad (7)$$

N แทนขนาดความยาวของการสุ่มตัวอย่างและแทนสมการทั้งสองด้วยสมการคู่ และสมการคี่ได้สมการใหม่คือ

$$X(k) = \sum_{n=0}^{N/2-1} x_{ev}(n)\, W_{N/2}^{nk} + \sum_{n=0}^{N/2-1} x_{od}(n)\, W_{N/2}^{nk} \qquad (8)$$

หรือ $$X(k) = X_{ev}(k) + W_{N/2}^{k}\, X_{od}(k) \qquad (9)$$

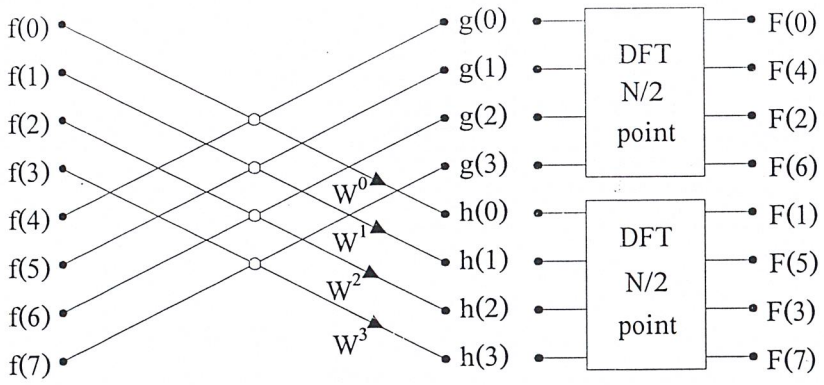
ในการนี้จะใช้ N/2 จุดเท่านั้นเพื่อคำนวณหาค่า X(k) โดย k ต้องมีค่าตั้งแต่ 0 ถึง N-1 อย่างไรก็ตาม คุณสมบัติของสมการคู่และสมการคี่หาได้จาก

$$X_{od}(k) = X_{ev}(k- N/2) \rightarrow \text{โดย } N/2 \leq k \leq (N-1) \qquad (10)$$

ผลลัพธ์ของการแปลงฟูเรียร์แบบไม่ต่อเนื่องสามารถทำซ้ำๆ จนกระทั่งเหลือข้อมูลแค่ตัวเดียวโดยต้องมีข้อมูลอินพุท 2 ตัวเพื่อใช้ในการคำนวณตลอดเวลา

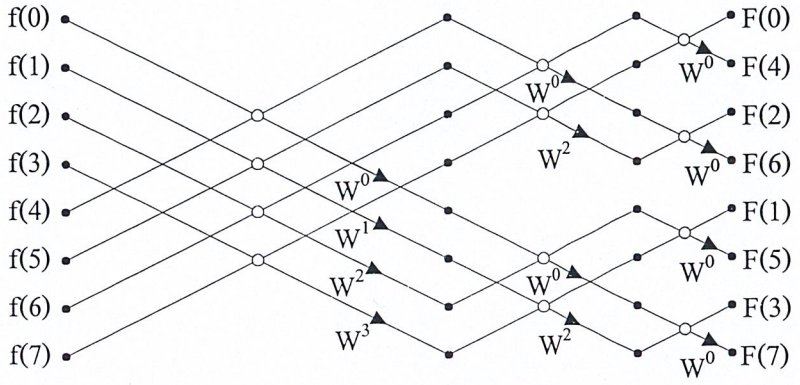
$$\Lambda(k) = \lambda(0) + \lambda(1)\, e^{-j2\pi k/2}$$
$$= \lambda(0) + \lambda(1) \quad \text{for all k , for k even, for k odd. ตามลำดับ}$$
$$= \lambda(0) - \lambda(1)$$

สำหรับการหาค่าฟาสต์ฟูเรียร์ทรานฟอร์มจะใช้แค่ 2 จุด ในการคำนวณหาค่าของฟาสต์ฟูเรียร์ทรานฟอร์มที่สมบูรณ์จะต้องมีการคูณโดยใช้แฟคเตอร์ที่เหมาะสมคือค่า "W" โดยค่า W จะต้องเริ่มจาก W^0 ถึง $W^{N/2-1}$ ในรูปที่ 2 แสดงกราฟของฟาสต์ฟูเรียร์ทรานฟอร์ม 8 จุด โดยต้องคำนวณหาค่าต่างๆตามวิธีการของบัตเตอร์ฟลายให้เหมาะสม



รูปที่ 2.3.2 (ก) การหาค่า 8 จุดฟาสต์ฟูเรียร์ทรานฟอร์ม

สำหรับการเริ่มต้นคำนวณต้องทราบค่าของ N ที่จะใช้ในการคำนวณ จากรูปที่ 2.3.2 จะใช้ค่า N= 8 และจะได้ค่า gn และ hn ที่คำนวณได้เป็นจำนวน N/2 จุด โดยมีค่า Wn มาคูณกับค่า hn ดังรูปที่ 1 และจะคำนวณในลำดับต่อไปดังรูปที่ 2.3.2 (ก)



รูปที่ 2.3.2 (ข) การหาค่า 8 จุดฟาสต์ฟูเรียร์ทรานฟอร์ม

จากรูปที่ 2.3.2 (ข) เมื่อคำนวณช่วงแรกเสร็จจะคำนวณช่วงที่ 2 ต่อโดยมีวิธีการคำนวณดังรูป และจะใช้ข้อมูลในการคำนวณทั้งสิ้นประมาณ N log2 (N) สามารถหาจำนวนของบัตเตอร์ฟลายได้ดังนี้

จำนวนของบัตเตอร์ฟลาย = N/2 log2 (N)

โดย N/2 = จำนวนแถวของบัตเตอร์ฟลาย(มี 2 อินพุท)

log2 (N) = จำนวนหลักของบัตเตอร์ฟลาย

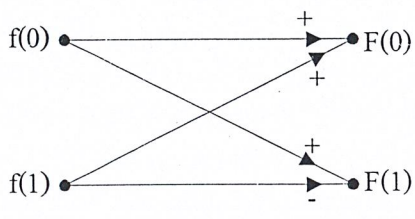
แต่ละบัตเตอร์ฟลายต้องมี 2 อินพุทเท่านั้น

2.3.3 การคำนวณโดยใช้คอมพิวเตอร์

ในการคำนวณโดยใช้คอมพิวเตอร์จะต้องทำการแปลงฟังก์ชั่นเอ็กโปแนนท์เชียลให้เป็นฟังก์ชั่นตรีโกณมิติจากสูตร

$$e^{\pm ia} = \cos a \pm i \sin a$$

เนื่องจากโปรแกรมที่ใช้เขียนไม่รู้จักฟังก์ชั่นเอ็กโปแนนท์เชียล ในการคำนวณจะยกตัวอย่างมาแค่บัตเตอร์ฟลายเดียว และจะต้องมีการเก็บค่าไว้ชั่วคราว ดังรูปที่ 2.3.2 (ค)



temporary = real(0) + real(1)
real(1) = real(0) - real(1)
real(0) = temporary
temporary = imaginary(0) - imaginary(1)
imaginary(0) = temporary

รูปที่ 2.3.2 (ค) หลักการคำนวณโดยใช้บัตเตอร์ฟลายเดียว

จากรูปที่ 2.3.2 (ค) จะได้ค่า $F(0) = f(0) + f(1)$

$F(1) = f(0) - f(1)$

โดยข้อมูลที่ได้จะมีทั้งค่าจริงและค่าจินตภาพ

ตัวอย่างการคำนวณหาฟาสต์ฟูเรียร์ทรานฟอร์ม

ตัวอย่างที่ 1 เป็นการหาค่า 16 จุด ฟาสต์ฟูเรียร์ทรานฟอร์ม โดยมีการสุ่มข้อมูล 16 จุด และมีค่าสัญญาณดังนี้

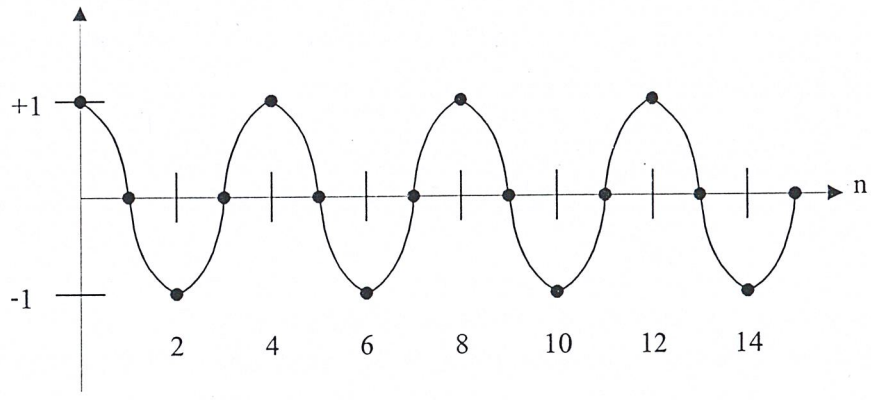
$$x(n) = \cos\left[2\pi(4n/16)\right]$$

จะเปรียบเทียบจำนวนบัตเตอร์ฟลายแบบดีสครีทฟูเรียร์ทรานฟอร์มและฟาสต์ฟูเรียร์ทรานฟอร์ม ที่ใช้ในการคำนวณ

Length of Transform (N)	DFT Operations (N^2)	FFT Operations $N \log_2 (N)$
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16384	896

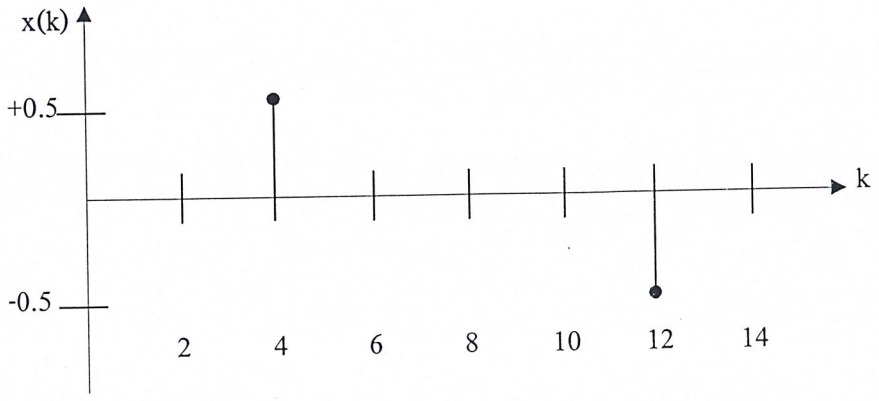
ตารางที่ 2.3.1 แสดงการเปรียบเทียบจำนวนการคำนวณระหว่างดีสครีทฟูเรียร์ทรานฟอร์มและฟาสต์ฟูเรียร์ทรานฟอร์ม

จากตัวอย่างมีค่าแอมปลิจูด "1" และสัญญาณเป็นค่าจริง ส่วนค่าจินตภาพเป็น "0" การสุ่มข้อมูลดูได้จากรูปที่ 2.3.3 (ก) ซึ่งมีจำนวนข้อมูล 16 จุด ตั้งแต่ x0 – x15



รูปที่ 2.3.3 (ก) อินพุท 16 จุด ฟาสต์ฟูเรียร์ทรานฟอร์ม

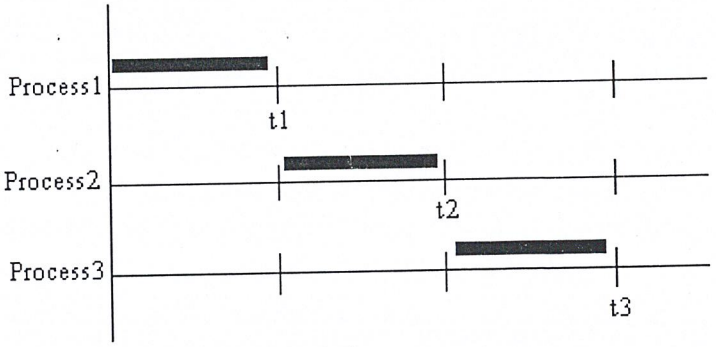
โดยค่าเอาท์พุทที่ได้จะโชว์ในรูปที่ 2.3.3 (ข) ซึ่งมีค่าอยู่ที่ k=4 โดยมีขนาดแอมปลิจูด 0.5 และที่ k = 12 มีขนาดแอมปลิจูด –0.5



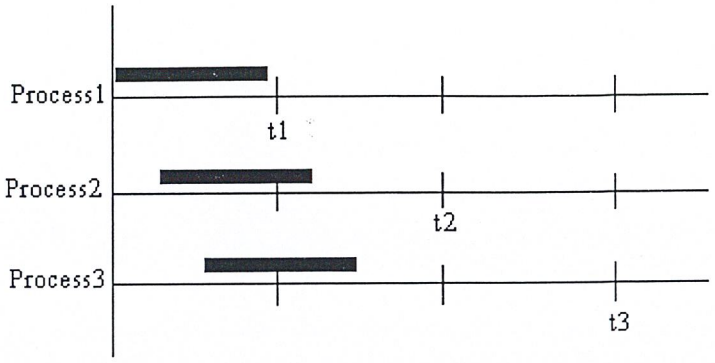
รูปที่ 2.3.3(ข) เอาท์พุทของ 16 จุด ฟาสท์ฟูเรียร์ทรานฟอร์ม

2.4 โปรแกรมแบบเรียลไทม์(Real Time Programming)

โปรแกรมแบบเรียลไทม์ คือ โปรแกรมที่ป้อนอินพุทให้กับโปรแกรมแล้วสามารถได้เอาท์พุทต่างๆที่ต้องการพร้อมกันโดยมีเวลาหน่วงเพียงน้อยนิด ตัวอย่างของการทำงานของโปรแกรมแบบเรียลไทม์ในชีวิตประจำวันก็คือการฟังการบรรยาย อินพุท คือ ผู้บรรยาย เอาท์พุท คือ ผู้ฟัง ต้องมีความความเข้าใจในสิ่งที่ผู้บรรยายได้พูด , มีการบันทึกลงสมุดบันทึก และ ข้อสรุปและวิจารณ์ ทันทีที่ผู้บรรยายได้บรรยายจบ นั่นก็คือ ผู้ฟังต้องทำงานหลายอย่างในขณะที่นั่งฟังบรรยาย คือ ฟัง , ทำความเข้าใจ , เขียน และ สรุป โดยการอาศัยช่วงเวลาที่ผู้บรรยายเว้นช่วงเวลาในการพูด เราจะใช้ช่วงเวลานี้ในการทำสิ่งอื่นๆที่ไม่ใช่การฟัง ซึ่งเป็นแนวคิดเดียวกันกับการออกแบบโปรแกรมที่ทำงานแบบเรียลไทม์ นั่นก็คือ พยายามใช้งานตัวโปรแซสเซอร์ให้มีเวลาว่างน้อยที่สุด โดยอาศัยช่วงเวลาที่โปรแซส(process)ปัจจุบันอยู่ในสภาวะรอ I/O ส่งงานอื่นให้ตัวโปรแซสเซอร์ทำแทนที่จะปล่อยให้โปรแซสเซอร์ว่างงาน จึงเปรียบสเหมือนว่าโปรแซสเซอร์สามารถทำงานได้หลายงานในขณะเดียวกัน โดยปกติแล้วในระบบคอมพิวเตอร์จะมีระบบปฏิบัติการเป็นตัวจัดสรรเวลาให้กับแต่ละโปรแซส รูปที่ 2.4.1 แสดงการทำงานแบบเรียลไทม์



(ก)



(ข)

รูปที่ 2.4.1 (ก) โปรแกรมแบบอนุกรม(Serial Control)

(ข) โปรแกรมแบบขนาน(Concurrent Control)

จากรูปที่ 2.4.1 ผลลัพธ์ของโปรแกรมทั้งสองจะต้องได้ออกมาเหมือนกันหามีการจัดสรรเวลาในการทำงานของแต่ละโปรเซสที่มีประสิทธิภาพ โดยรูป (ก) เป็นการทำงานาของโปรแกรมแบบอนุกรมซึ่งจะใช้เวลาในการทำงานมาก ในขณะที่รูป (ข) เป็นการทำงานของโปรแกรมแบบขนานซึ่งสามารถที่จะทำงานเป็นแบบเรียลไทม์ได้ ซึ่งในโครงงานนี้เลือกวิธีการออกแบบโปรแกรมเป็นแบบรูป (ข) เพราะสามารถสร้างเป็นโปรแกรมแบบเรียลไทม์ได้

2.5 สถาปัตยกรรม TMS320C50

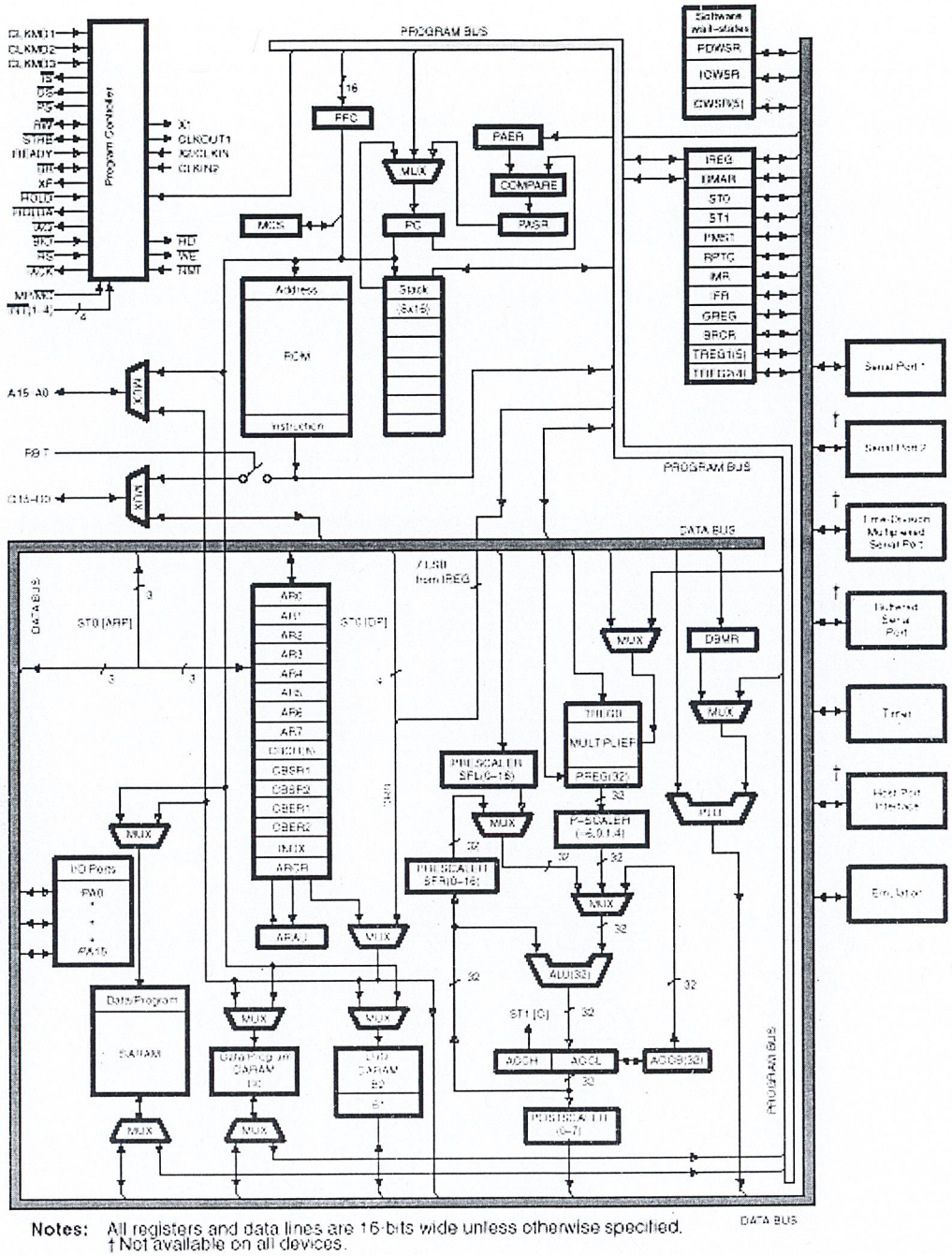
TMS320C50 เป็นโปรเซสเซอร์ที่ทำงานในกระมวลผลเชิงตัวเลขหรือ DSP (Digital Signal Processing) ซึ่งตระกูล TMS320 จะมีอยู่หลายเบอร์ด้วยกัน เช่น TMS320C50, TMS320C51 และ TMS320C53 ซึ่งเป็นการนำเอาสถาปัตยกรรมของ TMS320C25 มาปรับปรุงให้ดีขึ้นทั้งในด้านความสะดวกในการใช้งานและในเรื่องความเร็วในการประมวลผล

จุดเด่นของ TMS320C50

1. มีหน่วยความจำข้อมูล(RAM) บนบอร์ดขนาด 10 กิโลเวิร์ด
2. เวลาที่ใช้ในการประมวลผลจะเร็วขึ้นคือ 30-50 นาโนวินาทีต่อคำสั่ง
3. สามารถนำมาใช้งานแทน TMS320C1X และ TMS320C2X ได้
4. มีหน่วยความจำข้อมูล(RAM)สำหรับโปรแกรม/ข้อมูลขนาด 9K × 16 บิต
5. มี ROM ขนาด 2K × 16 บิต ไว้สำหรับการบูต
6. สามารถต่อหน่วยความจำข้างนอกได้ถึง 224 K × 16 บิต ซึ่งประกอบด้วย หน่วยความจำสำหรับเก็บโปรแกรม 64K หน่วยความจำเก็บข้อมูล 64K,สำหรับอินพุท/เอาท์พุทและอื่นๆ อีก 64 K
7. มี ALU (Arithmetic Logic Unit) , ACC (Accumulator) และ ACCB (Accumulator Buffer) ขนาด 32 บิต
8. มี PLU (Parallel Logic Unit) ขนาด 16 บิต

9. มีคำสั่งในการคูณ 16 บิตที่ทำงานใน 1 ไซเกิล
10. มีรีจิสเตอร์ถึง 8 ตัวในการคำนวณและเก็บค่า
11. มีสแต็ก (STACK)
12. มีคำสั่งสำหรับการเลื่อนบิตตั้งแต่ 0-16 บิต
13. มีการอ้างแอดเดรสแบบวนรอบ(Circular)โดยการใช้บัฟเฟอร์แบบวนรอบ(Circular Buffer)
14. มีคำสั่งสำหรับการทำซ้ำโดยเฉพาะสำหรับการเคลื่อนย้ายบล็อกภายในโดยใช้เพียงคำสั่งเดียว
15. มีคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูลและหน่วยความจำโปรแกรม
16. มีพอร์ตอนุกรมที่รับส่งข้อมูลแบบ ฟูลดูเพล็ก (Full Duplex) สำหรับ TMS320C50 กับอุปกรณ์อื่นๆ
17. มี TDM (Time-Division Multiple) ของพอร์ตอนุกรม
18. สามารถกำหนดสัญญาณนาฬิกา โดยใช้ไทม์เมอร์ (Timer) , เคาน์เตอร์ (Counter)
19. มีส่วนของโปรแกรม (Software) ที่ใช้ได้ทั้งให้ หยุด (Stop) , เริ่ม (Start) และรีเซต(Reset)
20. มีพอร์ตอินพุท/เอาท์พุทได้ถึง 64 K และมี 16 ตำแหน่ง สำหรับการเข้าถึงหน่วยความจำ
21. มีการทำงานแบบไปป์ไลน์(Pipeline)
22. สามารถผลิตสัญญาณนาฬิกา โดยการหารสัญญาณนาฬิกาที่จ่ายให้แก่ซีพียู
23. ใช้เทคโนโลยีของ ซีมอส (CMOS) และใช้ไฟเลี้ยงจ่ายให้โปรเซสเซอร์เพียง 5 V

สถาปัตยกรรมของ TMS320C50 สร้างขึ้นมาเพื่อให้มีความเร็วในการทำงานด้านการประมวลผลสัญญาณเชิง ตัวเลข (DSP) และเพื่อให้การทำงานของบัสไม่ขึ้นต่อกัน จึงแยกออกเป็น บัสโปรแกรม (Program Bus) และ บัสข้อมูล (Data Bus) ออกจากกัน โดยบัสโปรแกรมจะเป็นทางเข้าของรหัสคำสั่งและโอเปอร์แรนด์ของคำสั่ง ส่วนบัสข้อมูลจะเชื่อมต่อโดยตรงระหว่างหน่วยความจำที่ใช้เก็บข้อมูลกับวงจรการทำงานประมวลผล เช่น ALU และ AR0-AR7 ซึ่งโครงสร้างการคำนวณทางคณิตศาสตร์นี้ ยึดหลักการให้ทำงานอย่างมีประสิทธิภาพ เช่น การเลื่อนบิต , การคูณ และคำสั่งทางลอจิก



Notes: All registers and data lines are 16 bits wide unless otherwise specified.
† Not available on all devices.

