สเปคตรัมอะนาไลเซอร์ย่านความถี่เสียง

AF SPECTRUM ANALYZER

โดย

นาย กริช	นรขุน	37013050
นาย ชรัตน์	โฆษิตวณิชย์	37013058
นาย อรรถพร	วันดี	37013097

อาจารย์ที่ปรึกษา

อาจารย์ เกรียงไกร วงศ์โรจน์ภรณ์

ผศ.ดร. สุวิพล สิทธิชีวภาค

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าลาดกระบัง

เรื่อง สเปกตรัมอะนาไลเซอร์

AF SPECTRUM ANALYZER

ผู้จัดทำ

นายกริช	นรขุน	37013050
นายชรัตน์	โฆษิตวณิชย์	37013058
นายอรรถพร	วันดี	37013097

อาจารย์ที่ปรึกษา

(อาจารย์ เกรียงไกร วงศ์โรจน์ภรณ์)

อาจารย์ที่ปรึกษา

(ผศ.ดร. สุวิพล สิทธิชีวภาค)

สเปกตรัมอะนาไลเซอร์ ย่านความถี่เสียง

AF SPECTRUM ANALYZER

โดย	นายกริช	นรขุน	37013050
	นายชรัตน์	โฆษิตวณิชย์	37013058
	นายอรรถพร	วันดี	37013097

อาจารย์ที่ปรึกษา อาจารย์ เกรียงไกร วงศ์โรจน์ภรณ์

ผศ.ดร.สุวิพล สิทธิชีวภาค

บทคัดย่อ

ในระบบการสื่อสารโทรคมนาคม ในปัจจุบัน เครื่องมือวัดมีความจำเป็นอย่างมากเพื่อใช้ในการออกแบบ ทดลอง และการทดสอบ เครื่องวัดสเปกตรัมอะนาไลเซอร์ เป็นเครื่องมือที่ใช้สำหรับวิเคราะห์หาสเปกตรัมของความถี่ที่ใช้กันอย่างกว้างขวางในระบบการ สื่อสารโทรคมนาคม

เนื้อหาปริญญานิพนธ์ฉบับนี้นำเสนอ ทฤษฎีการออกแบบ การสร้าง และการทดสอบใช้งาน เครื่องสเปกตรัมอะนาไลเซอร์ ที่ทำงานในย่านความถี่เสียง ด้วยการวิเคราะห์ หาด้วยวิธีประมวลผลสัญญาณทางดิจิตอล จากโปรเซสเซอร์ประมวลผลทางดิจิตอล เพื่อแสดงค่าสเปกตรัมของความถี่ทางเครื่องไมโครคอมพิวเตอร์

ABSTRACT

At present, The telecommunication system measure equipment is very important to design, experiment and testing. The spectrum analyzer use for system of frequency analyser which is well-known in the telecommunication system

Thesis body present theory of design, construction, and testing The spectrum analysis which operate in audio frequency range by digital signal processing analysis for display spectrum of frequency value on micro computer.

สารบัญ

บทที่ 1

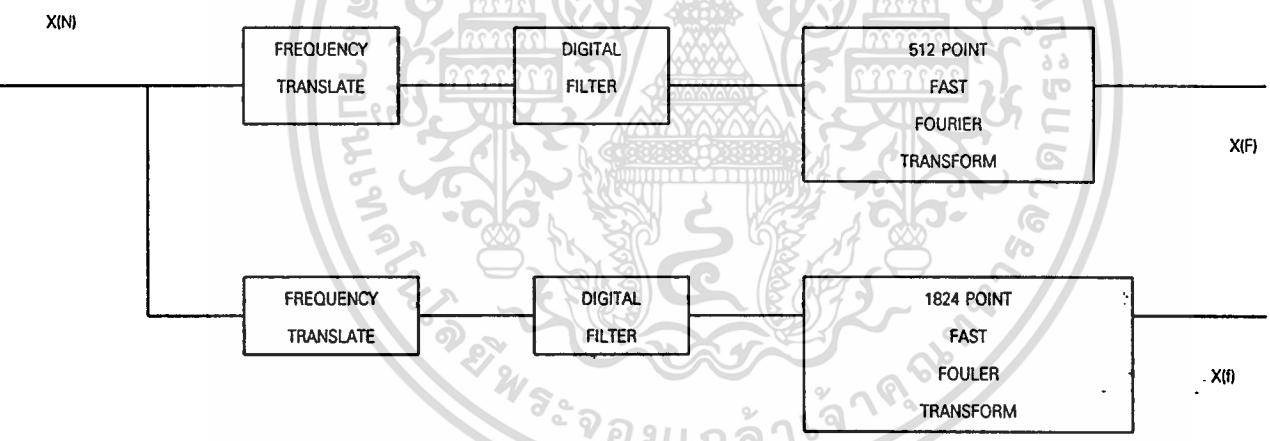
บทนำ

การประมวลผลสัญญาณเชิงเลข (Digital Signal Processing) ก็คือการเปลี่ยนรูปสัญญาณโดยใช้การคำนวณทางคณิตศาสตร์ด้วยโปรเซสเซอร์ประมวลผลสัญญาณเชิงเลข ซึ่งจะเห็นได้ว่าเราสามารถนำสัญญาณนั้นใส่ลงไปในสมการคณิตศาสตร์ แล้วได้ผลลัพธ์โดยตรงเลยทีเดียว

ในช่วง 10 ปีที่ผ่านมา ความก้าวหน้าทางด้านการประมวลผลสัญญาณเชิงเลขได้เกิดขึ้นเป็นอย่างมาก มีการค้นพบอัลกอลิทึม (Algorithms) ใหม่ๆ เพิ่มขึ้นเรื่อยๆ และการพัฒนาเครื่องมือใหม่ๆ เพื่อช่วยในด้านการออกแบบตามอัลกอลิทึมอีกด้วย ระบบการประมวลผลสัญญาณเชิงเลขนี้มีอยู่ด้วยกัน 2 แบบใหญ่ๆ คือ

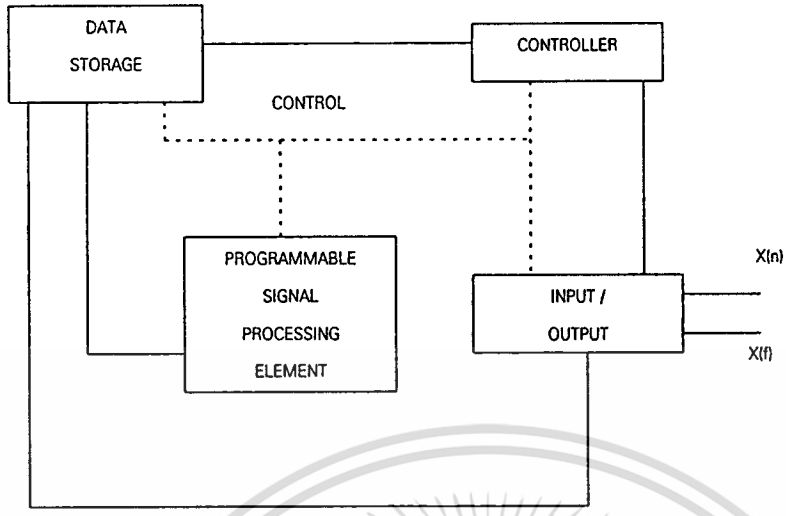
1.1 ระบบที่ออกแบบมาเพื่อให้ทำงานอย่างใดอย่างหนึ่งโดยเฉพาะ เช่น

รูปที่ 1.1



รูปที่ 1.1 การประมวลผลเฉพาะอย่าง

1.2 ระบบที่ใช้ไมโครโปรเซสเซอร์ ซึ่งจะสามารถทำงานเป็นอะไรก็ได้ขึ้นอยู่กับโปรแกรมที่ใส่ไว้ ดังรูป 1.2



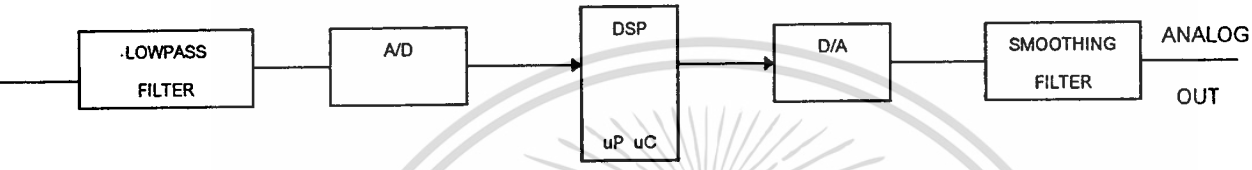
รูปที่ 1.2 การใช้ไมโครเซสเซอร์ประมวลผล

ความก้าวหน้าทางอิเล็กทรอนิกส์ครั้งใหญ่เกิดขึ้น เมื่อได้มีการพัฒนาสร้างโปรเซสเซอร์ที่ทำหน้าที่ในการประมวลผลสัญญาณเชิงเลขแบบความเร็วสูงขึ้น (High Speed Signal Processing) ซึ่งในปัจจุบันได้มีการใช้กันอย่างแพร่หลายในราคาที่ไม่แพงนักโปรเซสเซอร์ที่ทำหน้าที่ในการประมวลผลสัญญาณเชิงลบจำเป็นต้องมีความเร็วสูง และถูกออกแบบมาโดยเฉพาะให้สามารถทำงานได้อย่างมีประสิทธิภาพ โดยใช้ข้อดีของการประมวลผลแบบขนาน และการใช้ชุดคำสั่ง ทั้งนี้เพื่อการประมวลแบบเวลาแท้จริง คือเมื่อเราใส่สัญญาณอินพุทเข้าไปก็จะได้สัญญาณเอาท์พุทออกมาเลย โดยจะมีเวลาหน่วง (Delay) น้อยมาก ซึ่งจะมีประโยชน์ในการใช้งานมากกว่าการประมวลผลแบบไม่ใช่เวลาจริง

ด้วยความสามารถของการประมวลผลสัญญาณเชิงเลข จึงได้นำไปใช้ในแบบต่างๆ ซึ่งให้ผลลัพธ์ดีกว่าใช้วงจรทางอนาล็อก เช่น ความน่าเชื่อถือ ความยืดหยุ่น การทดแทนได้ง่าย ขนาดเล็กกว่า และเสถียรภาพในระยะยาวและความสามารถของโปรเซสเซอร์ที่ทำหน้าที่ในการประมวลผลสัญญาณเชิงเลข ที่สามารถเปลี่ยนโปรแกรมการทำงานได้ง่าย ทำให้การใช้งานโปรเซสเซอร์ที่ทำหน้าที่ ในการประมวลผลสัญญาณเชิงเลขนั้นน่าสนใจ เช่น สามารถเปลี่ยนให้ดีขึ้น เมื่ออัลกอลิทึมใหม่ๆ ถูกค้นหรือสามารถทำงานได้หลายๆ อย่าง โดยเพียงแต่เปลี่ยนโปรแกรมให้กับมันเท่านั้น ในทางอิเล็กทรอนิกส์นั้น โดยส่วนใหญ่มักจะเกี่ยวข้องกับสัญญาณทางไฟฟ้า เช่น การกรองสิ่งที่ไม่ต้องการออกจากสัญญาณอินพุทการแยกข้อมูลออกมา หรือไม่ก็เป็นการสร้างรูปคลื่นที่ต้องการขึ้นมา หรือเปลี่ยนลักษณะทางแอมปลิจูดของสัญญาณ ซึ่งในอดีตการกระทำเหล่านี้จะทำโดยใช้วงจร
อนาล็อกที่ออกแบบมาเพื่อใช้งานนั้นๆ นอกจากนั้น วงจรทางอนาล็อกยังไม่เหมาะที่จะสร้างเป็นวงจรรวมขนาดใหญ่ (VLSI) อีกด้วย คุณสมบัติของวงจร
อนาล็อกนั้นจะขึ้นอยู่กับค่าอุปกรณ์ที่ใช้ เช่น R, L ที่มีค่าไม่แน่นอน ที่สำคัญจะมีการรั่วไหลที่แปรตามเวลาและยังขึ้นอยู่กับโวลท์เตจและอุณหภูมิอีกด้วย การออกแบบวงจรทางอนาล็อกนั้น ยังน่าเบื่อ ทั้งวงจรหนึ่งๆ จะทำงานได้อย่างเดียวเท่านั้น แต่ถ้าต้องการให้ทำงานอื่นๆ ก็ต้องเปลี่ยนอุปกรณ์ และก็ต้องเดินสายใหม่

อุปกรณ์ และก็ต้องเดินสายใหม่ นอกจากนี้แล้วการทดสอบวงจรอนาล็อกอาจเป็นปัญหาได้ เนื่องมาจาก อิมพีแดนซ์ของเครื่องมือวัดที่ต่อเข้าไปในวงจร ปัญหาพื้นฐานของวงจรอนาล็อกเหล่านี้ สามารถแก้ได้ โดยใช้ DSP ข้อได้เปรียบอีกประการของการใช้ DSP แทนวงจรอนาล็อกคือ การเปลี่ยนคุณสมบัติ ของการประมวลผลสามารถทำได้โดยง่าย ซึ่งจะทำให้เราสามารถทดลอง ในทุกทางที่เป็นไปได้ เพื่อเลือกเอาผลที่ดีที่สุดที่เราพอใจ

โดยทั่วไป ระบบ DSP จะประกอบด้วยส่วนต่างๆ ดังรูปที่ 1.3

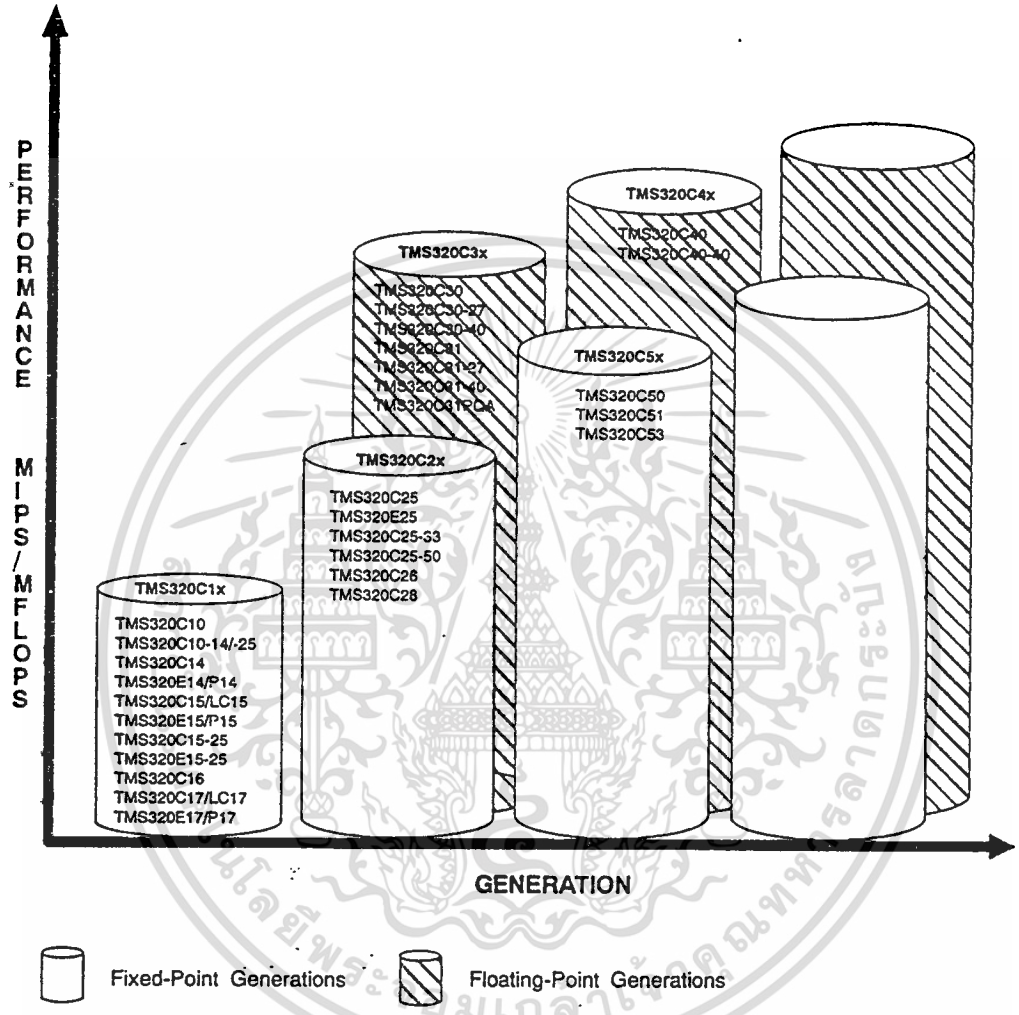


รูปที่ 1.3 ส่วนประกอบของ DSP

สัญญาณอนาล็อกอินพุทจะถูกป้อนเข้าสู่วงจรกรองความถี่ต่ำผ่าน เพื่อที่จะจำกัดช่วงความถี่ของสัญญาณอินพุทจากนั้นก็ส่งต่อไปยัง อนาล็อก/ดิจิตอล คอนเวอร์เตอร์ (Analog to Digital Converter : A/D) แล้วส่งให้ระบบประมวลผลสัญญาณเชิงเลขต่อไป ระบบประมวลผลสัญญาณเชิงเลข ก็จะนำเอาอัลกอลิทึมที่มีอยู่มากกระทำกับข้อมูลสัญญาณ จากนั้นก็นำจะส่งข้อมูลสัญญาณที่ถูกแก้ไขแล้วไปให้ ส่วน ดิจิตอล/อนาล็อก คอนเวอร์เตอร์ (Digital to Analog converter : D/A) เพื่อเปลี่ยนสัญญาณข้อมูลกลับให้เป็นสัญญาณอนาล็อกที่ต้องการแต่จากเอาท์พุทของ D/A

ลักษณะเป็นขั้นๆ แยกกัน ดังนั้นเราจึงจำเป็นต้องใช้วงจรกรองสัญญาณให้เรียบ (Low Pass Filter) เพื่อที่จะขจัดสัญญาณความถี่สูงที่ไม่ต้องการออกไปที่กล่าวมานั้นมีแต่ส่วนดีของ DSP แต่ส่วนเสียของ DSP ก็มีปัญหาที่เกิดขึ้นจะแตกต่างไปจากปัญหาที่เกิดขึ้นในวงจรอนาล็อก เช่น การเกิดควันไตซ์เซชัน (Quantization), การเกิดน๊อยด์ (Noise), Idel channal noise, ความไม่เรียบของสัญญาณเป็นต้น ตัวแปรที่สำคัญอีกตัวหนึ่งที่จะแยกระบบ DSP ต่างๆ ออกตามการใช้งานก็คือค่าอัตราการสุ่มข้อมูล (Sampling Rate) ยิ่งการสุ่มข้อมูลมากยิ่งต้องใช้ระบบประมวลผลที่มีความเร็วสูงยิ่งขึ้นและซับซ้อนยิ่งขึ้นด้วย อัตราการสุ่มข้อมูลนี้จะมีผลต่อการนำไปใช้งาน ถ้ามีอัตราการสุ่มข้อมูลที่สูงก็จะใช้งานในย่านความถี่สูงขึ้นได้ โดยคุณภาพของสัญญาณข้อมูลยังคงยอมรับได้

บริษัทเทกซัสอินสตรูเมนต์ จึงได้ผลิตไอซีที่เป็นตัวประมวลผลออกมาสำหรับใช้งานทางด้านของการประมวลผลสัญญาณเชิงเลขโดยเฉพาะ ซึ่งเป็นตระกูลที่มีชื่อว่า TMS320 ซึ่งมีวิวัฒนาการดังรูปที่ 1.4



รูปที่ 1.4 วิวัฒนาการของไอซีตระกูล TMS320

ไอซีตระกูล TMS320 นี้เป็นลักษณะไมโครโปรเซสเซอร์แบบชิพเดียว (Single Chip) สามารถทำงานได้กว้างขวาง และมีความเร็วในการประมวลผลสูง

ไอซีเบอร์ TMS 320C25 สามารถนำไปใช้งานได้อย่างกว้างขวางโดยเฉพาะการประมวลผลจากเวลาจริง (Real time) การประยุกต์ใช้งานไอซีเบอร์ TMS320C26 แสดงในตารางที่ 1.1

GENERAL-PURPOSE DSP	GRAPHICS/IMAGING	INSTRUMENTATION
Digital Filtering	3-D Rotation	Spectrum Analysis
Convolution	Robot Vision	Function Generation
Correlation	Image Transmission/	Pattern Matching
Hilbert Transforms	Compression	Seismic Processing
Fast Fourier Transforms	Pattern Recognition	Transient Analysis
Adaptive Filtering	Image Enhancement	Digital Filtering
Windowing	Homomorphic Processing	Phase-Locked Loops
Waveform Generation	Workstations	
	Animation/Digital Map	
VOICE/SPEECH	**CONTROL**	**MILITARY**
Voice Mail	Disk Control	Secure Communication
Speech Vocoding	Servo Control	Radar Processing
Speech Recognition	Robot Control	Sonar Processing
Speaker Verification	Laser Printer Control	Image processing
Speech Enhancement	Engine Control	Navigation
Speech Synthesis	Motor Control	Missile Guidance
Text-to-Speech		Redio Frequency Modems
TELECOMMUNICATIONS		**AUTOMOTIVE**
Echo Cancellation	FAX	Engine Control
ADPCM Transcoders	Cellular Telephones	Vibration Analysis
Digital PBXs	Speaker Phones	Antiskid Brakes
Line Repeaters	Digital Speech Interpolation(DSI)	Adaptive Ride Control
Channel Multiplexing	X.25 Packet Switching	Global Positioning
1200 to 19200-bps Modems	Video Conferancing	Navigation
Adaptive Equalizers	Spread Spectrum Communication	Voice Commands
DTMF Encoding/Decoding		Digital Radio
Data Encryption		Cellular Telephones
CONSUMER	**INDUSTRIAL**	**MEDICAL**
Redar Detectors	Robotics	Hearing Aids
Power Tools	Numeric Control	Patient Monitoring
Digital Audio/TV	Security Access	Ultrasound Equipment
Music Synthesizer	Power Line Monitors	Diagnostic Tools
Toys and Games		Prosthetics
Solid-state Answering Machines		Fetal Monitors

ตารางที่ 1.1 การประยุกต์ใช้งานไอซีเบอร์ TMS320C26

ในปริญญานิพนธ์นี้เราจะใช้ TMS320C26 มาประยุกต์ใช้ขานทางด้านเตรื่องวิเคราะห์แถบความถี่ย่านความถี่เสียง

บทที่ 2

ทฤษฎีและหลักการ

2.1 สถาปัตยกรรมของ TMS320C26

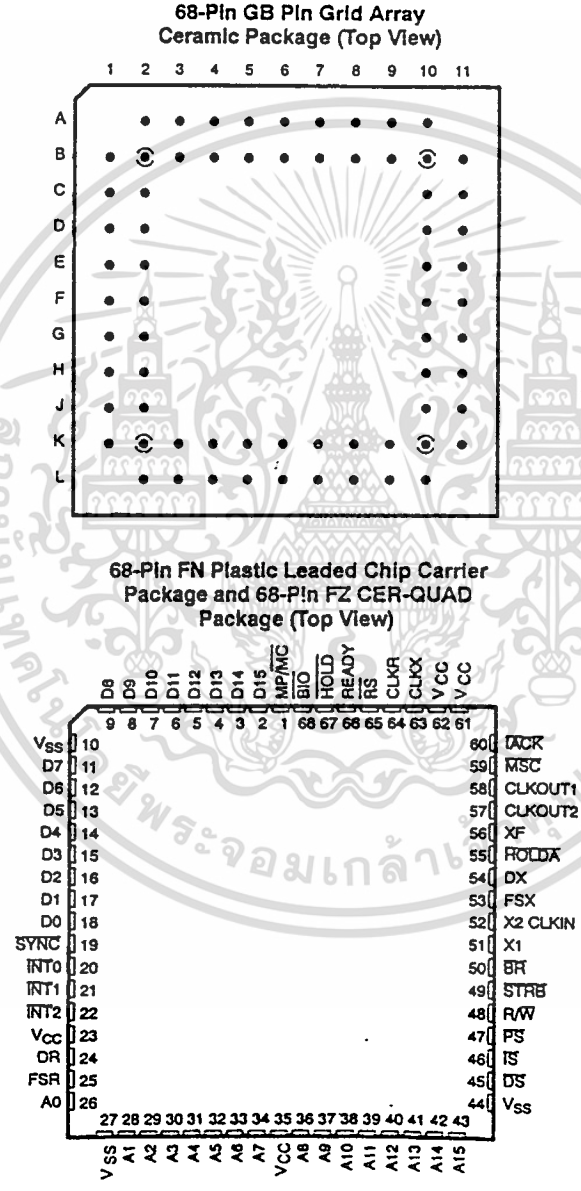
ในส่วนนี้ จะเป็นการกล่าวถึงสถาปัตยกรรม และโครงสร้างภายในโดยทั่วๆ ไป ของ TMS320C26 ซึ่งจะเห็นได้ว่าโครงสร้างภายในต่างๆ นั้นได้มีการพัฒนาและเพิ่มขีดความสามารถของการทำงานในด้านต่างๆ ขึ้นจากรุ่นก่อน คือ TMS320C1X มาก ซึ่งได้แก่การคูณเลขขนาด 16x16 บิท จะใช้เวลาทำงานเพียงไซเกิลเดียว คือ 100 นาโนวินาทีเท่านั้น และในการต่อกับหน่วยความจำภายนอกนั้น สามารถอ้างแอดเดรสในส่วนของหน่วยความจำข้อมูล และหน่วยความจำโปรแกรมได้ถึง อย่างละ 64 กิโลเวิร์ด และยังมีจำนวนฮาร์ดแวร์สแตคเพิ่มขึ้นถึง 8 ระดับ เป็นต้น นอกจากนี้ยังประกอบด้วยส่วนต่างๆ ที่เกี่ยวข้อง ซึ่งจะเป็นการเอื้ออำนวยประโยชน์ในการนำไปประยุกต์ใช้งานได้อย่างกว้างขวาง ดังนั้นในส่วนนี้จะประกอบไปด้วยส่วนต่างๆ ซึ่งพอจะสรุปได้ดังนี้

- โครงสร้างโดยทั่วไป
- สถาปัตยกรรมโดยทั่วไป
- ฟังก์ชั่นบล็อกไดอะแกรม (Function Block Diagram)
- การจัดหน่วยความจำ
- หน่วยความจำข้อมูล
- หน่วยความจำโปรแกรม
- การจัดผังหน่วยความจำ
- การจัดรีจิสเตอร์ของผังหน่วยความจำ
- รีจิสเตอร์ช่วย (Auxiliary Register)
- โหมดการอ้างแอดเดรสของหน่วยความจำ
- การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำ
- หน่วยศูนย์กลางในการคำนวณทางคณิตศาสตร์และลอจิก
- สเกลลิ่งชิฟเตอร์ (Scaling Shifter)
- แอคคิวมูเลเตอร์ และ ALU
- ตัวคูณ, รีจิสเตอร์ T และ P
- ส่วนควบคุมระบบ

- โปรแกรมเคาน์เตอร์และสแตด (Program Counter and Stack)
- ปฏิบัติการไปป์ไลน์ (Pipeline)
- รีเซต (Reset)
- รีจิสเตอร์สถานะ (Status Register)
- ปฏิบัติการเกี่ยวกับเวลา (Timer Operation)

2.2 โครงสร้างโดยทั่วไป

- วัฏจักรในการทำงานของคำสั่งใช้เวลา 100 นาโนวินาที
- มีหน่วยความจำข้อมูลภายในตัวชิพ (On-Chip) ขนาด 544 กิโลเวิร์ด
- มีหน่วยความจำโปรแกรมภายในตัวชิพ (On-Chip) ขนาด 4 กิโลเวิร์ด
- สามารถต่อหน่วยความจำเพิ่มในส่วนของ ข้อมูล/โปรแกรม ได้รวมทั้งหมด 128 กิโลเวิร์ด
- ALU และ แอคคิวมูเลเตอร์ (Accumulator) มีขนาด 32 บิท
- การคูณจะเป็นแบบขนานขนาด 16x16 บิท ซึ่งจะได้ผลลัพธ์เป็นเลข 32 บิท
- สามารถทำการคูณเลขโดยใช้เวลาเพียงไซเกิลเดียว (100 นาโนวินาที)
- มีคำสั่งในการเคลื่อนย้ายข้อมูล เป็นบล็อกในการจัดการกับข้อมูล หรือโปรแกรม
- มีวงจรนาฬิกา (Timer) ภายในชิพ เพื่อใช้ควบคุมการทำงานต่างๆ
- มีรีจิสเตอร์ช่วย (Auxiliary Registers) จำนวน 6 ตัว
- เพิ่มจำนวนฮาร์ดแวร์สแตคให้มีเพิ่มขึ้นถึง 8 ระดับ
- มีอินพุท และเอาท์พุตขนาด 16 แชนแนล
- มีตัวเลื่อนบิท (Shifter) แบบขนานขนาด 16 บิท
- มีสัญญาณ Wait Statues เพื่อสำหรับในการติดต่อกับอุปกรณ์ หรือ หน่วยความจำภายนอกที่ทำงานช้ากว่า
- มีพอร์ตอนุกรม เพื่อใช้สำหรับในการเชื่อมต่อ
- มีตัวกำหนดสัญญาณนาฬิกา (Clock) ภายในตัวชิพ
- ต้องการแหล่งจ่ายไฟฟ้าจากภายนอกเพียงชุดเดียว (5V.)
- เป็นอุปกรณ์ที่มีการบรรจุแบบ 68-Lead PLCC
- ใช้เทคโนโลยีในการผลิตแบบซีมอส

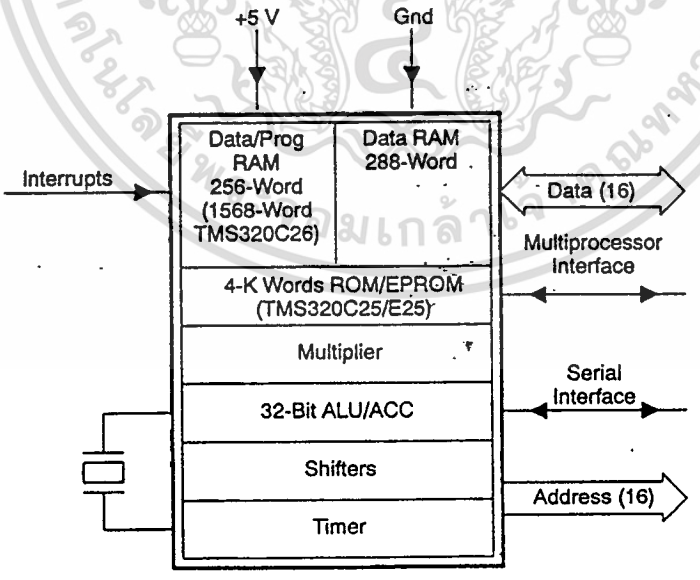


รูปที่ 2-1 ลักษณะภายนอกของ TMS320C26

2.3 สถาปัตยกรรมโดยทั่วไป

TMS320C26 เป็นโปรเซสเซอร์ ที่ใช้สำหรับในการประมวลผลสัญญาณเชิงเลข ซึ่งจะมีลักษณะการทำงานคล้ายคลึงกับ TMS320C1X (เป็น CPU เบอร์แรกในตระกูล TMS 320) และใช้สถาปัตยกรรมแบบเดียวกัน คือสถาปัตยกรรมแบบ "ฮาวาร์ด" โดยจะแบ่งโครงสร้างของหน่วยความจำออกเป็น 2 ส่วนแยกจากกัน คือ หน่วยความจำโปรแกรม และหน่วยความจำข้อมูล ซึ่งจะทำให้การทำงานต่างๆ เป็นไปด้วยความรวดเร็วสูง นอกจากนี้ยังสามารถทำการโอนย้ายข้อมูลระหว่างกันและกันได้

ในการคำนวณทางคณิตศาสตร์นั้น จะทำการคำนวณโดยใช้เลขแบบทูคอมพลีเมนท์ โดยใช้ ALU และแอคคิวมูเลเตอร์ ซึ่งมีขนาด 32 บิท ALU ซึ่งเป็นหน่วยกระทำการทางคณิตศาสตร์และลอจิกจะใช้ตัวกระทำขนาด 16 บิทเวิร์ด ซึ่งมาจากหน่วยความจำข้อมูล หรือได้มาจากคำสั่งในการอ้างถึงแบบทันที ในการใช้คำสั่งในการบวกจะแบ่งออกได้เป็น 2 ส่วน คือ ทางด้านบิทสูง (ACCH) และทางด้านบิทต่ำ (ACCL) ซึ่งก็คือ บิท 31-16 และบิท 15-0 ตามลำดับ สำหรับการคูณเลขขนาด 16x16 บิท สามารถจะทำการคูณได้ภายในไซเกิลเดียว (ใช้เวลา 100 นาโนวินาที) ในการคูณเลขจะประกอบด้วย 3 ส่วนใหญ่ๆ ด้วยกันคือ T, P รีจิสเตอร์ และมัลติพลายเออร์แอร์แรย์ โดย T รีจิสเตอร์จะใช้สำหรับเก็บตัวตั้งขนาด 16 บิท ส่วน P รีจิสเตอร์จะใช้เก็บค่าของผลคูณขนาด 32 บิท ค่าของตัวคูณอาจจะมาจากหน่วยความจำข้อมูล หรือมาจากหน่วยความจำโปรแกรมโดยการใช้คำสั่ง MAC/MACD หรือ ได้มาจากการใช้คำสั่ง MPYK (Multiply Immediate) ภายในตัวชิพจะมีการคำนวณต่างๆ ด้วยความเร็ว โดยการใช้ปฏิบัติการพื้นฐานของ DSP ซึ่งได้แก่ การคอนโวลูชั่น (Convolution) , คอรรีเลชั่น (Correlation) และการกรอง (Filtering)



รูปที่ 2-2 บล็อกไดอะแกรมอย่างง่าย ของ TMS320C26

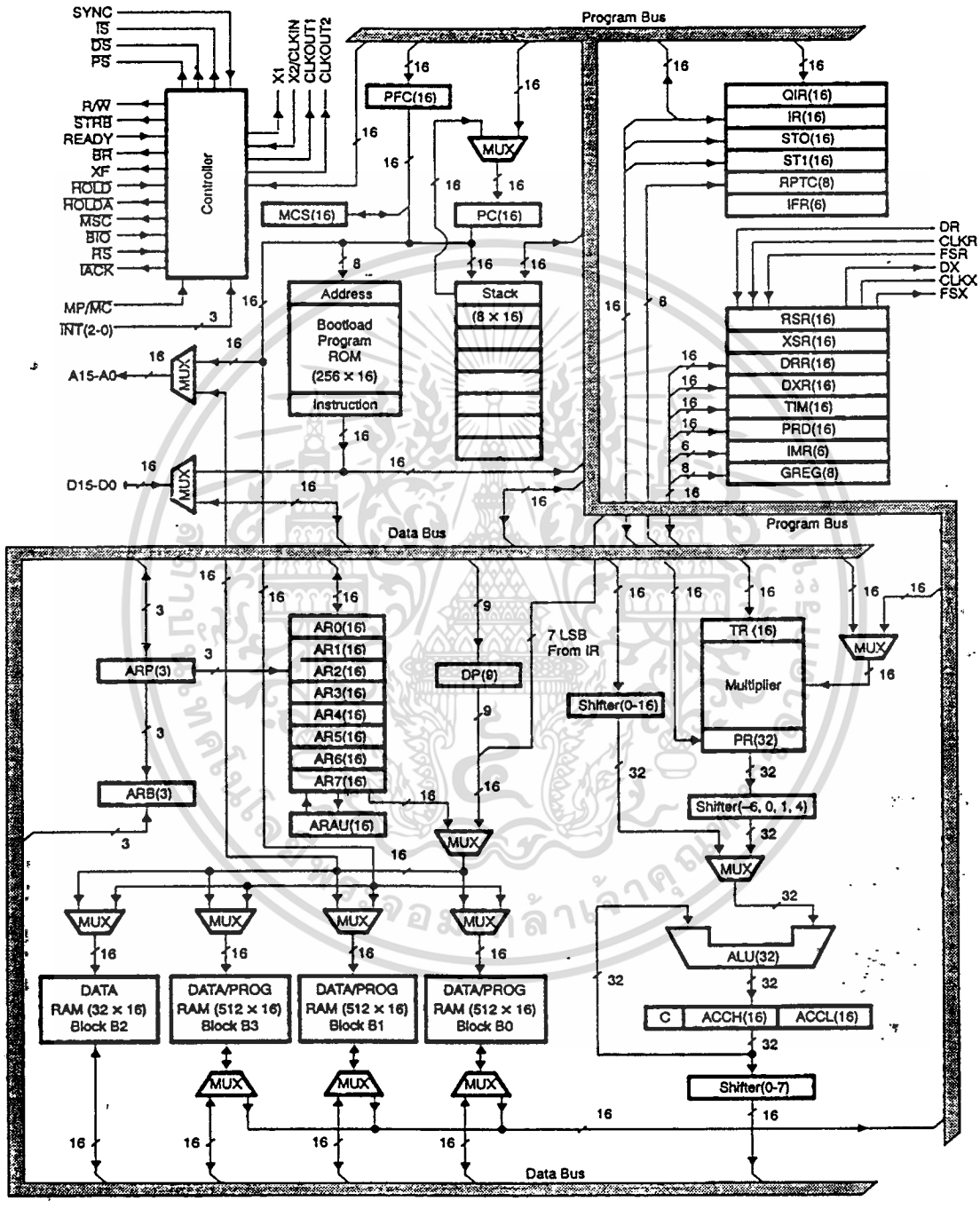
ในส่วนของการเลื่อนบิทของ TMS320C26 นั้นจัดให้มีสเกลลิ่งชิฟเตอร์ (Scaling Shifter) ซึ่งมีอินพุทขนาด 16 บิทต่ออยู่กับบัสข้อมูลและเอาท์พุทขนาด 32 บิท ซึ่งต่ออยู่กับ ALU โดยสามารถทำการเลื่อน

บิทของข้อมูลอินพุทไปทางซ้ายได้ 0-16 บิท โดยบิทซึ่งมีนัยสำคัญต่ำ (LSBs) จะถูกเติมด้วย 0 และบิทที่มีนัยสำคัญสูง (MSBs) อาจจะถูกเติมด้วย 0 หรือใช้เป็นส่วนขยายเครื่องหมาย

นอกจากนี้ยังมีฮาร์ดแวร์สแตคเพิ่มขึ้นถึง 8 ระดับ หน้าที่ของสแตคก็คือ ใช้ในการเก็บค่าของโปรแกรมเคาน์เตอร์ (PC) ระหว่างที่มีการขออินเตอร์รัพท์หรือมีการเรียกใช้โปรแกรมย่อย (Subroutine)

2.4 ฟังก์ชั่นบล็อกไดอะแกรม

ฟังก์ชั่นบล็อกไดอะแกรมที่แสดงตามใน รูปที่ 2-3 จะแสดงถึงการทำงานของ TMS320C26 และทางเดินของสัญญาณต่างๆ ภายในตัวชิพ ตลอดจนขาสัญญาณสำหรับอินเตอร์เฟสกับอุปกรณ์ภายนอก



NOTE: Shaded areas indicate a bus.

