การวิเคราะห์สัญญาณที่มีการปรับเปลี่ยนและการประยุกต์ใช้งาน

Adaptive Signal Analysis and its Application

โดย

นาย ไพรัตน์ ผสมทรัพย์ 42015611

นาย สิริพงศ์ ศรีมงคล 42015622

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตรบัณฑิต

สาขา เทคโนโลยีโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

หัวข้อปริญญานิพนธ์	การวิเคราะห์สัญญาณที่มีการปรับเปลี่ยนและการประยุกต์ใช้งาน
	Adaptive Signal Analysis and its Application
โดย	นาย ไพรัตน์ ผสมทรัพย์ 42015611
	นาย สิริพงศ์ ศรีมงคล 42015622
อาจารย์ที่ปรึกษา	อาจารย์ พิชญ สุพรรณกูล
	อาจารย์ พนารัตน์ ระวิวรรณ
ภาควิชา	เทคนิคอุตสาหกรรม
สาขาวิชา	เทคโนโลยีโทรคมนาคม
ปีการศึกษา	2543

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังอนุมัติให้ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรอุตสาหกรรมศาสตรบัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

.....ประธานกรรมการ
( )

.....กรรมการ
( )

.....กรรมการ
( )

.....กรรมการ
( )

.....กรรมการ
( )

ลิขสิทธิ์ของคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อปริญญานิพนธ์ **การวิเคราะห์สัญญาณที่มีการปรับเปลี่ยนและการประยุกต์ใช้งาน**
Adaptive Signal Analysis and its Application

ชื่อนักศึกษา นาย ไพรัตน์ ผสมทรัพย์ 42015611
นาย สิริพงศ์ ศรีมงคล 42015622

อาจารย์ที่ปรึกษา อาจารย์ พิชญ สุพรรณกูล
อาจารย์ พนารัตน์ ระวิวรรณ

ปริญญา อุตสาหกรรมศาสตรบัณฑิต
สาขาวิชา เทคโนโลยีโทรคมนาคม
ภาควิชา เทคนิคอุตสาหกรรม
ปีการศึกษา 2543

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ เป็นการนำเสนอการวิเคราะห์สัญญาณที่มีการปรับเปลี่ยน และประยุกต์ใช้งาน โดยผ่านการออกแบบและควบคุมการทำงานของ DSP board รุ่นTMDS3200031 starter kit ของบริษัท Texas Instruments โดยเป็นรุ่นที่ใช้ชิป C3x DSK ที่มีความสามารถในการประมวลผลทางจุดทศนิยมที่ความถี่แซมเปิล 20 kHz โดยจะนำ DSP board นี้มาประยุกต์ใช้ในการจำลองการทำงานเป็น Spectrum Analyzer โดยการใช้ทฤษฎี FFT ที่ N เท่ากับ 512 จุด มาใช้ในการเขียนโปรแกรมวิเคราะห์สัญญาณ แล้วแสดงผลผ่านทางหน้าจอคอมพิวเตอร์ เพื่อแสดงแถบความถี่ของสัญญาณในช่วง 100 Hz – 20 kHz โดยแสดงสเปคตรัมของสัญญาณ 3 ชนิด คือ สัญญาณรูปคลื่นซายน์ สัญญาณรูปคลื่นสามเหลี่ยม และสัญญาณรูปคลื่นสี่เหลี่ยม จากผลที่ได้จะเห็นได้ว่าเป็นไปตามทฤษฎี ดังนั้น DSP board นี้สามารถนำมาวิเคราะห์สัญญาณ โดยใช้กระบวนการวิเคราะห์สัญญาณดิจิตอลได้อย่างมีประสิทธิภาพ

Title **ADAPTIVE SIGNAL ANALYSIS AND ITS APPLICATION**

Student Mr. Pairat Phasomsab 42015611
Mr. Siripong Srimongkhol 42015622

Advisor Mr. Pichaya Supanakoon
Miss. Panarat Raviwan

Degree Bachelor Degree of Industrial Technology
Programme Telecommunication Technology
Department Industrial Technology
Academic Year 2000

ABSTRACT

This project presents the adaptive signal analysis and its application which is designed and controlled by Texas Instrument's DSP board TMDS3200031 starter kit .The chip C3x DSK is used in this DSP board for floating-point processors at sampling frequency 20 kHz and this DSP board is applied for simulation Spectrum Analyzer by using FFT at length of transform (N) 512 points which will be programmed for analyzing signal. Input signals are changed from time domain to frequency domain and display the results via the monitor to illustrate signals in range at 100 Hz-20 kHz. This processing operates in a real time and display spectrum of three signals. namely. sine wave. triangular wave. and rectangular wave. The results corresponds with theory. therefore this DSP board can be used to analyze signal by the digital signal analysis. effectively.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ได้จัดทำขึ้นเป็นผลสำเร็จ ทางคณะผู้จัดทำขอขอบพระคุณบิดามารดา ที่คอยให้กำลังใจ ขอขอบพระคุณอาจารย์ที่แต่งหนังสือ DSP ทุกๆท่าน ตลอดจนอาจารย์พิชญ สุพรรณกูล อาจารย์มนต์ชัย แช่มช้อย อาจารย์พนารัตน์ ระวิวรรณ ที่ได้คอยให้คำปรึกษาและชี้แนะแนวทางในการทำปริญญานิพนธ์ฉบับนี้ ขอขอบใจเพื่อนๆ ห้อง 2M ทุกๆคน และที่ขาดไม่ได้ขอขอบพระคุณ ภาคเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ได้เปิดโอกาสให้ทางคณะผู้จัดทำได้เข้ามาศึกษา ณ.สถานที่แห่งนี้

สุดท้ายนี้ คณะผู้จัดทำ ขอขอบพระคุณคณะอาจารย์ทุกๆท่านที่กรุณาประสิทธิ์ประสาทวิชาให้แก่คณะผู้จัดทำ จนปริญญานิพนธ์สำเร็จลุล่วงไปได้ด้วยดี

ไพรัตน์ ผสมทรัพย์

สิริพงศ์ ศรีมงคล

สารบัญ

สารบัญรูป

สารบัญรูป(ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 แนวความคิดในการทำปริญญานิพนธ์ฉบับนี้

ในสมัยก่อนการประมวลสัญญาณไฟฟ้านั้นจะใช้อุปกรณ์ทางแอนะลอกนำมาประกอบเป็นวงจรกรอง เพื่อจัดรูปแบบของสัญญาณใหม่ ซึ่งคุณสมบัติของวงจรกรองก็จะสามารถปรับปรุงเปลี่ยนแปลงไปได้ตามวงจรที่ประกอบและค่าของอุปกรณ์เอง ในการประกอบวงจรทางแอนะลอกสิ่งหนึ่งที่เป็นข้อด้อยที่หลีกเลี่ยงไม่ได้ก็คือ การที่ค่าของอุปกรณ์อาจเปลี่ยนแปลงไปตามอุณหภูมิส่งผลให้การทำงานผิดพลาดไปด้วย โดยเฉพาะอย่างยิ่งหากทำงานที่อุณหภูมิสูงๆ การประกอบวงจรจะเป็นไปด้วยความยากลำบาก ผลอันเนื่องจากสัญญาณรบกวน และการเดินทางของคลื่นไฟฟ้าในความถี่สูงๆจะเดินทางไปเฉพาะที่ผิวของวัตถุเท่านั้น ทำให้การประกอบอุปกรณ์ลงบนแผ่น PCB ต้องออกแบบและทำด้วยวัสดุพิเศษ

ปัจจุบันนี้วิทยาการทางด้านคอมพิวเตอร์ได้เจริญก้าวหน้ามาก และได้ก้าวไปอย่างรวดเร็ว โดยเฉพาะเครื่องประเภทคอมพิวเตอร์ส่วนบุคคล (PC) จะมีประสิทธิภาพสูงกว่าแต่ก่อนมาก ทั้งทางด้านความจุของหน่วยความจำและความเร็วในการประมวลผลแต่ราคากลับต่ำลงเรื่อยๆ ทำให้การศึกษาในเรื่องของการประมวลผลสัญญาณไฟฟ้าดิจิตอลนั้น สามารถกระทำได้ง่ายขึ้นเพราะสามารถนำเอาคอมพิวเตอร์มาประมวลผลได้ (ซึ่งนับเป็นข้อดีมากอย่างหนึ่งของการประมวลผลด้วยคอมพิวเตอร์ เพราะสามารถที่จะปรับปรุงพัฒนาตัววงจรกรองได้อย่างต่อเนื่องและรวดเร็ว เพราะถ้ามีข้อมูลเดิมอยู่ในหน่วยความจำอยู่ตลอดเวลา) ตัดผลของข้อเสียของระบบประมวลผลแอนะลอกเรื่องการเปลี่ยนแปลงค่าของอุปกรณ์ตามอุณหภูมิไปได้ ทั้งนี้เพราะเราสามารถจะสร้างวงจรกรองได้ โดยการเขียนโปรแกรมคอมพิวเตอร์ ทั้งนี้ยังง่ายในการเปลี่ยนแปลงคุณสมบัติได้โดยการเปลี่ยนแปลงค่าของพารามิเตอร์ในโปรแกรมเท่านั้นเอง สามารถจำลองผลการทำงาน บันทึกผลไว้ประมวลต่อไปได้ และหากเป็นการกระทำกับสัญญาณความถี่สูง ปัจจุบันก็มีอุปกรณ์ต่อเชื่อมกับคอมพิวเตอร์เพื่อประมวลผลสัญญาณโดยฮาร์ดแวร์ ซึ่งจะมีความเร็วสูงมาก และได้มีการพัฒนาความเร็วสูงขึ้นไปเรื่อยๆ เช่น DSP board เป็นต้น

1.2 วัตถุประสงค์ในการทำปริญญานิพนธ์

1. เพื่อศึกษาทฤษฎีของการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) และการนำไปประยุกต์ใช้งานทางด้านการสื่อสาร
2. สามารถออกแบบการควบคุมการทำงานของ DSP board เพื่อนำสัญญาณอินพุตมาประมวลผลให้ได้เอาต์พุตตามที่ต้องการ
3. เพื่อส่งเสริมแนวทางในการพัฒนารูปแบบการประมวลผล และวิเคราะห์สัญญาณ

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. มีความเข้าใจในทฤษฎีของการประมวลผลสัญญาณทางดิจิตอล
2. สามารถออกแบบ Algorithm เพื่อประมวลผลสัญญาณดิจิตอลได้
3. รู้จักการประยุกต์ใช้ และพัฒนาโปรแกรม เพื่อวิเคราะห์และปรับเปลี่ยนรูปแบบของสัญญาณต่างๆ ได้ดีขึ้น

บทที่ 2
ทฤษฎีและหลักการ

2.1 บทนำ

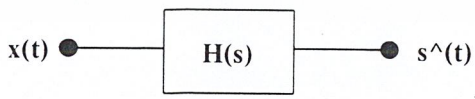
สำหรับทฤษฎีและหลักการในบทนี้จะได้นำเสนอถึงการประมวลผลสัญญาณทั้งแอนะลอกและดิจิตอล ตลอดจนทฤษฎีของการวิเคราะห์สัญญาณซึ่งจำแนกออกเป็นส่วนของอนุกรมฟูเรียส์แถบความถี่ที่ไม่ต่อเนื่อง ตัวอย่างชนิดอนุกรม รูปแบบจำนวนเชิงซ้อน จนถึงทฤษฎีของการแปลงฟูเรียส์ เป็นต้น และจะได้อธิบายถึงทฤษฎีการสุ่มสัญญาณ จนถึงวงจรกรองดิจิตอลด้วย A/D และ D/A ซึ่งจะได้นำมาใช้ในส่วนของการประมวลผลสัญญาณดิจิตอล

2.2 การประมวลผลสัญญาณแอนะลอก

วิธีการออกแบบวงจรกรองทั้งแอนะลอกและดิจิตอลนี้ จะต้องมีปัญหาเกี่ยวกับสัญญาณด้วยกันทั้งสิ้น สมมติว่า สัญญาณความถี่ต่ำ S(t) มีความถี่เป็น f เฮิร์ต และในกรณีที่มีสัญญาณรบกวนแทรกเข้ามา เราจะได้สมการของสัญญาณ X(t) เป็น

$$X(t) = S(t) + n(t) \tag{2.1}$$

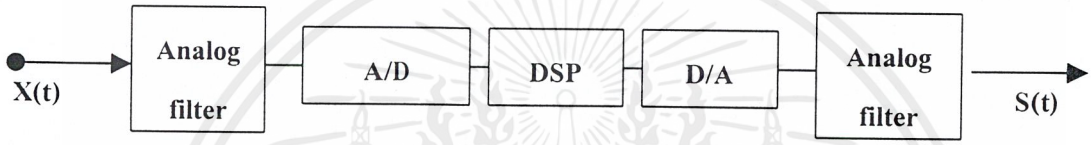
สัญญาณรบกวน n(t) นี้สมมติว่าเป็นสัญญาณรบกวนขาว (white noise) ที่แถบความถี่จำกัด โดยมีความเข้มสเปกตรัม (spectral) เป็นกำลังแบนด์วิดธ์ขนาดหนึ่งหน่วยจากกระแสไฟตรง (dc) ถึง ความถี่ของสัญญาณรบกวน (f_n) เฮิร์ต ซึ่งปัญหาของสมการ X(t) นี้ในบางกรณีเมื่อผ่านฟังก์ชั่นการถ่ายโอน H(s) แล้ว s^(t) ที่ได้ออกมาจะมีสัญญาณรบกวนที่ลดลงไป



รูปที่ 2.1 การประมวลผลสัญญาณแอนะลอกของสัญญาณแอนะลอก

2.3 การประมวลผลสัญญาณดิจิตอล (Digital Signal Processing)

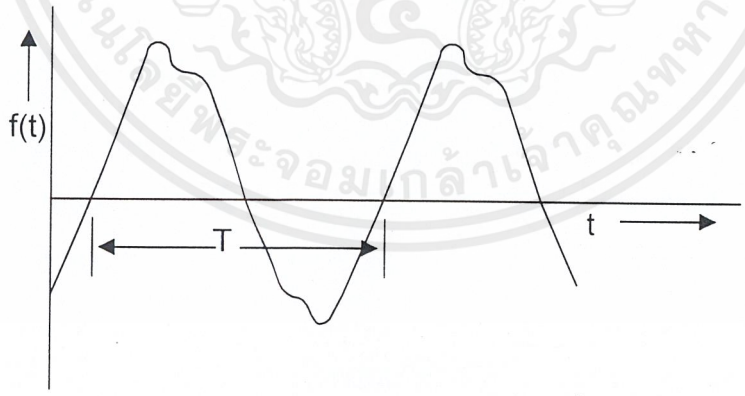
การประมวลผลสัญญาณดิจิตอลด้วยคอมพิวเตอร์นั้น สัญญาณไฟฟ้าที่จะถูกประมวลผลจะต้องถูกเปลี่ยนให้ไปเป็นสัญญาณดิจิตอลก่อน เราสามารถทำได้โดยการสุ่มสัญญาณ (Sampling) หรือเราเรียกอีกอย่างว่า การทำ A/D จากนั้นสัญญาณก็จะถูกส่งต่อไปประมวลผลที่คอมพิวเตอร์ เมื่อประมวลผลเสร็จผลที่ได้ก็จะถูกเปลี่ยนเป็นแอนะลอกโดย D/A (Digital to Analog) อีกครั้ง เพื่อใช้งานต่อไป หรืออาจจะถูกเก็บไว้ที่หน่วยความจำเพื่อใช้ในการปรับปรุงเปลี่ยนแปลงในอนาคตก็ได้ ซึ่งสามารถสรุปได้ง่ายดังรูปที่ 2.2



รูปที่ 2.2 ระบบประมวลผลสัญญาณดิจิตอลพื้นฐาน

2.4 การวิเคราะห์สัญญาณ (Signal analysis)

2.4.1 อนุกรมฟูเรียส์ (Fourier series)



รูปที่ 2.3 A periodic signal

ให้สัญญาณ f(t) เป็นสัญญาณที่ซ้ำๆกันเป็นคาบทุกๆ T วินาที ซึ่งสามารถเขียนฟังก์ชั่นของสัญญาณ f(t) แทนได้ด้วยอนุกรมฟูเรียส์ กล่าวคือ

$$f(t) = a_0 + \sum_{m=1}^{N} a_n \cos n\omega + \sum_{n=1}^{N} b_n \sin n\omega t \qquad (2.2)$$

เมื่อ a_n , b_n เป็นสัมประสิทธิ์ที่สามารถคำนวณได้จากสมการข้างล่างนี้

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos n\omega t dt \qquad (2.3)$$

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin n\omega t dt \qquad (2.4)$$

เมื่อ T เป็นคาบเวลา และ $\omega = 2\pi f$ เทอม dc คือ a_0 ซึ่งเป็นค่าเฉลี่ยของสัญญาณ f(t) ในช่วงคาบ T คำนวณได้ดังนี้

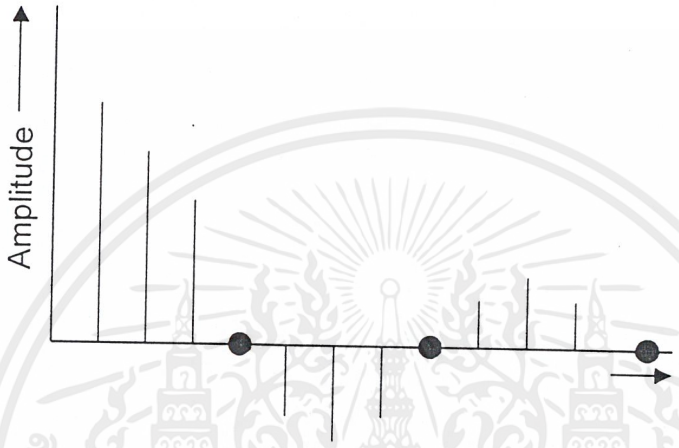
$$a_0 = \frac{1}{T} \int_{-T/2}^{T/2} f(t) dt \qquad (2.5)$$

ข้อสังเกต

1. ถ้า f(t) = f(-t) ฟังก์ชั่น f(t) เป็นฟังก์ชั่นคู่ จะเกิดความสมมาตรรอบจุดกำเนิด และจะให้เพียงเทอมของ Cosine ในสมการ 2.2
2. ถ้า f(t) = -f(-t) ฟังก์ชั่น f(t) เป็นฟังก์ชั่นคี่ จะปรากฏเพียงเทอมของ Sine เท่านั้น ในสมการ 2.2
3. ถ้า f(t+T/2) = f(t) ในสมการที่ 2.2 จะปรากฏเพียงเทอมฮาร์โมนิคคู่ (Even Harmonics)
4. f(t+T/2) = -f(t) ในสมการที่ 2.2 จะปรากฏเพียงเทอมฮาร์โมนิคคี่ (Odd Harmonics)

2.4.2 แถบความถี่ที่ไม่ต่อเนื่อง (Discrete Spectrum)

อนุกรมฟูเรียส์ที่ใช้แทนฟังก์ชั่นในแกนเวลา f(t) โดยเราจะแสดงให้เห็นองค์ประกอบทางความถี่ของสัญญาณนั้น องค์ประกอบความถี่เหล่านี้รวมกันเป็นแถบความถี่ที่ไม่ต่อเนื่อง โดยขนาดของแต่ละความถี่ถูกกำหนดโดยค่าสัมประสิทธิ์ A_n และ b_n ทุกความถี่จะเป็นฮาร์โมนิคของความถี่พื้นฐานที่เท่ากับ 1/T และช่วงกว้างของความถี่ต่างๆ นี้จะเรียกว่าแบนด์วิดท์ของสัญญาณ



รูปที่ 2.4 Discrete spectrum

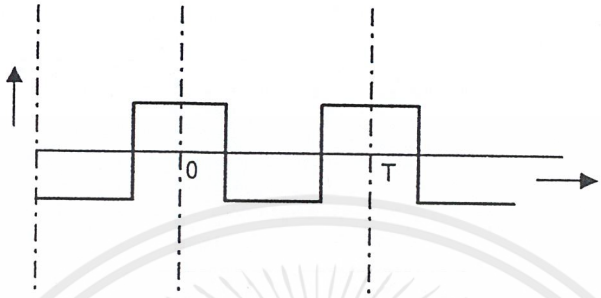
2.4.3 ตัวอย่างชนิดอนุกรม (Typical Series)

ถึงแม้แถบความถี่อาจจะประกอบด้วยความถี่ต่างๆ จำนวนอนันต์ก็ตาม ขนาดของความถี่จะลดลงเรื่อยๆ เมื่อความถี่สูงขึ้น (ค่าเพิ่มขึ้น) สำหรับในทางปฏิบัติแล้วเราพิจารณาเพียงความถี่จำนวนจำกัดที่พอเพียงกับการติดต่อสื่อสาร

ข้อควรคำนึงในการประมวลผลสัญญาณเชิงเลข โดยเฉพาะในด้านการสื่อสารนั้น จุดสำคัญของการติดต่อสื่อสารคือ การประหยัดในการใช้แถบความถี่ในระบบการติดต่อสื่อสาร ถ้าหากเรารู้ถึงแถบของความถี่จะช่วยให้ระบบการส่งและการรับสัญญาณทำได้อย่างมีประสิทธิภาพ และช่วยให้ประหยัดด้วยสำหรับลักษณะรูปคลื่นสัญญาณที่ปรากฏในระบบการประมวลผลสัญญาณเชิงเลขมีหลายรูปแบบ ในที่นี้จะแสดงอนุกรมสัญญาณเฉพาะที่ใช้กันมาก

องค์ประกอบของฟูเรียส์สำหรับรูปคลื่นตัวอย่างชนิดต่างๆ พอจะยกตัวอย่างได้ดังนี้

1. รูปคลื่นสี่เหลี่ยมสมมาตร (symmetrical square wave)



รูปที่ 2.5 รูปคลื่นแบบสมมาตร

เนื่องจาก f(t) มีลักษณะสมมาตรทางแนวนอน ค่าเฉลี่ยของพื้นที่จึงเป็นศูนย์ ทำให้เทอม dc มีค่าเท่ากับศูนย์ ($a_0 = 0$) ,และโดยที่ f(t) = f(-t) จึงทำให้สัญญาณดังกล่าวจะมีเฉพาะเทอมของ Cosine เท่านั้น กล่าวคือ $b_n = 0$ ส่วนค่า a_n คำนวณได้ดังนี้

$$a_n = \frac{2}{T}\int_{-\frac{T}{2}}^{+\frac{T}{2}} f(t)\cos n\omega t dt$$

เมื่อ

f(t) = -1 จาก -T/2 ถึง -T/4

f(t) = 1 จาก -T/4 ถึง T/4

f(t) = -1 จาก T/4 ถึง T/2

ดังนั้น

$$a_n = \frac{2}{T}\{\int_{-T/2}^{-T/4}(-1)\cos n\omega t dt + \int_{-T/4}^{T/4}(1)\cos n\omega t dt + \int_{T/4}^{T/2}(-1)\cos n\omega t dt\}$$

$$= \frac{2}{T}\{-\frac{1}{n\omega}\int_{-T/2}^{-T/4}\cos n\omega t dt + \frac{1}{n\omega}\int_{-T/4}^{T/4}\cos n\omega t dt + \frac{1}{n\omega}\int_{T/4}^{T/2}\cos n\omega t dt\}$$

$$= \frac{2}{n\omega T}\{-[\sin n\omega t\,|_{-T/2}^{-T/4}] + [\sin n\omega t\,|_{-T/4}^{T/4}] - [\sin n\omega t\,|_{T/4}^{T/2}]\}$$

$$= \frac{2}{n\omega T}\{\sin\frac{n\omega T}{4} - \sin\frac{n\omega T}{2} + \sin\frac{n\omega T}{4} + \sin\frac{n\omega T}{4} - \sin\frac{n\omega T}{2} + \sin\frac{n\omega T}{4}\}$$

$$= \frac{2}{n\omega T}\{4\sin\frac{n\omega T}{4} - 2\sin\frac{n\omega T}{2}\}$$

เนื่องจาก $\omega T = (2\pi f)T = 2\pi$ ดังนั้นจะได้

$$a_n = \frac{2}{n\pi}\{4\sin\frac{nT}{4} - 2\sin n\pi\} \tag{2.6}$$

แต่ $\sin n\pi = 0$ เมื่อ n=0.1.2.....∞

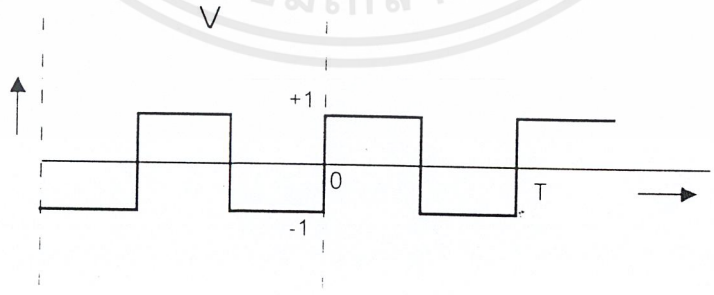
ดังนั้น $$a_n = \frac{4}{n\pi}\sin\frac{nT}{2} \tag{2.7}$$

$$a_1 = \frac{4}{\pi}\sin\left(\frac{\pi}{2}\right) = \frac{4}{\pi} \qquad a_2 = \frac{2}{\pi}\sin\pi = 0$$

$$a_3 = \frac{4}{3\pi}\sin\frac{3\pi}{2} = -\frac{4}{3\pi} \qquad a_4 = \frac{1}{\pi}\sin 2\pi = 0$$

$$f(t) = \frac{4}{\pi}(\cos\omega t - \frac{1}{3}\cos 3\omega t + \frac{1}{5}\cos 5\omega t - \ldots) \tag{2.8}$$

2. รูปคลื่นสี่เหลี่ยมแบบไม่สมมาตร (asymtrical square wave)



รูปที่ 2.6 รูปคลื่นแบบไม่สมมาตร

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

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin n\omega t dt \qquad (2.9)$$

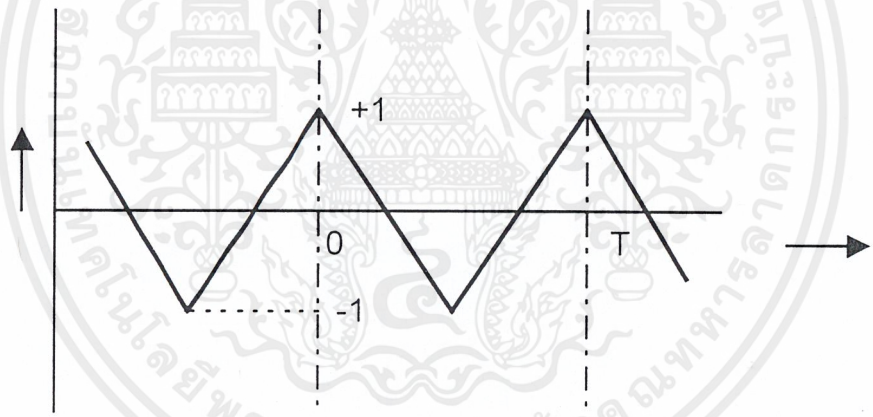
$$f(t) = -1 \qquad \text{จาก} \quad -T/2 \text{ ถึง } 0$$

$$= 1 \qquad \text{จาก} \quad 0 \text{ ถึง } T/2$$

จะได้

$$f(t) = \frac{4}{\pi}[\sin\omega t + \frac{1}{3}\sin 3\omega t + \frac{1}{5}\sin 5\omega t + ...] \qquad (2.10)$$

3. รูปคลื่นสามเหลี่ยม (triangular wave)

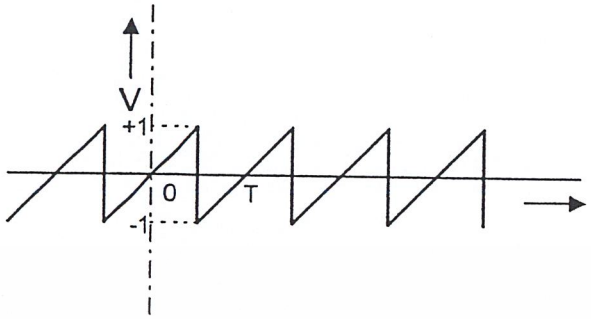


รูปที่ 2.7 รูปคลื่นสามเหลี่ยม

จากรูปข้างบนจะได้ว่า

$$f(t) = \frac{8}{\pi^2}[\cos\omega t + \frac{1}{9}\cos 3\omega t + \frac{1}{25}\sin 5\omega t + ...] \qquad (2.11)$$

4. รูปคลื่นฟันปลา (sawtooth wave)



รูปที่ 2.8 รูปคลื่นฟันปลา

จากรูปข้างบนจะได้ว่า

$$f(t) = \frac{2}{\pi}[\sin \omega t - \frac{1}{2}\sin 2\omega t + \frac{1}{3}\sin 3\omega t + ...] \qquad (2.12)$$

2.4.4 รูปแบบเชิงซ้อน (complex form)

ฟังก์ชั่น f(t) นี้สามารถเขียนแทนได้ด้วยปริมาตรเชิงซ้อน อีกวิธีหนึ่งโดยจากเทอมของสัญญาณรูปเชิงซ้อน ซึ่งจะให้ว่า

$$\cos n\omega t = \frac{e^{jn\omega t} + e^{-jn\omega t}}{2}$$

$$\sin n\omega t = \frac{e^{jn\omega t} - e^{-jn\omega t}}{2j}$$

ทำการแทนค่าของ Cos $n\omega$ t และ Sin $n\omega$ t ลงในอนุกรมฟูเรียส์ ก็จะได้

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \left(\frac{e^{jn\omega t} + e^{-jn\omega t}}{2} \right) + \sum_{n=1}^{\infty} b_n \left(\frac{e^{jn\omega t} + e^{-jn\omega t}}{2j} \right)$$

$$= a_0 + \sum_{1}^{\alpha} \left\{ \frac{(a_n - ib_n)e^{jn\omega t}}{2} + \frac{(a_n + jb_n)e^{-jn\omega t}}{2} \right\} \qquad (2.13)$$

กำหนดให้

$$c_n = \frac{1}{2}(a_n - jb_n)$$

$$c_{-n} = \frac{1}{2}(a_n + jb_n)$$

$$c_0 = a_0 \frac{1}{T}$$

โดย c_{-n} เป็น complex conjugate ของ c_n ดังนั้นการคำนวณหาค่า c_n ทำได้โดย

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)[\cos n\omega t - j\sin n\omega t]dt$$

$$c_{-n} = \frac{1}{T}\int_{-T/2}^{T/2} f(t)[\cos n\omega t - j\sin n\omega t]dt$$

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\cdot e^{-jn\omega t}dt$$

$$c_{-n}' = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\cdot e^{-jn\omega t}dt$$

ที่สุดจะได้

$$f(t) = c_0 + \sum_{n=1}^{\infty} c_n e^{jn\omega t} + \sum_{n=-\infty}^{-1} c_n e^{jn\omega t} \qquad (2.14)$$

$$f(t) = \sum_{n=-\infty}^{-1} c_n e^{jn\omega t} \qquad (2.15)$$

2.4.5 การแปลงฟูเรียส์ (Fourier Transform)

เทคนิคของอนุกรมฟูเรียส์สามารถปรับปรุงมาใช้กับรูปคลื่นที่ไม่เป็นคาบ เช่น ลูกพัลซ์เดี่ยว (simple pure) โดยการให้ค่า T เป็นค่าอนันต์ (α)