รูปที่ 2.5.1 บล็อกไดอะแกรมของ TMS320C50

2.5.1 การจัดตำแหน่งขาของ TMS320C50

ตารางที่ 2.5.1 แสดงตำแหน่งขาและหน้าที่การทำงานของ TMS320C50

สัญญาณ	ขา	สถานะ	การทำงาน
กลุ่มบัสแอดเดรสและบัสข้อมูล(Address and Data Bus)			
A15(MSB)	77	I/O/Z	เป็นบัสแบบขนาน (Parallel Address Bus) ที่ใช้สำหรับชี้ตำแหน่งของ หน่วยความจำข้อมูลและหน่วยความจำโปรแกรม หรือ I/O ภายนอก เมื่ออยู่ในโหมดโฮลด์ (Hold Mode) ก็จะอยู่สถานะของอิมพีแดนซ์สูง (High Impedance) สัญญาณเหล่านี้ใช้เป็นอินพุต สำหรับ DMA ภายนอกของแรมภายใน (On-Chip Single Access RAM) ซึ่งจะเป็นอินพุต เมื่อ $\overline{HOLDA}$ และ $\overline{BR}$ ถูกขับ (Drive) ให้มีสถานะต่ำ (Low)
A14	76		
A13	75		
A12	74		
A11	73		
A10	72		
A9	64		
A8	63		
A7	62		
A6	61		
A5	60		
A4	59		
A3	58		
A2	57		
A1	56		
A0 (LSB)	55		

สัญญาณ	ขา	สถานะ	การทำงาน
D15 (MSB) D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 (LSB)	6 7 8 9 10 11 12 13 23 24 25 26 27 28 29 30	I/O/Z	เป็นบัสข้อมูลแบบขนาน (Parallel Data Bus) ใช้ส่งผ่านข้อมูลระหว่างซีพียูหลัก (Core CPU) กับหน่วยความจำข้อมูล/โปรแกรมภายนอก หรือ อุปกรณ์ I/O เมื่อไม่มีเอ้าท์พุตสัญญาณเหล่านี้จะเป็นอิมพีแดนซ์สูง หรือ เมื่อขา RS หรือ HOLD อยู่ในสถานะต่ำ (Active Low) และขา OFF เป็นสถานะต่ำ นอกจากนี้ยังใช้สำหรับ DMA ภายนอกของแรม(Single Access RAM)
กลุ่มสัญญาณควบคุมหน่วยความจำ			
$\overline{DS}$ $\overline{PS}$ $\overline{IS}$	89 91 90	O/Z	เลือกหน่วยความจำข้อมูล/โปรแกรมและ I/O ปกติมีสถานะสูงแต่เมื่อเป็นสถานะต่ำจะเป็นการติดต่อกับภายนอกเมื่อขา *OFF* อยู่ในสถานะต่ำ ขาสัญญาณควบคุมจะอยู่ในสถานะอิมพิแดนซ์สูง
READY	128	I	สัญญาณข้อมูลพร้อม (Data Ready Input) ใช้แสดงเมื่ออุปกรณ์ภายนอกส่งข้อมูลเรียบร้อยแล้ว และเมื่อยังทำงานไม่เสร็จ (READY = 0) จะต้องมีการรอ 1 ไซเคิล (Wait 1 Cycle) และเช็คขา READY อีกครั้ง ในสภาวะปกติขา READY จะทำงานหลังจากที่มีสัญญาณ *BR*

สัญญาณ	ขา	สถานะ	การทำงาน
			กลุ่มสัญญาณควบคุมหน่วยความจำ
$R / \overline{W}$	92	I/O/Z	สัญญาณอ่าน/เขียน (Read/write Signal) เป็นสัญญาณควบคุมการอ่านและเขียนข้อมูล จะเป็นสถานะอิมพีแดนซ์สูง เมื่ออยู่ในโหมดการโฮลด์ ถูกใช้ใน DMA ของ RAM ภายนอก เมื่อ $\overline{HOLDA}$ และ $\overline{IAQ}$ อยู่ในสถานะต่ำ ใช้แสดงทิศทางของบัสข้อมูลสำหรับ DMA อ่าน (Read อยู่ในสถานะสูง) และ เขียน (Write อยู่ในสถานะต่ำ)
$\overline{STRB}$	93	I/O/Z	สัญญาณสโตรบ (Strob Signal) ปกติมักอยู่ในสถานะสูง จะเป็นสถานะต่ำ เมื่อค่าของบัสภายนอกเป็นอิมพีแดนซ์สูง ในโหมดการโฮลด์เมื่อ $\overline{HOLDA}$ และ $\overline{IAQ}$ แอคทีฟสัญญาณนี้จะใช้เลือกการเข้าถึงหน่วยความจำ
RD	82	O/Z	สัญญาณเลือกอ่าน (Read Select) ขานี้จะทำงานเมื่อมีการอ่าน จะต่อโดยตรงกับ OE ของอุปกรณ์ภายนอก สัญญาณนี้จะใช้ในการอ่านหน่วยความจำโปรแกรม/ข้อมูล และ I/O ภายนอก เป็นอิมพีแดนซ์สูงเมื่ออยู่ในโหมดการโฮลด์
กลุ่มสัญญาณมัลติโปรเซสซิ่ง(Multiprocessing)			
$\overline{HOLD}$	129	I	สัญญาณโฮลด์ (Hold Input) เป็นสัญญาณที่ใช้แสดงว่ากำลังมีการติดต่อกับบัสตำแหน่ง , บัสข้อมูล และบัสควบคุม เมื่อถูกตอบรับ (Acknowledge) โดย TMS320C5X จะอยู่ในสภาวะอิมพีแดนซ์สูง
$\overline{HOLDA}$	108	O/Z	สัญญาณตอบรับสัญญาณโฮลด์ (Hold Acknowledge Signal) ใช้แสดงว่าวงจรอยู่ในสภาวะโฮลด์ (Hold State) ทั้งบัสตำแหน่ง , บัสข้อมูล และบัสควบคุมอยู่ในสภาวะอิมพีแดนซ์สูง
$\overline{BR}$	94	I/O/Z	สัญญาณการขอใช้บัส (Bus Request Signal) แสดงเมื่อมีการติดต่อกับหน่วยความจำข้อมูล สัญญาณจากขานี้ใช้กับหน่วยความจำข้อมูลที่ว่างได้ 32 กิโลเวิร์ด เมื่อขา $\overline{HOLDA}$ อยู่ในสถานะต่ำ สัญญาณจากขานี้ใช้กับ DMA ของ แรมภายนอก $\overline{BR}$ จะเป็นสถานะต่ำ เมื่อติดต่อกับแรมภายนอก

สัญญาณ	ขา	สถานะ	การทำงาน
$\overline{IAQ}$	1	O/Z	สัญญาณรับคำสั่ง (Instruction Acquisition Signal) จะแสดงค่าสถานะต่ำ เมื่อมีการเข้าถึงคำสั่งที่อยู่บนแอดเดรสบัส ใช้กับ DMA ของ RAM ภายนอก เมื่อ *HOLDA* อยู่ในสถานะต่ำ
$\overline{BIO}$	130	I	สัญญาณควบคุมบรานซ์ (Branch Control Input) ถ้าเป็นสถานะต่ำจะเป็นการให้ทำคำสั่งที่เป็นเงื่อนไข สัญญาณนี้จะทำงานเมื่อมีการเฟทซ์คำสั่งที่เป็นเงื่อนไข
XF	109	O/Z	สัญญาณติดต่อภายนอก (External Flag Output) ถูกเซตให้เป็นสถานะสูงหรือสถานะต่ำ โดยคำสั่งพิเศษหรือโดยโหลดค่าในสเตจรีจิสเตอร์ (State Register(ST1)) เมื่อรีเซตขานี้จะเป็นสถานะสูง
$\overline{LACK}$	112	O/Z	สัญญาณตอบรับการอินเทอร์รัพต์ (Interrupt Acknowledge Signal) แสดงค่าเมื่อรับค่าการอินเทอร์รัพต์ (Interrupt Vector)ซึ่งกำหนด โดย A15 - A0
การอินนิเชียลไลซ์ (Initialization),อินเทอร์รัพท์ , คำสั่งรีเซต			
INT4 INT3 INT2 INT1	41 40 39 38	I	สัญญาณอินเทอร์รัพต์จากผู้ใช้ภายนอก (External User Interrupt Input) กำหนดโดยรีจิสเตอร์ควบคุมการอินเทอร์รัพต์(Interrupt Mask Register) และ บิตอินเทอร์รัพต์โหมด (Interrupt Mode bit) สามารถรีเซตผ่านรีจิสเตอร์บอกอินเทอร์รัพต์
$\overline{NMI}$	42	I	สัญญาณอินเทอร์รัพต์ที่ไม่สามารถควบคุมได้(Non–Maskable Interrupt) เป็นอินเทอร์รัพต์ภายนอก ไม่สามารถควบคุมโดย $\overline{INTM}$ หรือ MR เมื่อ *MNI* ทำงานจะมีการอินเทอร์รัพต์
MC / MP	5	I	ขาเลือกโหมดไมโครโปรเซสเซอร์/ไมโครคอมพิวเตอร์ (Microprocessor/Microcomputer Mode) ถ้าเป็นสถานะต่ำ (Microcomputer Mode) จะทำให้โปรแกรมรอมมภายในส่งไปยังหน่วยความจำโปรแกรมในโหมดไมโครโปรเซสเซอร์

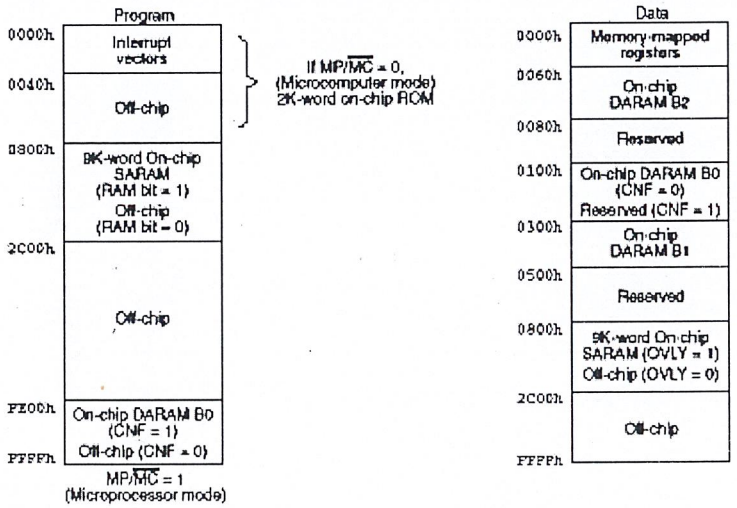
สัญญาณ	ขา	สถานะ	การทำงาน
สัญญาณออสซิลเลเตอร์/ไทม์เมอร์ และ CLKIN1/2			
CLKOUT 1	110	O/Z	สัญญาณนาฬิกาส่งออก (Master Clock Output Signal หรือ CLKIN2 Frequency) มีค่าไซเกิลเท่ากับอัตราแมชชีนไซเกิล (Machine-Cycle) ของซีพียู
CLKMD1 CLKMD2	71 103	I	CLKMD1 CLKMD2 0 0 สัญญาณนาฬิกาภายนอก เป็นสัญญาณนาฬิกาเข้าจากขาX2/CLKIN ทำให้ออสซิลเลเตอร์ (Oscillator) ภายใน และ PLL Disable 0 1 สำหรับตรวจสอบ 1 0 เป็นสัญญาณนาฬิกาอินพุท (Input Clock)สำหรับ CLCKIN2 ทำให้ออสซิลเลเตอร์ภายในหยุดทำงานและ PLL ทำงานแทน 1 1 เป็นสัญญาณนาฬิกาอินพุท สำหรับขา X2/CLKIN1 ทำให้ออสซิลเลเตอร์ภายในทำงาน และ PLL ภายในไม่ และ PLL ภายในไม่ทำงาน
X2/CLKIN	122	O	ขาอินพุทสำหรับออสซิลเลเตอร์ภายใน (Input Pin to Internal Oscillator From Crystal) ถ้าออสซิลเลเตอร์ภายในไม่ถูกต้อง สัญญาณนาฬิกาจะเป็นอินพุทสำหรับอุปกรณ์บนขานี้ แมชชีนไซเคิลภายในเป็นครึ่งหนึ่งของอัตรา CLK
X1			เป็นขาเอาท์พุทของออสซิลเลเตอร์ภายในสำหรับคริสตอล ถ้าไม่ใช้ออสซิลเลเตอร์ภายในจะไม่มีการต่อกับขานี้
CLKIN2			เป็นอินพุทสำหรับสัญญาณนาฬิกาสำหรับขับอัตราแมชชีน (Machine Rate)
TOUT			เอาท์พุทไทม์เมอร์ (Timer Output) ขานี้ให้สัญญาณพัลส์เมื่อไทม์เมอร์ภายใน (On-Chip Timer) นับถึง 0 ความกว้างพัลส์เท่ากับ CLKOUT 1 ไซเกิล

สัญญาณ	ขา	สถานะ	การทำงาน
สัญญาณพอร์ตอนุกรม			
CLKR CLKRT	46 126	I I	เป็นขาที่รับสัญญาณนาฬิกาจากข้างนอกเพื่อกำหนดให้การรับข้อมูล (DR/TDR) เข้าไปเก็บไว้ที่ RSR (Serial Port Receive Shift Register) แต่ถ้าขานี้ไม่ใช้สามารถที่จะใช้เป็นขาอินพุทของ IN0 ของ SPC/TSPC รีจิสเตอร์ได้
CLKX TCLKX	124 123	I/O/Z I/O/Z	เป็นขาที่แสดงสัญญาณนาฬิกาจากข้างนอกเพื่อกำหนดให้ DR/TDR ส่งข้อมูลไปที่ DX/TDX CLKX จะเป็นอินพุท ถ้า MCM บิต ที่การควบคุมพอร์ตอนุกรม (Serial Port Control) มีค่าเป็น 0 และอาจจะขับความถี่เป็น 1/4(CLKOUT1) เมื่อ MCM เป็น 1 ถ้าขานี้ไม่ใช้ สามารถที่จะทำเป็นอินพุทของบิต INT1 ของ SPC/TSPC รีจิสเตอร์
DR TDR	43 44	I I	เป็นขาเพื่อรับสัญญาณข้อมูล ซึ่งเมื่อรับเข้ามาแล้วจะเก็บไว้ที่ RSR (Serial Port Transmit Shift Register)
DX TDX	106 107	O/Z	เป็นขาเพื่อส่งสัญญาณข้อมูล ซึ่งข้อมูลจะส่งจาก XSR (Serial Port Receive Shift Register)
FSR TFSR/TSDD	45 125	I I/O/Z	แสดงสัญญาณการพร้อมของเฟรม (Frame Synchronization) สำหรับรับสัญญาณอินพุท TFSR จะเป็นได้ทั้งอินพุท/เอาท์พุท เมื่อ TSM ถูกเซตให้เป็น 1
FSX TFSX/TFRM	104 105	I/O/Z	แสดงสัญญาณการพร้อมของเฟรม (Frame Synchronization) สำหรับส่งสัญญาณขานี้จะเลือกได้โดยการใช้ซอฟแวร์ และจะเป็นเอาท์พุทเมื่อ TSM ถูกเซตให้เป็น 1
TCK	34	I	สัญญาณนาฬิกาตรวจสอบ JTAG (JTAG Test Clock) เป็นสัญญาณนาฬิกาแบบฟรีรันนิ่ง (Free-Running) ซึ่งมีค่าดิวตี้ไซเกิล 50% การเปลี่ยน TAP (Test Access Port) ซึ่งเป็นอินพุทสัญญาณนาฬิกาจะควบคุม TAP รีจิสเตอร์คำสั่ง (Instruction Register) หรือเลือกการทดลองรีจิสเตอร์ข้อมูล (Data Register) ที่ขอบขาขึ้นของ TCK ในการเปลี่ยน TAP จะเป็นสัญญาณเอาท์พุทจะปรากฏที่ขอบขาลงของ TCK
TDI	67	I	เป็นสัญญาณนาฬิกาเพื่อเลือกรีจิสเตอร์ในขอบขาขึ้นของ TCK
TDO	100	O/Z	เป็นการทดสอบเอาท์พุทข้อมูลในขอบขาลงของ TCK

สัญญาณ	ขา	สถานะ	การทำงาน
DMS	31	I	เป็นการเลือกโหมดทดสอบ JTAG และเป็นสัญญาณนาฬิกาอินพุทที่ทดสอบพอร์ต ตรวจสอบการเข้าถึง (Test Access Port (TAP)) จะทำงานที่ขอบขาขึ้นของ TCK
$\overline{TRST}$	2	I	ทดสอบการรีเซตจะเป็นสถานะสูง
EMU0 EMU1/OFF	118 119	I/O/Z	ขาอีมูเลเตอร์ 1 / ขาหยุดการทำงานของเอาท์พุท (Emulator Pin 1 / Disable all Output) จะทำงานที่สถานะต่ำ

2.5.2 การจัดหน่วยความจำของ TMS320C50

TMS320C50 จะมีการแบ่งหน่วยความจำเป็นส่วนของหน่วยความจำโปรแกรม และหน่วยความจำข้อมูลดังรูปที่ 2.5.2



รูปที่ 2.5.2 แสดงการจัดตำแหน่งของหน่วยความจำของ TMS320C50

หน่วยความจำข้อมูล(Data Memory)

ในการออกแบบของ TMS320C5X จะใช้สถาปัตยกรรมแบบฮาร์วาร์ด (Harward) ซึ่งเป็นการแยกหน่วยความจำข้อมูล และหน่วยความจำโปรแกรมแยกออกจากกัน ซึ่งจะมีผลดีในเรื่องความเร็ว หน่วยความจำข้อมูล ของ TMS320C5X สามารถขยายได้ถึง 64K × 16 บิต ซึ่ง TMS320C51 มีแรมบนชิพขนาด 1056 เวิร์ด บนชิพ ซึ่งจะมีข้อดีคือ

1. มีการปฏิบัติการได้เร็วไม่มีการรอ (Wait State)
2. ทำงานโดยวิธีการไปป์ไลน์ ทำให้มีการทำงานได้อย่างรวดเร็ว
3. ช่วยลดค่าใช้จ่ายเรื่องการต่อหน่วยความจำภายนอก
4. ช่วยประหยัดไฟ ซึ่งถ้าให้การต่อหน่วยความจำภายนอกจะเปลืองกว่า

ในการกำหนดการใช้หน่วยความจำข้อมูลจะต้องมีการเซ็ตค่าของบิตของรีจิสเตอร์ควบคุมหน่วยความจำ (Control State Register) ดังนี้

ตารางที่ 2.5.2 แสดงการเซ็ตบิตรีจิสเตอร์ควบคุมหน่วยความจำเพื่อใช้หน่วยความจำข้อมูล

CNF	OVLY	DRAM B0	DRAM B1	DRAM B2	SARAM	Off-Chip
0	0	100h-2FFh	300h-4FFh	60h-7Fh		800h-FFFFh
0	1	100h-2FFh	300h-4FFh	60h-7Fh	800h-2BFFh	2C00h-FFFFh
1	0	-	300h-4FFh	60h-7Fh		800h-FFFFh
1	1	-	300h-4FFh	60h-7Fh	800h-2BFFh	2C00h-FFFFh

เนื่องจากการติดต่อกับรีจิสเตอร์ต่างๆ และการส่งค่าต่างๆ จะกระทำบนเพจ 0 ซึ่งจะเป็นการใช้แบบส่งแบบรีจิสเตอร์ (Register Memory Map) ดังตารางที่ 2.5.3

ตารางที่ 2.5.3 แสดง ตำแหน่งแอดเดรสข้อมูล เพจ 0

Memory-Mapped Core Processor Registers			
Name	Address		Description
	Dec	Hex	
-----	0-3	0-3	Reserved
IMR	4	5	Interrupt Mask Register
GREG	5	5	Global Memory Allocation Register
IFR	6	6	Flag Interrupt Register
PMST	7	7	Processor Mode Status Register
RPTC	8	8	Repeat Counter Register
BRCR	9	9	Block Repeat Counter Register
PASR	10	A	Block Repeat Program Address Start Register
PAER	11	B	Block Repeat Program Address End Register
TREG0	12	C	Temporary Register Used for Multiplicand
TREG1	13	D	Temporary Register Used for Dynamic Shift Count(5 bits only)
TREG2	14	E	Temporary Register Used as Bit Pointer in Dynamic Bit Test(4 bits only)
DBMR	15	F	Dynamic Bit Manipulation Register
AR0	16	10	Auxiliary Register Zero
AR1	17	11	Auxiliary Register One
AR2	18	12	Auxiliary Register Two
AR3	19	13	Auxiliary Register Three
AR4	20	14	Auxiliary Register Four

Name	Address		Description
	Dec	Hex	
AR5	21	15	Auxiliary Register Five
AR6	22	16	Auxiliary Register Six
AR7	23	17	Auxiliary Register Seven
INDX	24	18	Index Register
ARCR	25	19	Auxiliary Register Compare Register
CBSR1	26	1A	Circular Buffer 1 Start Register
CBER1	27	1B	Circular Buffer 1 End Register
CBSR2	28	1C	Circular Buffer 2 Start Register
CBER2	29	1D	Circular Buffer 2 End Register
CBCR	30	1E	Circular Buffer Control Register
BMAR	31	1F	Block Move Address Register
Memory-mapped Peripheral Register			
DRR	32	20	Data Receive Register
DXR	33	21	Data Transmit Register
SPC	34	22	Serial Port Control Register
----	35	23	Reserved
TIM	36	24	Timer Register
PRD	37	25	Period Register
TCR	38	26	Timer Control Register
----	39	27	Reserved
PDWSR	40	28	Program/Data S/W Wait-State Register
IOWSR	41	29	I/O S/W Wait-State Control Register
CWSR	42	2A	S/W Wait-State Control Register
-	43-47	2B-2F	Reserved
TRCV	48	30	TDM Data Receive Register
TDXR	49	31	TDM Transmit Data Register
TSPC	50	32	TDM Serial Port Control Register
TCSR	51	33	TDM Channel Select Register
TRTA	52	34	TDM Receive/Transmit Address Register
TRAD	53	35	TDM Received Address Register
-----	54-79	36-4F	Reserved

Name	Address		Description
	Dec	Hex	
Memory-Mapped I/O Ports			
PA0	80	50	I/O Port 50h
PA1	81	51	I/O Port 51h
PA2	82	52	I/O Port 52h
PA3	83	53	I/O Port 53h
PA4	84	54	I/O Port 54h
PA5	85	55	I/O Port 55h
PA6	86	56	I/O Port 56h
PA7	87	57	I/O Port 57h
PA8	88	58	I/O Port 58h
PA9	89	59	I/O Port 59h
PA10	90	5A	I/O Port 5Ah
PA11	91	5B	I/O Port 5Bh
PA12	92	5C	I/O Port 5Ch
PA13	93	5D	I/O Port 5Dh
PA14	94	5E	I/O Port 5Eh
PA15	95	5F	I/O Port 5Fh

หน่วยความจำโปรแกรม (Program Memory)

ในการใช้งานของหน่วยความจำโปรแกรม ซึ่งสามารถขยายได้ถึง 64 K ซึ่ง TMS320C50 จะมีรอม(ROM) , SARAM และ DARAM ซึ่งมีความเร็วสูง โดยไม่ต้องมีสภาวะการรอ (Wait State) การทำงานของ TMS320C50 สามารถใช้ร่วมกับรอมภายในขนาด 4 K ซึ่งสามารถโปรแกรมจากโรงงาน มีความเร็วในการทำงานเต็มที่ ในการเลือกใช้จะต้องกำหนดที่ขาของ MP /MC หากขาดังกล่าวมีสภาวะสูงหรือเป็น 1 ตำแหน่ง 4 K แรก จะเป็นหน่วยความจำภายนอกชิพ แต่ถ้ามีสถานะต่ำหรือเป็น 0 ตำแหน่ง 4 K เวิร์ดแรกก็จะเป็นรอมภายในชิพดังแสดงให้เห็นในรูปการจัดแบ่งหน่วยความจำ การกำหนดสถานะ (Status) ของค่าต่างๆ จะกำหนดตามตารางที่ 2.5.4

ตารางที่ 2.5.4 แสดงการกำหนดค่าสำหรับใช้หน่วยความจำโปรแกรม

CNF	RAM	*MP/$\overline{MC}$*	ROM	SARAM	DRAM B0	Off-Chip
0	0	0	0000h-07FFh			0800h-FFFFh
0	0	1				0000h-FFFFh
0	1	0	0000h-07FFh	2000h-23FFh		2C00-FFFFh
0	1	1		2000h-23FFh		0000h-FFFFh
						2C00-FFFFh
1	0	0	0000h-07FFh		FE00h-FFFFh	0800h-FDFFh
1	0	1			FE00h-FFFFh	0000h-FDFFh
1	1	0	0000h-07FFh	2000h-23FFh	FE00h-FFFFh	2C00-FDFFh
1	1	1		2000h-23FFh	FE00h-FFFFh	0000h-07FFh
						2C00-FDFFh

การใช้งานของการอินเทอร์รัพต์ จะต้องกำหนดเวกเตอร์ของแอดเดรสในหน่วยความจำโปรแกรม ซึ่งอินเทอร์รัพต์เวกเตอร์จะแสดงดังตารางที่ 2.5.5

ตาราง 2.5.5 แสดงตำแหน่งของอินเทอร์รัพต์เวกเตอร์

Name	Location		Priority	Function
	Dec	Hex		
RS	0	0	1(highest)	External reset signal
INT1	2	2	3	External user interrupt #1
INT2	4	4	4	External user interrupt #2
INT3	6	6	5	External user interrupt #3
TINT	8	8	6	Internal timer interrupt
RINT	10	A	7	Serial port receive interrupt
XINT	12	C	8	Serial port transmit interrupt
TRNT	14	E	9	TDM port receive interrupt
TXNT	16	10	10	TDM port transmit interrupt
INT4	18	12	11	External user interrupt #4
-	20-33	14-21	N/A	Reserved
TRAP	34	22	N/A	Software trap instruction
NMI	36	24	2	Nonmaskable interrupt
-	38-41	26-29	N/A	Reserved for emulation and test
-	42-47	2A-2F	N/A	Software interrupts

2.5.3 โหมดการอ้างถึงหน่วยความจำ (Memory Addressing Mode)

การอ้างแอดเดรสสามารถอ้างได้ถึง 64 K สำหรับหน่วยความจำโปรแกรม และ 96 K สำหรับหน่วยความจำข้อมูล TMS320C50 มีวิธีการอ้างแอดเดรสได้ 8 วิธี ดังนี้