รูปที่ 2-3 บล็อกไดอะแกรมภายในของ TMS320C26

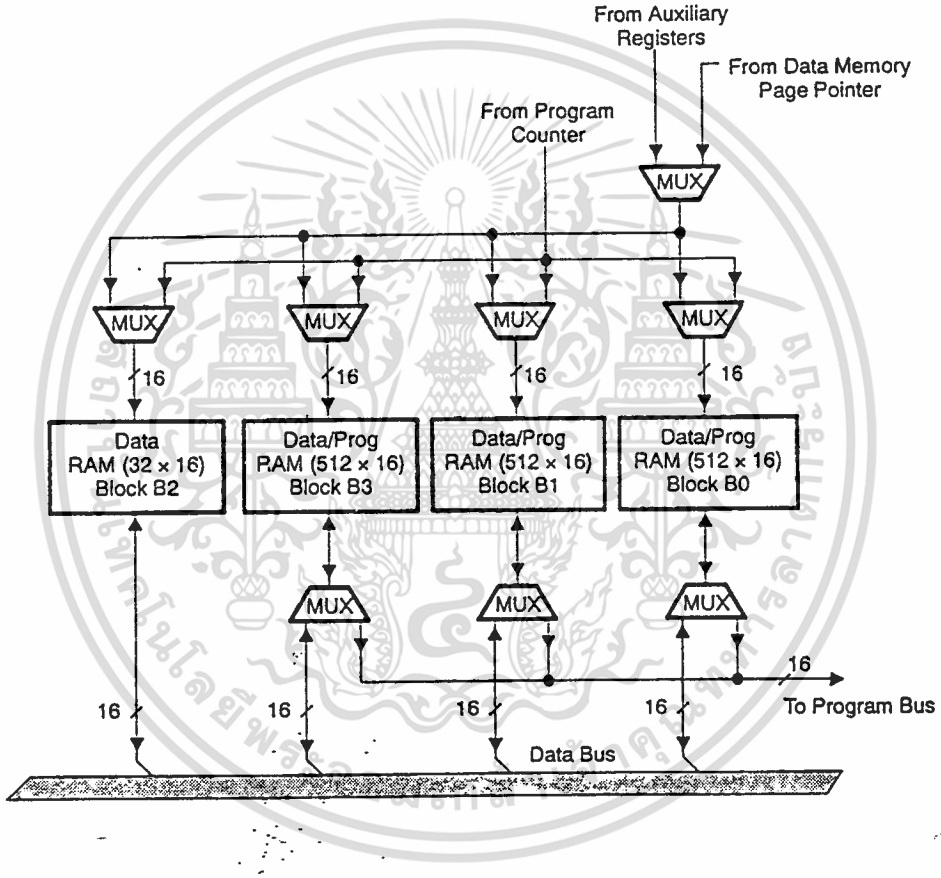
2.5 การจัดหน่วยความจำ

ลักษณะภายในของ TMS320C26 ได้กำหนดให้มีแรมข้อมูล (Data RAM) ภายในตัวชิพ (On-Chip) ขนาด 544 16 บิทเวิร์ด ซึ่งในส่วนของ 288 เวิร์ดแรกนั้นจะถูกใช้เป็นหน่วยความจำข้อมูล หรือ หน่วยความจำโปรแกรม นอกจากนี้ยังมี มาสเคเบิลโปรแกรมรอม (Maskable Program ROM) ขนาด 4 กิโลเวิร์ดอีกด้วย ในส่วนต่อไปนี้จะเป็น การอธิบายถึง การจัดการเกี่ยวกับข้อมูลภายในชิพ, หน่วยความจำข้อมูล, หน่วยความจำโปรแกรม, การจัดผังหน่วยความจำ, การจัดรีจิสเตอร์ของผังหน่วยความจำ, รีจิสเตอร์ช่วย, โหมดในการอ้างแอดเดรสของหน่วยความจำและ การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำ

2.5.1 หน่วยความจำข้อมูล

จากที่ทราบมาแล้วว่าแรมข้อมูลภายในตัวชิพ นั้นจะมีขนาดทั้งหมด 544 เวิร์ด ซึ่งจากรูปที่ 2-3 จะเห็นว่าสามารถแบ่งออกได้เป็น 3 บล็อก คือ B0, B1, B2 ในส่วนของบล็อก B0 ซึ่งมีขนาด 256 เวิร์ด จะถูกใช้งานร่วมกันระหว่างเป็นหน่วยความจำข้อมูล หรือหน่วยความจำโปรแกรมซึ่งก็แล้วแต่ความต้องการ (โดยการใช้คำสั่ง CNFP) และในส่วนของบล็อก B1 และ B2 ซึ่งมีขนาด 258 เวิร์ด จะถูกใช้งานเป็นหน่วยความจำข้อมูลเท่านั้น นอกจากนี้ยังสามารถต่อเพิ่มหน่วยความจำภายนอกได้อีก ซึ่ง TMS320C26 ยอมให้มีการต่อเพิ่มหน่วยความจำภายนอกได้ถึง 64 กิโลเวิร์ด (หน่วยความจำข้อมูล ภายในชิพและพื้นที่ภายในบางส่วนซึ่งถูกสงวนไว้รวมแล้วจะกินเนื้อที่ประมาณ 1 กิโลเวิร์ด ซึ่งจะดูได้จากรูปที่ 2-4) สำหรับสัญญาณ READY นั้นจะใช้สำหรับในการเชื่อมต่อกับอุปกรณ์ที่มีความเร็วต่ำและหน่วยความจำน้อย เช่น DRAMs

ตัวอย่างของชุดคำสั่งต่างๆ ที่ใช้งาน ได้แก่ คำสั่ง TBLW จะเป็นการส่งข้อมูลจากหน่วยความจำภายนอกเข้ามายังหน่วยความจำข้อมูลและคำสั่ง IN จะเป็นการอ่านข้อมูลจากพอร์ตภายนอกเข้ามายังหน่วยความจำข้อมูลภายใน



รูปที่ 2-4 หน่วยความจำข้อมูลภายในชิพของ TMS320C26

2.5.2 หน่วยความจำโปรแกรม

ในส่วนของ TMS320C26 จะมีหน่วยความจำรอมภายในตัวชิพ ขนาด4กิโลเวิร์ด ซึ่งจะเรียกว่า "มาสโปรแกรม (Mask-Programmed)" โดยผู้ใช้สามารถจะสั่งให้โรงานผู้ผลิตเป็นผู้โปรแกรมในส่วนนี้ได้ สำหรับรอมภายในตัวชิพจะยอมให้มีการเอ็กซีคิ้วท์ โปรแกรมจากหน่วยความจำภายนอกได้ การใช้งานหน่วยความจำนี้จะยอมให้บัสข้อมูลภายนอกเป็นอิสระต่อการเข้าถึงหน่วยความจำข้อมูลภายนอก ในการจัดผังหน่วยความจำของหน่วยความจำโปรแกรม ว่าต้องการจะใช้หน่วยความจำภายในหรือต่อจากภายนอกจะเป็นส่วนผู้ที่ใช้จะเป็นผู้กำหนด โดยหมายถึงขา MP/MC บน TMS320C26 คือ ถ้ากำหนดให้ขา MP/MC เป็นระดับสัญญาณสูง จะหมายถึง เป็นการใช้หน่วยความจำภายนอกชิพ แต่ถ้าหนดให้ขา MP/MC เป็นระดับสัญญาณต่ำ จะหมายถึง เป็นการใช้หน่วยความจำภายในชิพ สำหรับขา XF (External Flag) จะใช้ในการสลับขา MP/MC เพื่อให้รอมภายในชิพนั้น อีนาเบิล (Enable) หรือ ดิสเอเบิล (Disable)

2.5.3 การจัดผังหน่วยความจำ

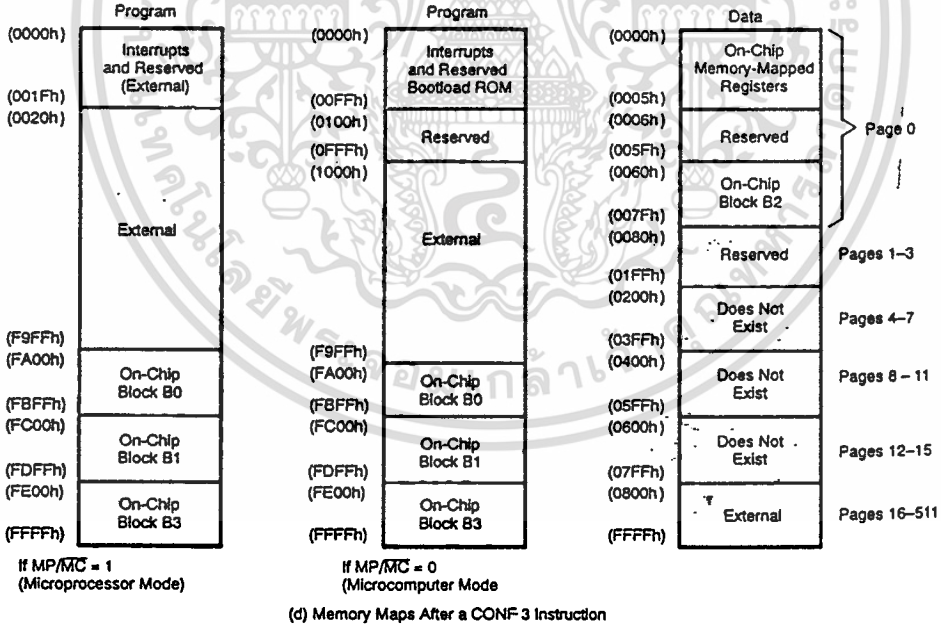
การจัดผังหน่วยความจำของ TMS320C26 นั้น จะมีการแบ่งส่วนของแอดเดรสออกเป็น 3 ส่วนด้วยกัน คือ หน่วยความจำโปรแกรม, หน่วยความจำข้อมูล และ I/O ซึ่งแสดง ดังรูปที่ 2-5 ถ้ามองจากภายนอกจะหมายถึง สัญญาณ /PS ,/DS และ /IS (เป็นสัญญาณที่ใช้ในการเลือกโปรแกรม , ข้อมูล หรือ I/O) ซึ่งสัญญาณ /PS ,/DS ,/IS และ /STRB จะแอคทีฟเมื่อมีแอดเดรสจากหน่วยความจำภายนอกเข้ามา

ในส่วนของแรมในตัวชิพนั้นจะประกอบด้วย 3 ส่วน คือ บล็อก B0 , B1 , B2 ซึ่งรวมกันแล้วจะมีขนาด 544 เวิร์ด ในส่วนที่ใช้เป็นโปรแกรม/ข้อมูล (แรมบล็อก B0 ขนาด 256 เวิร์ด) จะอยู่ในหน้าที่ 4 และ 5 ของผังหน่วยความจำข้อมูล (เมื่อใช้งานเป็นแรมข้อมูล) และจะอยู่ในแอดเดรส > FF00 ถึง > FFFF (เมื่อใช้งานเป็นโปรแกรมแรม) ส่วนบล็อก B1 จะอยู่ในหน้าที่ 6 และ 7 ของผังหน่วยความจำข้อมูลในขณะที่บล็อก B2 จะอยู่ในส่วนของ 32 เวิร์ดด้านบนของหน้า 0 และส่วนที่เหลืออยู่ในหน้า 0 นั้นจะประกอบด้วย การจัดรีจิสเตอร์ของผังหน่วยความจำทั้ง 6 ตัว และพื้นที่ภายในซึ่งจองหรือสงวนไว้ ในพื้นที่ที่จองหรือสงวนไว้นี้จะไม่สามารถนำมาใช้เก็บข้อมูลหรือใช้งานใดๆ และตั้งแต่แอดเดรสที่ 1024-65535 ซึ่งอยู่ในหน้าที่ 8-511 นั้นจะใช้สำหรับในการต่อกับหน่วยความจำภายนอก

ในส่วนต่อไปจะขอกล่าวถึง การจัดผังหน่วยความจำโปรแกรมจากที่ทราบมาแล้วว่า TMS320C26 จะมีโหมดการทำงานอยู่ 2 โหมดด้วยกัน คือ โหมดไมโครคอมพิวเตอร์ และโหมดไมโครโปรเซสเซอร์ซึ่งทั้ง 2 โหมดนี้จะมีการจัดผังของหน่วยความจำโปรแกรมไม่เหมือนกันดังจะกล่าวต่อไปนี้

เมื่อกำหนดให้ขา MP/MC = 1 (กำหนดให้อยู่ในโหมดไมโครโปรเซสเซอร์) ในโหมดนี้จะไม่มีรอมภายในตัวชิพ แต่จะเป็นการต่อหน่วยความจำจากภายนอกทั้งหมด 64 กิโลเวิร์ด

ถ้ากำหนดให้ขา MP/MC = 0 (กำหนดให้อยู่ในโหมดไมโครคอมพิวเตอร์) ในโหมดนี้ TMS320C26 จะกำหนดให้มีรอมภายในตัวชิพขนาด 4 กิโลเวิร์ด และยังสามารถทำการต่อขยายหน่วยความจำภายนอกได้อีก คือตั้งแต่แอดเดรสที่ 4096-65535 นอกจากนี้ยังมีคำสั่งซึ่งใช้ในการจัดผังหน่วยความจำใหม่ คำสั่งนั้นคือ CNFP จากรูปที่ 2-5 เป็นการจัดผังหน่วยโปรแกรมใหม่ภายหลังจากการใช้คำสั่ง CNFP ซึ่งจากรูปจะเห็นว่า ในส่วนของหน่วยความจำโปรแกรมตำแหน่งที่ 65280-65535 จะถูกใช้เป็นโปรแกรมแรม ซึ่งก็คือ บล็อก B0 ของหน่วยความจำข้อมูลนั่นเอง โดยโปรแกรมแรมในส่วนนี้สามารถจะใช้ในการเขียนโปรแกรมลงไป หรืออ่านออกมาภายนอกได้ โปรแกรมแรมในส่วนนี้จะมีขนาด 256 เวิร์ด



(d) Memory Maps After a CONF 3 Instruction

รูปที่ 2-5 การจัดผังหน่วยความจำหลังจากใช้คำสั่ง CNFP ของ TMS320C26

2.5.4 การจัดรีจิสเตอร์ของผังหน่วยความจำ

จากตารางที่ 2-1 แสดงถึงแอดเดรสและหน้าที่ของรีจิสเตอร์ภายในทั้ง 6 ตัวของผังหน่วยความจำข้อมูล และแสดงบล็อกไดอะแกรมของรีจิสเตอร์ต่างๆ ดังในรูปที่ 2-3

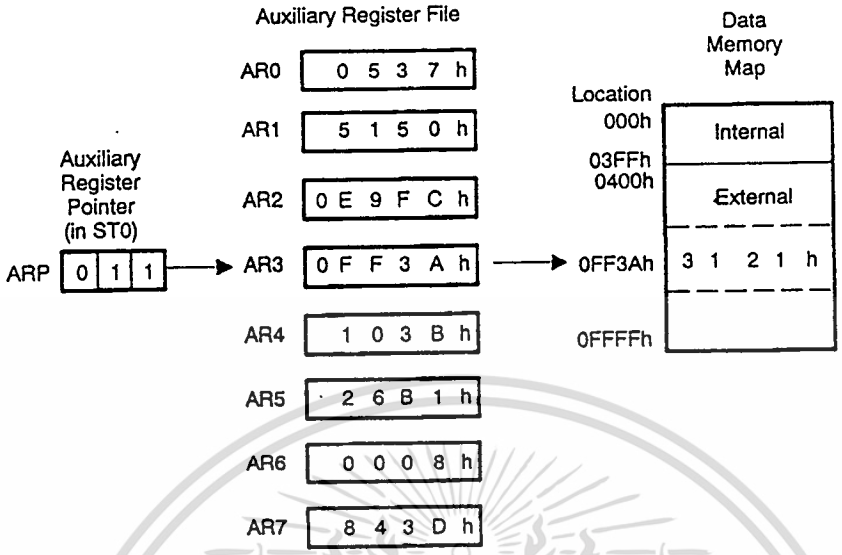
การจัดผังรีจิสเตอร์หน่วยความจำนี้อาจจะมีลักษณะการจัดเหมือนกับในหน่วยความจำข้อมูล อื่นๆ แต่จะมีข้อยกเว้น คือไม่สามารถจะใช้คำสั่งการเคลื่อนย้ายข้อมูลเป็นบล็อก (BLKD) จากรีจิสเตอร์หน่วยความจำเหล่านี้ได้

REGISTER NAME	ADDRESS LOCATION	DEFINITION
DRR(15-0)	0	Serial port data receive register
DXR(15-0)	1	Serial port data transmit register
TIM(15-0)	2	Timer register
PRD(15-0)	3	Period register
IMR(5-0)	4	Interrupt mask register
GREG(7-0)	5	Global memory allocation register

ตารางที่ 2-1 การจัดรีจิสเตอร์ของผังหน่วยความจำ

2.5.5 รีจิสเตอร์ช่วย

สำหรับรีจิสเตอร์ช่วยภายในของ TMS320C26 นั้นได้กำหนดให้มีจำนวนถึง 8 ตัว คือ AR0-AR7 โดยรีจิสเตอร์ช่วยขนาด 16 บิท ซึ่งใช้ในการอ้างแอดเดรสแบบทางอ้อมของหน่วยความจำข้อมูล หรือใช้สำหรับเป็นที่เก็บข้อมูลชั่วคราว รีจิสเตอร์นี้จะถูกชี้ค่าโดยค่าที่อยู่ใน ARP (เป็นรีจิสเตอร์ขนาด 3 บิท) ซึ่งมีค่าระหว่าง 0-7 จะหมายถึง AR0-AR7 ทั้ง ARP และรีจิสเตอร์ช่วยนี้อาจจะโหลดค่ามาจากหน่วยความจำข้อมูล หรืออาจจะมาจากโอเปอร์แรนด์ของคำสั่งแบบทันทีก็ได้



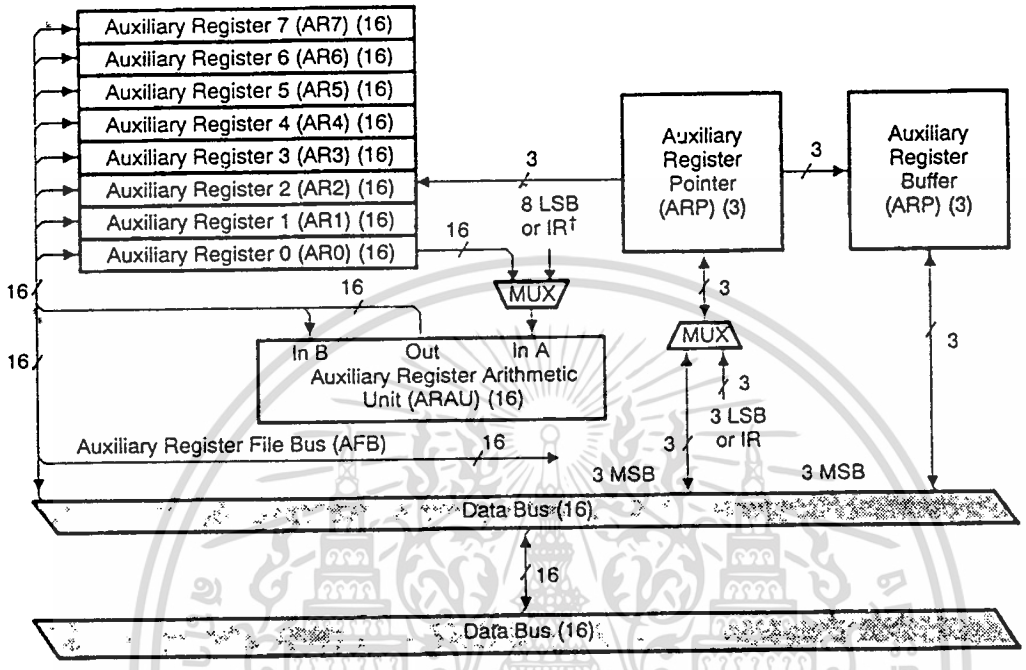
รูปที่ 2-6 ตัวอย่างของรีจิสเตอร์ช่วยที่ใช้ในการอ้างแอดเดรสแบบทางอ้อม

รีจิสเตอร์ช่วย (AR0-AR7) นี้จะต่ออยู่กับ Auxiliary Register Arithmetic Unit (ARAU) แสดงดังรูปที่ 2-7 จากรูปจะเห็นว่า AR0 จะถูกต่อเป็นอินพุทหนึ่งของ ARAU ส่วนอินพุทอื่นจะมาจาก AR ที่ใช้อยู่ในปัจจุบันโดยที่ ARAU จะมีการปฏิบัติการตามฟังก์ชั่นดังต่อไปนี้

AR(ARP) + AR0 —> AR(ARP) เป็นการชี้ตำแหน่งปัจจุบันของ AR โดยการบวกด้วยเลขจำนวนเต็มขนาด 16 บิท ซึ่งเก็บอยู่ใน AR0

AR(ARP) - AR0 —> AR(ARP) เป็นการชี้ตำแหน่งปัจจุบันของ AR โดยการลบด้วยเลขจำนวนเต็มขนาด 16 บิท ซึ่งเก็บอยู่ใน AR0

AR(ARP) + 1	—> AR(ARP)	เพิ่มค่าของ AR ขึ้นอีก
AR(ARP) - 1	—> AR(ARP)	ลดค่าของ AR ลง 1
AR(ARP)	—> AR(ARP)	ไม่มีการเปลี่ยนแปลง



รูปที่ 2-7 Auxiliary Register File

ในส่วนของฟังก์ชั่นที่นอกเหนือไปจากนี้ ARAU ของ TMS320C26 จะมีการปฏิบัติตามฟังก์ชั่นดังต่อไปนี้

AR(ARP) + IR(7-0)——> AR(ARP) บวกตำแหน่งของ AR ปัจจุบันด้วยค่าขนาด 8 บิท

AR(ARP) - IR(7-0) ——> AR(ARP) ลบตำแหน่งของ AR ปัจจุบันด้วยค่าขนาด 8 บิท

AR(ARP) + rcAR0 ——> AR(ARP) กลับบิทตัวชี้แล้วทำการบวก AR0 ด้วย reverse-carry (rc)

AR(ARP) - rcAR0 ——> AR(ARP) กลับบิทตัวชี้แล้วทำการลบ AR0 ด้วย reverse-carry (rc)

ส่วน Auxiliary Register Point Buffer (ARB) ดังแสดงในรูปที่ 3-6 ซึ่งเป็นรีจิสเตอร์ขนาด 3 บิทนั้นจะใช้สำหรับเก็บค่าของ ARP ในขณะที่เกิดการอินเตอร์รัพท์ หรือมีการเรียกใช้โปรแกรมย่อย

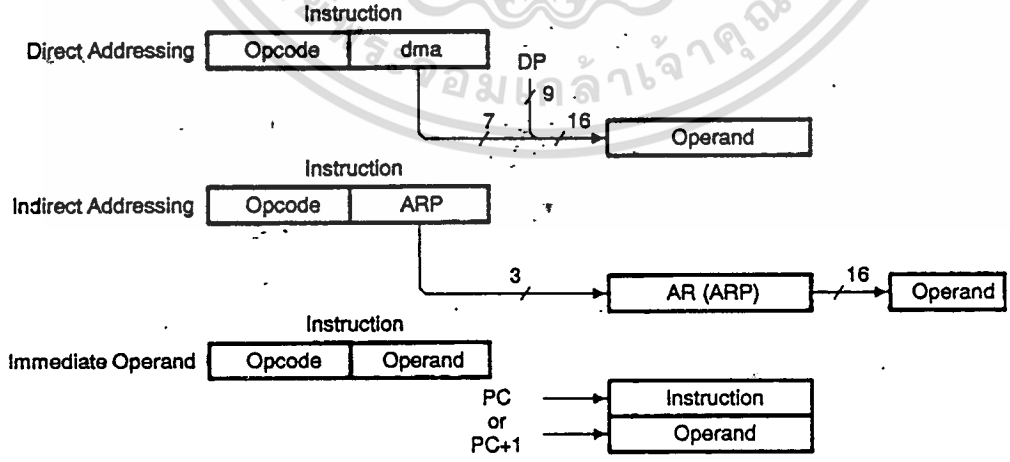
2.5.6 การอ้างแอดเดรสของหน่วยความจำ

TMS320C26 สามารถอ้างแอดเดรสของหน่วยความจำโปรแกรม ได้ทั้งหมด 64 กิโลเวิร์ด โดยส่วนหนึ่งจะถูกใช้เป็นรอมภายในตัวชิพเมื่ออยู่ในโหมดไมโครคอมพิวเตอร์ และนอกจากนี้ยังสามารถอ้างแอดเดรสของหน่วยความจำข้อมูลได้อีก 64 กิโลเวิร์ดโดยแอดเดรสในส่วนนี้จะถูกใช้เป็นแรมข้อมูลภายในตัวชิพส่วนหนึ่ง

ในการใช้ 16-Bit Data Address Bus (DAB) อ้างแอดเดรสของหน่วยความจำข้อมูลนั้นสามารถทำได้โดยใช้วิธีหนึ่งวิธีใด ดังต่อไปนี้

1. โดยการใช้ direct address bus (DRB) ซึ่งจะใช้ในโหมดการอ้างแอดเดรสแบบโดยตรง

2. โดยการใช้ auxiliary register file bus (AFB) ซึ่งจะใช้ในโหมดการอ้างแอดเดรสแบบทางอ้อมสำหรับในโหมดการอ้างแอดเดรสแบบทันทีนั้น โอเปอร์แรนด์ที่อยู่ใน PC จะถูกใช้เป็นแอดเดรสในโหมดการอ้างแอดเดรสแบบโดยตรงนั้นจะใช้ 9 บิทของ Data Memory Page Pointer (DP) เป็นตัวชี้ในหน้าหนึ่งหน้าใดจากทั้งหมด 512 หน้า ซึ่งในแต่ละหน้าจะมีจำนวน 128 Words สำหรับ Data Memory address (dma) จะใช้ 7 บิททางด้านต่ำของคำสั่งเป็นตัวชี้คำในแต่ละหน้านั้น ดังนั้นแอดเดรสที่ปรากฏบน DRB จึงประกอบด้วย DP จำนวน 9 บิทกับ dma จำนวน 7 บิทในโหมดการอ้างแอดเดรสแบบทางอ้อมนั้น การเลือกแอดเดรสของ AR (ARP) นั้น หน่วยความจำข้อมูลจะทำการผ่านค่าไปยัง AFB ขณะที่ทำการเลือก AR แล้วข้อมูล และแอดเดรสของหน่วยความจำข้อมูลจะถูกเคลื่อนย้ายโดย CALU และค่าที่อยู่ใน AR จะถูกเคลื่อนย้ายผ่าน ARAU



รูปที่ 2-8 ส่วนประกอบในคำสั่งซึ่งใช้ในการอ้างแอดเดรสแบบต่างๆ

2.5.7 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำ

TMS320C26 มีคำสั่งซึ่งใช้ในการเคลื่อนย้ายข้อมูลเป็นบล็อกเพื่อใช้สำหรับจัดการกับข้อมูลหรือโปรแกรม ซึ่งฟังก์ชั่นการเคลื่อนย้ายข้อมูลนี้จะเป็นประโยชน์อย่างมากสำหรับแรมภายในชิพ

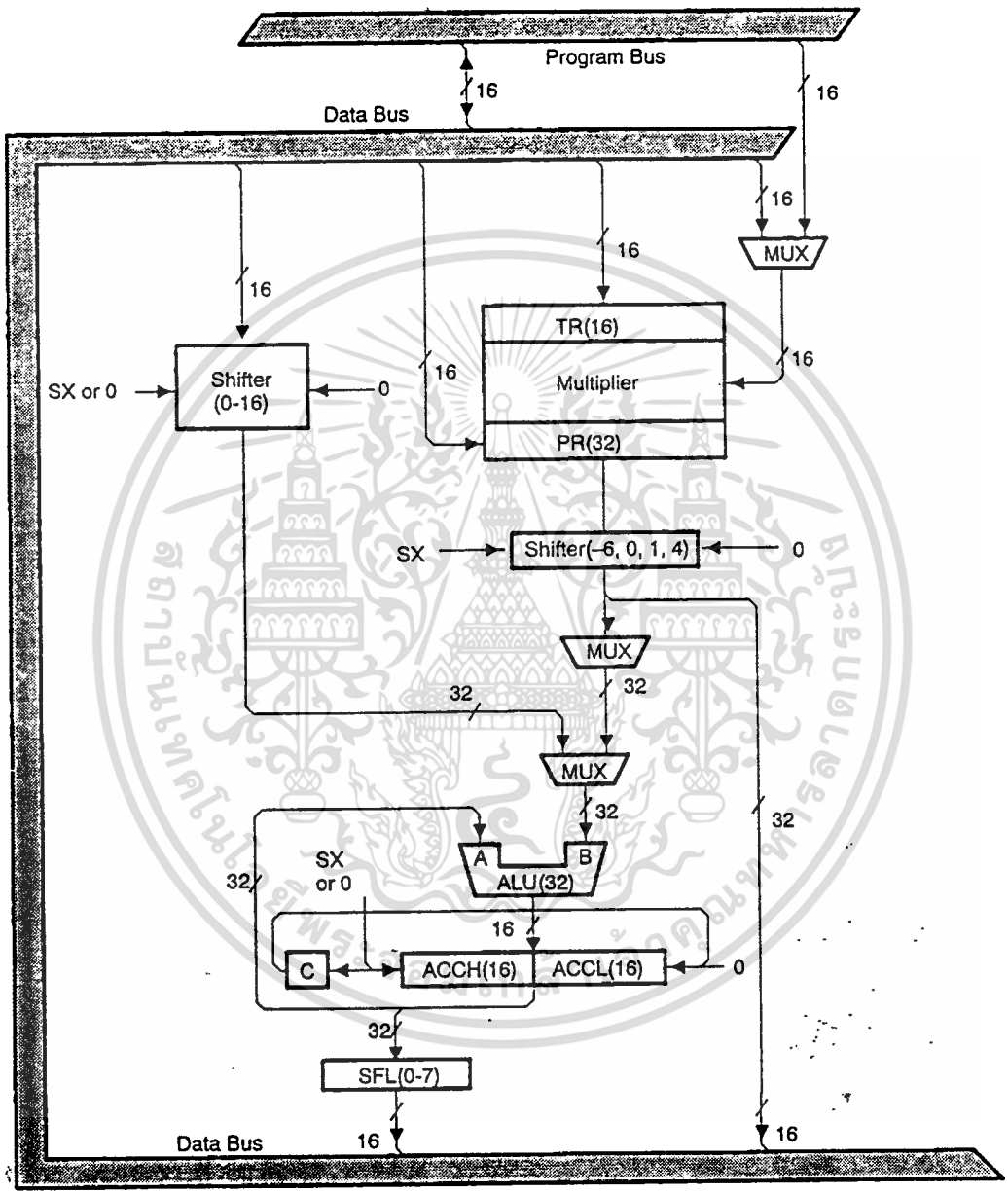
สำหรับคำสั่ง BLKD (block move from Data Memory to Data Memory) จะใช้ในการเคลื่อนย้ายข้อมูลภายในหน่วยความจำข้อมูล ส่วนคำสั่ง BLKP (block move from Program Memory to Data Memory) จะใช้สำหรับในการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำโปรแกรมไปยังหน่วยความจำข้อมูล เมื่อใช้งานร่วมกับคำสั่ง RPT/RPTK คำสั่ง BLKD/BLKP สามารถจะทำการเคลื่อนย้ายข้อมูลจากหน่วยความจำภายในหรือภายนอกชิพได้

2.5.8 หน่วยศูนย์กลางในการคำนวณทางคณิตศาสตร์และลอจิก

หน่วยศูนย์กลางในการคำนวณทางคณิตศาสตร์และลอจิกภายใน TMS320C26 นั้นจะประกอบไปด้วยสเกลลิ่งชิฟเตอร์ขนาด 16 บิท , ตัวคูณแบบขนานขนาด 16x16 บิท , ALU ขนาด 32 บิท , ACC ขนาด 32 บิท และตัวเลื่อนบิทที่ต่ออยู่กับ ACC และ มัลติพลายเออร์ (Multiplier) จากรูปที่ 3-8 จะแสดงถึงบล็อกไดอะแกรมของส่วนประกอบภายในต่างๆ ของ CALU

ลำดับขั้นต่างๆ ที่เกิดขึ้นภายในของ CALU หลังจากที่มีการใช้คำสั่งที่เกี่ยวกับ ALU

1. ข้อมูลต่างๆ ที่ต้องการประมวลผลจะถูกเฟ็ทช์จากแรมมาบนบัสข้อมูล
2. ข้อมูลต่างๆ จะผ่านมายังสเกลลิ่งชิฟเตอร์ และ ALU เพื่อมาทำการประมวลผล
3. ผลลัพท์ที่ได้จากการประมวลผลจะถูกนำไปเก็บไว้ในแอคคิวมูเลเตอร์ โดยอินพุทส่วนหนึ่งของ ALU จะได้มาจากแอคคิวมูเลเตอร์ ส่วนอินพุทที่เหลืออาจจะโอนย้ายมาจากรีจิสเตอร์ที่ใช้เก็บผลคูณ (PR) ของมัลติพลายเออร์ หรือมาจากสเกลลิ่งชิฟเตอร์ซึ่งจะโหลดค่ามาจากหน่วยความจำข้อมูล



รูปที่ 2-9 หน่วยศูนย์กลางในการคำนวณทางคณิตศาสตร์และลอจิก

2.5.9 สเกลลิ่งชิฟเตอร์

ในการเลื่อนบิทของ TMS320C26 นั้น จัดให้มีสเกลลิ่งชิฟเตอร์ซึ่งมีอินพุท ขนาด 16 บิท ต่ออยู่กับบัสข้อมูล และเอาท์พุทขนาด 32 บิทซึ่งต่ออยู่กับ ALU โดยสามารถทำการเลื่อนบิทของข้อมูลอินพุทไปทางซ้ายได้ 0-16 บิท โดยบิทซึ่งมีนัยสำคัญต่ำ (LSBs) จะถูกเติมด้วย 0 และบิทที่มีนัยสำคัญสูง (MSBs) อาจจะถูกเติมด้วย 0 หรือใช้เป็นส่วนขยายเครื่องหมาย โดยจะขึ้นอยู่กับสถานะที่บิท SXM (Sign Extension Mode) ของรีจิสเตอร์สถานะ ST1

นอกจากนี้ภายใน TMS320C26 ดังประกอบไปด้วยตัวเลื่อนบิทอื่นๆ อีกซึ่งจะมีการใช้งานในด้านต่างๆ เช่น Bit Extraction , การคำนวณแบบที่ต้องการความแม่นยำสูง , และป้องกันการเกิดค่าเกิน (Overflow) โดยตัวเลื่อนบิทเหล่านี้จะต่ออยู่ที่เอาท์พุทของมัลติพลายเออร์ และที่แอคคิวมูเลเตอร์

2.5.10 แอคคิวมูเลเตอร์ และ ALU

ALU จะเป็นหน่วยทำงานทางคณิตศาสตร์และลอจิกทั่วไปที่มีขนาด 32 บิท ซึ่งสามารถทำการบวกลบและทำงานทางลอจิกได้ โดยแอคคิวมูเลเตอร์จะเป็นตัวทำงานทั้งก่อนและหลังเสมอ การทำงานของลอจิกจะแสดงดังตารางที่ 2-2 ค่าในหน่วยความจำข้อมูลจะใช้กระทำกับบิททางด้านต่ำของแอคคิวมูเลเตอร์ (บิท = 15 ถึง บิท = 0) ส่วนค่า Zero จะใช้กระทำกับค่าทางด้านบิทสูงของแอคคิวมูเลเตอร์

FUCTION	ACCUMULATOR RESULT	
	ACC BITS 31 THROUGH 16	ACC BITS 15 THROUGH 0
XOR	(zero) + (ACC bits 31-16)	(data memory value) + (ACC bits 15-0)
AND	(zero) . (ACC bits 31-16)	(data memory value) . (ACC bits 15-0)
OR	(zero) + (ACC bits 31-16)	(data memory value) + (ACC bits 15-0)

ตารางที่ 2-2 ผลลัพธ์ที่เกิดในแอคคิวมูเลเตอร์

แอคคิวมูเลเตอร์จะเป็นตัวเก็บผลลัพธ์ของ ALU หรืออาจจะเก็บตัวตั้งของ ALU แอคคิวมูเลเตอร์นี้จะมีขนาด 32 บิท และถูกแบ่งออกเป็น 2 ส่วน คือ บิทบน (บิท 31-16) และบิทล่าง (บิท 15-0) การเก็บค่าในบิทบนหรือบิทล่างทำได้โดยคำสั่ง SACH และ SACL

สถานะของการเกิดค่าเกินที่แอคคิวมูเลเตอร์นั้น สามารถดูได้จากแฟลกรีจิสเตอร์ที่แสดงค่าเกิน (OV) ซึ่งรีจิสเตอร์นี้จะเซตเมื่อเกิดค่าเกินขึ้นในแอคคิวมูเลเตอร์ ซึ่งประโยชน์ของ OV จะใช้ในการทำคำสั่งที่มีการตัดสินใจของ DSP ซึ่งคำสั่งต่างๆ เหล่านี้แสดงดังตารางที่ 2-3

instruction	ACCUMULATOR COMDITION TESTED
BLZ	< 0
BLEZ	< 0
BGZ	> 0
BGEZ	> 0
BNZ	< > 0
BZ	= 0

ตารางที่ 2-3 การตรวจสอบสถานะของแอคคิวมูเลเตอร์

2.5.11 ตัวคูณ , รีจิสเตอร์ T และ P

สำหรับการคูณเลขขนาด 16x16 บิท นั้นสามารถจะทำการคูณได้ภายในไซเกิลเดียว (ใช้เวลา 100 นาโนวินาที) ในการคูณนั้นจะประกอบไปด้วย 3 ส่วนใหญ่ๆ ด้วยกัน คือ T, P รีจิสเตอร์และมัลติพลายเออร์แอร์แรย์โดยรีจิสเตอร์ T จะใช้สำหรับเก็บตัวตั้งขนาด 16 บิท ส่วนรีจิสเตอร์ P จะใช้เก็บผลคูณขนาด 32 บิท

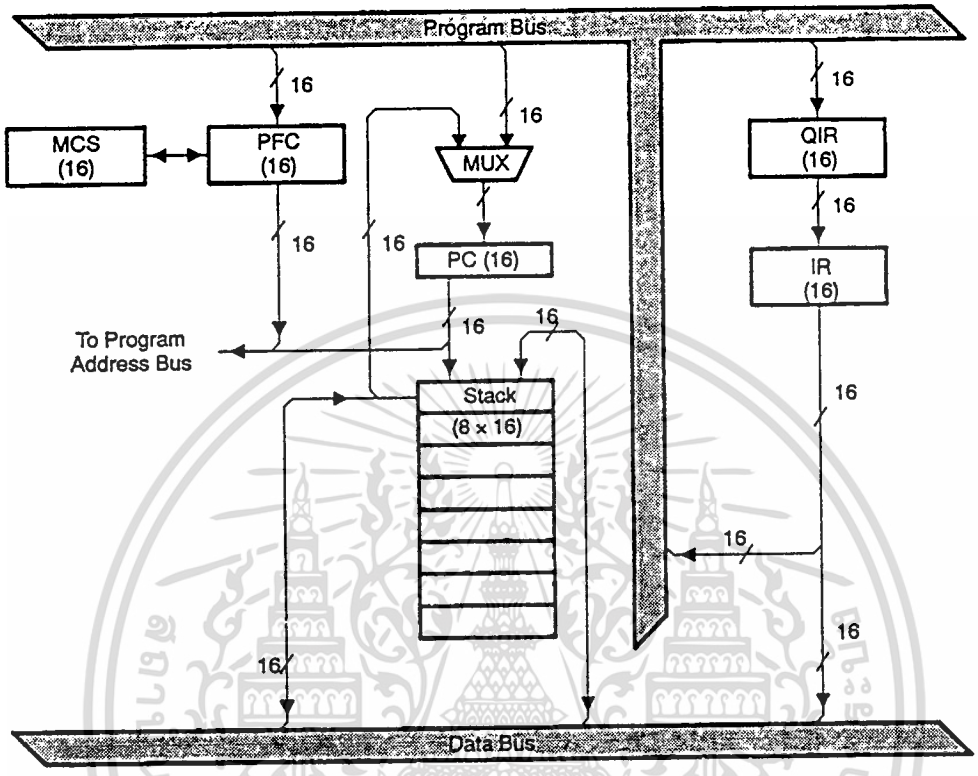
2.5.12 โปรแกรมเคาน์เตอร์และสแตค

โปรแกรมเคาน์เตอร์ (PC) และสแตค นั้นจะมีประโยชน์ในการทำงานดังนี้ การกระโดดข้ามตำแหน่งในการทำงาน, การเรียนโปรแกรมย่อย, การอินเตอร์รัพท์และการทำคำสั่ง TBLR หรือ TBLW (อ่าน , เขียนตาราง)

โปรแกรมเคาน์เตอร์ (PC) เป็นรีจิสเตอร์ขนาด 16 บิท ที่จะเก็บตำแหน่งของหน่วยความจำโปรแกรมของคำสั่งต่อไปที่จะถูกกระทำซึ่งค่าใน PC นี้จะเป็น "0" ทุกครั้งที่มีการรีเซตที่ตัว DSP ซิพ

นอกจากนี้ค่าของ PC สามารถเปลี่ยนแปลงค่าได้โดย การใช้คำสั่งกระโดด (Branch) ซึ่งจะทำให้ค่าของ PC เพิ่มหรือลดตามค่าที่กำหนดในโปรแกรม

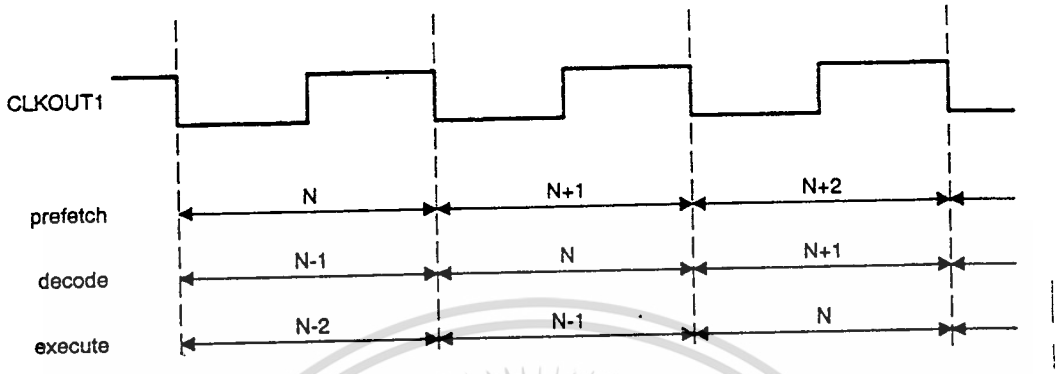
สแต็คมีขนาด 16 บิท และแบ่งออกเป็น 8 ระดับ เมื่อใช้คำสั่ง PUSH จะมีผลให้ DSP เก็บค่า 16 บิท ด้านต่ำของแอคคิวมูเลเตอร์ลงชั้นบนสุดของสแตค (TOS) และถ้าใช้คำสั่ง POP จะเป็นการนำค่า 16 บิท ของ TOS มาไว้ในแอคคิวมูเลเตอร์ ถ้าหากว่าใช้คำสั่ง SACL แล้วจะเป็นการนำค่าจาก TOS ไปเก็บไว้ในหน่วยความจำข้อมูล ซึ่งเป็นประโยชน์ในการเพิ่มความจุของสแตค



รูปที่ 2-10 โครงสร้างของสแตค และโปรแกรมเคาน์เตอร์

2.5.13 ปฏิบัติการไปป์ไลน์ (Pipeline Operation)

จากรูปที่ 2-11 แสดงให้เห็นถึงการซ้อนทับของการทำงานในการเฟ็ทช์ และเอ็กซีคิ้วซ์ที่ขอบขาลงของ CLKOUT , ค่าของ PC จะถูกโหลด (PC2) เพื่อที่จะทำตามคำสั่งเมื่อกระบวนการถอดรหัสคำสั่งแรก (Execute 1) เริ่มขึ้น เช่นเดียวกัน คำสั่งที่ 3 (Fetch 2) เมื่อกระบวนการทำงานตาม คำสั่งที่ 1 (Execute 1) เริ่มขึ้น เช่นเดียวกับคำสั่งที่ 3 (Fetch 3) จะเริ่มเมื่อมีการกระทำตามคำสั่งที่ 2 (Execute 2) ได้เริ่มขึ้น ซึ่งจะเรียกลักษณะการทำงานเช่นนี้ว่า "Pipeline-Operation"



รูปที่ 2-11 สถาปัตยกรรม HAVARD

2.5.14 รีเซต (Reset)

รีเซต (RS) เป็นสัญญาณอินเตอร์รัพท์จากภายนอกโดยเป็นสัญญษณอินเตอร์รัพท์ชนิด "นอนมาสเคเบิล" ซึ่งจะเกิดขึ้นได้ทุกๆ ขณะและ TMS320C26 สามารถที่จะทำการตรวจสอบได้ การนำสัญญาณรีเซตไปประยุกต์ใช้งานตามตัวอย่างนี้มีการใช้งานเมื่อเริ่มเปิดเครื่องใหม่

เมื่อมีสัญญาณเข้ามาที่ขา RS จะมีผลให้โปรแกรมเคาน์เตอร์มีค่าเป็น "0" และมีการเปลี่ยนแปลงเกิดขึ้นที่รีจิสเตอร์และบิทสถานะ สัญญาณรีเซตนี้จะต้องเกิดขึ้นอย่างน้อย 3 สัญญาณนาฬิกา เพื่อเป็นการยืนยันการรีเซตของอุปกรณ์

เมื่อ TMS320C26 ได้รับสัญญาณรีเซตจะมีการปฏิบัติตามกระบวนการ ดังต่อไปนี้

1. ที่บิท CNF ของรีจิสเตอร์สถานะ ST1 จะถูกโหลดด้วย "0" ซึ่งจะมีผลให้แรมทั้งหมดมีลักษณะเป็นหน่วยความจำข้อมูล
2. โปรแกรมเคาน์เตอร์จะถูกเซตให้เป็น 0 แอดเดรสบัส A15-A0 ก็จะมีค่าเป็น "0" ด้วย
3. บัสข้อมูล D15-D0 จะอยู่ในสถานะเป็นอิมพิแดนซ์สูง
4. สัญญาณที่ใช้ในการควบคุมต่างๆ (/PS ,/DS ,/IS ,R/w ,/STRB และ /BR) จะมีสถานะเป็น "1"
5. สัญญาณอินเตอร์รัพท์อื่นๆ จะถูกดิสเอเบิลโดยการเซตบิท INTM ให้เป็น "1" และอินเตอร์รัพท์แฟลกรีจิสเตอร์ (IFR) จะถูกรีเซตให้เป็น "0" ทั้งหมด
6. ที่บิทสถานะต่างๆ จะมีผลดังนี้

1 —> SXM , 0 —> HM , 0 —> FO , 1 —> C,

และ 1 —> FSM

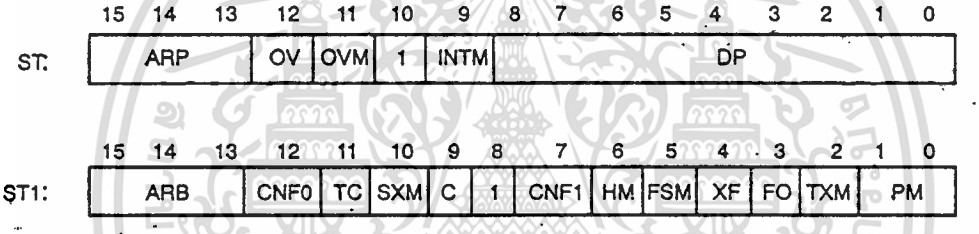
7. ข้อมูลต่างๆ ใน RPTC จะถูกเคลียร์ให้เป็น "0"