สมมติว่า f(t) เดิมเป็นสัญญาณที่เป็นคาบจะได้ว่า

$$f(t) = \sum_{n=-\alpha}^{\alpha} c_n e^{jn\omega t} \qquad (2.16)$$

เมื่อ $$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t).e^{-jn\omega t}dt$$

จากเงื่อนไขที่ว่าสัญญาณนั้นเป็นพัลซ์เดี่ยวๆ ทำให้

$$T \to \alpha \quad : \quad \omega = \frac{2\pi}{T} \to d\omega$$

ดังนั้น $$\frac{1}{T} = \frac{\omega}{2\pi} \to \frac{d\omega}{2\pi}$$

ยิ่งไปกว่านั้น ฮาร์โมนิคที่ n ของอนุกรมฟูเรียส์ $n\omega$ จะกลายเป็น $nd\omega$ ซึ่งถือว่ามีค่าเป็น ω นั่นเอง ในที่สุดเครื่องหมายผลรวม (sigma) จะกลายเป็นเครื่องหมานอินทิกรัล (Integral) ดังนั้น

$$c_n = \frac{d\omega}{2\pi} \int_{-\alpha}^{\alpha} f(t).e^{-j\omega t}\, dt$$

$$f(t) = \int_{-\alpha}^{\alpha} \frac{d\omega}{2\pi} [\int_{-\alpha}^{\alpha} f(t).e^{-j\omega t}\, dt] e^{j\omega t} \qquad (2.17)$$

เทอมที่อยู่ในเครื่องหมายก้ามปู เป็นเทอมของความถี่เพียงอย่างเดียวซึ่งให้เป็น $g(\omega)$ โดย

$$g(\omega) = \int_{-\alpha}^{\alpha} f(t).e^{-j\omega t}\, dt \qquad (2.18)$$

ซึ่ง $g(\omega)$ เรียกว่า การแปลงฟูเรียส์ของฟังก์ชั้น f(t) ดังนั้นสมการของ f(t) จะกลายเป็น

$$f(t) = \frac{1}{2\pi} \int_{-\alpha}^{\alpha} g(\omega).e^{-j\omega t}\, d\omega \qquad (2.19)$$

- ความต่อเนื่องของสเปคตรัม (continuous spectrum)

สัญญาณที่มีลักษณะเป็นพัลซ์เดี่ยวๆ สามารถเขียนแทนด้วยผลรวมของความถี่ต่างๆ จำนวนอนันต์ทั้งนี้เนื่องจาก $g(\omega)$ มี ω เป็นเทอมของความถี่ จากการอินทิเกรทนี้เองทำให้ได้ว่าสเปคตรัมที่ได้มีความต่อเนื่องตลอด ซึ่งตรงกันข้ามกับกรณีของรูปคลื่นที่เป็นคาบที่ให้สเปคตรัมที่ไม่ต่อเนื่อง โดยความจริงแล้วทุกความถี่จะ อยู่ใกล้กันมาก ทั้งนี้เพราะช่องว่างระหว่างความถี่นั้น คือ 1/T จะมีค่าเป็นศูนย์เมื่อ T เป็น α

โดยทั่วไปแล้ว $g(\omega)$ เป็นเทอมคอมเพล็กซ์ โดยที่ขนาดและเฟสสามารถนำมาพล็อตเป็นแถบความถี่ของสัญญาณ f(t) ขนาด $|g(\omega)|$ จะแปรไปตามคาบและ $|g(\omega)|$ เป็นพื้นที่ใต้กราฟภายในช่วง $d\omega$ ถูกเรียกว่า สเปคตรัลเดนซิตี้ (spectral density) แต่เนื่องจาก $d\omega/2\pi$ มีค่าเกือบเป็นหนึ่ง ดังนั้นพื้นที่ของ $|g(\omega)|\,d\omega/2\pi$ จึงมีค่าเพียงขนาดของ $|g(\omega)|$ ซึ่งเป็นแอมปลิจูดเดนซิตี้ (Amplitude density)

- การแปลงฟูเรียส์ของสัญญาณไม่ต่อเนื่อง (Discrete fourier transform)

สัญญาณต่อเนื่องที่จะนำมาแปลงด้วยฟูเรียส์ต้องมีการสุ่ม (Sampling) ให้เป็นสัญญาณที่ไม่ต่อเนื่องเพื่อนำเอาขนาดของแซมเปิ้ลไปคำนวณนี้

สูตรในการแปลงฟูเรียส์ของสัญญาณต่อเนื่องคือ

$$G(t) = \int_{-\alpha}^{\alpha} g(t).e^{-j2\pi f}\,dt \tag{2.20}$$

$$g(\omega) = \int_{-\alpha}^{\alpha} f(t).e^{-j2\pi f}\,dt \tag{2.21}$$

เมื่อแปลงให้เป็นการแปลงฟูเรียส์ของสัญญาณที่ไม่ต่อเนื่องจะได้

$$G(u) = \frac{1}{N}\sum_{x=0}^{N-1} g(x)e^{-\frac{j2\pi ux}{N}} \qquad ; u = 0,1,2,\ldots\ldots,N\text{-}1 \tag{2.22}$$

เมื่อ N เป็นจำนวนแซมเปิ้ล ในการแปลงกลับฟูเรียส์ของสัญญาณที่ไม่ต่อเนื่อง คือ

$$g(x) = \sum_{u=0}^{N-1} G(u)e^{-\frac{j2\pi ux}{N}} \qquad ; x = 0,1,2,\ldots\ldots,N\text{-}1 \tag{2.23}$$

ตัวอย่าง จากสัญญาณไม่ต่อเนื่องในรูปข้างบนจะให้ N = 4 โดย g(x) = {2,3,4,4} แถบความถี่ของสัญญาณดังกล่าวคำนวณได้จากสูตรการแปลงฟูเรียส์ของสัญญาณที่ไม่ต่อเนื่อง คือ

$$G(u) = \frac{1}{4}\sum g(x)e^{-\frac{j2\pi ux}{4}}$$

$$G(u) = \frac{1}{4}\sum_{x=0}^{3} g(x) = \frac{1}{4}[g(0) + g(1) + g(2) + g(3)]$$
$$= \frac{1}{4}[2 + 3 + 4 + 4]$$
$$= 3.25$$

สูตรการแปลงฟูเรียส์ของสัญญาณต่อเนื่อง

$$H(f) = \int_{-\alpha}^{\alpha} h(t).e^{-j2\pi ft}\,dt$$

$$h(t) = \frac{1}{2\pi}\int_{-\alpha}^{\alpha} H(f).e^{j\omega t}\,d\omega$$

สูตรการแปลงฟูเรียส์ของสัญญาณที่ไม่ต่อเนื่อง

$$H(k) = \frac{1}{N}\sum_{m=0}^{N-1} h(m)e^{-\frac{j2\pi km}{N}} \qquad ; k = 0,1,2,\ldots,N-1$$

$$h(m) = \sum_{m=0}^{N-1} h(m)e^{\frac{j2\pi km}{N}} \qquad ; m = 0,1,2,\ldots,N-1$$

จากลำดับของสัญญาณ h(m) = {2,3,4,4} ให้หาแถบความถี่ของสัญญาณนั้น คือ N = 4

$$H(k) = \frac{1}{4}\sum_{m=0}^{3} h(m)e^{-\frac{j2\pi km}{4}} = \frac{1}{4}\sum_{m=0}^{3} h(m)e^{-\frac{j\pi km}{2}}$$

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

$$= \frac{1}{4}[h(0) + h(1) + h(2) + h(3)]$$

$$= \frac{1}{4}[2 + 3 + 4 + 4] = 3.25$$

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

$$=\frac{1}{4}[h(0)e^{-j0}+h(1)e^{-\frac{j\pi}{2}}+h(2)e^{-j\pi}+h(3)e^{-\frac{j3\pi}{2}}]$$

$$=\frac{1}{4}[2(1)+3(-j)+4(-1)+4(j)]$$

$$=\frac{1}{4}[2-3j-4+4j]$$

$$=\frac{1}{4}[-2+j]$$

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

$$=\frac{1}{4}[h(0)e^{-j0}+h(1)e^{-j\pi}+h(2)e^{-j2\pi}+h(3)e^{-j3\pi}]$$

$$=\frac{1}{4}[2(1)+3(-1)+4(1)+4(-1)]$$

$$=\frac{1}{4}[2-3+4-4]$$

$$=-0.25$$

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

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

$$=\frac{1}{4}[2(1)+3(j)+4(-1)+4(-j)]$$

$$=\frac{1}{4}[2+3j-4-4j]$$

$$=-\frac{1}{4}[2+j]$$

เพราะฉะนั้น แถบความถี่คือ

$$|H(0)| = 3.25$$

$$|H(1)| = \frac{1}{4}\left|\sqrt{(-2)^2 + (1)^2}\right| = \frac{\sqrt{5}}{4}$$

$$|H(2)| = |-0.25| = 0.25$$

$$|H(3)| = \frac{1}{4}\left|\sqrt{(2)^2 + (1)^2}\right| = \frac{\sqrt{5}}{4}$$

2.4.6 การแปลงฟูเรียส์อย่างเร็ว (Fast Fourier Transform)

ขั้นตอนวิธีหรือลำดับการในการคำนวณฟูเรียส์ให้เร็วมีชื่อเรียกว่าการแปลงฟูเรียส์อย่างเร็ว คิดค้นโดย คูลีย์ (J.W. Cooley) กับทูคีย์ (J.W.Tukey) ซึ่งได้เสนอไว้ในปี ค.ศ. 1965 หลังจากนั้นทำให้เกิดการพัฒนาวิธีหลายวิธีแต่ในวิธีของคูลีย์และทูคีย์ช่วยให้การคำนวณเชิงซ้อนเพียง $N/\log_2 N$ เท่า ผลดีอีกประการหนึ่งก็คือทำให้การสร้างวงจรเฉพาะเพื่อการคำนวณ DFT ทำให้ง่าย และคำนวณได้เร็วขึ้น

จากหัวข้อที่ผ่านมาแสดงถึงการแปลงฟูเรียส์สำหรับสัญญาณไม่ต่อเนื่อง จะสังเกตได้ว่าจะมีส่วนที่เราต้องคำนวณซ้ำ ๆ คือ การคูณค่าจำนวนเชิงซ้อน ซึ่งมักเป็นเลขทศนิยมความละเอียด 1-2 เท่า กันมากทำให้เสียเวลาในการคำนวณ ดังในตารางแสดงให้เห็นถึงจำนวนที่ลดลงเมื่อมีการใช้การแปลงฟูเรียส์อย่างเร็ว เช่น N ขนาด 1024 จุด เมื่อใช้ DFT จะต้องคูณจำนวนเชิงซ้อน 1048576 แต่เมื่อใช้ FFT จะเหลือเพียง 10240 เท่านั้น

จากหัวข้อที่แล้วเราได้สูตรการแปลงฟูเรียส์ คือ

$$H(k) = \frac{1}{N}\sum_{m=0}^{N-1} h(m)W^{km} \qquad : k = 1.2.....N\text{-}1$$

โดยที่

$$W = e^{-\frac{j2\pi}{N}} \Rightarrow j = \sqrt{-1}$$

ตาราง 2.1 เปรียบเทียบจำนวนการคูณค่าเชิงซ้อนระหว่าง DFT กับ FFT

Length of Transform (N)	DFT Operations (N^2)	FFT Operations N $\log_2(N)$
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16384	896
256	65536	2048
512	262144	4608
1024	1048576	10240
2048	4194304	22528

คุณสมบัติและเทคนิคช่วยแปลงฟูเรียส์อย่างเร็วมีดังนี้

1. คุณสมบัติของการแปลงฟูเรียส์ (Motivation to search for an algorithm)

$$H(\frac{N}{2}+l) = \overline{H}(\frac{N}{2}-l) \qquad :l=1,2....N/2-1$$

โดยที่ $\overline{H}$ (k) เป็น Complex Conjugate ของ $\overline{H}$ (k)

2. เทคนิคสำหรับการลดการคำนวณ (Key to Developing the algorithm)

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

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

ทำนองเดียวกับ

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

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

ถ้าเราให้ $\bar{h}(m)$ เป็น ลำดับของข้อมูลที่ตัวแปรเป็นเลขฐานสองที่ใช้แทน h(m) ชี้เป็นเลขฐานสิบเรา จะได้ว่า

$$\begin{aligned} h(m) &= \bar{h}(m_{n-1}2^{n-1} + m_{n-2}2^{n-2} + \ldots + m_1 2^1 + m_0 2^0) \\ &= \bar{h}(m_{n-1}, m_{n-2}, \ldots, m_1, m_0) \end{aligned} \tag{2.24}$$

จากสมการที่ 2.24 จะได้ว่า

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

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

$$\begin{aligned} \sum_{m_0}\sum_{m_1}\bar{h}(m_1, m_0)W^{k(2m_1+m_0)} &= \sum_{m_0}\{\bar{h}(0, m_0)W^{km_0} + \bar{h}(1, m_0)W^{k(2+m_0)}\} \\ &= \sum_{m_0=0}^{1}\bar{h}(0, m_0)W^{km_0} + \sum_{m_0=0}^{1}\bar{h}(1, m_0)W^{k[2+m_0]} \\ &= \bar{h}(0,0) + \bar{h}(0,1)W^{k} + \bar{h}(1,0)W^{2k} + \bar{h}(1,1)W^{3k} \end{aligned} \tag{2.25}$$

ดังนั้นสมการที่ 2.25 จะได้ว่า

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

ตัวอย่างการนำไปใช้งาน เราเอากรณีที่ N = 8 เป็นพื้นฐานเบื้องต้นในการอธิบายขบวนการของ FFT นั้นคือ

$$H(k) = \frac{1}{8}\sum_{m=0}^{7} h(m)W^{km} \tag{2.26}$$

โดย k= 0,1,....7

เมื่อ $W = e^{-\frac{j2\pi}{8}} = e^{-\frac{2\pi}{4}}$ และเราให้ m เขียนอยู่ในรูปเลขฐานสอง คือ

$$m = m_2 2^2 + m_1 2^1 + m_0 2^0 \tag{2.27}$$

จากสมการ 2.26 เอา 8 คูณ ตลอดจะได้

$$8H(k) = \sum_{m=0}^{7} h(m) w^{km} \tag{2.28}$$

เมื่อแปลงเทอมทางขวามือของสมการที่ 6 ให้อยู่ในรูปของเลขฐานสอง จะได้ว่า

$$\begin{aligned} 8H(\kappa) &= \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\sum_{m_2=0}^{1} \bar{h}(m_2, m_1, m_0) w^{k[4m_2+2m_1+m_0]} \\ &= \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\sum_{m_2=0}^{1} \bar{h}(m_2, m_1, m_0) W^{k4m_2} W^{2km_2} W^{km_2} \end{aligned} \tag{2.29}$$

จากสมการที่ 2.29 เราให้ การรวมผลบวก (Summation) ในสุดเป็น m_2 นั่นคือ

$$M_2 = \sum_{m_2=0}^{1} \bar{h}(m_2, m_1, m_0) W^{k4m_2}$$

แต่เนื่องจาก $W^4 = -1$ และแทน k ด้วยเลขฐานสองจะได้

$$M_2 = \sum_{m_2=0}^{1} \bar{h}(m_2, m_1, m_0) W^{k4m_2} (-1)^{m_2[4k_2+2k_1+k_0]}$$

แต่เนื่องจาก $(-1)^{m_2[4k_2+2k_1+k_0]} = 1$, M_2 สามารถเขียนใหม่เป็น

$$M_2 = \sum_{m_2=0}^{1} \bar{h}(m_2, m_1, m_0) (-1)^{m_2[4k_2+2k_1+k_0]} \tag{2.30}$$

การรวมผลบวกของสมการที่ 2.30 เป็นการแทนค่า m_2 ด้วย 0 และ 1 ดังนั้นสมการที่ 2.30 จะมีตัวแปรที่ไม่ทราบค่าคือ k_0, m_1, และ m_0 ดังนั้น M_2 เขียนใหม่ได้เป็น

$$M_2 = \sum_{m_2=0}^{1} \bar{h}(m_2, m_1, m_0)(-1)^{m_2[4k_2+2k_1+k_0]} = \bar{h}(k_0, m_1, m_0) \qquad (2.31)$$

แทนค่าสมการ 2.31 ลงในสมการ 2.29

$$8H(k) = \sum_{m_0=0}^{1}\sum_{m_1=0}^{1} \overline{h_1}(k_0, m_1, m_0) W^{2km_1} W^{km_0} \qquad (2.32)$$

การรวมผลบวกชุดในของสมการที่ 2.32 เราสมมุติให้เป็น M_1 นั้นคือ

$$M_1 = \sum_{m_1=0}^{1} \overline{h_1}(k_0, m_1, m_0) W^{2km_1}$$

$$= \sum_{m_1=0}^{1} \overline{h_1}(k_0, m_1, m_0)(-j)^{(2k_1+k_0)m_1} \qquad (2.33)$$

โดยที่จากสมการที่ 2.33 นั้น $W^2 = e^{-j\frac{2\pi 2}{4}} = -j$ และเทอม $(-j)^{4k_2m_1} = 1$

ดังนั้น M_1 เขียนได้ใหม่เป็น

$$M_1 = \sum_{m_1=0}^{1} \overline{h_1}(k_0, m_1, m_0)(-j)^{(2k_1+k_0)m_1} \qquad (2.34)$$

จากสมการที่ 2.34 เมื่อทำการแปรค่า M_1 เป็น 0 และ 1 ดังนั้นเทอม M_1 จะเป็นตัวแปรของ k_0, k_1 และ m_0 นั่นคือสมการ 2.34 จึงกลายเป็น

$$M_1 = \overline{h_2}(k_0, m_1, m_0) \qquad (2.35)$$

แทน 2.35 ลงใน 2.31 จะได้

$$8H(k) = \sum_{m=0}^{7} \overline{h_2}(k_0, m_1, m_0) W^{m_0 k_1}$$

ในทำนองเดียวกับ M_2 และ M_1 เมื่อให้ M_0 เป็นผลการ รวมผลบวกของค่า M_0 นั้นคือ

$$\begin{aligned} M_0 &= \sum_{m_0=0}^{1} \overline{h_2}(k_0, k_1, m_0) W^{m_0[4k_2+2k_1+k_0]} \\ &= \sum_{m_0=0}^{1} \overline{h_2}(k_0, k_1, m_0) \left(\frac{1 - j^{[4k_2+2k_1+k_0]m_0}}{\sqrt{2}} \right) \end{aligned} \tag{2.36}$$

หลังจากทำการแปรค่า M_0 ไปแล้วจะพบว่า M_0 เป็นฟังก์ชันของ k_0, k_1 และ k_2 ซึ่งเราจะแทนด้วย

$$M_0 = \overline{h_3}(k_0, k_1, k_2) \tag{2.37}$$

ดังนั้นจากสมการที่ 2.29 เราจะได้คำตอบว่า

$$8H(k) = 8\overline{H}(k_2, k_1, k_0) = \overline{h}_3(k_0, k_1, k_2) \tag{2.38}$$

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

$$A_{2^r} = e^{-\frac{j2\pi}{2^r}} \qquad \text{เมื่อ r=1,2,....log}_2 \text{N} \tag{2.39}$$

โดย A_{2^r} จะมีคุณสมบัติคือ

(i) $A_{2^r} = W^{\frac{N}{2^r}}$ เมื่อ $W = e^{-\frac{j2\pi}{N}}$ (2.40a)

(ii) $(A_{2^r})^{\lambda+1} = -(A_{2^r})^l$ (2.40b)

เมื่อ $r = 1,2,....\log 2\ N;\ l = 0,1,...,2^{r-1}$

$\lambda = 2^{r-1}$

$$\text{(iii)} \quad (A_N)^{\frac{N}{2}} = 1 \tag{2.40c}$$

จากสมการที่ 2.31 เราเขียนอยู่ในเทอม $\overline{h_1}$ จะได้ว่า

$$\bar{h}_1(k_0, m_1, m_0) = \sum_{m_2} \bar{h}(k_0, m_1, m_0) A_2^{k_0}$$

นั่นคือ

$$\overline{h_1}(k_0, m_1, m_0) = \bar{h}(0, m_1, m_0) + \bar{h}(1, m_1, m_0) A_2^{k_0} \tag{2.41}$$

สมการที่ 2.40 นี้จะมีค่าไม่เป็น 0 ก็เป็น 1 จะได้ว่าค่าแต่ละค่าให้ 4 สมการคือ

case ที่ 1 $k_0 = 0$ (2.42a)

$$\bar{h}_1(0,0,0) = \bar{h}(0,0,0) + \bar{h}(1,0,0) \Rightarrow h_1(0) = h(0) + h(4)$$
$$\bar{h}_1(0,0,1) = \bar{h}(0,0,1) + \bar{h}(1,0,1) \Rightarrow h_1(1) = h(1) + h(5)$$
$$\bar{h}_1(0,1,0) = \bar{h}(0,1,0) + \bar{h}(1,1,0) \Rightarrow h_1(2) = h(2) + h(6)$$
$$\bar{h}_1(0,1,1) = \bar{h}(0,1,1) + \bar{h}(1,1,1) \Rightarrow h_1(3) = h(3) + h(7)$$

case ที่ 2 $k_0 = 1$ (2.42b)

$$\bar{h}_1(1,0,0) = \bar{h}(0,0,0) + A_2 \bar{h}(1,0,0) \Rightarrow h_1(4) = h(0) - h(4)$$
$$\bar{h}_1(1,0,1) = \bar{h}(0,0,1) + A_2 \bar{h}(1,0,1) \Rightarrow h_1(1) = h(1) - h(5)$$
$$\bar{h}_1(1,1,0) = \bar{h}(0,1,0) + A_2 \bar{h}(1,1,0) \Rightarrow h_1(6) = h(2) - h(6)$$
$$\bar{h}_1(1,1,1) = \bar{h}(0,1,1) + A_2 \bar{h}(1,1,1) \Rightarrow h_1(7) = h(3) - h(7)$$

สมการที่ 2.34 เมื่อแทน $(-j)$ ด้วยเทอม A_4 จะได้ว่า

$$\overline{h_2}(k_0, m_1, m_0) = \bar{h}_1(k_0, 0, m_0) + \bar{h}_1(k_0, 1, m_0) A_4^{(2k_1 - k_0)}$$

จากการ fixed ค่า k_0, k_1 แล้วทำการแปร m_0 ไปแต่ละครั้งจะได้ 2 สมการดังนี้ คือ

case ที่ 1 $(k_1, k_0) = (0,0)$ (2.43a)

$$\bar{h}_2(0,0,0) = \bar{h}_1(0,0,0) + \bar{h}_1(0,1,0) \Rightarrow h_2(0) = h_1(0) + h_1(2)$$
$$\bar{h}_2(0,0,1) = \bar{h}_1(0,0,1) + \bar{h}_1(0,1,1) \Rightarrow h_2(1) = h_1(1) + h_1(3)$$

case ที่ 2 $(k_1, k_0) = (0,1)$ (2.43b)

$$\bar{h}_2(1,0,0) = \bar{h}_1(1,0,0) + A_4\bar{h}_1(1,1,0) \Rightarrow h_2(4) = h_1(4) + A_4 h_1(6)$$
$$\bar{h}_2(1,0,1) = \bar{h}_1(1,0,1) + A_4\bar{h}_1(1,1,1) \Rightarrow h_2(5) = h_1(5) + A_4 h_1(7)$$

case ที่ 3 $(k_1, k_0) = (1,0)$ (2.43c)

$$\bar{h}_2(0,1,0) = \bar{h}_1(0,0,0) + A_4^2\bar{h}_1(0,1,0) \Rightarrow h_2(2) = h_1(0) - h_1(2)$$
$$\bar{h}_2(0,1,1) = \bar{h}_1(0,0,1) + A_4^2\bar{h}_1(0,1,1) \Rightarrow h_2(3) = h_1(1) - h_1(3)$$

case ที่ 4 $(k_1, k_0) = (1,1)$ (2.43d)

$$\bar{h}_2(1,1,0) = \bar{h}_1(1,0,0) + A_4^3\bar{h}_1(1,1,0) \Rightarrow h_2(6) = h_1(4) - A_4 h_1(6)$$
$$\bar{h}_2(1,1,1) = \bar{h}_1(1,0,1) + \bar{h}_1(1,1,1) \Rightarrow h_2(7) = h_1(5) - A_4 h_1(7)$$

สมการที่ 2.43a ถึง 2.43d เป็นการทำซ้ำ (iteration) ครั้งที่ 2 ดังรูปที่ 2.9 โดยการนำการทำซ้ำ ครั้งที่ 2 นี้ คือในสมการที่ 2.39 นั้นเอง

ในที่สุดสมการที่ 2.37 เราเขียนค่าสัมประสิทธิ์ในเทอมของ A_8 จะได้ว่า

$$\bar{h}_3(k_0, k_1, k_2) = \bar{h}_2(k_0, k_1, 0) + \bar{h}_2(k_0, k_1, 1)A_8^{[4k_2 + 2k_1 + k_0]}$$

ซึ่งจะได้ว่า

case ที่ 1 $(k_2, k_1, k_0) = (0,0,0)$ (2.44a)

$$\bar{h}_3(0,0,0) = \bar{h}_2(0,0,0) + \bar{h}_2(0,0,1) \Rightarrow h_3(0) = h_2(0) + h_2(1)$$

case ที่ 2 $(k_2, k_1, k_0) = (0,0,1)$ (2.44b)

$$\bar{h}_3(1,0,0) = \bar{h}_2(1,0,0) + A_8(1,0,1) \Rightarrow h_3(4) = h_2(4) + A_8 h_2(5)$$

case ที่ 3 $(k_2, k_1, k_0) = (0,1,0)$ (2.44c)

$$\bar{h}_3(0,1,0) = \bar{h}_2(0,0,0) + A_8^2(0,1,1) \Rightarrow h_3(2) = h_2(2) + A_8^2 h_2(3)$$

case ที่ 4 $(k_2, k_1, k_0) = (0,1,1)$ (2.44d)

$$\bar{h}_3(1,1,0) = \bar{h}_2(1,1,0) + A_8^2 \bar{h}_2(1,1,1) \Rightarrow h_3(6) = h_2(6) + A_8^3 h_2(7)$$

case ที่ 5 $(k_2, k_1, k_0) = (1,0,0)$ (2.44e)

$$\bar{h}_3(0,0,1) = \bar{h}_2(0,0,0) + A_8^4 \bar{h}_2(0,0,1) \Rightarrow h_3(1) = h_2(0) - A_8^4 h_2(1)$$

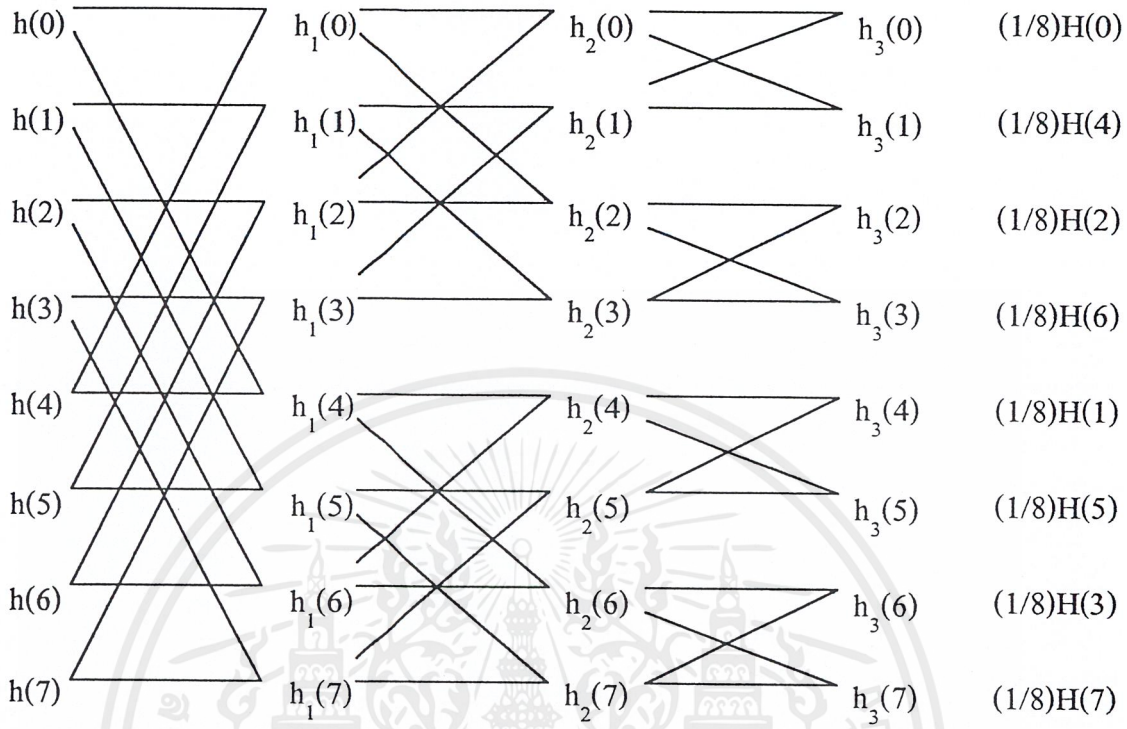
สมการที่เหลือเราใช้คุณสมบัติของ $H(\frac{N}{2}+1) = \overline{H}(\frac{N}{2}-1)$ นั่นเอง จากเอง จากสมการที่ (41)นี้เป็น การทำซ้ำ ครั้งที่ 3 นั้นคือ $r = 3$ ในสมการที่ 35 นั่นเอง
ในที่สุดจะได้ว่า

$$h_3(0) = 8H(0)$$
$$h_3(4) = 8H(1)$$
$$h_3(2) = 8H(2)$$
$$h_3(6) = 8H(3)$$
$$h_3(1) = 8H(4)$$

ส่วน $H(5), H(6)$ และ $H(7)$ ได้จากคุณสมบัติทาง Complex Conjugate หรือถ้าคำนวณจะพบว่า

$$8H(5) = h_3(5) = h_2(4) - A_8 h_2(5)$$
$$8H(6) = h_3(3) = h_2(2) - A_8^2 h_2(3)$$
$$8H(7) = h_3(7) = h_2(6) - A_8^3 h_2(7)$$

จากสมการที่ (38),(39) และ (40) จะได้ว่า Signal flow graph N = 8 คือ



รูปที่ 2.9 Signal flow graph N = 8

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

h(0) = 1	$h_1(0)$ = 4	$h_2(0)$ = 13	→ 1/8 →	H(0)
h(1) = 2	$h_1(1)$ = 4	$h_2(1)$ = -1	→ 1/8 →	H(4)
h(2) = 1	$h_1(2)$ = 2	$h_2(2)$ = 2-j	→ 1/8 →	H(2)
h(3) = 1	$h_1(3)$ = 3	$h_2(3)$ = 2+j	→ 1/8 →	H(6)
h(4) = 3	$h_1(4)$ = -2	$h_2(4)$ = -1.293+j0.707	→ 1/8 →	H(1)
h(5) = 2	$h_1(5)$ = 0	$h_2(5)$ = -2.707-j0.707	→ 1/8 →	H(5)
h(6) = 1	$h_1(6)$ = 0	$h_2(6)$ = -2.707+j0.707	→ 1/8 →	H(3)
h(7) = 2	$h_1(7)$ = -1	$h_2(7)$ = -1.293-j0.707	→ 1/8 →	H(7)

2.5 ทฤษฎีการสุ่มสัญญาณ (Sampling of continue signal)