1. โดยการเข้าถึงโดยตรง (Direct Address Bus) โดยการใช้แอดเดรสบัสโดยตรง ซึ่งต้องมีรีจิสเตอร์ชี้เพจของข้อมูล (Data Page Pointer (DP)) บอกด้วยว่าเป็นเพจใด DP สามารถอ้างได้ 512 เพจ แต่ละเพจจะมีค่าเท่ากับ 128 เวิร์ด ตัวอย่างเช่น ADD 01h หมายถึง เอาข้อมูลที่ชี้ที่ตำแหน่ง 010h (ซึ่ง DP ชี้บอกว่าอยู่ที่เพจใด) นำมาบวกกับ ACC
2. โดยการเข้าถึงหน่วยความจำ (Memory Map) จะเหมือนกับกับวิธีการแรกแต่จะใช้เฉพาะเพจ 0 เท่านั้น เช่น

   LAMM PMST โดยที่ PMST เป็นรีจิสเตอร์
3. โดยการใช้ ออกซิลารีรีจิสเตอร์ (Auxiliary Register) โดยการอ้างแอดเดรสที่บรรจุใน AR0 - AR7 เช่น ADD* ซึ่งจะเป็นตัวบอกว่า AR ตัวที่เท่าใด ตัวอย่างเช่น

ARP	4	4
AR4	0302h	0302h

   หน่วยความจำข้อมูล

302h	2h	2h
ACC	2h	4h

   จะเห็นว่าแอดเดรสที่ได้อยู่ใน AR4 (0302h)
4. โดยการใช้รีจิสเตอร์คำสั่ง (Instruction Register) ซึ่งเป็นการเอาแอดเดรสโดยตรงมาจากคำสั่งเลย เช่น

   ADD #0FFFh

   เป็นการนำเอาข้อมูลที่แอดเดรส 0FFh มาบวกกับ ACC โดยวิธีนี้จะอ้างได้แบบช่วงสั้น (Short Immediate) คือค่าแอดเดรส จะเป็นได้ตั้งแต่ 00h ไปจนถึง 0FFh
5. โดยการใช้ PC (Program Counter) จะเหมือนการใช้รีจิสเตอร์คำสั่ง แต่สามารถอ้างแบบช่วงยาวได้ (Long Immediate (0000h-0FFFFh)) เช่น

   ADD #0FFFh
6. โดยการเข้าถึงรีจิสเตอร์โดยตรง (Register Access) เป็นการใช้รีจิสเตอร์ที่ทำงานโดยเฉพาะ เช่น TREG0 , TREG1 , TREG2 , ARCR , DBMR ตัวอย่างการใช้คำสั่งเช่น

   APL * , AR7
7. โดยการใช้คำสั่งตัวที่ 2 เป็นตัวชี้แอดเดรส เช่น คำสั่ง BLDD #02345h จะหมายถึงให้นำเอาข้อมูลที่ชี้ที่แอดเดรส 02345h ไปเก็บไว้ที่แอดเดรส 012h
8. โดยการอ้างถึงหน่วยความจำเป็นบล็อก (Block Memory Address Register) ซึ่ง BMAR จะเป็นตัวชี้ข้อมูล ตัวอย่างการใช้งานคำสั่งเช่น

   BLDD BMAR ,012h

2.5.4 อุปกรณ์เสริม (Peripherals)

TMS320C50 สามารถที่จะต้องอินเตอร์เฟสกับภายนอกได้ 8 วิธี คือ

1. การอินเทอร์รัพต์
2. พอร์ตอนุกรม (Serial Port)
3. พอร์ตอนุกรมมแบบอาศัยการแบ่งช่วงทางเวลา (TDM Serial Port)
4. ไทม์เมอร์
5. รีจิสเตอร์ใช้โปรแกรมให้เกิดสภาวะการรอ (Software -Programable wait State)
6. พอร์ตติดต่อภายนอก (I/O Port)
7. รีจิสเตอร์หารสัญญาณนาฬิกา (Divide-by-Clock)
8. XF และ BIO

ซึ่งทั้งหมดนี้จะควบคุมผ่านรีจิสเตอร์ควบคุม โดยการอาศัยวิธีการอ้างหน่วยความจำ ในการใช้งานจะเป็นการใช้โดยการส่งลงที่แอดเดรสของหน่วยความจำโปรแกรมและข้อมูลในที่นี้จะขอกล่าวเฉพาะหัวข้อที่สำคัญๆ

การอินเตอร์รัพต์

TMS320C50 มีการอินเตอร์รัพต์ทั้งหมด 16 อินเตอร์รัพต์ (INT16-INT1) แต่จะไม่ใช้พร้อมกันทั้งหมด (โดยปกติจะใช้เพียง 9 อินพุท)

การรีเซต(Reset) จะเป็นการอินเตอร์รัพต์จากภายนอกแบบไม่สามารถหลีกเลี่ยงได้ (Nonmaskable External Interrupt) เมื่อมีการรีเซตเกิดขึ้นจะทำให้เกิดผลดังนี้

1. CNF = 0 หมายถึง การใช้หน่วยความจำข้อมูลบนชิพ
2. PC = 0000h
3. บิต INTM = 1 และ IFR = 0 ทำให้อินเตอร์รัพต์ทุกตัวไม่ทำงาน
4. บิตสถานะ (Status Bit) จะเป็นดังนี้

   0 → OV 1 → XF 1 → SXM 0 → PM 1 → HM

   0 → BRAI 0 → TRM 0 → NDX 0 → CENM1 0 → CENM2

   0 → IPTR 0 → OVLY 0 → AVIS 0 → RAM 0 → BIG

   0 → CNF 1 → INTM MP/MC(Pin) → PMST(MP/MC) 1 → C

5. GREG = 000000
6. RPTC = 00
7. IACK จะแสดงสภาวะการเก็บการรีเซ็ต

ในการควบคุมการอินเตอร์รัพต์จะมี IFR (Interrupt Flag Register) แสดงสภาวะการอินเตอร์รัพต์ และ IMR (Interrupt Mask Register) แสดงการ Mask การอินเตอร์รัพต์ ดังรูปที่ 2.5.3

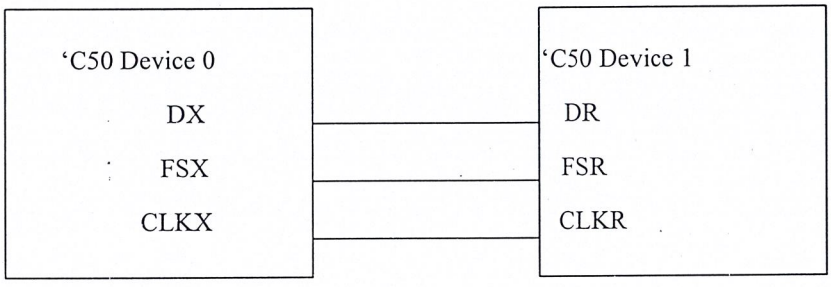
15 9	8	7	6	5	4	3	2	1	0
Reserved	*INT*4	*TXNT*	*TRNT*	*XINT*	*RINT*	*TINT*	*INT*3	*INT*2	*INT*1

รูปที่ 2.5.3 แสดงการอินเตอร์รัพต์แฟลก

ในการกำหนดให้ใช้อินเตอรรัพต์จะต้องทำการเคลียร์ที่ INTM บิต ด้วย

พอร์ตอนุกรม (serial Port)

Serial Port ของ TMS320C50 จะเป็นแบบฟูลดูเพล็ก คือสามารถรับและส่งได้ภายในเวลาเดียวกัน ทำให้สามารถติดต่อกับอุปกรณ์ภายนอกอื่นๆ ได้ไม่ว่าจะเป็น CODEC A/D หรือระบบอื่นๆ ในการต่อขาต่างๆจะเป็นตามรูปที่ 2.5.4



รูปที่ 2.5.4 แสดงการต่อพอร์ตอนุกรมกับพอร์ตภายนอก

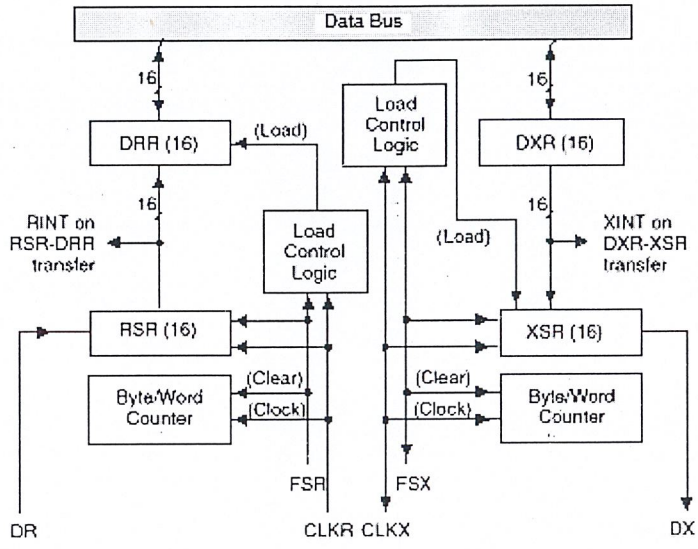
ตารางที่ 2.5.6 แสดงขาสัญญาณของพอร์ตอนุกรม

ขาสัญญาณ	ความหมาย
CLKX	สัญญาณนาฬิกาที่ใช้ในการส่ง
CLKR	สัญญาณนาฬิกาที่ใช้ในการรับ
DX	สัญญาณข้อมูลอนุกรมในการส่ง
DR	สัญญาณข้อมูลอนุกรมในการรับ
FSX	สัญญาณการซิงโครไนซ์เฟรมการส่ง
FSR	สัญญาณการซิงโครไนซ์เฟรมการรับ

ในการควบคุมพอร์ตอนุกรมจะมีรีจิสเตอร์ที่จำเป็นตามตารางที่ 2.5.7

ตารางที่ 2.5.7 แสดงรีจิสเตอร์พอร์ตอนุกรม

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

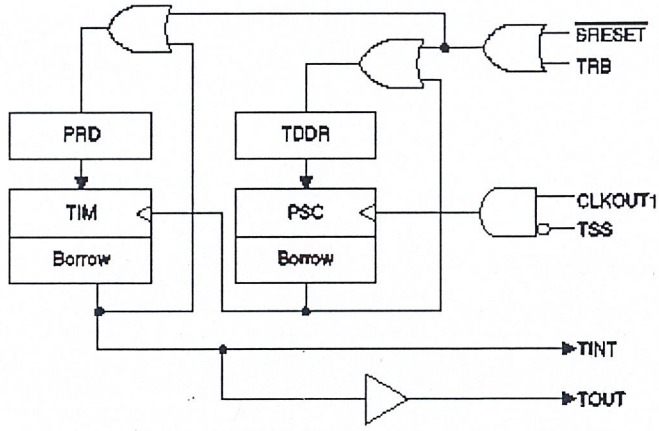


รูปที่ 2.5.5 แสดงบล็อกไดอะแกรมการทำงานของพอร์ตอนุกรม

ในการส่งข้อมูลทำได้โดยการเขียนข้อมูลลงใน DXR (Data Transmit Register) โดย XSR (Transmit Shift Register) จะเป็นตัวเลื่อนบิตตามสัญญาณนาฬิกาจนครบไปตามขา DX ส่วนการรับข้อมูลจะรับเข้ามาทางขา DR เข้ามาที่ RSR (Receive Shift Register) เมื่อครบแล้วจะทำการอินเตอร์รัพต์บอกให้ทราบ แล้วสามารถรับข้อมูลโดยการอ่านข้อมูลจาก DRR (Data Receive Register)

ไทม์เมอร์(Timer)

ในการใช้ไทม์เมอร์ของ TMS320C50 ซึ่งสามารถตั้งคาบเวลาของสัญญาณนาฬิกาได้ โดยสัญญาณจะออกที่ขา T_{OUT} ของ TMS320C50



รูปที่ 2.5.6 แสดงบล็อกไดอะแกรมของไทม์เมอร์

ซึ่งในการคำนวณคาบเวลา จะเป็นตามสูตร

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

ซึ่ง t_c เป็นคาบเวลาของสัญญาณนาฬิกาที่จ่ายให้ที่ขา CLKOUT1

TDDR (Timer Division Down Ratio)

PRD (Period Register) เป็นรีจิสเตอร์ที่สามารถใส่ค่าคาบเวลาได้

ในการทำงานจะต้องมีการใส่ค่าต่างๆ ลงบน TCR (Timer Control Register) ดังตารางที่2.5.8

ตารางที่ 2.5.8 แสดงรีจิสเตอร์ควบคุมไทม์เมอร์ (TCR)

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

2.6 DSP Starter Kit with TMS320C50 (DSK Board)

2.6.1โครงสร้างของ TMS 320C50 DSP Starter Kit

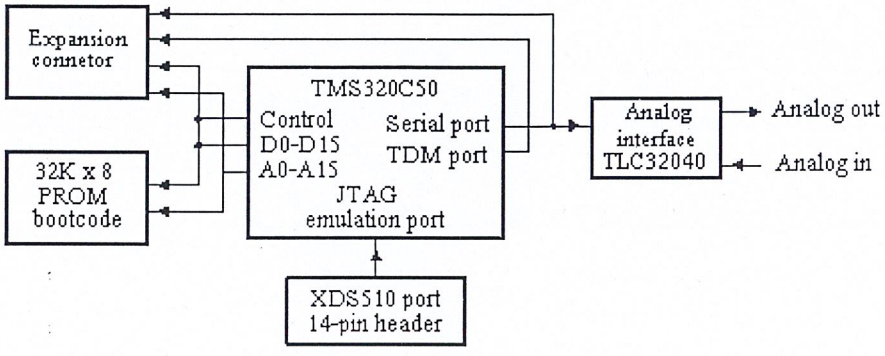
ลักษณะโดยทั่วไปของตัวมาตราฐานบอร์ด

- ใช้โปรเซสเซอร์เบอร์ TMS320C50 ซึ่งใช้งาน DSP โดยเฉพาะ
- คำสั่งจะใช้เวลาประมาณ 50 นาโนวินาที (Instruction Cycle Time)
- หน่วยความจำชนิด PROM ขนาด 32 กิโลไบต์
- สามารถใช้ในช่วงความถี่เสียง โดยติดต่อผ่านทาง TLC32040
- ใช้มาตราฐาน RCA คอนเน็คเตอร์ สำหรับสัญญาณอนาล็อกที่เข้าที่อินพุทและเอาท์พุท

เพื่อให้สามารถต่อเข้ากับไมโครโฟนและลำโพงได้โดยตรง

- ในการติดต่อกับส่วนการควบคุมจะมีคอนเน็คเตอร์ XDS510 ติดต่อกับคอมพิวเตอร์ทางพอร์ตอนุกรม
- สามารถขยาย I/O บัสได้เพื่อสำหรับการออกแบบภายนอก

รูปที่ 2.6.1 แสดงบล็อกไดอะแกรมของบอร์ดซึ่งประกอบไปด้วย โฮสอินเทอร์เฟส , อะนาล็อกอินเทอร์เฟส และอีมูเลชั่นพอร์ต ทำให้สามารถติดต่อกับพีซีได้โดยตรงผ่านทาง RS232 นอกจากนั้นยังมี PROM ขนาด 32 กิโลไบต์ ที่ใช้เก็บเคอร์เนลโปรแกรมเพื่อนำมาใช้ในการเริ่มต้นการบูตเครื่อง ในบอร์ดยังมี TLC32040 วงจรอินเทอร์เฟสสัญญาณอนาล็อก (Analog Interface Circuit:AIC) ที่มี RCA คอนเน็คเตอร์ 2 ตัวสำหรับอินพุทและเอาท์พุท

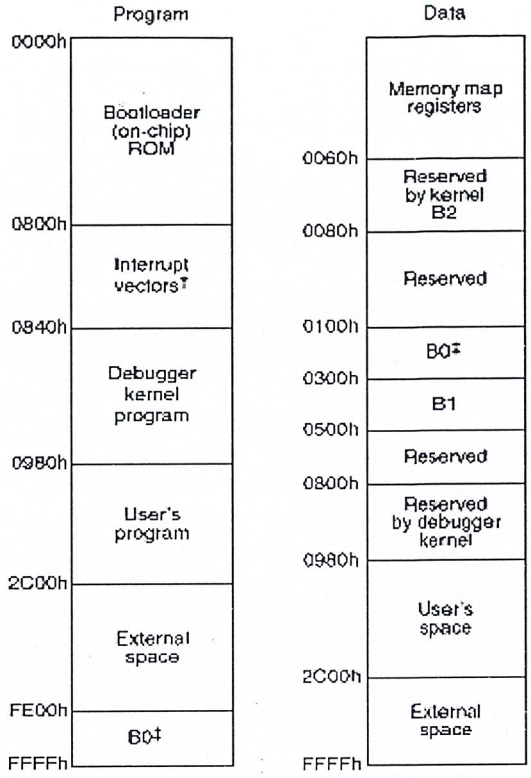


รูปที่ 2.6.1 บล็อกไดอะแกรมของ TMS320C5X DSK

หน่วยความจำใน TMS320C50 DSK นั้น จะแบ่งออกเป็นหน่วยความจำสำหรับโปรแกรมและหน่วยความจำสำหรับข้อมูลซึ่งในแต่ละแอดเดรสก็จะถูกใช้งานต่างๆ ตามแสดงในรูปที่ 2.6.2 และผู้ใช้สามารถใช้หน่วยความจำภายในได้ขนาด 10 K โดยไม่จำเป็นต้องต่อภายนอก

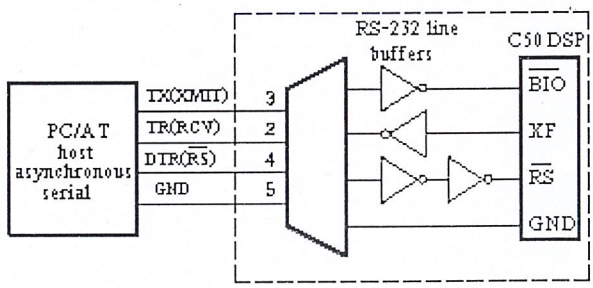
2.6.2 คุณสมบัติของ TLC32040 บนบอร์ด

- สามารถใช้ในการแปลงสัญญาณดิจิตอลเป็นสัญญาณอะนาล็อก(DAC) และการแปลงสัญญาณอะนาล็อกเป็นสัญญาณดิจิตอล(ADC)
- สามารถเปลี่ยนอัตราการแซมปลิ้งทั้งของ ADC , DAC และ ความถี่ฟิลเตอร์ได้ การติดต่อระหว่าง AIC และ TMS320C50 จะผ่านทางพอร์ตอนุกรมภายในบอร์ด



รูปที่ 2.6.2 แสดงหน่วยความจำภายใน TMS320C50 DSK

การต่อใช้งาน TMS320C50 กับ PC จะใช้ขา XF และ /BIO ที่ต่อผ่านพอร์ตอนุกรม RS232 ดังแสดงในรูปที่ 2.6.3



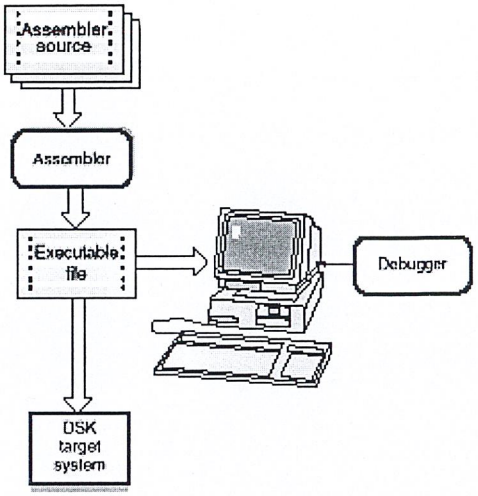
รูปที่ 2.6.3 แสดงการเชื่อมต่อระหว่าง DSK และ PC ผ่านทาง RS232

นอกจากนั้น DSK ยังมีแอสแซมเบอร์ และดีบักเกอร์ของมันเอง ซึ่งจะมีประโยชน์มากเนื่องจากภาษาแอสแซมบี้ที่ใช้ใน DSK นั้นจะสนับสนุนการทำงานด้านการประมวลผลสัญญาณ (Signal Processing) และดีบักเกอร์ก็มีความสามารถที่จะทำงานในแบบทีละขั้น (Single-Step) และเบรคพอยน์เพื่อช่วยในการแก้ไขโปรแกรม

2.6.3 การสร้างโปรแกรมเพื่อใช้กับ DSK

1. ทำการสร้างไฟล์หลัก (Source File) สำหรับโปรแกรม เช่น EAGLE.ASM
2. ทำการแปลงไฟล์หลัก โดยใช้ DSK แอสแซมเบอร์ ซึ่งมีคำสั่งดังนี้

   DSK5A EAGLE.ASM
3. หากต้องการตรวจสอบแก้ไขโปรแกรมโดยใช้ดีบักเกอร์ก็ทำได้โดยใช้โปรแกรม DSK5D.EXE ซึ่งได้ให้มากับบอร์ดแล้ว



รูปที่ 3.1.4 แสดงขั้นตอนการสร้างโปรแกรม

2.6.4 การเขียนโปรแกรมและคำสั่ง

- การกำหนดสถานะเริ่มต้นสำหรับ TMS320C50

ทุกครั้งที่มีการใช้งาน TMS320C50 จำเป็นที่จะต้องมีการกำหนดสถานะเริ่มต้น (Initialize) ตัวโปรเซสเซอร์เสียก่อนเสมอ ซึ่งจะต้องทำการเตรียมสภาวะของโปรเซสเซอร์ให้พร้อม โดยปกติจะทำไว้เมื่อเกิดการรีเซต ซึ่งการรีเซตจะเกิดขึ้นเมื่อขา *RS* อยู่ในสถานะต่ำ ซึ่งจะทำให้ IPTR (Interrupt Pointer) ซึ่งอยู่ใน PMST (Processor Mode Status Register) เกิดการเคลียร์ ทำให้เกิดการ Map เวกเตอร์ไปที่ Page 0 ในการเกิดการอินเตอรรัพต์นั้นเมื่อรีเซตจะทำให้ Disable

ในการกำหนดสถานะเริ่มต้นสำหรับโปรเซสเซอร์นั้นจะประกอบด้วย

- การ Map หน่วยความจำ และรีจิสเตอร์ควบคุม
- การเซตโครงสร้างของอินเตอร์รัพต์
- โหมดการควบคุมต่างๆ (OVM , SXM , PM ,AVIS , NDX , TRM)
- การควบคุมหน่วยความจำ (RAM , OVLY ,CNF)
- การกำหนดรีจิสเตอร์ช่วย (Auxiliary Register) และตัวชี้รีจิสเตอร์ช่วย
- การกำหนดตัวชี้เพจข้อมูล (Data Page Pointer :DP)

การกำหนดค่าใน PMST (Processor Mode Status Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IPTR						0	0	0	AVIS	0	OVLY	RAM	MP/MC	NDX	IRM	BRAT

รูปที่ 2.6.5 แสดงการกำหนดค่าใน PMST

ค่า IPTR เราจะกำหนดที่ IPTR = 10000B หรือเป็นที่เพจ 0 ซึ่งตรงกับโปรแกรมตำแหน่ง 8000h ส่วนการกำหนดการใช้ RAM ภายใน จะต้องใช้ RAM 9 K ซึ่งอยู่ที่ตัวชิพ กำหนด RAM ,OVLY เป็น "1"

RAM ที่ใช้จะเป็น SARAM โดยที่หน่วยความจำโปรแกรมมอยู่ที่ 0800h-2BFFh และหน่วยความจำข้อมูลอยู่ที่ 0800h-2BFFh

การเขียนโปรแกรมทางด้านคณิตศาสตร์และลอจิก (Logic and Arithmetic Operations)

TMS 320C50 มี PLU (Parallel Logic Unit) ซึ่งจะมีการกระทำทางลอจิก (Logic)ที่มีความเร็วมาก ในส่วนนี้มีรีจีสเตอร์ซึ่งใช้ในการประมวลผล ประกอบไปด้วย

1.) สเกลลิงชิฟเตอร์ (scaling shifter) 16 บิต

2.) ตัวคูณแบบขนาน16*16 บิต

3.) อริธเมติก ลอจิก ยูนิต (arithmetic logic unit: ALU) 32 บิต

4.) แอคคิวมูเลเตอร์ (accumulator : ACC) 32 บิต

5.) แอคคิวมูเลเตอร์บัฟเฟอร์ (accumulator buffer : ACCB)32 บิต

ซึ่งมีประสิทธิภาพในการประมวลผล ทางด้าน คณิตศาสตร์ และ ลอจิก ที่จำเป็นในการใช้งานทางด้านการประมวลผล ทางดิจิตอล

การใช้สเกลลิ่งชิฟเตอร์ (scaling shifter) โปรเซสเซอร์ตัวนี้สามารถประมวลผลแล้วทำการเลื่อนบิตได้ ทำให้สะดวกสำหรับการคูณในรูปแบบของ 2^a ตัวอย่างเช่น

LACC #01111h, 8

ซึ่งเป็นการโหลดค่าเข้า ACC ด้วยค่า 01111h แต่จะเลื่อนไป 8 บิต ผลคือ ACC =0111100000000B

คำสั่งของ TMS 320C50 ที่สำคัญ

ในการใช้งานของโปรเซสเซอร์ TMS320C50 จะต้องเขียนโปรแกรมเป็นภาษาแอสเซมบลีจึงจำเป็นต้องทราบคำสั่งและวิธีใช้ซึ่งมีอยู่มากมาย สามารถสรุปได้ดังหัวข้อต่อไปนี้

1.) คำสั่งที่เกี่ยวกับแอคคิวมูเลเตอร์ และ หน่วยความจำ ซึ่งเป็นคำสั่งที่ใช้บ่อย และสำคัญ

LACC โหลด ACC โดยการชิฟต์เข้ามา

LACL โหลดค่าเข้ามาในเวิร์ดต่ำของ ACC

ซึ่งคำสั่งเป็นการนำค่ามาเก็บไว้ที่แอคคิวมูเลเตอร์ซึ่งมีขนาด 32 บิต จะมีทั้งนำมาเก็บทั้ง 32 บิต หรือ ขนาด 16 บิต คำสั่งจะอำนวยความสะดวกในการใช้งานเพราะมีการเลื่อน ข้อมูลเข้ามาด้วย