8. ขา DX (Data Transmit) จะอยู่ในสถานะเป็นอิมพิแดนซ์สูง

9. รีจิสเตอร์ TIM จะถูกเซตให้เป็นค่าสูงสุด (>FFFF) และเมื่อสัญญาณ RS ถูกเปลี่ยนกลับไปอยู่ในสถานะ "1" ก็จะมีการเริ่มการทำงานใหม่ โดยเริ่มจากพื้นที่ "0" ของโปรแกรม

2.5.15 รีจิสเตอร์สถานะ (Status Register)

ในรีจิสเตอร์สถานะทั้งสอง คือ ST0 และ ST1 จะประกอบไปด้วยเงื่อนไขและโหมดต่างๆ ในการทำงานซึ่งสามารถที่จะเปลี่ยนแปลงได้ โดยรีจิสเตอร์สถานะนี้สามารถเก็บค่าต่างๆ ไว้ในหน่วยความจำข้อมูลและสามารถโหลดค่าออกมาได้ ในการเขียนค่ากับบิทสถานะต่างๆ เหล่านี้สามารถทำได้โดยการใช้คำสั่ง LST/LST1 และ SST/SST1



รูปที่ 2-12 การจัดรูปแบบของรีจิสเตอร์สถานะ (Status Register)

จากรูปที่ 2-12 แสดงถึงการจัดบิทต่างๆ ภายในรีจิสเตอร์สถานะทั้งสอง และจากรูปที่ 2-12 จะเห็นว่ามีการแยกส่วนต่างๆ ของรีจิสเตอร์ DP , ARP , ARB ออกมาให้เห็นอย่างชัดเจนเพราะว่ารีจิสเตอร์เหล่านี้ไม่มีคำสั่งที่ใช้ในการแยกเก็บค่าเหล่านี้ลงในแรมในส่วนอื่นๆ ของคำสั่งหรือฟังก์ชั่นที่อาจจะมีผลต่อบิทสถานะต่างๆ เหล่านี้ สามารถจะอธิบายได้ดังนี้

ARB (Auxiliary Register Pointer Buffer) เมื่อใดก็ตามที่มีการโหลดค่าเข้ามายัง ARP ค่าเก่าของ ARP จะถูกคัดลอกมายัง ARB ยกเว้นที่มีการใช้คำสั่ง LST เมื่อ ARB ถูกโหลดผ่านคำสั่ง LST1 ค่าที่เหมือนกันจะถูกคัดลอกไปยัง ARP

ARP (Auxiliary Register Pointer) ซึ่งมีขนาด 3 บิทใช้ในการเลือก AR โดยใช้ในการอ้างแอดเดรสแบบทางอ้อม ARP อาจจะมีการเปลี่ยนแปลงโดยการใช้คำสั่งอ้างอิงหน่วยความจำ เมื่อใช้การอ้างแอดเดรสแบบทางอ้อมด้วยคำสั่ง LARP , MAR และ LSP โดย ARP จะถูกโหลดด้วยค่าที่เหมือนกันกับใน ARB เมื่อใช้คำสั่ง LST1 ในการเอ็กซีคิ้วซ์ (Executed)

C บิทตัวทดนี้จะถูกเซตเป็น "1" ถ้าผลลัพธ์จากการบวกเกิดมีตัวทด และจะคำสั่งเป็น "0" เมื่อผลลัพธ์ที่เกิดจากการลบมีการขอยืม นอกจากกรณีที่กล่าวมาแล้ว มันจะเกิดการรีเซตหลังจากที่ใช้คำสั่งการบวก หรือรีเซตหลังจากการที่ใช้คำสั่งการลบ ยกเว้น ถ้าใช้คำสั่ง ADDH หรือ SUBH คำสั่ง ADDH สามารถทำให้เกิดการเซตและคำสั่ง SUBH จะทำให้เกิดการรีเซตกับบิทตัวทด นอกจากนี้คำสั่งที่ใช้ในการเลื่อนบิทหรือหมุนบิทก็จะมีผลต่อบิทตัวทดนี้ เช่น คำสั่ง SC , RC และ LST1 และเมื่อมีสัญญาณรีเซตบิทนี้จะถูกเซตเป็น "1"

CNF บิทนี้จะใช้ในการควบคุมการใช้แรมบนตัวชิพ ถ้าบิทนี้ถูกเซต "0" บล็อก BO จะถูกกำหนดให้เป็นหน่วยความจำข้อมูล นอกเหนือจากนี้บล็อก BO จะถูกกำหนดให้เป็นหน่วยความจำโปรแกรมบิท CNF นี้อาจจะมีการเปลี่ยนแปลงได้โดยการใช้คำสั่ง CNFD , CNFP และ LST1 และเมื่อมีสัญญาณรีเซตบิทนี้จะถูกเซตเป็น "0"

DP (Data Memory Page Pointer) 9 บิทของ DP นี้จะมีความสัมพันธ์กับ 7 บิททางด้านต่ำของเวิร์ดคำสั่ง ซึ่งเมื่อรวมกันแล้วจะเป็นรูปแบบของการอ้างแอดเดรสแบบโดยตรงขนาด 16 บิท DP นี้อาจจะมีการเปลี่ยนแปลงได้โดยการใช้คำสั่ง LST , LDP และ LDPK

FO บิทนี้แสดงการจัดรูปแบบเมื่อบิทนี้ถูกเซตเป็น "0" รีจิสเตอร์ของพอร์อนุกรมจะมีลักษณะเป็นรีจิสเตอร์ขนาด 16 บิท และเมื่อถูกเซตเป็น "1" รีจิสเตอร์ที่พอร์ตจะมีหน้าที่รับและส่งข้อมูลขนาด 8 บิทไบต์ บิท FO นี้อาจจะมีการเปลี่ยนแปลงได้โดยการใช้คำสั่ง FORT และ LST1 เมื่อมีสัญญาณรีเซตบิทนี้จะถูกเซตเป็น "0"

FSM (Frame Synchronization Mode Bit) บิทนี้ใช้ในการแสดงปฏิบัติการของพอร์ตอนุกรมด้วยสัญญาณเฟรมซิงค์โดยใช้พัลส์จากภายนอก เมื่อ FSM="1" จะมีการปฏิบัติการตามสัญญาณพัลส์ของเฟรมซิงค์ ซึ่งเป็นอินพุทของ FSX/FSR และถ้า FSM=0 จะไม่สนใจอินพุทของ FSX/FSR และการปฏิบัติการบนพอร์ตอนุกรมจะทำต่อไปโดยไม่ต้องการเฟรมซิงค์พัลส์ และเมื่อมีสัญญาณรีเซตบิทนี้จะถูกเซตเป็น "1"

HM บิทแสดงสภาวะโฮลด์เมื่อ HM="1" ขบวนการต่างๆ ภายในก็จะหยุดเอ็กซ์ซีคิ้ว และถ้า HM="0" ขบวนการต่างๆ ก็จะปฏิบัติการต่อจากที่หยุดไว้บิทซึ่งจะถูกเซตเป็น "1" เมื่อมีสัญญาณรีเซต

INTM บิทนี้ใช้แสดงสถานะของการอินเตอร์รัพท์ เมื่อถูกเซตเป็น "0" สัญญาณก็จะได้รับอีนาเบิลแต่ถ้าบิทนี้ถูกเซตเป็น "1" สัญญาณอินเตอร์รัพท์แบบมาสเคเบิลทั้หมดก็จะถูกดิสเอเบิล INTM นี้จะถูกเซตและรีเซตโดยใช้คำสั่ง DINT และ EINT นอกจากนี้ในการใช้งานคำสั่ง LST จะไม่เกิดผลใดๆ ต่อบิท INTM นี้

OV บิทแสดงการเกิดค่าเกินบิท OV นี้จะเซตเป็น "1" เมื่อเกิดค่าเกินขึ้นใน ALU นอกจากนี้ถ้าต้องการเคลียร์ค่าต่างๆ ใน OV เราก็สามารถทำได้โดยการใช้คำสั่ง BV , BNV และ LST

OVM บิทแสดงสถานะเมื่ออยู่ในโหมดค่าเกินบิทนี้จะมีสถานะเป็น "0" ในกรณีที่เกิดการยกเลิกโหมดนี้ ในการเซตและรีเซตค่าที่บิทนี้เราจะใช้คำสั่ง SOVM และ ROVM ตามลำดับนอกจากนี้คำสั่ง LST ก็จะมีผลให้เกิดการเปลี่ยนแปลงที่บิทนี้

PM โหมดการเลื่อนของผลคูณ ถ้าสองบิทนี้มีค่าเป็น 00 จะเป็นการโหลดค่าตัวคูณขนาด 32 บิท เข้าไปใน ALU โดยไม่มีการเลื่อน ถ้า PM=01 เอาท์พุทของ PR จะถูกเลื่อนไปทางซ้าย 1 ครั้งก่อนโหลดเข้าไปใน ALU โดยบิทที่มีนัยสำคัญต่ำ (LSBs) จะถูกเติมด้วย 0 ถ้า PM=10 เอาท์พุทของ PR จะถูกเลื่อนไปทางซ้าย 4 บิทก่อนจะถูกโหลดเข้าไปใน ALU โดยบิทที่มีนัยสำคัญต่ำ (LSBs) จะถูกเติมด้วย 0 และเมื่อ PM=11 ผลจากการคูณจะถูกเลื่อนไปทางขวา 6 บิท บิท PM นี้สามารถทำการโหลดค่าเข้าไปได้โดยใช้คำสั่ง SPM และ LST1 เมื่อเกิดสัญญาณ RS ที่บิทนี้จะถูกเคลียร์

SXM บิทแสดงเครื่องหมาย ถ้า SXM="1" ผลที่เกิดจากการคูณจะมีการคิดเครื่องหมาย แต่ถ้า SXM="0" จะไม่มีการคำนึงถึงเครื่องหมาย ตัวอย่างเช่น คำสั่ง ADDS เป็นคำสั่งการบวกโดยไม่คิดเครื่องหมาย และไม่คำนึงถึงผลของ SXM การจะเซตหรือรีเซตบิทนี้สามารถทำได้โดยการใช้คำสั่ง SSXM และ RSXM นอกจากนี้ยังสามารถใช้คำสั่ง LST1 ในการโหลดค่าได้ด้วย บิท SXM จะเซตเป็น "1" เมื่อเกิดสัญญาณรีเซต

TC บิททดสอบและควบคุมแฟลก บิท TC นี้จะมีผลเมื่อมีการเรียกใช้คำสั่ง BIT , BITT , CMPR , LST1 และ NORM บิทนี้จะเซตเป็น "1" ถ้าการทดสอบบิท (Test Bit) ด้วยคำสั่ง BIT หรือ BITT เป็น "1" หรือเมื่อมีการทดสอบด้วยคำสั่ง NORM แล้ว Exclusive-ORฟังก์ชั่น ของสองบิทที่มีนัยสำคัญสูงของแอคคิวมูเลเตอร์จากการเปรียบเทียบกันแล้วเป็นจริง

TXM บิทนี้ใช้แสดงสถานะการส่งเมื่อ TXM="1" ขา FSX ของพอร์ตอนุกรมจะถูกกำหนดให้เป็นเอาท์พุท ในโหมดนี้เมื่อ DXR ถูกโหลดค่าจะมีการสร้างพัลส์ให้กับ FSX การส่งจะเริ่มจากที่ขา DX ถ้า TXM="0" ขา FSX จะมีลักษณะเป็นอินพุท บิท TXM นี้จะสามารถเซตหรือรีเซตได้โดยการใช้คำสั่ง STXM และ RTXM เมื่อมีสัญญาณรีเซตบิท TXM นี้จะเป็น "0"

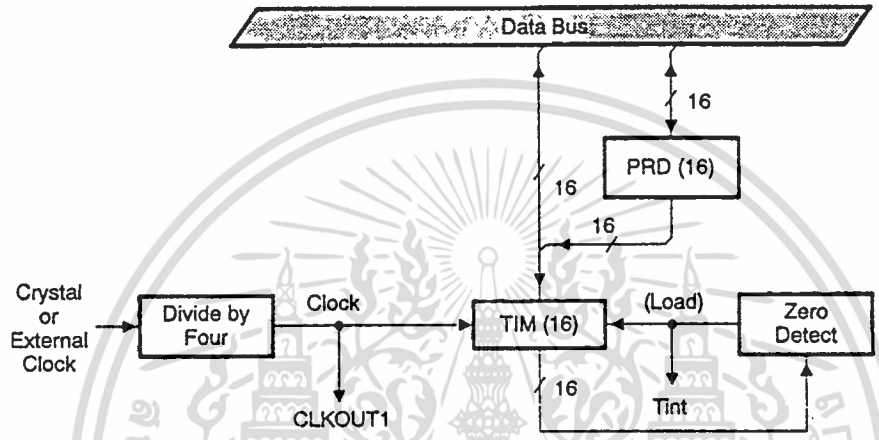
XF ที่บิทนี้จะเป็นการแสดงสถานะของขา XF ซึ่งโดยทั่วไปแล้วจะทำหน้าที่เป็นขาเอาท์พุทบิท XF นี้จะทำการเซตหรือรีเซตได้โดยการใช้ คำสั่ง SXF และ RXF หรืออาจจะโหลดค่าโดยการใช้คำสั่ง LST1 บิท XF นี้เมื่อเกิดสัญญาณรีเซตจะถูกเซตให้เป็น "1"

2.5.16 ปฏิบัติการเวลา

การจัดผังหน่วยความจำของ TMS320C26 กำหนดให้รีจิสเตอร์ไทม์เมอร์ (TIM) และ พีเรียด (PRD) มีขนาด 16 บิท ซึ่งแสดงดังในรูปที่ 2-20 ส่วนวงจรนาฬิกาภายในตัวชิพจะใช้สัญญาณนาฬิกา (Clock) จาก CLKOUT1

ในขณะที่เกิดสัญญาณรีเซตนั้นรีจิสเตอร์ TIM และรีจิสเตอร์ PRD นี้จะถูกเซตให้มีค่าสูงสุด (> FFFF) และจะลดค่าลงหลังจากมีการถอนสัญญาณรีเซต จากที่กล่าวมานี้รีจิสเตอร์TIM และ PRD ยังสามารถที่จะโหลดค่าใหม่ได้โดยใช้การโปรแกรมคอนโทรลสำหรับ TIM รีจิสเตอร์หน่วยความจำข้อมูลพื้นที่ 2 ไทม์เมอร์จะเริ่มนับที่มุกๆ N X CLKOUT1 เมื่อ N="1" รีจิสเตอร์ TIM นี้จะลดค่าลดหนึ่งส่วนรีจิสเตอร์ PRD และหน่วยความจำข้อมูลพื้นที่ 3 จะเริ่มนับโดยวงจรนาฬิกาในส่วนไทม์เมอร์อินเตอร์รัพท์ (TINT) จะสร้างสัญญาณออกมาตลอดเวลาที่ไทม์เมอร์เริ่มลดค่าลงไปถึง 0 ไทม์เมอร์นี้เราสามารถที่จะโหลดค่าเข้าไปใหม่ได้โดยใช้ค่าที่อยู่ใน

รีจิสเตอร์ PRD ภายในไซเกิลถัดไปหลังจากที่ลดค่าลงจนถึง 0 ไทม์เมอร์ และ พีเรียด รีจิสเตอร์สามารถที่จะอ่านหรือเขียนได้ภายในไซเกิลหนึ่งไซเกิลใดก็ได้ ซึ่งสามารถตรวจค่าข้อมูลได้โดยการอ่านค่าจากรีจิสเตอร์ TIM นี้ และถ้าไม่มีการใช้งานวงจรนาฬิกาเราสามารถที่จะทำให้ TINT นี้ดิสเคเบิลต่อสัญญาณอินเตอร์รัพท์ชนิดมาสเคเบิลได้ โดยการใช้คำสั่ง DINT จุดประสงค์ทั่วไปของรีจิสเตอร์ PRD คือใช้เป็นพื้นที่ของหน่วยความจำข้อมูล และถ้ามีการใช้งาน TINT รีจิสเตอร์ TIM และ PRD จะถูกใช้เป็นโปรแกรมก่อนที่จะถอนสัญญาณจาก TINT



รูปที่ 2-13 บล็อกไดอะแกรมของวงจรตั้งเวลา

2.5.17 การนับที่ทำให้เกิดการทำงานซ้ำ (Repeate Counter)

วงจรนับที่ทำให้เกิดการทำงานซ้ำ (RPTC) จะเป็นวงจรนับขนาด 8 บิท เมื่อมีการโหลดด้วยค่าตัวเลข N จะมีผลให้เกิดการเอ็กซีคิ้วคำสั่งถัดไปที่เวลา N + 1 โดยค่าที่สามารถโหลดเข้าไปยัง RPTK (Repeat Immediate) และผลลัพธ์ที่สามารถจะเอ็กซีคิ้วได้คือ 256 ข้อมูลต่าง ๆ ใน RPTC นี้จะถูกเคลียร์โดยสัญญาณรีเซต

ลักษณะของการทำงานแบบซ้ำ ๆ กันนี้ จะมีใช้ในคำสั่งที่เกี่ยวกับการคูณ (MAC / MACD) การเคลื่อนย้ายข้อมูลเป็นบล็อก (BLKD / BLKP), การโอนย้ายข้อมูลระหว่าง I / O (IN / OUT) และการอ่านเขียนตาราง (TBLR / TBLW) คำสั่งเหล่านี้ในการทำงานโดยปกติแล้วจะใช้หลายไซเกิล แต่ถ้าใช้การทำงานในลักษณะซ้ำ ๆ (REPEAT) จะทำให้สามารถทำงานได้โดยใช้เพียงไซเกิลเดียว ตัวอย่าง เช่น คำสั่งในการอ่านตารางในการเอ็กซีคิ้วจะใช้ 3 ไซเกิลหรือมากกว่า แต่ถ้าใช้การทำงานลักษณะซ้ำ ๆ กันนี้ พื้นที่ในตารางสามารถที่จะทำการอ่านได้ในทุก ๆ ไซเกิล

2.5.18 โหมดเพาว์เวอร์ดาวน์

เมื่อมีการทำงานในโหมดเพาว์ดาวน์ TMS320C26 จะอยู่ในภาวะที่ไม่มีการเปลี่ยนแปลง โดยต้องการกำลังงานเพียงครึ่งหนึ่งของกำลังงานปกติเท่านั้นเพื่อจ่ายให้กับอุปกรณ์โหมดเพาว์เวอร์ดาวน์ นี้สามารถจะทำการกระตุ้นได้โดยการใช้คำสั่ง IDLE หรือทำให้สัญญาณโฮลด์มีสถานะเป็นโลว์ โดยที่บิทสถานะ HM ถูกเซตเป็น "1"

การยกเลิกโหมดเพาว์ดาวน์นี้ ทำได้โดยการใช้คำสั่ง IDLE หรือโดยการยกเลิกสัญญาณโฮลด์

บทที่ 3
ทฤษฎีการสุ่มตัวอย่าง

อาจกล่าวได้ว่า สัญญาณในธรรมชาติส่วนมาก เช่นสัญญาณเสียง สัญญาณการสั่นสะเทือนของพื้นโลก คลื่นหัวใจ หรือการแปรค่าไปของอุณหภูมิ เหล่านี้เป็นไปในลักษณะแบบต่อเนื่องกับพิสัยเวลา หรือกล่าวได้ว่าเป็นสัญญาณเชิงอุปมาน การนำสัญญาณเหล่านี้ ไปประมวลผลในลักษณะการประมวลผลสัญญาณเชิงเลข หรือ การประมวลผลสัญญาณเชิงเต็มหน่วย (Discrete Signal Processing) ได้ต้องใช้ระบบการประมวลผลตามรูปที่ 3.1 ซึ่ง ตามรูป วงจร S/H เป็นวงจรสุ่มและคงค่าสัญญาณไว้เพื่อให้วงจรแปลงอานาล็อกเป็นดิจิตอล ทำการแปลงเป็นตัวเลขอีกทีหนึ่งสังเกตุว่า ระบบการประมวลผลสัญญาณเชิงเลขเป็นส่วนของระบบที่อยู่ระหว่างจุด A และ B โดยที่สัญญาณที่จุดนี้เป็นสัญญาณที่ถูกแปลงเป็นตัวเลขแล้ว

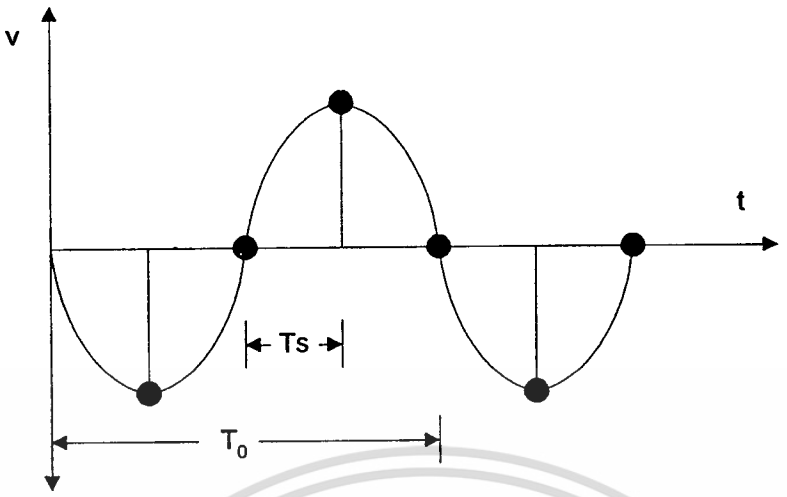
รูปที่ 3.1 ระบบการประมวลผลสัญญาณเชิงอุปมานโดยใช้ระบบประมวลผลสัญญาณเชิงเลข

อย่างไรก็ตามการเปลี่ยนสัญญาณเชิงอุปมานมาเป็นสัญญาณเชิงเลขนั้น ในเบื้องแรกต้องมีการสุ่มตัวอย่างก่อน ซึ่งความถี่ในการสุ่มตัวอย่างโดยไม่ทำให้สัญญาณสูญเสียข้อมูลสำคัญไป

โดยทั่วไปเราอาจสุ่มตัวอย่างด้วยค่าความถี่ $f_{SN} = 2f_0$ พอดี ค่าความถี่นี้มีชื่อเรียกว่า ความถี่ไนควิสต์ (Nyquist Frequency) และคาบเวลา $Tn=1/(2f_0)$ นี้เรียกว่า ช่วงเวลาสุ่มตัวอย่างไนควิสต์ (Interval)

ในทางปฏิบัติเพื่อหลีกเลี่ยงผลของ ปรากฏการณ์ ไม่เป็นเชิงเส้น (Nonlinearlity) ที่อาจเกิดจากการสุ่มตัวอย่าง เรามักใช้ความถี่ในการสุ่มตัวอย่าง $2f_s$ มากกว่าค่าความถี่ไนควิสต์หรือ f_{SN} ขึ้นไป ส่วนจะมีค่ามากกว่าเท่าใดนั้นขึ้นกับลักษณะงานไม่ได้มีการกำหนดค่าที่แน่นอน ซึ่ง อาจใช้ เป็น $2.5f_0$ หรือ $5f_0$ หรือ $10f_0$ ก็ ได้

ตามทฤษฎีการสุ่มตัวอย่างของ แชนนอน นั้นเห็นได้ว่าการจะสุ่มตัวอย่างสัญญาณได้ถูกต้องก็ต่อเมื่อต้องรู้ค่า แถบความถี่ปฏิบัติงานของสัญญาณ หรือพูดอีกนัยหนึ่ง สัญญาณต้องมีแถบความถี่ปฏิบัติงานจึงจะทำการสุ่มตัวอย่างได้ ดังนั้นในบางครั้งเพื่อให้มั่นใจว่าสัญญาณที่ทำการประมวลผล ถูกประมวลผลอย่างถูกต้อง ในภาคแรกของระบบประมวลผลเชิงเต็มหน่วยและเชิงเลข จึงอาจมีวงจรกรองผ่านความถี่ต่ำ (Low Pass Filter) ไว้เป็นตัวกำหนดแถบความถี่ปฏิบัติงานของสัญญาณ ดังแสดงในรูป



รูปที่ 3.1 แสดงตัวอย่างของความถี่ในการสุ่มสัญญาณ $Ts=1/(4f_0)$

บทที่ 4

ฟูเรียร์ (Fourier)

บทนี้เราจะศึกษาถึงการวิเคราะห์สัญญาณด้วยอนุกรมฟูเรียร์ที่เราสามารถนำมาใช้แสดงสัญญาณเป็นคาบ โดยจะให้อยู่ในรูปการการรวมเอาสัญญาณรูปคลื่นไซน์จำนวนอนันต์เข้าด้วยกัน ต่อจากนั้นก็จะมีการพัฒนาการแปลงฟูเรียร์ที่ทำงานในลักษณะเดียวกันแต่จะใช้วิเคราะห์สัญญาณที่ไม่เป็นคาบ ผลลัพธ์จากการวิเคราะห์จะให้สเปกตรัม (Spectrum) หรือแถบความถี่ของสัญญาณที่ถูกวิเคราะห์ทำให้รู้ว่าสัญญาณนั้นประกอบด้วยความถี่อะไรบ้าง

4.1 อนุกรมฟูเรียร์

ให้สัญญาณ f(t) เป็นสัญญาณที่ซ้ำๆ กันเป็นคาบทุกๆ T วินาที ซึ่งสามารถเขียนฟังก์ชั่นของสัญญาณ f(t) แทนได้ด้วยอนุกรมฟูเรียร์ กล่าวคือ

$$f(t) = a_0 + \sum_{n=1}^{N} a_n \cos nwt + \sum_{n=1}^{N} b_n \sin nwt \qquad (4.1)$$

เมื่อ a_n, b_n เป็นสัมประสิทธิ์ที่สามารถคำนวณได้จากสมการข้างล่างนี้

$$a_n = 2/T \int_{-T/2}^{T/2} f(t) \cos nwtdt \qquad (4.2)$$

$$b_n = 2/T \int_{-T/2}^{T/2} f(t) \sin nwtdt \qquad (4.3)$$

เมื่อ T เป็นคาบเวลา และ w = 2 f และเทอม dc คือ a_0 ซึ่งให้ค่าเฉลี่ยของสัญญาณ f(t) ในช่วงคาบ T คำนวณได้ดังนี้

$$a_0 = 1/T \int_{-T/2}^{T/2} f(t)\, dt \qquad (4.4)$$

ข้อสังเกตุ

1. ถ้า f(t) = f(-t) ฟังก์ชัน f(t) เป็นฟังก์ชั่นคู่ จะเกิดความสมมาตรรอบจุดกำเนิด และจะให้เพียงเทอมของ COSINE ในสมการ (2.1)

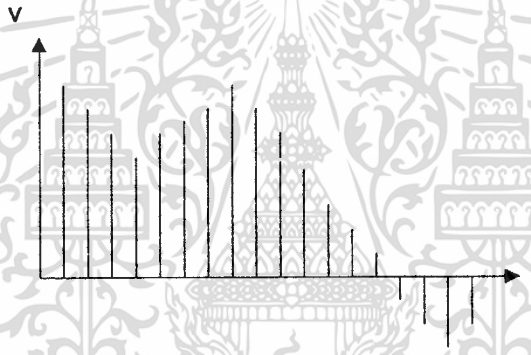
2. ถ้า f(t) = -f(-t) ฟังก์ชั่น f(t) เป็นฟังก์ชั่นคี่ จะเกิดปรากฏเพียงเทอมของ SINE เท่านั้นในสมการ (2.1)

3. ถ้า f(t+T/2) = f(t) ในสมการ (2.1) จะปรากฏเพียงเทอมฮาร์โมนิคคู่ (EVEN HARMONICS)

4. ถ้า f(t+T/2) = -f(t) ในสมการ (2.1) จะปรากฏเพียงเทอมฮาร์โมนิคคี่ (ODD HARMONICS)

4.2 แถบความถี่ที่ไม่ต่อเนื่อง (DISCRETE SPECTRUM)

อนุกรมฟูเรียร์ที่ใช้แทนฟังก์ชั่นในแกนเวลา f(t) โดยเราจะแสดงให้เห็นองค์ประกอบทางความถี่เหล่านี้รวมกันเป็นแถบความถี่ที่ไม่ต่อเนื่อง โดยขนาดของแต่ละความถี่ถูกกำหนดโดยค่าสัมประสิทธิ์ a_n และ b_n ทุกความถี่จะเป็นฮาร์โมนิคของความถี่พื้นฐานที่เท่ากับ 1/T และช่วงกว้างของความถี่ต่างๆ นี้จะเรียกว่า แบนด์วิดท์ของสัญญาณ



รูป 4.1 แถบความถี่ไม่ต่อเนื่อง

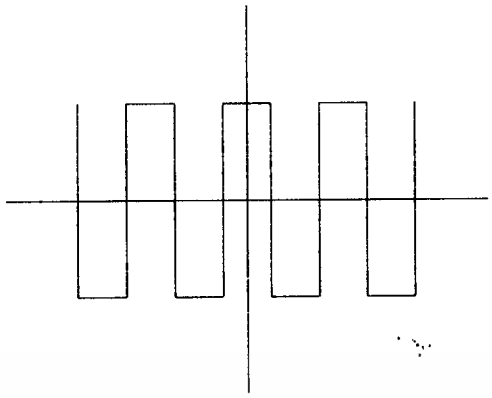
4.3 ตัวอย่างชนิดของอนุกรม (TYPICAL SERIES)

ถึงแม้ความถี่อาจจะประกอบไปด้วยความถี่ต่างๆ จำนวนอนันต์ก็ตามขนาดของความถี่จะลดลงเรื่อยๆ เมื่อความถี่สูงขึ้น (ค่าเพิ่มขึ้น) สำหรับในทางปฏิบัติแล้ว เราพิจารณาเพียงความถี่จำนวนจำกัดที่พอเพียงกับการติดต่อสื่อสาร

ข้อควรคำนึงในการประมวลผลสัญญาณเชิงเลข โดยเฉพาะในด้านการสื่อสารนั้น จุดสำคัญของการติดต่อสื่อสารคือ การประหยัดในการใช้แถบความถี่ (band with) ในระบบการติดต่อสื่อสาร ถ้าหากเรารู้ถึงแถบของความถี่จะช่วยให้ระบบการส่งและการรับสัญญาณทำได้อย่างมีประสิทธิภาพและช่วยให้ประหยัดด้วย สำหรับลักษณะรูปคลื่นสัญญาณที่ปรากฏในระบบการประมวลผลสัญญาณเชิงเลขมีหลายรูปแบบ ในที่นี้จะแสดงอนุกรมสัญญาณเฉพาะที่ใช้กันมาก

องค์ประกอบของฟูเรียร์สำหรับรูปคลื่นตัวอย่างชนิดต่างๆ พอจะยกตัวอย่างได้ดังนี้

1. รูปคลื่นสี่เหลี่ยมสมมาตร (SYMMETRICAL SQUARE WAVE)



รูปที่ 4.2 รูปคลื่นแบบสมมาตร

เนื่องจาก f(t) มีลักษณะสมมาตรทางแนวนอน ค่าเฉลี่ยของพื้นที่จึงเป็นศูนย์ ทำให้เทอม DC มีลักษณะเท่ากับศูนย์ ($a_0 = 0$) และโดยที่ f(t) = f(-t) จึงทำให้สัญญาณดังกล่าวจะมีเฉพาะเทอมของ COSINE เท่านั้น กล่าวคือ $b_n = 0$ ส่วนค่า a_n คำนวณได้ดังนี้

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos nwt\,dt$$

f(t) = -1 จาก -T/2 ถึง -T/4

= 1 จาก -T/4 ถึง T/4

= -1 จาก T/4 ถึง T/2

$$a_n = \frac{2}{T}\left\{\int_{-T/2}^{-T/4}(-1)\cos nwt\,dt + \int_{-T/4}^{T/4}(1)\cos nwt\,dt + \int_{-T/4}^{T/2}(-1)\cos nwt\,dt\right\}$$

$$= \frac{2}{T}\left\{\frac{-1}{nw}\int_{-T/4}^{T/2}\cos nwt\,dt + \frac{1}{nw}\int_{-T/4}^{T/4}\cos nwt\,dt - \frac{1}{wn}\int_{-T/4}^{T/2}\cos nwt\,dt\right\}$$

$$= \frac{2}{nwT}\{\sin nwT/4 - \sin nwT/2 + \sin nwT/4 + \sin nwT/4 - \sin nwT/2 + \sin nwT/4\}$$

$$= \frac{2}{nwT}\{4\sin nwT/4 - 2\sin nwT/2\}.$$

เนื่องจาก $wT = (2\ f).T = 2$ ดังนั้นจะได้

$$a_n = \frac{1}{n}\{4\sin nT/2 - 2\sin n\}$$

แต่ $\sin n = 0$ เมื่อ $n = 0, 1, 2 \ldots \infty$

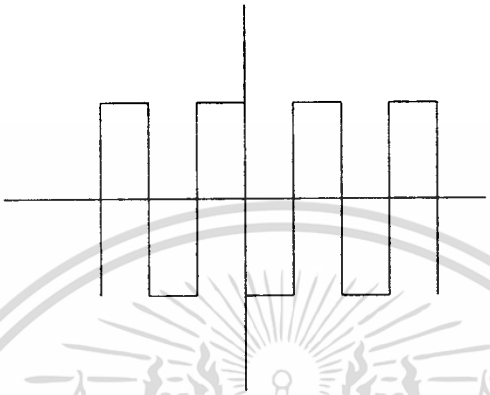
ดังนั้น $a_n = 4/n \sin nT/2$

$a_n = 4/ \sin /2 = 4/2$ $a_n = 2/ \sin = 0$

$a_n = 4/3 \sin 3 /2 = 4/3 \qquad a_n = 1/ \sin 2 = 0$

$f(1) = 4/ (\cos wt - 1/3 \cos 3wt + \cos 5wt...)$

2. รูปคลื่นสี่เหลี่ยมแบบไม่สมมาตร (ASYMMETRICAL SQUARE WAVE)



รูป 4.3 รูปคลื่นแบบไม่สมมาตร

ฟังก์ชั่น f(t) เป็นรูปคลื่นที่สมมาตรทางแนวนอน ดังนั้นค่าเฉลี่ยได้พื้นที่จึงเป็นศูนย์ ทำให้เทอม $a_0 = 0$ และเนื่องจาก f(t) = f(-1) จะได้เทอม $a_n = 0$ เหลือเฉพาะเทอม b_n ที่ต้องคำนวณ

$$b_n = 2/T \int_{-T/2}^{T/2} f(t).\sin nwtdt$$

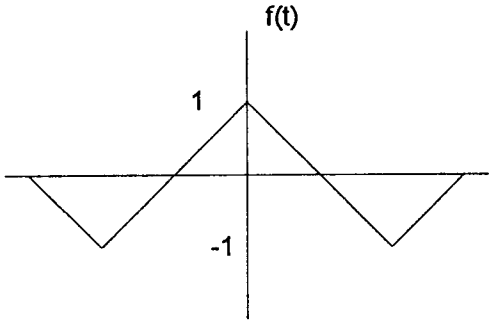
$f(t) = -1$ จาก $-T/2$ ถึง 0

$= 1$ จาก 0 ถึง $T/2$

จะได้

$$f(t) = 4/\P \ [\sin wt + 1/3\sin 3wt + 1/5\sin 5wt + ...]$$

3. รูปคลื่นสามเหลี่ยม (TRIANGULAR WAVE)

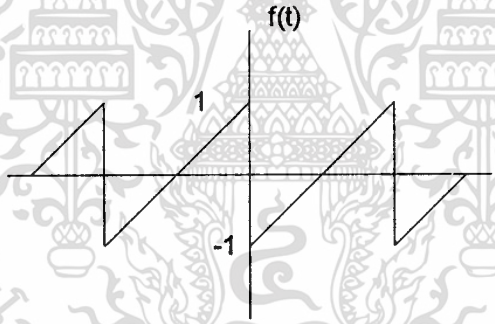


รูป 4.4 รูปคลื่นสามเหลี่ยม

จากรูปข้างบน จะได้ว่า

$$f(t) = 8/\pi^2 \quad (\cos wt + 1/9\cos 3wt + 1/25\cos 5wt + \ldots)$$

4. รูปคลื่นฟันปลา (SAWTOOTH WAVE)



รูป 4.5 รูปคลื่นฟันปลา

จากรูปข้างบนจะได้ว่า

$$f(t) = 2/\pi \quad (\sin wt - 1/2\sin 2wt + 1/3\sin 3wt + \ldots)$$

4.4 รูปคอมเพล็กซ์ (COMPLEX FORM)

ฟังก์ชั่น f(t) นี้สามารถเขียนแทนด้วยปริมาตรคอมเพล็กซ์ อีกวิธีหนึ่ง โดยจากเทอมของสัญญาณรูปคอมเพล็กซ์ ซึ่งจะให้ว่า

$$\cos nwt = e^{jnwt} + e^{-jnwt}/2$$

$$\sin nwt = e^{jnwt} + e^{-jnwt}/2j$$

ทำการแทนค่าของ cosnwt และ sinnwt ลงในอนุกรมของฟูเรียร์ ก็จะได้

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \, e^{jnwt} + e^{-jnwt} /2 + \sum_{n=1}^{\infty} b_n \, e^{jnwt} + e^{-jnwt} /2j$$

$$= a_0 + \sum_{n=1}^{\infty} \{(a_n - jb_n)e^{jnwt} /2 + (a_n + jb_n) \, e^{-jnwt} /2\}$$

กำหนดให้

$$C_n = 1/2 \, (a_n - jb_n)$$

$$C_{-n} = 1/2 \, (a_n + jb_n)$$

$$C_0 = a_0 \, 1/T$$

โดย C_{-n} เป็น Complex conjugate ของ C_n ดังนั้นการคำนวณหาค่า C_n ทำได้โดย

$$C_n = 1/T \int_{-T/2}^{T/2} f(t)[cosnwt - jsinnwt]dt$$

$$C_{-n} = 1/T \int_{-T/2}^{T/2} f(t)[cosnwt + jsinnwt]dt$$

$$C_n = 1/T \int_{-T/2}^{T/2} f(t). \, e^{-jnwt} \, dt$$

$$C_{-n} = 1/T \int_{-T/2}^{T/2} f(t). \, e^{jnwt} \, dt$$

ในที่สุดจะได้

$$f(t) = C_0 + \sum_{n=1}^{\infty} C_n.e^{jnwt} + \sum_{n=-\infty}^{-1} C_n.e^{jnwt}$$

$$f(t) = \sum_{n=\infty}^{\infty} C_n.e^{jnwt}$$

4.5 การแปลงฟูเรียร์ (FOURIER TRANSFORM)

เทคนิคของอนุกรมฟูเรียร์สามารถปรับปรุงมาใช้กับรูปคลื่นที่ไม่เป็นคาบอย่างเช่น ลูกพัลซ์เดี่ยวๆ (simple pulse) โดยการให้ค่า เป็นค่าอนันต์

สมมติว่า f(t) เดิมเป็นสัญลักษณ์ที่เป็นคาบจะได้ว่า

$$f(t) = \sum_{n=\infty}^{\infty} C_n.e^{jnwt}$$

เมื่อ $$C_n = 1/T \int_{-T/2}^{T/2} f(t).\, e^{-jnwt}\, dt$$

จากเงื่อนไขที่ว่าสัญญาณนั้นเป็นพัลซ์เดี่ยวๆ ทำให้

$$T \longrightarrow \infty \quad : \quad w = 2\pi /T \longrightarrow dw$$

ดังนั้น $$1/T = w/2\pi \longrightarrow dw/2$$

ยิ่งไปกว่านั้น ฮาร์โมนิคส์ที่ n ของอนุกรมฟูเรียร์ nw จะกลายเป็น ndw ซึ่งถือว่ามีค่าเป็น w นั่นเอง ในที่สุดเครื่องหมายซิกม่า (Sigma) จะกลายเป็นเครื่องหมายอินทริกรัล (Integral) ดังนั้น

$$Cn = dw/2 \int_{-\infty}^{\infty} f(t).e^{-jwt}\, dt$$

$$f(t) = \int_{-\infty}^{\infty} dw/2 \left[\int_{-\infty}^{\infty} f(t).e^{-jwt}\, dt\right]e^{jwt}$$

เทอมที่อยู่ในเครื่องหมายก้ามปู เป็นเทอมของความถี่เพียงอย่างเดียว ซึ่งให้เป็น g(w) โดย

$$g(w) = \int_{-\infty}^{\infty} f(t).e^{-jwt}\, dt$$

ซึ่ง g(w) เรียกว่าการแปลงฟูเรียร์ของฟังก์ชั่น f(t) ดังนั้นสมการของ f(t) จะกลายเป็น

$$f(t) = 1/2\pi \int_{-\infty}^{\infty} g(w).e^{-jwt}\, dw$$

4.5.1 **ความต่อเนื่องของสเปกตรัม** (CONTINUOUS SPECTRUM)

สัญญาณที่มีลักษณะเป็นพัลซ์เดี่ยวๆ สามารถเขียนแทนด้วยผลรวมของความถี่ต่างๆ จำนวนอนันต์ ทั้งนี้เนื่องจาก g(w) มี w เป็นเทอมของความถี่จากการอินทริเกรทนี้เอง ทำให้ได้ว่าสเปกตรัมที่ได้มีความต่อเนื่องตลอด ซึ่งตรงกันข้ามกับกรณีของรูปคลื่นที่เป็นคาบที่ให้สเปกตรัมไม่ต่อเนื่อง โดยความจริงแล้วทุกความถี่จะอยู่ใกล้กันมาก ทั้งนี้เพราะช่องว่างระหว่างความถี่นั้น คือ 1/T จะมีค่าเป็นศูนย์ เมื่อ T เป็น ∞

โดยทั่วไปแล้ว g(w) เป็นเทอมคอมเพล็กซ์ โดยที่ขนาดและเฟสสามารถนำมาพล็อตเป็นแถบความถี่ของสัญญาณ f(t) ขนาด /g(w)/จะแปรไปตามคาบและ /g(w)/ เป็นพื้นที่ได้กราฟภายในช่วง dw ถูกเรียกว่า สเปกตรัมเดนซิตี้ (SPECTRAL DENSITY) แต่เนื่องจาก dw/2 มีค่าเกือบเป็นหนึ่ง ดังนั้นพื้นที่ของ /g(w)/dw/2 จึงมีค่าเพียงขนาดของ /g(w)/ ซึ่งเป็นแอมพลิจูด เดนซิตี้
(AMPLITUDE DENSITY)

4.5.2 **การแปลงฟูเรียร์ของสัญญาณไม่ต่อเนื่อง**(Discrete Fourier Transform)

จากความเจริญก้าวหน้าทางเทคโนโลยี ทำให้ปัจจุบันมีเครื่องคอมพิวเตอร์ใช้กันอย่างทั่วถึง ด้วยเหตุนี้เองการคำนวณการแปลงฟูเรียร์ด้วยการอินทริเกรทจึงไม่นิยมกัน เพราะยุ่งยากและเสียเวลาทำให้มีการคิดโดรปโปรแกรมการแปลงฟูเรียร์ขึ้น ดังนั้นสัญญาณต่อเนื่องที่จะนำแปลงด้วยฟูเรียร์จึงต้องมีการสุ่ม (SAMPLING) ให้เป็นสัญญาณที่ไม่ต่อเนื่องเพื่อนำขนาดของแซมเปิ้ลไปคำนวณนี้