การอธิบายถึงผลของสัญญาณแอนะลอกหลังจากการเปลี่ยนไปเป็นสัญญาณดิจิตอลแล้วจะสามารถเข้าใจได้ง่ายหากเป็นการแสดงในเชิงความถี่หรืออธิบายโดย การแปลงฟูริเยร์ และจะขอกล่าวพอสังเขปในทางคณิตศาสตร์ที่สำคัญดังต่อไปนี้

จากการแปลงฟูริเยร์ของสัญญาณแอนะลอก x(t) เราแปลงสู่ โดเมนความถี่ ได้ดังนี้

$$x(j\Omega) = \int_{-\alpha}^{\alpha} x(t)e^{-j\Omega t}\,dt \qquad (2.45)$$

$$x(t) = \frac{1}{2\pi}\int_{-\alpha}^{\alpha} x(j\Omega)e^{-j\Omega t}\,d\Omega \qquad (2.46)$$

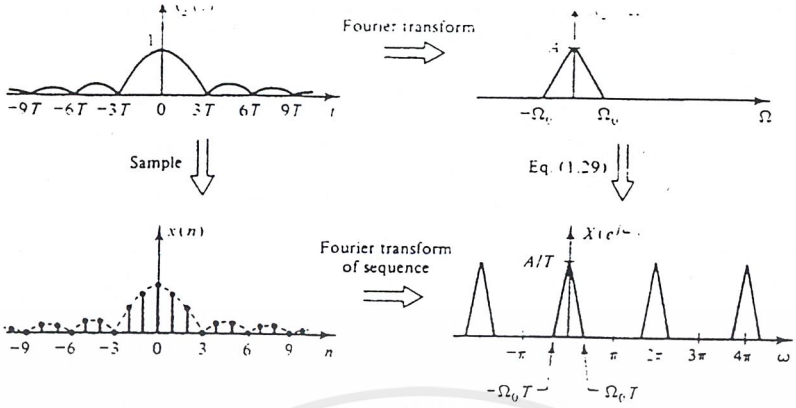
หากสัญญาณ x(t) ถูกสุ่มสัญญาณ(sampling)ด้วยระยะเท่ากันของเวลา T-second จาก $-\alpha$ ถึง $+\alpha$ สัญญาณนี้ก็จะกลายเป็นสัญญาณดิจิตอลที่เขียนแทนโดย *x(n)* หรือ

$$x(n) = x(t)\big|_{t=nt} \qquad (2.47)$$

จากข้างบนโดยอาศัยคุณสมบัติของ Delta function เพื่อใช้หาค่าของ การแปลงฟูริเยร์ดิจิตอล เราจะได้ผลดังสมการต่อไปนี้

$$x(e^{j\omega}) = \frac{1}{T}\sum_{r=-\alpha}^{\alpha} x\left[j\frac{1}{T}(\omega + 2\pi r)\right] \qquad (2.48)$$

จากสมการ $x(e^{j\omega})$ คือ ค่าสเปคตรัมของสัญญาณดิจิตอล ซึ่งก็จะมีค่าเป็นผลรวมของสเปคตรัมของสัญญาณแอนะลอก $X(j\Omega)$ ที่ตำแหน่งตั้งแต่ $r = -\alpha$ ถึง $+\alpha$ โดยแต่ละสเปคตรัม มีระยะห่างกันขนาด 2π ดังแสดงในรูปที่ 2.10



รูปที่ 2.10 แสดงถึงสเปคตรัมของสัญญาณแอนะลอกและดิจิตอล

จากรูปพบว่าเนื่องจากสปคตรัมของระบบดิจิตอลนี้จะมีการซ้ำตัวมันเองทุก 2π ส่งผลให้ความถี่ของสัญญาณ แอนะลอกอินพุตที่จะทำการสุ่มสัญญาณไม่อาจมีค่ามากกว่า π ได้ (ในทิศทางดิจิตอลแกนความถี่จะแสดงด้วย ω ดังนั้น $\omega = \Omega T$ เมื่อเทียบกับแกนความถี่ของ แอนะลอก) หรือค่าของ $\Omega_0 T < \pi$ หรือจะเขียนอีกอย่างได้ คือ

$$\frac{1}{T} > \Omega_0 / \pi \tag{2.49}$$

$$\frac{1}{T} > 2 f_0 \tag{2.50}$$

นั่นคือความถี่ของการสุ่มสัญญาณสัญญาณ (f_s) จะต้องมีค่ามากกว่าความถี่ที่จะสุ่มสัญญาณเป็นสองเท่า (อย่างน้อย) เพื่อที่จะไม่ให้เกิดการซ้อนเหลื่อม (Overlab) ของสเปคตรัมสัญญาณที่ได้แสดงในรูป 2.10 เพราะหากเกิดการซ้อนเหลื่อมแล้ว จะส่งผลให้ไม่สามารถทำการคืนกลับ (Reconstruct) สัญญาณนั้นคืนมาได้ นี่เป็นเหตุผลหนึ่งว่าทำไมที่ส่วนหน้าของภาค A/D นั้นจำเป็นต้องมีวงจรกรองแอนะลอกอยู่ ก็เพราะต้องจำกัดสัญญาณให้อยู่ในช่วงของความถี่ที่สามารถทำการคืนกลับสัญญาณกลับมาได้นั่นเอง

จากเหตุที่ว่าความถี่ของสัญญาณอินพุตที่จะนำมาสุ่มสัญญาณนั้นมีค่าได้ไม่เกิน π ได้ และสัญญาณความถี่สุ่มสัญญาณ จะต้องมีค่าเป็นสองเท่าของสัญญาณอินพุตทำให้ความถี่สุ่มสัญญาณควรจะอยู่ที่ 2π หรือเป็นจุดศูนย์กลางของสเปคตรัมเงาของแต่ละสเปคตรัมที่เกิดขึ้น หากสมมติว่า

ความถี่ในการสุ่มสัญญาณสัญญาณนั้นมีค่าเป็น 6 KHz ดังนั้น สเปคตรัม เงาก็จะเกิดอีกครั้งที่ 12 KHz และ 18 KHz และต่อเนื่องไปตามลำดับ และสัญญาณอินพุตไม่ควรจะเกิน 3 KHz

ส่วนในการ นำสัญญาณคืนกลับมานั้นเราจะมีความคิดง่ายๆ คือ จากรูปที่ 2.10 จะเห็นว่าสิ่งที่ต่างกันของสเปคตรัมดิจิตอล (Digital Spectrum) และสเปคตรัมแอนะลอก (Analog Spectrum) นั้นคือสเปคตรัมดิจิตอลนั้นจะมีสเปคตรัมเงาเกิดขึ้นมากมาย ดังนั้นหลักการก็คือเราต้องใช้วงจรกรองความถี่ต่ำผ่านที่มีผลตอบสนองความถี่คล้ายวงจรกรองความถี่ต่ำผ่านอุดมคติ (Ideal Low-Pass Filter) มาทำการเลือกเอาเฉพาะช่วงความถี่สเปคตรัมแรกเท่านั้น ($-\Omega$ ถึง $+\Omega$)

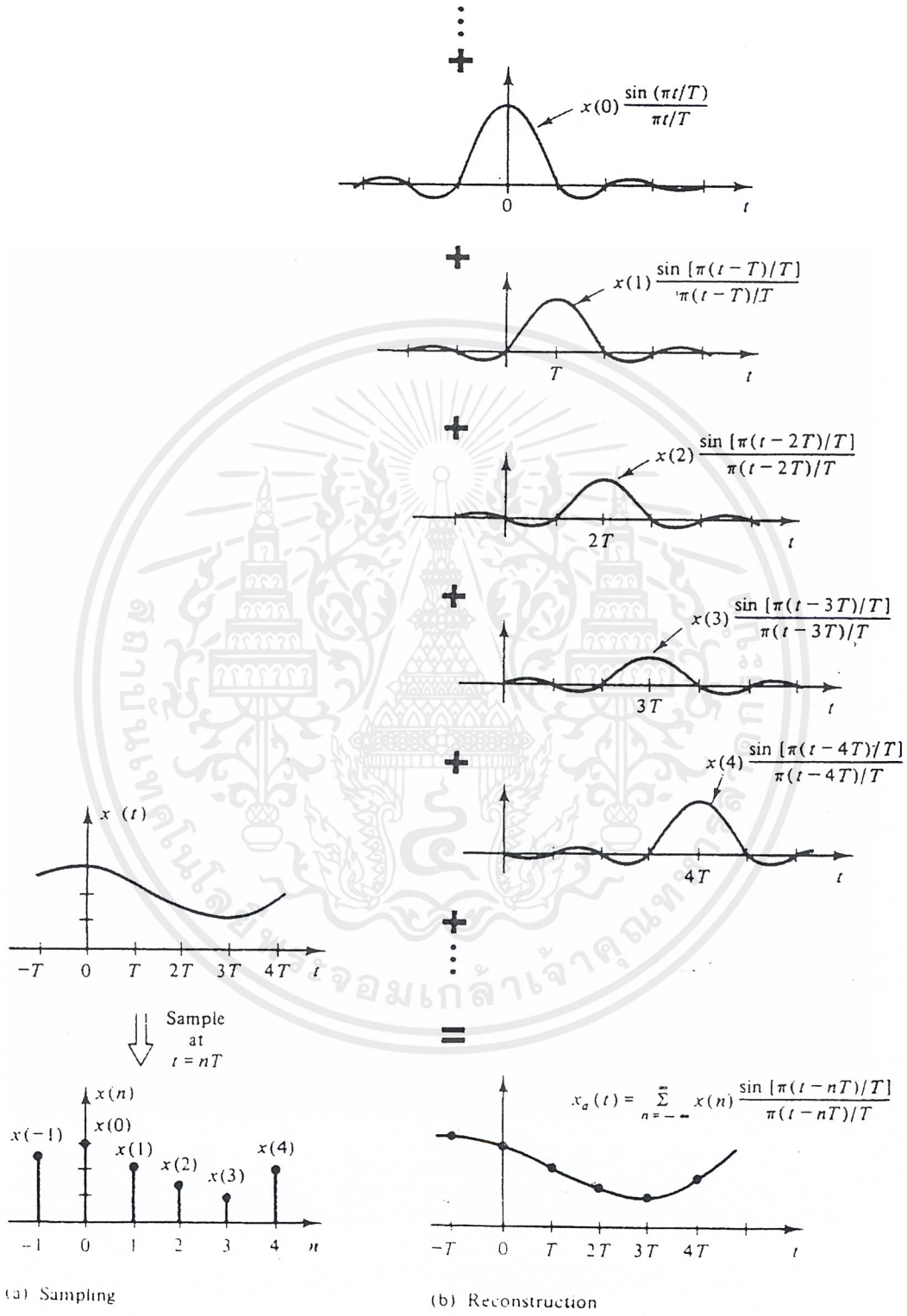
$$x(j\Omega) = \begin{cases} T.X(e^{j\omega})\big|\omega = \Omega T \\ 0 \end{cases} \qquad |\Omega| \le \frac{\pi}{T} \tag{2.51}$$

จากนั้นเราก็ทำการแปลงฟูริเยร์กลับ โดยต้องปรับสเกล (Scale) คืนด้วย ก็จะได้สัญญาณเดิมคืนมาซึ่งสมการการคืนกลับสัญญาณ ก็จะได้ดังนี้ (บางครั้งก็จะเรียกว่า Interpolation formula)

$$x(t) = \sum_{n=-\alpha}^{\alpha} x(n) \frac{\sin[\pi(t-nT)/T]}{[\pi(t-nT)/T} \tag{2.52}$$

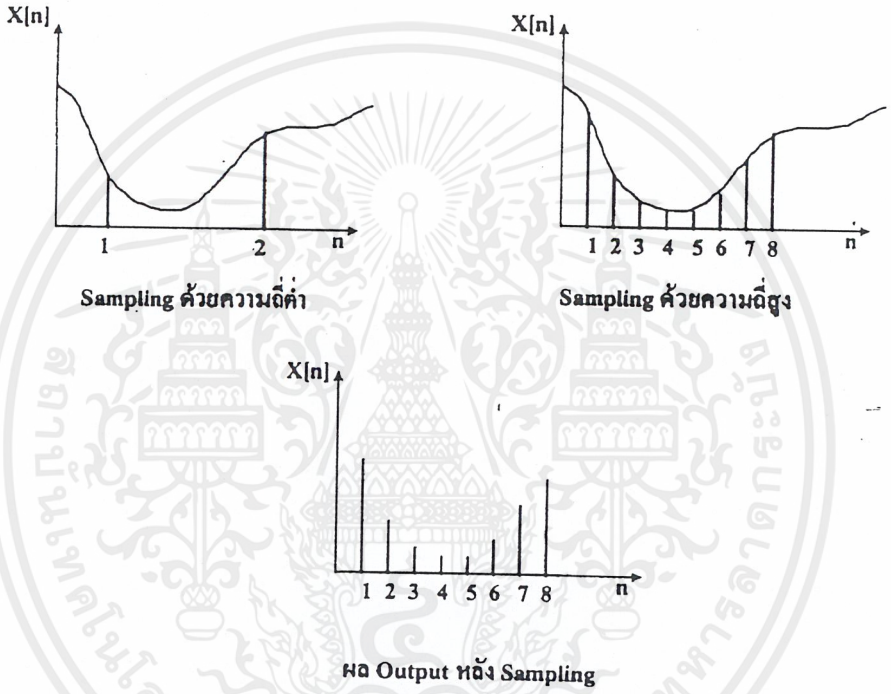
หากดูจากสมการข้างบนนั้นเป็นการบ่งบอกถึงสัญญาณ x(t) ที่ได้จากการบวกกันของค่าสัมประสิทธิ์ x(n) คูณกับ Function Sinc (Function Sinc เป็น Function ที่ใช้มากในการประมวลผลสัญญาณดิจิตอล เพราะจะมีรูปแบบ Curve ที่คล้ายกับ Sinusodial ที่ปลายทั้งสองข้าง Decay ลงสู่ Zero ที่ $\pm\alpha$) ที่ของแต่ละจุดตลอดจาก n = $-\alpha$ ถึง $-\alpha$ ซึ่งสามารถแสดงให้เห็นได้โดย รูปที่ 2.11

ดังนั้นสรุปได้ดังต่อไปนี้สมมุติเรามีสัญญาณแอนะลอกอยู่และต้องการเปลี่ยนเป็นสัญญาณดิจิตอลเพื่อประมวลผล จะสามารถทำได้โดยการสุ่มสัญญาณ หรือตัดสัญญาณนั้นออกเป็นส่วนๆ โดยในแต่ละส่วนจะแสดงให้เห็นได้ดังรูปที่ 2.12 เป็นเส้นตั้งซึ่งจะถูกวัดระดับความสูง(Amplitude) ได้เป็นเลขฐานสองออกมา (ซึ่งคอมพิวเตอร์สามารถเข้าใจได้) ส่งไปประมวลผลต่อไป จะเห็นได้ว่าส่วนที่เราตัดออกเป็นชิ้นๆนั้น จะมีระดับความสูงที่ต่างๆ กันตามตำแหน่งของสัญญาณแอนาลอกนั้นและเมื่อกระทำต่อเนื่องก็จะเป็นเส้นตั้งเรียงกันไป ดังนั้นเราจะได้สัญญาณดิจิตอลเรียงเป็นตัวเลขติดต่อกันไป ปกติเราจะเรียกสัญญาณเหล่านี้ว่าสัญญาณต่อเนื่อง (Sequence) ใช้สัญลักษณ์เป็น x[n] โดย n เป็นเหมือนเลขบอกตำแหน่งของข้อมูล(เส้นตั้ง)แต่ละตัวใน สัญญาณต่อเนื่อง สิ่งที่สำคัญมากที่สุดคือเราจะสุ่มสัญญาณสัญญาณด้วยความถี่เท่าใดจึงจะเหมาะสม (หรือจำนวนเส้นตั้งเท่าใดในหนึ่งวินาที)



รูปที่ 2.11 แสดงการสุ่มสัญญาณและการคืนกลับสัญญาณแอนะล็อก

สมมุติเรามีสัญญาณอินพุตเป็นสัญญาณเสียงซึ่งมีความถี่ประมาณ 20 Hz ถึง 20 KHz แน่นอนเราคงคิดได้โดยไม่ยากจากการดูรูปที่ 2.12 เราควรจะสุ่มสัญญาณสัญญาณที่ความถี่สูงๆ เพื่อให้สัญญาณคงรูปเดิมหลังจากสุ่มสัญญาณแล้ว ทั้งนี้เพราะหากเรา สุ่มสัญญาณด้วยความถี่ต่ำๆ การคืนสัญญาณสู่แอนาลอกด้วย D/A จะกระทำไม่ได้เลย แต่หากเราสุ่มสัญญาณสัญญาณด้วยความถี่สูงๆ แล้วสิ่งที่ตามมาคืนความถี่ที่สุ่มสัญญาณจะมีค่าสูงที่จำกัด และจะต้องเสียพื้นที่ในหน่วยเก็บความจำที่เก็บสัญญาณที่เราสุ่มสัญญาณแล้วนั้นมากขึ้น



รูปที่ 2.12 การสุ่มสัญญาณแอนะลอก

การประมวลผลสัญญาณดิจิตอลนี้ได้มีการศึกษาถึงผลของการสุ่มสัญญาณสัญญาณไฟฟ้ามาก่อนแล้วอย่างจริงจังโดย " ทฤษฎีของแชนนอล " (Shannon of famous sampling) และได้ข้อสรุปออกมาเช่นเดียวกันกับที่ได้กล่าวมาข้างต้น คือสัญญาณควรจะถูกสุ่มสัญญาณอย่างน้อยที่ความถี่สุ่มสัญญาณประมาณ 2 เท่าของความถี่สูงสุดที่เราจะทำการประมวลผล หากเราให้สัญญาณอินพุตเป็น f_i เราก็จะได้สมการง่ายๆ คือ

$$f_s = 2f_i \qquad (2.53)$$

ดังนั้นที่ความถี่ 3 KHz เราจึงควรจะสุ่มสัญญาณสัญญาณนี้ด้วยความถี่อย่างน้อย 6 KHz หรือขนาดเท่ากับ 2 Sample/sec และถ้าหากเราให้เวลาของการสุ่มสัญญาณแทนค่าด้วย T (Sample interval) เราก็จะเขียนสมการง่ายๆ ได้อีกคือ

$$T = 1/2f_1 \tag{2.54}$$

หรือเขียนได้อีกในรูปของ Ω (ซึ่งจะใช้อ้างถึงแกนความถี่ในทางแอนะลอก) ได้เป็น

$$\Omega_1 = 2\pi f_1 = \frac{2\pi}{T} \quad \text{radian/sec} \tag{2.55}$$

ซึ่งทำให้แน่ใจได้ว่าจะสามารถทำ D/A กลับสัญญาณคืนสู่แอนะลอกได้ (ในการปฏิบัติกับสัญญาณจริงๆ เรามักจะเพิ่มการสุ่มสัญญาณเป็น 2-3 เท่าของความถี่ที่ต้องการสุ่มสัญญาณ) ในรูปที่ 2.13 แสดงถึงสเปคตรัมของสัญญาณอินพุต 20 KHz-3 KHz ซึ่งสามารถหาได้โดยใช้ การแปลงฟูริเยร์ เรายังไม่ต้องสนถึงรูปร่างของสัญญาณในตอนนี้ ในรูปบอกเราได้ว่ามีความถี่อินพุตในช่วง 20 KHz-3 KHz โดย f_1 จะเป็นจุดที่เป็นความถี่สูงสุด เราพล๊อต (Plot) โดยใช้เฉพาะระดับความสูงของสัญญาณเท่านั้นตอนนี้ยังไม่ได้พูดถึงเฟสของสัญญาณแต่อย่างใด ในรูปจะเห็นได้ว่าสัญญาณจะเป็นสเปคตรัมแบบฟังก์ชั่นคู่ (even function) สังเกตได้ว่ารูปมีความถี่ทางด้านลบที่มีรูปร่างเหมือนกันกับทางด้านบวก เป็นการบอกเราได้ว่าสัญญาณแต่ละความถี่นั้นประกอบไปด้วยผลบวกของสองเอกซ์โพเนลเชียล (Exponentials) ดังนั้นส่วนประกอบของสัญญาณ $A\cos(\Omega_1)$ ซึ่งมีความสูงของสัญญาณเป็น A และความถี่ Ω radian/sec สามารถเขียนได้เป็น

$$A\cos(\Omega_1) = (A/2)\exp(j\Omega t) + (A/2)\exp(j(-\Omega)t) \tag{2.56}$$

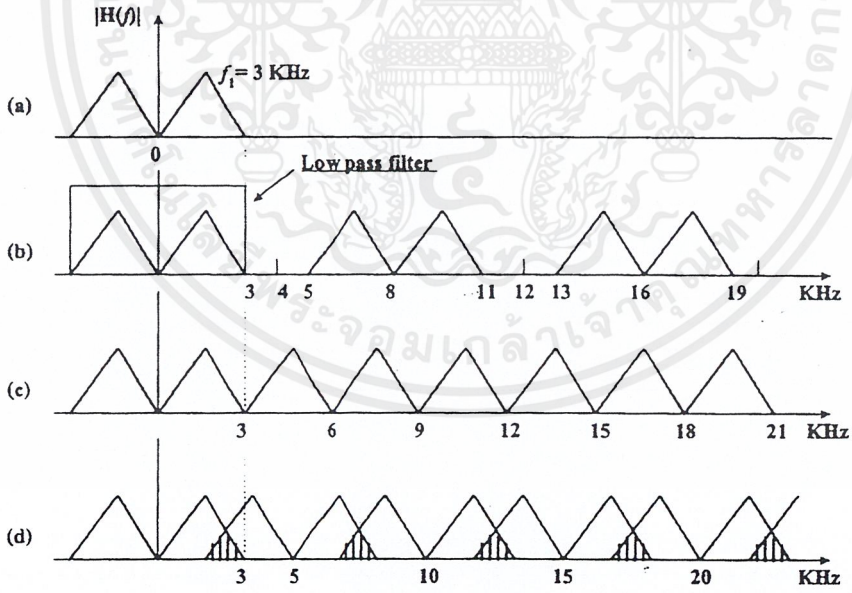
คือจะแสดงออกเป็น2 สเปคตรัมที่มีขนาดความสูงของสัญญาณเป็น (A/2) ที่ความถี่ $\pm\Omega$ และผลของการสุ่มสัญญาณ จะทำให้เกิดสเปคตรัมที่ซ้ำตัวเองไปตลอดดังแสดงในรูปที่ 2.13 (b) และหากเราสุ่มสัญญาณด้วยความถี่ 8 KHz ฉะนั้นความถี่ก็จะซ้ำตัวเองที่ 8 KHz,16 KHz และต่อไปเรื่อยๆ

จากทฤษฎีเราต้องการจะประมวลผลเพียงสัญญาณในช่วง 20 KHz-3 KHz เท่านั้นดังนั้นหากเราจะคืนสัญญาณจากดิจิตอลเป็นแอนะลอกเพื่อให้ได้ดังในรูป 2.13 (a) นั้น จึงสามารถทำได้

โดยการใช้วงจรกรองความถี่ต่ำผ่านมากรองสัญญาณเอาเฉพาะช่วงที่ต้องการเท่านั้น โดยวงจรกรองควรจะจำกัดสัญญาณที่ความถี่ที่ 5 KHz ขึ้นไป

จะเกิดอะไรขึ้นหากเราลดความถี่สุ่มสัญญาณลงจนถึงความถี่ต่ำสุดที่ทฤษฎีสุ่มสัญญาณกำหนดจากตัวอย่างนี้เราลดลงมาที่ 6 KHz ผลคือเราจะเห็นว่าความถี่จะเริ่มซ้ำที่ความถี่ 6 KHz . 12 KHz และต่อไปเรื่อยๆสัญญาณก็ยังคงจะสามารถแปลงคืนเป็นแอนะลอกหรือกระทำเป็น D/A ได้

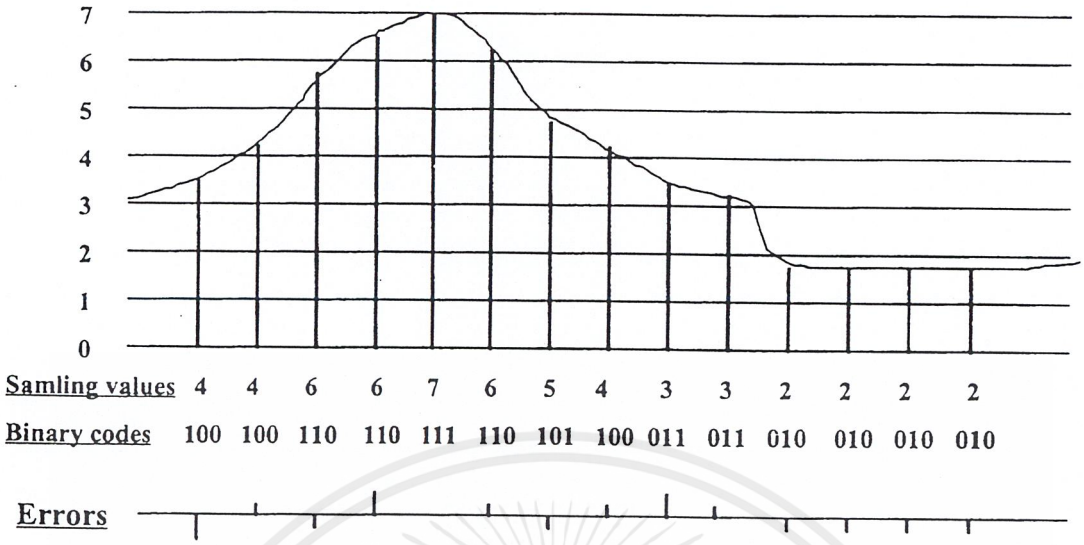
แต่หากเราลดความถี่ลงต่อไปอีกสเปคตรัมของความถี่ก็จะขยับเข้ามาอีกจากผลของความถี่สุ่มสัญญาณที่ลดลง ทำให้เกิดการซ้อนเหลื่อมของสเปคตรัมความถี่ ดังแสดงในรูปที่ 2.13 (d) นั้นคือเราจะไม่สามารถทำการแปลงสัญญาณคืนสู่แอนะลอกได้ และโดยปกติแล้วหากเราสุ่มสัญญาณด้วยความถี่ตามทฤษฎีคือ 2f ในทางปฏิบัติเราไม่สามารถคืนสัญญาณได้เพราะในทางปฏิบัติเราจะไม่สามารถออกแบบวงจรกรองความถี่ต่ำผ่านที่มีลักษณะของความถี่คัทออฟ (cut off frequency) ได้ อย่างแสดงในรูป 2.13 (b) ดังนั้นจึงควรอย่างยิ่งที่ต้องสุ่มสัญญาณ ความถี่ด้วยความถี่ที่สูงกว่าทฤษฎีกำหนดเพื่อประกันว่าสัญญาณจะสามารถคืนเป็นแอนะลอกได้แน่นอน และในที่นี้จะเรียกความถี่ของสัญญาณอินพุตว่าที่ความถี่สูงสุดว่า "Nyquist frequency" (ในที่นี้คือความถี่ 3 KHz)



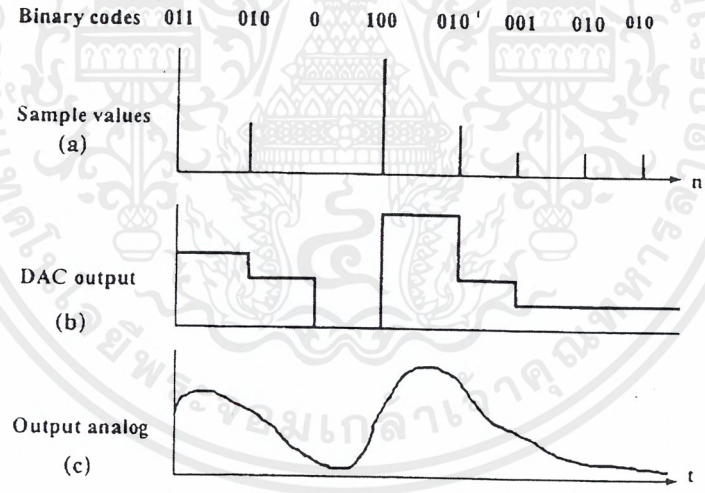
รูปที่ 2.13 ผลของการสุ่มสัญญาณ

และเรียกค่าต่ำสุดของสัญญาณอัตราสุ่มสัญญาณ (Sampling Rate) ตามทฤษฎีแล้วความถี่สูงสุดของสัญญาณอินพุต (ในที่นี้คือ 3 KHz) ก็จะเรียกได้อีกอย่างว่า "Folding frequency" (เนื่องจากว่าหากเราพับสเปคตรัมของสัญญาณดูโดยใช้ Nyquist frequency เป็นศูนย์กลางก็จะพบว่าสเปคตรัมทั้งสองด้านเท่ากันพอดี) ดังนี้จึงต้องมีวงจรกรองมาต่อที่ก่อนหน้าสัญญาณ อินพุตก่อนที่จะผ่าน A/D ทั้งนี้ก็เพื่อป้องกันความถี่อินพุตที่เกินค่ามาตรฐานการสุ่มสัญญาณ (ความถี่เกิน $f_s/2$) ซึ่งจะก่อให้เกิดการซ้อนเหลื่อม ของสเปคตรัมสัญญาณได้ เรามักจะเรียกการเกิดซ้อนเหลื่อมนี้ว่า"Aliasing" และเรียกวงจรกรองที่ป้องกันการเกิดอาการนี้ว่า "Anti-aliasing filter"

ในการสุ่มสัญญาณจะมีอีกคำหนึ่งที่มักพบบ่อยๆ คือ "Quantization noise" ซึ่งจะมีความสัมพันธ์กับค่าความละเอียดของการสุ่มสัญญาณซึ่งไม่เกี่ยวกับความถี่ของการสุ่มสัญญาณ แต่เกี่ยวกับค่าตัวเลข ฐานสอง (Binary) แต่ละตัวที่ได้จากการสุ่มสัญญาณ เช่นหากเราใช้เลขฐานสอง 8 บิท (bit) แสดงค่าของการสุ่มสัญญาณเราจะได้ความละเอียดของสัญญาณดิจิตอลเป็น $2^8 = 256$ ระดับ และหากเราใช้ค่าเลขฐานสองขนาด 16 บิท เราก็จะได้ $2^{16} = 65,536$ ระดับ ซึ่งจะละเอียดขึ้นมากแต่ก็เปลืองเนื้อที่ในหน่วยความจำมากเช่นกันแต่หากเราใช้จำนวนบิทน้อยก็จะเกิดผลไม่ดีคือ Quantization error พิจารณารูปที่ 2.14 ประกอบจะเห็นได้ว่าข้อผิดพลาด (error) จะมีเกิดขึ้นเนื่องมาจากการแบ่งสเกลหรือเรียกอีกอย่างหนึ่งว่า "Quantization levels" หากเราใช้ 3 บิท เพื่อทำการแบ่งสเกลในภาพที่ 2.14 เราก็จะได้ค่าผิดพลาดสูงสุดเป็น ±1/16 ของค่าระดับความสูงที่สูงสุดของสัญญาณ ดังนั้นในระบบดิจิตอลสิ่งที่สำคัญมากอย่างหนึ่งก็คือค่าจำนวนบิทที่เราใช้ในการแบ่งสเกล อันนี้หากมีค่ามากก็จะต้องใช้คอมพิวเตอร์ที่มีเทคโนโลยีสูงคือมีช่วงความยาวของคำ (wordlength) ที่มากบิท ซึ่งค่าใช้จ่ายก็จะสูงตามไปด้วย ในงานบางงานเราไม่จำเป็นต้องใช้จำนวนบิทมาก และสัญญาณแอนะ-ลอกจริงๆ ก็จะมีสัญญาณรบกวน (noise) เข้ามาด้วย ซึ่งหากเราสุ่มสัญญาณสัญญาณนั้นแล้วส่วนของ Quantization noise ที่เกิดขึ้นก็อาจเทียบได้เป็นข้อผิดพลาดจากสัญญาณรบกวนได้ก็ต้องใช้วิธีอื่นแก้ปัญหาต่อไป