SACH เก็บค่าในเวิร์ดสูงของ ACC โดยการชิฟต์เข้ามา

SACL เก็บค่าในเวิร์ดต่ำของ ACC โดยการชิฟต์เข้ามา

จะตรงข้ามกับคำสั่งที่ผ่านมา ซึ่งจะมีทั้ง 32 บิต และ 16 บิต ส่วนอีกคำสั่งหนึ่งเป็นการติดต่อระหว่าง ACC กับ เมมโมรีรีจิสเตอร์แมป (memory register map) คือ

SAMM ส่งค่า ACCL ไปยังเมมโมรีแมปรีจิสเตอร์

LAMM โหลด ACC ด้วยค่าในเมมโมรีแมปรีจิสเตอร์

2.) คำสั่งเกี่ยวกับ รีจิสเตอร์ช่วย และ เพจข้อมูล

LAR โหลด AR

LDP โหลดเพจข้อมูล

MAR ขยายรีจิสเตอร์ช่วย (Modify Auxiliary Register)

ซึ่งจะมีรีจิสเตอร์ช่วยถึง 8 ตัวคือ AR0-AR7 ในการใช้งานจะต้องกำหนด ARP (Auxiliary Register Pointer) ด้วย ส่วนในการกำหนดเพจ ของข้อมูลมีทั้งหมด 512 เพจ เพจละ 128 เวิร์ด ซึ่งจะกล่าวในเรื่องหน่วยความจำอีกครั้งหนึ่ง

3.) คำสั่งเกี่ยวกับ พาราเลลลอจิกยูนิต (Parallel Logic Unit)

TMS320C50 มีคำสั่งเกี่ยวกับ logic ซึ่งจำเป็นในการใช้งานถึงระดับบิต ที่สำคัญคือ

SPLK เก็บค่าลงบนหน่วยความจำข้อมูล

ซึ่งเป็นคำสั่งการเก็บค่าลงบนหน่วยความจำข้อมูล ซึ่งกระทำในระดับบิต ส่วนมากจะใช้ในการควบคุมบิตต่างๆ ในรีจิสเตอร์ ที่จะเป็นไม่ว่าจะเป็น PMST (Processor Mode Status Register) , IMR (Interrupt Mask Register) , BRCR (Block Repeat Countor Register) เป็นต้น

4.) คำสั่งในการคูณ

ในการประมวลผล การคูณมีความจำเป็นอย่างมาก TMS320C50 จึงมีคำสั่งที่ใช้รองรับเกี่ยวกับการประมวลผลทางดิจิตอลไว้มากไม่ว่าจะเป็นการคอนโวลูชัน (convolution) , คอร์รีเลชั่น(correlation)

LT โหลด TREGO

MACD การคูณ / แอคคิวมูเลต (accumulate) โดยชิฟต์ข้อมูล

ZPR Zero Product Register

คำสั่ง MACD เป็นการคูณแล้วเลื่อนข้อมูลไปในตัวซึ่งสะดวกในการทำ คอนโวลูชัน

5.) คำสั่งสำหรับการกระโดด

B กระโดดโดยไม่มีเงื่อนไข

BANZ กระโดด ถ้า ARX = 0

BCND กระโดดโดยมีเงื่อนไข

CALL เรียก โดยการไม่มีเงื่อนไข

RETI รีเทอร์นอินเตอร์รัพต์ (Return Interrupt)

RET รีเทอร์น (Return)

จะเห็นว่าคำสั่ง BCND (Branch Condition) จะเป็นคำสั่งที่สำคัญเพราะมีเงื่อนไขมากดังนี้

ACC = 0	EQ
ACC $\neq$ 0	NEQ
ACC $<$ 0	LT
ACC $\leq$ 0	LEQ
ACC $>$ 0	GT
ACC $\geq$ 0	GEQ
C = 0	NC
C = 1	C

OV = 0	NOV
OV = 1	OV
BIO low	BIO
TC = 0	NTC
TC = 1	TC
Uncondition	UNC

6.) คำสั่งที่เกี่ยวกับการควบคุมระบบ

จากที่ได้กล่าวในพาราเลลลอจิกยูนิต (Parallel Logic Unit) ซึ่งกระทำระดับบิต จำเป็นสำหรับการควบคุมระบบแล้ว ยังมีคำสั่งซึ่งเจาะจงระดับบิตมากขึ้น ดังนี้

SETC	เซตบิต
CLRC	เคลียร์บิต
BIT	ทดสอบบิต

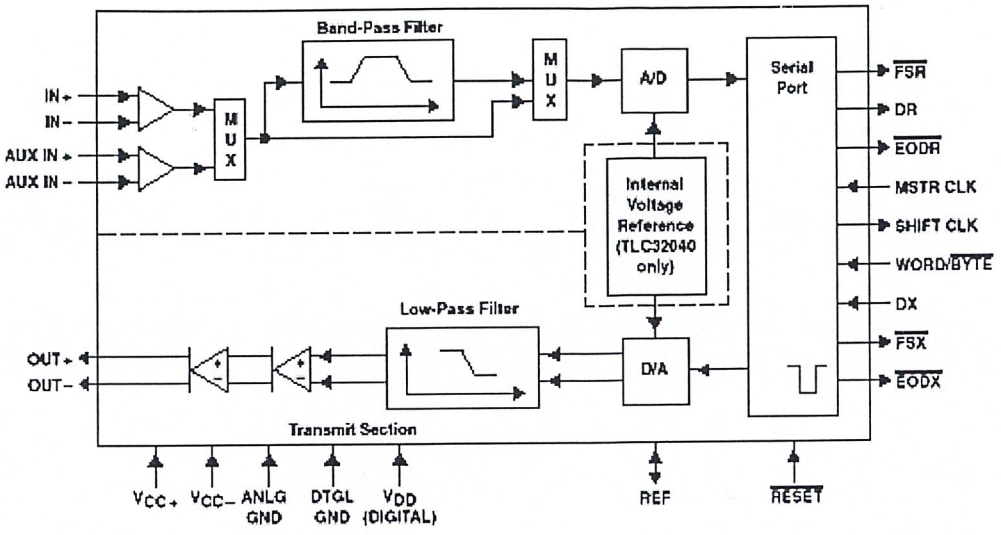
ซึ่งคำสั่ง BIT จะใช้มากเพราะมีเงื่อนไขหลายอย่างเช่น BIT 0h , 15 เลข 15 จะเป็นตัวกำหนด บิต ซึ่งมีได้ตั้งแต่ 0 ถึง15 นั้นหมายถึงเราจะทดสอบบิตได้ทุกตำแหน่งในหน่วยความจำด้วย

2.6.5 วงจรอินเทอร์เฟสสัญญาณอนาล็อก TLC32040

TLC 32040 เป็น ชิฟซีมอส (CMOS Chip) ที่ใช้เชื่อมต่อกับ TMS 320C50

ลักษณะสำคัญของ TL320C40 มีดังนี้

- ใช้เทคโนโลยีการผลิต Advance Lin CMOSTM Silicon-Gate Process
- ความละเอียดของ ADC และ DAC เป็น 14 บิต
- สามารถเปลี่ยนแปลงอัตราการสุ่มข้อมูลของ ADC และ DAC ได้ถึง 19,200 ครั้ง/วินาที
- มี Switched-Capacitor Antialiasing Input Filter และ Output-Reconstruction Filter
- มีพอร์ตควบคุมสำหรับใช้ในการติดต่อได้โดยตรงกับTMS3211,TMS320C17,TMS320C25
- สามารถปรับอัตราการเปลี่ยนแปลงของADCและDACทั้งแบบซิงโครนัส หรือ อะซิงโครนัส โดยอาศัยโปรแกรมช่วยในการควบคุม



รูปที่ 2.6.6 แสดงฟังก์ชั่นบล็อกไดอะแกรม ของ TLC 32040

ตำแหน่งขาและหน้าที่การทำงานของแต่ละขา

ตารางที่ 2.6.1 แสดงตำแหน่งขาและหน้าที่การทำงานของแต่ละขา

ชื่อ	หมายเลข	I/O	รายละเอียด
ANLG GND	17,18		กราวด์อนาล็อก
AUX IN+	24	I	ขาอินพุทเสริมแบบ นอน-อินเวอร์ติ้ง
AUX IN-	23	I	ขาอินพุทเสริมแบบ อินเวอร์ติ้ง
DGTL GND	9		กราวด์ดิจิตอล
DR	5	O	ใช้สำหรับส่งเอาท์พุท ADC จาก AIC (Analog Interface Circuit) ไปยัง TMS320C50 ผ่านทางพอร์ตอนุกรม ซึ่งการส่งผ่านพอร์ตอนุกรมจะต้องซิงค์กับ Shift Clock
DX	12	I	ใช้รับอินพุท DAC หรือคำสั่งการควบคุมจาก TMS320C50 ผ่านทางพอร์ตอนุกรมและจะต้องซิงค์กับสัญญาณ Shift Clock
EODR	3	O	สัญญาณหยุดรับข้อมูล(End of Data Receiver):ในการติดต่อกันผ่านทางพอร์ตอนุกรมในโหมดเวิร์ด (Word Mode) สัญญาณ /EODR จะอยู่ในสภาวะต่ำทันทีเมื่อทั้ง 16 บิต ของเอาท์พุท A/D ได้ถูกส่งจาก AIC ไปยัง TMS320C50 ซึ่งสามารถใช้สัญญาณนี้ในการอินเตอร์รัพท์ไมโครโปรเซสเซอร์ให้ทราบว่า

			สิ้นสุดการติดต่อแล้ว หรือใช้สโตรบและให้รีจิสเตอร์เลื่อนข้อมูลออก(Enable External Serial-to-Parallel Shift Register) ก็ได้ แต่ถ้าเป็นโหมดไบต์ (Byte Mode) สัญญาณ /EODR จะอยู่ในสถานะต่ำหลังจากไบต์แรกได้ส่งไปยัง TMS320C50 แล้วและยังคงรักษาสถานะต่ำจนกระทั่งไบต์ที่สองได้ส่งไป
EODX	11	O	สัญญาณหยุดส่งข้อมูล(End of Data Transmit) ก็คล้ายกับ /EODR ซึ่งจะบอกให้ทราบว่าการติดต่อจาก TMS320C50 ไปยัง AIC นั้นเสร็จแล้วทั้งในโหมดเวิร์ด และโหมดไบต์ ก็คล้ายกับ /EODR
ชื่อ	หมายเลข	I/O	คำอธิบาย
FSR	4	O	สัญญาณซิงค์การรับ (Frame Sync Receiver): ในการติดต่อทางพอร์ตอนุกรม /FSR จะอยู่ในสภาวะต่ำตลอดการส่งจาก AIC ไปยัง TMS320C50 ซึ่งบิตแรกที่จะส่งจะต้องพร้อมอยู่ที่ขา DR ก่อน /FSR จะอยู่ในสถานะต่ำ
FSX	14	O	สัญญาณซิงค์การส่ง (Frame Sync Transmit) : เมื่อสัญญาณนี้อยู่ในสถานะต่ำ พอร์ตอนุกรม TMS320C50 จะส่งบิตไปยัง AIC โดยมาที่ขา DX ในการติดต่ออนุกรมทุกโหมด /FSR จะอยู่ในสถานะต่ำตลอดการส่ง
IN+	26	I	ขาอินพุทชนิด นอน- อินเอวอร์ติ้ง
IN-	25	I	ขาอินพุทชนิด อินเวอร์ติ้ง
MST CLK	6	I	สัญญาณนาฬิกามาสเตอร์(Master Clock) จะใช้ในการควบคุมทุกส่วนภายใน AIC ไม่ว่าจะเป็นสัญญาณนาฬิกาเลื่อน(Shift Clock),สัญญาณนาฬิกาควบคุมฟิลเตอร์(Switched-Capacitor Filter Clock), A/D และ D/A Timimg
OUT+	22	O	ขาเอาท์พุทชนิด นอน-อินเวอร์ติ้ง
OUT-	21	O	ขาเอาท์พุทชนิด อินเวอร์ติ้ง
REF	8	I/O	สำหรับ TLC32040 และ TLC32040 แรงดันอ้างอิงภายในจะถูกส่งออกมาที่ขานี้ แต่ถ้าเป็น TLC32040

			และ TLC32040 แรงดันอ้างอิงจากภายนอกจะต่อเข้าที่ขานี้
RESET	2	I	รีเซตเพื่อตั้งค่า TA , TA' , TB ,RA , RA' , RB และรีจิสเตอร์ควบคุม (Control Register) ให้เป็นค่าดีฟอลท์ (Default) ซึ่งจะบอกภายหลัง
SHIFT CLK	10	O	สัญญาณนาฬิกาเลื่อนเกิดจากการหารความถี่ของสัญญาณนาฬิกามาสเตอร์ด้วย 4 ซึ่งสัญญาณนี้จะใช้ในการติดต่อทางพอร์ตอนุกรม
ชื่อ	หมายเลข	I/O	คำอธิบาย
VDD	7		ขาไฟเลี้ยงส่วนวงจรดิจิตอล(Digital Supply Voltage) 5V ± 5%
VCC+	20		ขาไฟเลี้ยงส่วนวงจรอนาล็อกที่เป็นบวก(Positive Analog Supply Voltage) 5V ± 5%
VCC-	19		ขาไฟเลี้ยงส่วนวงจรอนาล็อกที่เป็นลบ(Negative Analog Supply Voltage) 5V ± 5%
WORD/BYTE	13	O	ขานี้จะทำงานร่วมกับรีจิสเตอร์ควบคุมเพื่อใช้ในการเลือกโหมดการติดต่ออนุกรม ซึ่งมี 4 แบบ ดังนี้คือ - <u>การติดต่อแบบ อะซิงโครนัส</u> - <u>ในโหมดไบต์</u> (WORD/BYTE = สถานะต่ำ) พอร์ตอนุกรมจะติดต่อโดยตรงกับ TMS320C50 และจะติดต่อทีละ 8 บิต 2 ครั้ง ซึ่งมีขั้นตอนการทำงานดังนี้ 1. /FSX หรือ /FSR อยู่ในสถานะต่ำ 2. 8 บิตแรกถูกส่งออกไป หรือรับเข้ามา 3. /EODX หรือ /EODR อยู่ในสถานะต่ำ 4. /FSX หรือ /FSR อยู่ในสถานะสูงประมาณ 4 สัญญาณนาฬิกาที่ใช้เลื่อนข้อมูลแล้วอยู่ในสถานะต่ำ 5. 8 บิตต่อมา (ของไบต์ที่สอง)ถูกส่งหรือรับเข้ามา 6. /EODX หรือ /EODR อยู่ในสถานะสูง 7. /FSX หรือ /FSR อยู่ในสถานะสูง - <u>ในโหมดเวิร์ด</u>

			พอร์ตอนุกรมจะต่อตรงกับพอร์ตอนุกรมของ TMS320C50 และมีการติดต่อครั้งเดียว 16 บิต

ชื่อ	หมายเลข	I/O	คำอธิบาย
			ซึ่งมีขั้นตอนดังนี้ 1. /FSX หรือ /FSR อยู่ในสถานะต่ำ 2. 16 บิต ถูกส่งหรือรับเข้ามา 3. /FSX หรือ /FSR อยู่ในสถานะสูง 4. /EODX หรือ /EODR อยู่ในสถานะต่ำ - <u>การติดต่อแบบซิงโครนัส</u> ในกรณี แบนด์พาสฟิลเตอร์ (Bandpass Filter) และอัตราการแปลง A/D(A/D Conversion Timing) จะถูกกำหนดจาก TX เคาน์เตอร์ A ,TX เคาน์เตอร์ B และ TA , TA' และ TB แทนส่วนในการติดต่อทั้งในแบบโหมดเวิร์อดและโหมดไบต์นั้นมีขั้นตอนเหมือนกับการติดต่อแบบอะซิงโครนัส

การทำงานของ TLC320C40

<u>อินพุททางอนาล็อก (Analog Input)</u>

สำหรับอนาล็อกอินพุตจะมีอยู่ 2 กลุ่ม คือ IN+ , IN- และ AUX IN+,AUX IN- ซึ่งสามารถเลือกใช้งานกลุ่มใดกลุ่มหนึ่งโดยใช้ในแบบ ดิฟเฟอร์เรนเชียล หรือซิงเกิล-เอนด์ ค่าเกนสำหรับอินพุท IN+,IN-,AUX IN+ และ AUX IN- สามารถจะใช้โปรแกรมตั้งค่าได้(มี 3 ค่าคือ 1,2 หรือ 4)การเลือกใช้กลุ่มอินพุทใดทำได้โดยอาศัยการโปรแกรมควบคุม

<u>A/D Bandpass Filter,A/D Bandpass Filter Clocking และ A/D Conversion Timing</u>

สำหรับ A/D Bandpass Filter เราสามารถที่จะกำหนดให้สามารถใช้งานได้โดยอาศัยตัวโปรแกรมควบคุมความถี่ของสัญญาณนาฬิกาควบคุมการฟิลเตอร์จะเป็นตัวกำหนดทรานเฟอร์ฟังก์ชั่นของฟิลเตอร์โดยคิดอัตราส่วนจากความถี่สัญญาณนาฬิกาควบคุมฟิลเตอร์ 28 KHz ที่ความถี่ต่ำที่เริ่มมีลักษณะเป็นความถี่สูงผ่านจะมีความถี่เป็น 300 Hz อัตราการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก ก็หาได้จากความถี่ที่หาร 228 KHz ด้วย RX เคาน์เตอร์ B

<u>Analog Output</u>

ขาที่ให้สัญญาณเอาท์พุทออกมาเป็นสัญญาณอนาล็อกจะมีวงจรขยายซึ่งจะให้เอาท์พุทออกมาได้ทั้งแบบไม่มีการกลับขั้วของสัญญาณ(Noninverting)และแบบมีการกลับขั้วของสัญญาณ(inverting)ทำให้

สามารถที่จะขับทรานสฟอร์เมอร์ชนิดไฮ-บริดจ์ (Transformer Hybrid) หรือที่โหลดอิมพิแดนซ์ต่ำได้ทั้งแบบดิฟเฟอเรนเชียลหรือ ซิงเกิล-เอนด์

วงจรกรองความถี่ต่ำของ การแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก ,วงจรกรองความถี่แบบแบนด์พาสของการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก,สัญญาณนาฬิกาควบคุมกรองความถี่ต่ำ และอัตราการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก(D/A Lowpass Filter,D/A Lowpass Filter Clocking และ D/A Conversion Time)

เช่นเดียวกันกับ การกรองความถี่ของการแปลงสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอลโดยมีทรานสเฟอร์ฟังก์ชั่นจะถูกกำหนดจากอัตราส่วนกับความถี่ 228 KHz และอัตราการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกก็หาได้จากความถี่ 228 KHz หารด้วย TX เคาน์เตอร์ B

<u>การต่อกลับ(Loopback)</u>

การต่อกลับจะให้ผู้ใช้สามารถตรวจสอบวงจรได้โดย OUT+ และ OUT- จะต่อเข้าภายในกับ IN+ และ IN- ดังนั้นบิตข้อมูลของการแปลงสัญญาณดิจิตอลให้เป็นสัญญาณอนาล็อก D15-D2 จะถูกส่งไปยังขา DX และเปรียบเทียบกับบิตข้อมูลของการแปลง

ตารางที่ 2.6.2 แสดงการกำหนดค่าใน D1 และ D0 ในการทำงานในโหมดต่างๆของ AIC

D1	D0	การทำงาน
0	0	จะทำการโหลดค่าในรีจิสเตอร์ TA และ RA ไปยัง TX และ RX เคาน์เตอร์ A ตามลำดับ และจะทำการโหลด TX และ RX เคาน์เตอร์ ด้วยค่าในรีจิสเตอร์ TB และ RB
0	1	TX และ RX เคาน์เตอร์ A จะถูกโหลดด้วยค่า TA+TA' และ RA+RA' แต่จะทำการโหลด TX และ RX เคาน์เตอร์ ด้วยค่าในรีจิสเตอร์ TB และ RB
1	0	TX และ RX เคาน์เตอร์ A จะถูกโหลดด้วยค่า TA-TA' และ RA-RA' แต่จะทำการโหลด TX และ RX เคาน์เตอร์ ด้วยค่าในรีจิสเตอร์ TB และ RB
1	1	TX และ RX เคาน์เตอร์ A จะถูกโหลดด้วยค่าในรีจิสเตอร์ TA และ RA ส่วน TX และ RX เคาน์เตอร์ B จะถูกโหลดด้วยค่าในรีจิสเตอร์ TB และ RB หลังจากนั้นจะรอประมาณ 4 สัญญาณนาฬิกาเลื่อนก็จะทำการส่งครั้งที่2 (Secondary Communication) เพื่อไปโปรแกรม AIC ให้ทำงานตามต้องการ

ฟังก์ชั่นรีเซต(Reset)

ฟังก์ชั่นรีเซตจะเป็นการตั้งค่าอัตราการเปลี่ยนแปลงของการแปลง แปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกและการแปลง สัญญาณอนาล็อกเป็นสัญญาณดิจิตอล ให้เป็น 8 KHz ถ้าใช้สัญญาณนาฬิกา 5.184 MHz โดยค่าในรีจิสเตอร์จะถูกตั้งค่าเริ่มต้นเป็น

TA = RA = 9 , TA' = RA' = 1 , TB = RB = 24 และบิตในรีจิสเตอร์ควบคุมเป็น D7 = 1 , D6 = 1 , D5 = 1 , D4 = 0 , D3 = 0 , D2 = 1

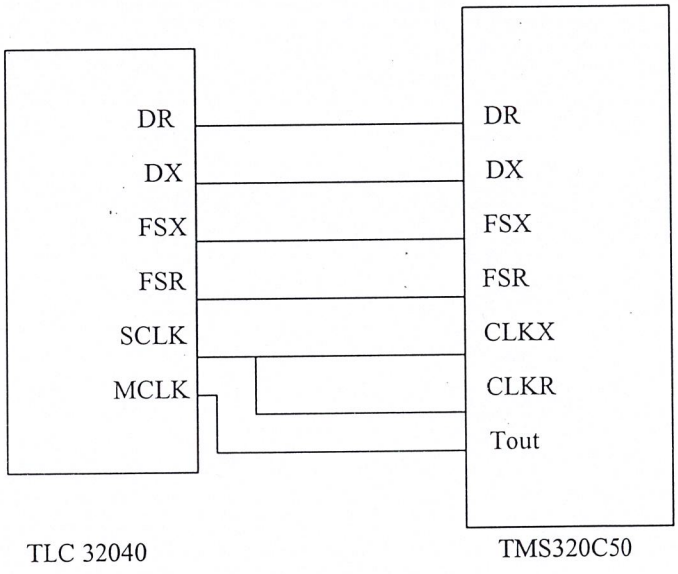
ข้อบังคับในการกำหนดค่าในรีจิสเตอร์ของ AIC

1. รีจิสเตอร์ TA ในโหมดเวิร์ด ต้องมีค่า ≥ 4
2. รีจิสเตอร์ TA ในโหมดเวิร์ด ต้องมีค่า ≥ 5
3. รีจิสเตอร์ TA' สามารถเป็นได้ทั้งค่า บวก , ลบ และ ศูนย์
4. รีจิสเตอร์ RA ในโหมดเวิร์ด ต้องมีค่า ≥ 4
5. รีจิสเตอร์ RA ในโหมดเวิร์ด ต้องมีค่า ≥ 5
6. รีจิสเตอร์ RA สามารถเป็นได้ทั้งค่า บวก , ลบ และ ศูนย์
7. (TA $\pm$ TA') จะต้องมีค่ามากกว่า 1
8. (RA $\pm$ RA') จะต้องมีค่ามากกว่า 1
9. TB จะต้องมีค่ามากกว่า 1

ถ้าไม่เป็นไปตามเงื่อนไขทั้ง 9 ข้อนี้ AIC ก็จะทำตามเงื่อนไขดังตารางที่ 2.6.3

การอินเตอร์เฟสระหว่าง TMS320C50 และ TLC32024

การใช้อินเตอร์รัพท์ของพอร์อนุกรม



รูปที่ 2.6.7 แสดงการเชื่อมต่อระหว่าง TLC32040 กับ TMS320C50

ตารางที่ 2.6.3 แสดงเงื่อนไขของการตอบสนอง AIC

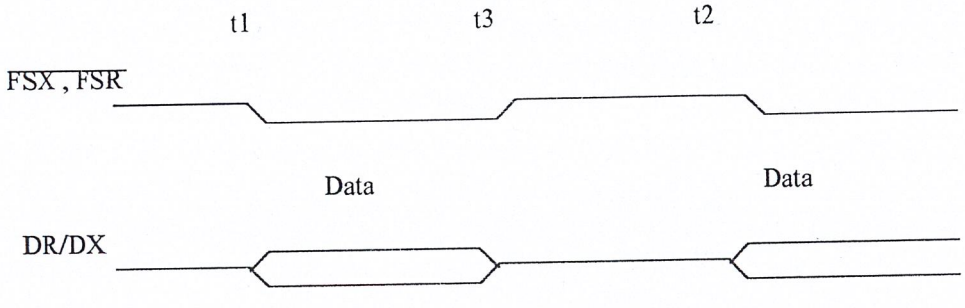
เงื่อนไข	การตอบสนองของ AIC
1. ถ้า TA + TA' = 0 หรือ 1 ถ้า TA - TA' = 0 หรือ 1	จะทำการโหลดค่าในรีจิสเตอร์ TA ลงใน TX เคาน์เตอร์ A แทน
2. ถ้า TA + TA' < 0	MODULO 64 จะถูกใช้เพื่อให้แน่ใจว่า ค่าที่บวกถูกโหลดไปไว้ที่ TX เคาน์เตอร์ A แทน TA+TA'+40h โหลดไปไว้ที่ TX เคาน์เตอร์ A
3. ถ้า RA + RA' = 0 หรือ 1 ถ้า RA – RA' = 0 หรือ 1	จะทำการโหลดค่าในรีจิสเตอร์ RA ลงใน RX เคาน์เตอร์ A
4. ถ้า RA + RA' = 0 หรือ 1	MODULO 64 ถูกใช้เหมือนข้อ 2 เพียงแต่โหลดไปที่ RX เคาน์เตอร์ A
5. ถ้า TA = 0 หรือ 1 หรือ RA = 0 หรือ 1	AIC หยุดทำงาน
6. ถ้า TA < 4 ในโหมดเวิร์ด TA < 5 ในโหมดเวิร์ด RA < 4 ในโหมดเวิร์ด RA < 4 ในโหมดเวิร์ด	การติดต่อทางพอร์ตอนุกรมของ AIC ไม่ทำงาน
7. ถ้า TB = 0 หรือ 1	โหลดค่า TB ใหม่ ด้วยค่า 24 h
8. ถ้า RA = 0 หรือ 1	โหลดค่า RB ใหม่ด้วยค่า 24 h
9. ถ้า AIC และ DSP ไม่สามารถติดต่อกันได้	จะเก็บค่า DAC Output ไว้