สูตรในการแปลงฟูเรียร์ของสัญญาณต่อเนื่องคือ

$$G(f) = \int_{-\infty}^{\infty} g(t).e^{-j2f} dt$$

$$g(w) = \int_{-\infty}^{\infty} f(t).e^{-j2f} dt$$

เมื่อแปลงให้เป็นการแปลงฟูเรียร์ของสัญญาณที่ไม่ต่อเนื่องจะได้

$$G(u) = 1/N \sum_{x=0}^{N-1} g(x)e^{-j2\pi}$$

เมื่อ N เป็นจำนวนแซมเปิ้ล ในการแปลงกับฟูเรียร์ของสัญญาณที่ไม่ต่อเนื่อง คือ

$$g(x) = \sum_{x=0}^{N-1} G(u)e^{-j2\pi/n}$$

ตัวอย่าง จากสัญญาณไม่ต่อเนื่องในรูปข้างบนจะให้ N=4 โดย g (r)=(2,3,4,4)
สเปกตรัมของสัญญาณดังกล่าวคำนวณได้จากสูตรการแปลงฟูเรียร์ของสัญญาณที่ไม่ต่อเนื่องคือ

$$G(0) = 1/4 \sum_{x=0}^{3} g(x) = 1/4[g(0) + g(1) + g(2) + g(3)]$$

= 1/4[2+3+4+4]

= 3.25

สูตรการแปลงฟูเรียร์ของสัญญาณต่อเนื่อง

$$H(f) = \int_{-\infty}^{\infty} h(t)e$$

$$h(t) = 1/2\pi \int_{-\infty}^{\infty} H(f)\, e\, dw$$

สูตรการแปลงฟูเรียร์ของสัญญาณที่ไม่ต่อเนื่อง

$$H(k) = 1/N \sum_{m=0}^{N-1} h(m)\, e\, j2\ /N \quad ; k = 0, 1, 2, \ldots\ldots, N-1$$

$$h(m) = \sum_{m=0}^{N-1} h(m)\, e\, j2\ /N \quad ; m = 0, 1, 2, \ldots\ldots, N-1$$

จาก Sequence ของสัญญาณ h(m)=(2,3,4,4) ให้หา Spectrum ของสัญญาณนั้นคือ N=4

$$H(k) = 1/4 \sum_{m=0}^{3} h(m)e = 1/4 \sum_{m=0}^{3} h(m)e$$

$$H(0) = 1/4 \sum_{m=0}^{3} h(m)$$

$$= 1/4\ [\ h(0) + h(1) + h(2) + h(3)\ ]$$

$$= 1/4\ [2+3+4+4] = 3.25$$

$$H(1) = 1/4 \sum_{m=0}^{3} h(m)e$$

$$= 1/4\ [h(0)\, e^{-jo} + h(1)\, e^{-jx/2} + h(2)\, e^{-jx} + h(3)\, e^{-j3x/2}$$

$$= 1/4\ [2.1 + 3.(-j) + 4.(-1) + 4\ (j\ )$$

$$= 1/4\ [2-3j - 4 + 4\ j]$$

$$= 1/4\ [-2 + j]$$

$$H(2) = 1/4 \sum_{m=0}^{3} h(m)\, e^{-jmx}$$

$$= 1/4\ [h(0)\, e^{-jo} + h(1)\, e^{-jx/2} + h(2)\, e^{-jx} + h(3)\, e^{-j3x/2}$$

$$= 1/4\ [\ 2.1 + 3.\ (-1) + 4.\ (1) + 4\ (-1)$$

$$= 1/4\ [2 - 3 + 4 - 4\ ]$$

$$= -0.25$$

$$H(3) = 1/4 \sum_{m=0}^{3} h(m)e^{-j3m\pi/2}$$

$$= 1/4\,[h(0)\,e^{-j0} + h(1)\,e^{-j3\pi/2} + h(2)\,e^{-j3\pi} + h(3)\,e^{-j9\pi/2}$$

$$= 1/4\,[2.1 + 3.(j) + 4.(-1) + 4(-j)]$$

$$= 1/4\,[2 + 3j - 4 - 4j]$$

$$= -1/4\,[2 + j_1$$

เพราะฉะนั้น สเปกตรัม คือ

$$/H(0)/ = 3.25$$

$$/H(1)/ = 1/4\,/\sqrt{(-2)^2 + (1)^2}\,/ = 5/4$$

$$/H(2)/ = /0.25/ = 0.25$$

$$/H(3)/ = 1/4\,/\sqrt{(-2)^2 + (1)^2}\,/ = 5/4$$

4.5.3 การแปลงฟูเรียร์อย่างเร็ว (Fast Fourier Transform)

ขั้นตอนวิธี หรือ ลำดับการในการคำนวณฟูเรียร์ให้เร็วมีชื่อเรียกว่า การแปลงฟูเรียร์อย่างเร็ว คิดค้นโดย คูลีย์ (J.W. Cooley) กับ ทูคีย์ (J.W. Tukey) ซึ่งได้เสนอไว้ในปี ค.ศ. 1965 หลังจากนั้นทำให้เกิดการพัฒนาวิธีหลายวิธี แต่ในวิธีของ คูลีย์ และ ทูคีย์ช่วยให้การคำนวณเชิงซ้อนเพียง $N\log_2 N$ หรือลดจำนวนครั้งในการคูณตัวเลขลดลงไปถึง $N/\log_2 N$ เท่า ผลดีอีกประการหนึ่งก็คือทำให้การสร้างวงจรเฉพาะเพื่อการคำนวณ DFT ทำให้ง่าย และคำนวณได้เร็วขึ้น

จากหัวข้อที่ผ่านมาแสดงถึงการแปลงฟูเรียร์สำหรับสัญญาณไม่ต่อเนื่อง จะสังเกตุได้ว่าจะมีส่วนที่เราต้องคำนวณซ้ำๆ คือ การคูณจำนวนเชิงซ้อนซึ่งมักเป็นเลขเทศนิยมความละเอียด 1-2 เท่ากันมากทำให้เสียเวลาในการคำนวณดังในตารางแสดงให้เห็นถึงจำนวนที่ลดลงเมื่อมีการใช้การแปลงฟูเรียร์อย่างเร็ว เช่น N ขนาด 1024 จุด เมื่อใช้ DFT จะต้องคูณจำนวนเชิงซ้อน 1048576 แต่เมื่อใช้ FFT จะเหลือเพียง 10240 เท่านั้น

จากหัวข้อที่แล้วเราได้สูตรการแปลงฟูเรียร์ คือ

$$H(k) = 1/N \quad h(m)W^{km} \sum_{m=0}^{N-1} \qquad k = 0,1,\ldots,N-1$$

โดยที่

$$= e^{j2/N} \qquad \text{โดย } j = \sqrt{-1}$$

Lenght of Transform (N)	DFT Operations (N^2)	FFT Operations $NLOG_2(N)$
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16384	896
256	65536	1024
512	262144	4096
1024	1048576	10240
2048	4194304	22528

ตารางที่ 4-1 เปรียบเทียบจำนวนการคูณค่าเชิงซ้อนระหว่าง DFT กับ FFT

คุณสมบัติและเทคนิคช่วยแปลงฟูเรียร์อย่างเร็วมีดังนี้

1. คุณสมบัติของการแปลงฟูเรียร์ (Motivation to search for an algorithum)

$H(N/2+l) = H(N/2-l)$ $\quad l = 1,2,...,N/2-1$

โดย H(k) เป็น Complex Conjugate ของ H(k)

2. เทคนิคสำหรับการลดการคำนวณ (Key to Developing the algorithm)

$$m = m_{n-1}\,2^{n-1} + m_{n-2}\,2^{n-2} + ... + m_1 2^1 + m_0 2^0$$

โดย $m_v = 0$ หรือ 1 เมื่อ $v = 0,1,...,n-1$ โดย $n = \log_2 N$

ทำนองเดียวกัน

$$k = k_{n-1}\,2^{n-1} + k_{n-2}\,2^{n-2} + ... + k_1 2^1 + k_0 2^0$$

โดย $k_v = 0$ หรือ 1 เมื่อ $v = 0,1,...,n-1$ โดย $n = \log_2 N$

ทำนองเดียวกัน

ถ้าเราให้ h(m) เป็น sequence ของข้อมูลที่ตัวแปรเป็นเลขฐานสองที่ใช้แทน h(m) ชี้เป็นเลขฐานสิบ เราจะได้ว่า

$$h(m) = h(m_{n-1}\,2^{n-1} + m_{n-2}\,2^{n-2} + \ldots + m_1 2^1 + m_0 2^0)$$
$$= h(m_{n-1} + m_{n-2} + \ldots + m_1 + m_0) \qquad (4.5)$$

จากสมการที่ (2.5) จะได้ว่า

$$\sum_{m=0}^{N-1} h(m)W^{km} = \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\cdots\sum_{m_{n-1}=0}^{1} h(m_{n-1}, m_{n-2}, \ldots, m0)\, W^{k[m_{n-1}\,2^{n-1} + m_{n-2}\,2^{n-2}}$$

ตัวอย่างในกรณีที่ N=4 หรือ $n = \log_2 4 = 2$ ดังนั้นสมการที่ (2) จะเป็น

$$\sum_{m_0}\sum_{m_1} h(m_1,m_0)W^{k[2m1+m0]} = \sum_{m_0}\{h(0,m0)\,Wk^{m0} + h(1,m_0)W^{k(2+m0)}\}$$
$$= \sum_{m_0=0}^{1} h(0,m_0)Wk^{m0} + \sum_{m_0=0}^{1} h(1,m_0)W^{k(2+m0)}$$
$$= h(0,0) + h(0,1)W^{k} + h(1,0)W^{2k} + h(1,1)\,W^{3k} \qquad (4.6)$$

ดังนั้นสมการที่ (4.6) จะได้ว่า

$$\sum_{m_0}\sum_{m_1} h(m_1,m_0)W^{k[2m1+m0]} = \sum_{m=0}^{3} h(m)W^{km}$$

ตัวอย่างการนำไปใช้งาน

เราเอากรณี N=8 เป็นพื้นฐานเบื้องต้นในการอธิบายขบวนการของ FFT นั้นคือ

$$H(k) = 1/8 \sum_{m=0}^{7} h(m)W^{km} \quad \text{โดย } k = 0,1,\ldots,7 \qquad (4.7)$$

เมื่อ $W = e^{-j2\,/8} = e^{-j\,/4}$ และเราให้ m เขียนอยู่ในรูป binary คือ

$$m = m_2 2^2 + m_1 2^1 + m_0 2^0 \qquad (4.8)$$

จากสมการ (4.7) เอา 8 คูณตลอดจะได้

$$8H(k) = \sum_{m=0}^{7} h(m)W^{km} \qquad (4.9)$$

เมื่อแปลงเทอมทางขวามือของสมการที่ 6 ให้อยู่ในรูปของเลขฐานสอง จะได้ว่า

$$8H(k) = \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\sum_{m_2=0}^{1} h(m_1, m_2m_0)\, W^{k[4m2+2m1+m0)}$$

$$= \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\sum_{m_2=0}^{1} h(m_1, m_2m_0)\, W^{4km2} + W^{2km1} + W^{km0} \qquad (4.10)$$

จากสมการที่ (2.11)เราให้ Summation ในที่สุดเป็น M_2 นั้นคือ

$$M_2 = \sum_{m_2=0}^{1} h(m_1, m_2m_0)\, W^{4km2}$$

แต่เนื่องจาก W^4=-1 และแทน k ด้วยเลขฐานสองจะได้

$$M_2 = \sum_{m_2=0}^{1} h(m_1, m_2m_0)\, (-1)^{m_2[4m2+2m1+m0)}$$

แต่เนื่องจาก $(-1)^{m2[4k2+2k1]} = 1$, M_2 สามารถเขียนใหม่เป็น

$$M_2 = \sum_{m_2=0}^{1} h(m_1, m_2m_0)\, (-1)^{m_2[4m2+2m1+m0)} \qquad (4.12)$$

ผลรวม ของสมการที่ (4.11) เป็นการแทนค่า m_2 ด้วย 0 และ 1 ดังนั้นสมการที่ (4.13) จะมีตัวแปรที่ไม่ทราบค่าคือ k_0, m_1และ m_0 ดังนั้น M_2 เขียนใหม่เป็น

$$M_2 = \sum_{m_2=0}^{1} h(m_1, m_2m_0)\, (-1)^{m_2[4m2+2m1+m0)} = h_1(k_0, m_1, m_0) \qquad (4.13)$$

แทนค่าสมการ (4.13) ลงในสมการ (4.11)

$$8H(k) = \sum_{m_0=0}^{1}\sum_{m_1=0}^{1} h_1(k_0, m_2m_0) W^{2km1}\, W^{km0} \qquad (4.14)$$

ผลรวม ชุดในของสมการที่ (4.14) เราสมมติให้เป็น M1 นั่นคือ

$$M_1 = \sum_{m_1=0}^{1} h_1(k_0, m_2m_0) W^{2km1}$$

$$= \sum_{m_1=0}^{1} h_1(k_0, m_2m_0)(-j)^{(4k2+2k1+k0)m1} \qquad (4.15)$$

โดยที่จากสมการที่ (4.15) นั้น $W^2 = e^{-j2\pi 2/4} = -j$ และเทอม $(-j)^{4k2m1} = 1$

ดังนั้น M_1 เขียนได้ใหม่เป็น

$$M_1 = \sum_{m_1=0}^{1} h_1 (k_0 , m_2 m_0)(-j)^{(+2k1+k0)m} \qquad (4.16)$$

จากสมการที่(4.16) เมื่อทำการแปรค่า M_1 เป็น 0 และ 1 ดังนั้นเทอม M_1 จะเป็นตัวแปรของ k_0 , k_1 และm_0 นั่นคือสมการ (4.16) จึงกลายเป็น

$$M_1 = h_2 (k_0 , m_2 , m_0) \qquad (4.17)$$

แทน (4.17) ลงใน (4.13) จะได้

$$8H(k) = \sum_{m_1=0}^{1} h_2 (k_0 , m_2 m_0) W^{m0k1}$$

ในทำนองเดียวกับ M_2 และ M_1 เมื่อให้M_0 เป็นผลการรวม ของค่า M_0 นั่น คือ

$$M_0 = \sum_{m_1=0}^{1} h_1 (k_0 , m_2 , m_0) W^{(4k2+2k1+k0)}$$

$$= \sum_{m_1=0}^{1} h_1 (k_0 , m_2 , m_0)(1-j/\sqrt{2})^{(4k2+2k1+k0)} \qquad (4.18)$$

หลังจากทำการแปรค่าM_0ไปแล้วจะพบว่าM_0เป็นฟังก์ชั่นของ k_0, k_1 และ k_2 ซึ่งเราจะแทนด้วย

$$M_0 = h_3 (k_0 , k_2 , k_0) \qquad (4.19)$$

ดังนั้นจากสมการที่ (4.11) เราจะได้คำตอบว่า

$$8H(k) = 8H (k_0 , k_2 , k_0) = h_3 (k_0 , k_2 , k_0) \qquad (4.20)$$

จากสัมประสิทธิ์ (-1),(-j) และ (l-j) /$\sqrt{2}$ ค่าเหล่านี้จะเป็นรากของ e^{-j2} นั่นเองกล่าวคือ (-1),(-J) และ (l-j)/$\sqrt{2}$ เป็นรากที่ 2,4 และที่ 8 ของ unity ซึ่งเราจะใช้สัญลักษณ์แทนด้วย

$$A_{2^r} = e^{-j2\pi/2^r} \quad \text{เมื่อ } r = 1,2,....,\log_2 N \qquad (4.21)$$

โดย A_{2^r} จะมีคุณสมบัติคือ

(I) $A_{2^r} = W^{N/2^r}$ $\quad W = e^{-j2\pi/2^r}$ (4.22a)

(ii) $(A_{2^r})^{\lambda+1} = -(A_{2^r})^{l}$ เมื่อ $r = 1,2,....,\log_2 N$, $l = 0,1,...,2^{r-1}$

$\lambda = 2^{r-1}$ (4.22b)

(iii) $(A_N)^{N/2} = -1$ (4.22c)

จากสมการที่ (4.13) เราเขียนอยู่ในเทอม จะได้ว่า

$$h_1(k_0, m_2, m_0) = \sum_{m_2} h(m_1, m_2, m_0) A_2^{k0}$$

นั่นคือ

$$h_1(k_0, m_2, m_0) = h_1(0, m_1, m_0) + h(1, m_2, m_0) A_2^{k0} \quad (4.23)$$

สมการที่ (4.33) นี้ จะมีค่าไม่เป็น 0 ก็เป็น 1 จะได้ว่าค่า แต่ละค่าให้ 4 สมการ คือ

case ที่ 1 $k_0 = 0$ (4.24a)

$h_1(0,0,0) = h(0,0,0) + h(1,0,0) \longrightarrow h_1(0) = h(0) + h(4)$

$h_1(0,0,1) = h(0,0,1) + h(1,0,1) \longrightarrow h_1(1) = h(1) + h(5)$

$h_1(0,1,0) = h(0,1,0) + h(1,1,0) \longrightarrow h_1(2) = h(2) + h(6)$

$h_1(0,1,1) = h(0,1,1) + h(1,1,1) \longrightarrow h_1(3) = h(3) + h(7)$

case ที่ 2 $k_0 = 1$ (4.24b)

$h_1(1,0,0) = h(0,0,0) + A_2 h(1,0,0) \longrightarrow h_1(0) = h(0) - h(4)$

$h_1(0,0,1) = h(0,0,1) + A_2 h(1,0,1) \longrightarrow h_1(1) = h(1) - h(5)$

$h_1(0,1,0) = h(0,1,0) + A_2 h(1,1,0) \longrightarrow h_1(2) = h(2) - h(6)$

$h_1(0,1,1) = h(0,1,1) + A_2 h(1,1,1) \longrightarrow h_1(3) = h(3) - h(7)$

สมการที่ (4.16) เมื่อแทน (-j) ด้วยเทอม A_4 จะได้ว่า

$$h_2(k_0, m_2, m_0) = h_1(k_0, 0, m_0) + h_1(k_0, 1, m_0) A_4^{(2k1-k0)}$$

จากการ FIXED ค่า k_0, k_1 แล้วทำการแปร m_0 ไปแต่ละครั้งจะได้สมการ 2 สมการดังนี้ คือ

case 1 $k_1, k_0 = (0,0)$ (4.25a)

$h_2(0,0,0) = h_1(0,0,0) + h_1(1,0,0) \longrightarrow h_2(0) = h_1(0) + h_1(4)$

$h_2(0,0,1) = h_1(0,0,1) + h_1(1,0,1) \longrightarrow h_2(1) = h_1(1) + h_1(5)$

case 2 $k_1, k_0 = (0,1)$ (4.25b)

$h_2(1,0,0) = h_1(0,0,0) + h_1(1,1,0) \longrightarrow h_2(4) = h_1(4) + h_1(6)$

$h_2(1,0,1) = h_1(0,0,1) + h_1(1,1,1) \longrightarrow h_2(5) = h_1(5) + h_1(7)$

case 3 $k_1, k_0 = (1,0)$ (4.25c)

$h_2(0,1,0) = h_1(0,0,0) + h_1(1,0,0) \longrightarrow h_2(2) = h_1(0) - h_1(2)$

$h_2(0,1,1) = h_1(0,0,1) + h_1(1,0,1) \longrightarrow h_2(3) = h_1(1) - h_1(3)$

case 4 $k_1, k_0 = (1,1)$ (4.25d)

$$h_2(1,1,0) = h_1(1,0,0) + h_1(1,1,0) \longrightarrow h_2(6) = h_1(4) - A_4 h_1(6)$$

$$h_2(1,1,1) = h_1(1,0,1) + h_1(1,1,1) \longrightarrow h_2(7) = h_1(5) - A_4 h_1(7)$$

สมการที่ (4.55.a) ถึง (4.55.d) เป็นการทำ ITERATION ครั้งที่ 2 ดังรูปที่ 1 โดยให้ทำ ITERATION ครั้งที่ 2 นี้คือ ในสมการที่ (2.21) นั้นเอง

ในที่สุดสมการที่ (2.19) เราเขียนค่าสมประสิทธิ์ในเทอมของ A8 จะได้ว่า

$$h_3(k_0, k_1, k_2) = h_2(k_0, k_1, 0) + h_2(k_0, k_2, 1) A_8^{[4k2+2k1+k0]}$$

ซึ่งจะได้ว่า

case 1 $k_2, k_1, k_0 = (0,0,0)$ (4.26a)

$$h_3(0,0,0) = h_2(0,0,0) + h_2(0,0,1) \longrightarrow h_3(0) = h_2(0) + h_2(1)$$

case 2 $k_2, k_1, k_0 = (0,0,1)$ (4.26b)

$$h_3(1,0,0) = h_2(1,0,0) + A_8 h_2(0,0,1) \longrightarrow h_3(4) = h_2(4) + A_8 h_2(1)$$

case 3 $k_2, k_1, k_0 = (0,1,0)$ (4.26c)

$$h_3(0,1,0) = h_2(0,1,0) + A_8^2 h_2(0,1,1) \longrightarrow h_3(2) = h_2(2) + A_8^2 h_2(3)$$

case 4 $k_2, k_1, k_0 = (0,1,1)$ (4.26d)

$$h_3(1,1,0) = h_2(1,1,0) + A_8^3 h_2(1,1,1) \longrightarrow h_3(6) = h_2(6) + A_8^3 h_2(7)$$

case 5 $k_2, k_1, k_0 = (1,0,0)$ (4.26e)

$$h_3(0,0,1) = h_2(0,0,0) + A_8^4 h_2(0,0,1) \longrightarrow h_3(1) = h_2(0) - A_8^4 h_2(1)$$

สมการที่เหลือเราใช้คุณสมบัติของH(N/2+1)=H(N/2-1) นั่นเอง จากกลุ่มสมการที่ (23) นี้เป็นการทำ iteration ครั้งที่ 3 นั้นคือ r=3 ในสมการที่ (17) นั่นเอง

ในที่สุดจะได้ว่า

$$h_3(0) = 8H(0)$$

$$h_3(4) = 8H(1)$$

$$h_3(2) = 8H(2)$$

$$h_3(6) = 8H(3)$$

$$h_3(1) = 8H(4)$$

ส่วน H(5),H(6) และH(7) ได้จากคุณสมบัติทาง COMPLEX CONJUGATE หรือถ้าคำนวณจะพบว่า

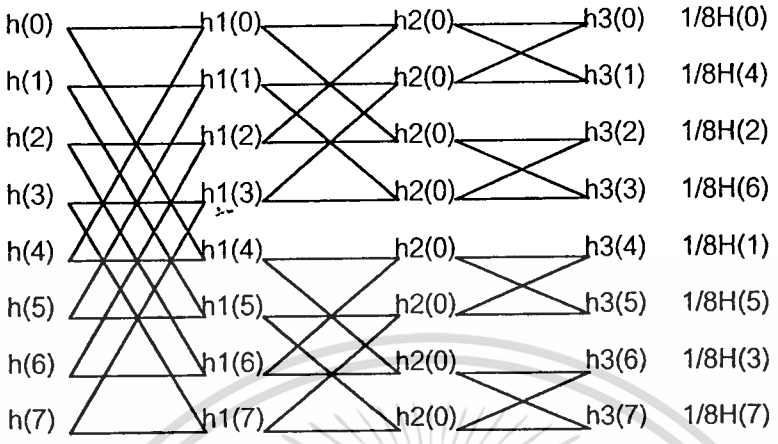
$$8H(5) = h_3(5) = h_2(4) - A_8 h_2(5)$$

$$8H(6) = h_3(3) = h_2(2) - A^2_8 h_2(3)$$

$$8H(7) = h_3(7) = h_2(6) - A^3_8 h_2(7)$$

จากสมการที่ (20),(21) และ (23) จะได้ว่า Signal flow graph N=8 คือ

จากสมการที่ (20),(21) และ (23) จะได้ว่า Signal flow graph N=8 คือ



ถ้า {h(m)} = {1,2,1,1,3,2,1,2} ให้หา H(k)

h(0) = 1	*h1(0) = 4*	*h2(0) = 13*	*--> 1/8-->H(0)*
h(1) = 2	*h1(1) = 4*	*h2(0) = -1*	*--> 1/8-->H(4)*
h(2) = 1	*h1(2) = 2*	*h2(0) = 2-j*	*--> 1/8-->H(2)*
h(3) = 1	*h1(3) = 3*	*h2(0) = 2+j*	*--> 1/8-->H(6)*
h(4) = 3	*h1(4) = -2*	*h2(0) = -1.293+j0.707*	*--> 1/8-->H(1)*
h(5) = 2	*h1(5) = 0*	*h2(0) = -2.707-j0.707*	*--> 1/8-->H(5)*
h(6) = 1	*h1(6) = 0*	*h2(0) = -2.707+j0.707*	*--> 1/8-->H(3)*
h(7) = 2	*h1(7) = -1*	*h2(0) = -1.293-j0.707*	*--> 1/8-->H(7)*

บทที่ 5

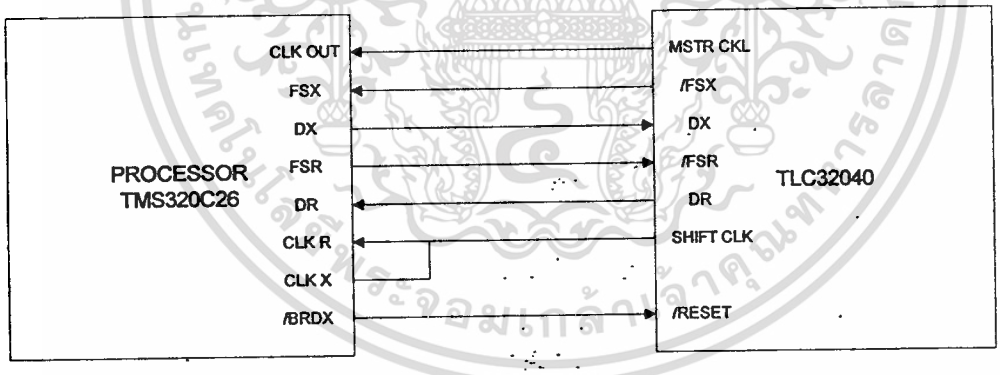
การคำนวณและการออกแบบการสร้าง

5.1 การออกแบบทางฮาร์ดแวร์

โครงงานทางฮาร์ดแวร์ได้ใช้ชุด DSK (Digital Sinal Processing Start Kit) ของบริษัท Texas Instruments ภายในประกอบด้วย โปรเซสเซอร์ DSP ขนาด 16 บิต แบบ ฟิกส์พอยต์ ทำงานด้วยความถี่ 40 MHz มีชุด A/D ,D/A และชุดเชื่อมต่อ RS-232

5.1.1 การเชื่อมต่อ A/D , D/A

การเชื่อมต่อ A/D ,D/A นี้ใช้ ไอซี AIC (Analog Interface Circuit) เบอร์ TLC 32040 มีความสามารถทำงานเป็นได้ทั้ง A/D และ D/A มีความละเอียด 14 บิต อัตราแซมปลิ้ง สูงสุด 20 Khz การติดต่อระหว่าง AIC กับ T M S 320C26 เป็นการติดต่อ ข้อมูล แบบอนุกรม ส่งแบบ ซิงโครนัส สัญญาณอินพุทอานาลอก จะเข้าที่ขา IN+ หลังจากนั้น AIC จะทำการ แบนด์พาสฟิลเตอร์ ซึ่งสามารถบายพาสได้จากโปรแกรม แล้วทำการ แปลง A/D แล้วส่งข้อมูลเป็นอนุกรมทางขา DR และควบคุมการส่งข้อมูลด้วย /FSR การเชื่อมต่อแสดงดังรูปที่ 5.1

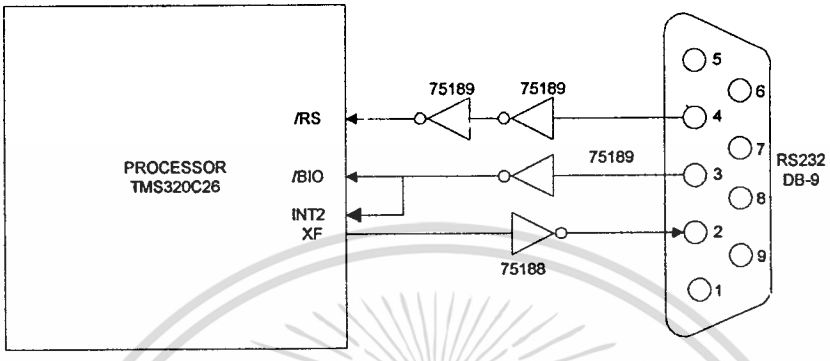


รูปที่ 5.1 แสดงการเชื่อมต่อระหว่าง TMS320C26 และ TLC32040

5.1.2 การเชื่อมต่อ RS- 232

การต่อ RS-232 เพื่อการส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยัง TMS320C26 และจาก TMS320C26 ไปยังเครื่องคอมพิวเตอร์ (PC-Host) การส่งข้อมูลจากเครื่องคอมพิวเตอร์ จะส่งโปรแกรมที่ได้การแอสเซมบลี้แล้ว ไปทำการเอ็กซีคิ้วบนชุด DSK โดยภายในตัวโปรเซสเซอร์ TMS320C26 จะมีโปรแกรม Bootloader สำหรับรอรับการส่งข้อมูล และดีเทค บอดเรท อยู่ภายใน ดังนั้นการติดต่อกับชุด DSK ทาง RS-232 ต้องมีการทริกสัญญาณ รีเซต ทาง DTR ก่อนเสมอ แล้วสัญญาณ BIO จะเริ่มทำงานรองรับข้อมูล และสัญญาณ XF

จะใช้เป็นสัญญาณส่งข้อมูลอนุกรมเนื่องจากสัญญาณ XF สามารถควบคุมได้ง่ายจากคำสั่งโดยตรง การทำให้ระดับแรงดันระดับ TTL เป็นไปตามมาตราฐาน RS-232 จะใช้ไอที 75188 เป็นตัวจัดระดับสัญญาณและใช้ไอซี 75189 สำหรับจัดแรงดัน RS-232 เป็นระดับแรงดัน TTL

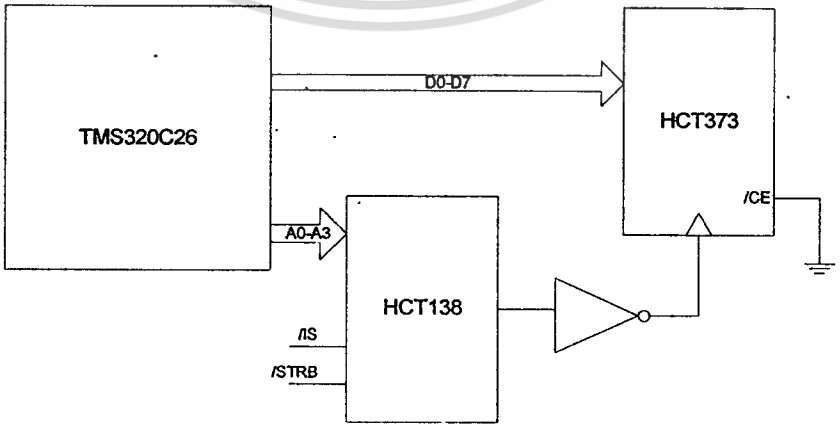


รูปที่ 5.2 แสดงการเชื่อมต่อระหว่าง TMS320C26 และเครื่องไมโครคอมพิวเตอร์ทางพอร์ทอนุกรม

5.1.3 การเชื่อมต่อพอร์ตเอาท์พุท

การออกแบบพอร์ตเอาท์พุทสำหรับโครงงานนี้เพื่อตรวจสอบเวลาในการทำงานของ FFT โดยเป็น TMS 320C26 ได้ออกแบบให้สามารถต้องอ้างหมายเลขพอร์ตได้ 16 พอร์ต ควบคุมได้ทั้ง 16 บิต และมีสัญญาณควบคุมการใช้พอร์ตด้วยสัญญาณ IS และ STRB รอบการทำงานของการควบคุมพอร์ตสามารถได้เช่นเดียวกับการต่อหน่วยความจำภายนอกไอซี 74HCT373 เป็นพอร์ตเอาท์พุทแลทช์ข้อมูลขนาด 8 บิต เพื่อให้มีสภาวะเอาท์พุทค้างไว้จนกว่าจะมีข้อมูลชุดต่อไปเข้ามา

ไอซี 74HCT138 ทำหน้าที่ดีโค๊ดหมายเลขพอร์ต ในวงจรนี้ได้ออกแบบให้ทำงานที่พอร์ตหมายเลข 0 ตามรูปที่ 5.3



รูปที่ 5.3 แสดงการต่อเอาท์พุทพอร์ท

5.1.4 การเชื่อมต่อหน่วยความจำ

TMS320C26 สามารถเชื่อมต่อกับ EPROM และ RAM ได้โดยใช้ขาสัญญาณต่าง ๆ เหล่านี้

-16 บิต แอดเดรสบัส (A0-A15)

-16 บิต ดาต้าบัส (D0-D15)

-/PS,/DS (Program,Data Space Select)

-R/W และ /STRB

-READY

ความเร็วในการเข้าถึงหน่วยความจำถูกจำกัดด้วยอุปกรณ์หน่วยความจำที่เลือกใช้ ถ้าเวลาในการเข้าถึงหน่วยความจำเร็วได้ตามที่ต้องการ TMS320C26 จะสามารถที่จะทำงานได้โดยไม่มีสภาวะคอย (On Wait State) ในทางกลับกันถ้าเวลาการเข้าถึงของหน่วยความจำที่ใช้ช้ากว่าระบบสัญญาณนาฬิกา เราจำเป็นต้องให้สภาวะคอยกับ TMS320C26 ขาสัญญาณ READY ระดับสัญญาณต่ำ จะถูกนำมาในช่วงที่สภาวะคอย

TMS320C26 ได้แบ่งชนิดบริเวณหน่วยความจำเป็น 2 แบบ คือ Progran Space (64 Kword) และ Data Space (64 Kword) การแยกแยะระหว่างหน่วยความจำทั้งสองทำได้โดยใช้สัญญาณ /PS และ /DS

Timing Diagram การอ่านและเขียนหน่วยความจำอ้างจาก Data Sheet ในที่นี้จะให้ Q เป็นตัวแสดงช่วงเวลา 1/4 เฟสของสัญญาณนาฬิกา (CLKOUT1 และ CLKOUT2) และเนื่องจากสัญญาณนาฬิกาที่ใช้ในระบบเป็น 40 MHz นั่นคือ Q=25nS

ช่วงการอ่านเหตุการณ์ลำดับต่อไปนี้

1. ภายหลัง CLKOUT1 เป็นระดับสัญญาณต่ำเล็กน้อย แอดเดรสบัสและสัญญาณเลือกชนิดของหน่วยความจำแบบใดแบบหนึ่ง (PS หรือ DS) ที่ถูกต้องจะออกมาและ RW จะเป็นระดับสัญญาณสูงเพื่อบอกว่าอยู่ในช่วงการอ่าน

2. /STRB จะเป็นระดับสัญญาณต่ำในช่วงไม่เกิน $t_{su}(A)$ = Q-12nS ภายหลังจากที่แอดเดรสบัสที่ถูกต้องออกมา

3. หลังจากที่ช่วงครึ่งไซเคิลที่สองสัญญาณอินพุท READY จะถูกสุ่มขึ้นมา ช่วงนี้ READY ต้องคงสภาวะไว้ (ระดับสัญญาณสูงหรือต่ำ) ให้ TMS320C26 ไม่นานไปกว่า $t_d(SLR)$=Q-20nS หลังจากที่ /STRB เปลี่ยนเป็นระดับสัญญาณต่ำ

4. การไม่ทำสภาวะคอย (READY ระดับสัญญาณ) ข้อมูลต้องปรากฏมาไม่ช้าไปกว่า $t_a(SC)= t_a(A)-t_{su}(A)$ = 2Q-23nS หลังจากที่ /STRB เปลี่ยนเป็นระดับสัญญาณต่ำ

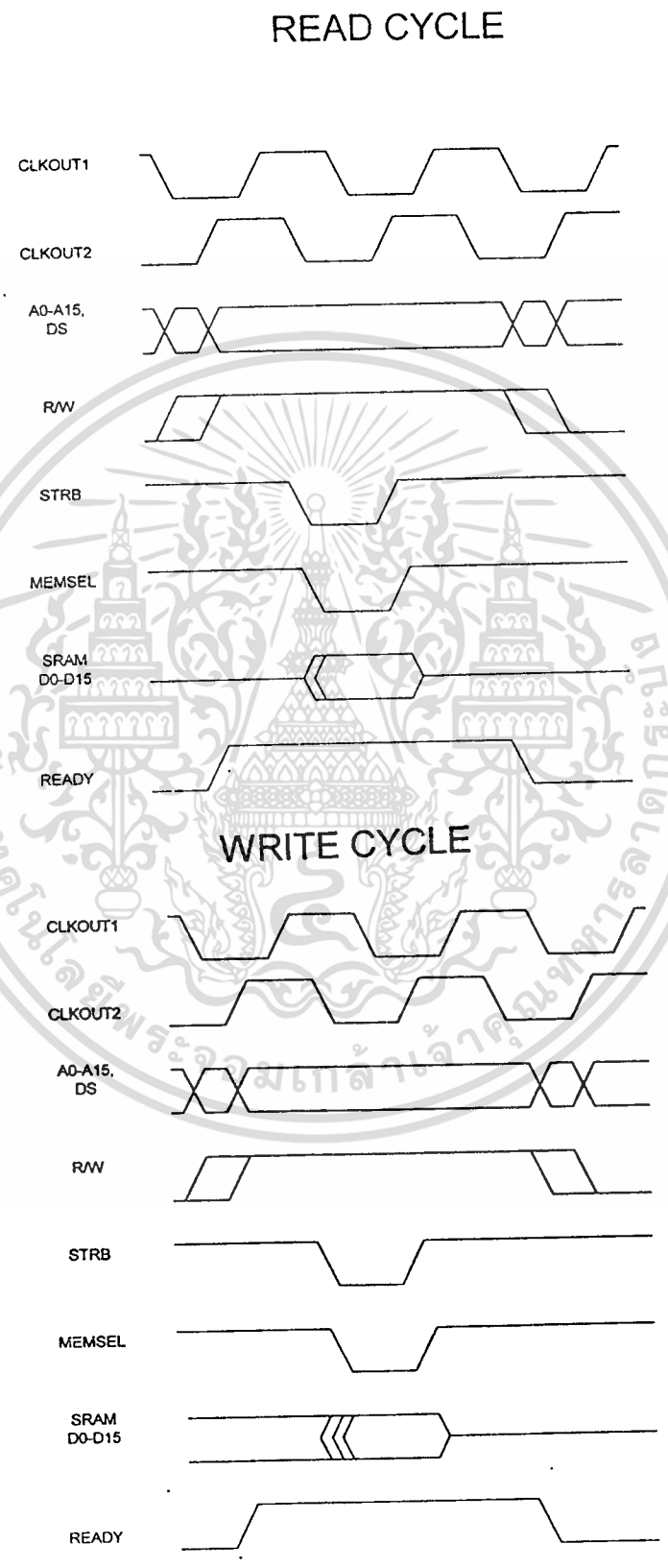
ช่วงการเขียน

ลำดับเหตุการณ์ของช่วงการเขียนเหมือนกันกับช่วงการอ่าน ต่างกันเฉพาะบางเหตุการณ์ดังนี้

1. R/W จะเปลี่ยนเป็นระดับสัญญาณต่ำเพื่อบอกว่าเป็นช่วงการเขียน

2. บัสข้อมูลจะเกิดขึ้นประมาณเวลาเดียวกันกับที่ /STRB เปลี่ยนเป็นระดับสัญญาณต่ำ

3. หลังจาก /STRB เปลี่ยนเป็นระดับสัญญาณสูง ดาต้าบัสจะเข้าสู่สภาวะอิมพีแดนซ์สูงไม่ช้าไปกว่า t_{dS}(D)=Q+15nS



รูปที่ 5.4 แสดงไซเคิลของเวลาของการอ่านและเขียนข้อมูล

5.2 การออกแบบทางด้านซอฟท์แวร์

การออกแบบทางด้านซอฟท์แวร์นั้นสามารถแบ่งออกได้เป็น 2 อย่างคือ ซอฟท์แวร์บนไมโครคอมพิวเตอร์และซอฟท์แวร์บนบอร์ดประมวลผลทางดิจิตอล

5.2.1 การออกแบบซอฟท์แวร์บนเครื่องไมโครคอมพิวเตอร์

1. การอัพโหลดไฟล์ข้อมูลไปยัง DSK เซอร์กิตบอร์ด

การทำงานภายใน DSK เซอร์กิตบอร์ด เมื่อมีสัญญาณ DTR จะทำหน้าที่รีเซตตัวโปรเซสเซอร์ TMS 326C26 แล้วจะทำเอ็กซีคิวโปรแกรม Bootloader คือโปรแกรมที่คอยรับการรับข้อมูลที่ ดาวน์โหลดมา ขั้นตอนการ ดาวน์โหลดไฟล์มีขั้นตอนดังรูป 5.5

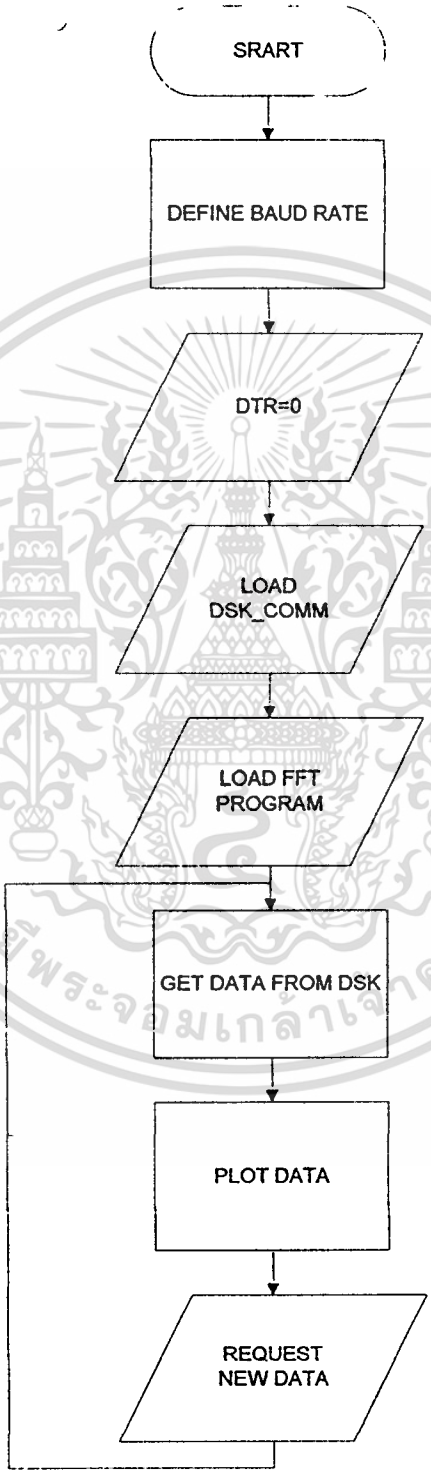
Baud Detect Word จะถูกส่งไปเพื่อให้ DSK เซอร์กิตบอร์ด ดีเทคว่าบอดเรทที่ใช้เป็นเท่าไร โดยกำหนดให้บิตข้อมูล MSB ต้องเป็น 1 หรือ 1xxxxxxx