รูปที่ 2.14 แสดงข้อผิดพลาดจากการสุ่มสัญญาณ



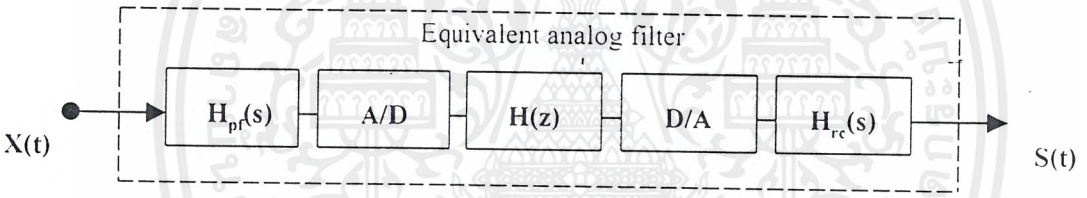
รูปที่ 2.15 แสดงการแปลงสัญญาณ D/A ชนิดหนึ่ง

หลังจากที่เราทำการประมวลผลสัญญาณดิจิตอลแล้ว ก็อาจต้องมีการแปลงสัญญาณกลับเป็นสัญญาณแอนะลอกอีกทีซึ่งนอกจากวิธีการที่ได้กล่าวมาแล้วข้างต้นคือการใช้วงจรกรองความถี่ต่ำผ่านกรองเอาเฉพาะช่วงความถี่ที่ต้องการออกมาจากนั้นก็ทำการใช้ การแปลงฟูริเยร์ กลับจากโดเมนความถี่ไปสู่โดเมนเวลาตามลำดับ แต่เรายังสามารถทำได้ในโดเมนเวลา โดยตรงได้โดยการใช้วงจรเปลี่ยนสัญญาณดิจิตอลเป็นแอนะลอก (Digital to Analog converter : DAC or D/A)

ปกติเราจะใช้วงจรง่ายๆ คือการคงค่าอันดับศูนย์(Zero-order-hold) หรือที่รู้จักดีก็ในชื่อ Sample-and-hold หลักการง่ายๆ คือวงจรรับสัญญาณดิจิตอลเข้ามาแล้วก็คงค่าระดับความสูงของสัญญาณนั้นไว้จนกระทั่งค่าของดิจิตอลตัวต่อไปเข้ามาอีกที ดังแสดงในรูปที่ 2.15 (b) และหากเรานำไปผ่านวงจรกรองความถี่ต่ำผ่านอีกทีก็จะได้รูปคลื่นที่เป็นแบบแอนะลอกดังแสดงในรูปที่ 2.15 (c) อย่างไรก็ตามวงจรกรองต้องมีค่าความถี่คัทออฟ ที่ครอบคลุมความถี่ที่เราต้องการด้วย

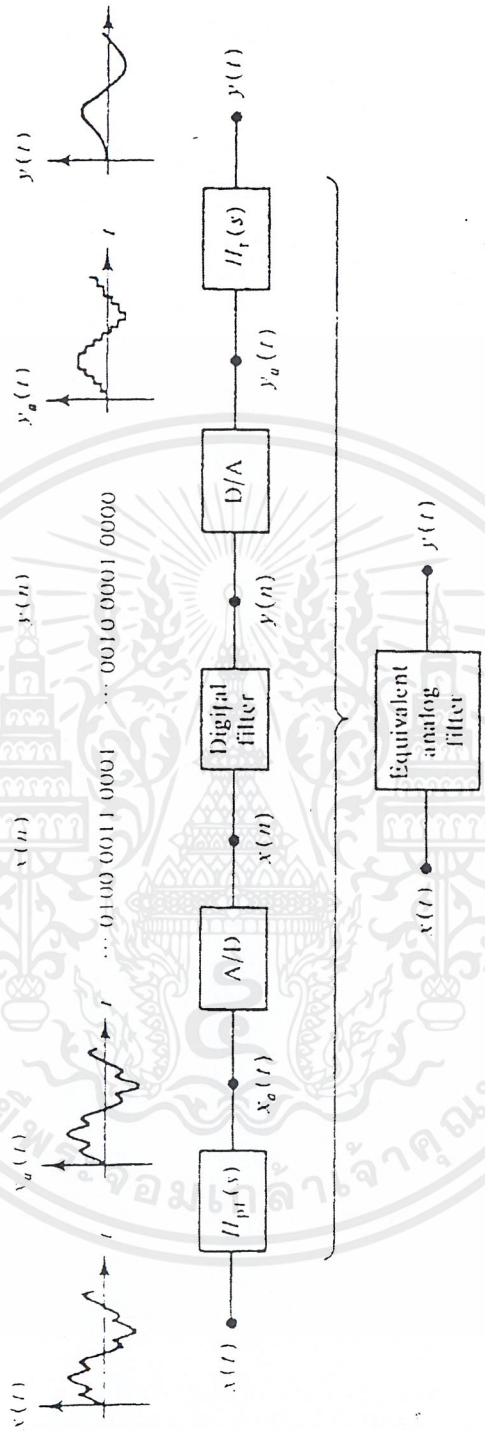
2.6 วงจรกรองดิจิตอลด้วย A/D และ D/A (Digital Filter with A/D and D/A)

วงจรเปลี่ยนสัญญาณ A/D เป็นวิธีการควบคุมให้ชุดรหัสเลขฐานสอง (binary code word) ที่คล้ายกัน เพื่อทำการวิเคราะห์ระดับของสัญญาณอินพุตเวลาต่อเนื่อง (continous-time input signal) ที่เวลานั้น วงจรกรองดิจิตอลได้สื่อถึงสัญลักษณ์ H(z) ให้เป็นขั้นตอนการคำนวณซึ่งทำหน้าที่ผลิตลำดับเอาต์พุต y(n) จากลำดับอินพุต x(n) ดังรูปที่ 2.16



รูปที่ 2.16 การประมวลผลสัญญาณดิจิตอลของสัญญาณแอนะลอก

วงจรเปลี่ยนสัญญาณ D/A จะรับสัญญาณลำดับอินพุตเข้ามาเพื่อทำการผลิตสัญญาณเอาต์พุตต่อเนื่องในรูปของสัญญาณขั้นบันได (staircase form) และรูปแบบของสัญญาณขั้นบันไดนี้จะถูกทำให้เรียบโดยใช้วงจรกรองคืนสัญญาณ (reconstruction filter) เพื่อผลิตสัญญาณเอาต์พุต y(t) ตามที่ต้องการ ดังแสดงไว้ในรูปที่ 2.17

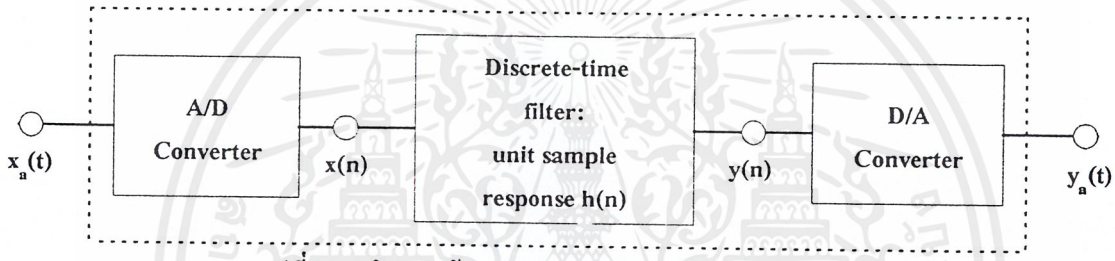


รูปที่ 2.13 แสดงลำดับรูปแบบของการประมวลผลสัญญาณดิจิตอล

เมื่อพิจารณาถึงโครงสร้าง A/D – วงจรกรอง – D/A ซึ่งแสดงไว้ในรูปที่ 2.18 โดยวงจรเปลี่ยน A/D ทั่วไปแล้วนั้นตัวอย่างของสัญญาณที่เข้ามาใหม่ที่ระดับเดียวกันกับเวลาจะแสดงลำดับของ x(n) ได้เป็น

$$x(n) = x_a(t)\big|_{t=nT} = x_a(nT) \tag{2.57}$$

วงจรกรองดิสกรีตไทม์ (discrete-time filter) ได้ทำงานโดยการรับสัญญาณ x(n) เข้ามาใหม่เพื่อทำการส่งลำดับเอาต์พุต y(n) ซึ่งได้ถูกเปลี่ยนเป็นสัญญาณเอาต์พุตต่อเนื่อง $y_a(t)$ ผ่านทางวงจรเปลี่ยนสัญญาณดิจิตอลเป็นแอนะลอก



รูปที่ 2.18 โครงสร้าง A/D – discrete-time filter – D/A

สำหรับจุดมุ่งหมายในส่วนนี้เพื่อหารการตอบสนองความถี่ (frequency response) ของวงจรกรองแอนะลอกสมมูลย์ (equivalent analog filter) ซึ่งแสดงไว้ในเส้นประตามรูปที่ 2.18 ที่ได้ทำการหาสถานะการตอบสนองที่สม่ำเสมอสำหรับระบบโดยรวมเพื่อไซน์นูซอยด์ (sinusoid) ของความถี่ที่กำหนดเป็น Ω_0 ซึ่งน้อยกว่า π/T แสดงได้ดังสมการที่

$$x(n) = A\cos\Omega_0 t \qquad \text{เมื่อ} \quad \Omega_0 < \pi/T \tag{2.58}$$

เมื่อผ่านวงจรเปลี่ยน A/D แล้วเราจะได้ x(n) เป็น

$$x(n) = x_a(nT) = A\cos\Omega_0 nT = A\cos[\overbrace{(\Omega_0 T}^{\omega_0})n] = A\cos\omega_0 n \tag{2.59}$$

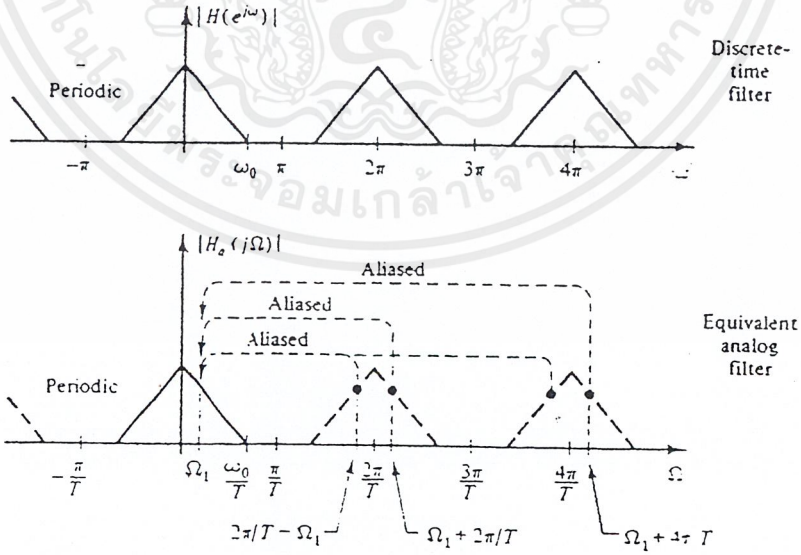
เมื่อ ω_0 เท่ากับ $\Omega_0 T$ ได้ถูกเรียกว่า ความถี่ดิจิตอล ถึงอย่างไรก็ดีหน่วยของมันจะอยู่ใน radians/sample สถานะของเอาต์พุต y(n) ที่ผ่านวงจรกรองดิสกรีตไทม์สามารถเขียนได้เป็น

$$y_{ss}(n) = \left|H(e^{j\omega})\right|_{\omega=\omega_0=\Omega_0 T} \cos[(\Omega_0 T)n + \arg H(e^{j\omega})\,|_{\omega=\omega_0=\Omega_0 T}] \qquad (2.60)$$

สัญญาณเอาต์พุต $y_a(t)$ ที่คืนสถานะได้อย่างสมบูรณ์หลังจากผ่านวงจรเปลี่ยน D/A นั้นสามารถเขียนสมการได้เป็น

$$y_a(t) = A\left|H(e^{j\Omega_0 T})\right| \cos[(\Omega_0 T + \arg H(e^{j\Omega_0 T})] \qquad (2.61)$$

ทั้งนี้ขนาดของแอมปลิจูด (amplitude) พิจารณาได้จาก $\left|H(e^{j\Omega_0 T})\right|$ และเฟส (phase) สามารถถูกเปลี่ยนโดย $\arg H(e^{j\Omega_0 T})$ ด้วยเหตุนี้ แมกนิจูด (magnitude) ได้ถูกกำหนดโดยการคำนวณค่าการตอบสนองความถี่ดิจิตอล $H(e^{j\omega})$ ที่ $\omega = \Omega_0 T$ ในรูปที่ 2.19 แมกนิจูดของการตอบสนองความถี่แอนะลอกสมมูลย์ของโครงสร้าง A/D - H(z) - D/A ซึ่งคล้ายกันกับเฟสของการตอบสนองความถี่ ที่ได้ถูกวาดขึ้นสำหรับวงจรกรองดิสกรีตไทม์ และวงจรกรองแอนะลอกสมมูลย์



รูปที่ 2.19 การตอบสนองความถี่วงจรกรองแอนะลอกสมมูลย์
สำหรับโครงสร้างแบบ A/D – วงจรกรอง – D/A

นอกจากนี้ เมื่อพิจารณาจากรูปที่ 2.16 อีกครั้งจะเห็นว่ามีวงจรกรองมาต่อที่ก่อนหน้าสัญญาณอินพุต ก่อนที่จะผ่าน A/D ก็เพื่อป้องกันความถี่อินพุตที่เกินค่ามาตรฐานการสุ่มสัญญาณ (ความถี่เกิน $f_s/2$) ซึ่งจะก่อให้เกิดการซ้อนเหลื่อม ของสเปคตรัมสัญญาณได้ เรามักจะเรียกการเกิดซ้อนเหลื่อมนี้ว่า "Aliasing" พิจารณาตำแหน่งการถูกซ้อนเหลื่อมได้จากรูปที่ 2.19 และเรียกวงจรกรองที่ป้องกันการเกิดอาการนี้ว่า "Anti-aliasing filter" ดังที่ได้เคยกล่าวมาแล้วในเรื่องของการสุ่มสัญญาณ

บทที่ 3
โครงสร้างบอร์ดประมวลผลสัญญาณดิจิตอล

3.1 โครงสร้างทางฮาร์ดแวร์

3.1.1 บอร์ดประมวลผลสัญญาณดิจิตอล

ปัจจุบันผู้ที่อยู่ในวงการอิเลคทรอนิกส์และคอมพิวเตอร์จะทราบดีว่าได้มีการนำความรู้เรื่องการประมวลสัญญาณดิจิตอล มาใช้กันอย่างแพร่หลายมากขึ้นทุกวัน ในอุปกรณ์อิเลคทรอนิกส์แทบทุกประเภทรวมทั้งในคอมพิวเตอร์ และโทรคมนาคมโดยอาจจะเขียนเป็นซอฟท์แวร์ทั้งหมด หรืออาจจะอยู่ในรูปของวงจรรวม (Integrated Circuit) ที่เรียกเป็นภาษาอังกฤษว่า DSP Chip ผสมกับซอฟท์แวร์ที่เราต้องเขียนขึ้นมาบางส่วน ความสะดวกของการใช้วงจรรวมราคาที่ถูกลงทุกวัน ขณะที่สมรรถนะได้รับการปรับปรุงเพิ่มขึ้นตลอดเวลาเช่นกัน ทำให้เราพบ DSP Chip มากขึ้นในอุปกรณ์ต่างๆ

ในปัจจุบันได้มีผู้นำเข้า TMDS3200031 DSP Starter Kit เรียกย่อว่า DSK ของเท็กซัส อินสตรูเมนท์เข้ามาจำหน่าย เพื่อให้นักศึกษา นักออกแบบ ได้ทดลองศึกษาใช้งาน

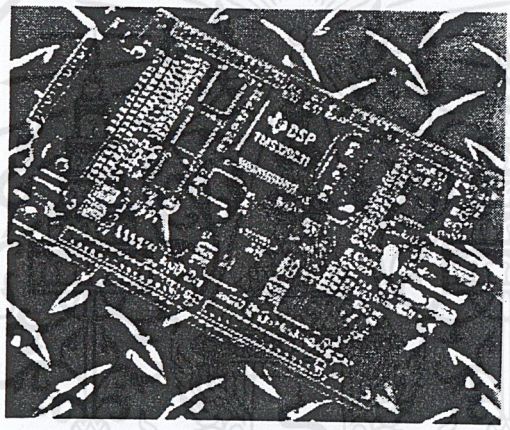
3.1.2 ระบบที่บอร์ด DSK ต้องการ

สำหรับบอร์ดประมวลผลสัญญาณดิจิตอลรุ่นนี้จะใช้ต่อร่วมกับคอมพิวเตอร์ส่วนบุคคลทั่วๆไปซึ่งมีคุณสมบัติขั้นต่ำคือ

- ใช้หน่วยความจำ RAM 640 กิโลไบต์
- เนื้อที่ในฮาร์ดดิสก์ 1.2 เมกะไบต์
- ฟลอปปี้ดิสก์ขนาด 1.44 นิ้ว
- มีพอร์ตขนาน
- ใช้ได้กับจอภาพแบบโมโนโครมหรือแบบจอสีก็ได้
- ใช้กับระบบปฏิบัติการ เอ็มเอสดอส(MS-DOS) หรือ วินโดวส์ (Windows) วินโดวส์ 95 (Windows 95) หรือ โอเอสทู (OS/2)
- สายต่อขนาดเส้นผ่านศูนย์กลาง 2.1 มิลลิเมตร
- ใช้หม้อแปลงป้อนไฟเลี้ยงขนาด 500-1500 มิลลิแอมป์ และแรงดันไฟขนาด 7-12 โวลท์ สำหรับไฟกระแสตรง(DC) และ 6-9 โวลท์สำหรับไฟกระแสสลับ(AC)

3.1.3 รายละเอียดของบอร์ด DSK

บอร์ดประมวลผลสัญญาณดิจิตอลนี้เป็นรุ่นที่ใช้ชิป C3x DSK มีความสามารถในการประมวลผลทางจุดทศนิยม (floating point) ที่ความถี่แซมเปิล 20 กิโลเฮิร์ต และชิป TLC32040 เป็นวงจรเชื่อมต่อทางแอนะลอกซึ่งมีความสามารถในการตรวจสอบข้อผิดพลาดของโปรแกรม (Debugger) ข้อมูลแอนะลอกทางด้านอินพุตและเอาต์พุต จะผ่านชิป TLC32040 โดยความถี่แซมเปิลของสัญญาณแอนาลอกจะเป็นอิมพัลซ์ขนาด 14 บิต ที่ความเที่ยงตรง 20 กิโลเฮิร์ต โดยผ่านสวิตช์คาปาซิเตอร์แอนติอลาสซิงฟิลเตอร์ (antialiasing filter) ซึ่งสวิตช์คาปาซิเตอร์นี้ใช้กรองสัญญาณความถี่ต่ำทางเอาต์พุตของสัญญาณดิจิตอล สำหรับสัญญาณแอนะลอกทางด้านอินพุตและเอาต์พุตจะส่งผ่านโดยใช้มาตรฐานการเชื่อมต่อแจ็ค (jack) แบบ RCA รูปร่างของบอร์ด DSK ดังแสดงในรูปที่ 3.1



รูปที่ 3.1 รูปร่างจริงของบอร์ด DSP

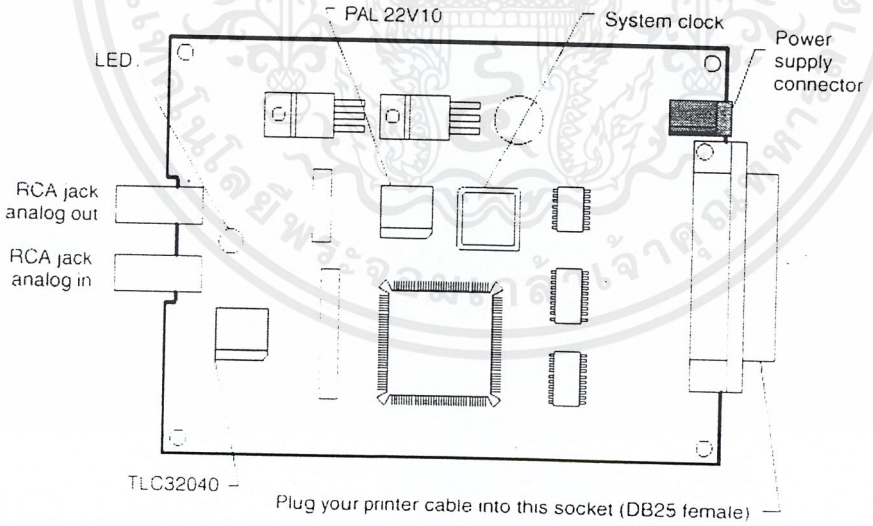
องค์ประกอบทางฮาร์ดแวร์ของบอร์ด (Hardware Component Overview)

อุปกรณ์พื้นฐานที่เกี่ยวข้องกับบอร์ด DSK มีดังต่อไปนี้

- ออสซิลเลเตอร์ (Oscillator) สำหรับบนบอร์ด DSP นี้จะได้ความถี่ที่ 50 MHz เพื่อขับสัญญาณนาฬิกาอินพุต (clock input) ของ C31
- หัวต่อพอร์ตปริ๊นเตอร์ขนาน (Parallel printer port connector) หัวต่อ DB25 ขนาด 25 หัวเข็ม ซึ่งก็ จะได้ต่อตรงสู่สายพอร์ตปริ๊นเตอร์ขนาน
- แจ็ค RCA (RCA jacks) แจ็ค RCA นั้นเป็นแจ็คที่เป็นทางผ่านสัญญาณเข้าสู่หัวเข็มทั้งอินพุตและเอาต์พุตของ AIC และเป็นทั้งตัวส่งผ่านสัญญาณแอนะลอกอินพุตและเอาต์พุต
- TLC 32040 AIC เป็นวงจรการเชื่อมต่อสัญญาณแอนะลอกที่เตรียมขึ้นเพื่อให้ C31 ได้เข้าถึงสัญญาณแอนะลอก AIC จะทำการเปลี่ยนข้อมูลแอนะลอกเข้าสู่ดิจิตอลโดยการวิเคราะห์

ของ C31 และการทำงานของ C31 นี้ข้อมูลดิจิตอลก็จะทำการคืนกลับของข้อมูลเข้าสู่ AIC เพื่อเปลี่ยนเป็นสัญญาณแอนะลอกได้เช่นกัน

- TMS 320c31 เป็นตัวประมวลผลหลักขนาด 32 บิท และประมวลผลสัญญาณดิจิตอลทางจุดทศนิยม ทั้งนี้สามารถที่จะพัฒนามาตรฐานของการประยุกต์ใช้งาน และทำการบรรจุ (load) ลงบนหน่วยความจำของชิพ c31 ได้
- อุปกรณ์ควบคุมแรงดัน (Voltage Regulator) DSK ใช้แรงดันไฟตรงขนาด 7-12 โวลต์ หรือแรงดันไฟสลับขนาด 6-9 โวลต์ โดยแหล่งจ่ายแรงดันไฟตรงขนาด 7-12 โวลต์ ได้ถูกปรับกระแสให้เป็นรูปคลื่นเต็มคลื่นได้ นอกจากนั้นปรับแรงดันขึ้นไปถึง 5 โวลต์ โดย LMT7805 และก็ได้เปลี่ยนแรงดันเป็น –5 โวลต์ได้โดยใช้ วงจรสวิทช์ชิ่งคาปาร์ซิเตอร์ด้วย LT1054 และทำการปรับแต่งโดยใช้ LM7905 สำหรับแรงดันไฟสลับ 6-9 โวลต์ ได้ถูกปรับกระแสให้เป็นรูปคลื่นเต็มคลื่น และทำการปรับแต่งโดยใช้ LM7805 และ LM7905 ไปเป็นแรงดัน +5 โวลต์ และ –5 โวลต์ ตามลำดับ แหล่งจ่ายแรงดัน +5 โวลต์ และ –5 โวลต์ จะเป็นกำลังงานทั้งหมดสำหรับการใช้งานวงจรบอร์ด DSK ทั้งนี้ TLC32040AIC นั้นจะเป็นตัวกำหนดแหล่งจ่ายแรงดัน –5 โวลต์



รูปที่ 3.2 แสดงการต่อสายพอร์ตขนาน และหม้อแปลงเข้าสู่บอร์ด DSK

3.1.4 การต่อบอร์ด DSK ร่วมกับคอมพิวเตอร์ส่วนบุคคล

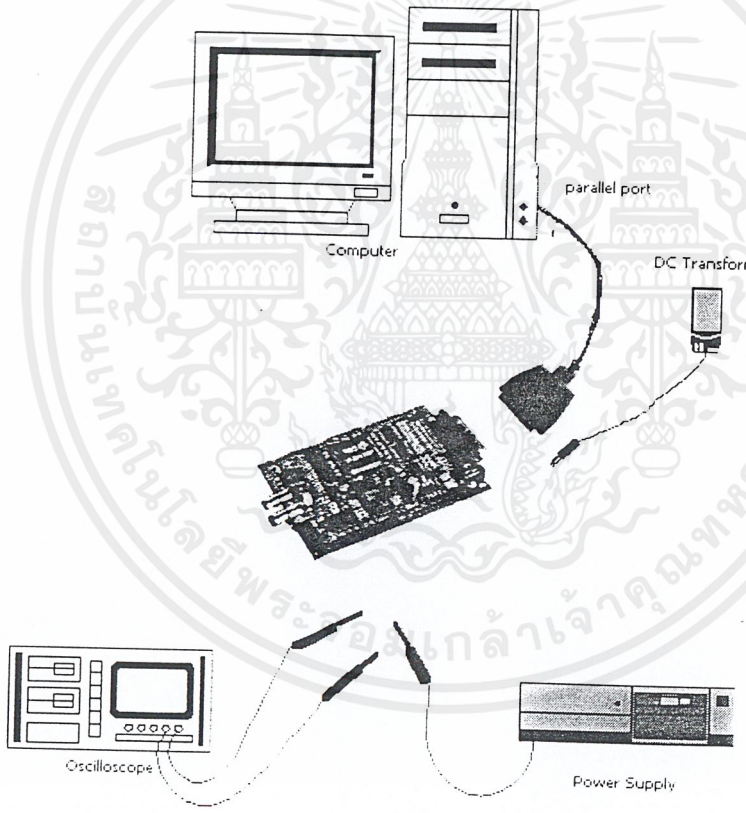
สำหรับบอร์ดประมวลผลสัญญาณดิจิตอลรุ่นนี้จะใช้ต่อร่วมกับคอมพิวเตอร์ส่วนบุคคลโดยต่อผ่านทางพอร์ตขนานประกอบด้วยอุปกรณ์ดังนี้

1 สายพอร์ตปริ๊นเตอร์

2 หม้อแปลงไฟกระแสตรงขนาด 7-12 โวลท์

3 เครื่องกำเนิดสัญญาณ(function generator) ใช้ป้อนสัญญาณเข้าอินพุต

4 ออสซิลโลสโคป ใช้วัดสัญญาณ



รูปที่ 3.3 แสดงการต่องานจริงของบอร์ด DSK

3.1.5 เกี่ยวกับชิพ C3x

การผลิต TMS320C3x ของตัวประมวลผลสัญญาณดิจิตอล เป็นการออกแบบ CMOS 32 บิท ด้วยการประมวลผลชนิดจุดทศนิยม ที่มีสมรรถภาพสูงในตระกูลของ TMS320 ในรูปแบบของ DSP ชิพเดี่ยว

การผลิต C3xที่ได้รวมระบบของการควบคุม และฟังก์ชั่นทางคณิตศาสตร์เข้าไว้ด้วยกันบนการควบคุมของชิพเพียงตัวเดียว จะมีผลทางด้านการประมวลผลตัวเลขที่ความเร็วสูง ทำให้ง่ายต่อการดำเนินงานของข้อมูล โครงสร้างภายในที่มีสมรรถนะสูงของ DSP นี้ ได้ออกแบบไว้ด้วยความเร็วและความยืดหยุ่นของการทำงานที่ 60 ล้านจุดทศนิยมต่อหนึ่งหน่วยวินาที (MFLOPS)

3.2 โครงสร้างทางซอฟท์แวร์

ในโครงงานนี้ซอฟแวร์ของบอร์ด DSP ที่ถูกนำมาใช้งาน จะแบ่งออกเป็น 2 ส่วน ได้แก่

1. ส่วนของซอฟแวร์ที่ใช้ทำหน้าที่ในการติดต่อกับบอร์ด DSP ซึ่งจะใช้ภาษาแอสเซมบลี้ (assembly) ในการเขียนโปรแกรม

2. ส่วนของซอฟแวร์ที่ใช้ ทำหน้าที่ในการแสดงผลออกทางหน้าจอคอมพิวเตอร์ ซึ่งจะใช้ภาษาซี (Borland C++ Version 3.1 for Dos) โดยจะทำการเชื่อมโยง(Link) กับภาษาแอสแซมบลี้ (assembly)

ลำดับขั้นตอนการต่อบอร์ด DSK กับคอมพิวเตอร์ส่วนบุคคล

ขั้นตอนที่ 1 การต่อ DSK เข้ากับคอมพิวเตอร์ส่วนบุคคล

1. ปิดแหล่งจ่ายไฟที่เครื่องคอมพิวเตอร์
2. ต่อสายพอร์ตปริ้นเตอร์ขนานเข้าสู่พอร์ตขนานที่เครื่องคอมพิวเตอร์
3. ปลั๊ก (plug) ของสายพอร์ตปริ้นเตอร์ขนานถูกต่อเข้าสู่หัวต่อ DSK DB25
4. จ่ายไฟเลี้ยงขนาด 7-12 Vdc หรือ 6-9 Vac ให้กับบอร์ด
5. เปิดแหล่งจ่ายไฟให้กับเครื่องคอมพิวเตอร์
6. ไฟที่หลอด LED ของบอร์ดจะให้แสงสว่างเป็นสีแดง หรือ สีเขียว

ขั้นตอนที่ 2 การติดตั้งซอฟแวร์ของ DSK

1. ทำการติดตั้ง(Install)ไฟล์ข้อมูลจากแผ่นซีดีรอมของบอร์ด
2. โดยสร้างไดเรคทอรี่ (directory) ที่ชื่อdsktools และไดเรคทอรี่นี้จะบรรจุซอฟแวร์แอสเซมเบอร์ (assembler) และดีบักเกอร์ (debugger)