TMS320C50

- การส่ง โดยการเขียน DXR จะเป็นการขออินเตอร์รัพท์ของ XINT (โดยไม่มีการมาร์ค)
- การรับ เมื่อ RSR → DRR เต็มจะเกิดการอินเตอร์รัพท์ของ RINT (โดยไม่มีการมาร์ค)

TLC32040

เนื่องจากการส่งระหว่าง TLC32040 กับ TMS320C50 เป็นแบบซิงโครนัส TLC32040 จะส่งและรับทุกๆ เฟรม-ซิงโครนัส (Frame Synchronous)



รูปที่ 2.6.8 แสดงช่วงเวลาในการส่งและรับข้อมูลระหว่าง TLC32040 กับ TMS320C50

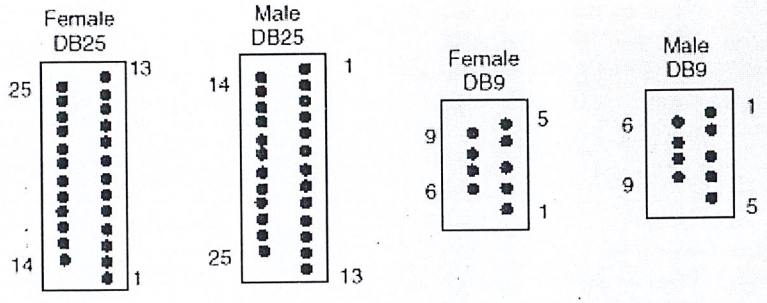
เนื่องจากความถี่ในการแซมปลิ้งสูงสุดเท่ากับ 19.2 KHz เพราะฉะนั้นข้อมูลต้องมาก่อน(ภายใน t2-t1) ต้องไม่น้อยกว่า 52.08 ไมโครวินาที ข้อมูลที่มาจาก TLC32040 แล้วทำการประมวลผลทางคณิตศาสตร์แล้วส่งกลับมาที่ TLC32040 ต้องใช้เวลาน้อยกว่า 1/Fs จึงจะสามารถได้สัญญาณที่มีความถี่เข้าเท่ากับความถี่ออก เนื่องจาก t3-t1 ช่องของข้อมูล (16 บิต) ใช้เวลาประมาณ (1/Shift Clock)×16 ประมาณเท่ากับ 6.17 ไมโครวินาที เพราะฉะนั้น 6.17 ไมโครวินาที เป็นช่วงเวลาที่ใช้ในการส่งข้อมูล 1 ข้อมูล จะต้องใช้เวลาเหลือประมวลผล (52.08 - 6.17) = 45.91 ไมโครวินาที หรือ ประมาณ (49.91 ไมโครวินาที)/(50 นาโนวินาที ต่อ 1คำสั่ง) ซึ่งเท่ากับ 918 คำสั่ง สรุปสามารถ เขียนโปรแกรมถึงกว่า 900 คำสั่ง เพื่อประมวลผลข้อมูล 1 ข้อมูลก่อนที่จะเกิดการอินเตอร์รัพท์อีกครั้ง แต่เราใช้ได้ถึงเพราะ Fs มักมีค่าน้อยกว่า 19.2 KHz

บทที่ 3

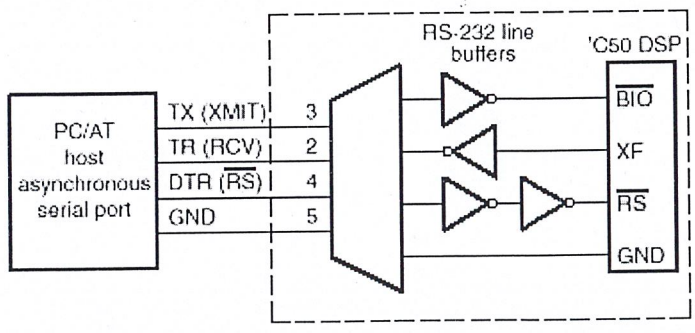
การออกแบบและการสร้าง

3.1 การเชื่อมต่อทางฮาร์ดแวร์

เครื่องวิเคราะห์แถบความถี่ของสัญญาณที่ได้พัฒนาขึ้นในโครงงานนี้แบ่งการทำงานทางฮาร์ดแวร์ออกเป็น 2 ส่วน คือ เครื่องไมโครคอมพิวเตอร์ และ DSK Board ซึ่งการติดต่อสื่อสารกันระหว่างเครื่องไมโครคอมพิวเตอร์กับ DSK Board จะเชื่อมต่อโดยผ่านทางพอร์ทอนุกรม RS232 ดังแสดงในรูปที่ 3.1.1



Signal Name	Pin Assignments	
	DB25	DB9
Protective ground	1	
Transmit data	2	3†
Receive data	3	2†
Request to send	4	7
Clear to send	5	8
Data set ready	6	6
Signal ground	7	5†
Carrier detect	8	1
Data terminal ready	20	4†
Ring indicator	22	9



รูปที่ 3.1.1 แสดงการต่อสายสัญญาณ RS232 โดยผ่าน DB9/DB25 Connector

3.2 การออกแบบทางด้านซอร์ฟแวร์

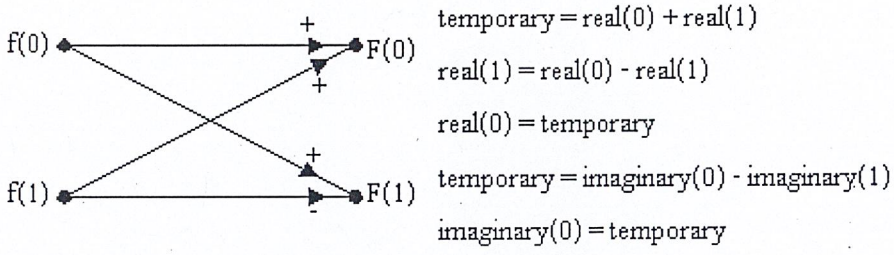
ฟาสท์ฟูเรียร์ทรานฟอร์มเป็นการคำนวณฟูเรียร์เพื่อสามารถคำนวณได้เร็วขึ้นโดยใช้คอมพิวเตอร์เป็นตัวควบคุมการคำนวณทั้งหมด

3.2.1 การคำนวณโดยใช้คอมพิวเตอร์

ในการคำนวณ โดยใช้คอมพิวเตอร์จะต้องทำการแปลงฟังก์ชั่นเอ็กโปเนนเชียลให้เป็นฟังก์ชั่นตรีโกณมิติจากสูตร

$$e^{\pm ia} = \cos a \pm i \sin a$$

เนื่องจากโปรแกรมที่ใช้เขียนไม่รู้จักฟังก์ชั่นเอ็กโปเนนเชียล(Exponential) ในการคำนวณจะยกตัวอย่างมาแค่บัตเตอร์ฟลายเดียว และจะต้องมีการเก็บค่าไว้ชั่วคราว ดังรูปที่ 3.2.1



รูปที่ 3.2.1 หลักการคำนวณโดยใช้บัตเตอร์ฟลายเดียว

จากรูปที่ 3.2.1 จะได้ค่า $F(0) = f(0) + f(1)$

$F(1) = f(0) - f(1)$

โดยข้อมูลที่ได้จะมีทั้งค่าจริงและค่าจินตภาพ

ตามรูปที่ 2 โดยจะต้องรู้ว่าสุ่มสัญญาณมากี่จุดและต้องหาค่า W มาให้ครบทุกค่าและวิธีการนี้ต้องการอินพุทแค่ 2 อินพุท โดยค่าอินพุทที่ได้จะแบ่งเป็น อัปเปอร์อินพุท(Upper Input) และโลเวอร์อินพุท(Lower Input) และจะได้เอาท์พุทมี 2 ค่าเช่นกันคือ อัปเปอร์เอาท์พุท(Upper Output) และ โลเวอร์เอาท์พุท(Lower Output) สมมุติให้มีบัตเตอร์ฟลายแค่ตัวเดียวจะได้

อัปเปอร์เอาท์พุท = อัปเปอร์อินพุท + โลเวอร์อินพุท

โลเวอร์เอาท์พุท = W (อัปเปอร์อินพุท - โลเวอร์อินพุท)

โดยวิธีการฟาสท์ฟูเรียร์ที่ใช้ในโปรแกรมของโครงงานนี้ จะเป็นแบบ Complex radix-2 decimation in frequency FFT โดยใช้การสุ่มสัญญาณ 256 จุดจะทำให้ได้จำนวนของ Stage เท่ากับ $\log_2 256 = 8$ Stage

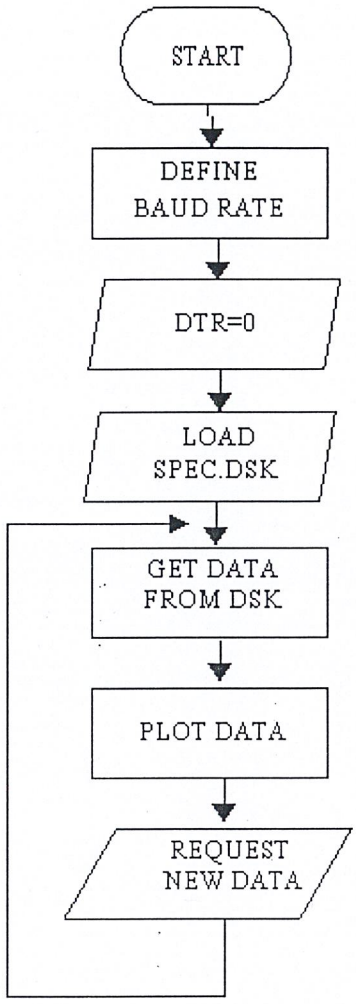
จำนวนของบัตเตอร์ฟลายในแต่ละ Stage เท่ากับ 256/2 = 128 บัตเตอร์ฟลาย

จำนวนของการคูณจำนวนเชิงซ้อน เท่ากับ $N/2 \log_2 N = 1024$ ครั้ง

3.2.2 การออกแบบซอร์ฟแวร์บนเครื่องไมโครคอมพิวเตอร์

<u>การอัพโหลด(Up Load) โปรแกรมไปยัง DSK Board</u>

การทำงานภายใน DSK Board เมื่อมีสัญญาณ DTR ตัวโปรแซสเซอร์ TMS320C50 จะถูกรีเซ็ต เมื่อโปรแซสเซอร์ถูกรีเซ็ตมันก็จะกระโดดไปทำการแอ็คซีคิ้วโปรแกรม Bootloader ซึ่งเป็นมอร์นิเตอร์โปรแกรมของ DSK Board ทำหน้าที่คอยรับข้อมูลที่อัพโหลดลงไป ขั้นตอนการทำงานของโปรแกรมบนเครื่องไมโครคอมพิวเตอร์แสดงในรูปที่ 3.2.2

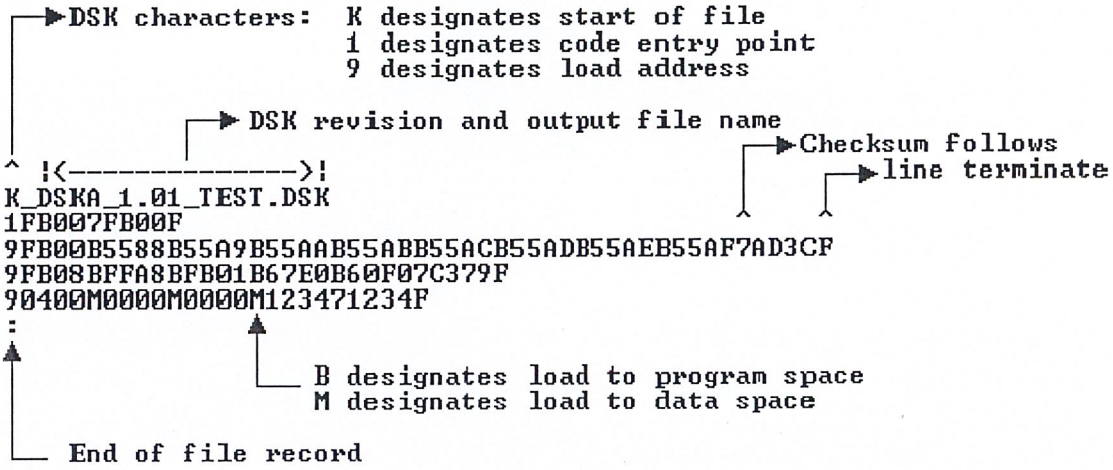


รูปที่ 3.2.2 การทำงานของโปรแกรมบนเครื่องไมโครคอมพิวเตอร์

เมื่อ DSK Board รันโปรแกรม Boot loader แล้วเครื่องไมโครคอมพิวเตอร์จะส่ง Baud Detect Word เพื่อให้ DSK Board ทำการตรวจสอบว่าค่าบอดเรทที่ใช้มีค่าเป็นเท่าไร โดยกำหนดให้บิตข้อมูล MSB ต้องเป็น “1” หรือ 1xxxxxxx

Status Word	กำหนดสภาวะให้แก่ DSK Borad
Interrupt Word	กำหนด Interrupt mask register และกำหนดโครงสร้างหน่วยความจำ
Program Length	กำหนดความยาวของโปรแกรม
Program Word	ข้อมูลโปรแกรมที่จะส่งโดยส่งไบต์ล่างก่อน
Check Sum	เมื่อส่งข้อมูลครบตามจำนวนจะส่งค่า Check sum ด้วย ถ้าข้อมูลที่ได้รับถูกต้องค่าที่ตอบกลับมาทาง TR จะเป็นค่า FFh

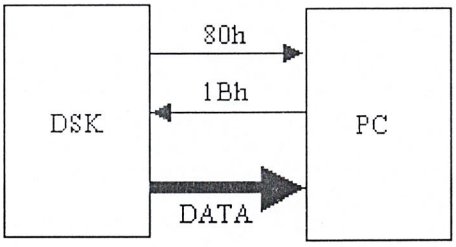
Synchronization ถ้าสัญญาณที่ตอบกลับมาว่าค่า Check sum ถูกต้อง PC-HOST จะส่งข้อมูลตอบกลับไปอีกครั้งด้วยค่า 00 เป็นการสิ้นสุดการอัพโหลดโปรแกรมลงสู่ DSK Board



รูปที่ 3.2.3 แสดงโครงสร้างของไฟล์ DSK ที่ทำการอัพโหลด

<u>การรับข้อมูลจาก DSK Board โดยผ่าน RS-232</u>

จำนวนข้อมูลที่ส่งในพอร์ตอนุกรม RS-232 มีโครงสร้างรูปแบบเป็นข้อมูล 8 บิท ไม่มีพาริตี้บิท และ สต็อปบิท 2 บิท การส่งข้อมูลจะส่งเป็นเฟรม 1 เฟรมจะบรรจุข้อมูล 256 ชุด เมื่อต้องการส่งข้อมูลจะต้องทำการแฮนด์เช็คระหว่าง DSK Board กับ เครื่องไมโครคอมพิวเตอร์เสียก่อนขั้นตอนการทำงานดังแสดงในรูปที่ 3.2.4



รูปที่ 3.2.4 แสดงการแฮนด์เช็คระหว่างการรับข้อมูล

จากรูป 3.2.4 ไมโครคอมพิวเตอร์พูลลิ่งสัญญาณพร้อมส่ง (80h) จาก DSK Board

(1) DSK Board ส่งรหัส 80h เพื่อร้องขอส่งข้อมูล (ข้อมูลรออยู่ที่บัฟเฟอร์)

(2) เมื่อไมโครคอมพิวเตอร์ได้รับรหัส 80h ก็จะส่งรหัส 1Bh เพื่อตอบสนองการร้องขอ

(3) ข้อมูล 1 เฟรมจะถูกส่งจาก DSK Board ผ่านทาง RS-232

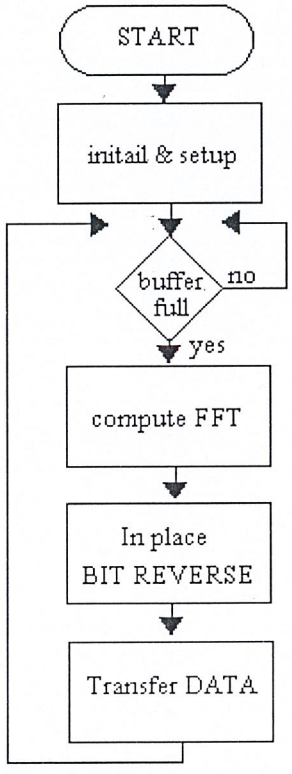
<u>การพร็อตกราฟบนจอไมโครคอมพิวเตอร์</u>

ข้อมูลที่รับได้จาก RS-232 จะถูกนำมาเก็บไว้ในบัฟเฟอร์บนเครื่องไมโครคอมพิวเตอร์จากนั้นด้วยความสามารถที่มีไม่จำกัดของไมโครคอมพิวเตอร์สามารถนำข้อมูลมาแสดงผลเป็นกราฟบนจอมอร์นิ

เตอร์และยังสามารถเพิ่มฟังก์ชั่นต่างๆให้กับข้อมูลดังกล่าว เช่น เพิ่มฟังก์ชั่นการอ่านกราฟ , บันทึกกราฟไว้สำหรับพิมพ์ ฯลฯ

3.2.3 การออกแบบซอร์ฟแวร์บน DSK Board

สำหรับซอร์ฟแวร์ในส่วนของ TMS320C50 จะเขียนด้วยภาษาแอสเซมปลี้บนหน่วยความจำโปรแกรมภายในของตัวโปรแซสเซอร์เอง ซึ่งหน้าที่ของโปรแกรมในส่วนนี้ก็คือก็คือรับข้อมูลสัญญาณดิจิตอลจากไอซีดีทูเอ(TLC32040) ซึ่งส่งสัญญาณดิจิตอลเข้ามาแบบอนุกรมเข้าทางขาดาต้ารีซีฟเวอร์ของ TMS320C50 โดยโปรแกรมาจะทำการรับข้อมูลเข้ามาเก็บในหน่วยความจำดาต้าแมมโมรี โดยโปรแกรมจะทำการรับข้อมูลโดยการอินเทอร์รัพท์ (RINT) จากนั้นนำข้อมูลมาทำการคำนวณหาแถบความถี่ของสัญญาณเมื่อคำนวนเสร็จแล้วก็จะทำการส่งข้อมูลที่ได้จากการคำนวณขึ้นไปแสดงผลบนเครื่องไมโครคอมพิวเตอร์ ดังแสดงในรูปที่ 3.2.5



รูปที่ 3.2.5 การทำงานของโปรแกรมบน TMS320C50

จากรูปที่ 3.2.5 มีการขั้นตอนการทำงานดังนี้

(1) เริ่มจากการทำการตั้งค่าความเร็วของการสุ่มสัญญาณของไอซี TLC32040 โปรแกรมการทำงานของไทม์เมอร์และทำการกำหนดการใช้งานของอินเทอร์รัพท์ต่างๆซึ่งในที่นี้จะใช้อินเทอร์รัพท์การรับส่งข้อมูลอนุกรมเพียงอย่างเดียว

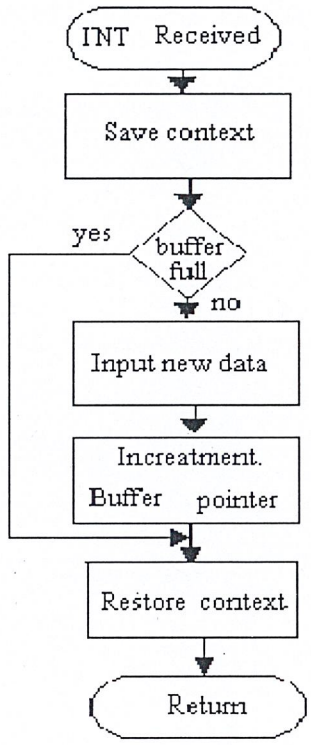
(2) ตรวจสอบบัฟเฟอร์ข้อมูลที่ได้จากการสุ่มว่าเต็ม(พร้อม)หรือยัง

(3) หากเต็ม(พร้อม) ทำการคำนวณ FFT

(4) เนื่องจากข้อมูลที่ได้หลังจากการคำนวณนี้ ตำแหน่งการวางของข้อมูลเป็นบิทรีเวอร์ซอท (Bit Reverseal) ดังนั้นก่อนที่จะทำการส่งข้อมูลจึงจำเป็นที่จะต้องทำการรีเวอร์ตำแหน่งเสียก่อน

(5) ส่งข้อมูลที่สำเร็จแล้วไปแสดงผลที่เครื่องไมโครคอมพิวเตอร์

(6) กลับไปยังขั้นตอนที่ (2)



รูปที่ 3.2.6 การทำงานของโปรแกรมบริการอินเทอร์รัพท์การรับข้อมูลอนุกรม

จากรูปที่ 3.2.6 มีขั้นตอนการทำงานดังนี้

(1) เมื่อเข้าสู่โปรแกรมบริการอินเทอร์รัพท์โปรแกรมจะทำการเก็บค่าของรีจีสเตอร์สถานะไว้

(2) ทำการตรวจสอบว่าข้อมูลที่ได้จากสุ่มเต็มยบัฟเฟอร์หรือยัง

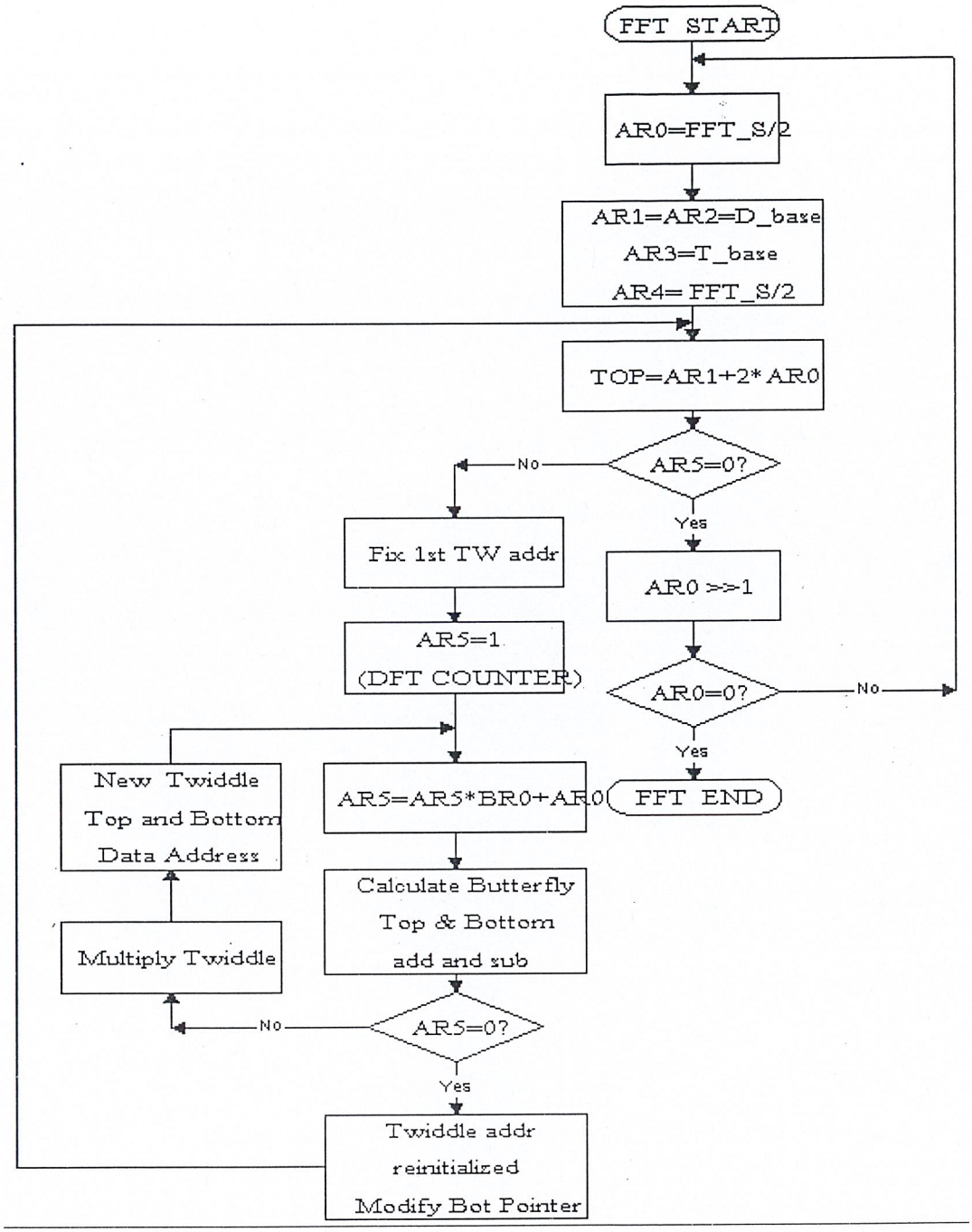
(3) ถ้าข้อมูลยังไม่เต็ม ให้ทำการโอนย้ายข้อมูลจากรีจีสเตอร์ DRR มาเก็บไว้ในบัฟเฟอร์ที่ชี้โดย AR6