Status Word	กำหนดสภาวะให้ DSK เซอร์กิตบอร์ด
Interrupt Word	กำหนด interrupt mask register และกำหนดโครงสร้างหน่วยความจำ
Program Length	กำหนดความยาวโปรแกรม
Program Word	ข้อมูลโปรแกรมที่จะส่งโดยกำหนดให้ไบต์ล่างก่อน
CHECK SUM	เมื่อส่งกำหนดครบตามจำนวนจะส่งค่า check sum ด้วย ถ้าข้อมูลที่ได้รับถูกต้อง ค่าที่ตอบกลับมาทาง TR จะเป็น FFH
SYNCHRONIZAION	ถ้าสัญญาณที่ตอบกลับมาว่า CHECK SUM ถูกต้อง PC-HOST จะส่งข้อมูลตอบกลับไปอีกครั้งด้วยค่า 00

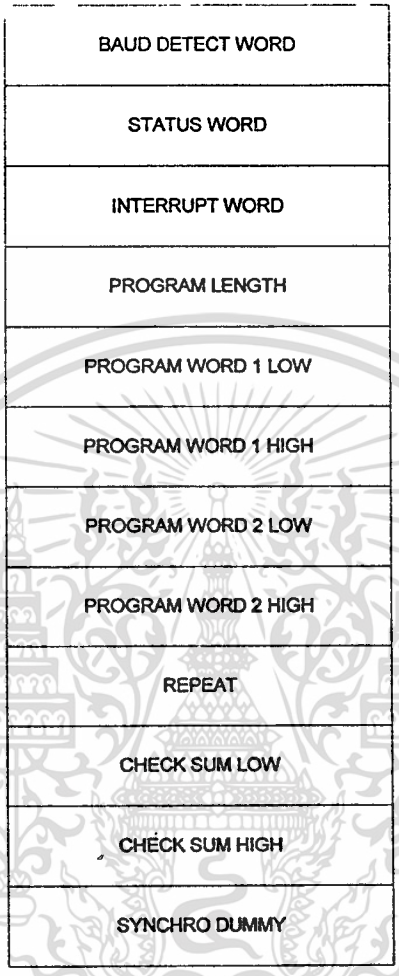
เป็นการสิ้นสุดการดาวน์โหลดครั้งที่ 1 ซึ่ง เป็นการดาวน์โหลดไฟล์ชื่อ DSK_COMM.DSK เพื่อใช้ติดต่อกับ DSK

เซอร์กิตบอร์ดได้ดีขึ้นดีคือมีความสามารถให้มีการดาวน์โหลดข้อมูลในบริเวณเนื้อที่หน่วยความจำที่กำหนดได้

การดาวน์โหลดข้อมูลตอนที่ 2 เป็นการดาวน์โหลดไฟล์ชื่อ FFT ขนาด 256 ไฟล์ FFT256.DSK เป็นไฟล์ที่ได้จากการ แอสแซมบลี้เป็นข้อมูลฐาน 16 ในรูปของค่าสตริงไม่สามารถส่งออกไปได้ทันทีต้องมีการถอดรหัสและจัดข้อมูลและรวมคำสั่งลงไป ไฟล์ DSK มีรหัสและตัวอย่างดังรูปที่ 5.6



รูปที่ 5.5 แสดงผังการดาวน์โหลดไฟล์ของ TMS320C26



รูปที่ 5.6 แสดงการจัดเรียงข้อมูล

2. การออกแบบโปรแกรมรับส่งข้อมูลและแสดงผลบนเครื่องคอมพิวเตอร์

วัตถุประสงค์การออกแบบโปรแกรมบนเครื่องคอมพิวเตอร์คือเพื่อให้สามารถ

1. สามารถอัพโหลดไฟล์ข้อมูลไปยัง DSK เซอร์กิตเบอร์ดแล้วทำการเอ็กซีคิวโปรแกรมการทำงาน

2. สามารถรับข้อมูลที่ได้ DSK เซอร์กิตเบอร์ด โดยการทำแฮนด์เช็คตลอดเวลาที่มีการส่งข้อมูล

3. สามารถพล๊อตกราฟและคำนวณค่า ลักษณะการพล๊อตกราฟข้อมูลจะได้ข้อมูลจากบัฟเฟอร์ทั้งเฟรมมาพล๊อตทั้งหมดบนแสดงแล้วให้สามารถบอกค่าต่ำแหน่งบนกราฟมาแสดงเป็นความถี่และขนาดได้

ภาษาที่ใช้สำหรับการเขียนคือ Borland Delphi 2.0

การเขียนตามวัตถุประสงค์ที่ 1 คือ ในการส่งข้อมูลให้ DSK เซอร์กิตเบอร์ดโดยมีโปรแกรม Bootloader ของ TMS320C26 รอรับต้องมีขั้นตอนดังนี้

1. โปรแกรมที่ส่งให้กับ TMS320C26 โปรแกรมแรกคือ DSK_COMM.DSK เมื่อโหลดโปรแกรมให้แล้วจึงโหลดโปรแกรมชื่อ FFT.DSK

3. **การรับข้อมูลผ่าน** RS-232

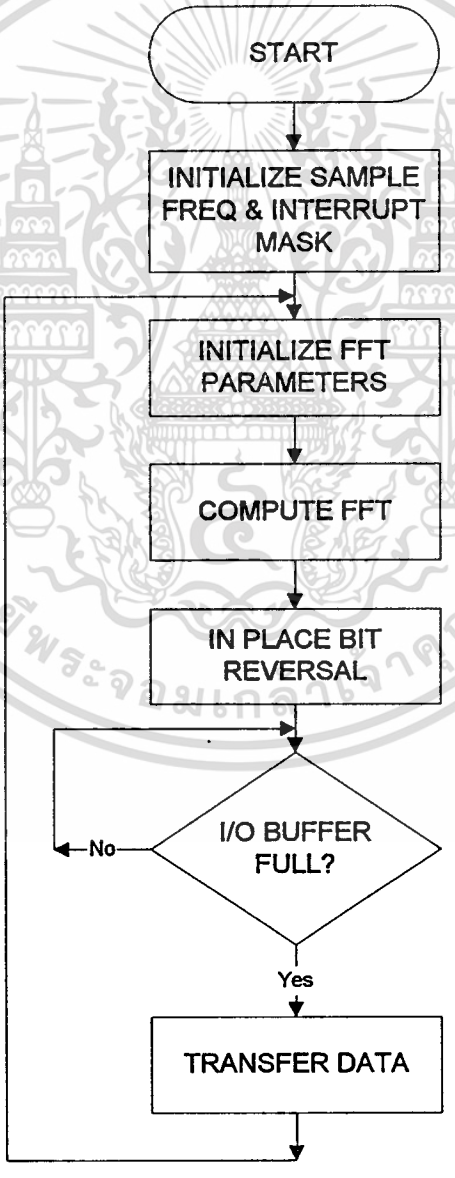
จำนวนข้อมูลที่ส่งในพอร์ตอนุกรม RS-232 มีขนาด 8 บิตข้อมูลไม่มีพาริตี้และ 2 บิตสต๊อฟมีการส่งเป็นเฟรม เมื่อมีการร้องขอข้อมูลมาโดยการร้องขอข้อมูลเพียงมีการส่งข้อมูลเพียงการส่งข้อมูล 0FFH ออกไปข้อมูลจะถูกส่งข้อมูลจะถูกส่งเข้ามาหมดทั้งเฟรม

4. การพร๊อตกราฟ

การพร๊อตกราฟจะเป็นขั้นตอนต่อจากการรับข้อมูลการแสดงผลเนื่องจากการเขียนโปรแกรมเป็นแบบ OOP ทำให้สามารถนำคอมโปแนนท์มาใช้ได้ คอมโปแนนท์ที่ใช้ชื่อ Graph 1 กำหนดให้มีขนาด -40 ถึง 40 ทั้งแกน xและ y การพล๊อตกราฟใช้ event lineto (x,y) โดยกำหนดให้แกน x เป็นแกนความถี่ และแกน y เป็นแกนขนาด การพล๊อตค่า x,y เป็นค่า real type ทำให้สามารถกำหนดให้กราฟแสดงค่าทางแกน x ได้หลายขนาด ตัวอย่างเช่น แกน x กว้าง 80 หารด้วยจำนวนจุดที่ต้องการพล๊อต 256 จุดได้ค่าเป็นขนาดความก้วางของแต่ละจุดส่วนค่า y ที่ได้รับจะสเกลค่าลงเช่นเดียวกับทางแกน x

5.2.2 **ซอฟท์แวร์ในส่วนของ** TMS320C26

สำหรับซอฟท์แวร์ในส่วนของ TMS320C26 นี้จะเขียนด้วยภาษาแอสเซมบลีของหน่วยประมวลผล TMS320C26 ซึ่งตามหน้าที่หลักของหน่วยประมวลผลตัวนี้จะทำการรับข้อมูลสัญญาณดิจิตอลจากไอซีดีทูเอ (TLC32040) ซึ่งส่งสัญญาณดิจิตอลเข้ามาแบบอนุกรมและเข้าทางขาสัญญาณดาต้ารีซิฟเวอร์ของ TMS320C26 โดยโปรแกรมจะทำการรับข้อมูลเข้ามาเก็บไว้ในหน่วยความจำดาต้าเมมโมรี โดยโปรแกรมจะตั้งโหมดในการรับข้อมูลนี้โดยใช้อินเตอร์รัพท์ จากนั้นจะทำการคำนวณหาแถบความถี่ของสัญญาณเมื่อคำนวณเสร็จแล้วก็จะทำการส่งข้อมูลที่ได้จากการคำนวณขึ้นไปแสดงผลบนเครื่องไมโครคอมพิวเตอร์ ดังแสดงในผังการทำงานดังรูปที่ 5.7



รูปที่ 5.7 แสดงผังการทำงานของโปรแกรมบน TMS320C26

จากรูปที่ 5.7 ขั้นตอนการทำงานจะเป็นดังนี้

1. ทำการตั้งค่าความเร็วของการสุ่มสัญญาณของไอซี.TLC32040 และทำการกำหนดการใช้งานของการอินเตอร์รัพท์ต่าง ๆ ซึ่งในที่นี้ใช้อินเตอร์รัพท์การรับส่งข้อมูลอนุกรมเพียงอย่างเดียว

2. ทำการกำหนดค่าความละเอียดในการคำนวณซึ่งสามารถคำนวณได้ตั้งแต่ 64,128,246,512 จุด ในโปรเจคนี้ใช้ 256 จุด โดยจะนำค่าความละเอียดของการคำนวณ,ตำแหน่งของหน่วยความจำในการคำนวณตำแหน่งต่าง ๆ ไปเก็บไว้ในรีจิสเตอร์

3. ทำการคำนวณฟาสฟูเรียร์ (FFT) จากข้อมูลที่รับเข้ามาและนำไปเก็บไว้ในหน่วยความจำ

4. เนื่องจากข้อมูลที่ได้หลังจากการคำนวณนี้ ตำแหน่งในการวางข้อมูลเป็นบิทรีเวอร์ซอล(Bit Reversal) ดังนั้นก่อนที่จะทำการการส่งข้อมูลจึงจำเป็นที่จะต้องทำการรีเวอร์สตำแหน่งเสียก่อนซึ่งจะทำการรีเวอรส์ ตำ แ หน่ง ใน ขั้น ตอนนี้

5. ทำการรอจนกว่าข้อมูลชุดใหม่จะเข้ามาจนเต็มก่อนแล้วจึงทำในขั้นตอนถัดไป

6. เมื่อข้อมูลเข้ามาจนเต็มแล้วก็จะทำการส่งข้อมูลขึ้นไปแสดงผลบนเครื่องไมโครคอมพิวเตอร์ และเมื่อส่งข้อมูลจนครบก็จะกลับไปทำงานในข้อ 2 ใหม่

การทำงานของโปรแกรมบริการอินเตอร์รัพท์การรับข้อมูลอนุกรม (RINT INTERRUPT SERVICE ROUTINE FLOW DIAGRAM)

1. จากรูปที่ 5.8 เมื่อเกิดการอินเตอร์รัพท์ขึ้นโปรแกรมจะทำการเก็บค่าของรีจิสเตอร์สถานะไว้

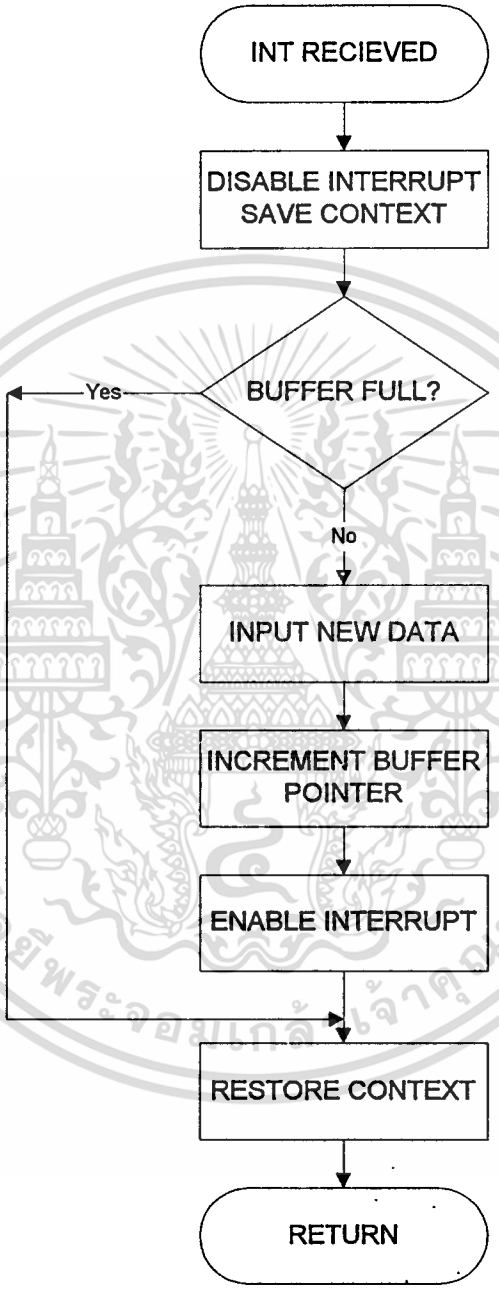
2. ทำการตรวจดูว่ารับข้อมูลเข้ามาจนครบหรือยังถ้าเต็มแล้วให้ใส่ค่าของรีจิสเตอร์ AR7 ให้เป็น 0 แล้วจึงทำการคืนค่ารีจิสเตอร์สถานะมาเป็นดังเดิม

3. ถ้าข้อมูลยังไม่เต็ม ให้ทำการโอนย้ายค่าจาก DRR (Serial Port Data Receive Register) ไปเก็บไว้ในหน่วยความจำตามที่รีจิสเตอร์ AR6 ชี้อยู่

4. เลื่อนตำแหน่งของรีจิสเตอร์ AR6

5. สั่งให้ทำการรับการอินเตอร์รัพท์ได้ดังเดิม

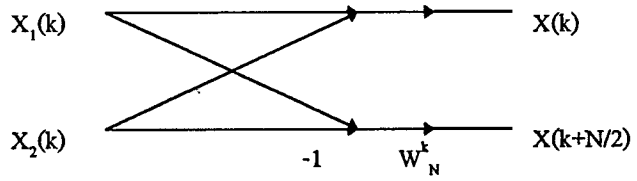
6. คืนค่ารีจิสเตอร์สถานะให้เป็นเหมือนเดิม



รูปที่ 5.8 ผังการทำงานของโปรแกรมบริการอินเตอร์รัพท์การรับข้อมูลอนุกรม

การทำงานของโปรแกรมคำนวณฟาสฟูเรียร์

วิธีการฟาสฟูเรียร์ที่ใช้ในโปรแกรมนี้คือ Complex radix-2 decimation in frequency FFT คือจะมีรูปแบบของบัทเตอร์ฟลาย (Butterfly) ตามรูปที่ 5.9



รูปที่ 5.9 แสดงบัทเตอร์ฟลายของ Complex radix-2 decimation in frequency FFT

เมื่อใช้การสุ่มสัญญาณจำนวน 256 จุด จะทำให้

จำนวนของ Stage เท่ากับ $\log_2 256=8$ Stages

จำนวนของบัทเตอร์ฟลายในแต่ละ Stage เท่ากับ 256/2=128 บัทเตอร์ฟลาย

จำนวนของการคูณทางคอมเพล็ก CMULT = $N/2 \log_2 N$

CMUL = $128 \log_2 256 = 1024$ ครั้ง

จำนวนของการบวกทางคอมเพล็ก CADD = $N \log_2 N$

CADD = $256 \log_2 256 = 2048$ ครั้ง

และค่าของทวิดเดิลแฟกเตอร์(Twiddle Factor) เราสามารถหาได้จาก

$$W^n_N = e^{-2j\pi n/N}$$

โดยที่ค่า n เราหาได้จากตาราง 5.1

m	n										
1	0										
2	0	64									
3	0	32	64	96							
4	0	16	32	48	64	80	96	112			
5	0	8	16	24	32	40	48	56	64	72	80.....120
6	0	4	8	12	16	20	24	28	32	36	40.....124
7	0	2	4	6	8	10	12	14	16	18	20.....126
8	0	1	2	3	4	5	6	7	8	9	10 127

ตารางที่ 5.1 แสดงค่าของ n ในแต่ละ Stage

การทำงานของโปรแกรมคำนวณฟาสฟูเรียร์จากรูป 5.10

1. กำหนดให้รีจิสเตอร์ AR0 เท่ากับจำนวนของการสุ่มสัญญาณ ซึ่งเท่ากับ 128

2. ให้รีจิสเตอร์ AR1 และ AR2 ทำหน้าที่เป็นตัวชี้ในหน่วยความจำในส่วนของบัทเตอร์ฟลาย รีจิสเตอร์ AR3 เป็นตัวชี้ตำแหน่งของค่าทวิดเดิลแฟคเตอร์ และ รีจิสเตอร์ AR4 เป็นตัวนับจำนวนของบัทเตอร์ฟลาย

3. คำนวณให้ส่วนบนของบัทเตอร์ฟลายเท่ากับ AR1+(2*AR0)

4. ตรวจดูว่าค่าในรีจิสเตอร์ AR4 มีค่าเท่ากับ ศูนย์หรือไม่เป็นการตรวจดูว่าโปรแกรมทำการคำนวณบัทเตอร์ฟลายครบหรือยังถ้าครบแล้ว รีจิสเตอร์ AR4 จะเท่ากับศูนย์

5. ถ้ารีจิสเตอร์ AR4 เท่ากับศูนย์แล้วลำดับต่อไปจะทำการเปลี่ยนค่าในรีจิสเตอร์ AR0 โดยให้ A R0 = A R0+AR0 แบบบิทรีเวอร์ซอล

6. ตรวจดูว่าค่าในรีจิสเตอร์ AR0 มีค่าเท่ากับศูนย์หรือไม่ถ้าเท่ากับศูนย์ก็จะเสร็จสิ้นการคำนวณฟาสฟูเรียร์ แต่ถ้าไม่เท่ากับศูนย์ก็จะกลับไปทำงานในข้อที่ 3 อีก

7. ในกรณีที่ค่ารีจิสเตอร์ AR4 ไม่เท่ากับศูนย์แล้วโปรแกรมจะลดค่าของ AR4 ออกไปเท่ากับ 1 และทำการจัดค่ารีจิสเตอร์ที่ทำหน้าที่ชี้ตำแหน่งในทวิดเดิลแฟคเตอร์ก็คือรีจิสเตอร์ AR3 นั่นเอง จากนั้นให้ค่ารีจิสเตอร์ AR5 = 1+AR0 เป็นการบวกแบบบิทรีเวอร์ซอล

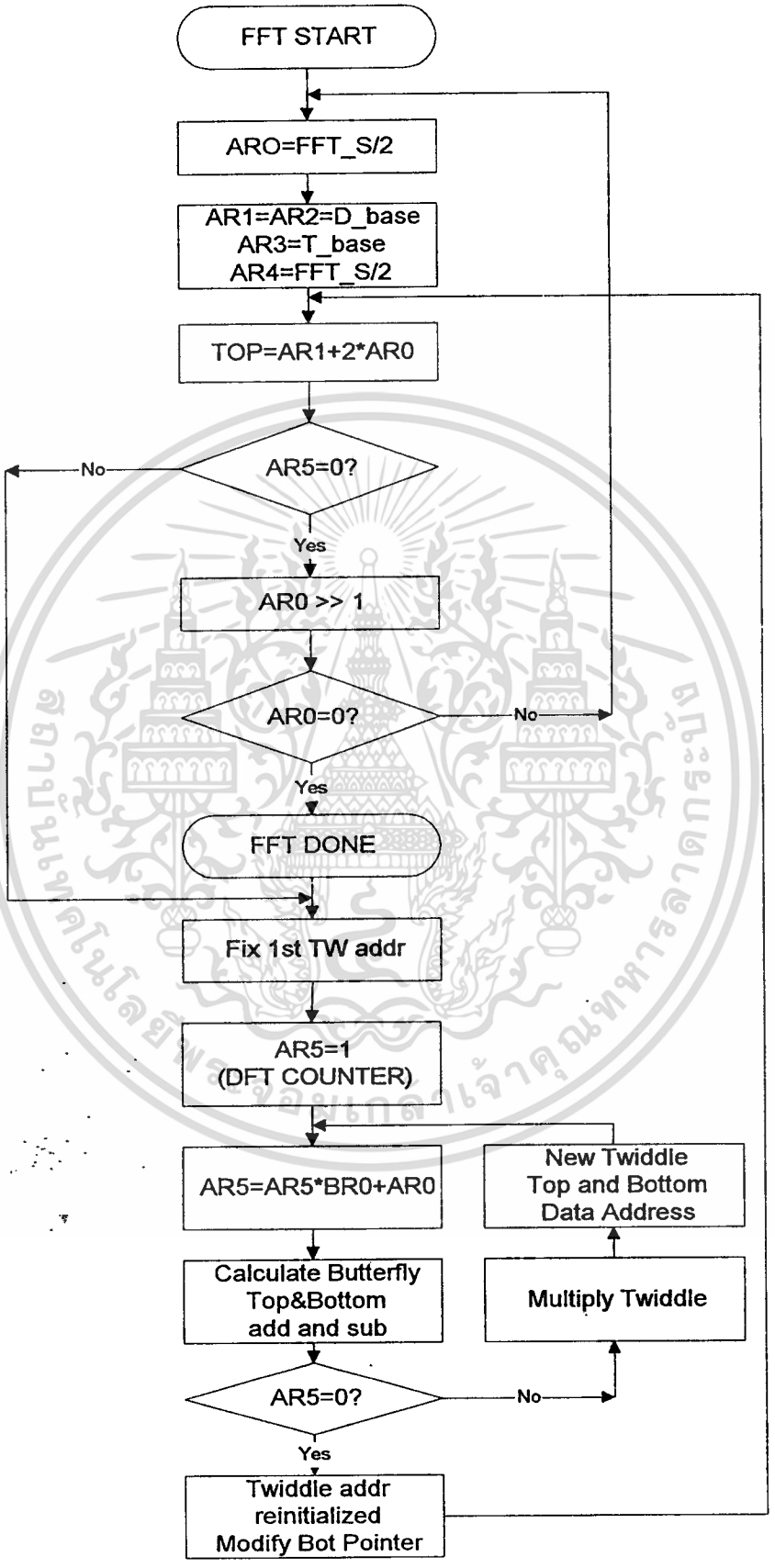
8.ทำการคำนวณบัทเตอร์ฟลาย และคำนวณค่าในรีจิสเตอร์ AR5= AR5+BR0 แบบรีเวอร์ซอล

9. ตรวจดูว่าค่าในรีจิสเตอร์ AR5 เท่ากับศูนย์หรือไม่

10. ถ้าค่าในรีจิสเตอร์ AR5 มีค่าไม่เท่ากับศูนย์ให้ทำการคูณค่าที่ได้จากบัทเตอร์ฟลายกับค่าของทวิดเดิลแฟคเตอร์

11. จัดค่าตำแหน่งของทวิดเดิลแฟคเตอร์และตำแหน่งของบัทเตอร์ฟลายใหม่และกลับไปคำนวณบัทเตอร์ฟลายในข้อ 8

12. แต่ถ้าค่าในรีจิสเตอร์ AR5 มีค่าเท่ากับศูนย์แล้วให้ตั้งค่าตำแหน่งของบัทเตอร์ฟลายและทวิดเดิลแฟคเตอร์ใหม่และกลับไปทำในข้อที่ 3



รูปที่ 5.10 แสดงไดอะแกรมการทำงานของการคำนวณฟาสฟูเรียร์

บทที่ 6

การทดลองและผลการทดลอง

<u>การทดลอง</u> **วัดความเร็วการคำนวณ FFT**

<u>วัตถุประสงค์</u> เพื่อหาเวลาในการทำงาน FFT ขนาดต่าง ๆ บนเครื่อง DSP บอร์ด

<u>วิธีการทดลอง</u> 1. ต่อวงจรเพิ่มเติมกับ Board DSP

2. ใช้ ออสซิลโลสโคปจับขาสัญญาณ D4 ที่ พอร์ต #0

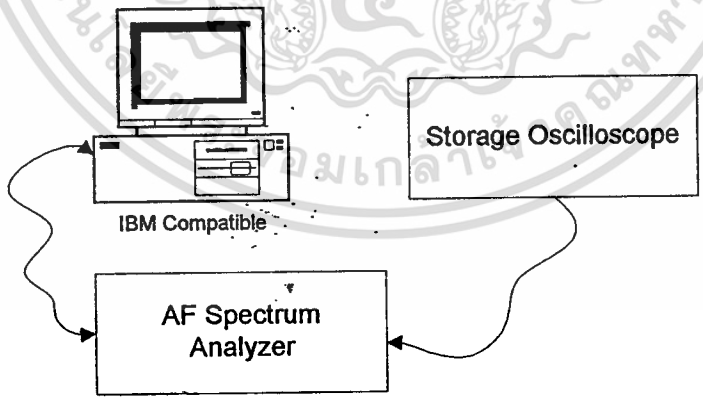
3. ดาวน์โหลดไฟล์ FFT สู่บอร์ด DSP โดยเพิ่มโปรแกรม OUT PORT ทำให้แอลอีดีติดขณะที่กำลังทำการคำนวณ และทำให้แอลอีดีดับขณะที่กำลังทำการส่งข้อมูลขึ้นมาบนเครื่องไมโครคอมพิวเตอร์ โดยคำสั่งที่เพิ่มเข้าไปขณะที่กำลังทำการคำนวณฟาสฟูเรียร์คือ

```
larp    AR0
lrlk    AR0,00
out     *,00
```

และโปรแกรมที่เพิ่มเข้าไปขณะที่ส่งข้อมูลที่คำนวณเสร็จแล้วขึ้นไปที่เครื่องไมโครคอมพิวเตอร์คือ

```
larp    AR0
lrlk    AR0,0xffff
out     *,00
```

4. อ่านค่าแล้วบันทึกผลการทดลอง



รูปที่ 6.1 แสดงการทำการทดลอง

ผลการทดลอง

FFT	เวลาที่ใช้
64 จุด	4 mS
128 จุด	8.12 mS
256 จุด	16.5 mS
512 จุด	33 mS

สรุปผลการทดลอง

ภาพที่อ่านได้จากออสซิสโลสโคป คือ เมื่อมีการเริ่มคำนวนจะให้เอาท์พุทพอร์ตเป็น 0000H และเมื่อทำ FFT เสร็จให้เอาท์พุท พอร์ต FFFFH คาบเวลาที่แรงดันเป็นศูนย์คือช่วงเวลาที่ในการคำนวนยิ่งความละเอียดของการทำ FFT มาก เวลาที่ใช้จะมากขึ้น โดยเวลาที่ใช้ไปส่วนใหญ่ใช้จากการอินเตอรัพท์ในการรับข้อมูลจากพอร์ทสื่อสารอนุกรมดังนั้นเราสามารถ เนื่องจากเวลาที่วัดได้นี้เป็นผลรวมของเวลาระหว่างการอินเตอรัพท์รับข้อมูลและการคำนวณฟาสฟูเรียร์ ดังนั้นเราสามารถคำนวณหาเวลาที่แท้จริงในการคำนวณได้คือ

เวลาที่ใช้ในการคำนวณ = เวลาที่วัดได้ - เวลาที่ใช้ในการอินเตอรัพท์รับข้อมูลทั้งหมด

สำหรับเวลาที่ใช้ในการอินเตอรัพท์รับข้อมูลนั้นระยะเวลาต่าง ๆ จะไม่เท่ากัน จะขึ้นอยู่กับว่าความระเอียดในการคำนวณนั้นเป็นเท่าไรและนำมาคูณกับระยะเวลาในการทำการอินเตอรัพท์หนึ่งครั้งจึงเท่ากับเวลาทั้งหมดในการอินเตอรัพท์รับข้อมูล

การคำนวณ

คำสั่งที่ใช้ในการตอบสนองการอินเตอรัพท์มีดังนี้

```
        b       RINT                (300nS)
        sst1    STAT1               (100nS)
        larp    AR7                 (100nS)
        banz    more_buf,*-,AR6             (300nS)
        lark    AR7,0               (100nS)
        lst1    STAT1               (100nS)
        ret                         (300nS)
more_buf
        sacl    ACCU_lo             (100nS)
        sach    ACCU_hi             (100nS)
        zalh    *                   (100nS)
        sach    DXR                 (100nS)
        lac     DRR                 (100nS)
        bit     TEMPX,15            (100nS)
```

```
        bbz     NO_NVRT         (300nS)
        neg                     (100nS)
NO_NVRT
        sacl    *+              (100nS)
        lac     TEMPX           (100nS)
        xork    1               (200nS)
        sacl    TEMPX           (100nS)
        zalh    ACCU_hi         (100nS)
        adds    ACCU_lo         (100nS)
        lst1    STAT1           (100nS)
        eint                    (100nS)
        ret                     (300nS)
```

ฉะนั้นเวลารวมในการอินเตอรัพท์ 1 ครั้งจะใช้เวลาประมาณ

t_{int} = (3000nS*n)+500nS

โดยที่ค่าของ n คือค่าของความละเอียดที่ใช้ในการคำนวณ

ดังนั้นเราสามารถหาคาบเวลาในการคำนวณฟาสฟูเรียร์ได้คือ

ที่ความละเอียด 64 จุด

เวลาที่ใช้ในการคำนวณ = 4mS-[(3000nS*64)+500nS]

= 3.8075 mS

ที่ความละเอียด 128 จุด

เวลาที่ใช้ในการคำนวณ = 8.12mS-[(3000nS*128)+500nS]

= 7.7355 mS

ที่ความละเอียด 256 จุด

เวลาที่ใช้ในการคำนวณ = 16.5mS-[(3000nS*256)+500nS]

= 15.7315 mS

ที่ความละเอียด 512 จุด

เวลาที่ใช้ในการคำนวณ = 33mS-[(3000nS*512)+500nS]

= 31.4635 mS

บทที่ 7
สรุปและวิจารณ์

7.1 สรุป

เครื่องวิเคราะห์แถบความถี่โดยใช้หน่วยประมวลผล TMS320C26 นี้สร้างขึ้นเพื่อการประมวลผลวิเคราะห์แถบความถี่ โดยระบบประมวลผลสัญญาณเชิงเลข โดยมีขอบเขตที่วางไว้ในขั้นต้นคือสามารถแสดงแถบความถี่ของสัญญาณที่ป้อนเข้าไปสามารถแสดงค่าของความถี่และแสดงค่าของแรงดันได้และยังสามารถหยุดภาพได้โดยกดปุ่ม pause และสามาถที่จะทราบค่าความถี่และแรงดันได้โดยการเลื่อนเมาท์ไปตรงตำแหน่งที่ต้องการจะวัด

จากการที่ได้ศึกษาทดลองและสร้างปรากฏว่าผลยังไม่เป็นที่น่าพอใจเท่าที่ควรเพราะยังไม่สามารถวัดความถี่ได้สูงถึง 20KHz เพราะว่าความถี่สูงสุดในการสุ่มสัญญาณมีเพียง 20KHz เท่านั้น

ข้อดีของเครื่องวิเคราห์แถบความถี่นี้คือมีราคาถูกกว่าเครื่องที่ขายตามท้องตลาดแต่ประสิทธิภาพสามารถนำมาทดแทนกันได้

7.2 ปัญหาที่พบ

7.2.1 ปัญหาในส่วนของฮาร์ดแวร์

ปัญหา ไม่สามารถทำการติดต่อติดต่อกับหน่วยความจำภายนอกได้เนื่องจากเวลาในการเข้าถึงข้อมูลของหน่วยความจำที่มีอยู่มีมากทำให้ต้องทำการ Wait State

แนวทางแก้ไข เปลี่ยนมาใช้หน่วยความจำแคชแทน ทำให้ใช้เวลาในการเข้าถึงข้อมูลน้อยลง

7.2.2 ปัญหาในส่วนของซอฟท์แวร์

ปัญหา หาฟังก์ชั่นในการควบคุมยาก

แนวทางแก้ไข หาคอมโปแนนซ์จากอินเทอร์เนตมาใช้งาน

ปัญหา ในการเขียนโปรแกรมและการติดต่อกับ DSK ซึ่งไม่เคยศึกษามาก่อน

แนวทางแก้ไข ต้องใช้เวลาในการศึกษาและทดลองมาก

7.3 แนวทางการพัฒนา

1. เครื่องวิเคราะห์แถบความถี่นี้สามารถนำไปประยุกต์ใช้งานทางด้านอื่นได้อีกตามต้องการโดยการเขียนโปรแกรมขึ้นใหม่และคอมไพล์โดยตั้งชื่อเหมือนกับโปรแกรมที่ใช้ในการโหลดข้อมูลลงไป เมื่อเปิด เครื่องเครื่องไมโครคอมพิวเตอร์ก็จะส่งไฟล์ที่ตั้งชื่อแล้วนั้นไปให้กับ DSK จากนั้น DSK ก็จะเริ่มทำงานเอง ซึ่งมีประโยชน์คือใช้ในการศึกษาการทำงานและศึกษาการเขียนโปรแกรมบนโปรเซสเซอร์ TMS320C26 และตระกูลอื่น ๆ และสามารถนำไปใช้งานได้อย่างแพร่หลาย เพราะในปัจจุบัน ดีเอสพี เข้ามามีบทบาทมากในเครื่องใช้ไฟฟ้าต่าง ๆ ไม่ว่าจะเป็นโทรศัพท์มือถือ โมเด็ม แฟกซ์ และอื่น ๆ อีกมากมาย

2. ทำการขยายย่านในการวัดให้ได้มากขึ้นโดยการเพิ่มความเร็วของการสุ่มตัวอย่าง

3. ทำการเพิ่มบอดเรทในการส่งข้อมูลเพื่อที่โปรแกรมจะได้สามารถกลับไปคำนวณฟาสฟูเรียร์ได้เร็วขึ้น

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงจะสำเร็จลุล่วงไม่ได้หากขาดผู้ช่วยนั่นคือ เพื่อนๆทุกคนได้แก่ เซ่ , กอส , วัช , แบน , โย่ง , ปอ , เอ้ , อ๊อด , วัฒน์ , ปุ้ย ที่ช่วยเอื้อเฟื้อเครื่องมืออุปกรณ์ และแนวความคิดในการแก้ปัญหาต่างๆ ขอบคุณ ดร.ไกรสิน และน้องอาร์ตที่เอื้อเฟื้อห้องทดลองเพื่อใช้ในการทดลองเวลาในเวลาที่ไม่มีเครื่องออสซิลโลสโคปใช้ ที่ขาดไม่ได้คือท่านอาจารย์ เกรียงไกร วงศ์โรจน์ภรณ์ และ ผศ.ดร. สุวิพล สิทธิชีวภาค ที่สนับสนุนห้องทดลอง และให้คำปรึกษา และ สุดท้ายที่ขาดไม่ได้ก็คือ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ผู้ให้กำเนิดและสนับสนุนค่าใช้จ่ายในการทำงานชิ้นนี้เสมอมาครับ

เอกสารอ้างอิง

1. Texas Instrument, "TMS320C2X User's Guide" : Texas Instrument,Inc. , 1993

2. Texas Instrument, "TMS320C2X DSP Starter Kit User's Guide" : Texas Instrument,Inc. , 1993

3. Lonnie C. Ludeman, "Fundamentals of Digital Signal Processing"John Wiley & Sons, Inc. , 1986

4. Dan Osier,Steve Grovman,Steve Batson, "Teach Yourself Delphi 2 in 21 days" : Sams Publish. , 1996

FFT.ASM

```
;-------------------------------------------------
Pragram Spectrum Analyzer
;-------------------------------------------------
YES             .set   1          ;
NO              .set   0          ;
FFT_S           .set   256    ;
FFT_S-1         .set   255    ;
FFT_S/2         .set   128    ;
(FFT_S/2)-1     .set   127    ;
;-------------------------------------------------;
AIC_1           .set   0x0C18  ;TB =TA = 6    0000110000011000=0x0C18
AIC_2           .set   0x0205  ;TA'=TA'= 1    0000001000000101=0x0205
AIC_3           .set   0x264e  ;RB =TB = 0x13 0010011001001110=0x264c
AIC_CMD         .set   0x0003 ;  COMMAND    0000000000000011=0x0083
;-------------------------------------------;
BCMD            .set   0xFA10    ;JUMP CMD
BXMIT           .set   0xFA12  ;JUMP XMIT
BXMIT16         .set   0xFA14  ;JUMP XMIT16
BRECV           .set   0xFA16  ;JUMP RECV
BRECV16         .set   0xFA18  ;JUMP RECV16
BCXMIT          .set   0xFA1A  ;JUMP CXMIT
;-----------------------------------------
STAT1           .set   0x72   ;
ACCU_lo         .set   0x78   ;
ACCU_hi         .set   0x79   ;
REAL            .set   0x7a   ;
IMAG            .set   0x7b   ;
TEMPX           .set   0x7c   ;
AUX0            .set   0x7d
AUX1            .set   0x7e
;-------------------------------------------------
;   SECONDARY VECTOR TABLE LOACTED IN B0 PROGRAM RAM
;-------------------------------------------------
```

```
        .include "mmregs.asm" ;       >, USERCODE SHOULD NOT OVERWRITE DSKD  <
        .ps    0xfa00         ;       > VECTORS.  ON LOAD, INT2 IS RESTORED <
    ;B     start                       ;RS   > BY DSKD, BUT TRAP IS NOT          <
    ;B     start                       ;INT0
    ;B     start                       ;INT1
    ;B     start                       ;INT2  > DSKD LOAD IGNORES INT2 VECTOR
    ;B     start                       ;TINT


        .ps    0fa0ah                  ;
        B      RINT                    ;RINT  Branch to receive interrupt routine
        eint                           ;XINT  XINT is only for timing, so just return
        ret                    ;
   ; Begin TRAP/DSKD Kernal  ;DSKD load does not restore this code!
;-------------------------------------------------
;   APPLICATION CODE IS LOCATED ABOVE DSKD KERNAL
;-------------------------------------------------
        .ps    0xFB00                  ;
        .entry                         ;
;-------------------------------------------------
start:  sxf
        ssxm
        sovm                           ; catch accumulator overflows
        ldpk   0                       ; All direct addressing is to MMRs and B2
        fort   0                       ; Serial port : 16 bit
        rtxm                           ; ext. FSX
        sfsm           ;               ; burst mode
        lack   0x80                    ; AIC reset by pulsing /BR (Global Data)
        sach   DXR                     ; send 0 to DXR (AIC)
        sacl   GREG                    ; 256 * 100 nS /BR pulse
        lrlk   AR0,0xFFFF              ;
        rptk   255                     ; read junk from address 0xFFFF
        lac    *,0,AR0                 ;
        conf   1                       ; B1,B3 as DRAM if direct bootload
```

```
;  ;-------------------------
   AIC_RS lack    0x20              ; Turn on XINT
          sacl    IMR               ;
          idle                      ;
          lalk    AIC_1             ; Load each AIC configuration word
          call    AIC_2nd           ; and load it into the AIC
          lalk    AIC_2             ;
          call    AIC_2nd           ;
          lalk    AIC_3             ;
          call    AIC_2nd           ;
          lalk    AIC_CMD           ;
          call    AIC_2nd           ;
   ;-----------------------------------------------
          lark    AR7,0             ; Buffer initialy filled
          lack    0x10              ; AIC RINT
          sacl    IMR               ; where INT0 indicates EOC (End Of Conv)
          ;-----------------------------------------------
          lark    AR7,0             ; Buffer initialy filled
   FFT:   lrlk    AR0,FFT_S/2       ;
          larp    AR0               ; start FFT with AR0=FFTSize
   new_stg lrlk   AR1,_D_base              ; AR1 is the TOP BFLY address
          lrlk    AR2,_D_base              ; AR2 is the BOT BFLY address
          lrlk    AR3,_T_base+1     ;AR3 is the TWiddle pointer
          lrlk    AR4,FFT_S/2       ; AR4 counts DFT blocks
          b       n_DFT2,*,AR1              ;
   DFT:   mar     *BR0+,AR5         ; complete circular buffer for TW's
          lark    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:  lac     *,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
        lac     *,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
        larp    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:         larp    AR2             ;Transform REAL & IMAG to log magnitude
        lrlk    AR2,_D_base             ;AR3=FFT data pointer
        lrlk    AR3,FFT_S-1             ;AR5=FFT loop counter
        lrlk    AR0,FFT_S
        ;-------------------------------------------------;
        ; WINDOW: Performs post FFT raised cosine windowing!    ;
        ; This is done by using the frequency coefficients of the  ;
        ; window in a convolution filter of the spectrum.       ;
        ;-------------------------------------------------;
        ;mar    *BR0+                   ; don't start at DC
more_MAG
        mar     *BR0-                   ; -IMAG[-1]  1-COS(nwt/N)     + 1
        lac     *BR0+,15                ;  IMAG[-0]  filter by post   |
        subh    *BR0+                   ; +IMAG[+1]  convolution   <--+++-->
        add     *BR0-,15                ;  IMAG                    + + -.5
        sach    IMAG                    ;
        mar     *+                      ;  REAL
        mar     *BR0-                   ; -REAL[-1]
        lac     *BR0+,15                ;  REAL[-0]  X[-1] -2*X[0] + X[1]
        subh    *BR0+                   ; +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
        addk    0x1                     ;Set up a floor value; log(0) not legal!
        lark    AR1,22                  ;pre-scaling exponent shifts Y axis
        rptk    31                      ;
        norm    *-                      ;
        larp    AR2                     ;
```

```
        mar     *BR0-           ;-REAL;dump log(f) into oldest REAL (odd addr)
        sach    *,2             ;clr explicit 1.0 and sign bit from mant
        zals    *               ;load into ACCU_lo
        sar     AR1,*           ;then append exponent (AR1)
        addh    *               ;
        rptk    10              ;jam result into ACCU_hi
        sfl                     ;If needed, Use ADDH to saturate overflow
;       sach    *               ;
;       addh    *               ;
        sach    *               ;
        lac     *               ;
        andk    0xfffc,0        ;
        sacl    *BR0+           ; REAL
        mar     *-              ; IMAG
        mar     *BR0+,AR3       ;+IMAG
        banz    more_MAG,*-,AR2 ;keep going until all done
        ;---------------------------------------
BITREV:         lrlk    AR0,FFT_S       ;Now perform Output bit reversal
        lrlk    AR1,_D_base             ;by moving the magnitude, which
        lrlk    AR2,_D_base+1   ;is in the REAL slots, into the
        lrlk    AR3,FFT_S-1     ;IMAG slots of the FFT data array
more_BR
        lac     *+              ;load the magnitude
        mar     *+,AR1          ;
        sacl    *BR0+,0,AR3             ;move it to an open IMAG slot
        banz    more_BR,*-,AR2          ;more data to move?
        ;---------------------------------------
MOVE_IO
        larp    AR7             ;wait until buffer is full
        banz    MOVE_IO,*,AR2   ;(AR7 is decremented by ISR)
        ;---------------------------------------
        lrlk    AR3,_D_base             ;AR3=FFT data pointer
        lrlk    AR4,_B_base             ;AR4=BUFF data pointer
```

```
        lrlk    AR5,FFT_S-1         ;AR5=FFT loop counter
        lrlk    AR6,_B_base                 ;AR6=ISR BUFF data pointer
        lrlk    AR7,FFT_S-1         ;AR7=ISR BUFF loop counter
        ;-------------------------------------------------
        dint
        zac
        call    DAT2HOST
        larp    AR2
more_IO
        lar     AR2,*,AR3           ;Get A/D value from buffer
        lac     *,0,AR4             ;ACCU= log magnitude (from even address)
        sacl    *+,0,AR3            ;

        rptk    7
        sfr
        ork     1
        call    DAT2HOST
        larp    AR3

        zac                         ;
        sach    *+,0                ;IMAG=0
        sar     AR2,*+,AR5          ;

        banz    more_IO,*-,AR4              ;
        eint                        ; BUFF clear so enable INT's
        b       FFT                 ;
;-------------------------------------------------
RINT:   sst1    STAT1               ;Recover ARP from ARB by LST1 last
        larp    AR7                 ;AR6 = current buffer position
        banz    more_buf,*-,AR6             ;if buffer is full RET w/o EINT
        lark    AR7,0               ;
        lst1    STAT1               ;
        ret                 ;
```

```
more_buf                                ;
        sacl    ACCU_lo                 ;Use NORM start val to adj Y offset
        sach    ACCU_hi                 ;post log convert scaling ajsts magnitude
        zalh    *                       ;Get value
        sach    DXR                     ;
        ;-------------------------------------------------------
        lac     DRR                     ;
        bit     TEMPX,15                ;Inverting every other input aliases the
        bbz     NO_NVRT                 ;frequency domain, swapping DC and Nyquist!
        neg                             ;
NO_NVRT                                 ;
        sacl    *+                      ;<<< store DRR, and point to next
        lac     TEMPX                   ;
        xork    1                       ;
        sacl    TEMPX                   ;
        zalh    ACCU_hi                 ;
        adds    ACCU_lo                 ;
        lst1    STAT1                   ;
        eint                            ;
        ret                     ;
*****************************************************************
AIC_2nd adlk    6,15                    ;set ACCU_hi = 3 for secondary XMIT
                idle                    ;Wait for a XINT
        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                            ;
        eint                            ;
        ret                     ;
*****************************************************************
DAT2HOST
```

```
        sacl    ACCU_lo         ;
        sach    ACCU_hi         ;
        sar     AR0,AUX0        ;
        sar     AR1,AUX1        ;
        call    BXMIT           ;
        zals    ACCU_lo         ;
        addh            ACCU_hi         ;
        lar     AR0,AUX0        ;
        lar     AR1,AUX1        ;
        ret                     ;
;=================================================================
====
        .listoff        ;
        .ds     0x400           ;NOTE: Twiddles are relocated to
        .include "dsk_twid.asm"         ;0x400 (B2) using CONF 1
        .liston
        .end
```