ขั้นตอนที่ 3 การดัดแปลง Path Statement

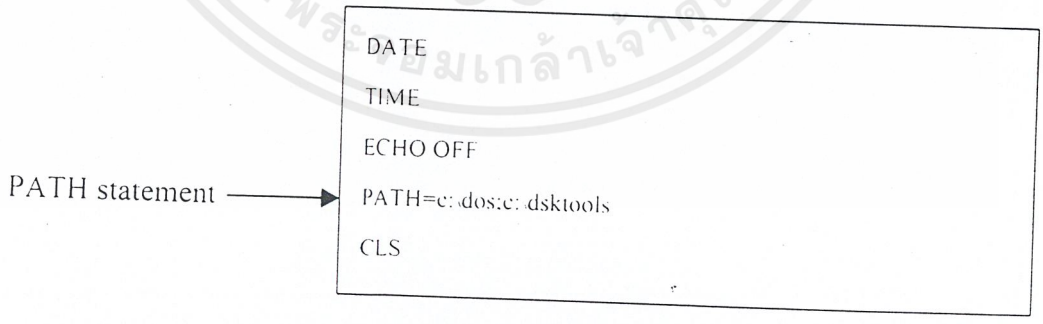
เพื่อรับรองว่าดีบักเกอร์และแอสเซมเบอร์ได้ถูกเรียกจากไดเรคทอรี่ในเครื่องคอมพิวเตอร์ จึงต้องทำการดัดแปลง Path Statement เพื่อระบุไดเรคทอรี่ของ dsktools สำหรับกรณีนี้แล้วต้องไม่ทำก่อนการเรียกดีบักเกอร์เพียงแค่ครั้งแรกเท่านั้นแต่จะต้องทำทุกครั้งที่ทำการเปิดเครื่องคอมพิวเตอร์

ทั้งนี้จะได้ผลเป็นอย่างดีโดยการเข้าเฉพาะคำสั่ง DOS แต่ก็ไม่ซับซ้อนที่จะวางคำสั่งในไฟล์ autoexec.bat ของระบบเครื่องคอมพิวเตอร์ รูปแบบทั่วไปของการทำแสดงได้ดังนี้

PATH=C:\dsktools;*pathname2*;*pathname3*

กรณีนี้จะยอมรับการเรียกดีบักเกอร์โดยปราศจากการระบุชื่อของไดเรคทอรี่ซึ่งบรรจุไฟล์ทำงานดีบักเกอร์

ถ้าเกิดแก้ไขไฟล์ autoexec.bat และมี PATH statement เป็นที่เรียบร้อยแล้ว ในช่วงท้ายจะแสดงด้วย c:\dsktools ดังรูปที่ 3.3



รูปที่ 3.4 การติดตั้งคำสั่ง DOS สำหรับสภาพแวดล้อมของ DSK

ถ้าทำการแก้ไขไฟล์ autoexec.bat แล้ว การเรียกไฟล์นี้จะเข้าโดย

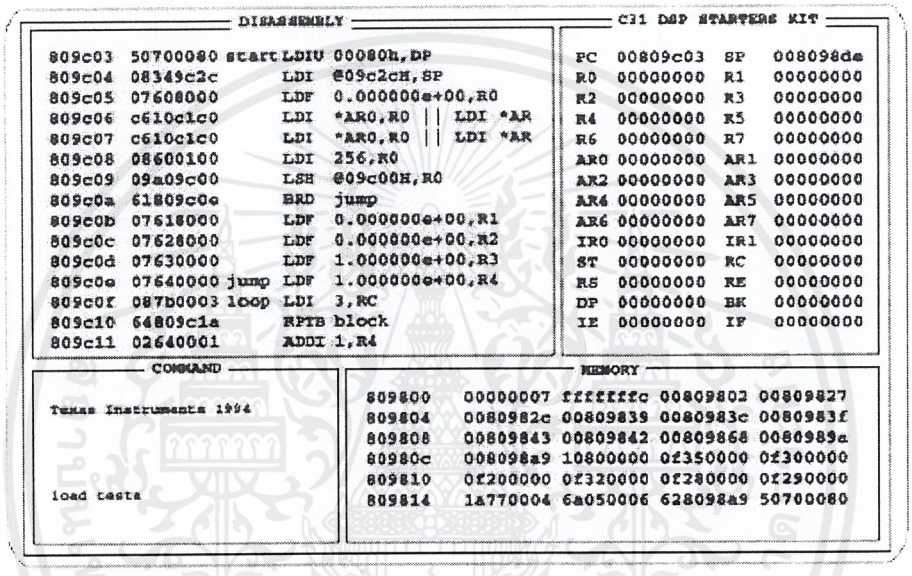
c:\> autoexec

ขั้นตอนที่ 4 ความถูกต้องของการติดตั้ง

เพื่อรับรองว่าได้ทำการติดตั้งบอร์ด DSK แอสเซมเบอร์ และดีบักเกอร์อย่างถูกต้อง ให้ใช้คำสั่งตามข้างล่างนี้เพื่อทำการเริ่มต้นดีบักเกอร์ของ DSK

c:\ > dsk3d

หลังจากการเข้าคำสั่ง dsk3d ก็จะพบการแสดงผลดังรูปที่ 3.5



รูปที่ 3.5 การแสดงผลดีบักเกอร์เบื้องต้น

การเขียนโปรแกรม

ก่อนอื่นต้องทำการติดตั้งโปรแกรมเหล่านี้ก่อนคือ

1 Dsk Tools ซึ่งจะประกอบด้วย ดีบักเกอร์(Debugger) และ คอมไพเลอร์ (Complier) ของแอสเซมบลี้ โดยที่ดีบักเกอร์จะทำหน้าที่ในการตรวจสอบโปรแกรมที่เขียนโดยภาษาแอสเซมบลี้ของชิป C3x ว่าสามารถทำงานได้ถูกต้องและเป็นไปตามต้องการหรือไม่

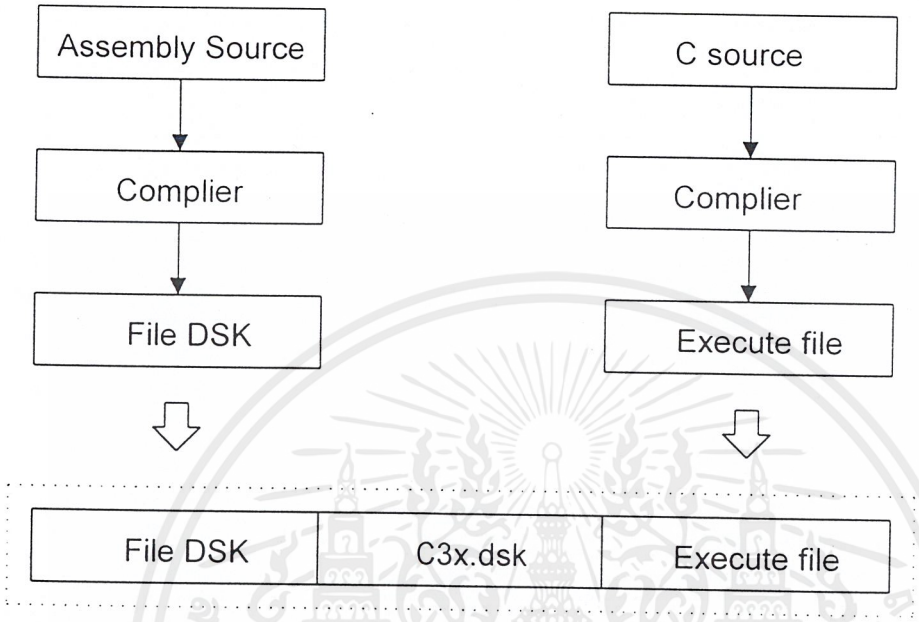
ส่วนคอมไพเลอร์จะทำหน้าที่ในการแปลงซอร์ดโคดของแอสเซมบลี้ (Assembly source) ให้เป็นไฟล์ DSK ซึ่งจะเป็นส่วนที่ทำให้สามารถติดต่อกับบอร์ดได้

2 ภาษาซีจะเป็นส่วนที่ทำหน้าที่ในการแสดงผลทางหน้าจอซึ่จะประกอบไปด้วยไฟล์ต่างๆ ดังนี้

SPECTRUM.CPP	ใช้ในการแสดงผลกราฟฟิกทางหน้าจอคอมพิวเตอร์
DRIVER.CPP	ใช้ในการควบคุมการต่อพอร์ตขนานของบอร์ด DSP
TARGET.CPP	จะทำหน้าที่ถ่ายเทข้อมูลระหว่างคอมพิวเตอร์ และ ตัวประมวลผล C31 เข้าสู่โครงสร้าง Packet (packet มีหน้าที่ในการส่งคำสั่ง ไปที่ C31
OBJECT.CPP	ใช้รูทีน ทาร์เกท-เลเวล และรูทีนไดร์เวอร์-เล่เวล (target - and driver -level0 Routine) เพื่อกำหนดค่าเบื้องต้นและทำการโหลดโปรแกรมเข้าสู่ชิป C31
DSK_COFF.CPP	ประกอบด้วยตัวโหลด ไฟล์ DSK และ ไฟล์ COFF (Common Object File Format)
ERRORMSG.CPP	ประกอบด้วยลำดับของข้อความพร้อมกับฟังก์ชั่นการส่งกลับ
SYMBOLS.CPP	ประกอบด้วยรูทีนที่สนับสนุนตารางสัญลักษณ์
TEXTWIN.CPP	DOS level text window functions

ซึ่งไฟล์ต่างๆเหล่านี้จะใช้งานร่วมกันรวมอยู่ใน ไฟล์โปรเจ็ก

แสดงบล็อกไดอะแกรมขั้นตอนการพัฒนาโปรแกรม



รูปที่ 3.6 ขั้นตอนการเขียนโปรแกรม

จากบล็อกไดอะแกรมจะแสดงให้เห็นว่าในการเขียนโปรแกรมขึ้นมาเพื่อให้บอร์ดสามารถทำงานได้สำหรับโครงงานนี้ จะต้องเขียนโปรแกรมขึ้นมา 2 ส่วน คือ

ส่วนของการควบคุมบอร์ด จะต้องเขียนแอสเซมบลี้ขึ้นมาแล้วทำการคอมไพล์ โดยใช้ตัวคอมไพล์เลอร์ dsk3a.exe ซึ่งเป็นตัวคอมไพเลอร์ของตัวประมวลผล C3x จะทำให้ได้ไฟล์ dsk ซึ่งเป็นตัวที่ทำหน้าที่ในการติดต่อกับบอร์ด

ส่วนของการแสดงผลทางหน้าจอคอมพิวเตอร์จะใช้ภาษาซี ซึ่งซอร์ดโค้ดเป็นไฟล์โปรเจ็คคือเป็นไฟล์ของภาษาซี ที่รวมกันเป็นไฟล์เดียวโดยแต่ละไฟล์จะมีหน้าที่ต่างๆ ดังที่กล่าวมาข้างต้น

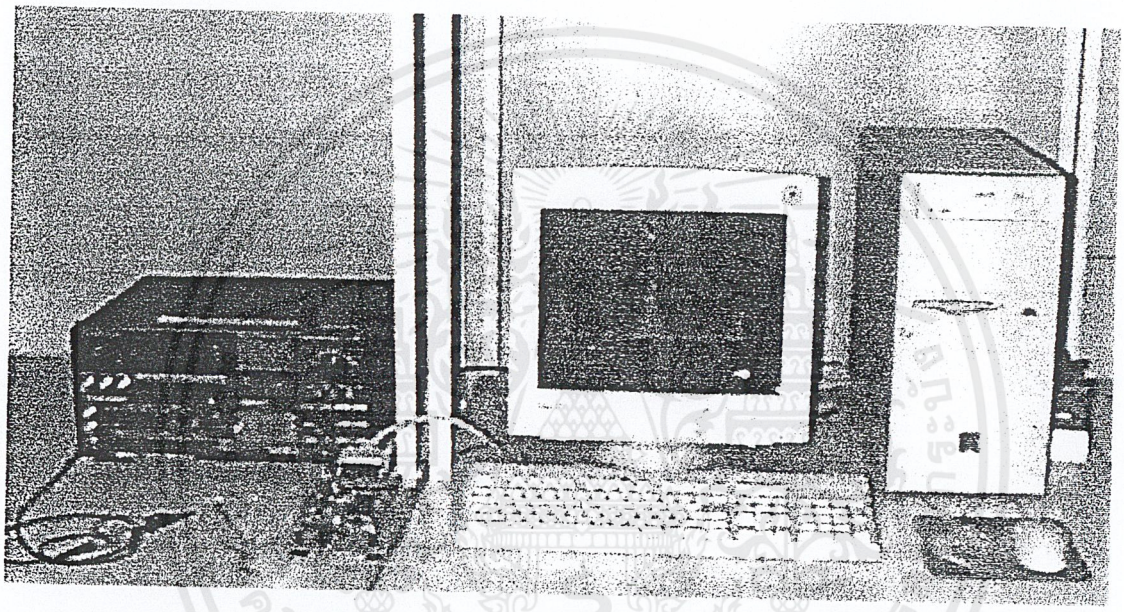
ในการที่จะให้บอร์ดสามารถทำงานได้จะต้องประกอบไฟล์ 3 ไฟล์ ดังรูป 3.6

บทที่4

การทดลองและผลการทดลอง

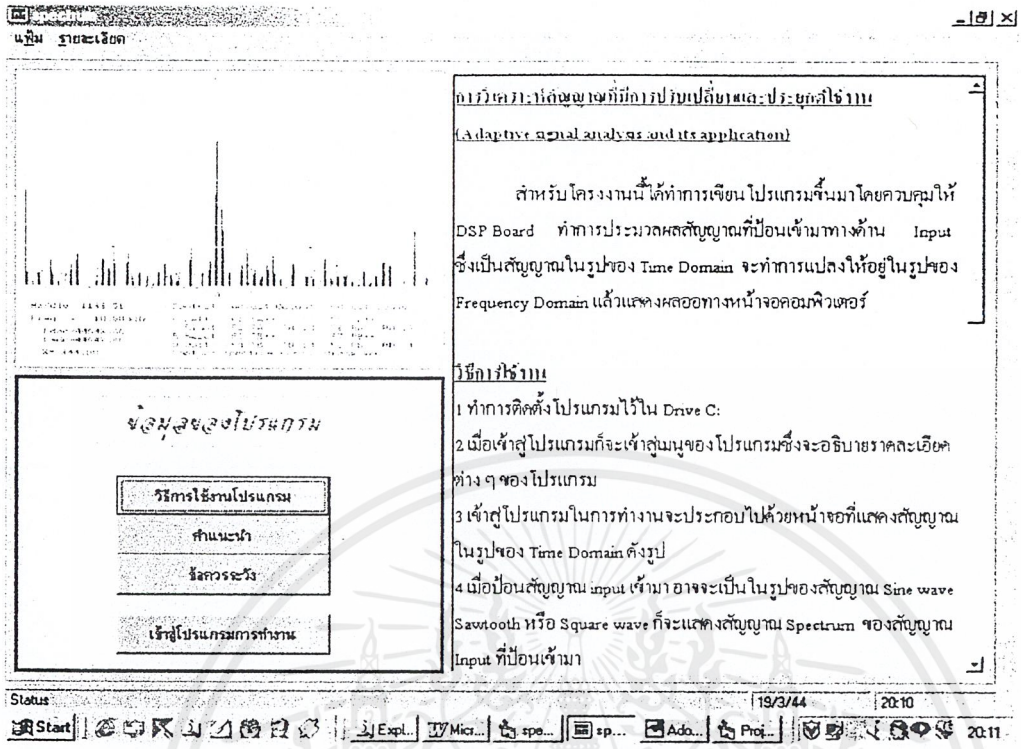
4.1 วิธีการใช้งานโปรแกรม

1. การต่อบอร์ดประมวลผลสัญญาณดิจิตอลเข้ากับคอมพิวเตอร์ ซึ่งบอร์ดจะต่อเข้าทางพอร์ตขนานของเครื่องคอมพิวเตอร์



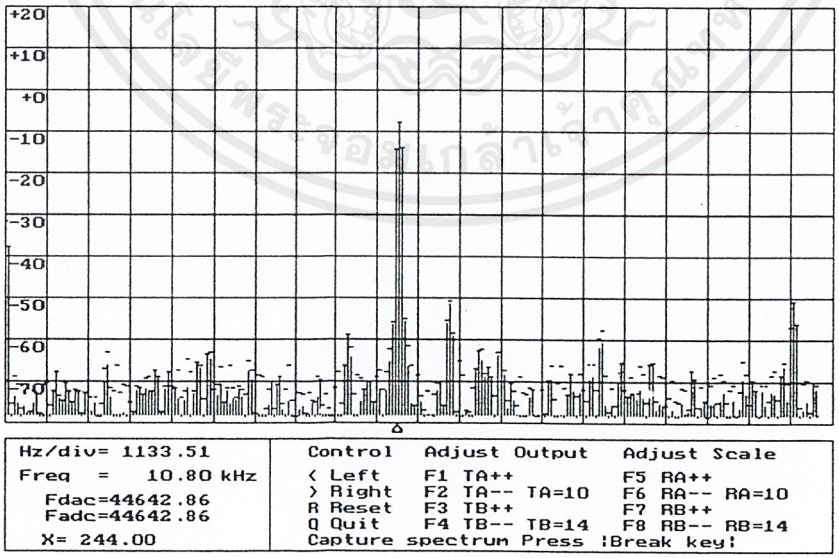
รูปที่ 4.1 แสดงการต่อบอร์ดประมวลผลสัญญาณดิจิตอลเข้ากับคอมพิวเตอร์

2. การติดตั้งโปรแกรม ลงในไดเรกทอรี่ C:\Spectrum
3. เข้าสู่โปรแกรมการใช้งาน โดยเข้าไปที่ Spectrum.exe ก็จะเข้าสู่หน้าต่างดังรูปซึ่งจะแสดงข้อมูลของโปรแกรม รวมถึงวิธีการใช้งานโปรแกรม



รูปที่ 4.2 แสดงโปรแกรม

4. เข้าสู่หน้าต่างของการวัดสัญญาณสเปกตรัมโดยคลิกที่ ปุ่มเข้าสู่โปรแกรมการทำงาน และเมื่อป้อนสัญญาณอินพุตเข้าไปที่บอร์ดก็จะปรากฏสัญญาณดังรูปที่ 4.3

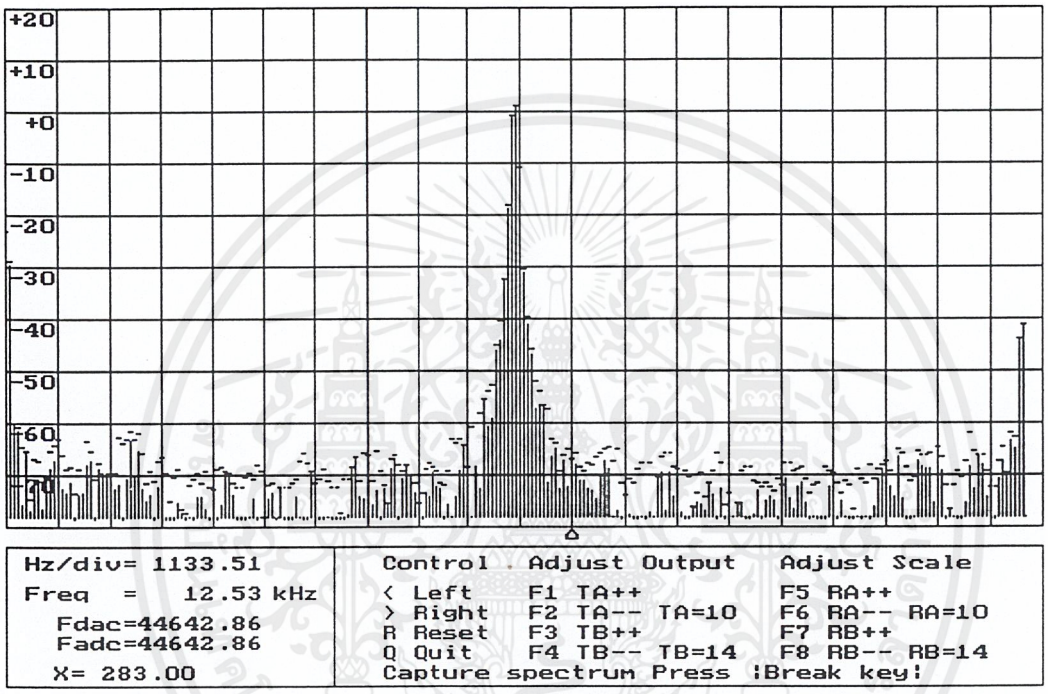


รูปที่ 4.3 แสดงการทำงานของโปรแกรมในการแสดงสัญญาณในโดเมนความถี่

4.2 การทดสอบโปรแกรม

4.2.1 การทดลองที่ 1

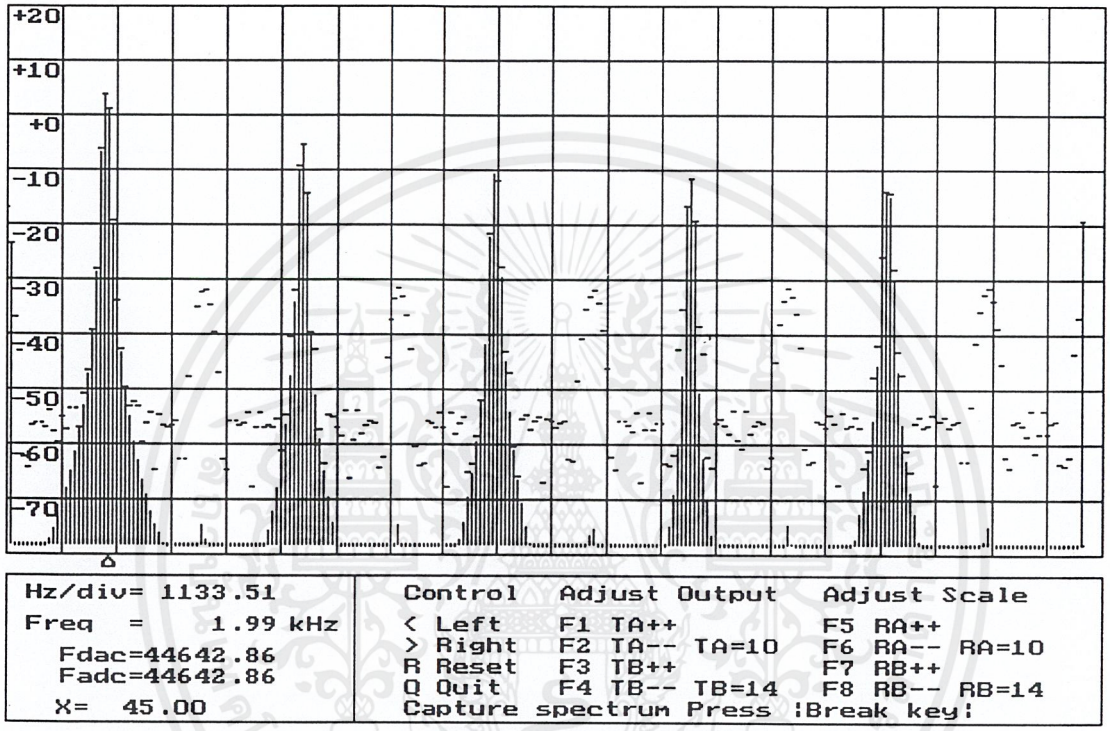
ทำการทดลองป้อนสัญญาณอินพุทเป็น sinewave เมื่อวัดสัญญาณในโดเมนความถี่ออกมาจะได้ดังรูป



รูปที่ 4.4 แสดงสเปคตรัมของสัญญาณรูปคลื่นซายน์

4.2.2 การทดลองที่ 2

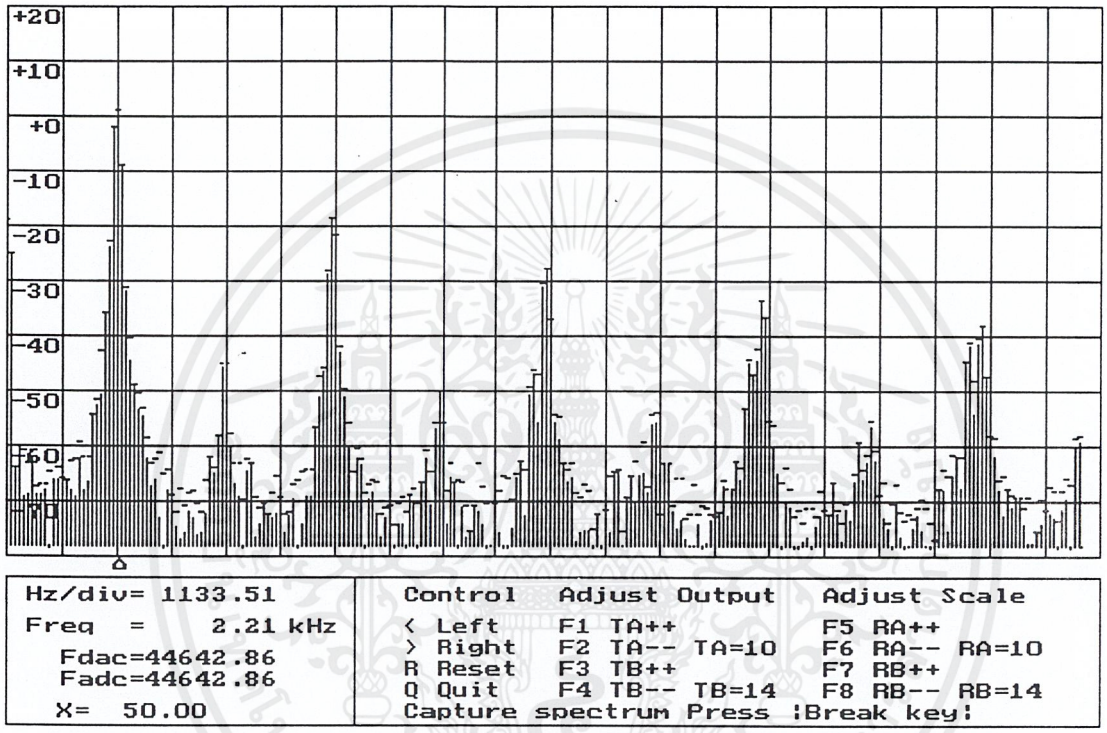
ทำการทดลองป้อนสัญญาณอินพุทเป็น squarewave เมื่อวัดสัญญาณในโดเมนความถี่ออกมาจะได้ดังรูป



รูปที่ 4.5 แสดงสเปคตรัมของสัญญาณรูปคลื่นสี่เหลี่ยม

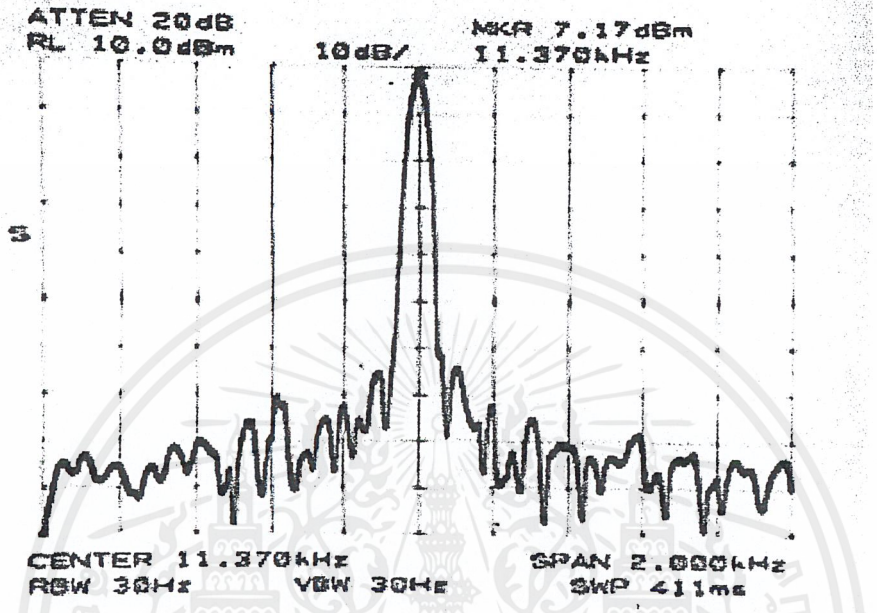
4.2.3 การทดลองที่ 3

ทำการทดลองป้อนสัญญาณอินพุทเป็น triangle wave เมื่อวัดสัญญาณในโดเมนความถี่ออกมาจะได้ดังรูป



รูปที่ 4.6 แสดงสเปคตรัมของสัญญาณรูปคลื่นสามเหลี่ยม

สัญญาณสเปกตรัมของสัญญาณรูปคลื่นซายน์ โดยวัดจากเครื่องสเปกตรัมอนาไลย์เซอร์ (Spectrum Analyser) แสดงดังรูป



รูปที่ 4.7 แสดงสเปคตรัมของสัญญาณที่ได้วัดจากเครื่องสเปกตรัม

บทที่ 5
บทสรุปและแนวทางในการพัฒนา

ปริญญานิพนธ์ฉบับนี้ เป็นการนำเสนอการวิเคราะห์สัญญาณที่มีการปรับเปลี่ยน และประยุกต์ใช้งาน โดยผ่านการออกแบบและควบคุมการทำงานของ DSP board ซึ่งจะทำการเปลี่ยนแปลงสัญญาณอินพุตที่เข้ามาในรูปของโดเมนเวลาเป็นโดเมนความถี่ของสัญญาณอินพุตต่างๆ

ทั้งนี้สามารถนำทฤษฎีต่างๆ มาประยุกต์ใช้กับบอร์ด DSP ได้ดังนี้

- จากทฤษฎีของอนุกรมฟูเรียส์สามารถนำมาใช้แสดงสัญญาณเป็นคาบโดยให้อยู่ในรูปการรวมสัญญาณรูปคลื่นไซน์จำนวนอนันต์เข้าด้วยกัน
- จากทฤษฎีการแปลงฟูเรียส์ จะใช้วิเคราะห์สัญญาณที่ไม่เป็นคาบ ผลลัพธ์จากการวิเคราะห์จะให้สเปคตรัม หรือแถบความถี่ของสัญญาณที่ถูกวิเคราะห์ทำให้รู้ว่าสัญญาณนี้ประกอบด้วยความถี่อะไรบ้าง
- จากทฤษฎีของการสุ่มสัญญาณจะใช้ในการสุ่มสัญญาณอินพุตที่เข้ามา โดยตามทฤษฎีของการสุ่มสัญญาณแล้ว จะได้ว่า $f_s \geq 2f_{in}$ ซึ่งบอร์ด DSP ใช้งานนี้มีความถี่การสุ่มสัญญาณที่ 20 KHz นั่นหมายความว่า ความถี่อินพุตควรมีค่าไม่เกิน 10 KHz เพราะถ้าเกินค่านี้ จะทำให้เกิดการซ้อนเหลื่อมของสัญญาณได้

จากการทดลองเป็นการเปรียบเทียบสัญญาณต่างๆ ที่เข้ามา โดยใช้บอร์ด DSP เป็นตัววิเคราะห์ในโดเมนความถี่เปรียบเทียบกับ spectrum analyzer เพื่อศึกษาถึงความเหมือนและความแตกต่างระหว่างการวิเคราะห์สัญญาณจากบอร์ด DSP กับ spectrum analyzer

จากการทดลองที่ 1

เมื่อสัญญาณอินพุตเป็นรูปคลื่นซายน์ ตามทฤษฎีแล้วสเปกตรัมของสัญญาณซายน์จะมีเพียงความถี่หลักเท่านั้น (Fundamental Frequency) ไม่มีความถี่ฮาโมนิก (Hamonic Frequency)