(4) ถ้าข้อมูลเต็ม ให้ทำการเปลี่ยนตัวชี้บัฟเฟอร์ให้ไปชี้ในบัฟเฟอร์สำรอง

(5) เลื่อนตำแหน่งของอินเด็กซ์รีจีสเตอร์ AR6

(6) คืนค่ารีจีสเตอร์สถานะ

(7) กลับสู่โปรแกรมหลัก



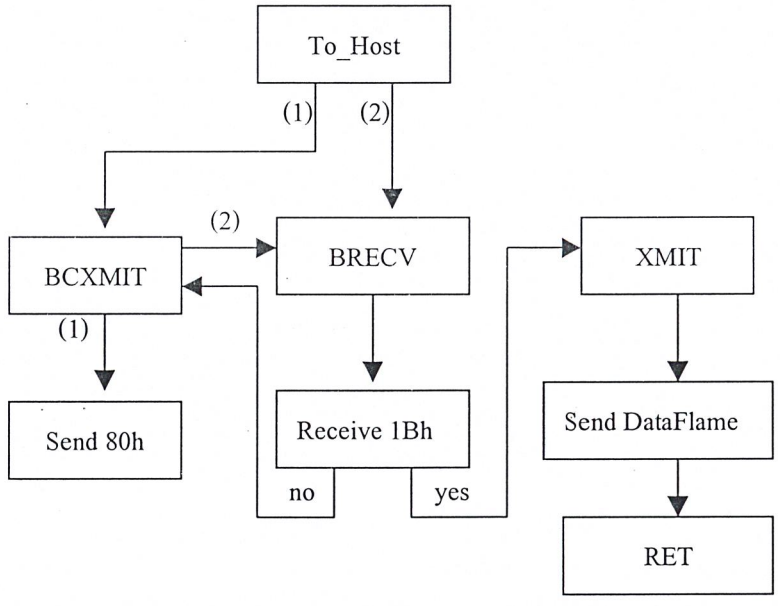
รูปที่ 3.2.7 แสดงไดอะแกรมการทำงานของการคำนวณฟาสท์ฟูเรียร์

จากรูปที่ 3.2.7 อัลกอริทึมและการทำงานของโปรแกรมคำนวณฟาสฟูเรียร์

1. กำหนดให้ AR0 มีค่าเท่ากับ N/2
2. ให้ AR1 และ AR2 เป็นตัวชี้ในหน่วยความจำในส่วนของบัตเตอร์ฟลาย AR3 เป็นตัวชี้ตำแหน่งของค่าทวิดเดิลแฟคเตอร์ และ AR4 เป็นตัวนับจำนวนของบัตเตอร์ฟลาย
3. คำนวณให้ส่วนบนของบัตเตอร์ฟลายเท่ากับ AR1+(2×AR0)
4. ตรวจดูว่าค่าใน AR4 = 0 หรือไม่ เป็นการตรวจสอบว่าโปรแกรมทำการคำนวณบัตเตอร์ฟลายครบหรือยัง ถ้าครบแล้ว AR4 จะเท่ากับศูนย์

5. ถ้า AR4 = 0 แล้วจะทำการเปลี่ยนค่าในรีจิสเตอร์ AR0 โดยให้ AR0 = AR0+AR0แบบรีเวอร์ซอล
6. ตรวจดูว่าค่าใน AR0 = 0 หรือไม่ ถ้าเท่ากับศูนย์ก็จะเสร็จสิ้นการคำนวณฟาสท์ฟูเรียร์ แต่ถ้าไม่เท่ากับศูนย์ก็จะกลับไปทำงานในข้อที่ 3 อีก
7. ในกรณีที่ค่าใน AR4 ไม่เท่ากับศูนย์แล้วโปรแกรมจะทำการลดค่าของ AR4 ออกไป 1ค่า และทำการจัดค่ารีจีสเตอร์ที่ทำหน้าที่ชี้ตำแหน่งในทวิดเดิลแฟคเตอร์ก็คือ AR3 จากนั้นให้ค่ารีจิสเตอร์ AR5 = 1+ AR0 เป็นการบวกแบบบิทรีเวอร์ซอล
8. ทำการคำนวณบัตเตอร์ฟลาย และคำนวณค่าในรีจิสเตอร์ AR5 = AR5+BR0แบบรีเวอร์ซอล
9. ตรวจดูค่าในรีจิสเตอร์ AR5 เท่ากับศูนย์หรือไม่

10. ถ้าค่าใน AR5 มีค่าไม่เท่ากับศูนย์ให้ทำการคูณค่าที่ได้จากบัตเตอร์ฟลายกับค่าของทวิดเดิลแฟคเตอร์
11. จัดค่าตำแหน่งของทวิดเดิลแฟคเตอร์และตำแหน่งของบัตเตอร์ฟลายใหม่ และกลับไปคำนวณบัตเตอร์ฟลายในข้อ 8
12. แต่ถ้าค่าในรีจิสเตอร์ AR5 มีค่าเท่ากับศูนย์แล้วให้ตั้งค่าตำแหน่งของบัตเตอร์ฟลายและทวิดเดิลแฟคเตอร์ใหม่และกลับไปทำในข้อที่ 3



รูปที่ 3.2.8 แสดงการทำงานของฟังก์ชั่น To_Host

รูปที่ 3.2.7 คือ ฟังก์ชั่น To_Host ทำหน้าที่ส่งผลการวิเคราะห์ไปให้เครื่องไมโครคอมพิวเตอร์ ขั้นตอนแรกของการทำงาน คือ แฮนด์เช็คกับเครื่องไมโครคอมพิวเตอร์(ฟังก์ชั่น BCXMIT กับ BRECV) เมื่อ

ทั้งคู่พร้อมก็จะทำการรับ-ส่งข้อมูล(ฟังก์ชั่น XMIT) โดยข้อมูลจะถูกแพ็คเป็นเฟรม, 1 เฟรมเท่ากับ 128 ชุดข้อมูล, 1 ชุดข้อมูลประกอบด้วย 8 บิทเดต้า 2 สต็อปบิท

ฟังก์ชั่น BCXMIT ทำหน้าที่ร้องของการส่งข้อมูลโดยการส่งรหัส 80h ไปบอกเครื่องไมโครคอมพิวเตอร์

ฟังก์ชั่น BRECV ทำหน้าที่รอรับสัญญาณตอบสนองการร้องขอ โดยหากรหัสที่รับได้จากเครื่องไมโครคอมพิวเตอร์เป็นรหัส 1Bh แสดงว่าเครื่องไมโครคอมพิวเตอร์ตองสนองการร้องขอการแฮนด์เช็คก็จะประสบความสำเร็จ

ฟังก์ชั่น XMIT ทำหน้าที่ส่งข้อมูลอนุกรมไปยังเครื่องไมโครคอมพิวเตอร์ โดยการไดร์ฟระดับสัญญาณที่ขา XF ของ TMS320C50 ให้เป็นไปตามค่าข้อมูลที่จะส่งและจังหวะของสัญญาณนาฬิกา

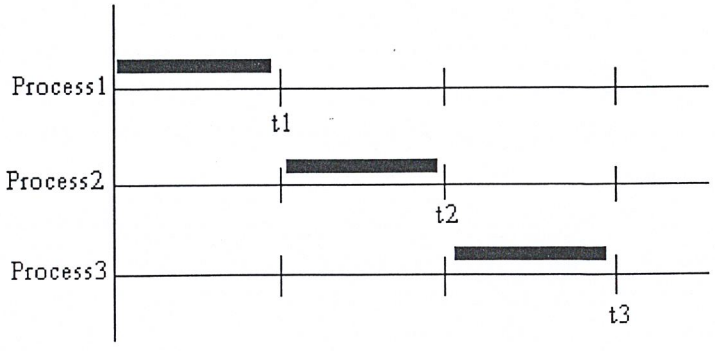
3.3 เรียลไทม์สวิทซ์ (Real Time Switch)

เรียลไทม์สวิทซ์ก็คือการจัดสรรเวลาที่แต่ละโปรแซส(Process) จะได้ยึดครองตัวโปรแซสเซอร์ ซึ่งในโครงงานนี้จะมีโปรแซสอยู่ 3 โปรแซส คือ โปรแซสการคำนวณ FFT , โปรแซสการส่งข้อมูลไปยังไมโครคอมพิวเตอร์*(ToHost)* และ โปรแซสการรับข้อมูลที่ได้มาจากการสุ่มสัญญาณอินพุท *(RINT)* ซึ่งแต่ละโปรแซสมีคุณสมบัติดังตารางที่ 3.3.1

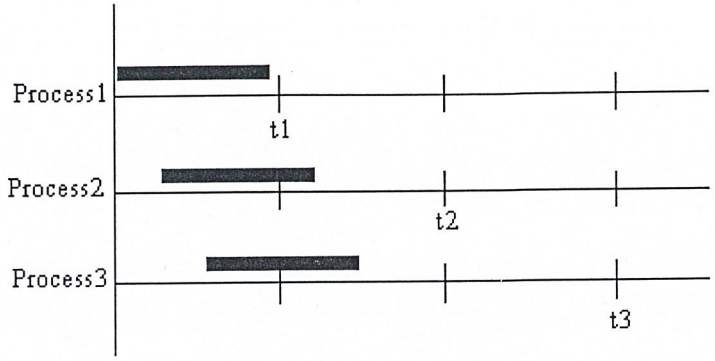
ชื่อโปรแซส	ชนิดโปรแซส	เวลาที่ใช้	เปอร์เซ็นการใช้งานโปรแซสเซอร์
1. คำนวณ FFT	Foreground	7.469 mS	95%
2. ToHost	Foreground	23.91 mS	95 %
3. RINT	Background	25.81 mS	5 %

ตารางที่ 3.3.1 แสดงเวลาที่แต่ละโปรแซสใช้ในการทำงานจนสำเร็จในรอบการทำงาน

เนื่องจากโปรแซสรับข้อมูลที่ได้จากการสุ่มสัญญาณอินพุท(RINT) เป็นโปรแซสที่มีเปอร์เซ็นการใช้งานโปรแซสเซอร์น้อยที่สุดจึงเหมาะที่จะนำมารันเป็นโปรแซสชนิดเบคกราวด์ ดังนั้นหากทำการจัดสรรการใช้งานโปรแซสเซอร์ให้มีเปอร์เซ็นการใช้งานให้มากที่สุด หรือ ให้ตัวโปรแซสเซอร์ว่างงานน้อยที่สุดจะทำให้โปรแกรมที่ออกแบบมีประสิทธิภาพสูง นำแต่ละโปรแซสมาเขียนเป็นกราฟเทียบกับแกนเวลาได้ดังรูปที่ 3.3.1



(ก) Serial Control



(ข) Concurrent Control

รูปที่ 3.3.1 การยึดครองโปรแซสเซอร์ของแต่ละโปรแซส

จากรูปที่ 3.3.1 แสดงให้เห็นการออกแบบโปรแกรมเป็นแบบทำงานแบบอนุกรม(Serial Control) และทำงานแบบขนาน(Concurrent Control) ซึ่งจะเห็นว่าโปรแกรมแบอนุกรม(รูป ก.)จะใช้เวลามากกว่าการทำงานจะสำเร็จ(terminate) นั่นก็เป็นเพราะปล่อยให้ตัวโปรแซสเซอร์ว่างงานมากเกินไปซึ่งเป็นโปรแกรมที่ไม่สามารถใช้งานเป็นโปรแกรมประเภทเรียลไทม์ได้ แต่เราสามารถจัดสรรเวลาในการทำงานเสียใหม่ดังรูป (ข) จะเห็นว่าโปรแซสที่ 1 และ โปรแซสที่ 2 สามารถที่จะใช้เวลาในขณะที่โปรแซสที่ 3 อยู่ในสภาวะรอ I/O(โปรแซสเซอร์ว่างงาน) และโปรแซสทั้งสองก็สำเร็จก่อนที่โปรแซสที่ 1 จะสำเร็จเสียอีก ซึ่งในโครงงานนี้จะใช้การออกแบบดังรูป (ข) ซึ่งจะทำให้การใช้งานโปรแซสเซอร์มีประสิทธิภาพขึ้นและเป็นโปรแกรมประเภทเรียลไทม์

บทที่ 4

การทดลองและผลการทดลอง

การทดลองที่1 การวัดความเร็วของโปรแกรมการคำนวณ FFT

วัตถุประสงค์ หาเวลาในการคำนวณ FFT เพื่อนำข้อมูลมาออกแบบเรียลไทม์สวิทช์

สมมุติฐานผลการทดลอง

การทำงานของโปรแกรมคำนวณ FFT ในทุกครั้งการทำงานจะต้องมีเวลาการทำงานที่แน่นอนถึง แม้ว่าในแต่ละครั้งของการคำนวณจะทำกับอินพุทที่ต่างค่ากัน แต่การทำงานจะมีรอบการทำงานที่ตายตัว ไม่มีเงื่อนไขพิเศษที่ทำให้การทำงานเปลี่ยนไปจากรอบการทำงานปกติ

วิธีการทดลอง

1. เชื่อมต่อบอร์ด DSK เข้ากับเครื่อง PC โดยผ่านทางพอร์ทอนุกรม
2. ดาว์นโหลดโปรแกรม FFT ลงบอร์ด DSK
3. ดาว์นโหลดโปรแกรมนับเวลาลงบอร์ด DSK
4. สั่งรันโปรแกรม-ส่งค่าเวลาที่นับได้กลับให้แก่ PC
5. อ่านค่า-บันทึกผลการทดลอง

ผลการทดลอง

โปรแกรม FFT	เวลาที่ใช้
64 จุด	1.9 mS
128 จุด	3.8 mS
256 จุด	7.7 mS
512 จุด	15.5 mS

วิเคราะห์ผลการทดลอง

ค่าเวลาที่นับได้จากโปรแกรมคือค่าของตัวนับที่เพิ่มขึ้นทีละหนึ่งเมื่อเวลาล่วงเลยไปในทุกๆ 10 ไมโครวินาที ซึ่งในที่นี้จะอิมพรีเมนท์(implement) ด้วยการใช้ไทม์เมอร์อินเทอร์รัพท์ ฉนั้นค่าเวลาที่วัดได้จากตารางผลการทดลองจึงเป็นเวลารวมของ เวลาคำนวณโปรแกรม FFT + เวลาโปรแกรมบริการไทม์

$$T_{FFT} = T_{total} + (T_c * N)$$

เมอร์อินเทอร์รัพท์ ดังนั้นเราสามารถหาค่าเวลาที่ใช้ในการคำนวณ FFT จริงๆได้จากสูตร

เมื่อ

T_{FFT} = เวลาที่ในการคำนวณ FFT

T_{total} = เวลาที่วัดได้

T_c = เวลาที่ใช้ในโปรแกรมบริการไทม์เมอร์อินเทอร์รัพท์

N = จำนวนครั้งที่ให้บริการไทม์เมอร์อินเทอร์รัพท์

คำสั่งของโปรแกรมนับเวลาที่อยู่ในโปรแกรมบริการไทม์เมอร์อินเทอร์รัพท์

```
TINT:  MAR   *,AR7        ( 50 nS )
       MAR   *+           ( 50 nS )
       RETI               (200 nS)
```

ดังนั้น

เวลาที่ใช้ในโปรแกรมบริการไทม์เมอร์อินเทอร์รัพท์ = 300 nS

ที่การคำนวณ FFT 64 จุด

เวลาที่ใช้ในการคำนวณ FFT = 1.9 mS – (300nS x 190)

= 1.843 mS

ที่การคำนวณ FFT 128 จุด

เวลาที่ใช้ในการคำนวณ FFT = 3.8 mS – (300nS x 380)

= 3.686 mS

ที่การคำนวณ FFT 256 จุด

เวลาที่ใช้ในการคำนวณ FFT = 7.7 mS – (300nS x 770)

= 7.469 mS

ที่การคำนวณ FFT 512 จุด

เวลาที่ใช้ในการคำนวณ FFT = 15.5 mS – (300nS x 1550)

= 15.035 mS

<u>วิจารณ์ผลการทดลอง</u>

ความเร็วในการคำนวณทางคณิตศาสตร์ของโปรแซสเซอร์TMS320C50 นับได้ว่าสามารถทำงานได้รวดเร็วมากเมื่อเทียบกับโปรแซสเซอร์ตระกูลอื่นๆหรือโปรแซสเซอร์ตระกูลเดียวกันในรุ่นก่อน จากการทดลองในการคำนวณโปรแกรม FFT ซึ่งเป็นการแก้ปัญหาทางคณิตศาสตร์ที่ซับซ้อน เวลาที่ใช้ในการคำนวณที่วัดได้นับได้ว่าน้อยมาก ดังนั้นจึงสามารถที่จะนำมาสร้างเครื่องวิเคราะห์แถบความถี่ได้

การทำงานของโปรแกรม FFT ต้องการเวลาในการทำงานจนสำเร็จอยู่ค่าเวลาที่ตายตัวซึ่งแน่นอน เมื่อจำนวนของอินพุทยิ่งมากเท่าไหร่ค่าเวลาที่ต้องการในการทำงานก็จะยิ่งมากขึ้นตามลำดับ ดังนั้นเราสามารถที่จะออกแบบค่าไทม์สวิทซ์ในค่าที่เหมาะสมสำหรับการทำงานของโปรแกรมในโครงงานนี้

การทดลองที่2 การวัดความเร็วของโปรแกรมส่งข้อมูลมายังเครื่องไมโครคอมพิวเตอร์

วัตถุประสงค์ หาเวลาในการคำนวณ FFT เพื่อนำข้อมูลมาออกแบบเรียลไทม์สวิทซ์

สมมุติฐานการทดลอง

การสื่อสารอนุกรมเป็นการรับ-ส่งข้อมูลที่มีอัตราความเร็วในการรับ-ส่งที่แน่นอน ตามมาตรฐาน UART คิดความเร็วเป็น บิท ต่อ วินาที (Bit per Sec.) ดังนั้นเมื่อโปรแกรมการรับ-ส่งข้อมูลตามค่าบอร์ดเรท ที่ต้องการแล้วการรับส่งข้อมูลควรจะใช้เวลาต่อหน่วยข้อมูลที่คงที่

วิธีการทดลอง

1. เชื่อมต่อบอร์ด DSK เข้ากับเครื่อง PC โดยผ่านทางพอร์ทอนุกรม
2. ดาว์นโหลดโปรแกรม ToHost ลงบอร์ด DSK
3. ดาว์นโหลดโปรแกรมนับเวลาลงบอร์ด DSK
4. สั่งรันโปรแกรม-ส่งค่าเวลาที่นับได้กลับให้แก่ PC
5. อ่านค่า-บันทึกผลการทดลอง

ผลการทดลอง

โปรแกรม ToHost	เวลาที่ใช้
64 ชุดข้อมูล	6.5 mS
128 ชุดข้อมูล	12.87 mS
256 ชุดข้อมูล	24.65 mS
512 ชุดข้อมูล	51.19 mS

วิเคราะห์ผลการทดลอง

ค่าเวลาที่นับได้จากโปรแกรมคือค่าของตัวนับที่เพิ่มขึ้นทีละหนึ่งเมื่อเวลาล่วงเลยไปในทุกๆ 100 ไมโครวินาที ซึ่งในที่นี้จะอิมพรีเมนท์(implement) ด้วยการใช้ไทม์เมอร์อินเทอร์รัพท์ ฉนั้นค่าเวลาที่วัดได้จากตารางผลการทดลองจึงเป็นเวลารวมของ เวลาคำนวณโปรแกรม ToHost + เวลาโปรแกรมบริการไทม์เมอร์อินเทอร์รัพท์ ดังนั้นเราสามารถหาค่าเวลาที่ใช้ในการส่งดาต้าไปยังเครื่องไมโครคอมพิวเตอร์จริงๆ ได้จากสูตร

$$T_{ToHost} = T_{total} + (T_c * N)$$

เมื่อ

T_{ToHost} = เวลาที่ในการส่งดาต้า

T_{total} = เวลาที่วัดได้

T_c = เวลาที่ใช้ในโปรแกรมบริการไทม์เมอร์อินเทอร์รัพท์

N = จำนวนครั้งที่ให้บริการไทม์เมอร์อินเทอร์รัพท์

ดังนั้น

ที่การส่งดาต้า 64 ชุดข้อมูล

เวลาที่ใช้ในส่ง = 6.5 mS – (300nS x 650)

= 6.3 mS

ที่การส่งดาต้า 128 ชุดข้อมูล

เวลาที่ใช้ในส่ง = 12.87 mS – (300nS x 1287)

= 12.48 mS

ที่การส่งดาต้า 256 ชุดข้อมูล

เวลาที่ใช้ในส่ง = 24.65 mS – (300nS x 2465)

= 23.91 mS

ที่การส่งดาต้า 512 ชุดข้อมูล

เวลาที่ใช้ในส่ง = 51.19 mS – (300nS x 5119)

= 49.65 mS

<u>วิจารณ์ผลการทดลอง</u>

การรับส่งข้อมูลอนุกรมถูกจำกัดด้วยความเร็วดังนั้นไม่ควรออกแบบให้มีการรับส่งข้อมูลด้วยวิธีนี้ แต่สำหรับโครงงานนี้แล้วนับว่าไม่เป็นอุปสรรคแต่อย่างใดเป็นเพราะขอบเขตของการพัฒนายังอยู่ในช่วงความเร็วที่สามารถใช้การสื่อสารอนุกรมได้ แต่ในการพัฒนาระบบให้มีความสามารถยิ่งๆขึ้นไปจะต้องออกแบบระบบทางฮาร์ดแวร์และรูปแบบการสื่อสารเสียใหม่

การรับส่งข้อมูลด้วยการสื่อสารอนุกรมแม้ว่าจะมีการกำหนดความเร็วในการรับส่งที่แน่นอนแต่ในทางปฏิบัติจริงๆแล้วเวลาที่ใช้ในการแลกเปลี่ยนข้อมูลจำเป็นจะต้องทำการแฮนด์เช็คกันระหว่างโฮสต์จึงทำให้เวลาที่ใช้ในการส่งข้อมูลแต่ละชุดไม่เท่ากัน แต่จากการทดลองความแตกต่างของการรับส่งข้อมูลแต่ละครั้งมิได้แตกต่างกันไปมากมายนัก ดังนั้นจึงสามารถนำค่าเวลาที่วัดได้มาออกแบบเรียลไทม์สวิทซ์ได้

การทดลองที่3 การทดสอบการวัดแถบความถี่ของรูปคลื่นไซน์โดยใช้ PC Spectrum Analyzer

วัตถุประสงค์ ทดสอบการใช้งานจริงและนำผลการวัดมาเทียบกับมาตรฐาน

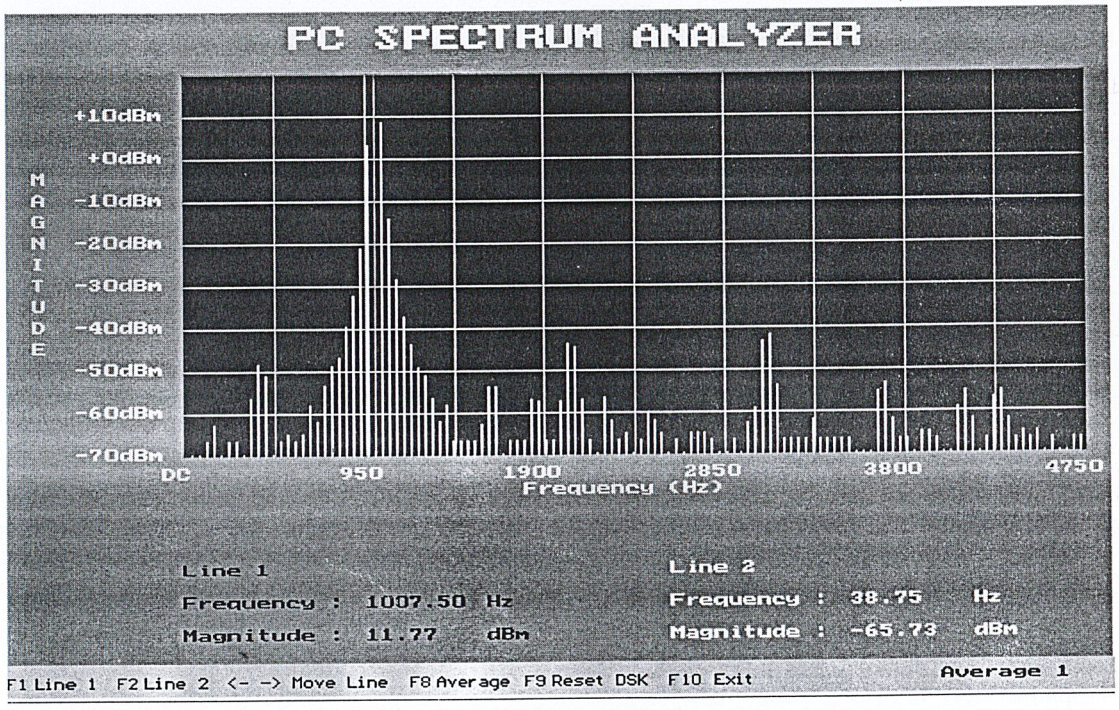
สมมุติฐานการทดลอง

เครื่องวิเคราะห์แถบความถี่ที่พัฒนาขึ้นในโครงงานนี้ควรจะให้ผลการวิเคราะห์ตรงกับการใช้การวิเคราะห์ทางทฤษฎีทางคณิตศาสตร์ หรือ เครื่องวิเคราะห์แถบความถี่มาตรฐาน สำหรับการทดลองนี้ควรจะได้ผลการวิเคราะห์ตรงกับผลการวิเคราะห์โดยใช้โปรแกรม MATLAB