MMREGS.ASM

```
;-----------------------------------------------;
; MMREGS.ASM                                    ;
;     TMS320Cxx DSP Applications                ;
;                                               ;
; Contains a list of commonly used values       ;
;-----------------------------------------------;
DRR     .set    0
DXR     .set    1
TIM     .set    2
PRD     .set    3
IMR     .set    4
GREG    .set    5
B2_D    .set    060h
B0_D    .set    0200h
B1_D    .set    0400h
B4_D    .set    08000h
B3_D    .set    0600h
B0_P    .set    0FA00h
B1_P    .set    0FC00h
B3_P    .set    0FE00h
; use a comment to end .include files ending with .set
; to avoid an ERROR.  Also, comment must start in column 2
```

DSK_COMM.ASM

```
;-------------------------------------------------------
        b       CMD             ;FA10  JUMP table to KLKRNL routines
        b       XMIT            ;FA12
        b       XMIT16          ;FA14
        b       RECV            ;FA16
        b       RECV16          ;FA18
        b       CXMIT           ;FA1A
        b       XMIT1           ;FA1C
;-------------------------------------------------------
KLKRNL: dint                    ;
        sxf                     ;initialize line
        ldpk    0               ;
        conf    1               ;
        zac                     ;
        sacl    IMR             ;
        lac     BITLEN,1        ;2 x
        add     BITLEN          ;3 x
        sfr                     ;1.5 x
        subk    7               ;best BITLEN skew
        sacl    BITLEN          ;Store adjusted BITLEN
        sxf

CMD     call    RECV,*,AR0      ;get 8 bit CMD from host
        bz      CMD             ;
        subk    1               ;
        bz      W_PS            ;1 = write program space
        subk    1               ;
        bz      R_PS            ;2 = read program space
        subk    1               ;
        bz      W_DS            ;3 = write data space
        subk    1               ;
        bz      R_DS            ;4 = read data space
        subk    1               ;
```

```
        bnz     CMD             ;5 = Execute program at next word
        call    RECV16          ;
        lac     WORD16          ;
        bacc                    ;
;=============================================================
; W_PS & R_PS are used to access program memory
;=============================================================
W_PS    call    KRNL            ;
W_PS2   call    RECV16          ;copy length words to PS
        sacl    WORD            ;
        lac     PADDR           ;ACCU=src address
        tblw    WORD            ;
        addk    1               ;
        sacl    PADDR           ;
        larp    AR3             ;
        banz    W_PS2,*-,AR2    ;
        b       CMD             ;

R_PS    call    KRNL            ;
R_PS2   lac     PADDR           ;
        tblr    TEMP            ;
        addk    1               ;
        sacl    PADDR           ;
        lac     TEMP            ;
        call    XMIT16          ;
        larp    AR3             ;
        banz    R_PS2,*-,AR2    ;next word
        b       CMD             ;
;=============================================================
; W_DS & R_DS are used to access data memory
;=============================================================
W_DS    call    KRNL            ;
W_DS2   call    RECV16          ;build data word
```

```
        larp    AR2             ;       ;
        sacl    *+,0,AR3                ;write the word   <<< missing shift!!!
        banz    W_DS2,*-,AR2    ;
        b       CMD             ;

R_DS    call    KRNL            ;
R_DS2   larp    AR2             ;get word
        lac     *+,0            ;
        call    XMIT16,*        ;
        larp    AR3             ;
        banz    R_DS2,*-        ;next word
        b       CMD             ;
;=============================================================
; KRNL is common to all block transfers
;=============================================================
KRNL    call    RECV16          ;1 st word is address
        lar     AR2,WORD16      ;
        sar     AR2,PADDR       ;
        call    RECV16          ;2 nd word is length-1
        lar     AR3,WORD16      ;
        larp    AR2             ;
        ret                     ;
;=============================================================
; RECV returns word in bottom 8 bits of ACCL
;=============================================================
RECV16: call    RECV            ;recv LO
        sacl    WORD16          ;
        call    RECV            ;recv HI
        rptk    7               ;
        sfl                     ;build HI+LO
        or      WORD16          ;
        sacl    WORD16          ;
        ret                     ;
```

```
RECV    zac                     ;
        sfr                     ;
        lark    AR1,8           ;take 9 samples (START >> shifted out!)
        lar     AR0,BITLEN      ;
        larp    0               ;
        mar     *BR0+           ;
WAIT    bioz    NBIT,*,AR0      ;1st time wait 1/2 BITLEN extra
        b       WAIT            ;
NBIT    banz    $,*-            ;wait BITLEN
        sfr                     ;shift to new posn
        bioz    RB0,*,AR1       ;test bit
RB1     addk    0x80            ;if 1, add new bit
RB0     lar     AR0,BITLEN      ;
        banz    NBIT,*-,AR0     ;last bit?
        andk    0xff
        bioz    $               ;wait for STOP before RET
        ret                     ;ACCU = RECEIVED CHAR
;===============================================================
; XMIT assumes word is in bottom 8 bits of ACCL
;===============================================================
XMIT16: call    XMIT            ;xmit LO
        call    XMIT            ;xmit HI
        ret                     ;
;---------------------------------------------------------------
XMIT    sacl    TEMPx           ;
        call    RECV            ;Handshake with slow host
        lac     TEMPx           ;
CXMIT   sfl                     first bit is 0
        lark    AR1,8           ;1 start, 8 DATA
SENDBIT ror                     ;TEMP = .....X XXXXXXXXs
        bnc     XB0,*,AR0       ;
        sxf                     ;
```

```
        b       XB1,*,AR0               ;
XB0     rxf                             ;
XB1     lar     AR0,BITLEN              ;timout BITLEN
        banz    $,*-                    ;
        larp    AR1                     ;
        banz    SENDBIT,*-,AR0          ;send another bit?
        sxf                             ;
        lar     AR0,BITLEN              ;
        banz    $,*-                    ;
        ret                             ;

;-------------------------------------------------------------------
XMIT1   sacl    TEMPx                   ;
        ;call   RECV                    ;Handshake with slow host
        lac     TEMPx                   ;
CXMIT1  sfl                             ;first bit is 0
        lark    AR1,8                   ;1 start, 8 DATA
SENDBIT1 ror                            ;TEMP = .....X XXXXXXXXs
        bnc     XB01,*,AR0              ;
        sxf                             ;
        b       XB11,*,AR0              ;
XB01    rxf                             ;
XB11    lar     AR0,BITLEN              ;timout BITLEN
        banz    $,*-                    ;
        larp    AR1                     ;
        banz    SENDBIT1,*-,AR0 ;send another bit?
        sxf                             ;
        lar     AR0,BITLEN              ;
        banz    $,*-                    ;
        ret                             ;
```

MMREGS.ASM

```
;-----------------------------------------;
; MMREGS.ASM                              ;
;    TMS320Cxx DSP Applications           ;
;                                         ;
; Contains a list of commonly used values ;
;-----------------------------------------;
DRR     .set    0
DXR     .set    1
TIM     .set    2
PRD     .set    3
IMR     .set    4
GREG    .set    5
B2_D    .set    060h
B0_D    .set    0200h
B1_D    .set    0400h
B4_D    .set    08000h
B3_D    .set    0600h
B0_P    .set    0FA00h
B1_P    .set    0FC00h
B3_P    .set    0FE00h
; use a comment to end .include files ending with .set
; to avoid an ERROR.  Also, comment must start in column 2
```

DSK_COMM.ASM

```
;       DSK_COMM.ASM
;       CMD             CMD 4
;       LENGTH LO       CALL LO
;       LENGTH HI       CALL HI
;       ADDR LO
;       ADDR HI
;       DATA LO
;       DATA HI
;
YES     .set    1
NO      .set    0
KL      .set    1
;-------------------------------------------------------------;
TEMP            .set    0x60    ;
WORD16          .set    0x61    ;
PADDR           .set    0x62    ;
WORD            .set    0x63    ;
TEMPx           .set    0x64    ;
SCRATCH         .set    0x65    ;
BITLEN          .set    0x66    ;RS232 bitwidth in C26 clock ticks
;-------------------------------------------------------------;
        .include "mmregs.asm"
        .ps     0xFA00          ;
        B       KLKRNL          ;FA00  RS
        B       KLKRNL          ;FA02   INT0
        B       KLKRNL          ;FA04   INT1
        B       KLKRNL          ;FA06   INT2
        B       KLKRNL          ;FA08  TINT
        ret                     ;FA0A  RINT
        nop                     ;
        ret                     ;FA0C  XINT
        nop                     ;
        B       KLKRNL          ;FA0E  TRAP
```

```
;-------------------------------------------------------------
        b       CMD             ;FA10  JUMP table to KLKRNL routines
        b       XMIT            ;FA12
        b       XMIT16          ;FA14
        b       RECV            ;FA16
        b       RECV16          ;FA18
        b       CXMIT           ;FA1A
        b       XMIT1           ;FA1C
;-------------------------------------------------------------
KLKRNL: dint                    ;
        sxf                     ;initialize line
        ldpk    0               ;
        conf    1               ;
        zac                     ;
        sacl    IMR             ;
        lac     BITLEN,1        ;2 x
        add     BITLEN          ;3 x
        sfr                     ;1.5 x
        subk    7               ;best BITLEN skew
        sacl    BITLEN          ;Store adjusted BITLEN
        sxf

CMD     call    RECV,*,AR0      ;get 8 bit CMD from host
        bz      CMD             ;
        subk    1               ;
        bz      W_PS            ;1 = write program space
        subk    1               ;
        bz      R_PS            ;2 = read program space
        subk    1               ;
        bz      W_DS            ;3 = write data space
        subk    1               ;
        bz      R_DS            ;4 = read data space
        subk    1               ;
```

```
        bnz     CMD             ;5 = Execute program at next word
        call    RECV16          ;
        lac     WORD16          ;
        bacc                    ;
;=============================================================
; W_PS & R_PS are used to access program memory
;=============================================================
W_PS    call    KRNL            ;
W_PS2   call    RECV16          ;copy length words to PS
        sacl    WORD            ;
        lac     PADDR           ;ACCU=src address
        tblw    WORD            ;
        addk    1               ;
        sacl    PADDR           ;
        larp    AR3             ;
        banz    W_PS2,*-,AR2    ;
        b       CMD             ;

R_PS    call    KRNL            ;
R_PS2   lac     PADDR           ;
        tblr    TEMP            ;
        addk    1               ;
        sacl    PADDR           ;
        lac     TEMP            ;
        call    XMIT16          ;
        larp    AR3             ;
        banz    R_PS2,*-,AR2    ;next word
        b       CMD             ;
;=============================================================
; W_DS & R_DS are used to access data memory
;=============================================================
W_DS    call    KRNL            ;
W_DS2   call    RECV16          ;build data word
```

```
        larp    AR2             ;       ;
        sacl    *+,0,AR3                ;write the word   <<< missing shift!!!
        banz    W_DS2,*-,AR2    ;
        b       CMD             ;

R_DS    call    KRNL            ;
R_DS2   larp    AR2             ;get word
        lac     *+,0            ;
        call    XMIT16,*        ;
        larp    AR3             ;
        banz    R_DS2,*-        ;next word
        b       CMD             ;
;=============================================================
; KRNL is common to all block transfers
;=============================================================
KRNL    call    RECV16          ;1 st word is address
        lar     AR2,WORD16      ;
        sar     AR2,PADDR       ;
        call    RECV16          ;2 nd word is length-1
        lar     AR3,WORD16      ;
        larp    AR2             ;
        ret                     ;
;=============================================================
; RECV returns word in bottom 8 bits of ACCL
;=============================================================
RECV16: call    RECV            ;recv LO
        sacl    WORD16          ;
        call    RECV            ;recv HI
        rptk    7               ;
        sfl                     ;build HI+LO
        or      WORD16          ;
        sacl    WORD16          ;
        ret                     ;
```

```
RECV    zac                     ;
        sfr                     ;
        lark    AR1,8           ;take 9 samples (START >> shifted out!)
        lar     AR0,BITLEN      ;
        larp    0               ;
        mar     *BR0+           ;
WAIT    bioz    NBIT,*,AR0      ;1st time wait 1/2 BITLEN extra
        b       WAIT            ;
NBIT    banz    $,*-            ;wait BITLEN
        sfr                     ;shift to new posn
        bioz    RB0,*,AR1       ;test bit
RB1     addk    0x80            ;if 1, add new bit
RB0     lar     AR0,BITLEN      ;
        banz    NBIT,*-,AR0     ;last bit?
        andk    0xff
        bioz    $               ;wait for STOP before RET
        ret                     ;ACCU = RECEIVED CHAR
;===============================================================
; XMIT assumes word is in bottom 8 bits of ACCL
;===============================================================
XMIT16: call    XMIT            ;xmit LO
        call    XMIT            ;xmit HI
        ret                     ;
;---------------------------------------------------------------
XMIT    sacl    TEMPx           ;
        call    RECV            ;Handshake with slow host
        lac     TEMPx           ;
CXMIT   sfl                     first bit is 0
        lark    AR1,8           ;1 start, 8 DATA
SENDBIT ror                     ;TEMP = .....X XXXXXXXXs
        bnc     XB0,*,AR0       ;
        sxf                     ;
```

```
        b       XB1,*,AR0               ;
XB0     rxf                             ;
XB1     lar     AR0,BITLEN              ;timout BITLEN
        banz    $,*-                    ;
        larp    AR1                     ;
        banz    SENDBIT,*-,AR0          ;send another bit?
        sxf                             ;
        lar     AR0,BITLEN              ;
        banz    $,*-                    ;
        ret                             ;

;-------------------------------------------------------------
XMIT1   sacl    TEMPx                   ;
        ;call   RECV                    ;Handshake with slow host
        lac     TEMPx                   ;
CXMIT1  sfl                             ;first bit is 0
        lark    AR1,8                   ;1 start, 8 DATA
SENDBIT1 ror                            ;TEMP = .....X XXXXXXXXs
        bnc     XB01,*,AR0              ;
        sxf                             ;
        b       XB11,*,AR0              ;
XB01    rxf                             ;
XB11    lar     AR0,BITLEN              ;timout BITLEN
        banz    $,*-                    ;
        larp    AR1                     ;
        banz    SENDBIT1,*-,AR0 ;send another bit?
        sxf                             ;
        lar     AR0,BITLEN              ;
        banz    $,*-                    ;
        ret                             ;
```

PLOT.ASM

```
unit plotter;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  CMButton , ExtCtrls, StdCtrls, VSSComm32, Wordcap, SuperTimer, Graphitg,
  ComCtrls, Menus;

type
  TForm1 = class(TForm)
    Program1: TListBox;
    Program2: TListBox;
    SerialPort1: TVSSComm32;
    MSOfficeCaption1: TMSOfficeCaption;
    BufferMemo: TMemo;
    Graph1: TGraphItGraph;
    label1: TLabel;
    label2: TLabel;
    StatusBar1: TStatusBar;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    MainMenu1: TMainMenu;
    StartMenu: TMenuItem;
    Setup1: TMenuItem;
    Start2: TMenuItem;
    Stop: TMenuItem;
    Exit: TMenuItem;
    TopPanel: TPanel;
    StartBtn: TCMBtn;
```

```
{$R *.DFM}
{$H+}
procedure TForm1.StartBtnClick(Sender: TObject);
var  ToRequest: String;
begin
{ initial comm  && download file  }
  SerialPort1.StartComm;
  Form2.Show;
  LoadDSK_Comm;
  LoadFFT;
  Form2.Hide;
{ show status }
StatusBar1.Panels.Items[0].Text :='Comm Opened';

{ Request to Data }
 ToRequest := 'D';
 SerialPort1.WriteCommData (Pchar(ToRequest),length(ToRequest));

{ Plot }
TimerForPlot.Enabled := True;

{—when being plot some button cannot use-}
  PauseBtn.Enabled:= True;
  PauseBtn.Visible:= True;
  ContBtn.Visible:=False;
  StopBtn.Enabled:= True;
  SetupBtn.Enabled:=False;
  Setup1.Enabled:=False;
  StartBtn.Enabled := False;
{————————————————}
end;
```

```
    procedure SetupBtnClick(Sender: TObject);
    procedure COM11Click(Sender: TObject);
    procedure COM21Click(Sender: TObject);
    procedure COM31Click(Sender: TObject);
    procedure COM41Click(Sender: TObject);
    procedure N48001Click(Sender: TObject);
    procedure N96001Click(Sender: TObject);
    procedure N192001Click(Sender: TObject);
    procedure N384001Click(Sender: TObject);
    procedure N1281Click(Sender: TObject);
    procedure N2561Click(Sender: TObject);
    procedure N5121Click(Sender: TObject);
    procedure N10241Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure TimerForPlotTimer(Sender: TObject);
    procedure ExitClick(Sender: TObject);
    procedure StopClick(Sender: TObject);
    procedure SerialPort1ReceiveData(Buffer: Pointer; BufferLength: Word);
    procedure LoadFFT;
    procedure LoadDSK_Comm;
    Function  CharToInt(InChar :Char) : Integer;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Unit3, Unit4, Unit2;
```

```
    PauseBtn: TCMBtn;
    SetupBtn: TCMBtn;
    ExitBtn: TCMBtn;
    ContBtn: TCMBtn;
    StopBtn: TCMBtn;
    About1: TMenuItem;
    Image1: TImage;
    Label10: TLabel;
    Label11: TLabel;
    CommPort1: TMenuItem;
    BaudRate1: TMenuItem;
    COM11: TMenuItem;
    COM21: TMenuItem;
    COM31: TMenuItem;
    COM41: TMenuItem;
    N48001: TMenuItem;
    N96001: TMenuItem;
    N192001: TMenuItem;
    N384001: TMenuItem;
    FFT1: TMenuItem;
    N1281: TMenuItem;
    N2561: TMenuItem;
    N5121: TMenuItem;
    N10241: TMenuItem;
    TimerForPlot: TSuperTimer;
    Memo1: TMemo;

    procedure StartBtnClick(Sender: TObject);
    procedure PauseBtnClick(Sender: TObject);
    procedure ContBtnClick(Sender: TObject);
    procedure StopBtnClick(Sender: TObject);
    procedure Graph1MouseMove(Sender: TObject);
    procedure ExitBtnClick(Sender: TObject);
```

```
procedure TForm1.PauseBtnClick(Sender: TObject);
begin
  PauseBtn.Visible := False;
  ContBtn.Visible := True;

  { Pause to Plot }
  TimerForPlot.Enabled := False;
  StatusBar1.Panels.Items[2].Text := 'Pause Receiving ';
end;

procedure TForm1.ContBtnClick(Sender: TObject);
begin
  ContBtn.Visible := False;
  PauseBtn.Visible := True;
  StatusBar1.Panels.Items[2].Text := 'Receiving Data';
  { Continue to Plot }
  TimerForPlot.Enabled := True;
end;

procedure TForm1.StopBtnClick(Sender: TObject);
begin
  { Stop Plot}
  TimerForPlot.Enabled :=False;
  { Shutdown Communication Port }
  SerialPort1.StopComm;
  StatusBar1.Panels.Items[0].Text :='Comm Closed';
  StatusBar1.Panels.Items[2].Text :='';
  StartBtn.Enabled := True;
  ContBtn.Visible := False;
  PauseBtn.Visible := True;
  PauseBtn.Enabled := False;
  StopBtn.Enabled :=False;
  SetupBtn.Enabled :=True;
```

```
  Setup1.Enabled:=True;
end;

procedure TForm1.Graph1MouseMove(Sender: TObject);
  var  Frequency  : real;
       Amplitude  : real;
begin
    Frequency := ABS(Graph1.XMousePosition);
    Amplitude := Graph1.XMousePosition;
    label1.caption:=FloatToStr(Frequency) + ' Hz';
    label2.caption:=FloatToStr(Amplitude) + ' dB';
end;

procedure TForm1.ExitBtnClick(Sender: TObject);
begin
     Application.Terminate;
end;

procedure TForm1.SetupBtnClick(Sender: TObject);
begin
    StartBtn.Enabled:=False;
    PauseBtn.Enabled:=False;
    StopBtn.Enabled:=False;
    Form3.Visible:=True;
end;

procedure TForm1.COM11Click(Sender: TObject);
begin
   Form3.SerialRdGp.ItemIndex:=0;
   SerialPort1.CommPort:=Com1;
   StatusBar1.Panels.Items[4].Text:='COM 1';
   Com11.Checked:=True;
   Com21.Checked:=False;
```

```
  Com31.Checked:=False;
  Com41.Checked:=False;
end;

procedure TForm1.COM21Click(Sender: TObject);
begin
  Form3.SerialRdGp.ItemIndex:=1;
  SerialPort1.CommPort:=Com2;
  StatusBar1.Panels.Items[4].Text:='COM 2';
  Com11.Checked:=False;
  Com21.Checked:=True;
  Com31.Checked:=False;
  Com41.Checked:=False;
end;

procedure TForm1.COM31Click(Sender: TObject);
begin
  Form3.SerialRdGp.ItemIndex:=2;
  SerialPort1.CommPort:=Com3;
  StatusBar1.Panels.Items[4].Text:='COM 3';
  Com11.Checked:=False;
  Com21.Checked:=False;
  Com31.Checked:=True;
  Com41.Checked:=False;
end;

procedure TForm1.COM41Click(Sender: TObject);
begin
  Form3.SerialRdGp.ItemIndex:=3;
  SerialPort1.CommPort:=Com4;
  StatusBar1.Panels.Items[4].Text:='COM 4';
  Com11.Checked:=False;
  Com21.Checked:=False;
```

```
  Com31.Checked:=False;
  Com41.Checked:=True;
end;

procedure TForm1.N48001Click(Sender: TObject);
begin
  Form3.BaudRdGp.ItemIndex:=0;
  SerialPort1.BaudRate:=__4800;
  StatusBar1.Panels.Items[5].Text:='4800 bps';
  N48001.Checked:=True;
  N96001.Checked:=False;
  N192001.Checked:=False;
  N384001.Checked:=False;
end;

procedure TForm1.N96001Click(Sender: TObject);
begin
  Form3.BaudRdGp.ItemIndex:=1;
  SerialPort1.BaudRate:=__9600;
  StatusBar1.Panels.Items[5].Text:='9600 bps';
  N48001.Checked:=False;
  N96001.Checked:=True;
  N192001.Checked:=False;
  N384001.Checked:=False;
end;

procedure TForm1.N192001Click(Sender: TObject);
begin
  Form3.BaudRdGp.ItemIndex:=2;
  SerialPort1.BaudRate:=_19200;
  StatusBar1.Panels.Items[5].Text:='19200 bps';
  N48001.Checked:=False;
  N96001.Checked:=False;
```

```
    N192001.Checked:=True;
    N384001.Checked:=False;
end;

procedure TForm1.N384001Click(Sender: TObject);
begin
   Form3.BaudRdGp.ItemIndex:=3;
   SerialPort1.BaudRate:=__38400;
   StatusBar1.Panels.Items[5].Text:='38400 bps';
   N48001.Checked:=False;
   N96001.Checked:=False;
   N192001.Checked:=False;
   N384001.Checked:=True;
end;

procedure TForm1.N1281Click(Sender: TObject);
begin
  Form3.SizeRdGp.ItemIndex:=0;
  StatusBar1.Panels.Items[6].Text:='FFT : 128 points';
  N1281.Checked:=True;
  N2561.Checked:=False;
  N5121.Checked:=False;
  N10241.Checked:=False;
end;

procedure TForm1.N2561Click(Sender: TObject);
begin
  Form3.SizeRdGp.ItemIndex:=1;
  StatusBar1.Panels.Items[6].Text:='FFT : 256 points';
  N1281.Checked:=False;
  N2561.Checked:=True;
  N5121.Checked:=False;
  N10241.Checked:=False;
```

```
end;

procedure TForm1.N5121Click(Sender: TObject);
begin
  Form3.SizeRdGp.ItemIndex:=2;
  StatusBar1.Panels.Items[6].Text:='FFT : 512 points';
  N1281.Checked:=False;
  N2561.Checked:=False;
  N5121.Checked:=True;
  N10241.Checked:=False;
end;

procedure TForm1.N10241Click(Sender: TObject);
begin
  Form3.SizeRdGp.ItemIndex:=3;
  StatusBar1.Panels.Items[6].Text:='FFT : 1024 points';
  N1281.Checked:=False;
  N2561.Checked:=False;
  N5121.Checked:=False;
  N10241.Checked:=True;
end;


procedure TForm1.About1Click(Sender: TObject);
begin
  AboutForm.show;
  About1.Enabled := False;
end;

procedure TForm1.TimerForPlotTimer(Sender: TObject);
var IndexPlot : Integer;
    y : real;
    temp : integer;
```

```
    temptrim :string;
    Plot : string[255];
    FFTsize :integer;
    scale : real;
    SendData: String;
begin
{---checked what mode FFTsize >> Scale --}
If Form3.SizeRdGp.ItemIndex=0 then
   begin
    scale:=80/128;
    FFTsize:=128;
   end;
If Form3.SizeRdGp.ItemIndex=1 then
    begin
     scale:=80/256;
     FFTsize:=256;
    end;
If Form3.SizeRdGp.ItemIndex=2 then
    begin
     scale:=80/512;
     FFTsize:=512;
    end;
If Form3.SizeRdGp.ItemIndex=3 then
    begin
     scale:=80/1024;
     FFTsize:=1024;
    end;
{-------------------------}
 Graph1.Reset;
 Plot:=BufferMemo.Text;
{--------Plot Graph----------}
Graph1.Moveto(-40,-40);
for  IndexPlot:=1 to Length(Plot) do
```

```
 begin
    y :=Ord(Plot[IndexPlot])*scale;
    Graph1.LineTo(((IndexPlot-1)*scale)-40,y-40);
 end;
Graph1.Refresh;
{————End of Plot Graph————}
{————request to data ————}
  SendData := 'T';
  SerialPort1.WriteCommData (Pchar(Senddata),length(senddata));
end;

procedure TForm1.ExitClick(Sender: TObject);
begin
   Application.Terminate;
end;

procedure TForm1.StopClick(Sender: TObject);
begin
  { Stop Plot}
  TimerForPlot.Enabled :=False;
  { Shutdown Communication Port }
  SerialPort1.StopComm;
  StatusBar1.Panels.Items[0].Text :='Comm Closed';
  StatusBar1.Panels.Items[2].Text :='';
  StartBtn.Enabled := True;
  ContBtn.Visible := False;
  PauseBtn.Visible := True;
  PauseBtn.Enabled := False;
  StopBtn.Enabled :=False;
  SetupBtn.Enabled :=True;
  Setup1.Enabled:=True;
end;
```

```
procedure TForm1.SerialPort1ReceiveData(Buffer: Pointer;
  BufferLength: Word);
begin
{ ------Receiving data---------}
  BufferMemo.Clear;
  BufferMemo.lines.Add(strpas(buffer));
  StatusBar1.Panels.Items[2].Text :='Receiving Data';
{-------------------------}
end;
procedure TForm1.LoadDSK_Comm;
var
     LINE    : String;
     DATA    : String;
     ENTRY   : String;
     CHKSUM  : String;
     Initial : String;
     Nextline: Boolean;
     EndFile : Boolean;
     i,oldi    : integer;
     temp1,temp2 :integer;
     WordLng : integer;
begin
  Program2.Items.LoadFromFile('DSK_Comm.DSK');
  i:=1;
   SetLength(Line,4096);
   LINE := Program2.Items.Text;

  While LINE[i] <> #10  do  i:=i+1;
  i:=i+1;  {jump to new  line}
  If LINE[i] = '1' Then
    begin
      temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
      temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
```

```
    ENTRY:= chr(temp2)+chr(temp1);
  end;
 i:=i+13; {jump to new line }
{-Find wordLength-&&-CheckSum---}
 oldi:=i;
 WordIng:=0;
 EndFile:=False;
 Repeat
  case LINE[i] of
    '9':
       begin
        i:=i+5;
       end;
    'B':
       begin
        i:=i+5;
        WordLng:=WordLng+1;
       end;
    '7':
       begin
        { CheckSum:}
        temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
        temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
        {================================}
        i:=i+8;
        end;
    ':':
       begin
         EndFile:=True;
       end;
   end;
 until EndFile=True;
 i:=oldi;
```

```
{==========================================}
Label1.Caption:=IntToStr(WordIng);
{-------Send Initial Word to DSK---------}
 Initial:=#255+#255+#255+#255;
 SerialPort1.WriteCommData(PChar(Initial),Length(Initial));
    Memo1.lines.Add('Initial');
    Memo1.lines.Add(IntToStr(Ord(Initial[1])));
    Memo1.lines.Add(IntToStr(Ord(Initial[2])));
    Memo1.lines.Add(IntToStr(Ord(Initial[3])));
    Memo1.lines.Add(IntToStr(Ord(Initial[4])));
{==============================================}
{------ Send  Data Word to DSK ------------------------------}
 EndFile:=False;
 Repeat
  case LINE[i] of
    '9':
        begin
         i:=i+5;
        end;
    'B':
        begin
         temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
         temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
         DATA:= chr(temp2)+chr(temp1);
         SerialPort1.WriteCommData(PChar(DATA),Length(DATA));
            Memo1.lines.Add('DATA DSK');
            Memo1.lines.Add(IntToStr(Ord(DATA[1])));
            Memo1.lines.Add(IntToStr(Ord(DATA[2])));
         i:=i+5;
        end;
    '7':
        begin
         i:=i+8;
```

```
          end;
    ':':
          begin
            EndFile:=True;
          end;
      end;
  until EndFile=True;
{================================================}
{ Send——> Check Sum & Dummy————————————————— }
   CHKSUM :=#101+#141;
  SerialPort1.WriteCommData(PChar(CHKSUM),Length(CHKSUM));
       Memo1.lines.Add('Check Sum');
       Memo1.lines.Add(IntToStr(Ord(CHKSUM[1])));
       Memo1.lines.Add(IntToStr(Ord(CHKSUM[2])));
    DATA:= #0;
  SerialPort1.WriteCommData(PChar(DATA),Length(DATA));
      Memo1.lines.Add('Dummy');
      Memo1.lines.Add(IntToStr(Ord(DATA[1])));
{=============================================}
end;


procedure TForm1.LoadFFT;
var
     LINE    : String;
     ENTRY   : String;
     ADDR    : String;
     DATA    : String;
     CMD     : String;
     OneWord :  String;
     Nextline: Boolean;
     Send,EndFile : Boolean;
     i    : integer;
```

```
    temp1,temp2 :integer;
    ProgLegth :integer;
    addr1,addr2:integer;
begin
  Program2.Items.LoadFromFile('FFT.DSK');
  i:=1;
   SetLength(Line,8192);
   ProgLegth :=Length(Program2.Items.Text);
   LINE := Program2.Items.Text;
  While LINE[i] <> #10  do  i:=i+1;
   i:=i+1;
repeat
 repeat
   Case LINE[i]of
     '1' :
         begin
          temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
          temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
          ENTRY:= chr(temp2)+ chr(temp1);
          Send:=False;
          Nextline:=False;
          EndFile:=False;
         end;
    '7' :
          begin
          temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
          temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
          Send:=False;
          Nextline:=False;
          EndFile:=False;
        end;
    '9' :
         begin
```

```
      addr1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
      addr2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
      Send:=False;
      NextLine:=False;
      EndFile:=False;
     end;
'B' :
     begin
      CMD:=#01;
      temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
      temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
      DATA:= chr(temp2)+chr(temp1);
      Nextline:=False;
      send:=True;
      EndFile:=False;
     end;
'M' :
     begin
      CMD:=#03;
      temp1:= CharToInt(LINE[i+1])*16 + CharToInt(LINE[i+2]);
      temp2:= CharToInt(LINE[i+3])*16 + CharToInt(LINE[i+4]);
      DATA:= chr(temp2) + chr(temp1);
      Send:=True;
      Nextline:=False;
      EndFile:=False;
     end;
'F':
     begin
       Send:=False;
       NextLine:=True;
       Endfile:=False;
     end;
':':
```

```
        begin
          Send:=False;
          NextLine:=True;
          EndFile:=True;
        end;
    end;

If Send =True Then
    begin
    { Send—> CMD & ADDR & Length=1 & DATA    }
     SerialPort1.WriteCommData(PChar(CMD),Length(CMD));
       Memo1.lines.Add('CMD');
       Memo1.lines.Add(IntToStr(Ord(CMD[1])));
      ADDR:=chr(Addr2)+chr(Addr1);
      Addr2:=Addr2+1;
     SerialPort1.WriteCommData(PChar(ADDR),Length(ADDR));
       Memo1.lines.Add('Address');
       Memo1.lines.Add(IntToStr(Ord(ADDR[1])));
       Memo1.lines.Add(IntToStr(Ord(ADDR[2])));
    OneWord:=#0+#0;
    SerialPort1.WriteCommData(PChar(OneWord),Length(OneWord));
       Memo1.lines.Add('OneWord');
       Memo1.lines.Add(IntToStr(Ord(OneWord[1])));
       Memo1.lines.Add(IntToStr(Ord(OneWord[2])));
    SerialPort1.WriteCommData(PChar(DATA),Length(DATA));
       Memo1.lines.Add('DATA FFT');
       Memo1.lines.Add(IntToStr(Ord(DATA[1])));
       Memo1.lines.Add(IntToStr(Ord(DATA[2])));
    {=========================================}
    end;
     i:=i+5;
     If LINE[i]='F' Then i:=i-2;
Until NextLine= True;
```

```
Until EndFile=True;
{ Send Endding Code && to Execute Code at address ENTRY }
    DATA:=#0;
    SerialPort1.WriteCommData(PChar(DATA),Length(DATA));
      Memo1.lines.Add('SYNC');
      Memo1.lines.Add(IntToStr(Ord(DATA[1])));
    DATA:=#5;
    SerialPort1.WriteCommData(PChar(DATA),Length(DATA));
      Memo1.lines.Add('BRANCH');
      Memo1.lines.Add(IntToStr(Ord(DATA[1])));
    SerialPort1.WriteCommData(PChar(ENTRY),Length(ENTRY));
      Memo1.lines.Add('ENTRY');
      Memo1.lines.Add(IntToStr(Ord(ENTRY[1])));
      Memo1.lines.Add(IntToStr(Ord(ENTRY[2])));
    DATA:=#0;
    SerialPort1.WriteCommData(PChar(DATA),Length(DATA));
      Memo1.lines.Add('SYNC');
      Memo1.lines.Add(IntToStr(Ord(DATA[1])));
 {=========================================================}
end;

Function TForm1.CharToInt(InChar :Char) : Integer;
 begin
    If (Ord(Inchar)>= 48) and (Ord(Inchar)<=57) Then
      CharToInt:= StrToInt(Inchar)
    else; begin
      If Inchar = 'A' Then CharToInt:=10;
      If Inchar = 'B' Then CharToInt:=11;
      If Inchar = 'C' Then CharToInt:=12;
      If Inchar = 'D' Then CharToInt:=13;
      If Inchar = 'E' Then CharToInt:=14;
      If Inchar = 'F' Then CharToInt:=15;
    end;
```

```
  end;
end.
```

Appendix B

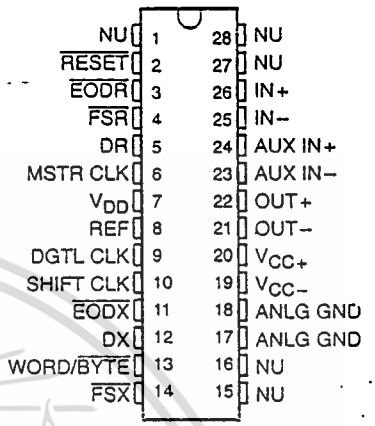
TLC32040 Analog Interface Circuit Data Sheet

Appendix B is the TLC32040 data sheet. This data sheet provides all specifications of the analog interface circuit used by the DSK starter kit.

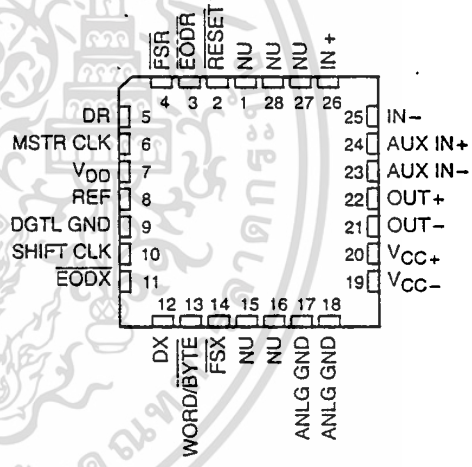
- Advanced LinCMOS™ Silicon-Gate Process Technology
- 14-Bit Dynamic Range ADC and DAC
- Variable ADC and DAC Sampling Rate Up to 19,200 Samples per Second
- Switched-Capacitor Antialiasing Input Filter and Output-Reconstruction Filter
- Serial Port for Direct Interface to TMS32011, TMS320C17, TMS32020, and TMS320C25 Digital Signal Process
- Synchronous or Asynchronous ADC and DAC Conversion Rate With Programmable Incremental ADC and DAC Conversion Timing Adjustments
- Serial Port Interface to SN74299 Serial-to-Parallel Shift Register for Parallel Interface to TMS32010, TMS320C15, or Other Digital Processors
- 600-Mil Wide N Package (C_L to C_L)

PART NUMBER	DESCRIPTION
TLC32040	Analog interface circuit with internal reference. Also a plug-in replacement for TLC32041.
TLC32041	Analog interface circuit without internal reference
TLC32042	Identical to TLC32040, but has a slightly wider bandpass filter bandwidth

N PACKAGE
(TOP VIEW)



FN PACKAGE
(TOP VIEW)



NU – Nonusable; no external connection should be made to these pins.

description

The TLC32040, TLC32041, and TLC32042 are complete analog-to-digital and digital-to-analog input/output systems, each on a single monolithic CMOS chip. This device integrates a bandpass switched-capacitor antialiasing input filter, a 14-bit-resolution A/D converter, four microprocessor-compatible serial port modes, a 14-bit-resolution D/A converter, and a low-pass switched-capacitor output-reconstruction filter. The device offers numerous combinations of master clock input frequencies and conversion/sampling rates, which can be changed via digital processor control.

Typical applications for this integrated circuit include modems (7.2-, 8-, 9.6-, 14.4-, and 19.2-kHz sampling rate), analog interface for digital signal processors (DSPs), speech recognition/storage systems, industrial process control, biomedical instrumentation, acoustical signal processing, spectral analysis, data acquisition, and instrumentation recorders. Four serial modes, which allow direct interface to the TMS32011, TMS320C17, TMS32020, and TMS320C25 digital signal processors, are provided. Also, when the transmit and receive

Advanced LinCMOS™ is a trademark of Texas Instruments Incorporated

description (continued)

sections of the analog interface circuit (AIC) are operating synchronously, it will interface to two SN74299 serial-to-parallel shift registers. These serial-to-parallel shift registers can then interface in parallel to the TMS32010, TMS320C15, other digital signal processors, or external FIFO circuitry. Output data pulses are emitted to inform the processor that data transmission is complete or to allow the DSP to differentiate between two transmitted bytes. A flexible control scheme is provided so that the functions of this integrated circuit can be selected and adjusted coincidentally with signal processing via software control.