จากการทดลองที่ 2

เมื่อสัญญาณอินพุตเป็นรูปคลื่นสี่เหลี่ยม ตามทฤษฎีแล้วสเปกตรัมของสัญญาณจะมี ความถี่หลัก (Fundamental Frequency) และมีความถี่ฮาโมนิค (Hamonic Frequency) โดยความถี่ฮาโมนิคลำดับที่ 2 จะเกิดขึ้นเป็นฮาโมนิคคี่

จากการทดลองที่ 3

เมื่อสัญญาณอินพุตเป็นรูปคลื่นสามเหลี่ยมตามทฤษฎีแล้วสเปกตรัมของสัญญาณจะมีความถี่หลักและมีฮาความถี่ฮาโมนิคเกิดขึ้นเป็น สเปคตรัมของฮาร์โมนิคคี่

ปัญหาในการทำโครงงานนี้

1. เนื่องจากคุณสมบัติในการประมวลผลของบอร์ดรุ่นนี้ มีความสามารถในการวิเคราะห์สัญญาณความถี่ที่ต่ำ ดังนั้นจึงแสดงผลของสัญญาณที่จะวิเคราะห์ได้ไม่เกิน 10 KHz
2. การเขียนโปรแกรมในการควบคุมบอร์ด และแสดงผลบนจอคอมพิวเตอร์นั้น มีความซับซ้อนมาก ทำให้สิ้นเปลืองเวลาในการทำความเข้าใจและศึกษาโปรแกรม

แนวทางในการพัฒนา

การเขียนโปรแกรมในการควบคุมบอร์ด DSP นี้ สามารถปรับปรุงเปลี่ยนแปลงได้ตามลักษณะการใช้งาน ดังนั้นการศึกษาเพิ่มเติมในส่วนของโปรแกรมทั้งในด้านแอสเซมบลี้ และภาษา C ก็จะทำให้การใช้งานของบอร์ดมีประสิทธิภาพเพิ่มมากขึ้น ซึ่งผู้จัดทำเชื่อว่าเมื่อมีการพัฒนาแก้ไขข้อบกพร่องต่างๆ แล้ว จะทำให้โครงงานนี้สมบูรณ์แบบ และพร้อมที่จะนำไปใช้งานจริงได้ดียิ่งขึ้น

เอกสารอ้างอิง

1. อรรถสิทธิ์ หล่าสกุล "Digital Signal Processing" ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

2. F.R. Connor "Signals second Edition" Arnold.

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

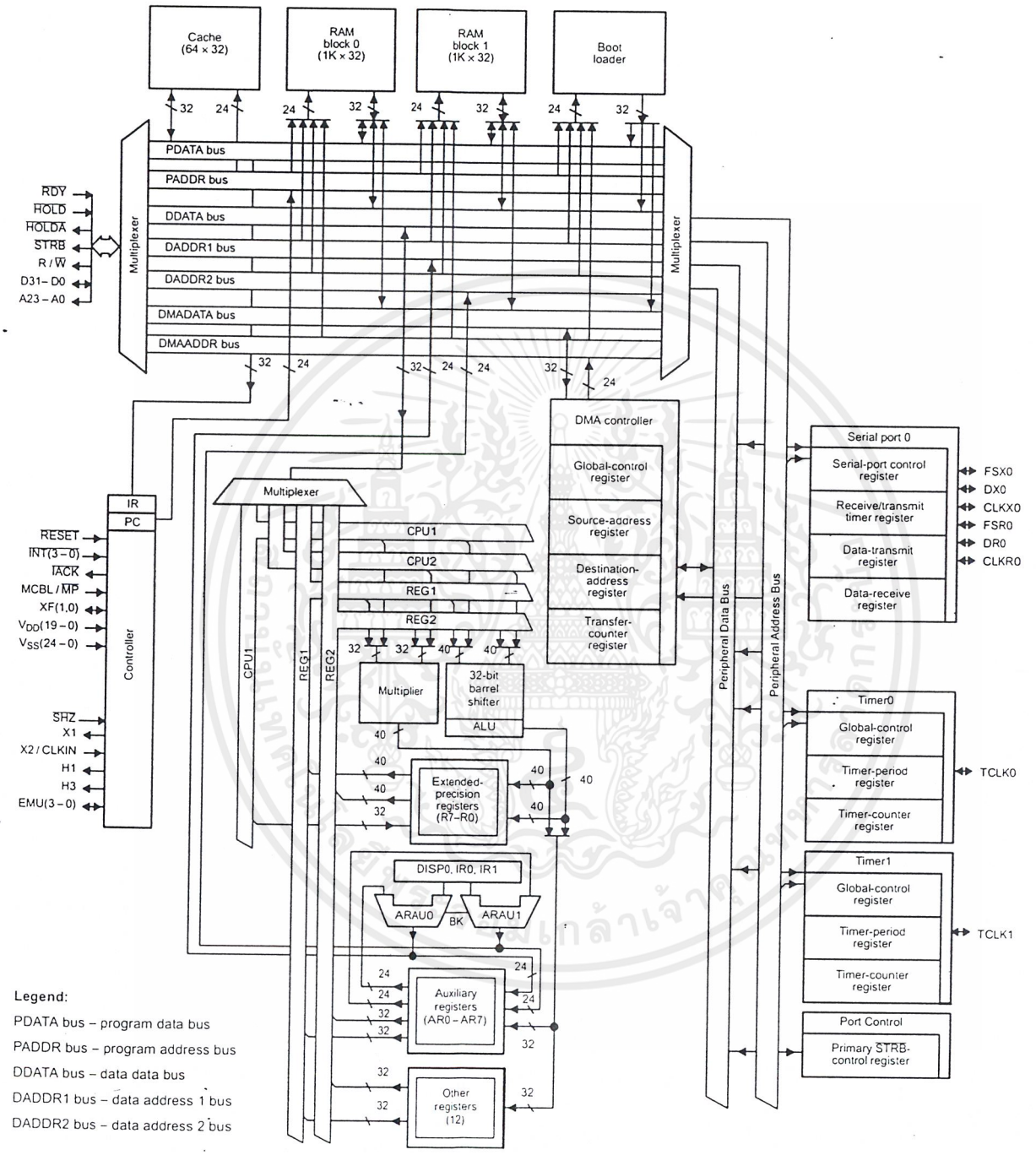
4. Texas Instrument. "TMS320C3x User'sGuide".1997.

5. Texas Instrument. "TMS320C3x DSP Starter Kit User'sGuide".1996.

ภาคผนวก

ก. แสดงวงจรในส่วนต่างๆ

Figure 2–2. TMS320C31 Block Diagram



Legend:

PDATA bus – program data bus

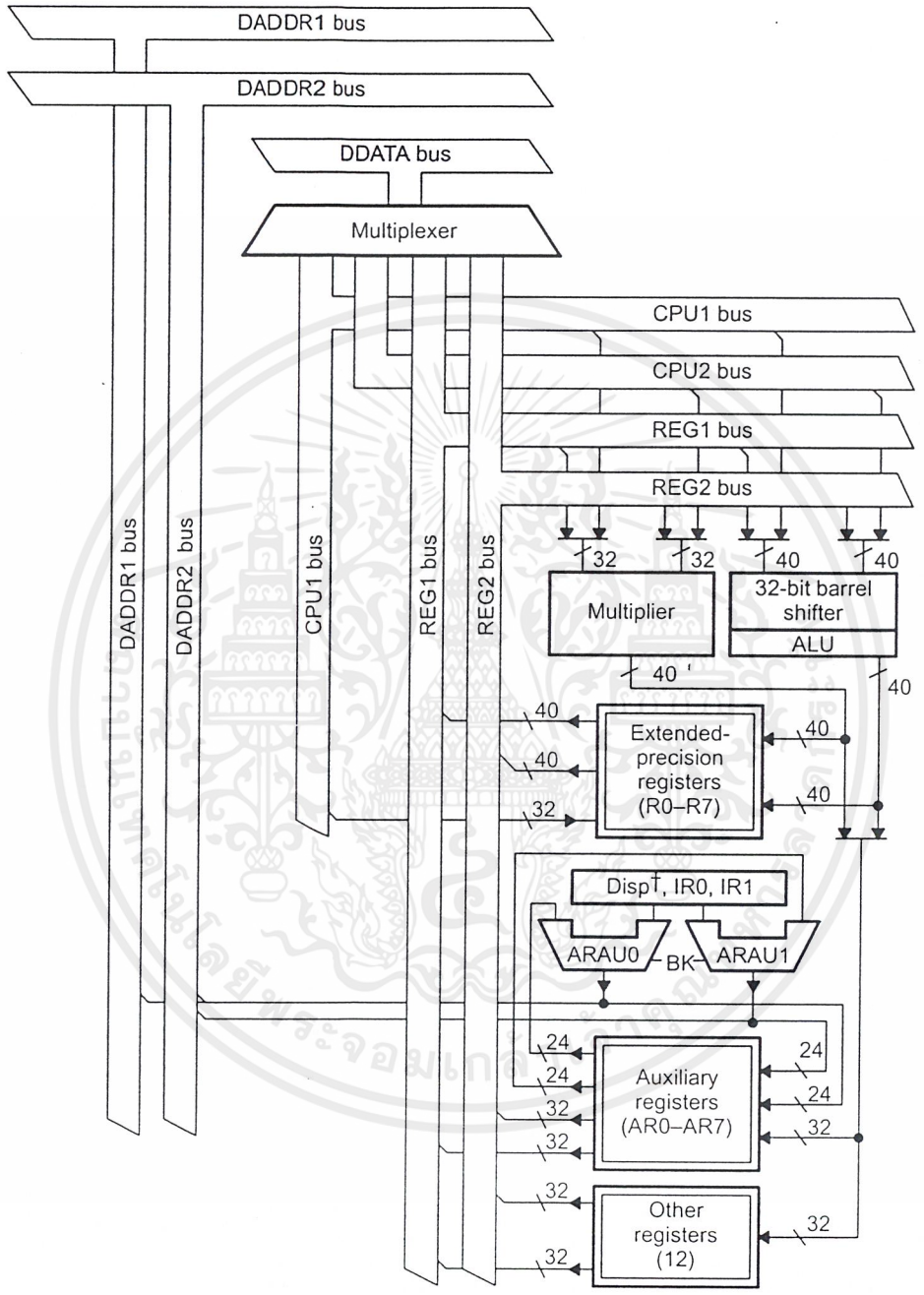
PADDR bus – program address bus

DDATA bus – data data bus

DADDR1 bus – data address 1 bus

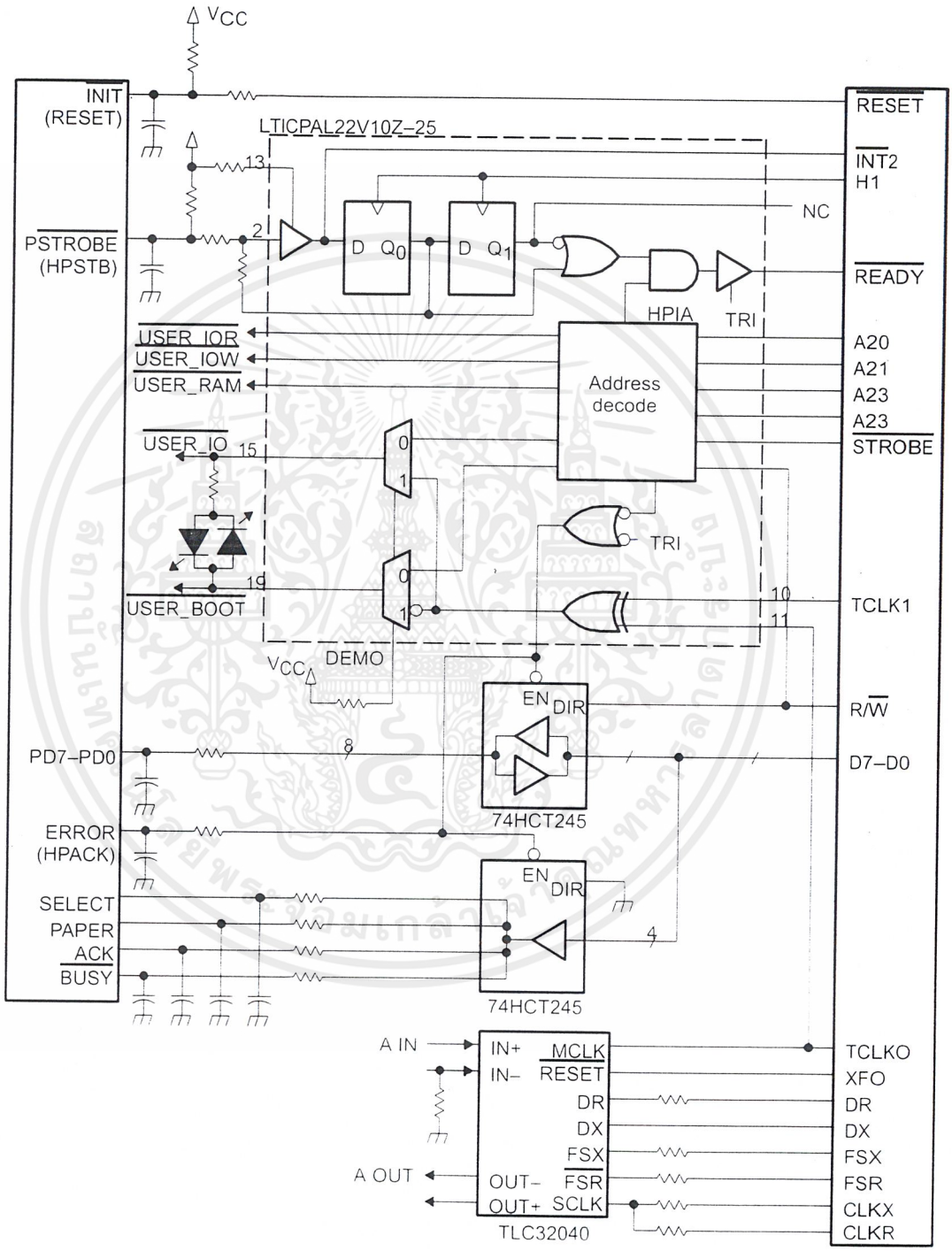
DADDR2 bus – data address 2 bus

Figure 2–4. Central Processing Unit (CPU)

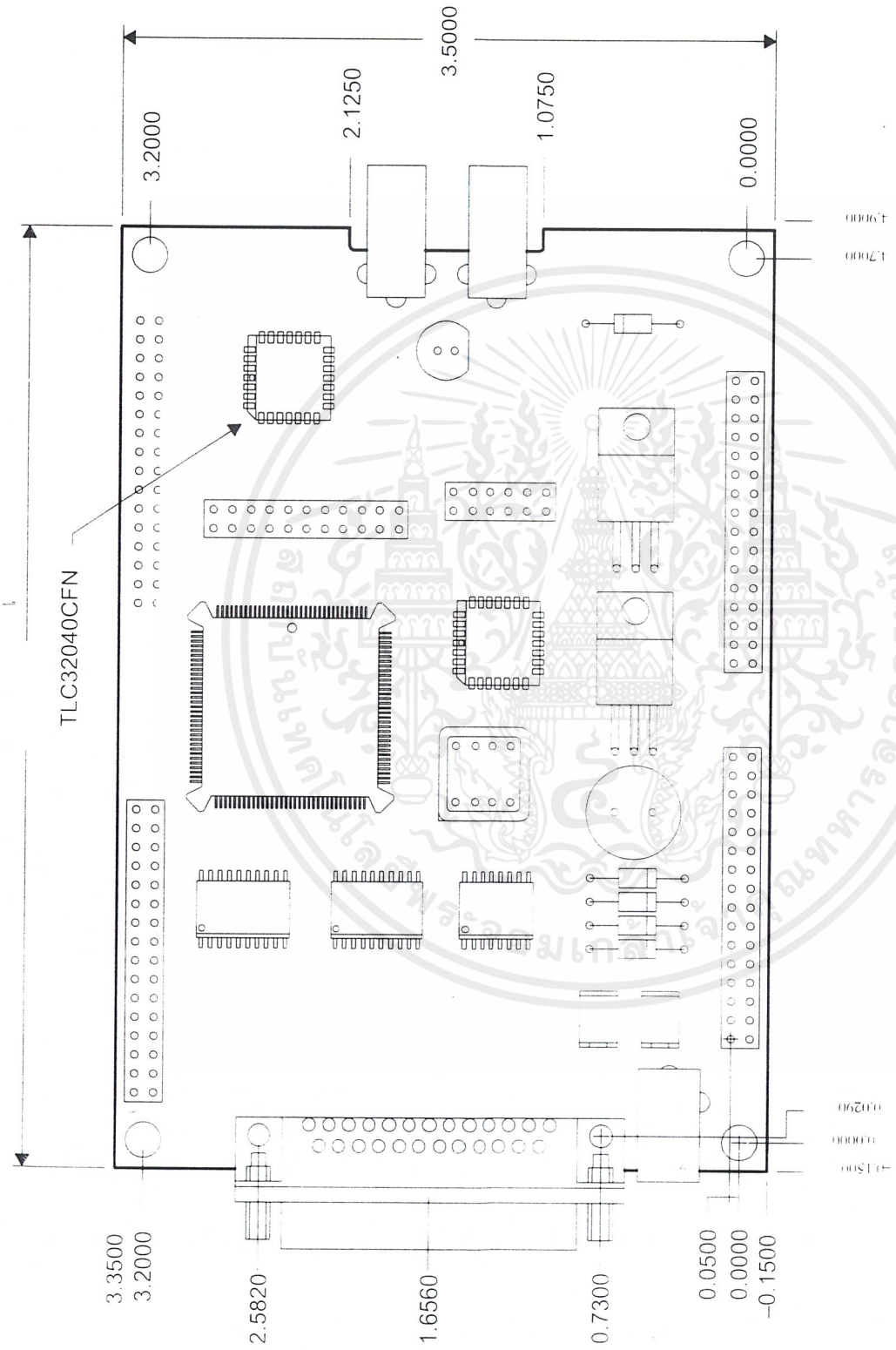


† Disp = an 8-bit integer displacement carried in a program-control instruction

Figure 4–1. TMS320C3x DSK Functional Circuit Diagram



ircuit Board Dimensions



NOTES, UNLESS OTHERWISE SPECIFIED:

1. VCC IS APPLIED TO PIN 8 OF ALL 8-PIN IC'S, PIN 14 OF ALL 14-PIN IC'S, PIN 16 OF ALL 16-PIN IC'S, PIN 20 OF ALL 20-PIN IC'S, ETC.
2. GROUND IS APPLIED TO PIN 4 OF ALL 8-PIN IC'S, PIN 7 OF ALL 14-PIN IC'S, PIN 8 OF ALL 16-PIN IC'S, PIN 10 OF ALL 20-PIN IC'S, ETC.
3. RESISTANCE VALUES ARE IN OHMS.
4. CAPACITANCE VALUES ARE IN MICROFARADS.
5. HIGHEST REFERENCE DESIGNATOR USED:

A. CAPACITORS	C57
B. RESISTORS	R55
C. DIODES	D6
D. IC'S	U9
E. PORTS	P1
F. HEADERS	JP6
G. CONNECTORS	J3

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED
A	ECN563640(E) M. DANG 3-5-96		J. CLARK
	FORMAL RELEASE		

REVISION STATUS OF SHEETS									
REV									
SH									
REV									
SH								D600335	7257
REV	A	A	A	A	A	A		NEXT ASSY	USED ON
SH	1	2	3	4	5	6		APPLICATION	

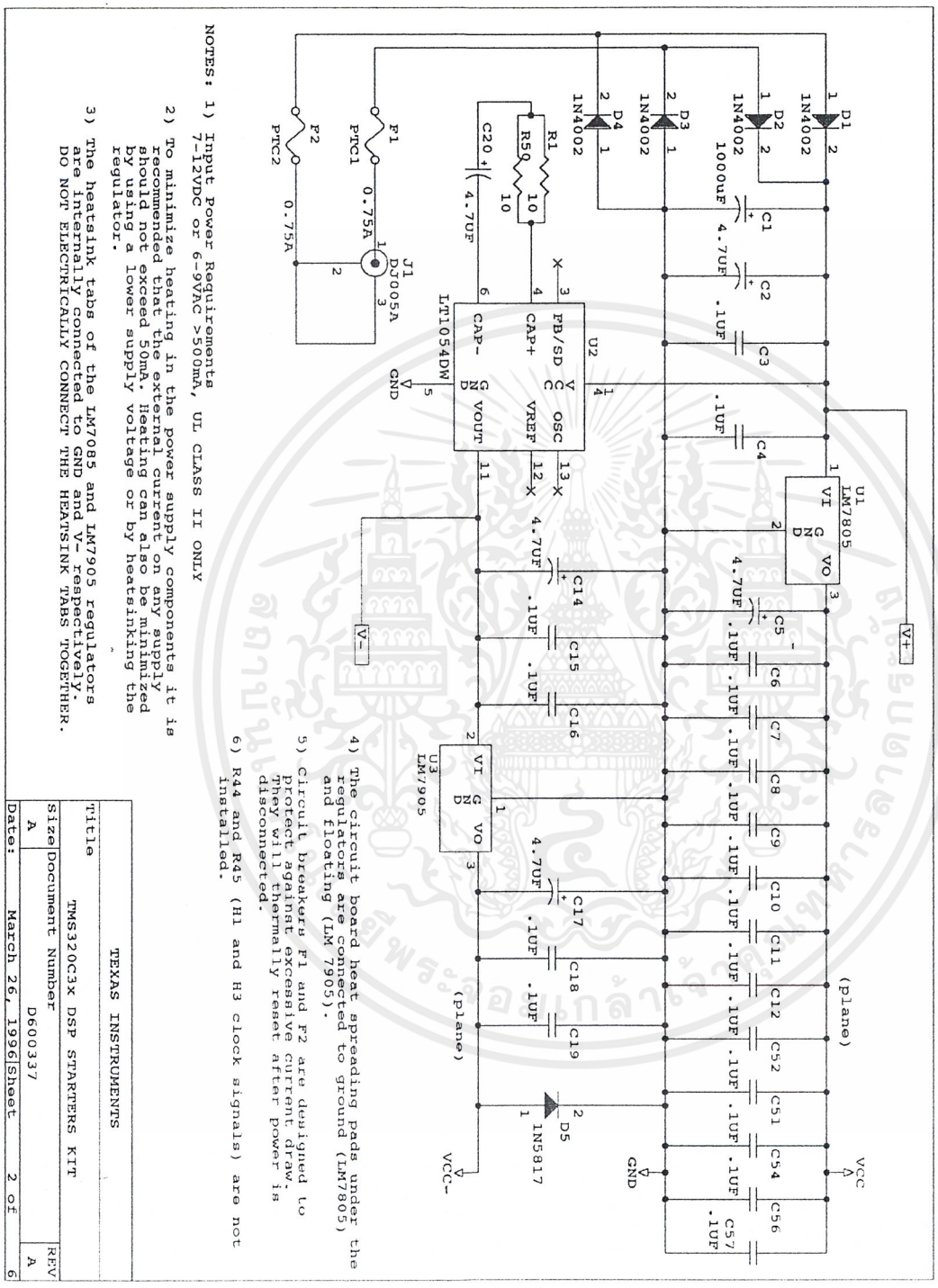
DWN M. DANG	DATE 2-19-96
CHK M. DAWKINS	DATE
ENGR K. LARSON	DATE
ENGR-MGR T. COOMES	DATE
QA J. WHISONANT	DATE
MFG M. JACKSON	DATE
RLSE J. CLARK	DATE

TEXAS INSTRUMENTS
SOFTWARE DEVELOPMENT SYSTEMS
SEMICONDUCTOR GROUP
HOUSTON, TEXAS

Title: TMS320C3x DSP STARTERS KIT

Size	Document Number	REV
A	D600337	A

Date: March 7, 1996 Sheet 1 of 6



NOTES: 1) Input Power Requirements
7-12VDC or 6-9VAC >500mA, UL CLASS II ONLY

2) To minimize heating in the power supply components it is recommended that the external current on any supply should not exceed 50mA. Heating can also be minimized by using a lower supply voltage or by heatsinking the regulator.

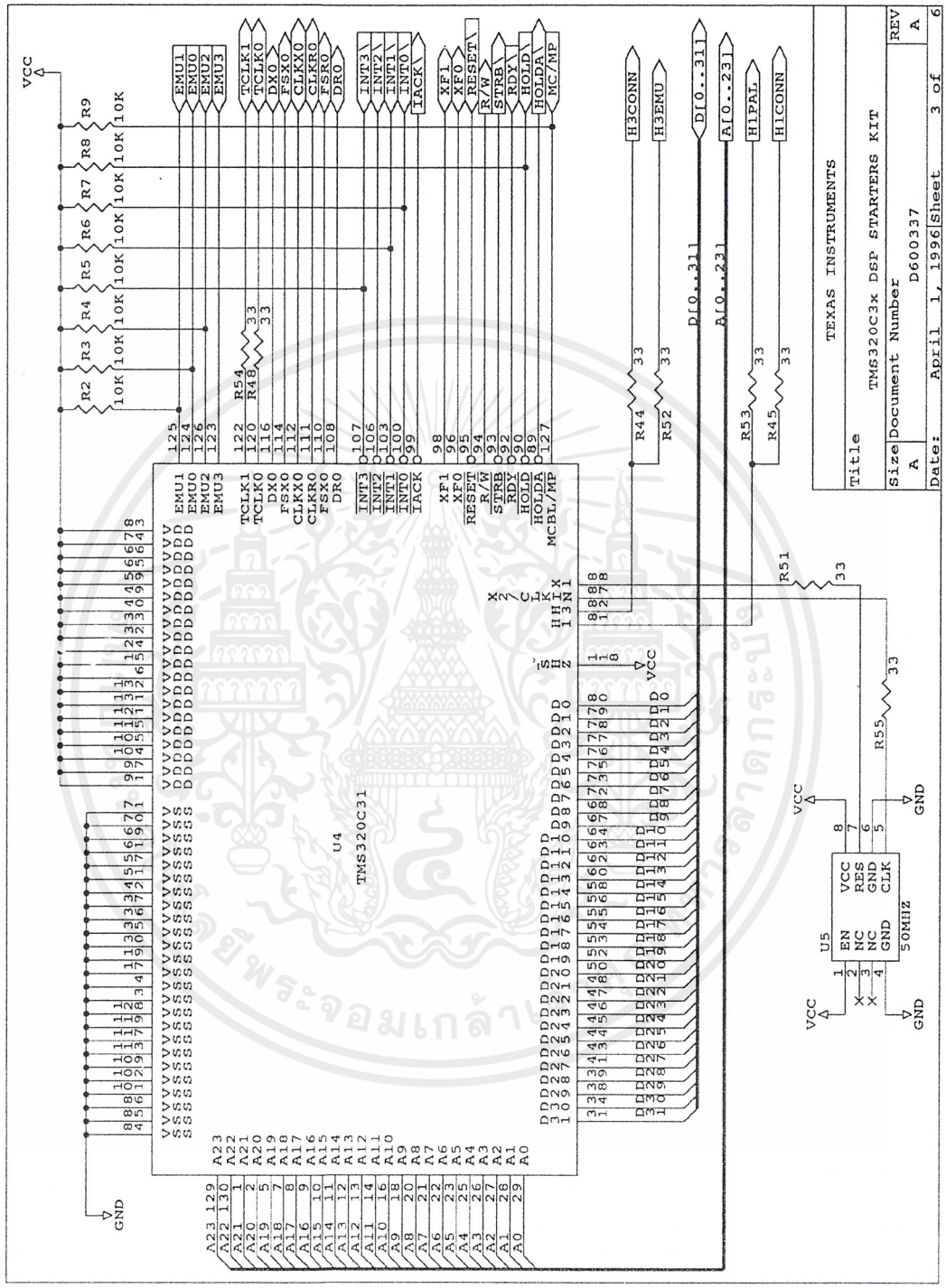
3) The heatsink tabs of the LM7085 and LM7905 regulators are internally connected to GND and V- respectively. DO NOT ELECTRICALLY CONNECT THE HEATSINK TABS TOGETHER.

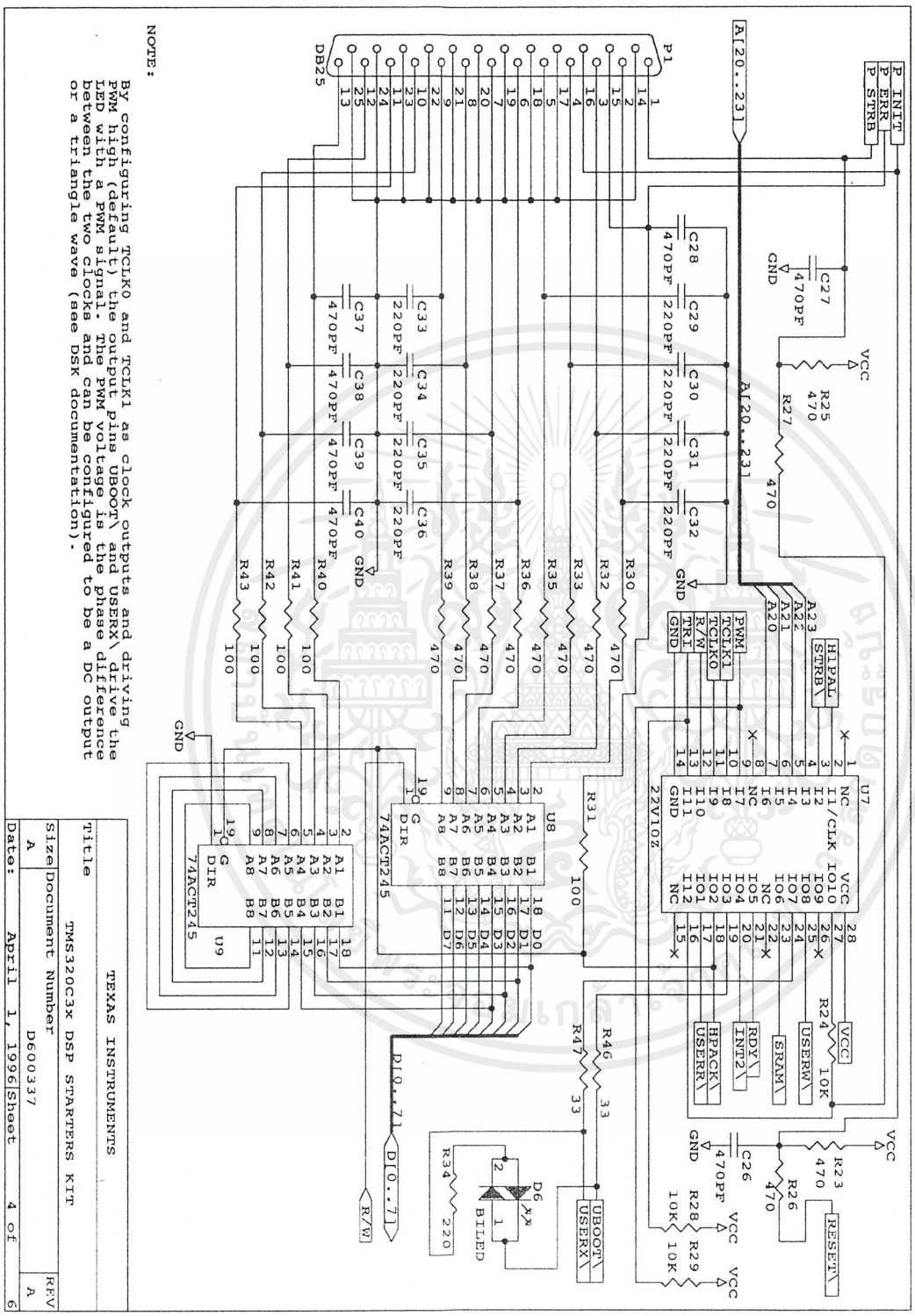
4) The circuit board heat spreading pads under the regulators are connected to ground (LM7805) and floating (LM 7905).