วิธีการทดลอง

1. เชื่อมต่อบอร์ด DSK เข้ากับเครื่อง PC โดยผ่านทางพอร์ทอนุกรม
2. พิมพ์โปรแกรมชื่อ SPEC50.C ในภาคผนวก แล้วคอมไพล์เป็น EXE ไพล์
3. พิมพ์โปรแกรมชื่อ SPEC50.ASM ในภาคผนวก แล้วคอมไพล์เป็น DSK ไฟล์
2. ใช้ฟังก์ชั่นเจนเนอร์เรเตอร์จ่ายสัญญาณไซน์เวฟ 1 KHz 8Vpp เข้ายัง Jack IN ของ DSK Borad
3. สั่ง RUN โปรแกรม โดยเรียก Spec50 –c2 บนเครื่องไมโครคอมพิวเตอร์
4. อ่านค่า-บันทึกผลการทดลอง

ผลการทดลอง



วิเคราะห์ผลการทดลอง

ผลการวัดแถบความถี่ของเครื่อง พีซี สเปกตรัมอนาไลซ์เซอร์ ปรากฏว่าได้ผลคำตอบตรงตามทฤษีนั่นคือความความถี่ฟันดาเมนทอล (1KHzz) มีขนาดทางแรงดันเป็น 11.77 dBm ซึ่งผลจากการคำนวณโดยใช้ทฤษีฎีทางคณิตศาสตร์ตามสูตร 20Log (Vpp ^{2}/($\sqrt{0.6}\sqrt{2}$)) แล้วจะได้คำตอบเป็น 11.722 dBm ซึ่งจากผลการทดลอ งทำให้ทราบว่าขนาดทางแรงดันของส่วนประกอบความถี่ที่วัดได้มีความคลาดเคลื่อนเล็ก

น้อยแต่ อย่างไรก็ตามเครื่องวิเคราะห์แถบความถี่ (พีซี สเปกตรัมอนาไลซ์เซอร์) นี้ก็ยังสามารถใช้ทดแทนเครื่องวิเคราะห์แถบความถี่ที่มีขายอยู่ตามท้องตลาดได้โดยเครื่อง พีซี สเปกตรัมอนาไลซ์เซอร์ที่สร้างขึ้นมานี้สามารถแสดงส่วนประกอบของความถี่ได้ 128 ค่าของความถี่ฮาร์โมนิกส์

<u>วิจารณ์ผลการทดลอง</u>

เครื่องวิเคราะห์แถบความถี่ที่พัฒนาขึ้นในโครงงานนี้แม้มีผลการวิเคราะห์คลาดเคลื่อนกับผลการวิเคราะห์ โดยโปรแกรม MATLAB เล็กน้อยเป็นเพราะว่าในโปรแกรม MATLAB มีการซิมมูเลทสัญญาณอินพุทโดยโปรแกรมดังนั้นจึงได้ข้อมูลที่ตรงตามทฤษฎีซึ่งต่างจากการให้อินพุทจากแหล่งกำเนิดสัญญาณเพราะสัญญาณดังกล่าวจะมีสัญญาณรบกวนต่างๆหรือแม้แต่การสูญเสียพลังงนภายในสายส่ง สิ่งต่างๆเหล่านี้เป็นปัจัยทำให้ค่าของอินพุทจริงๆมีค่าที่ผิดพลาดไปจากค่าจริงซึ่งเป็นเหตุให้ผลการวิเคราะห์คลาดเคลื่อนได้

การทดลองที่4 การทดสอบผลการวัดอ้างอิงกับโปรแกรมวิเคราะห์ MATLAB

วัตถุประสงค์ ทดสอบการใช้งานจริงและนำผลการวัดมาเทียบกับมาตรฐาน

สมมุติฐานการทดลอง

ผลการวิเคราะห์แถบความถี่ของสัญญาณจะต้องตรงกับการวิเคราะห์โดยใช้ทฤษฎีทางคณิตศาสตร์

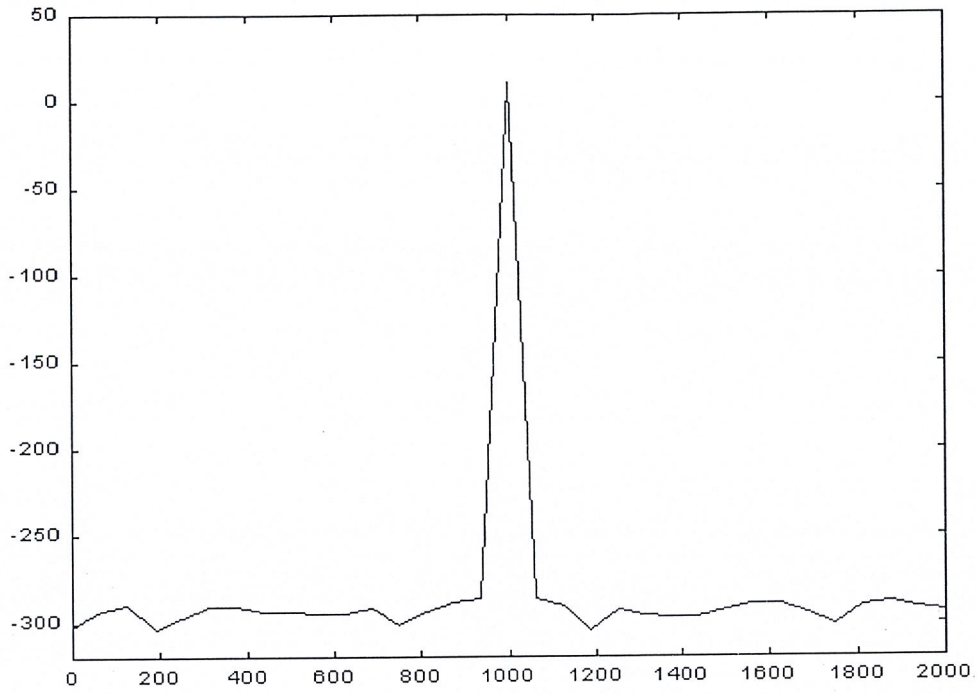
วิธีการทดลอง

1. เชื่อมต่อบอร์ด DSK เข้ากับเครื่อง PC โดยผ่านทางพอร์ทอนุกรม
2. เขียนโปรแกรมสร้างฟังก์ชั่นสัญญาณไซน์เวฟ 1 KHz 8Vppและเขียนโปรแกรมพร็อตกราฟผลการวิเคราะห์โดยโปรแกรม ดังนี้

```
N=128;
T=1/8000;
fc=1000;
K=0:1:N-1;
freq=K*(1/(N*T));
f=8*sin(2*pi*fc*K*T);
x=(1/N)*(fft(f));
x11=abs(x);
x1=20*log10(x11/(sqrt(2)*sqrt(0.6)));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
plot(K,f);
plot(freq(1:N/2),x1(1:N/2));
axis([0 2000 -320 50]);
zoom
```

3. สั่ง RUN โปรแกรมโดย MATLAB
4. อ่านค่า-บันทึกผลการทดลอง

ผลการทดลอง



วิเคราะห์ผลการทดลอง

ผลการวัดให้ค่าที่คลาดเคลื่อนกันเล็กน้อยอาจเนื่องด้วยสัญญาณอินพุทของการวิเคราะห์ทั้งสอง (MATLAB – PC Spectrum Analyzer) แบบนี้อาจไม่เที่ยงตรงกันเพราะเป็นสัญญาณที่เป็นซิมโมเลทโดยโปรแกรมจะเที่ยงตรงกว่าสัญญาณที่กำเนิดโดยเครื่องกำเนิดสัญญาณที่มีการสูญเสียและเกิดสัญญาณรบกวนระหว่างการส่งถ่ายพลังงาน

วิจารณ์ผลการทดลอง

ผลการวิเคราะห์โดยโปรแกรม MATLAB ถือเป็นมาตรฐานอ้างอิงได้ เป็นเพราะโปรแกรม MATLAB เป็นโปรแกรมที่ได้รับการยอมรับและใช้ในงานออกแบบทางวิศวกรรม

บทที่ 5
สรุปและวิจารณ์

5.1 สรุป

เครื่องวิเคราะห์แถบความถี่โดยใช้หน่วยประมวลผล TMS320C50 นี้สร้างเพื่อวิเคราะห์แถบความถี่ โดยระบบประมวลผลสัญญาณเชิงตัวเลข โดยมีขอบเขตเบื้องต้นคือสามารถวิเคราะห์แถบความถี่ในย่านความถี่เสียงได้นั่นคือช่วงความถี่ 0-2 กิโลเฮิร์ท และมีฟังก์ชั่นต่างๆในการแสดงผลที่สะดวกและประโยชน์ต่อการใช้งาน

จากการทดลองและศึกษาและสร้างปรากฏว่าผลที่ได้ยังไม่เป็นที่น่าพอใจด้วยเหตุเพราะอุปกรณ์แปลงสัญญาณอนาล็อคเป็นสัญญาณดิจิตอลที่มีใช้นั้นมีอัตราสุ่มข้อมูลสูงสุดแค่ 19.2 กิโลเฮิร์ท แต่ผลสุดท้ายก็ได้ผลเป็นที่พอใจระดับหนึ่งเพราะว่าในทางปฏิบัติแล้วช่วงความถี่เสียงที่มนุษย์ส่วนใหญ่ได้ยินได้อย่างชัดเจนก็คือช่วง 200Hz - 5KHz เท่านั้น อย่างไรก็ตามเครื่องวิเคราะห์แถบความถี่นี้สามารถใช้งานทดแทนเครื่องที่มีขายตามท้องตลาดที่มีราคาซื้อขายที่สูงลิ่วได้

5.2 ปัญหาที่พบ

ปัญหา - ข้อมูลเกี่ยวกับการประมวลผลสัญญาณเชิงดิจิตอล(DSP) หายากเนื่องตัวชิฟรุ่นใหม่ที่มีความสามารถตามที่ต้องการเป็นชิฟที่ใหม่ ดังนั้นข้อมูล- ด้าต้าชีส หาไม่ได้ตามท้องตลาด

แก้ปัญหา - สืบค้นข้อมูลทางอินเตอร์เน็ตที่เวฟไซน์ http://www.ti.com และ ขอข้อมูลจากทางบริษัท Texas Instruments

ปัญหา - ความยุ่งยากในการอิมพลีเมนต์อัลกอริทึมโปรแกรม FFT

แก้ปัญหา - ดาวน์โหลดโปรแกรม FFT จากทางบริษัท

ปัญหา - สร้างฮาร์ดแวร์สำหรับอิมพลีเมนต์โครงงานยากเนื่องจากเป็นการออกแบบสำหรับใช้ในงานความถี่สูง

แก้ปัญหา - ซื้อชุดพัฒนา DSP Start Kit ที่บริษัทขาย

ปัญหา - การทรานเฟอร์ข้อมูลไปแสดงผลที่เครื่องไมโครคอมพิวเตอร์ถูกจำกัดด้วยความเร็วของการสื่อสารอนุกรม

แก้ปัญหา - ใช้การส่งข้อมูลแบบซิงก์โครนัสโดยจะส่งข้อมูลทีละเฟรม เฟรมละ 128 ชุดข้อมูลจะทำให้ประหยัดเวลาในการส่ง

ปัญหา - โปรแซสเซอร์ TMS320C50 และ DSK Board ไม่มีระบบปฏิบัติการที่คอยจัดการควบคุมโปรแซสการทำงานให้เป็นแบบมัติแทสติ่ง

แก้ปัญหา - ตรวจวัดเวลาที่แต่ละโปรแซสต้องการในการทำงานสำเร็จในแต่ละรอบการทำงานแล้วจัดสรรคเวลาให้แต่ละโปรแซสตามต้องการ

5.3 แนวทางการพัฒนาต่อไป

เครื่องวิเคราะห์แถบความถี่ที่ได้จัดสร้างในโครงงานนี้เป็นเครื่องวัดที่มีประสิทธิภาพเป็นที่น่าพอใจในระดับหนึ่ง ซึ่งสามารถทำงานได้ในช่วงความถี่ 0-4 กิโลเฮิร์ต เป็นเพราะว่าการออกแบบในโครงงานนี้ได้ใช้เครื่องไมโครคอมพิวเตอร์เป็นตัวแสดงผลโดยการทรานเฟอร์ข้อมูลจาก DSK Board มายังเครื่องไมโครคอมพิวเตอร์กระทำผ่านทางพอร์ทสื่อสารอนุกรมจึงโดนจำกัดโดยความเร็วของการรับ-ส่ง และ อีกทั้งความสามารถของตัวแปลงสัญญาณอนาล็อคเป็นสัญญาณดิจิตอลมีอัตราความเร็วในการสุ่มข้อมูลสูงสุดแค่ 19.2 กิโลเฮร์ต จึงทำให้การทำงานจึงอยู่ในช่วงความถี่ที่จำกัด แต่การทำงานจะสามารถขยายช่วงความถี่มากยิ่งขึ้นโดยการเปลี่ยนระบบการสื่อสารมาเป็นแบบขนานและต่ออุปกรณ์ตัวแปลงสัญญาณอนาล็อคเป็นสัญญาณดิจิตอลอินเทอร์เฟสเข้ากับ DSK Board เพื่อขยายอัตราความเร็วในการสุ่มข้อมูล อีกทั้งสามารถสร้างระบบปฏิบัติการไว้คอยจัดการควบคุมการจัดสรรคเวลาให้กับการทำงานของแต่ละโปรแซสเพื่อการใช้เวลาที่มีประสิทธิภาพและเป็นการทำงานที่เป็นแบบเรียลไทม์มากขึ้น

บรรณานุกรม

[1] Brigham, E.O., The Fast Fourier Transform, Englewood Cliffs, NJ; Prentice-Hall, Inc., 1974.

[2] Burrus, C.S., and T.W. Parks, DFT/FFT and Convolution Algorithms, New York, NY: John Wiley and Sons, Inc., 1984

[3] Texas Instruments, Digital Signal Processing Application with the TMS320 Family, 1986; Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987

[4] Texas Instruments, TMS320C5x User's Guide, Texas Instruments, Inc., 1995

[5] Texas Instruments, TMS320C5x DSP Start Kit User's Guide, Texas Instruments, Inc., 1995

[6] http://www.ti.com

[7] ผศ. นิรันดร์ คำประเสริฐ, อนุกรมฟูเรียร์และฟูเรียร์ทรานส์ฟอร์ม, ศูนย์สื่อเสริมกรุงเทพ, 2538

ภาคผนวก

ก. โปรแกรมที่ทำงานบนเครื่องไมโครคอมพิวเตอร์เขียนด้วยภาษาซี

SOURCE CODE FILE SPEC50.C

```
# include "spec50.h"

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

FILE  *stream;       // global for this file

const STRING programme ="Spectrum Analyzer";

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

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,restart = 0,new_line1=0,new_line=0,x_position=5,y_position;
int x_position1 = 5,TrueKey=0;
int TempX,zoom=0,step=4,LineActive=0;
char string[20];
clock_t  duration;
char sa_kernel_out[]={"spec50.dsk"};
//---------------------------------------------------------------------
// Main program
//---------------------------------------------------------------------
void main(void)
{
 char *ptr1, *ptr2, *ptr3, *ptr4;
 char *ptr5, *ptr6, *ptr7, *ptr8, *ptr0;
 char *tmp0, *tmp1, *tmp2, *tmp3, *tmp4;
 char *tmp5, *tmp6, *tmp7, *tmp8, *pk_t;
 int  x, y, old_y, xx, *y_pk;
 int TA;
 int two_block_length,block_length,address;
 int k,c, number_overrun,BRD;
 float SCF, FDAC;
 prm.com = 1;
 clrscr();
 prm.EntryPoint=prm.PGM_CNT=0x0a00;  // standard entry point
 prm.INVERSE  =YES;            // for standard boards like DSK
re_start:
 prm.speed=57600;              // default !
 InitializeMonitor();  // Establish communication at 57600 baud using old kernel
//---------------------------------------------------------------------
// Load Spectrum Analyzer Kernel
//---------------------------------------------------------------------
 printf("\nDownloading Spectrum Analyzer Program...\n");
 stream=fopen(sa_kernel_out,"rb");
 duration = LoadDsk();
 ExePgm();
//---------------------------------------------------------------------
```

```
// Reinitialize PC serial port to 115200 baud
//-----------------------------------------------------------------
 prm.speed=115200l;
 InitPort();
 init_graphics();
//-----------------------------------------------------------------
// This module test the transfering of data without handshaking
//-----------------------------------------------------------------
 two_block_length=128;
 tmp0 = buf_0;
 tmp1 = buf_1; tmp2 = buf_2; tmp3 = buf_3; tmp4 = buf_4;
 tmp5 = buf_5; tmp6 = buf_6; tmp7 = buf_7; tmp8 = buf_8;
 ptr0 = tmp0;
 for(x=0;x<256;x+=2) *ptr0++ = 128;  // init 1st line
 for(x=0;x<512;x++){
   buf_y_pk[x] = 0;
   buf_pk_t[x] = NULL;
 }
 for(      ;      ;    ) {
    ptr0 = tmp0;                    // Set buffer data pointers
    ptr1 = tmp1; ptr2 = tmp2; ptr3 = tmp3; ptr4 = tmp4;
    ptr5 = tmp5; ptr6 = tmp6; ptr7 = tmp7; ptr8 = tmp8;
    y_pk = buf_y_pk;
    pk_t = buf_pk_t;
    for(x=0;x<512;x+=step/*4*/) {
            old_y = 128 - *ptr0; //Result = 256
            switch(avg_on) {
              case  1: *ptr0 = (*ptr1+*ptr2) >> 1;   // avg of 2
                       break;
              case  2: *ptr0 = (*ptr1+*ptr2+*ptr3+*ptr4) >> 2; // avg of 4
                       break;
              case  3: *ptr0 = (*ptr1+*ptr2+*ptr3+*ptr4+      // avg of 8
                                *ptr5+*ptr6+*ptr7+*ptr8) >> 3;
                       break;
              default: *ptr0 = *ptr1;
                       break;                    // avg of 1
            }
            y = 128 - *ptr0;
            *pk_t += 1;
            if((*pk_t > 8) || (y < *y_pk)) {
              if(y < *y_pk) {
```

```
                    *y_pk = y;
                    *pk_t = 0;
                }
                else
                    *y_pk += 6;
                setcolor(WHITE);
            }
            y_pk++;
            pk_t++;
//--------------------------
            if(y > old_y){
              old_y++;//setcolor(RED);
              line(x+5,old_y,x+5,y);
            }
            if(y < old_y){
              y++;//setcolor(BLUE);
              line(x+5,old_y,x+5,y);
            }
            if(new_line>0) //Modify Draw Line 2 for Analysis
               {
                    setcolor(14);  //Set color Yellow
                    line(x_position,0,x_position,270);
                    setcolor(WHITE);
               }
            if(new_line1>0) //Modify Draw Line 1 for Analysis
               {
                    setcolor(12);  //Set color Ligth Red
                    line(x_position1,0,x_position1,270);
                    setcolor(WHITE);
               }
            ptr0++;
            ptr1++; ptr2++; ptr3++; ptr4++;          // next data
            ptr5++; ptr6++; ptr7++; ptr8++;
        }

        if(kbhit()) check_key();
        if(restart){
            avg_on=restart=0;
            closegraph();
            goto re_start;
        }
```

```
    ptr1 = tmp8;                // Rotate the buffer pointers
    tmp8 = tmp7; tmp7 = tmp6; tmp6 = tmp5; tmp5 = tmp4;
    tmp4 = tmp3; tmp3 = tmp2; tmp2 = tmp1; tmp1 = ptr1;

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

// Turn off interrupt and reading the data from DSP
    asm cli;
    WaitFor(XMT_BUF_EMPTY);
    outport(pcom,NULL);
    for(k=0;k<two_block_length;k++)
            *ptr1++=Bcrcv(pcom);
    asm sti;
  }
}

void check_key(void)
{
  int key;
  key = (bioskey(0)) >> 8;
  switch(key)                   // Key traps for various routines
  {
    case 66   : avg_on += 1;  //F8 (A)veraging select
                        if(avg_on >3)
                          avg_on = 0;
                        TrueKey = 1;
                        break;
```

```
case 59   : new_line1 +=1; //F1
                    LineActive=1;
                    TrueKey=1;
                    break;
case 60   : new_line +=1;  //F2
                    LineActive=2;
                    TrueKey=1;
                    break;
case 0x4D : TrueKey=1;
                    switch (LineActive)
                     {
                      case 2 : x_position +=step/*4*/;   // Right Arrow
                                  if(x_position>500)
                                  x_position=5;
                                  break;
                      case 1 : x_position1 +=step;
                                  if(x_position1>500)
                                 x_position1=5;
                                 break;
                     }
                    break;
case 0x4B : TrueKey=1;
                   switch (LineActive)
                    {
                     case 2 : x_position -=step/*4*/;   // Left Arrow
                                 if(x_position<5)
                                 x_position=5;
                                 break;
                     case 1 : x_position1 -=step;
                                 if(x_position1<5)
                                 x_position1=5;
                                 break;
                    }
                    break;
case 0x43 : restart = 1;break;  // F9 (R)eboot code
case 0x44 : closegraph();       // (Q)uit to DOS
            exit(0);       break;
default   : TrueKey=0;     break;
}
if(TrueKey)
 {
```

```
    setup_vals();                    // redraw the setup values
    setviewport(100,40,600,274,1);   // Set window to show display
    setwritemode(1);                 // display lines are XOR drawn
    setcolor(15);                    // light gray
   }
}
void InitializeMonitor(void) /*altered on 03/01/95*/
{
 UINT num;
 UINT error;
 int pass=FALSE;
 InitPort(); //initialise serial com regiaters.
 pass=FALSE;
 BaudRateDetect(); //send 0x80 to DSP to calculate BITLEN.
 delay(2);  // what is the right time ??? was 1 up to version 0.98
 switch(error=InitMonitor())
 {
          case 0:
          case 1:
          case 2:  pass=FALSE;
                   break;
          case 3:  pass=TRUE;
                   break;
 }
 if(!pass){
   clrscr();
   printf("\nCommunication not successful\n");
   switch(error) {
    case 0: printf("\nNo Response.\n");
            break;
    case 1: printf("\nTest Error.\n");
            break;
    case 2: printf("\nNo Escape.\n");
            break;
    default: break;
   }
   exit(1);
 }
}

void InitPort(void)
```

```
{
 UINT  COMADD[]={0x3f8, 0x2f8, 0x3e8, 0x2e8};
 int  BRD=115200l/prm.speed;
 pcom = COMADD[prm.com];
 UINT port_no=pcom;
 asm  mov  DX,word ptr port_no  //
 asm  add  DX,3              //
 asm  mov  AL,087h            // SET BAUD access
 asm  out  DX,AL             //
 asm  sub  DX,3              //
 asm  mov  AX,word ptr BRD  //LO// Set 19200 baud
 asm  out  DX,AL             //
 asm  add  DX,1              //
 asm  mov  AL,byte ptr BRD  //HI//
 asm  xchg AL,AH              //
 asm  out  DX,AL             //
 asm  sub  DX,1              //
 asm  add  DX,3              //
 asm  mov  AL,07h             // CLR BAUD access...  N-8-2
 asm  out  DX,AL             //
 asm  out  DX,AL             // N-8-2
 asm  mov  DX,port_no         //0x3FC 0x2FC  modem control register
 asm  add  DX,5              //check LINE_STATUS  0x2FD  0x3FD (+5)
xempty0:                    //
 asm  in   AL,DX             //
 asm  and  AL,060h            //wait for DXR & TXR to empty before RST!
 asm  cmp  AL,060h            //
 asm  jne  xempty0            //
}
//---------------------------------------------------------------------
UINT InitMonitor(void)
{
 int  i=10000;
 int  ans=inportb(pcom+LSR);
 int  byte_return;
 while(!(ans & DATA_READY) && i){  // data received?
            ans = inportb(pcom+LSR);
            i--;
 }
 if(i<1)
    return  0;
```

```
  if( inportb(pcom)!=KB_ESC)
    return 2;

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

//---------------------------------------------------------------------
void reset50()
{
 UINT  port_no=pcom;
 if(prm.INVERSE) {          // do the inverse DTR for reset of c50
   asm  mov  DX,word ptr port_no//modem control register  02FC 03FC
   asm  add  DX,4           //
   asm  mov  AL,0xB         //RTS=1, DTR=0
   asm  out  DX,AL
   delay(1);
   asm  mov  DX,word ptr port_no//modem control register
   asm  add  DX,4           //
   asm  mov  AL,0xA         //RTS=1, DTR=0
   asm  out  DX,AL
   delay(1);
   asm  mov  DX,word ptr port_no//modem control register
   asm  add  DX,4           //
   asm  mov  AL,0xB         //RTS=1, DTR=0
   asm  out  DX,AL
```

```
    delay(1);
  }
  else {                    // else we have a hardware negate function from PC to c50
    asm  mov  DX,word ptr port_no//modem control register
    asm  add  DX,4          //
    asm  mov  AL,0xA        //RTS=1, DTR=0
    asm  out  DX,AL
    delay(1);
    asm  mov  DX,word ptr port_no//modem control register
    asm  add  DX,4          //
    asm  mov  AL,0xB        //RTS=1, DTR=0
    asm  out  DX,AL
    delay(1);
    asm  mov  DX,word ptr port_no//modem control register
    asm  add  DX,4          //
    asm  mov  AL,0xA        //RTS=1, DTR=0
    asm  out  DX,AL
    delay(1);
  }
}
//---------------------------------------------------------------------
void BaudRateDetect()
{
 reset50();
 delay(12);       // v.02 ROM code need delay for boot loader

 if(prm.speed<57600){
   while(!(inportb(pcom+LSR) & XMT_BUF_EMPTY));
   outportb(pcom,0x80);   // write a byte to the com port (320c50)
 }
 else
   outportb(pcom,0x80);   // write a byte to the com port (320c50)
}
void WaitFor(UINT what)
{
 int j=10000,i=j;
 while(!(inportb(pcom+LSR) & what) && i--);
 if(i<1){
   fprintf(stdout,"Handshake error"
           "\r\nWait loop finished in WaitFor() with start=%d",j);
   exit(1);
```

```
}
else return;
}

int screen(UINT x,UINT y,char *fmt,...)
{
 const MAXSCREENLEN=700;
 static char buffer[MAXSCREENLEN];
 va_list argptr;
 va_start(argptr,fmt);
 int cnt=vsprintf(buffer,fmt,argptr);
 va_end(argptr);
 gotoxy(x,y);
 cprintf(buffer);
 clreol();
 return cnt;
}

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

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

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

UINT sendword(UINT send)
{
 UINT  s1;

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

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

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

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

return receive; // return sent word for check if necessary
}
//---------------------------------------------------------------------
//--- Load a program/data file in TI-Dsk Format: default ext. dsk   --
//---------------------------------------------------------------------
double LoadDsk(void)
{
 char   linebuf[MAXLINE], *pbuf=linebuf;
 UINT   line=0;       // line counter of file
 UINT   data[MAXLINE]; // max 100. words to download
 UINT   numdata=0;    // num of words to download
 ULONG  sumdata=0;    // sum of all downloaded words
 UINT   address;      // address for    download
 BOOLE  program;      // type: if program 'yes' , no if data 'no'
 clock_t start=clock();
 while(1) {                          // for all lines
    pbuf=fgets(linebuf,MAXLINE,stream);
    switch(*pbuf){                    // analyse line
          case  ':':
          case NULL: fclose(stream);
                     fprintf(stdout,"\nLoading of prog/data finished");
                     double d=clock()-start;
                     return d>0 ? d : 0.001;
          case  'K': numdata=0;break;          // Do something with the name ?
          case  '9': address=GetDskAdd(pbuf+1); // DSK for an address
                     switch(*(pbuf+5))
```

```
                    {
                     case '7': break;       // Do nothing with checksum  ?
                     case 'M': program=NO; break;
                     case 'B': program=YES; break;
                     default : fprintf(stdout,"\n\nerror 1: 'Wrong Tag "
                                                "in Pos 6'!\n\n\rPlease check whether"
                                               " the file is a valid *.dsk file!\n\n\r");
                                              return 0;
                     }
                    numdata=GetDskData(pbuf+5,data);
                    sumdata +=numdata;
                    break;
          case  '1': address=GetDskAdd(pbuf+1);         // Entry address
                    if(address==0)
                     break; // dska generates address=0 if no entry defined!

                    prm.PGM_CNT=prm.EntryPoint=address;
                    break;
          default  : fprintf(stdout,"\n\nerror 2: 'Wrong Tag in Pos 1' in LoadDsk()"
                                               "!\n\n\rPlease check whether"
                                               " the file is a valid *.dsk file!\n\n\r");
                    return 0;
     }
   if(numdata){                          // download line
          UINT i,cmd=(program==YES) ? LP:LD;

          fprintf(stdout,"\r   %4d %5d %8ld  %s",
                        ++line,numdata,sumdata,program ? "program":"data");
          sendcommand(cmd,address,numdata);
          for(i=0;i<numdata;i++) sendword(data[i]);
   }
 }
}

//---------------------------------------------------------------------
UINT GetDskAdd(STRING pbuf)
{
 UINT x=0,i=4;
 char c=*pbuf++;
 for(;i;i--,c=*pbuf++)
  {
```