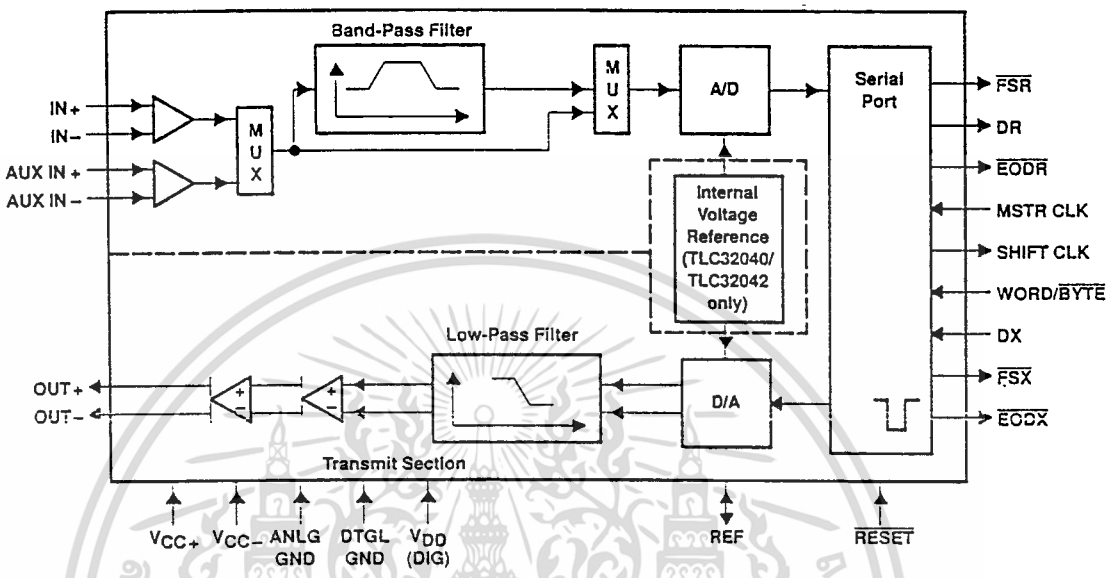
The antialiasing input filter comprises seventh-order and fourth-order CC-type (Chebyshev/elliptic transitional) low-pass and high-pass filters, respectively and a fourth-order equalizer. The input filter is implemented in switched-capacitor technology and is preceded by a continuous time filter to eliminate any possibility of aliasing caused by sampled data filtering. When no filtering is desired, the entire composite filter can be switched out of the signal path. A selectable, auxiliary, differential analog input is provided for applications where more than one analog input is required.

The A/D and D/A converters each have 14 bits of resolution. The A/D and D/A architectures ensure no missing codes and monotonic operation. An internal voltage reference is provided on the TLC32040 and TLC32042 to ease the design task and to provide complete control over the performance of this integrated circuit. The internal voltage reference is brought out to a pin and is available to the designer. Separate analog and digital voltage supplies and grounds are provided to minimize noise and ensure a wide dynamic range. Also, the analog circuit path contains only differential circuitry to keep noise to an absolute minimum. The only exception is the DAC sample and hold, which utilizes pseudo-differential circuitry.

The output-reconstruction filter is a seventh-order CC-type (Chebyshev/elliptic transitional low-pass filter followed by a fourth-order equalizer) and is implemented in switched-capacitor technology. This filter is followed by a continuous-time filter to eliminate images of the digitally encoded signal.

The TLC32040C, TLC32041C, and TLC32042C are characterized for operation from 0°C to 70°C, and the TLC32040I, TLC32041I and TLC32042I are characterized for operation from –40°C to 85°C.

functional block diagram



analog input

Two sets of analog inputs are provided. Normally, the IN+ and IN– input set is used; however, the auxiliary input set, AUX IN+ and AUX IN–, can be used if a second input is required. Each input set can be operated in either differential or single-ended modes, since sufficient common-mode range and rejection are provided. The gain for the IN+, IN–, AUX IN+, and AUX IN– inputs can be programmed to be either 1, 2, or 4 (see Table 2). Either input circuit can be selected via software control. It is important to note that a wide dynamic range is assured by the differential internal analog architecture and by the separate analog and digital voltage supplies and grounds.

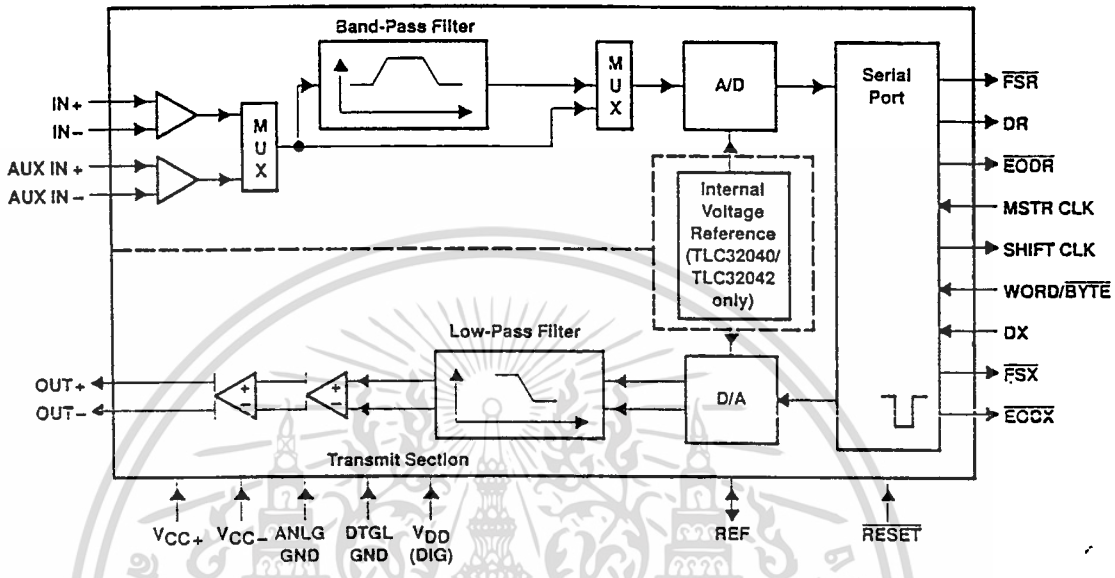
A/D bandpass filter, A/D bandpass filter clocking, and A/D conversion timing

The A/D bandpass filter can be selected or bypassed via software control. The frequency response of this filter is presented in the following pages. This response results when the switched-capacitor filter clock frequency is 288 kHz. Several possible options can be used to attain a 288-kHz switched-capacitor filter clock. When the filter clock frequency is not 288 kHz, the filter transfer function is frequency scaled by the ratio of the actual clock frequency to 288 kHz. The low-frequency roll-off of the high-pass section is 300 Hz. However, the high-pass section low-frequency roll-off is less steep for the TLC32042 than for the TLC32040 and TLC32041.

The internal timing configuration and AIC DX data word format sections of this data sheet indicate the many options for attaining a 288-kHz bandpass switched-capacitor filter clock. These sections indicate that the RX Counter A can be programmed to give a 288-kHz bandpass switched-capacitor filter clock for several master clock input frequencies.

The A/D conversion rate is then attained by frequency dividing the 288-kHz bandpass switched-capacitor filter clock with the RX Counter B. Thus, unwanted aliasing is prevented because the A/D conversion rate is an integral submultiple of the bandpass switched-capacitor filter sampling rate, and the two rates are synchronously locked.

functional block diagram



analog input

Two sets of analog inputs are provided. Normally, the IN+ and IN– input set is used; however, the auxiliary input set, AUX IN+ and AUX IN–, can be used if a second input is required. Each input set can be operated in either differential or single-ended modes, since sufficient common-mode range and rejection are provided. The gain for the IN+, IN–, AUX IN+, and AUX IN– inputs can be programmed to be either 1, 2, or 4 (see Table 2). Either input circuit can be selected via software control. It is important to note that a wide dynamic range is assured by the differential internal analog architecture and by the separate analog and digital voltage supplies and grounds.

A/D bandpass filter, A/D bandpass filter clocking, and A/D conversion timing

The A/D bandpass filter can be selected or bypassed via software control. The frequency response of this filter is presented in the following pages. This response results when the switched-capacitor filter clock frequency is 288 kHz. Several possible options can be used to attain a 288-kHz switched-capacitor filter clock. When the filter clock frequency is not 288 kHz, the filter transfer function is frequency scaled by the ratio of the actual clock frequency to 288 kHz. The low-frequency roll-off of the high-pass section is 300 Hz. However, the high-pass section low-frequency roll-off is less steep for the TLC32042 than for the TLC32040 and TLC32041.

The internal timing configuration and AIC DX data word format sections of this data sheet indicate the many options for attaining a 288-kHz bandpass switched-capacitor filter clock. These sections indicate that the RX Counter A can be programmed to give a 288-kHz bandpass switched-capacitor filter clock for several master clock input frequencies.

The A/D conversion rate is then attained by frequency dividing the 288-kHz bandpass switched-capacitor filter clock with the RX Counter B. Thus, unwanted aliasing is prevented because the A/D conversion rate is an integral submultiple of the bandpass switched-capacitor filter sampling rate, and the two rates are synchronously locked.

A/D converter performance specifications

Fundamental performance specifications for the A/D converter circuitry are presented in the A/D converter operating characteristics section of this data sheet. The realization of the A/D converter circuitry with switched-capacitor techniques provides an inherent sample-and-hold.

analog output

The analog output circuitry is an analog output power amplifier. Both noninverting and inverting amplifier outputs are brought out of this integrated circuit. This amplifier can drive transformer hybrids or low-impedance loads directly in either a differential or single-ended configuration.

D/A low-pass filter, D/A low-pass filter clocking, and D/A conversion timing

The frequency response of this filter is presented in the following pages. This response results when the low-pass switched-capacitor filter clock frequency is 288 kHz. Like the A/D filter, the transfer function of this filter is frequency scaled when the clock frequency is not 288 kHz. A continuous-time filter is provided on the output on the output of the D/A low-pass filter to greatly attenuate any switched-capacitor clock feedthrough.

The D/A conversion rate is then attained by frequency dividing the 288-kHz switched-capacitor filter clock with TX Counter B. Thus, unwanted aliasing is prevented because the D/A conversion rate is an integral submultiple of the switched-capacitor low-pass filter sampling rate, and the two rates are synchronously locked.

PRINCIPILES OF OPERATION

asynchronous versus synchronous operation

If the transmit section of the AIC (low-pass filter and DAC) and receive section (bandpass filter and ADC) are operated asynchronously, the low-pass and band-pass filter clocks are independently generated from the master clock signal. Also, the D/A and A/D conversion rates are independently determined. If the transmit and receive sections are operated synchronously, the low-pass filter clock drives both low-pass and bandpass filters. In synchronous operation, the A/D conversion timing is derived from, and is equal to, the D/A conversion timing. (See description of the WORD/$\overline{\text{BYTE}}$ pin in the Terminal Functions table.)

D/A converter performance specifications

Fundamental performance specifications for the D/A converter circuitry are presented in the D/A converter operating characteristics section of the data sheet. The D/A converter has a sample-and-hold that is realized with a switched-capacitor ladder.

system frequency response correction

The sin x/x correction circuitry is performed in the digital processor software. The system frequency response can be corrected via DSP software to ±0.1-dB accuracy to band edge of 3000 Hz for all sampling rates. This correction is accomplished with a first-order digital correction filter, which requires only seven TMS320 instruction cycles. With a 200-ns instruction cycle, seven instructions represent an overhead factor of only 1.1% and 1.3% for sampling rates of 8 and 9.6 kHz, respectively (see the sin x/x correction section for more details).

serial port

The serial port has four possible modes that are described in detail in the Terminal Functions table. These modes are briefly described below and in the description for pin 13, WORD/$\overline{\text{BYTE}}$.

1. The transmit and receive sections are operated asynchronously, and the serial port interfaces directly with the TMS32011 and TMS320C17.
2. The transmit and receive sections are operated asynchronously, and the serial port interfaces directly with the TMS32020 and the TMS320C25.
3. The transmit and receive sections are operated synchronously, and the serial port interfaces directly with the TMS32011 and TMS320C17.
4. The transmit and receive sections are operated synchronously, and the serial port interfaces directly with the TMS32020, TMS320C25, or two SN74299 serial-to-parallel shift registers, which can then interface in parallel to the TMS320C10, TMS32015, to any other digital signal processor, or to external FIFO circuitry.

operation of TLC32040 or TLC32042 with internal voltage reference

The internal reference of the TLC32040 and TLC32042 eliminates the need for an external voltage reference and provides overall circuit cost reduction. Thus, the internal reference eases the design task and provides complete control over the performance of this integrated circuit. The internal reference is brought out to a pin and is available to the designer. To keep the amount of noise on the reference signal to a minimum, an external capacitor may be connected between REF and ANLG GND.

PRINCIPLES OF OPERATION

operation of TLC32040, TLC32041, or TLC32042 with external voltage reference

The REF pin may be driven from an external reference circuit if so desired. This external circuit must be capable of supplying 250 μA and must be adequately protected from noise such as crosstalk from the analog input.

reset

A reset function is provided to initiate serial communications between the AIC and DSP and allow fast, cost-effective testing during manufacturing. The reset functional will initialize all AIC registers, including the control register. After a negative-going pulse on the $\overline{\text{RESET}}$ pin, the AIC will be initialized. This initialization allows normal serial port communications activity to occur between AIC and DSP (see AIC DX data word format section).

loopback

This feature allows the user to test the circuit remotely. In loopback, the OUT+ and OUT– pins are internally connected to the IN+ and the IN– pins. Thus, the DAC bits (d15 to d2), which are transmitted to the DX pin, can be compared with the ADC bits (d15 to d2), which are received from the DR pin. An ideal comparison would be that the bits on the DR pin equal the bits on the DX pin. However, in practice there will be some difference in these bits due to the ADC and DAC output offsets.

In loopback, if the IN+ and the IN– pins are enabled, the external signals on the IN+ and the IN– pins are ignored. If the AUX IN+ and AUX IN– pins are enabled, the external signals on these pins are added to the OUT+ and OUT– signals in loopback operation.

The loopback feature is implemented with digital signal processor control by transmitting the appropriate serial port bit to the control register (see AIC DX data word format section).

Terminal Functions

PIN NAME	PIN NO.	I/O	DESCRIPTION
ANLG GND	17,18		Analog ground return for all internal analog circuits. Not internally connected to DGTL GND.
AUX IN+	24	I	Noninverting auxiliary analog input state. This input can be switched into the bandpass filter and A/D converter path via software control. If the appropriate bit in the control register is a 1, the auxiliary inputs will replace the IN+ and IN– inputs. If the bit is a 0, the IN+ and IN– inputs will be used (see the AIC DX data word format section).
AUX IN–	23	I	Inverting auxiliary analog input (see the above AUX IN+ pin description)
DGTL GND	9		Digital ground for all internal logic circuits. Not internally connected to ANLG GND.
DR	5	O	This pin is used to transmit the ADC output bits from the AIC to the TMS320 serial port. This transmission of bits from the AIC to the TMS320 serial port is synchronized with the SHIFT CLK signal.
DX	12	I	This pin is used to receive the DAC input bits and timing and control information from the TMS320. This serial transmission from the TMS320 serial port to the AIC is synchronized with the SHIFT CLK signal.
$\overline{\text{EODR}}$	3	O	End of data receive. See the WORD/$\overline{\text{BYTE}}$ pin description and the Serial Port Timing diagrams. During the word-mode timing, this signal is a low-going pulse that occurs immediately after the 16 bits of A/D information have been transmitted from the AIC to the TMS320 serial port. This signal can be used to interrupt a microprocessor upon completion of serial communications. Also, this signal can be used to strobe and enable external serial-to-parallel shift registers, latches, or external FIFO RAM, and to facilitate parallel data bus communications between the AIC and the serial-to-parallel shift registers. During the byte-mode timing, this signal goes low after the first byte has been transmitted from the AIC to the TMS320 serial port and is kept low until the second byte has been transmitted. The TMS32011 or TMS320C17 can use this low-going signal to differentiate between the two bytes as to which is first and which is second. $\overline{\text{EODR}}$ does not occur after secondary communication.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

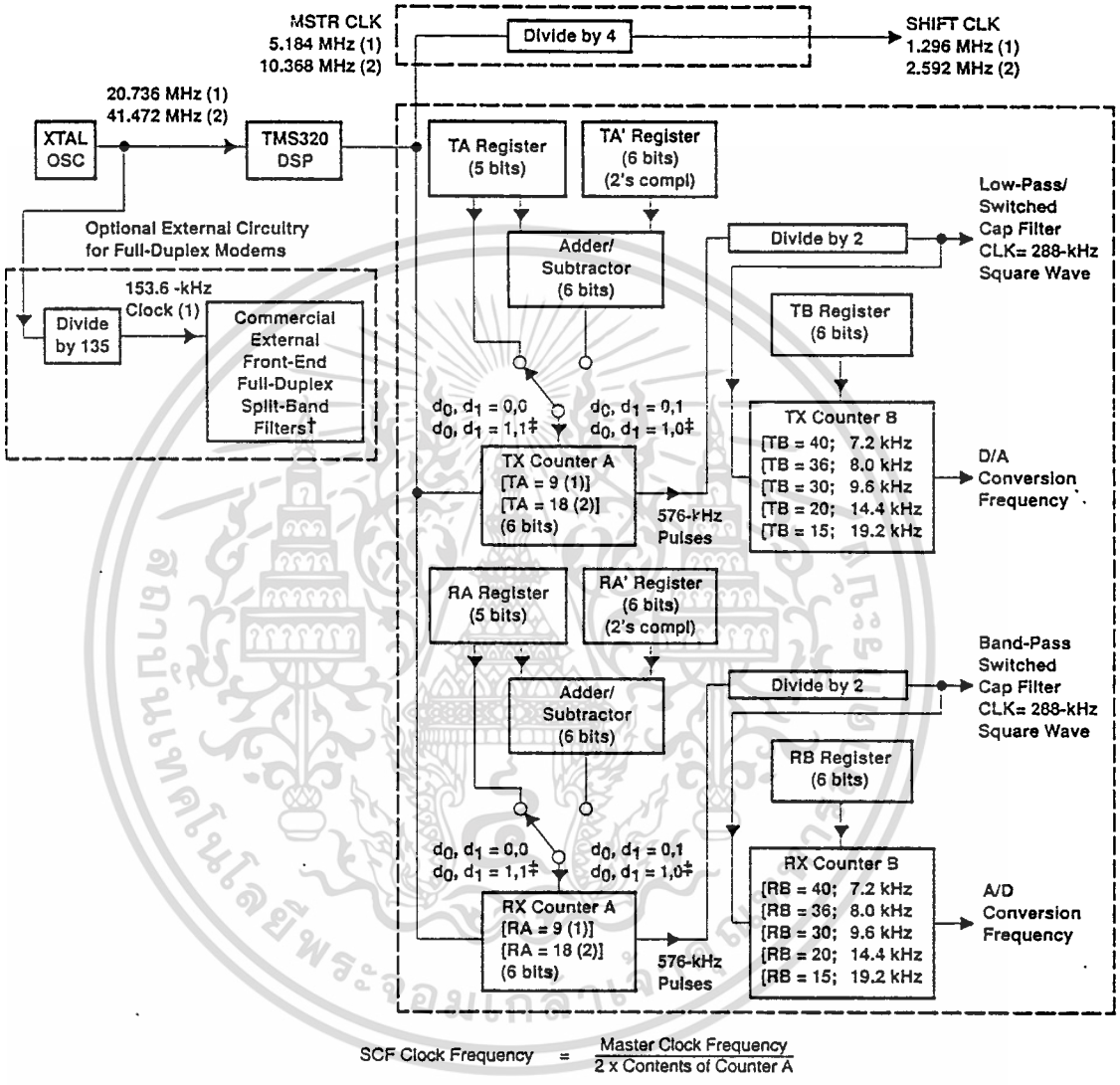
Terminal Functions (continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
$\overline{EODX}$	11	O	End of data transmit. See the WORD/$\overline{BYTE}$ pin description and the Serial Port Timing diagram. During the word-mode timing, this signal is a low-going pulse that occurs immediately after the 16 bits of D/A converter and control or register information have been transmitted from the TMS320 serial port to the AIC. This signal can be used to interrupt a microprocessor upon the completion of serial communications. Also, this signal can be used to strobe and enable external serial-to-parallel shift registers, latches, or an external FIFO RAM, and to facilitate parallel data-bus communications between the AIC and the serial-to-parallel shift registers. During the byte-mode timing, this signal goes low after the first byte has been transmitted from the TMS320 serial port to the AIC and is kept low until the second byte has been transmitted. The TMS32011 or TMS320C17 can use this low-going signal to differentiate between the two bytes as to which is first and which is second.
$\overline{FSR}$	4	O	Frame sync receive. In the serial transmission modes, which are described in the WORD/$\overline{BYTE}$ pin description, the $\overline{FSR}$ pin is held low during bit transmission. When the $\overline{FSR}$ pin goes low, the TMS320 serial port will begin receiving bits from the AIC via the DR pin of the AIC. The most significant DR bit will be present on the DR pin before $\overline{FSR}$ goes low. (See Serial Port Timing and Internal Timing Configuration diagrams.) $\overline{FSR}$ does not occur after secondary communication.
FSX	14	O	Frame Sync Transmit. When this pin goes low, the TMS320 serial port will begin transmitting bits to the AIC via the DX pin of the AIC. In all serial transmission modes, which are described in the WORD/$\overline{BYTE}$ pin description, the $\overline{FSX}$ pin is held low during bit transmission (see the Serial Port Timing and Internal Timing Configuration diagrams).
IN+	26	I	Noninverting input to analog input amplifier stage
IN−	25	I	Inverting input to analog input amplifier stage
MSTR CLK	6	I	The master clock signal is used to derive all the key logic signals of the AIC, such as the shift clock, the switched-capacitor filter clocks, and the A/D and D/A timing signals. The Internal Timing Configuration diagram shows how these key signals are derived. The frequencies of these key signals are synchronous submultiples of the Master Clock frequency to eliminate unwanted aliasing when the sampled analog signals are transferred between the switched-capacitor filters and the A/D and D/A converters (see the Internal Timing Configuration).
OUT+	22	O	Noninverting output of analog output power amplifier. Can drive transformer hybrids or high-impedance loads directly in either a differential or a single-ended configuration.
OUT−	21	O	Inverting output of analog output power amplifier. Functionally identical with and complementary to OUT+.
REF	8	I/O	For the TLC32040 and TLC32042, the internal voltage reference is brought out on this pin. For the TLC32040, TLC32041, and TLC32042, an external voltage reference can be applied to this pin.
$\overline{RESET}$	2	I	A reset function is provided to initialize the TA, TA', TB, RA, RA', RB, and control registers. This reset function initiates serial communications between the AIC and DSP. The reset function will initialize all AIC registers including the control register. After a negative-going pulse on the $\overline{RESET}$ pin, the AIC registers will be initialized to provide an 8-kHz data conversion rate for a 5.184-MHz master clock input signal. The conversion rate adjust registers, TA' and RA', will be reset to 1. The control register bits will be reset as follows (see AIC DX data word format section): d7 = 1, d6 = 1, d5 = 1, d4 = 0, d3 = 0, d2 = 1 This initialization allows normal serial-port communication to occur between AIC and DSP.
SHIFT CLK	10	O	The shift clock signal is obtained by dividing the master clock signal frequency by four. This signal is used to clock the serial data transfers of the AIC, described in the WORD/$\overline{BYTE}$ pin description below (see the Serial Port Timing and Internal Timing Configuration diagrams).
V_{DD}	7		Digital supply voltage, 5 V ±5%
V_{CC+}	20		Positive analog supply voltage, 5 V ±5%
V_{CC-}	19		Negative analog supply voltage, −5 V ±5%

Terminal Functions (continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
WORD/$\overline{\text{BYTE}}$	13	I	This pin, in conjunction with a bit in the control register, is used to establish one of four serial modes. These four serial modes are described below. *AIC transmit and receive sections are operated asynchronously.* The following description applies when the AIC is configured to have asynchronous transmit and receive sections. If the appropriate data bit in the control register is a 0 (see the AIC DX data word format section), the transmit and receive sections will be asynchronous. L Serial port directly interfaces with the serial port of the TMS32011 or TMS320C17 and communicates in two 8-bit bytes. The operation sequence is as follows (see Serial Port Timing diagrams). 1. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought low. 2. One 8-bit byte is transmitted or one 8-bit byte is received. 3. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin is brought low. 4. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin emits a positive frame-sync pulse that is four shift clock cycles wide. 5. One 8-bit byte is transmitted or one 8-bit byte is received. 6. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin is brought high. 7. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought high. H Serial port directly interfaces with the serial port of the TMS32020, TMS320C25, or TMS320C30 and communicates in one 16-bit word. The operation sequence is as follows (see Serial Port Timing diagrams): 1. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought low. 2. One 16-bit word is transmitted or one 16-bit word is received. 3. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought high. 4. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin emits a low-going pulse. *AIC transmit and receive sections are operated synchronously.* If the appropriate data bit in the control register is a 1, the transmit and receive sections will be configured to be synchronous. In this case, the bandpass switched-capacitor filter and the A/D conversion timing will be derived from the TX Counter A, TX Counter B, and TA, TA', and TB registers, rather than the RX Counter A, RX Counter B, and RA, RA', and RB registers. In this case, the AIC $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ timing will be identical during primary data communication; however, $\overline{\text{FSR}}$ will not be asserted during secondary data communication since there is no new A/D conversion result. The synchronous operation sequences are as follows (see Serial Port Timing diagrams). L Serial port directly interfaces with the serial port of the TMS32011 or TMS320C17 and communicates in two 8-bit bytes. The operation sequence is as follows (see Serial Port Timing diagrams): 1. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought low. 2. One 8-bit byte is transmitted and one 8-bit byte is received. 3. The $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ pin are brought low. 4. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins emit positive frame-sync pulses that are four Shift Clock cycles wide 5. One 8-bit byte is transmitted and one 8-bit byte is received. 6. The $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ pins are brought high. 7. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought high. H Serial port directly interfaces with the serial port of the TMS32020, TMS320C25, or TMS320C30 and communicates in one 16-bit word. The operation sequence is as follows (see Serial Port Timing diagrams): 1. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought low. 2. One 16-bit word is transmitted and one 16-bit word is received. 3. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought high. 4. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pins emit low-going pulses. Since the transmit and receive sections of the AIC are now synchronous, the AIC serial port with additional NOR and AND gates will interface to two SN74299 serial-to-parallel shift registers. Interfacing the AIC to the SN74299 shift register allows the AIC to interface to an external FIFO RAM and facilitates parallel data bus communications between the AIC and the digital signal processor. The operation sequence is the same as the above sequence (see Serial Port Timing diagrams).

INTERNAL TIMING CONFIGURATION



$$\text{SCF Clock Frequency} = \frac{\text{Master Clock Frequency}}{2 \times \text{Contents of Counter A}}$$

NOTE: Frequency 1,20.736 MHz is used to show how 153.6 kHz (for commercially available modem split-band filter clock), popular speech and modem sampling signal frequencies, and an internal 288-kHz switched-capacitor filter clock can be derived synchronously and as submultiples of the crystal oscillator frequency. Since these derived frequencies sre synchronous submultiples of the crystal frequency, aliasing does not occur as the sampled analog signal passes between the analog converter and switched-capacitor filter stages. Frequency 2,41.472 MHz is used to show that the AIC can work with high-frequency signals, which are used by high-speed digital signal processors.

† Split-band filtering can alternatively be performed after the analog input function via software in the TMS320.

‡ These control bits are described in the AIC DX data word format section.

explanation of internal timing configuration

All of the internal timing of the AIC is derived from the high-frequency clock signal that drives the MSTR CLK input pin. The SHIFT CLK signal, which strobes the serial port data between the AIC and DSP, is derived by dividing the master clock input signal frequency by four.

$$\text{SCF Clock Frequency} = \frac{\text{Master Clock Frequency}}{2 \times \text{Contents of Counter A}}$$

$$\text{Conversion Frequency} = \frac{\text{SCF Clock Frequency}}{\text{Contents of Counter B}}$$

$$\text{Shift Clock Frequency} = \frac{\text{Master Clock Frequency}}{4}$$

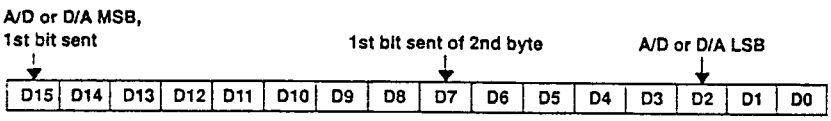
TX Counter A and TX Counter B, which are driven by the MSTR CLK signal, determine the D/A conversion timing. Similarly, RX Counter A and RX Counter B determine the A/D conversion timing. In order for the switched-capacitor low-pass and band pass filters to meet their transfer function specifications, the frequency of the clock inputs of the switched-capacitor filters must be 288 kHz. If the frequencies of the clock inputs are not 288 kHz, the filter transfer function frequencies are scaled by the ratios of the clock frequencies to 288 kHz. Thus, to obtain the specified filter responses, the combination of master clock frequency and TX Counter A and RX Counter A values must yield 288-kHz switched-capacitor clock signals. These 288-kHz clock signals can then be divided by the TX Counter B and RX Counter B to establish the D/A and A/D conversion timings.

TX Counter A and TX Counter B are reloaded every D/A conversion period, while RX Counter A and RX Counter B are reloaded every A/D conversion period. The TX Counter B and RX Counter B are loaded with the values in the TB and RB Registers, respectively. Via software control, the TX Counter A can be loaded with either the TA Register, the TA Register less the TA' Register, or the TA Register plus the TA' Register. By selecting the TA Register less the TA' Register option, the upcoming conversion timing will occur earlier by an amount of time that equals TA' times the signal period of the Master Clock. By selecting the TA Register plus the TA' Register option, the upcoming conversion timing will occur later by an amount of time that equals TA' times the signal period of the master clock. Thus, the D/A conversion timing can be advanced or retarded. An identical ability to alter the A/D conversion timing is provided. In this case, however, the RX Counter A can be programmed via software control with the RA Register, the RA Register less the RA' Register, or the RA Register plus the RA' Register.

The ability to advance or retard conversion timing is particularly useful for modem applications. This feature allows controlled changes in the A/D and D/A conversion timing. This feature can be used to enhance signal-to-noise performance, to perform frequency-tracking functions, and to generate nonstandard modem frequencies.

If the transmit and receive sections are configured to be synchronous (see WORD/$\overline{\text{BYTE}}$ pin description), then both the low-pass and bandpass switched-capacitor filter clocks are derived from TX Counter A. Also, both the D/A and A/D conversion timing are derived from the TX Counter A and TX Counter B. When the transmit and receive sections are configured to be synchronous, the RX Counter A, RX Counter B, RA Register, RA' Register, and RB Registers are not used.

AIC DR or DX word bit pattern



AIC DX data word format section

d15	d14	d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0	COMMENTS
primary DX serial communication protocol																
←d15 (MSB) through d2 go to the D/A converter register													→	0	0	The TX and RX Counter As are loaded with the TA and RA register values. The TX and RX Counter Bs are loaded with TB and RB register values.
←d15 (MSB) through d2 go to the D/A converter register													→	0	1	The TX and RX Counter A's are loaded with the TA + TA' and RA + RA' register values. The TX and RX Counter Bs are loaded with TB and RB register values. NOTE: d1 = 0, d0 = 1 will cause the next D/A and A/D conversion periods to be changed by the addition of TA' and RA' master clock cycles, in which TA' and R/A' can be positive or negative or zero. Please refer to Table 1.
←d15 (MSB) through d2 go to the D/A converter register													→	1	0	The TX and RX Counter As are loaded with the TA – TA' and RA – RA' register values. The TX and RX Counter Bs are loaded with TB and RB register values. NOTE: d1 = 1, d0 = 0 will cause the next D/A and A/D conversion periods to be changed by the subtraction of TA' and RA' master clock cycles, in which TA' and R/A' can be positive or negative or zero. Please refer to Table 1.
←d15 (MSB) through d2 go to the D/A converter register													→	1	1	The TX and RX Counter As are loaded with the TA and RA register values. The TX and RX Counter Bs are loaded with the TB and RB register values. After a delay of four shift clock cycles, a secondary transmission will immediately follow to program the AIC to operate in the desired configuration.

NOTE: Setting the two least significant bits to 1 in the normal transmission of DAC information (primary communications) to the AIC will initiate secondary communications upon completion of the primary communications.

Upon completion of the primary communication, $\overline{FSX}$ will remain high for four SHIFT CLK cycles and will then go low and initiate the secondary communication. The timing specifications for the primary and secondary communications are identical. In this manner, the secondary communication, if initiated, is interleaved between successive primary communications. This interleaving prevents the secondary communication from interfering with the primary communications and DAC timing, thus preventing the AIC from skipping a DAC output. It is important to note that in the synchronous mode, $\overline{FSR}$ will not be asserted during secondary communications.

secondary DX serial communication protocol

x x I ← to TA register → I x x I ← to RA register → I 0 0	d13 and d6 are MSBs (unsigned binary)
x I ← to TA' register → I x I ← to RA' register → I 0 1	d14 and d7 are 2's complement sign bits
x I ← to TB register → I x I ← to RB register → I 1 0	d14 and d7 are MSBs (unsigned binary)
x x x x x x x x d7 d6 d5 d4 d3 d2 1 1 Control Register	d2 = 0/1 deletes/inserts the bandpass filter d3 = 0/1 disables/enables the loopback function d4 = 0/1 disables/enables the AUX IN+ and AUX IN– pins d5 = 0/1 asynchronous/synchronous transmit receive sections d6 = 0/1 gain control bits (see gain control section) d7 = 0/1 gain control bits (see gain control section)

reset function

A reset function is provided to initiate serial communications between the AIC and DSP. The reset function will initialize all AIC registers, including the control register. After power has been applied to the AIC, a negative-going pulse on the $\overline{\text{RESET}}$ pin will initialize the AIC registers to provide an 8-kHz A/D and D/A conversion rate for a 5.184-MHz master clock input signal. The AIC, except the control register, will be initialized as follows (see AIC DX data word format section):

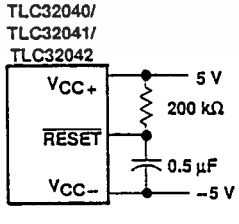
REGISTER	INITIALIZED REGISTER VALUE (HEX)
TA	9
TA'	1
TB	24
RA	9
RA'	1
RB	24

The control register bits will be reset as follows (see AIC DX data word format section):

d7 = 1, d6 = 1, d5 = 1, d4 = 0, d3 = 0, d2 = 1

This initialization allows normal serial port communications to occur between AIC and DSP. If the transmit and receive sections are configured to operate synchronously and the user wishes to program different conversion rates, only the TA, TA', and TB register need to be programmed, since both transmit and receive timing are synchronously derived from these registers (see the pin descriptions and AIC DX word format sections).

The circuit shown below will provide a reset on power up when power is applied in the sequence given under power-up sequence. The circuit depends on the power supplies reaching their recommended values a minimum of 800 ns before the capacitor charges to 0.8 V above DGTL GND.



power-up sequence

To ensure proper operation of the AIC, and as a safeguard against latch-up, it is recommended that a Schottky diode with a forward voltage less than or equal to 0.4 V be connected from V_{CC-} to ANLG GND (see Figure 17). In the absence of such a diode, power should be applied in the following sequence: ANLG GND and DGTL GND, V_{CC-}, then V_{CC+} and V_{DD}. Also, no input signal should be applied until after power up.

AIC responses to improper conditions

The AIC has provisions for responding to improper conditions. These improper conditions and the response of the AIC to these conditions are presented in Table 1 below.

AIC register constraints

The following constraints are placed on the contents of the AIC registers:

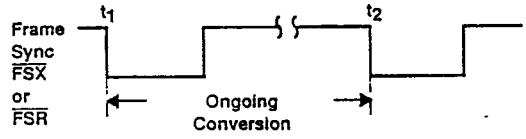
1. TA register must be ≥ 4 in word mode (WORD/$\overline{\text{BYTE}}$ = high).
2. TA register must be ≥ 5 in byte mode (WORD/$\overline{\text{BYTE}}$ = low).
3. TA' register can be either positive, negative, or zero.
4. RA register must be ≥ 4 in word mode (WORD/$\overline{\text{BYTE}}$ = high).
5. RA register must be ≥ 5 in byte mode (WORD/$\overline{\text{BYTE}}$ = low).
6. RA' register can be either positive, negative, or zero.
7. (TA register ± TA' register) must be > 1.
8. (RA register ± RA' register) must be > 1.
9. TB register must be > 1.

Table 1. AIC Responses To Improper Conditions

IMPROPER CONDITIONS	AIC RESPONSE
TA register + TA' register = 0 or 1 TA register – TA' register = 0 or 1	Reprogram TX Counter A with TA register value
TA register + TA' register < 0	MODULO 64 arithmetic is used to ensure that a positive value is loaded into the TX Counter A, i.e., TA register + TA' register + 40 HEX is loaded into TX Counter A.
RA register + RA' register = 0 or 1 RA register – RA' register = 0 or 1	Reprogram RX Counter A with RA register value
RA register + RA' register = 0 or 1	MODULO 64 arithmetic is used to ensure that a positive value is loaded into RX Counter A, i.e., RA register + RA' register + 40 HEX is loaded into RX Counter A.
TA register = 0 or 1 RA register = 0 or 1	The AIC is shut down.
TA register < 4 in word mode TA register < 5 in byte mode RA register < 4 in word mode RA register < 5 in byte mode	The AIC serial port no longer operates.
TB register = 0 or 1	Reprogram TB register with 24 HEX
RB register = 0 or 1	Reprogram RB register with 24 HEX
AIC and DSP cannot communicate	Hold last DAC output

improper operation due to conversion times being too close together

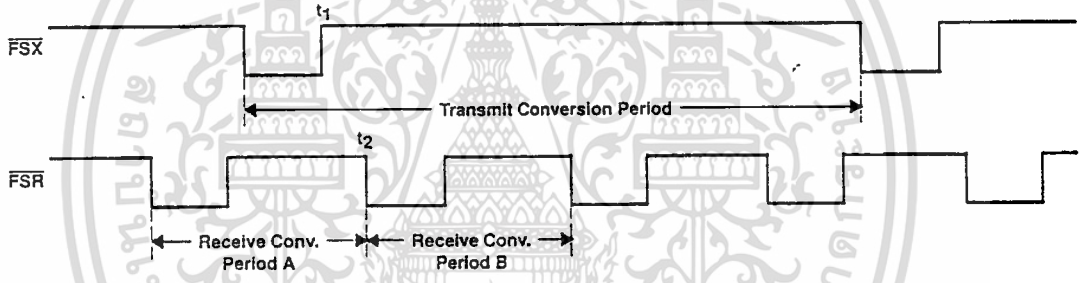
If the difference between two successive D/A conversion frame syncs is less than 1/19.2 kHz, the AIC operates improperly. In this situation, the second D/A conversion frame sync occurs too quickly and there is not enough time for the ongoing conversion to be completed. This situation can occur if the A and B registers are improperly programmed or if the A + A' register or A_X– A' register result is too small. When incrementally adjusting the conversion period via the A + A' register options, the designer should be very careful not to violate this requirement (see following diagram).



$t_2 - t_1 \geq 1/19.2$ kHz

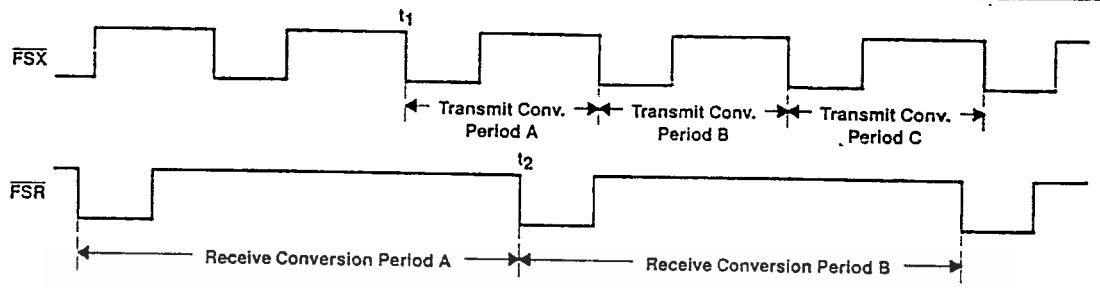
asynchronous operation — more than one receive frame sync occurring between two transmit frame syncs

When incrementally adjusting the conversion period via the A + A' or A – A' register options, a specific protocol is followed. The command to use the incremental conversion period adjust option is sent to the AIC during a $\overline{FSX}$ frame sync. The ongoing conversion period is then adjusted. However, either receive conversion period A or B may be adjusted. For both transmit and receive conversion periods, the incremental conversion period adjustment is performed near the end of the conversion period. Therefore, if there is sufficient time between t1 and t2, the receive conversion period adjustment will be performed during receive conversion period A. Otherwise, the adjustment will be performed during receive conversion period B. The adjustment command only adjusts one transmit conversion period and one receive conversion period. To adjust another pair of transmit and receive conversion periods, another command must be issued during a subsequent $\overline{FSX}$ frame (see figure below).