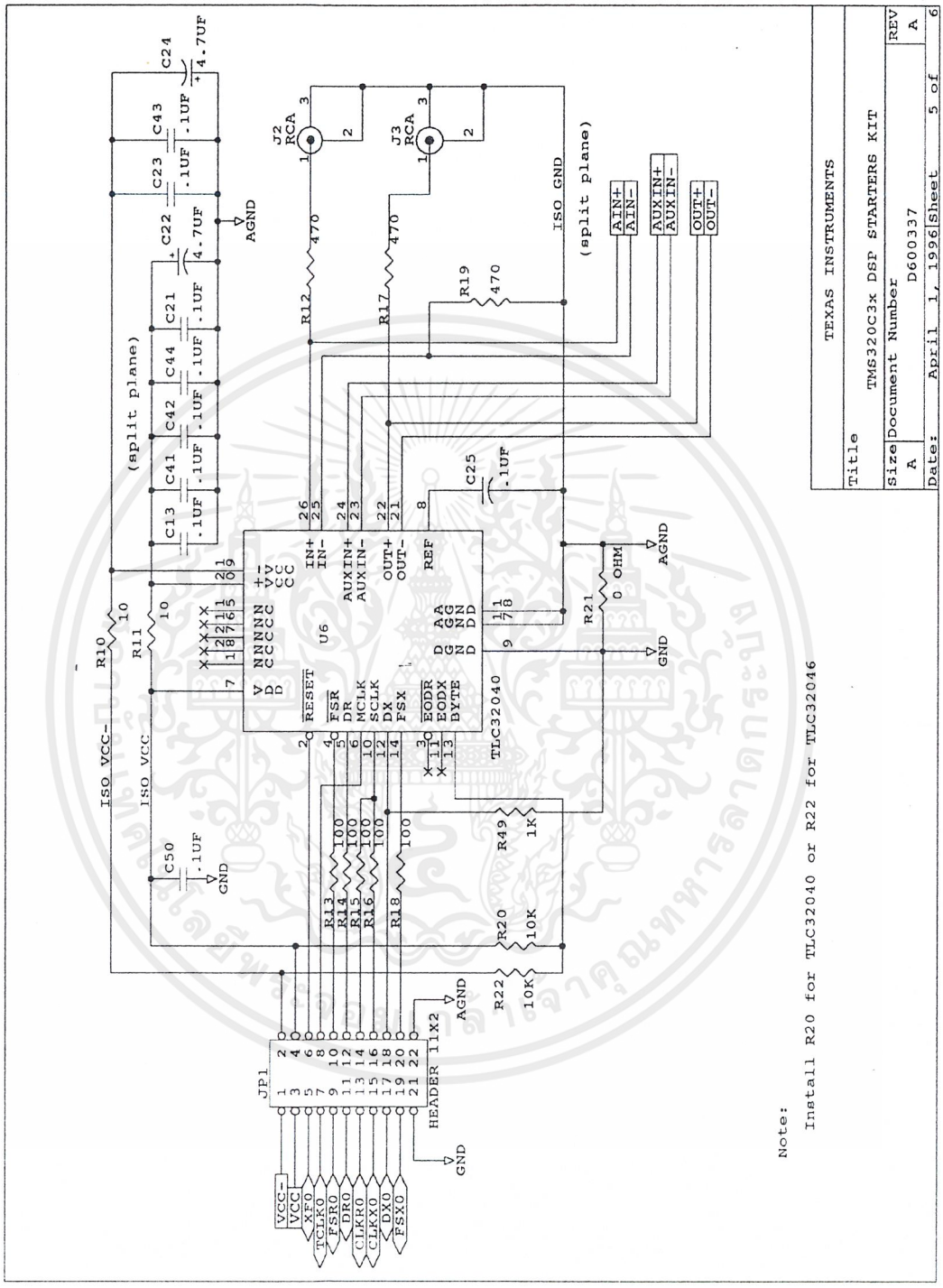
5) Circuit breakers F1 and F2 are designed to protect against excessive current draw. They will thermally reset after power is disconnected.

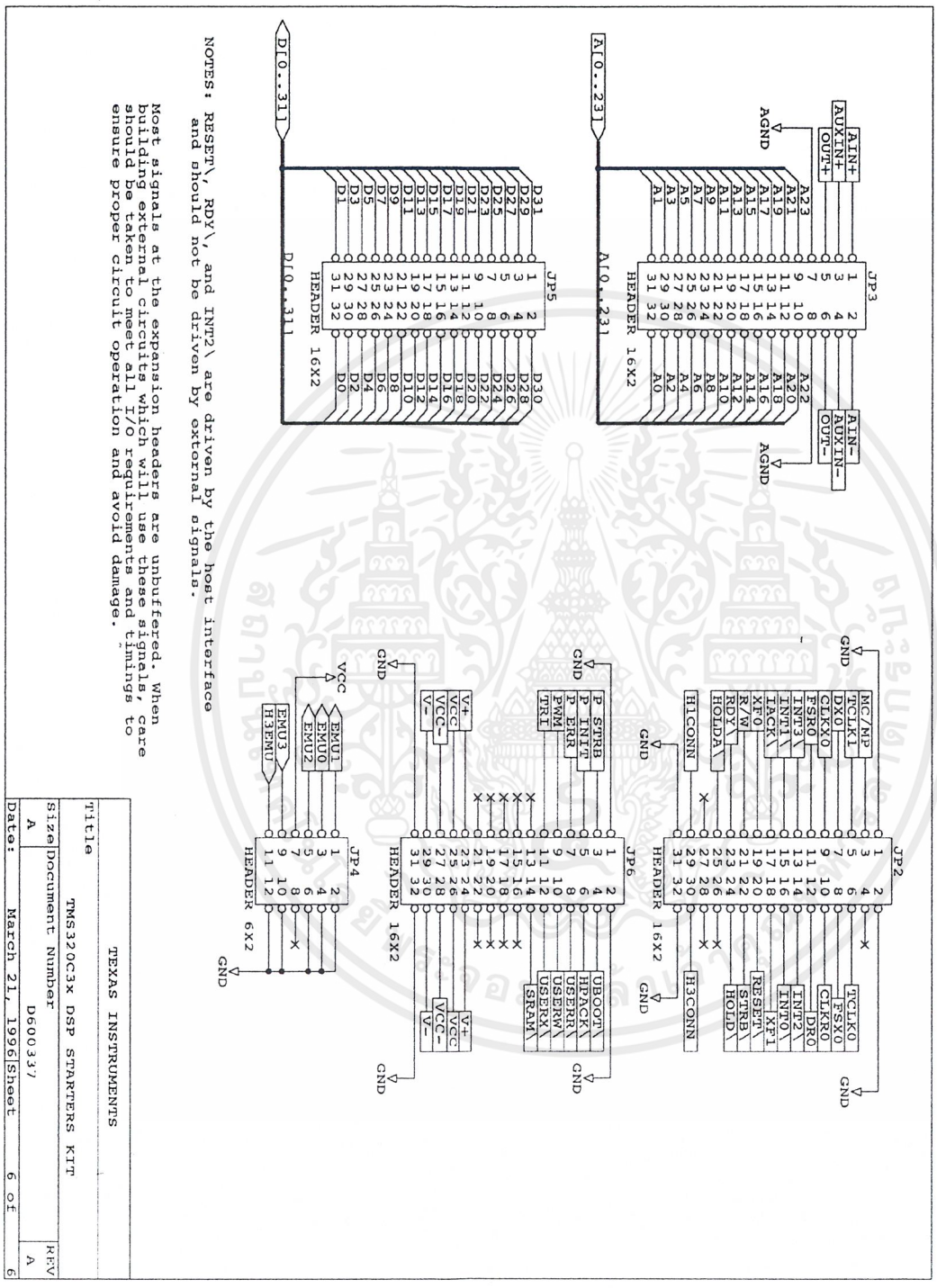
6) R44 and R45 (H1 and H3 clock signals) are not installed.

TEXAS INSTRUMENTS			
Title: TMS320C3x DSP STARTERS KIT			
Size: A	Document Number: D600337		REV: A
Date: March 26, 1996	Sheet: 2 of 6		





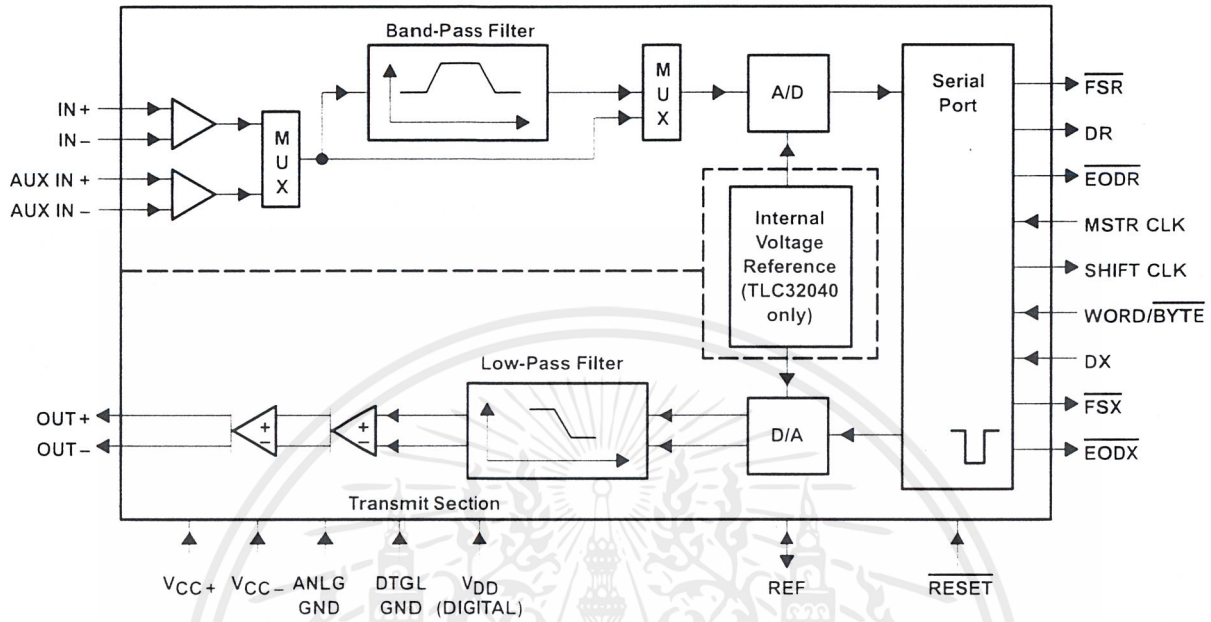




ภาคผนวก

ข.รายละเอียดของอุปกรณ์ที่สำคัญ

functional block diagram



Terminal Functions

TERMINAL NAME	TERMINAL 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 replace the IN+ and IN– inputs. If the bit is a 0, the IN+ and IN– inputs are used (see the AIC DX data word format section).
AUX IN–	23	I	Inverting auxiliary analog input (see the above AUX IN+ description)
DGTL GND	9		Digital ground for all internal logic circuits. Not internally connected to ANLG GND.
DR	5	O	DR 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	DX 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/$\overline{\text{BYTE}}$ description and the Serial Port Timing diagrams. During the word-mode timing, $\overline{\text{EODR}}$ 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. $\overline{\text{EODR}}$ can be used to interrupt a microprocessor upon completion of serial communications. Also, $\overline{\text{EODR}}$ 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, $\overline{\text{EODR}}$ 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.

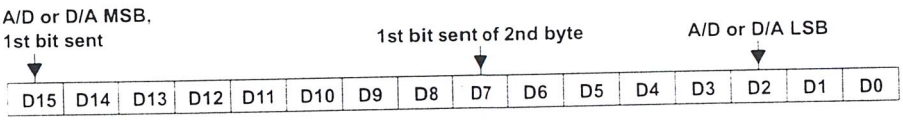
Terminal Functions (continued)

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
$\overline{EODX}$	11	O	End of data transmit. See the WORD/$\overline{BYTE}$ description and the Serial Port Timing diagram. During the word-mode timing, $\overline{EODX}$ 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. $\overline{EODX}$ can be used to interrupt a microprocessor upon the completion of serial communications. Also, $\overline{EODX}$ 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, $\overline{EODX}$ 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/$\overline{BYTE}$ description, $\overline{FSR}$ is held low during bit transmission. When $\overline{FSR}$ goes low, the TMS320 serial port begins receiving bits from the AIC via DR of the AIC. The most significant DR bit is present on DR 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 FSX goes low, the TMS320 serial port begins transmitting bits to the AIC via DX of the AIC. In all serial transmission modes, which are described in the WORD/$\overline{BYTE}$ description, FSX 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	Master clock. MSTR CLK 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. OUT+ 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. OUT– is functionally identical with and complementary to OUT+
REF	8	I/O	Internal voltage reference for the TLC32040. For the TLC32040 and TLC32041 an external voltage reference can be applied to this terminal.
$\overline{RESET}$	2	I	Reset. 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 initializes all AIC registers including the control register. After a negative-going pulse on $\overline{RESET}$, the AIC registers are 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', are reset to 1. The control register bits are 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	Shift clock. SHIFT CLK is obtained by dividing the master clock signal frequency by four. SHIFT CLK is used to clock the serial data transfers of the AIC, described in the WORD/$\overline{BYTE}$ 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)

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
WORD/$\overline{\text{BYTE}}$	13	I	WORD/$\overline{\text{BYTE}}$, 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 are 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. $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ is brought low. 2. One 8-bit byte is transmitted or one 8-bit byte is received. 3. $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ is brought low. 4. $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ 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. $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ is brought high. 7. $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ 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. $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ is brought low. 2. One 16-bit word is transmitted or one 16-bit word is received. 3. $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ is brought high. 4. $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ 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 are configured to be synchronous. In this case, the bandpass switched-capacitor filter and the A/D conversion timing are 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 are identical during primary data communication; however, $\overline{\text{FSR}}$ is not 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. $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ are brought low. 2. One 8-bit byte is transmitted and one 8-bit byte is received. 3. $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ are brought low. 4. $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ 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. $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ are brought high. 7. $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ 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. $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ are brought low. 2. One 16-bit word is transmitted and one 16-bit word is received. 3. $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ are brought high. 4. $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ 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).

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 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. Bits d1 = 0 and d0 = 1 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 (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. Bits d1 = 1 and d0 = 0 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 (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 immediately follows 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 initiates secondary communications upon completion of the primary communications.

Upon completion of the primary communication, $\overline{\text{FSX}}$ remains high for four SHIFT CLK cycles and then goes low and initiates 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. In the synchronous mode, $\overline{\text{FSR}}$ is not asserted during secondary communications.

secondary DX serial communication protocol

x x | ← to TA register → | x x | ← to RA register → |	0 0	d13 and d6 are MSBs (unsigned binary)
x | ← to TA' register → | x | ← to RA' register → |	0 1	d14 and d7 are 2's complement sign bits
x | ← to TB register → | x | ← to RB register → |	1 0	d14 and d7 are MSBs (unsigned binary)
x x x x x x x x d7 d6 d5 d4 d3 d2 Control register	1 1	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– terminals 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 initializes all AIC registers, including the control register. After power has been applied to the AIC, a negative-going pulse on $\overline{\text{RESET}}$ initializes 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, is initialized as follows (see AIC DX data word format section):

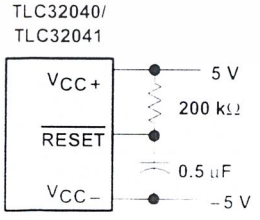
REGISTER	INITIALIZED REGISTER VALUE (HEX)
TA	9
TA'	1
TB	24
RA	9
RA'	1
RB	24

The control register bits are 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 terminal descriptions and AIC DX word format sections).

The circuit shown below provides 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

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, T_A: TLC32040C, TLC32041C . . . . . 0°C to 70°C
TLC32040I, TLC32041 . . . . . −40°C to 85°C
Storage temperature range, T_{stg} . . . . . −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

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

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	0		70	°C
	TLC32040I, TLC32041I	−40		85	

NOTES: 2. Voltages at analog inputs and outputs, REF, V_{CC+}, and V_{CC-}, are with respect to ANLG GND. Voltages at digital inputs and outputs and V_{DD} are with respect to DGTL GND.
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 ANLG 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

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_I = −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_I = −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_I = −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_I = −0 dB to −24 dB referred to V_{ref}, See Note 7		65		dB
	Differential		57	65		

† All typical values are at T_A = 25°C

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

7. The test condition V_I 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.

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_I = –6 dB to –0.1 dB	58		>58§		>58§		dB
	V_I = –12 dB to –6 dB	58		58		>58§		
	V_I = –18 dB to –12 dB	56		58		58		
	V_I = –24 dB to –18 dB	50		56		58		
	V_I = –30 dB to –24 dB	44		50		56		
	V_I = –36 dB to –30 dB	38		44		50		
	V_I = –42 dB to –36 dB	32		38		44		
	V_I = –48 dB to –42 dB	26		32		38		
	V_I = –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_I = –6 dB to 0 dB	58		dB
	V_I = –12 dB to –6 dB	58		
	V_I = –18 dB to –12 dB	56		
	V_I = –24 dB to –18 dB	50		
	V_I = –30 dB to –24 dB	44		
	V_I = –36 dB to –30 dB	38		
	V_I = –42 dB to –36 dB	32		
	V_I = –48 dB to –42 dB	26		
	V_I = –54 dB to –48 dB	20		

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

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

† A_V is the programmable gain of the input amplifier

‡ All typical values are at T_A = 25°C

§ A value >58 is overrange and signal clipping occurs

NOTES: 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.

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

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

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	

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

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	MIN	TYP†	MAX	UNIT
Transmit noise	Single ended	DX input = 00 00 00 00 00 00 00 00, 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

† 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.

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 is correspondingly reduced. The noise is computed by statistically evaluating the digital output of the A/D converter.

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%	
	$\overline{\text{RESET}}$ 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

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 $\overline{\text{FSR}}/\overline{\text{FSX}}/\overline{\text{FSD}}$↓		30		ns
$t_{d(CH-FH)}$	Delay from SCLK↑ to $\overline{\text{FSR}}/\overline{\text{FSX}}/\overline{\text{FSD}}$↑		35	90	ns
$t_{d(CH-DR)}$	DR valid after SCLK↑			90	ns
$t_{d(CH-EL)}$	Delay from SCLK↑ to $\overline{\text{EODX}}/\overline{\text{EODR}}$↓ in word mode			90	ns
$t_{d(CH-EH)}$	Delay from SCLK↑ to $\overline{\text{EODX}}/\overline{\text{EODR}}$↑ in word mode			90	ns
$t_{f(EODX)}$	$\overline{\text{EODX}}$ fall time		2	8	ns
$t_{f(EODR)}$	$\overline{\text{EODR}}$ fall time		2	8	ns
$t_{d(CH-EL)}$	Delay from SCLK↑ to $\overline{\text{EODX}}/\overline{\text{EODR}}$↓ in byte mode			90	ns
$t_{d(CH-EH)}$	Delay from SCLK↑ to $\overline{\text{EODX}}/\overline{\text{EODR}}$↑ 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

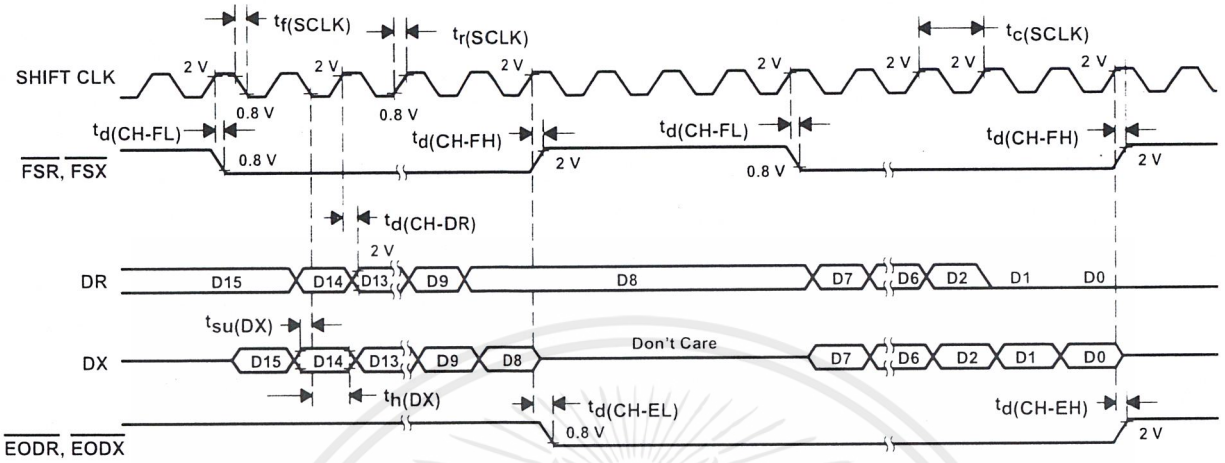
NOTE 12: $\overline{\text{RESET}}$ pulse duration is the amount of time that the reset terminal is held below 0.8 V after the power supplies have reached their recommended values

serial port – AIC output signals

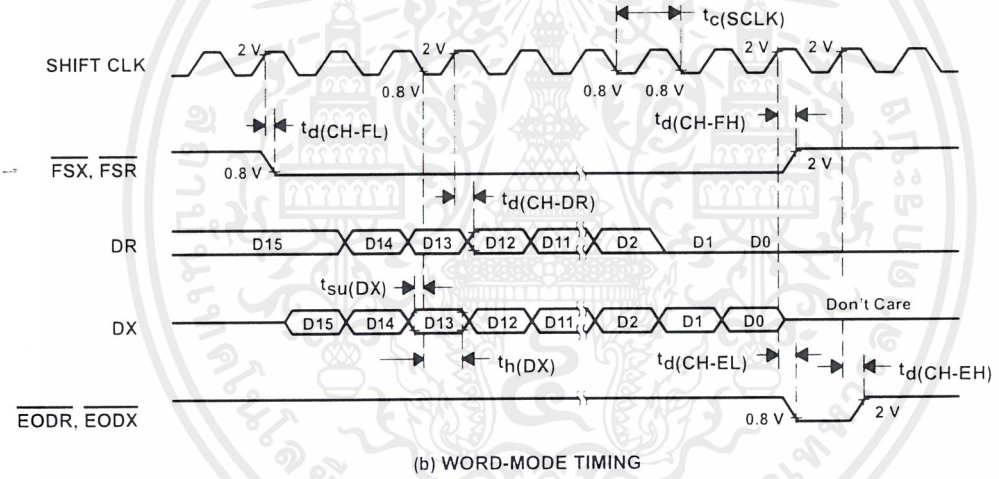
		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
$t_{c(SCLK)}$	Shift clock (SCLK) cycle time		380			ns
$t_{f(SCLK)}$	Shift clock (SCLK) fall time				50	ns
$t_{r(SCLK)}$	Shift clock (SCLK) rise time				50	ns
	Shift clock (SCLK) duty cycle		45		55	%
$t_{d(CH-FL)}$	Delay from SCLK↑ to $\overline{FSR}$/$\overline{FSX}$↓	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
$t_{d(CH-EL)}$	Delay from SCLK↑ to $\overline{EODX}$/$\overline{EODR}$↓ in word mode				90	ns
$t_{d(CH-EH)}$	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
$t_{d(CH-EL)}$	Delay from SCLK↑ to $\overline{EODX}$/$\overline{EODR}$↓ in byte mode				100	ns
$t_{d(CH-EH)}$	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

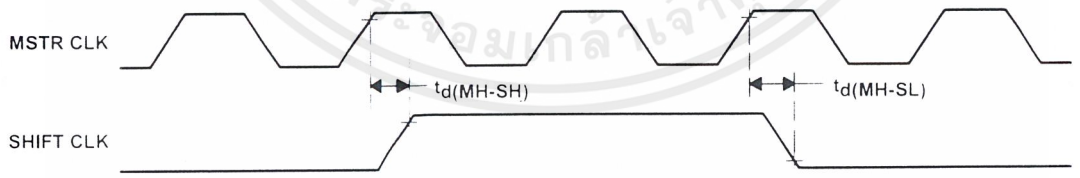
PARAMETER MEASUREMENT INFORMATION



(a) BYTE-MODE TIMING



(b) WORD-MODE TIMING



(c) SHIFT-CLOCK TIMING

Figure 4. Serial Port Timing

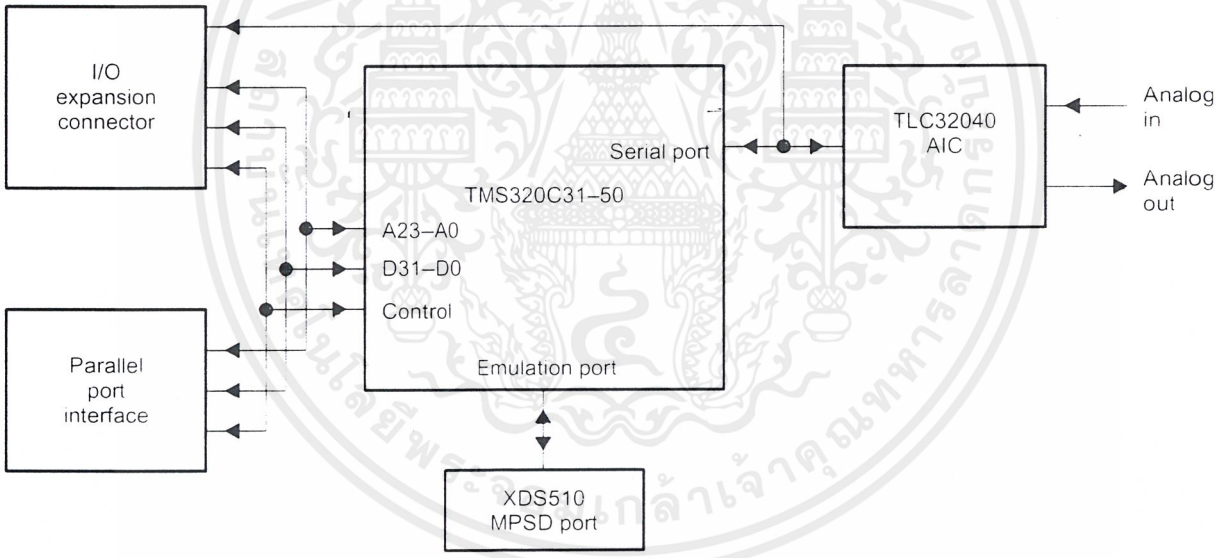
1.2 DSK Overview

Figure 1–1 depicts the block diagram of the TMS320C3x DSK hardware. The basic components are the TMS320C31 DSP, the TLC32040 AIC, expansion connectors, system clock, parallel printer port interface, and tri-color LED. The parallel printer port connects the DSK to a host PC and allows the TMS320C31 to communicate with PC programs.

All of the signals for the 'C3x are routed to expansion connectors. The expansion connectors include four 32-pin headers, an 11-pin jumper block, and a 12-pin XDS510 header.

The TLC32040 AIC interfaces to the TMS320C3x serial port. A jumper block allows removal of this connection to route the serial port to a DSK daughtercard that you supply. Two RCA connectors provide analog input and output on the board.

Figure 1–1. TMS320C3x DSK Block Diagram



See Appendix B, *DSK Circuit Board Dimensions and Schematic Diagrams*, for an explanation of the basic DSK components.

2.3 CPU Primary Register File

The 'C3x provides 28 registers in a multiport register file that is tightly coupled to the CPU. Table 2–1 lists the register names and functions.

All of the primary registers can be operated upon by the multiplier and ALU and can be used as general-purpose registers. The registers also have some special functions. For example, the eight extended-precision registers are especially suited for maintaining extended-precision floating-point results. The eight auxiliary registers support a variety of indirect addressing modes and can be used as general-purpose 32-bit integer and logical registers. The remaining registers provide such system functions as addressing, stack management, processor status, interrupts, and block repeat. See Chapter 3, *CPU Registers,* for more information.

Table 2–1. Primary CPU Registers

Register Name	Assigned Function	Section	Page
R0	Extended-precision register 0	3.1.1	3-3
R1	Extended-precision register 1	3.1.1	3-3
R2	Extended-precision register 2	3.1.1	3-3
R3	Extended-precision register 3	3.1.1	3-3
R4	Extended-precision register 4	3.1.1	3-3
R5	Extended-precision register 5	3.1.1	3-3
R6	Extended-precision register 6	3.1.1	3-3
R7	Extended-precision register 7	3.1.1	3-3
AR0	Auxiliary register 0	3.1.2	3-4
AR1	Auxiliary register 1	3.1.2	3-4
AR2	Auxiliary register 2	3.1.2	3-4
AR3	Auxiliary register 3	3.1.2	3-4
AR4	Auxiliary register 4	3.1.2	3-4
AR5	Auxiliary register 5	3.1.2	3-4
AR6	Auxiliary register 6	3.1.2	3-4
AR7	Auxiliary register 7	3.1.2	3-4
DP	Data-page pointer	3.1.3	3-4
IR0	Index register 0	3.1.4	3-4

Table 2–1. Primary CPU Registers (Continued)

Register Name	Assigned Function	Section	Page
IR1	Index register 1	3.1.4	3-4
BK	Block-size register	3.1.5	3-4
SP	System-stack pointer	3.1.6	3-4
ST	Status register	3.1.7	3-5
IE	CPU/DMA interrupt-enable register	3.1.8	3-9
IF	CPU interrupt flag	3.1.9	3-11
IOF	I/O flag	3.1.10	3-16
RS	Repeat start-address	3.1.11	3-17
RE	Repeat end-address	3.1.11	3-17
RC	Repeat counter	3.1.11	3-17

The **extended-precision registers (R7–R0)** can store and support operations on 32-bit integers and 40-bit floating-point numbers. Any instruction that assumes the operands are floating-point numbers uses bits 39–0. If the operands are either signed or unsigned integers, only bits 31–0 are used; bits 39–32 remain unchanged. This is true for all shift operations. See Chapter 5, *Data Formats and Floating-Point Operation,* for extended-precision register formats for floating-point and integer numbers.

The 32-bit **auxiliary registers (AR7–AR0)** are accessed by the CPU and modified by the two ARAUs. The primary function of the auxiliary registers is the generation of 24-bit addresses. They also can be used as loop counters or as 32-bit general-purpose registers that are modified by the multiplier and ALU. See Chapter 6, *Addressing Modes*, for detailed information and examples of the use of auxiliary registers in addressing.

The **data-page pointer (DP)** is a 32-bit register. The eight least significant bits (LSBs) of the data-page pointer are used by the direct addressing mode as a pointer to the page of data being addressed. Data pages are 64K words long, with a total of 256 pages.

The 32-bit **index registers (IR0, IR1)** contain the value used by the ARAU to compute an indexed address. See Chapter 6, *Addressing Modes*, for examples of the use of index registers in addressing.

The ARAU uses the 32-bit **block size register (BK)** in circular addressing to specify the data block size.

The **system-stack pointer (SP)** is a 32-bit register that contains the address of the top of the system stack. The SP always points to the last element pushed onto the stack. A *push* performs a preincrement; a *pop* performs a postdecrement of the system-stack pointer. The SP is manipulated by interrupts, traps, calls, returns, and the PUSH and POP instructions. See Section 6.10, *System and User Stack Management*, on page 6-29, for more information.