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

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

void ExePgm(void)
{
 sendbyte(XG); sendword(prm.PGM_CNT);   // execute
 outportb(pcom,NULL);            // Startpulse / sync for windows/OS2 ??????
 return;
}
void setup_vals(void)
{
 setwritemode(0);                // turn off XOR write
 setviewport(0,300,200,309,1);       // clear window
```

```
clearviewport();                    //
//--------Modify----------
setviewport(0,getmaxy()-10,getmaxx(),getmaxy(),1);
setcolor(WHITE);
line(0,getmaxy()-10,getmaxx(),getmaxy());
//------------------------
setviewport(100,0,620,30,1);        // Write the title in yellow
setcolor(14);
settextstyle(0,0,2);
outtextxy(60,10,"PC SPECTRUM ANALYZER");
setviewport(0,30,620,300,1);// Write to window in bright blue 360
setcolor(11);                       //
settextstyle(0,0,0);
outtextxy(10,33, "    +10dBm");
outtextxy(10,59, "    +0dBm");
outtextxy(10,72, " M      ");
outtextxy(10,85, " A  -10dBm");
outtextxy(10,98, " G      ");
outtextxy(10,111," N  -20dBm");
outtextxy(10,124," I      ");
outtextxy(10,137," T  -30dBm");
outtextxy(10,150," U      ");
outtextxy(10,163," D  -40dBm");
outtextxy(10,176," E      ");
outtextxy(10,189,"    -50dBm");
outtextxy(10,215,"    -60dBm");
outtextxy(10,241,"    -70dBm");
outtextxy(96,250,"DC");
outtextxy(185,250," 950");
outtextxy(285,250,"1900");
outtextxy(385,250,"2850");
outtextxy(485,250,"3800");
outtextxy(585,250,"4750");
outtextxy(224,260,"       Frequency (Hz)       ");
outtextxy(100,290,"(Q)uit to DOS   ");  //
setviewport(500,460,639,479,1);
setfillstyle(SOLID_FILL,LIGHTGRAY);
bar(0,0,150,20);
setcolor(BLUE);
switch(avg_on)
{
```

```
 case  1: outtextxy(20,5,"Average 2");break; //
 case  2: outtextxy(20,5,"Average 4");break; //
 case  3: outtextxy(20,5,"Average 8");break; //
 default: outtextxy(20,5,"Average 1");break; //
}
setviewport(200,330,285,440,1);
setfillstyle(SOLID_FILL,BLACK);
bar(0,0,65,100);
setcolor(RED);
//--------------------------------
char *Dot = ".",*Zero="0",st1[20],st2[20];
float Hz=(x_position1/step)*38.75;
int temp1 = Hz;
float temp2 = (Hz-temp1)*100;
itoa(Hz,st1,10);
itoa(temp2,st2,10);
strcpy(string,st1);
strcat(string,Dot);
if (temp2<10) strcat(string,Zero);
strcat(string,st2);
outtextxy(0,30,string);
//--------------------------------
int dbtemp;
float db=buf_0[(x_position1/step)-1];
float dbp,TempdB;
dbp=(db-77)*0.38;
dbtemp=dbp;
TempdB=(dbp-dbtemp)*100;
itoa(dbtemp,st1,10);
if (TempdB<0) TempdB = TempdB*(-1);
itoa(TempdB,st2,10);
strcpy(string,st1);
strcat(string,Dot);
if (TempdB<10) strcat(string,Zero);
strcat(string,st2);
 outtextxy(0,50,string);
//---------------------
setviewport(470,330,555,440,1);
setfillstyle(SOLID_FILL,BLACK);
bar(0,0,65,100);
setcolor(YELLOW);
```

```
//----------------------------
float Hz1=(x_position/step)*38.75;
int tmp1 = Hz1;
float tmp2 = (Hz1-tmp1)*100;
itoa(Hz1,st1,10);
itoa(tmp2,st2,10);
strcpy(string,st1);
strcat(string,Dot);
if (tmp2<10) strcat(string,Zero);
strcat(string,st2);
//----------------------------
outtextxy(0,30,string);
//**********************
int dbtemp1;
float db1=buf_0[(x_position/step)-1];
float dbp1,TempdB1;
dbp1=(db1-77)*0.38;
dbtemp1=dbp1;
TempdB1=(dbp1-dbtemp1)*100;
itoa(dbtemp1,st1,10);
if (TempdB1<0) TempdB1 = TempdB1*(-1);
itoa(TempdB1,st2,10);
strcpy(string,st1);
strcat(string,Dot);
if (TempdB1<10) strcat(string,Zero);
strcat(string,st2);
//**********************
outtextxy(0,50,string);
}
void init_graphics(void)
{
 int gdriver = VGA/*EGA*/, gmode = VGAHI, errorcode, Y, X;
 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();                    //
 //-----------------------------
 settextstyle(0,0,0);
 setcolor(RED);
 outtextxy(100,340,"Line 1");
 outtextxy(100,360,"Frequency :        Hz");
 outtextxy(100,380,"Magnitude :        dBm");
 setcolor(YELLOW);
 outtextxy(370,340,"Line 2");
 outtextxy(370,360,"Frequency :        Hz");
 outtextxy(370,380,"Magnitude :        dBm");
 setfillstyle(SOLID_FILL,LIGHTGRAY);
 bar(0,460,639,479);
 settextstyle(2,0,0);
 setcolor(RED);outtextxy(5,465,"F1");
 setcolor(BLACK);outtextxy(20,465,"Line 1");
 setcolor(RED);outtextxy(65,465,"F2");
 setcolor(BLACK);outtextxy(80,465,"Line 2");
 setcolor(RED);outtextxy(125,465,"\x3c\x2d \x2d\x3e"); //Left & Right Arrow
 setcolor(BLACK);outtextxy(160,465,"Move Line");
 setcolor(RED);outtextxy(225,465,"F8");
 setcolor(BLACK);outtextxy(240,465,"Average");
 setcolor(RED);outtextxy(290,465,"F9");
 setcolor(BLACK);outtextxy(305,465,"Reset DSK");
 setcolor(RED);outtextxy(370,465,"F10");
 setcolor(BLACK);outtextxy(395,465,"Exit");
 //-------------------------------
 setup_vals();                       // dispaly the setup
 setviewport(100,40,600,274,1);      // Set window to show display
 setcolor(2);                        // reticle is green
 for(Y=0;Y<=234;Y+=26) line(0,Y,500,Y);  // draw reticle
 for(X=0;X<=500;X+=50) line(X,0,X,234);  //
  setwritemode(1);                   // display lines are XOR drawn
 setcolor(15);                       // light gray
}

unsigned char Bcrcv(int port_no)
{
 unsigned char c;
 asm  mov  DX,word ptr port_no // RECV a character
 asm  add  DX,5              // check LINE_STATUS  0x2FD  0x3FD (+5)
```

```
  asm  mov  BX,4000          // 836 loops is marginal at 4800 baud.
chkst:                     // with 33MHz 486
  asm  sub  bx,1             //
  asm  jz   timerr           //
  asm  in   AL,DX            //
  asm  and  AL,061h          //
  asm  cmp  AL,061h          //
  asm  jne  chkst            // wait for DRR & RSR to fill
  asm  sub  DX,5             //
  asm  in   AL,DX            //
  asm  mov  byte ptr c,AL    //
  return(c);                 //
timerr:                    //
  return(-1);                //
}
```

ข. โปรแกรมที่ทำงานบนชุดพัฒนา DSP Starter Kit เขียนด้วยภาษาแอสแซมบลี้ของโปรเซสเซอร์ตระกูล TMS320

SOURCE CODE FILE "SPEC50.ASM"

```
;-------------------------------------------------------;
BITLEN2   .set   54h
BITLEN    .set   0a8h
;=================================
   .ds 0x60                        ; AIC/FFT Setup data in upper 1/2 B2
;=========================================================================
TA    .word  12                                ;control sampling rate for AIC
TB    .word  42
;*******************************
AIC_CMD          .word  0x0003 ; 0x72  0000 0000 0000 0011=0x0003
SIG_DEL          .word  0x0000 ; 0x73
STAT0            .word  0x0000 ;
STAT1            .word  0x0000 ; 0x74
ACCU_lo          .word  0x0000 ; 0x75
ACCU_hi          .word  0x0000 ; 0x76
REAL             .word  0x0000 ; 0x77
IMAG             .word  0x0000 ; 0x78
FFT_S            .word  256    ;
FFT_S-1          .word  255    ;
FFT_S/2          .word  128    ;
FFT_S/2-1        .word  127    ;
;-------------------------------------------------------------
;   SECONDARY VECTOR TABLE LOCATED IN B0 PROGRAM RAM
;-------------------------------------------------------------
     .mmregs          ;
_D_base+128   .set   0x1080
     .ps   080ah      ;
     B     RINT       ;RINT  Branch to receive interrupt routine
     B     XINT       ;
;-------------------------------------------------------------
;   APPLICATION CODE IS LOCATED ABOVE DSKD KERNAL
;-------------------------------------------------------------
     .ps   0x0a00     ;
     .entry           ;
start:                ;
     SETC             INTM
     SETC             SXM
     SETC             OVM                ; catch accumulator overflows
     LDP              #0          ; All direct addressing is to MMRs and B2
     SPLK             #0830h,PMST
     LACC             #0
     SAMM             CWSR               ; Setsoftware wait stat eto 0
     SAMM             PDWSR              ;
     SPLK             #020h,IMR          ; Using XINT syn TX & RX
                                 ; initialize AIC and enable interrupts
AIC_RS SPLK           #01h,PRD           ; To generate  10MHz from Tout
```

```
    SPLK    #20h,TCR        ; for AIC maste rclock
    MAR     *,AR0
    LACC    #0008h  ; Non continuous mode
    SACL    SPC     ; FSX as input
    LACC    #00c8h  ; 16 bit words
    SACL    SPC
    LACC    #080h   ;
    SACH    DXR
    SACL    GREG
    LAR     AR0,#0FFFFh     ; Reset AIC by pulsing BR\pin
    RPT     #500    ; and taking it high after 500 cycles
    LACC    *,0,AR0         ; (.25ms at 50ns )
    SACH    GREG
    SETC    SXM     ;
;------------------------------
    LACC    AIC_CMD         ; AIC control
    CALL    AIC_2nd         ; call loader
    ;----------------------
    LACC    TB,2    ; AIC timing B regs
    ADD     TB,9    ;
    ADD     #2      ;
    CALL    AIC_2nd         ; call loader
    ;----------------------
    LACC    TA,2    ; AIC timing A regs
    ADD     TA,9    ;
    CALL    AIC_2nd         ; call loader
    ;----------------------
    LDP     #0
    LACC    #010h   ; AIC RINT
    SACL    IMR     ; where INT0 indicates EOC (End Of Conv)
;*****************************************************************************
    LAR     AR6,#_B_base1   ; _D_base = A1[n],A2[n],A1[n+1],A2[n+1],...
    LAR     AR7,#255        ;
    MAR     *,AR7   ; enable buffer fill
    CLRC    INTM            ; Fill FFT array with 512 samples
FILL  BANZ  FILL,*,AR7      ;wait for buffer full
MAIN:  CALL Init_S  ; Load 512 alternating data points
    LAR     AR0,FFT_S       ; Sigma FFT
    CALL    FFT     ;
    CALL    TO_HOST         ;
    B       MAIN
    ;----------------------

;*****************************************************************************
Init_S:
    MAR     *,AR7   ; enable buffer fill
    CLRC    INTM            ; Fill FFT array with 512 samples
FILL2  BANZ FILL2,*,AR7     ;wait for buffer full
    LAR     AR7,#255
    LAR     AR6,#_B_base1
    CALL    Init_D
```

```
    RET                         ;
;*************************************************************************
Init_D: LAR         AR0,#_D_base        ;
    LAR             AR1,#_B_base1       ; Flip Flop data buffer and FFT array
    LAR             AR2,#255            ;
    MAR             *,AR1       ;
FILLD  LACL         *,AR0       ; get x[n+256]
    SACL            *+          ; REAL=X[n]
    SACH            *+,0,AR1            ; IMAG=0
    MAR             *+,AR2
    BANZ            FILLD,*-,AR1        ;
    RET                         ;
;*************************************************************************
TO_HOST:
    LACC            #80h
    CALL            BCXMIT              ; Send frame synch (START)
    CALL            BRECV               ; Wait for host start word (01Bh)
    SUB             #01Bh
    NOP                         ; Insert NOPs to ensure condition was ready
    NOP                         ; while execute BCND
    BCND            TO_HOST,NEQ
    LAR             AR3,#_D_base        ;AR3= FFT data array
    LAR             AR5,FFT_S/2-1       ;AR5= loop counter
    MAR             *,AR3       ;
    CLRC            INTM
    IDLE                        ;TimeSync mark for RINT
more2  LACC         *+          ;scale log magnitude for host
    MAR             *+          ;
    RPT             #7          ;
    SFR                         ;
    SETC            INTM
    CALL            XMIT                ;send data 8 bit
    CLRC            INTM
    LAR             AR1,#BITLEN
    MAR             *,AR1
S_BIT1: BANZ        S_BIT1,*-,AR1       ;send 2 STOP BIT
    LAR             AR1,#BITLEN
    MAR             *,AR1
S_BIT2: BANZ        S_BIT2,*-,AR1

    MAR             *,AR5       ;
    BANZ            more2,*-,AR3        ;
    RET
;*************************************************************************
FFT:   MAR          *,AR0               ; Calculate FFT parameters from AR0 = Size
    SAR             AR0,FFT_S           ;
    MAR             *-          ;
    SAR             AR0,FFT_S-1         ;
    MAR             *+          ;
    MAR             *BR0+               ; AR0 = AR0 >> 1;
    SAR             AR0,FFT_S/2         ;
```

```
        MAR     *-
        SAR     AR0,FFT_S/2-1           ;
        ;----------------------
        LAR     AR0,FFT_S/2             ;
new_stg LAR     AR1,#_D_base            ; AR1 is the TOP BFLY address
        LAR     AR2,#_D_base            ; AR2 is the BOT BFLY address
        LAR     AR3,#_T_base+1  ; AR3 is the TWiddle pointer
        LAR     AR4,FFT_S/2             ; AR4 counts DFT blocks
        B       n_DFT2,*,AR1            ;
DFT:    MAR     *BR0+,AR5               ; complete circular buffer for TW's
        LAR     AR5,#1                  ; set up DFT loop with *BR0+/BANZ
        MAR     *BR0+,AR1               ; using 1 cuts *BR0+ loop in half!
        ;---------------------------------------
        ; AR1=Top AR2=Bottom AR3=Twiddle
        ;---------------------------------------
BFLY:   LACC    *,14,AR2                ;(imag1+imag2)/4
        ADD     *,14,AR1                ;
        SACH    *+,1,AR2                ;store TOP imag
        SUB     *,15            ;(imag1-imag2)/2
        SACH    *+,1,AR1                ;store BOT imag
        LACC    *,14,AR2                ;(real1+real2)/4
        ADD     *,14,AR1                ;
        SACH    *+,1,AR2                ;store TOP real
        SUB     *,15            ;(real1-real2)/2
        SACH    *,1,AR5                 ;store BOT real
        BANZ    OK,*BR0+,AR3    ;If at DFT end quit early
        ;----------------------
        MAR     *+,AR2                  ;clean up TW base (xxx0000+1)
        MAR     *+              ;modify BOTom DATA pointer
        MAR     *0+             ;
        MAR     *0+,AR1                 ;
n_DFT2: MAR     *0+             ;modify the TOP pointer
        MAR     *0+,AR4                 ;
        BANZ    DFT,*0-,AR3             ;dec DFT block count AR4 by OFFset
        MAR     *,AR0                   ;
        MAR     *BR0+       ;
        BANZ    new_stg,*               ;if OFFset was 1, now cleared
        B       endFFT                  ;
        ;----------------------
OK      LT      *-,AR2                  ;TREG=TWR    *NOTE* Twiddles are Q15
        MPY     *-              ;PREG=REAL*TWR
        LTP     *+,AR3                  ;TREG=IMAG    ACCU=REAL*TWR
        MPY     *               ;PREG=IMAG*TWI          AR2=R AR3=I
        LTS     *+,AR2                  ;TREG=TWI    ACCU=REAL*TWR-IMAG*TWI
        MPY     *               ;PREG=REAL*TWI
        SACH    *-,1,AR2        ;<<;
        LTP     *,AR3           ;TREG=IMAG    ACCU=REAL*TWI
        MPY     *BR0+,AR2               ;PREG=IMAG*TWR
        APAC                    ;       ACCU=IMAG*TWR+REAL*TWI
        SACH    *+,1,AR2        ;<<;
        B       BFLY,*+,AR1             ;
```

```
    ;-------------------------------------------------------------
endFFT: MAR     *,AR2       ;Transform REAL & IMAG to log magnitude
    LAR         AR2,#_D_base        ;AR3=FFT data pointer
    LAR         AR3,FFT_S-1         ;AR5=FFT loop counter
    LAR         AR0,FFT_S
    ;-------------------------------------------------------------;
    ; WINDOW: Performs post FFT raised cosine windowing!      ;
    ; This is done by using the frequency coefficients of the  ;
    ; window in a convolution filter of the spectrum.         ;
    ;-------------------------------------------------------------;
more_MAG
    MAR         *BR0-       ; -IMAG[-1] 1-COS(nwt/N)    + 1
    LACC        *BR0+,15            ; IMAG[-0] filter by post   |
    SUB         *BR0+,16            ; +IMAG[+1] convolution  <--++++-->
    ADD         *BR0-,15            ; IMAG              + + -.5
    SACH        IMAG        ;
    MAR         *+          ; REAL
    MAR         *BR0-       ; -REAL[-1]
    LACC        *BR0+,15            ; REAL[-0] X[-1] -2*X[0] + X[1]
    SUB         *BR0+,16            ; +REAL[+1]
    ADD         *BR0-,15,AR1        ; REAL
    SACH        REAL        ;
    SQRA        IMAG                ;IMAG & REAL can be at most 0x7fff Q15
    LTP         REAL                ;MPY will result (at most) in max positive
    MPY         REAL                ;
    APAC                    ;output is positive Q30
    ADD         #0x1        ;Set up a floor value; log(0) not legal!
    LAR         AR1,#17             ;pre-scaling exponent shifts Y axis
    RPT         #31         ;
    NORM        *-          ;
    NOP
    NOP
    MAR         *,AR2       ;
    MAR         *BR0-       ;-REAL;dump log(f) into oldest REAL (odd addr)
    SACH        *,2         ;clr explicit 1.0 and sign bit from mant
    LACL        *           ;load into ACCU_lo
    SAR         AR1,*       ;then append exponent (AR1)
    ADD         *,16        ;
    RPT         #10         ;jam result into ACCU_hi
    SFL                     ;If needed, Use ADDH to saturate overflow
    SACH        *           ;
    LACC        *           ;
    AND         #0xfffc,0   ;
    SACL        *BR0+               ; REAL
    MAR         *-          ; IMAG
    MAR         *BR0+,AR3           ;+IMAG
    BANZ        more_MAG,*-,AR2 ;keep going until all done
BITREV: LAR     AR0,FFT_S           ;Now perform Output bit reversal
    LAR         AR1,#_D_base        ;by moving the magnitude, which
    LAR         AR2,#_D_base+1  ;is in the REAL slots, into the
    LAR         AR3,FFT_S-1         ;IMAG slots of the FFT data array
```

```
more_BR LACC    *+          ;load the magnitude
    MAR         *+,AR1      ;
    SACL        *BR0+,0,AR3 ;move it to an open IMAG slot
    BANZ        more_BR,*-,AR2  ;more data to move?
RET             ; FFT is finished!!
*****************************************************************
AIC_2nd SACH    DXR
    CLRC        INTM
    IDLE
    ADD         #6,15       ;set ACCU_hi = 3 for secondary XMIT
    SACH        DXR         ;
    IDLE                    ;ACCU_hi requests 2nd XMIT
    SACL        DXR         ;
    IDLE                    ;ACCU_lo sets up registers
    SACL        DXR,2               ;close command with LSB = 00
    IDLE
    RET                     ;
*****************************************************************
; Transmitt a character
*****************************************************************
;===============================================================================
; Trasmit word routine
;===============================================================================
BCXMIT  BCND    $+4,bio         ; expect a sync (0) signal from PC
    B       $-2
;-------------------------------------------------------------------------
; xmtbyte follows now       ; send lowbyte and return to calling pgm
;  'xmtbyte' requires ARP->AR0 and sets ACCU='mask for byte'
;  send startbit (0) +  databyte + stopbits (2)
;-------------------------------------------------------------------------
XMIT    CLRC        c       ; startbit=0
    LAR         ar1,#8      ; counter: 1 startbit+ 8 databits (+ 2 stopbits)
nextbit1 BCND       snd0,nc     ; if c=1 send 1 else send 0
snd1    SETC        xf      ; send one
    B           snd
snd0    CLRC        xf
snd     RPT         #BITLEN             ; send one bit
    MAR         *,ar1
    ROR                     ; lsb(accu) -> carrybit
    BANZ        nextbit1,*-         ; repeat for entire word (9 bits)
    SETC        xf
    RET
;===============================================================================
; Read Routine
;===============================================================================
BRECV:
wait    BCNDD       STOK,bio            ;wait for start bit
    LAR         AR1,#7
    LACL        #0
    B           wait    ;
STOK    RPT         #BITLEN2            ;BITLEN is scaled and
```

```
        NOP
        MAR     *,AR1       ;number of bits - 1
WTBIT   SFR
        RPT     #BITLEN             ;decremented by 8/3 for
        NOP                 ;BITLEN/2 wait
        BCND    ZEROBT,bio
        ADD     #80h
ZEROBT  BANZ    WTBIT,*-            ;last bit ?
        RET                 ;ACC = read value
;***********************************************************************
; NOTE: This code is NOT written with context restore!
;***********************************************************************
RINT:   SST     #1,STAT1            ;Recover ARP from ARB by LST1
        SST     #0,STAT0
        PUSH
        MAR     *,AR7       ;
        BANZ    more_buf,*-,AR6     ;if buffer is full RET w/o EINT
        LAR     AR7,#0              ;
        LST     #1,STAT1            ;
        LST     #0,STAT0
        POP
        RET                 ;
more_buf                    ;
        LACC    DRR         ;
        SACL    *+          ;<<< store Ax[n], point to next
FASTRET LST     #1,STAT1            ;
        LST     #0,STAT0
        POP
        RETE                ;
XINT:   RETE
;=======================================================================
        .listoff            ;
        .ds     01000h      ;NOTE: Twiddles are relocated to
        .include "dsk_twid.asm"     ;   0x400 (B2) using CONF 1
        .liston
        .end
```