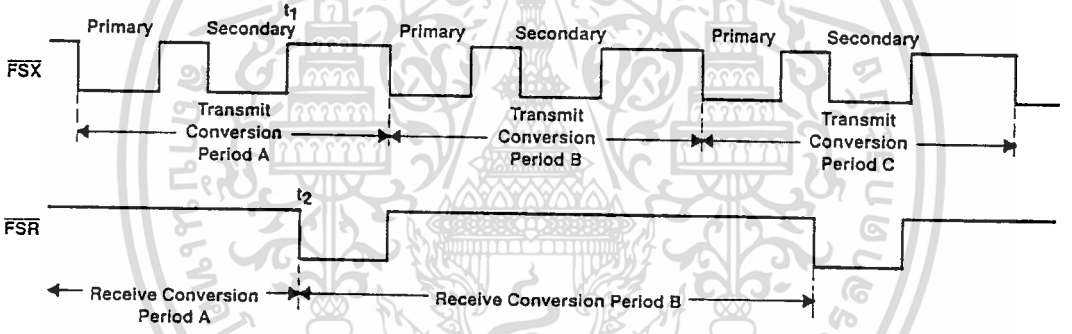
asynchronous operation — more than one receive frame sync occurring between two receive frame syncs

When incrementally adjusting the conversion period via the A + A' or A – A' register options, a specific protocol is followed. For both transmit and receive conversion periods, the incremental conversion period adjustment is performed near the end of the conversion period. The command to use the incremental conversion period adjust options is sent to the AIC during a $\overline{FSX}$ frame sync. The ongoing transmit conversion period is then adjusted. However, three possibilities exist for the receive conversion period adjustment in the diagram as shown in the following figure. If the adjustment command is issued during transmit conversion period A, receive conversion period A will be adjusted if there is sufficient time between t1 and t2. Or, if there is not sufficient time between t1 and t2, receive conversion period B will be adjusted. Or, the receive portion of an adjustment command may be ignored if the adjustment command is sent during a receive conversion period, which is already being or will be adjusted due to a prior adjustment command. For example, if adjustment commands are issued during transmit conversion periods A, B, and C, the first two commands may cause receive conversion periods A and B to be adjusted, while the third receive adjustment command is ignored. The third adjustment command is ignored since it was issued during receive conversion period B, which already will be adjusted via the transmit conversion period B adjustment command.



asynchronous operation — more than one set of primary and secondary DX serial communication occurring between two receive frame sync (see AIC DX data word format section)

The TA, TA', TB, and control register information that is transmitted in the secondary communications is always accepted and is applied during the ongoing transmit conversion period. If there is sufficient time between t1 and t2, the TA, RA', and RB register information, which is sent during transmit conversion period A, will be applied to receive conversion period A. Otherwise, this information will be applied during receive conversion period B. If RA, RA', and RB register information has already been received and is being applied during an ongoing conversion period, any subsequent RA, RA', or RB information that is received during this receive conversion period will be disregarded (see diagram below).



absolute maximum ratings over operating free-air temperature (unless otherwise noted)

Supply voltage range, V_{CC-} (see Note 1)	−0.3 V to 15 V
Supply voltage range, V_{DD}	−0.3 V to 15 V
Output voltage range, V_O	−0.3 V to 15 V
Input voltage range, V_I	−0.3 V to 15 V
Digital ground voltage range	−0.3 V to 15 V
Operating free-air temperature range: TLC32040C, TLC32041C, TLC32042C	0°C to 70°C
TLC32040I, TLC32041I, TLC32042I	−40°C to 85°C
Storage temperature range	−40°C to 125°C
Case temperature for 10 seconds: FN package	260°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds: N package	260°C

NOTE 1: Voltage values for maximum ratings are with respect to V_{CC-}

recommended operating conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC+} (see Note 2)		4.75	5	5.25	V
Supply voltage, V_{CC-} (see Note 2)		−4.75	−5	−5.25	V
Digital supply voltage, V_{DD} (see Note 2)		4.75	5	5.25	V
Digital ground voltage with respect to ANLG GND, DGTL GND			0		V
Reference input voltage, $V_{ref(ext)}$ (see Note 2)		2		4	V
High-level input voltage, V_{IH}		2		V_{DD}+0.3	V
Low-level input voltage, V_{IL} (see Note 3)		−0.3		0.8	V
Load resistance at OUT+ and/or OUT−, R_L		300			Ω
Load capacitance at OUT+ and/or OUT−, C_L				100	pF
MSTR CLK frequency (see Note 4)		0.075	5	10.368	MHz
Analog input amplifier common mode input voltage (see Note 5)				±1.5	V
A/D or D/A conversion rate				20	kHz
Operating free-air temperature, T_A	TLC32040C, TLC32041C, TLC32042C	0		70	°C
	TLC32040I, TLC32041I, TLC32042I	−40		85	

NOTES: 2. Voltages at analog inputs and outputs, REF, V_{CC+}, and V_{CC-}, are with respect to the ANLG GND terminal. Voltages at digital inputs and outputs and V_{DD} are with respect to the DGTL GND terminal.
3. The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels and temperature only.
4. The bandpass low-pass switched-capacitor filter response specifications apply only when the switched-capacitor clock frequency is 288 kHz. For switched-capacitor filter clocks at frequencies other than 288 kHz, the filter response is shifted by the ratio of switched-capacitor filter clock frequency to 288 kHz.
5. This range applies when (IN+ – IN−) or (AUX IN+ – AUX IN−) equals ± 6 V.

electrical characteristics over recommended operating free-air temperature range, V_{CC+} = 5 V, V_{CC-} = −5 V, V_{DD} = 5 V (unless otherwise noted)

total device, MSTR CLK frequency = 5.184 MHz, outputs not loaded

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage		V_{DD} = 4.75 V, I_{OH} = −300 µA	2.4			V
V_{OL}	Low-level output voltage		V_{DD} = 4.75 V, I_{OL} = 2 mA			0.4	V
I_{CC+}	Supply current from V_{CC+}	TLC3204_C				35	mA
		TLC3204_I				40	
I_{CC-}	Supply current from V_{CC-}	TLC3204_C				−35	mA
		TLC3204_I				−40	
I_{DD}	Supply current from V_{DD}		$f_{MSTR\ CLK}$ = 5.184 MHz			7	mA
V_{ref}	Internal reference output voltage			3		3.3	V
α_{Vref}	Temperature coefficient of internal reference voltage				200		ppm/°C
r_o	Output resistance at REF				100		kΩ

receive amplifier input

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
	A/D converter offset error (filters bypassed)			25	65	mV
	A/D converter offset error (filters in)			25	65	mV
CMRR	Common-mode rejection ratio at IN+, IN−, or AUX IN+, AUX IN−	See Note 6		55		dB
r_i	Input resistance at IN+, IN−, or AUX IN+, AUX IN−, REF			100		kΩ

transmit filter output

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OO}	Output offset voltage at OUT+, OUT−, (single-ended relative to ANG GND)			15	75	mV
V_{OM}	Maximum peak output voltage swing across R_L at OUT+ or OUT−, (single ended)	$R_L \geq 300\ \Omega$, Offset voltage = 0	±3			V
V_{OM}	Maximum peak output voltage swing between R_L at OUT+ and OUT−, (differential output)	$R_L \geq 600\ \Omega$	±6			V

† All typical values are at T_A = 25°C.

NOTE 6: The test condition is a 0-dBm, 1-kHz input signal with an 8-kHz conversion rate.

electrical characteristics over recommended operating free-air temperature range, V_{CC+} = 5 V, V_{CC-} = −5 V, V_{DD} = 5 V (unless otherwise noted) (continued)

system distortion specifications, SCF clock frequency = 288 kHz

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
Attenuation of second harmonic of A/D input signal	Single ended	V_{in} = −0.5 dB to −24 dB referred to V_{ref}, See Note 7		70		dB
	Differential		62	70		
Attenuation of third and higher harmonics of A/D input signal	Single ended	V_{in} = −0.5 dB to −24 dB referred to V_{ref}, See Note 7		65		dB
	Differential		57	65		
Attenuation of second harmonic of D/A input signal	Single ended	V_{in} = −0 dB to −24 dB referred to V_{ref}, See Note 7		70		dB
	Differential		62	70		
Attenuation of third and higher harmonics of D/A input signal	Single ended	V_{in} = −0 dB to −24 dB referred to V_{ref}, See Note 7		65		dB
	Differential		57	65		

A/D channel signal-to-distortion ratio

PARAMETER	TEST CONDITIONS (see note 7)	A_V = 1‡		A_V = 2‡		A_V = 4‡		UNIT
		MIN	MAX	MIN	MAX	MIN	MAX	
A/D channel signal-to-distortion ratio	V_{in} = −6 dB to −0.1 dB	58		>58§		>58§		dB
	V_{in} = −12 dB to −6 dB	58		58		>58§		
	V_{in} = −18 dB to −12 dB	56		58		58		
	V_{in} = −24 dB to −18 dB	50		56		58		
	V_{in} = −30 dB to −24 dB	44		50		56		
	V_{in} = −36 dB to −30 dB	38		44		50		
	V_{in} = −42 dB to −36 dB	32		38		44		
	V_{in} = −48 dB to −42 dB	26		32		38		
	V_{in} = −54 dB to −48 dB	20		26		32		

D/A channel signal-to-distortion ratio

PARAMETER	TEST CONDITIONS (see note 7)	MIN	MAX	UNIT
D/A channel signal-to-distortion ratio	V_{in} = −6 dB to 0 dB	58		dB
	V_{in} = −12 dB to −6 dB	58		
	V_{in} = −18 dB to −12 dB	56		
	V_{in} = −24 dB to −18 dB	50		
	V_{in} = −30 dB to −24 dB	44		
	V_{in} = −36 dB to −30 dB	38		
	V_{in} = −42 dB to −36 dB	32		
	V_{in} = −48 dB to −42 dB	26		
	V_{in} = −54 dB to −48 dB	20		

† All typical values are at T_A = 25°C.
‡ A_V is the programmable gain of the input amplifier.
§ A value >58 is overrange and signal clipping occurs.
NOTE 7: The test condition V_{in} is a 1-kHz input signal with an 8-kHz conversion rate (0 dB relative to V_{ref}). The load impedance for the DAC is 600 Ω.

electrical characteristics (continued)

gain and dynamic range

PARAMETER	TEST CONDITIONS	MIN	TYP†	MAX	UNIT
Absolute transmit gain tracking error while transmitting into 600 Ω	−48-dB to 0-dB signal range, see Note 8		±0.05	±0.15	dB
Absolute receive gain tracking error	−48-dB to 0-dB signal range, see Note 8		±0.05	±0.15	dB
Absolute gain of the A/D channel	Signal input is a −0.5-dB, 1-kHz sinewave		0.2		dB
Absolute gain of the D/A channel	Signal input is a 0-dB, 1-kHz sinewave		−0.3		dB

NOTE 8: Gain tracking is relative to the absolute gain at 1 kHz and 0 dB (0 dB relative to V_{ref}).

power supply rejection and crosstalk attenuation

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{CC+} or V_{CC-} supply voltage rejection ratio, receive channel	f = 0 to 30 kHz	Idle channel, supply signal at 200 mV p-p measured at DR (ADC output)		30		dB
	f = 30 kHz to 50 kHz			45		
V_{CC+} or V_{CC-} supply voltage rejection ratio, transmit channel (single ended)	f = 0 to 30 kHz	Idle channel, supply signal at 200 mV p-p measured at OUT+		30		dB
	f = 30 kHz to 50 kHz			45		
Crosswalk attenuation, transmit-to-receive (single ended)				80		dB

† All typical values are at T_A = 25°C.

delay distortion, SCF clock frequency = 288 kHz ±2%, input (IN+ – IN–) is ±3-V sinewave

Please refer to filter response graphs for delay distortion specifications.

TLC32040 and TLC32041 bandpass filter transfer function (see curves), SCF clock frequency = 288 kHz, ±2%, input (IN+ – IN–) is a ±3-V sinewave (see Note 9)

PARAMETER	TEST CONDITIONS	FREQUENCY RANGE	MIN	MAX	UNIT
Filter gain (see Note 10)	Input signal reference is 0 dB	f = 100 Hz		–42	dB
		f = 170 Hz		–25	
		300 Hz ≤ f ≤ 3.4 kHz	–0.5	0.5	
		f = 4 kHz		–16	
		f ≥ 4.6 kHz		–58	

TLC32042 bandpass filter transfer function (see curves), SCF clock frequency = 288 kHz ±2%, input (IN+ – IN–) is a ±3-V sinewave (see Note 9)

PARAMETER	TEST CONDITIONS	FREQUENCY RANGE	MIN	MAX	UNIT
Filter gain (see Note 10)	Input signal reference is 0 dB	f = 100 Hz		–27	dB
		f = 170 Hz		–2	
		300 Hz ≤ f ≤ 3.4 kHz	–0.5	0.5	
		f = 4 kHz		–16	
		f ≥ 4.6 kHz		–58	

low-pass filter transfer function, SCF clock frequency = 288 kHz ±2% (see Note 9)

PARAMETER	TEST CONDITIONS	FREQUENCY RANGE	MIN	MAX	UNIT
Filter gain (see Note 10)	Output signal reference is 0 dB	f ≤ 3.4 kHz	–0.5	0.5	dB
		f = 3.6 kHz		–4	
		f = 4 kHz		–30	
		f ≥ 4.4 kHz		–58	

serial port

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	I_{OH} = –300 µA	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 2 mA			0.4	V
I_I	Input current				±10	µA
C_i	Input capacitance			15		pF
C_o	Output capacitance			15		pF

† All typical values are at T_A = 25°C.

NOTES: 9. The above filter specifications are for a switched-capacitor filter clock range of 288 kHz ±2%. For switched-capacitor filter clocks at frequencies other than 288 kHz ±2%, the filter response is shifted by the ratio of switched-capacitor filter clock frequency to 288 kHz.

10. The filter gain outside of the passband is measured with respect to the gain at 1 kHz. The filter gain within the passband is measured with respect to the average gain within the passband. The passbands are 300 to 3400 Hz and 0 to 3400 Hz for the bandpass and low-pass filters respectively.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

operating characteristics over recommended operating free-air temperature range, V_{CC+} = 5 V, V_{CC-} = −5 V, V_{DD} = 5 V

noise (measurement includes low-pass and bandpass switched-capacitor filters)

PARAMETER		TEST CONDITIONS	TYP†	MAX	UNIT
Transmit noise	Single ended	DX input = 0000000000000, constant input code	200		μV rms
	Differential		300	500	μV rms
			20		dBrncO
Receive noise (see Note 11)		Inputs grounded, gain = 1	300	475	μV rms
			20		dBrncO

timing requirements

serial port recommended input signals

		MIN	MAX	UNIT
$t_{c(MCLK)}$	Master clock cycle time	95		ns
$t_{r(MCLK)}$	Master clock rise time		10	ns
$t_{f(MCLK)}$	Master clock fall time		10	ns
	Master clock duty cycle	42%	58%	
	$\overline{RESET}$ pulse duration (see Note 12)	800		ns
$t_{su(DX)}$	DX setup time before SCLK↓	20		ns
$t_{h(DX)}$	DX hold time after SCLK↓	$t_{c(SCLK)}/4$		ns

NOTES: 11. The noise is reffered to the input with a buffer gain of one. If the buffer gain is two or four, the noise figure will be correspondingly reduced. The noise is computed by statistically evaluating the digital output of the A/D converter.

12. $\overline{RESET}$ pulse duration is the amount of time that the reset pin is held below 0.8 V after the power supplies have reached their recommended values.

serial port — AIC output signals, C_L = 30 pF for SHIFT CLK output, C_L = 15 pF for all other outputs

		MIN	TYP†	MAX	UNIT
$t_{c(SCLK)}$	Shift clock (SCLK) cycle time	380			ns
$t_{f(SCLK)}$	Shift clock (SCLK) fall time		3	8	ns
$t_{r(SCLK)}$	Shift clock (SCLK) rise time		3	8	ns
	Shift clock (SCLK) duty cycle	45		55	%
$t_{d(CH-FL)}$	Delay from SCLK↑ to $\overline{FSR}/\overline{FSX}/\overline{FSD}$↓		30		ns
$t_{d(CH-FH)}$	Delay from SCLK↑ to $\overline{FSR}/\overline{FSX}/\overline{FSD}$↑		35	90	ns
$t_{d(CH-DR)}$	DR valid after SCLK↑			90	ns
$t_{dw(CH-EL)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↓ in word mode			90	ns
$t_{dw(CH-EH)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↑ in word mode			90	ns
$t_{f(EODX)}$	$\overline{EODX}$ fall time		2	8	ns
$t_{f(EODR)}$	$\overline{EODR}$ fall time		2	8	ns
$t_{db(CH-EL)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↓ in byte mode			90	ns
$t_{db(CH-EH)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↑ in byte mode			90	ns
$t_{d(MH-SL)}$	Delay from MSTR CLK↑ to SCLK↓		65	170	ns
$t_{d(MH-SH)}$	Delay from MSTR CLK↑ to SCLK↑		65	170	ns

† Typical values are at T_A = 25°C.

timing requirements (continued)

serial port — AIC output signals

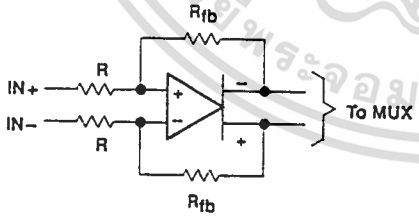
		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
$t_{c(SCLK)}$	Shift clock (SCLK) cycle time		380			ns
$t_{f(SCLK)}$	Shift clock (SCLK) fall time				50	ns
$t_{r(SCLK)}$	Shift clock (SCLK) rise time				50	ns
	Shift clock (SCLK) duty cycle		45		55	%
$t_{d(CH-FL)}$	Delay from SCLK↑ to $\overline{FSR}/\overline{FSX}$↓	C_L = 50 pF			52	ns
$t_{d(CH-FH)}$	Delay from SCLK↑ to $\overline{FSR}/\overline{FSX}$↑	C_L = 50 pF			52	ns
$t_{d(CH-DR)}$	DR valid after SCLK↑				90	ns
$t_{dw(CH-EL)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↓ in word mode				90	ns
$t_{dw(CH-EH)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↑ in word mode				90	ns
$t_{f(EODX)}$	$\overline{EODX}$ fall time				15	ns
$t_{f(EODR)}$	$\overline{EODR}$ fall time				15	ns
$t_{db(CH-EL)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↓ in byte mode				100	ns
$t_{db(CH-EH)}$	Delay from SCLK↑ to $\overline{EODX}/\overline{EODR}$↑ in byte mode				100	ns
$t_{d(MH-SL)}$	Delay from MSTR CLK↑ to SCLK↓			65		ns
$t_{d(MH-SH)}$	Delay from MSTR CLK↑ to SCLK↑			65		ns

† Typical values are at T_A = 25°C.

Table 2. Gain Control Table Analog Input Signal Required for Full-Scale A/D Conversion

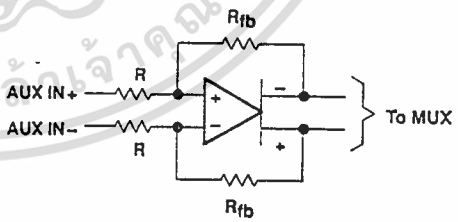
INPUT CONFIGURATIONS	CONTROL REGISTER BITS d6	CONTROL REGISTER BITS d7	ANALOG INPUT‡	A/D CONVERSION RESULT
Differential configuration Analog input = IN+ – IN– = AUX IN+ – AUX IN–	1	1	±6 V	Full scale
	0	0		
	1	0	±3 V	Full scale
	0	1	±1.5 V	Full scale
Single-ended configuration Analog input = IN+ – ANLG GND = AUX IN+ – ANLG GND	1	1	±3 V	Half scale
	0	0		
	1	0	±3 V	Full scale
	0	1	±1.5 V	Full scale

‡ In this example, V_{ref} is assumed to be 3 V. In order to minimize distortion, it is recommended that the analog input not exceed 0.1 dB below full scale.



R_{fb} = R for d6 = 1, d7 = 1
d6 = 0, d7 = 0
R_{fb} = 2R for d6 = 1, d7 = 0
R_{fb} = 4R for d6 = 0, d7 = 1

Figure 1. IN+ and IN– Gain Control Circuitry



R_{fb} = R for d6 = 1, d7 = 1
d6 = 0, d7 = 0
R_{fb} = 2R for d6 = 1, d7 = 0
R_{fb} = 4R for d6 = 0, d7 = 1

Figure 2. AUX IN+ and AUX IN– Gain Control Circuitry

sin x/x correction section

The AIC does not have sin x/x correction circuitry after the digital-to-analog converter.The sin x/x correction can be accomplished easily and efficiently in digital signal processor (DSP) software. Excellent correction accuracy can be achieved to a band edge of 3000 Hz by using a first-order digital correction filter. The results, which are shown below, are typical of the numerical correction accuracy that can be achieved for sample rates of interest. The filter requires only seven instruction cycles per sample on the TMS320 DSPs. With a 200-ns instruction cycle, nine instructions per sample represents an overhead factor of 1.4% and 1.7% for sampling rates of 8000 Hz and 9600 Hz, respectively. This correction will add a slight amount of group delay at the upper edge of the 300–3000-Hz band.

sin x/x roll-off for a zero-order hold function

The sin x/x roll-off for the AIC DAC zero-order hold function at a band-edge frequency of 3000 Hz for the various sampling rates is shown in the table below.

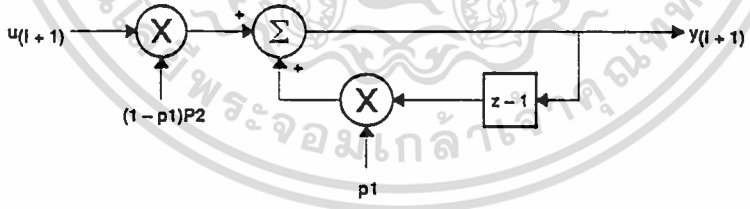
Table 3. sin x/x Roll-Off

f_S (Hz)	20 log $\frac{\sin \pi f/f_S}{\pi f/f_S}$ (f = 3000 Hz) (dB)
7200	−2.64
8000	−2.11
9600	−1.44
14400	−0.63
19200	−0.35

Note that the actual AIC sin x/x roll-off will be slightly less than the above figures, because the AIC has less than a 100% duty cycle hold interval.

correction filter

To compensate for the sin x/x roll-off of the AIC, a first-order correction filter shown below, is recommended.



The difference equation for this correction filter is:

$yi + 1 = p2(1 - p1)\,(u_{i+1}) + p1\ yi$

where the constant p1 determines the pole locations.

The resulting squared magnitude transfer function is:

$$|H(f)|^2 = \frac{p2^2(1 - p1)^2}{1 - 2p1\ \cos(2\ \pi\ f/f_s) + p1^2}$$

correction results

Table 4 below shows the optimum p values and the corresponding correction results for 8000-Hz and 9600-Hz sampling rates.

Table 4.

f (Hz)	ERROR (dB) f_s = 8000 Hz p1 = −0.14813 p2 = 0.9888	ERROR (dB) f_s = 9600 Hz p1 = −0.1307 p2 = 0.9951
300	−0.099	−0.043
600	−0.089	−0.043
900	−0.054	0
1200	−0.002	0
1500	0.041	0
1800	0.079	0.043
2100	0.100	0.043
2400	0.091	0.043
2700	−0.043	0
3000	−0.102	−0.043

TMS320 software requirements

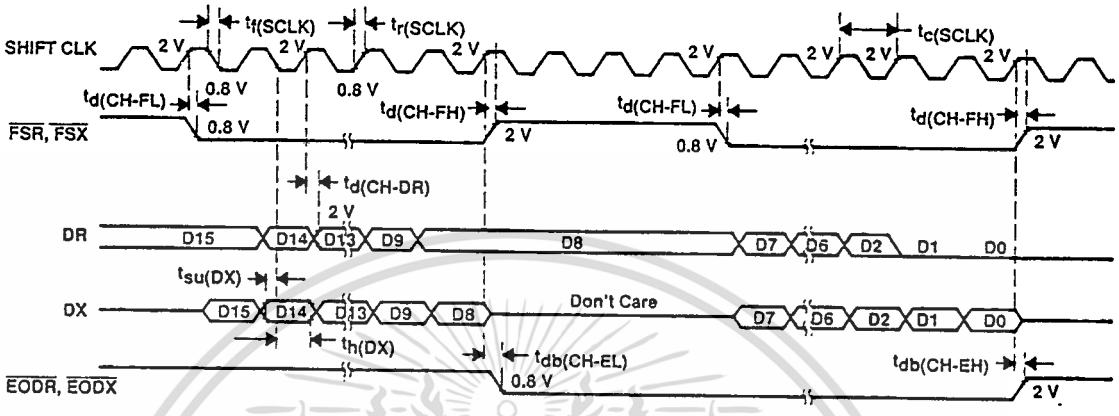
The digital correction filter equation can be written in state variable form as follows:

$$Y = k1Y + k2U$$

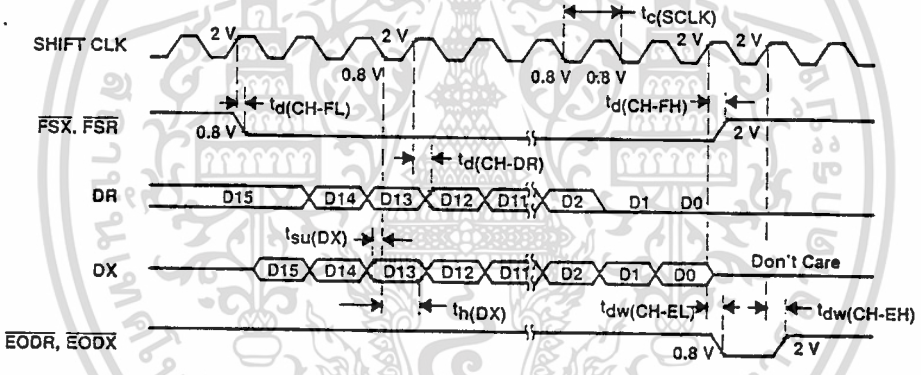
where k1 equals p1 (from the preceding page), k2 equals (1 – p1) p2 (from the preceding page), Y is the filter state, and U is the next I/O sample. The coefficients k1 and k2 must be represented as 16-bit integers. The SACH instruction (with the proper shift) will yield the correct result. With the assumption that the TMS320 processor page pointer and memory configuration are properly initialized, the equation can be executed in seven instructions or seven cycles with the following program:

```
ZAC
LT K2
MPY U
LTA K1
MPY Y
APAC
SACH (dma), (shift)
```

byte-mode timing



word-mode timing



shift-clock timing

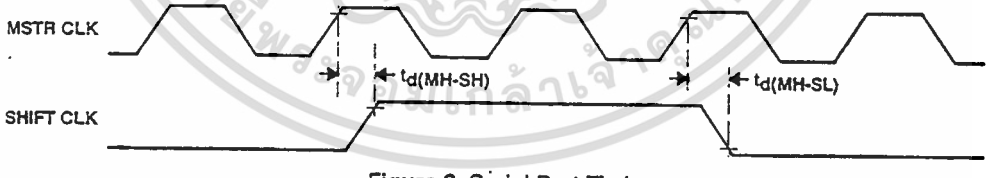


Figure 3. Serial Port Timing

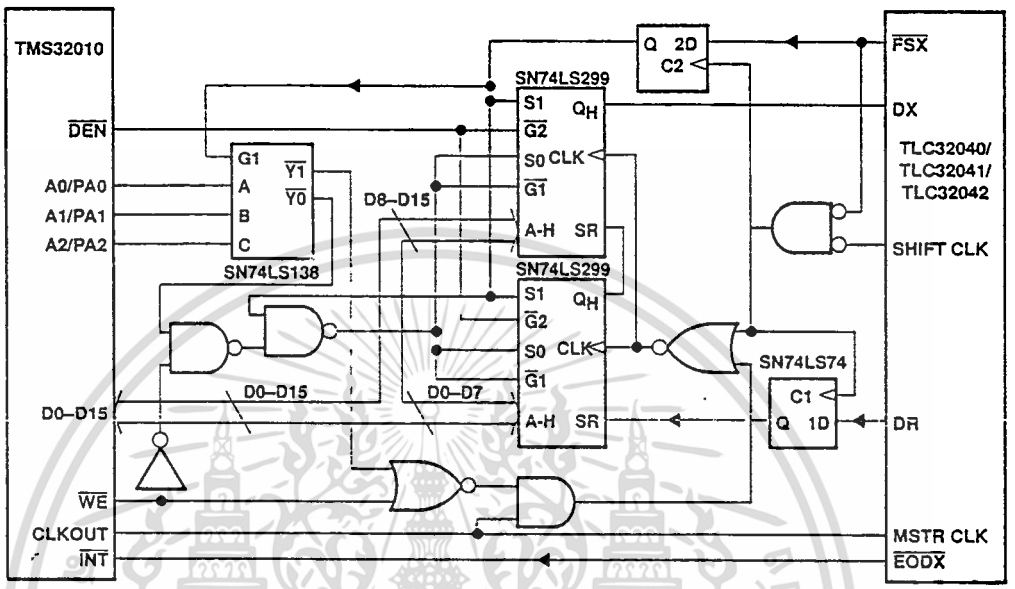
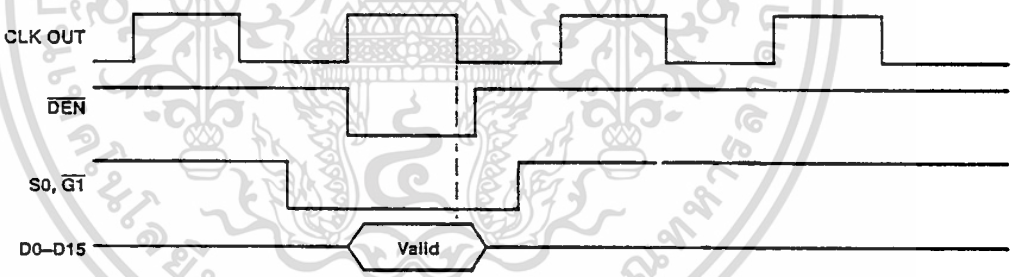


Figure 4. TMS32010-TLC32040/TLC32041/TLC32042 Interface Circuit

in instruction timing



out instruction timing

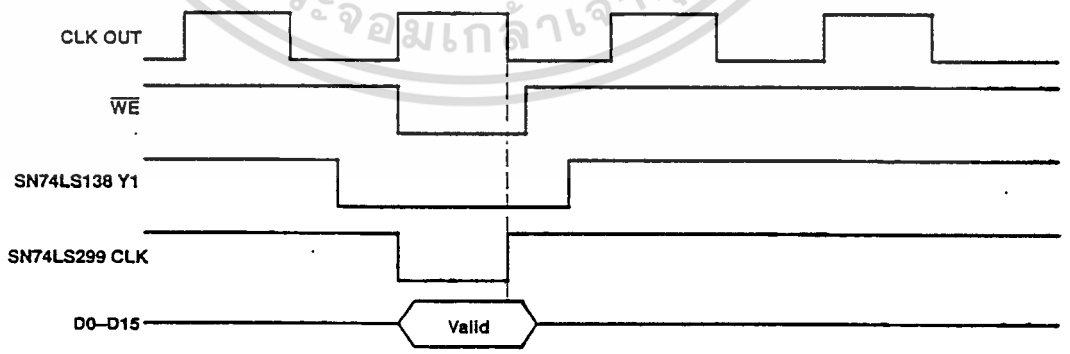
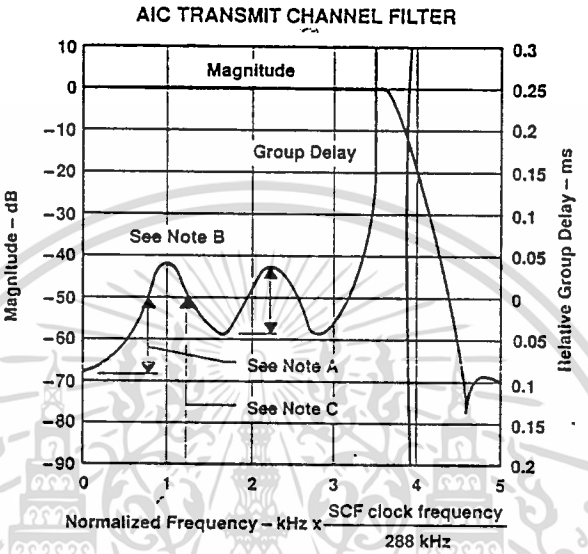


Figure 5. TMS32010-TLC32040/TLC32041/TLC32042 Interface Timing

TYPICAL CHARACTERISTICS

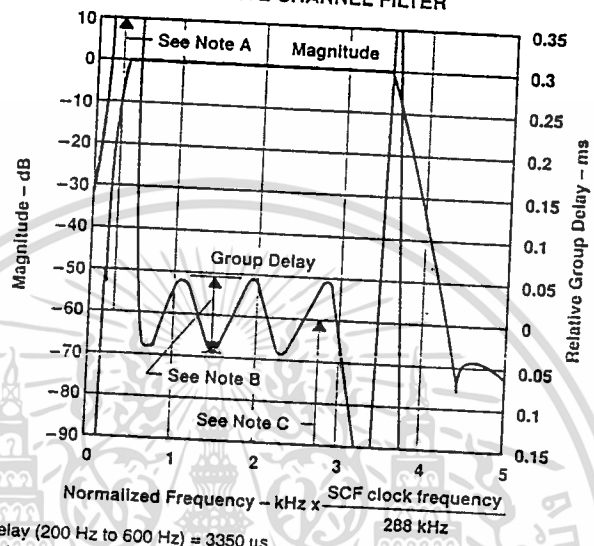


NOTES: A. Maximum relative delay (0 Hz to 600 Hz) = 125 μs
B. Maximum relative delay (600 Hz to 3000 Hz) = ± 50 μs
C. Absolute delay (600 Hz to 3000 Hz) = 700 μs
D. Test conditions are V_{CC+}, V_{CC-}, and V_{DD} within recommended operating conditions, SCF clock f = 288 kHz ±2% input = ±3-V sinewave, and T_A = 25°C.

Figure 6

TYPICAL CHARACTERISTICS

TLC32040 AND TLC32041
RECEIVE CHANNEL FILTER



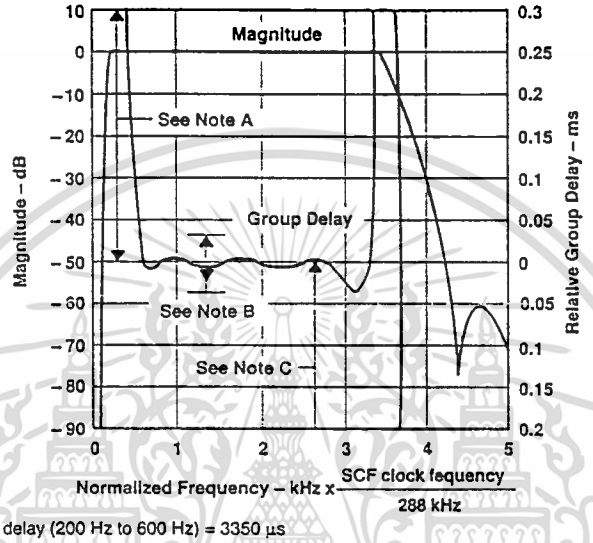
NOTES: A. Maximum relative delay (200 Hz to 600 Hz) = 3350 µs
B. Maximum relative delay (600 Hz to 3000 Hz) = ± 50 µs
C. Absolute delay (600 Hz to 3000 Hz) = 1230 µs
D. Test conditions are V_{CC+}, V_{CC-}, and V_{DD} within recommended operating conditions, SCF clock f = 288 kHz ±2%, input = ±3-V sinewave, and T_A = 25°C.

Figure 7

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

TYPICAL CHARACTERISTICS

TLC32042
RECEIVE CHANNEL FILTER



NOTES: A. Maximum relative delay (200 Hz to 600 Hz) = 3350 µs
B. Maximum relative delay (600 Hz to 3000 Hz) = ± 50 µs
C. Absolute delay (600 Hz to 3000 Hz) = 1080 µs
D. Test conditions are V_{CC+}, V_{CC-}, and V_{DD} within recommended operating conditions, SCF clock f = 288 kHz ±2%, input = ±3-V sinewave, and T_A = 25°C.

Figure 8

TYPICAL CHARACTERISTICS

A/D SIGNAL-TO-DISTORTION RATIO
vs
INPUT SIGNAL

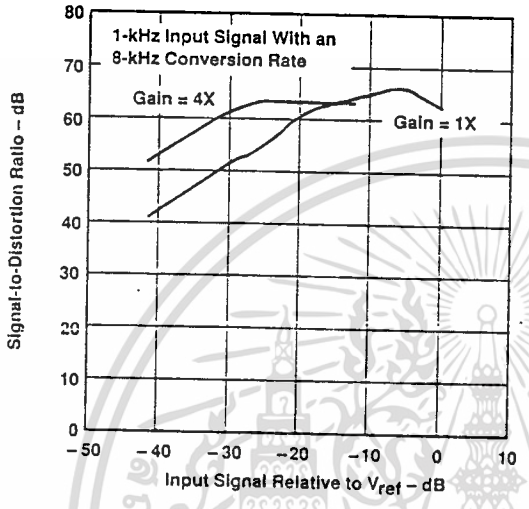


Figure 9

A/D GAIN TRACKING
(GAIN RELATIVE TO GAIN
AT 0-dB INPUT SIGNAL)

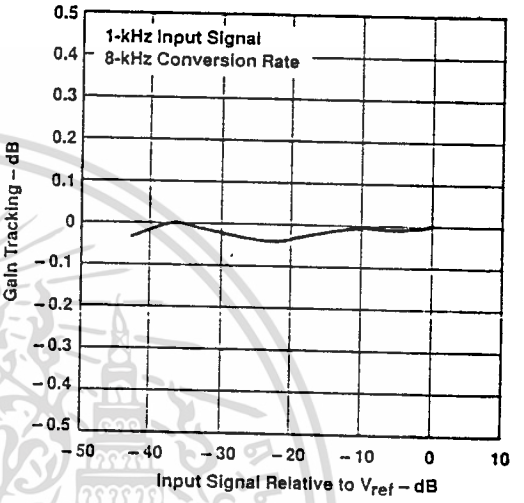


Figure 10

D/A CONVERTER SIGNAL-TO-DISTORTION RATIO
vs
INPUT SIGNAL

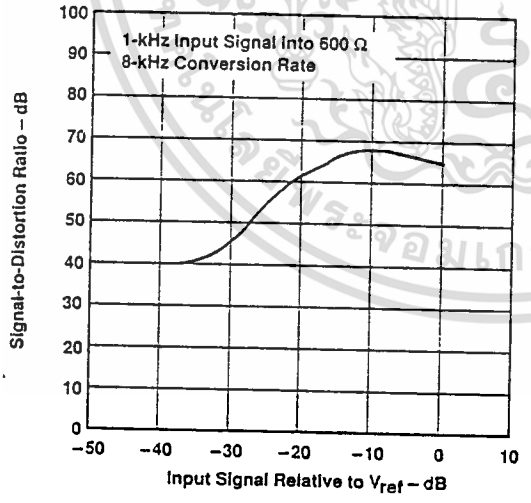


Figure 11

D/A GIAN TRACKING
vs
(GAIN RELATIVE TO GAIN
AT 0 0dB INPUT SIGNAL)

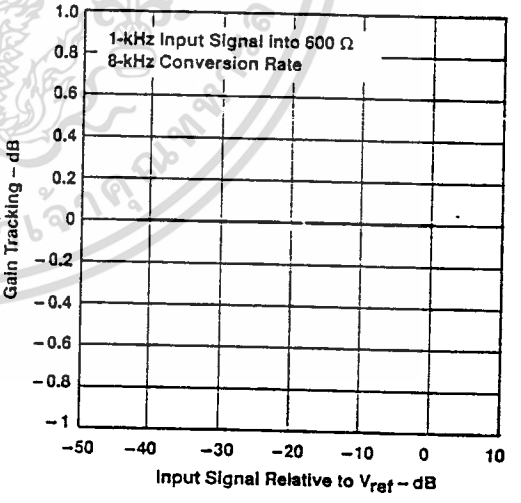


Figure 12

NOTE: Test conditions are V_{CC+}, V_{CC-} V_{DD} and within recommended operating conditions set clock f = 288 kHz ±2%, and T_A = 25°C.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

TYPICAL CHARACTERISTICS

ATTENUATION OF SECOND HARMONIC OF A/D INPUT
vs
INPUT SIGNAL

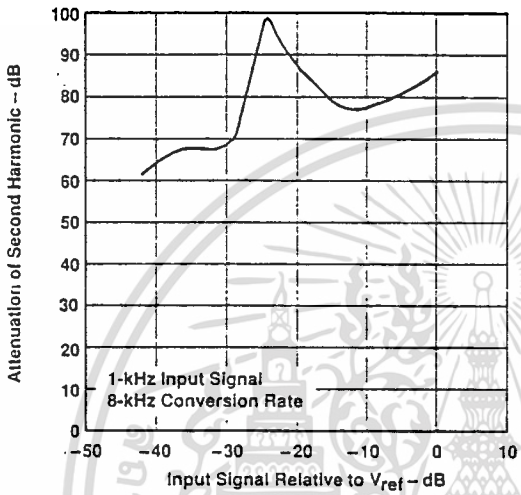


Figure 13

ATTENUATION OF THIRD HARMONIC OF A/D INPUT
vs
INPUT SIGNAL

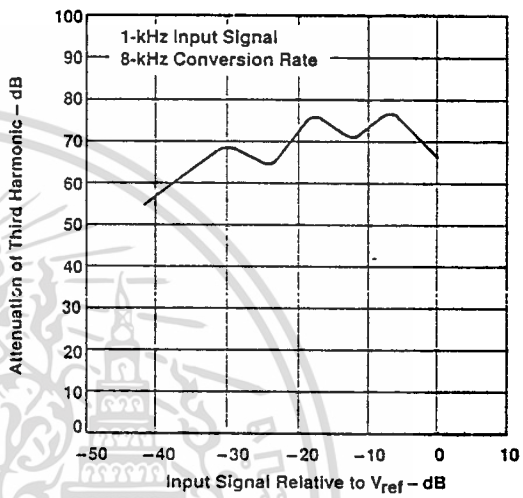


Figure 14

ATTENUATION OF SECOND HARMONIC OF D/A INPUT
vs
INPUT SIGNAL

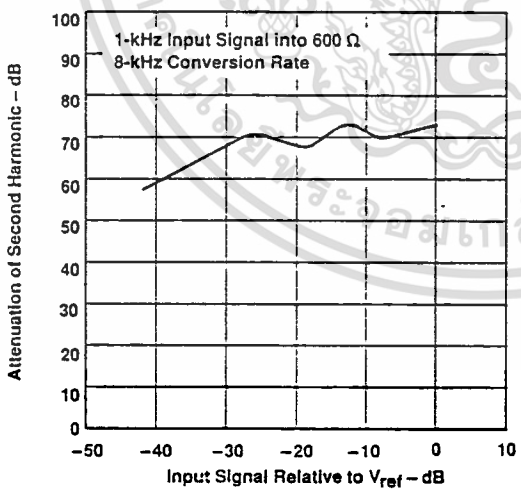


Figure 15

ATTENUATION OF THIRD HARMONIC OF D/A INPUT
vs
INPUT SIGNAL

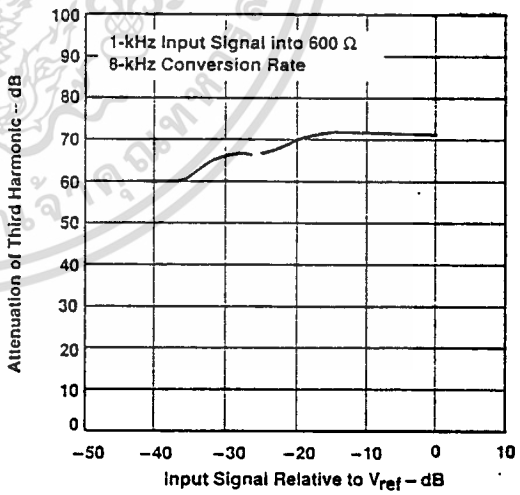
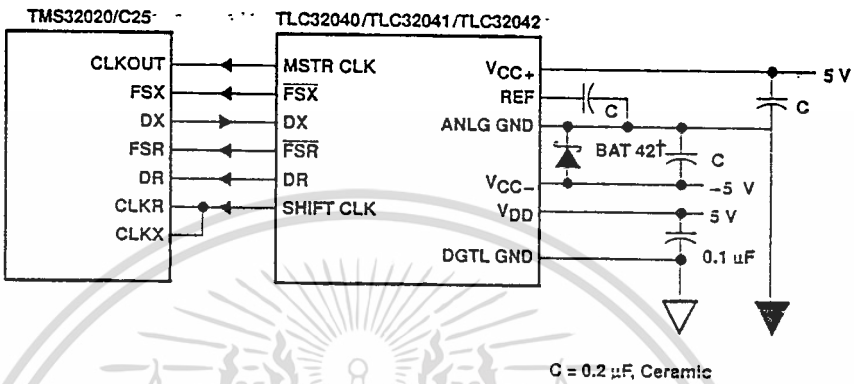


Figure 16

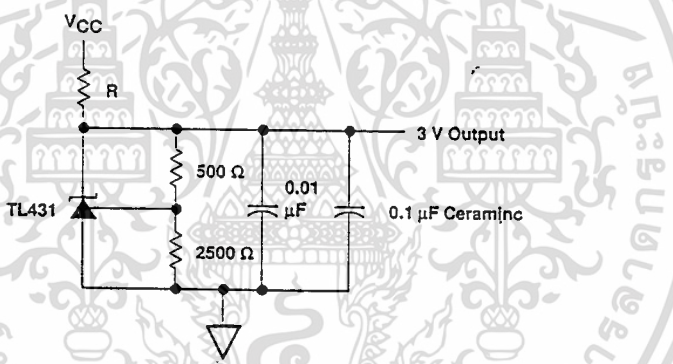
NOTE: Test conditions are V_{CC+}, V_{CC-}, and V_{DD} within recommended operating conditions set clock f = 288 kHz ±2%, and T_A = 25°C.

TYPICAL APPLICATION INFORMATION



C = 0.2 µF, Ceramic

Figure 17. AIC Interface to the TMS32020/C25 Showing Decoupling Capacitors and Schottky Diode†



For: V_{CC} = 12 V, R = 7200 Ω
V_{CC} = 10 V, R = 5600 Ω
V_{CC} = 5 V, R = 1600 Ω

Figure 18. External Reference Circuit For TLC32045

† Thomson Semiconductors