The **status register (ST)** contains global information relating to the state of the CPU. Operations usually set the condition flags of the status register according to whether the result is 0, negative, etc. These include register load and store operations as well as arithmetic and logical functions. When the status register is loaded, however, a bit-for-bit replacement is performed with the contents of the source operand, regardless of the state of any bits in the source operand. Following a load, the contents of the status register are identical to the contents of the source operand. This allows the status register to be easily saved and restored. See Table 3–2 on page 3-6 for a list and definitions of the status register bits.

The **CPU/DMA interrupt-enable register (IE)** is a 32-bit register. The CPU interrupt-enable bits are in locations 10–0. The DMA interrupt-enable bits are in locations 26–16. A 1 in a CPU/DMA interrupt-enable register bit enables the corresponding interrupt. A 0 disables the corresponding interrupt. See Section 3.1.8 on page 3-9 for more information.

The **CPU interrupt flag register (IF)** is also a 32-bit register. A 1 in a CPU interrupt flag register bit indicates that the corresponding interrupt is set. A 0 indicates that the corresponding interrupt is not set. See Section 3.1.9 on page 3-11 for more information.

The **I/O flag register (IOF)** controls the function of the dedicated external pins, XF0 and XF1. These pins may be configured for input or output and may also be read from and written to. See Section 3.1.10 on page 3-16 for more information.

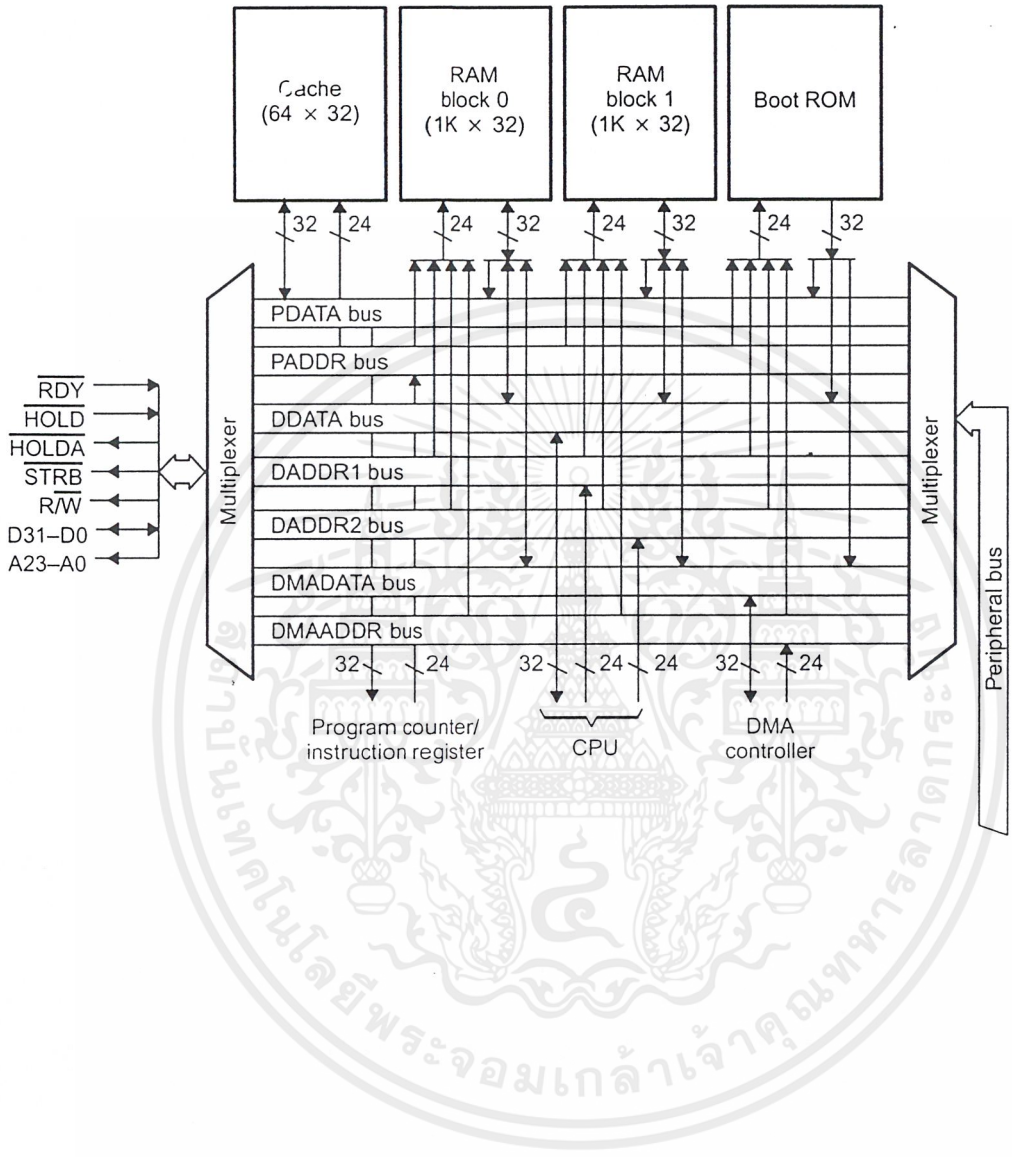
The **repeat-counter (RC)** is a 32-bit register that specifies the number of times to repeat a block of code when performing a block repeat. When the processor is operating in the repeat mode, the 32-bit *repeat start-address register (RS)* contains the starting address of the block of program memory to repeat, and the 32-bit *repeat end-address register (RE)* contains the ending address of the block to repeat.

2.4 Other Registers

The **program-counter (PC)** is a 32-bit register containing the address of the next instruction to fetch. Although the PC is not part of the CPU register file, it is a register that can be modified by instructions that modify the program flow.

The **instruction register (IR)** is a 32-bit register that holds the instruction opcode during the decode phase of the instruction. This register is used by the instruction decode control circuitry and is not accessible to the CPU.

Figure 2–6. Memory Organization of the TMS320C31



2.6 Internal Bus Operation

Much of the 'C3x's high performance is due to internal busing and parallelism. Separate buses allow for parallel program fetches, data accesses, and DMA accesses:

- **Program buses:** PADDR and PDATA
- **Data buses:** DADDR1, DADDR2, and DDATA
- **DMA buses:** DMAADDR and DMADATA

These buses connect all of the physical spaces (on-chip memory, off-chip memory, and on-chip peripherals) supported by the 'C3x. Figure 2–5, Figure 2–6, and Figure 2–7 show these internal buses and their connections to on-chip and off-chip memory blocks.

The program counter (PC) is connected to the 24-bit program address bus (PADDR). The instruction register (IR) is connected to the 32-bit program data bus (PDATA). These buses can fetch a single instruction word every machine cycle.

The 24-bit data address buses (DADDR1 and DADDR2) and the 32-bit data data bus (DDATA) support two data-memory accesses every machine cycle. The DDATA bus carries data to the CPU over the CPU1 and CPU2 buses. The CPU1 and CPU2 buses can carry two data-memory operands to the multiplier, ALU, and register file every machine cycle. Also internal to the CPU are register buses REG1 and REG2, which can carry two data values from the register file to the multiplier and ALU every machine cycle. Figure 2–4 shows the buses internal to the CPU section of the processor.

The DMA controller is supported with a 24-bit address bus (DMAADDR) and a 32-bit data bus (DMADATA). These buses allow the DMA to perform memory accesses in parallel with the memory accesses occurring from the data and program buses.

2.9 Peripherals

All 'C3x peripherals are controlled through memory-mapped registers on a dedicated peripheral bus. This peripheral bus is composed of a 32-bit data bus and a 24-bit address bus. This peripheral bus permits straightforward communication to the peripherals. The 'C3x peripherals include two timers and two serial ports (only one serial port and one DMA coprocessor are available on the 'C31 and one serial port and two DMA coprocessor channels on the 'C32). Figure 2–9 shows these peripherals with their associated buses and signals. See Chapter 12, *Peripherals*, for more information.

Figure 2–9. Peripheral Modules

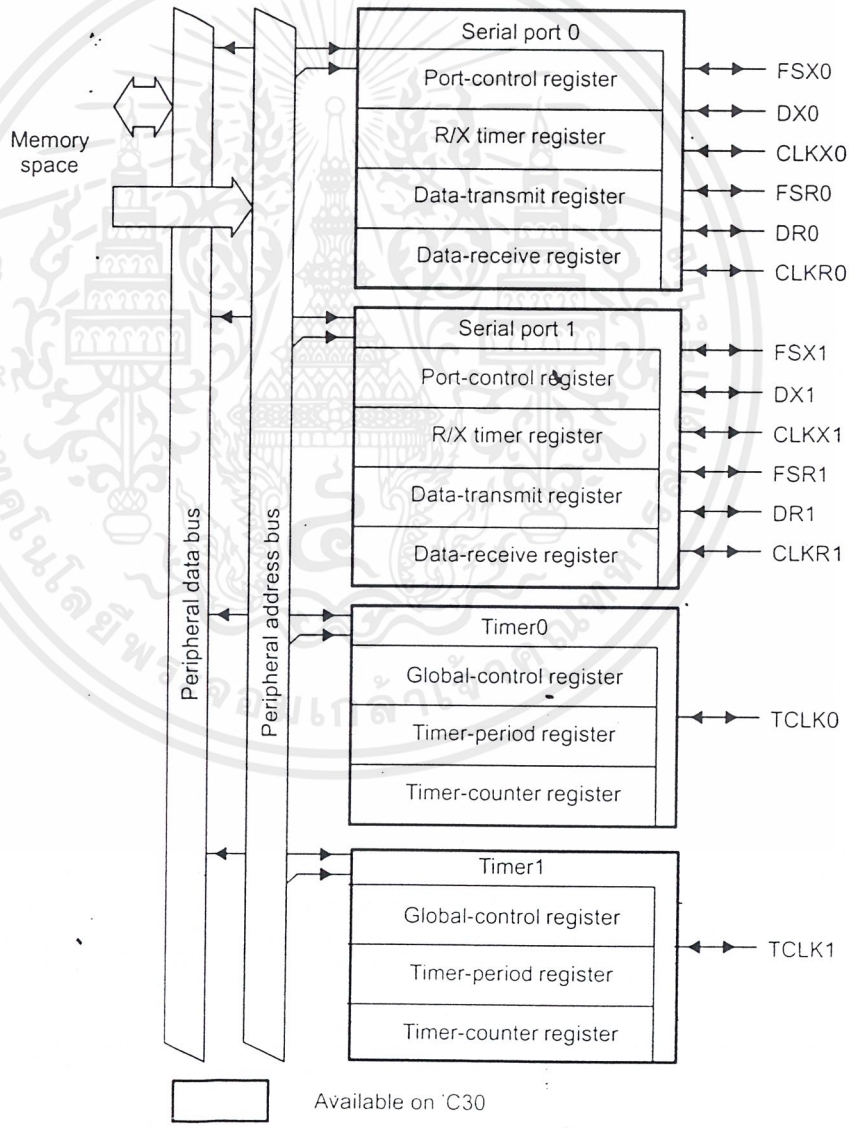
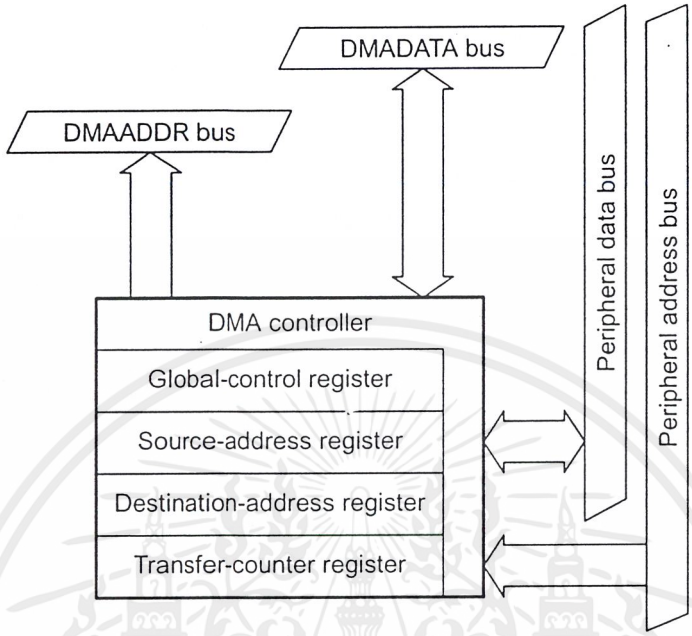


Figure 2–10. DMA Controller



3.1 CPU Multiport Register File

The 'C3x provides 28 registers in a multiport register file that is tightly coupled to the CPU. The program counter (PC) is not included in the 28 registers. All of these registers can be operated on by the multiplier and the ALU and can be used as general-purpose 32-bit registers.

Table 3–1 lists the registers' names and assigned functions of the 'C3x.

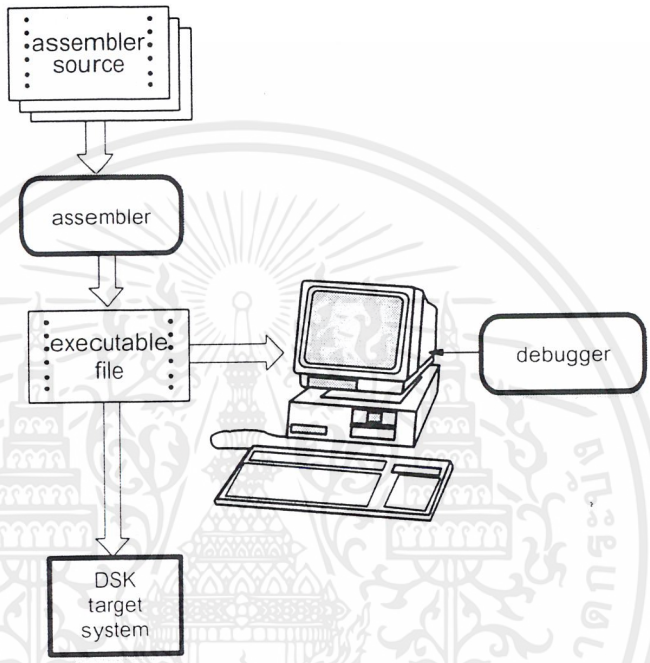
Table 3–1. CPU Registers

Register Symbol	Register Machine Value (hex)	Assigned Function Name	Section	Page
R0	00	Extended-precision register 0	3.1.1	3-3
R1	01	Extended-precision register 1	3.1.1	3-3
R2	02	Extended-precision register 2	3.1.1	3-3
R3	03	Extended-precision register 3	3.1.1	3-3
R4	04	Extended-precision register 4	3.1.1	3-3
R5	05	Extended-precision register 5	3.1.1	3-3
R6	06	Extended-precision register 6	3.1.1	3-3
R7	07	Extended-precision register 7	3.1.1	3-3
AR0	08	Auxiliary register 0	3.1.2	3-4
AR1	09	Auxiliary register 1	3.1.2	3-4
AR2	0A	Auxiliary register 2	3.1.2	3-4
AR3	0B	Auxiliary register 3	3.1.2	3-4
AR4	0C	Auxiliary register 4	3.1.2	3-4
AR5	0D	Auxiliary register 5	3.1.2	3-4
AR6	0E	Auxiliary register 6	3.1.2	3-4
AR7	0F	Auxiliary register 7	3.1.2	3-4
DP	10	Data-page pointer	3.1.3	3-4
IR0	11	Index register 0	3.1.4	3-4
IR1	12	Index register 1	3.1.4	3-4
BK	13	Block-size register	3.1.5	3-4
SP	14	System-stack pointer	3.1.6	3-4
ST	15	Status register	3.1.7	3-5
IE	16	CPU/DMA interrupt-enable	3.1.8	3-9
IF	17	CPU interrupt flags	3.1.9	3-11
IOF	18	I/O flags	3.1.10	3-16
RS	19	Repeat start-address	3.1.11	3-17
RE	1A	Repeat end-address	3.1.11	3-17
RC	1B	Repeat counter	3.1.11	3-17

3.3 Developing Code for the DSK

Figure 3–2 illustrates the DSK code development flow.

Figure 3–2. DSK Software Development Flow



The following list describes the tools shown in Figure 3–2.

assembler

The **assembler** translates DSK assembly language source files into machine language object files for the TMS320C3x family of processors. Only the most essential assembler features are incorporated. This is *not* a COFF assembler, although executable object files created by the TI TMS320 floating-point DSP assembly language tools will also load and run on the DSK.

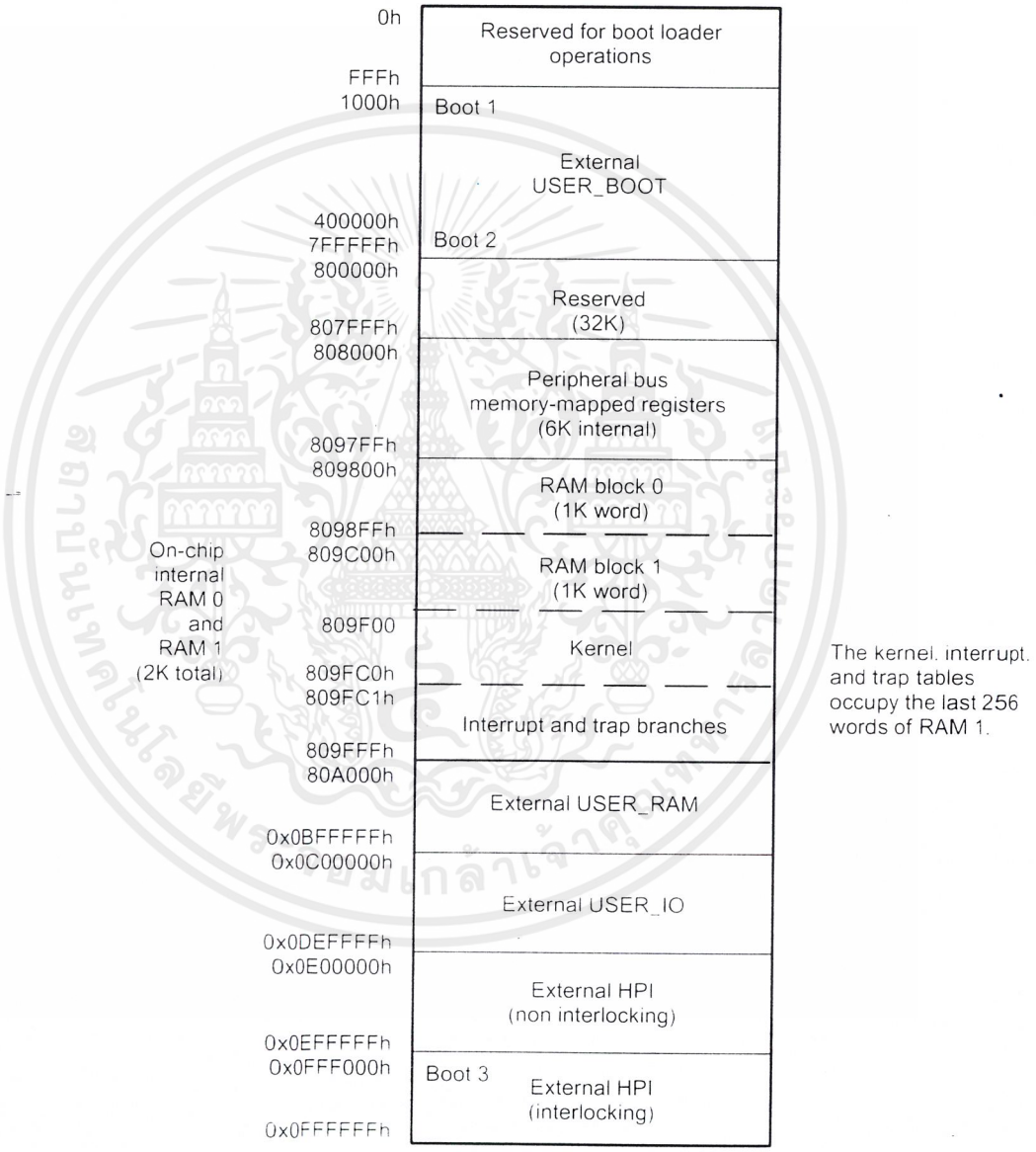
debugger

The main purpose of the development process is to produce a module that can be executed in a **DSK target system.** You can use the debugger to refine and correct your code.

DSK memory map

Because host communications occur through the 'C31 parallel bus, the PAL decodes the address of the 'C31 to determine when it is accessing the host interface according to the memory map shown in Figure 4–4.

Figure 4–4. DSK Memory Map



4.4 Host Software

The DSK software includes several source-code files that manipulate the parallel printer port and perform the necessary functions to initialize and communicate with the 'C31. The commands in each of the source-code files are summarized in the following subsections. The source files that are typically linked include:

driver.cpp	includes driver-level routines that control the host's parallel printer port interface.
target.cpp	includes the low-level routines that manipulate the data transmissions into packets that are recognized by the 'C31 communications kernel.
object.cpp	uses the target- and driver-level routines to initialize and download programs to the 'C31.
dsk_coff.cpp	includes DSK and COFF file loader and utilities.
errormsg.cpp	includes text strings associated with function returns.
symbols.cpp	includes symbol table support routines.
helpmsg.cpp	includes command-line help message.

The following subsections describe the routines contained in each of these files.

DSK software also includes an assembler and a debugger. These are described in Chapter 5, *Using the DSK Assembler*, and Chapter 7, *Using the DSK Debugger*.

Host communications target routines

The communications kernel resident in the 'C31 assumes that data transfers to and from the host are organized into packets as shown in Figure 4–5 on page 4-8. The target.cpp file includes routines that manipulate data transmissions between the host and the 'C31 into this packet structure. These routines read and write blocks of data from the 'C31 memory, send commands to the 'C31, perform context save and restores, and provide debugging commands, such as run, single-step, and halt.

getmem *Get Memory*

Syntax

MSGS getmem (ulong addr, ulong length, ulong *data)

Description

The **getmem** routine reads a block of data from the 'C31 memory.

Arguments

addr Address of the data to be read
length Size of memory block to read
data Pointer to host memory address in which to place data read from the 'C31

Return Value

NO_ERR Block read completed successfully
RECV_ERR Failed reception
XMIT_ERR Failed transmission

putmem *Put Memory*

Syntax

MSGS putmem (ulong addr, ulong length, ulong *data)

Description

The **putmem** routine writes a block of data into 'C31 memory.

Arguments

addr Starting address to write the data to
length Size of memory block to write
data Pointer to host memory address to read data from. The data is then placed into 'C31 memory.

Return Value

NO_ERR Block write completed successfully
XMIT_ERR Failed transmission

SSTEP_CPU — *Single-Step Command*

Syntax

MSGS SSTEP_CPU (void)

Description

The **SSTEP_CPU** routine single-steps one instruction by restoring the context of the CPU, executing one instruction, and then saving the CPU context. This command places the CPU in command mode.

Arguments

None

Return Value

NO_ERR	Command and data completed successfully
XMIT_ERR	Failed transmission
RECV_ERR	Failed reception

RUN_CPU — *Run Command*

Syntax

MSGS RUN_CPU (void)

Description

The **RUN_CPU** routine executes instructions starting at the program counter obtained from the CPU context save area and ending at a breakpoint, if one has been set.

Arguments

None

Return Value

NO_ERR	Command and data completed successfully
XMIT_ERR	Failed transmission

HALT_CPU — *Halt Command*

Syntax

MSGS HALT_CPU (void)

Description

The **HALT_CPU** routine halts the execution of instructions. This command places the CPU in command mode and saves the CPU context.

Arguments

None

Return Value

NO_ERR	Command completed successfully
RECV_ERR	Failed reception

GET_DEBUG_CTXT

Return CPU Context Save Address

Syntax

MSGS GET_DEBUG_CTXT (void)

Description

The **GET_DEBUG_CTXT** routine retrieves the 'C31 context save location starting address. The context address value is placed in the global variable DEBUG_CTXT.

Arguments

External unsigned long DEBUG_CTXT.

Return Value

NO_ERR	Command completed successfully
RECV_ERR	Failed reception
XMIT_ERR	Failed transmission

Host communications driver routines

To facilitate the data transfer from the host to the 'C31, the DSK software includes several driver-level routines in the file driver.cpp. This file includes routines that manipulate the hardware interface circuitry of the host to reset, send, and receive data through unidirectional and bidirectional parallel printer ports.

DSK_reset — *Reset*

Syntax

MSGS DSK_reset (void)

Description

The **reset** routine resets the DSK by toggling the INIT signal.

Arguments

None

Return Value

NO_ERR	Reset sequence completed
RESET_ERR	Reset has failed

input_rdy — *Input Ready*

Syntax

char input_rdy (void)

Description

The **input_rdy** routine indicates that the DSK is ready to receive.

Arguments

None

Return Value

0	DSK ready to receive data
1	DSK not responding to host command

recv_long_byte — *Receive Long Byte*

Syntax

MSGS recv_long_byte (ulong * rcv_data)

Description

The **recv_long_byte** routine receives a 32-bit value in four 8-bit data transfers (to be used only in bidirectional parallel printer ports).

Arguments

rcv_data	Address of the value to receive

Return Value

NO_ERR	Successful reception
RECV_ERR	Failed reception

recv_long — *Receive Long*

Syntax

MSGS recv_long (ulong *rcv_data)

Description

The **recv_long** routine receives a 32-bit value in eight 4-bit data transfers (to be used in bidirectional and unidirectional parallel printer ports).

Arguments

rcv_data Address of the value to receive

Return Value

NO_ERR	Successful reception
RECV_ERR	Failed reception

xmit_long — *Transmit Long*

Syntax

MSGS xmit_long (ulong snd_data)

Description

The **xmit_long** routine transmits a 32-bit value in four 8-bit data transfers (to be used in bidirectional and unidirectional parallel printer ports).

Arguments

snd_data Value to transmit

Return Value

NO_ERR	Successful transmission
XMIT_ERR	Failed transmission

xmit_byte — *Transmit Byte*

Syntax

MSGS xmit_byte (char snd_data)

Description

The **xmit_byte** routine transmits an 8-bit value in a single data transfer (to be used in bidirectional and unidirectional parallel printer ports)

Arguments

snd_data Value to transmit

Return Value

NO_ERR	Successful transmission
XMIT_ERR	Failed transmission

Host communications object routines

Using the low-level driver routines, the DSK software provides several high-level routines that allow the loading of programs or data from dsk3a files or COFF (Common Object File Format), that move binary data from the host to the DSK, and that initialize the DSK system. These routines assume an active communications kernel resident on the 'C31 to send and receive packets of data. See Appendix A of the *TMS320 Floating-Point Assembly Language Tools User's Guide* for a detailed description of the COFF format.

LF — *Load File*

Syntax

Load_File (char *file, TASK task)

Description

The **Load_File** function performs several tasks depending on the enumerated TASK given to it. DSK and COFF file formats are distinguished by the extension of the file. The enumerated TASK list is defined in the file DSK_COFF.H.

An ASCII hexadecimal file format that contains the bootloader header information and raw data is also supported. Since the header information defines where and how long a section is, this file format can be used to either bootload or load files. This file format is easily converted to ROM files with a user-defined post processor.

TASK	Task to perform
LOAD	Loads a DSK or COFF file into the DSK target.
BOOT	Boots a DSK or COFF file into the DSK target.
FILE2HEX	Creates loadable/bootloadable ascii .HEX file.
BOOTHEX	Bootloads FILE.HEX into the DSK.
LOADHEX	Loads (using kernel) FILE.HEX into the DSK
DSK2COFF	Convert DSK file to COFF file.
SLOAD	Loads symbols from the file.

Arguments

***file**	Pointer to the name of the file to load
task	Task to perform

Return Value

NO_ERR	Successful transmission
OPEN_ERR	Cannot open file
ACCESS_ERR	File not found
INV_COFF_MGC	COFF file not created for a TMS320C31
MAX_SECTN	More than 64 sections
BAD_OPTN_HDR	Incorrect optional COFF header
COM_ERR	Communication failure

Init_ Communication

Initialize Communication

Syntax

MSGS Init_Communication (int init_n_times)

Description

The **Init_Communication** function first attempts to communicate with the DSK assuming that a valid communications kernel already exists. If this fails, the DSK is reset and the kernel is bootloaded up to init_n_times. This function also queries an existing communications kernel to determine if the kernel is configured for bytewide- or nibble-mode readback.

After initializing communications with the DSK, the **Load_File** function then loads the desired application.

Arguments

init_n_times Number of times to attempt bootloading the communications kernel before failing. Typically, this value is set to a large value to allow you to connect cables and power to the DSK.

Return Value

NO_ERR The DSK communications link is valid.
INIT_ERR The communications link has failed.

6.1 Using the DSK Assembler Directives

Table 6–1 summarizes the assembler directives. Note that all source statements that contain a directive may have a label and a comment. To improve readability, they are not shown as part of the directive syntax.

Table 6–1. Assembler Directives Summary

(a) Directives that define sections

Mnemonic and Syntax	Description	Page
.data	Assemble source code into data memory	6-18
.sect "*section name*"	Assemble source code into a named (initialized) section	6-27
.text	Assemble source code into program memory	6-32

(b) Directives that initialize constants (data and memory)

Mnemonic and Syntax	Description	Page
.byte $value_1$ [,..... $value_n$]	Initialize one or more 8-bit integers	6-16
.fill *size in words*	Reserve *size* words in the current section; note that a label points to the beginning of the reserved space	6-29
.float *expression*	Initialize a 32-bit TMS320C3x floating-point constant	6-21
.float16 *expression*	Initialize a 16-bit TMS320C3x floating-point constant	6-21
.float8 *expression*	Initialize an 8-bit TMS320C3x floating-point constant	6-21
.ieee *expression*	Initialize one or more 32-bit, IEEE single-precision, floating-point constants	6-22
.int $value_1$ [,..... $value_n$]	Initialize one or more 16-bit integers	6-16
.long $value_1$ [, ... , $value_n$]	Initialize one or more 32-bit integers	6-16
.pfloat16	Initialize 16-bit TMS320C3x floating-point constants into a single word	6-21
.pfloat8	Initialize 8-bit TMS320C3x floating-point constants into a single word	6-21
.q*xx* $value_1$ [,..... $value_n$]	Initialize a 16-bit, signed 2s-complement integer, whose decimal point is displaced *xx* places from the LSB	6-25
.space *size in words*	Reserve *size* words in the current section; note that a label points to the beginning of the reserved space	6-29

(b) Directives that initialize constants (data and memory) (Continued)

Mnemonic and Syntax	Description	Page
.string *"string$_1$"* [,...., *"string$_n$"*]	Initialize one or more text strings	6-31
.word *value$_1$* [, ... , *value$_n$*]	Initialize one or more 32-bit integers	6-16

(c) Directives that reference other files

Mnemonic and Syntax	Description	Page
.copy ["]*filename*["]	Include source statements from another file	6-17
.include "*filename*"	Include source statements from another file	6-17

(d) Directives that enable conditional assembly

Mnemonic and Syntax	Description	Page
.else	Optional conditional assembly	6-23
.endif	End conditional assembly	6-23
.if *well-defined expression*	Begin conditional assembly	6-23
.loop [*well-defined expression*]	Begin repeatable assembly of a code block; the loop count is determined by the *well-defined expression*.	6-24
.endloop	End .loop code block	6-24

(e) Directives that modify the section program counter (SPC)

Mnemonic and Syntax	Description	Page
.align [*size in bytes*]	Align the SPC on a boundary specified by *size in bytes*. which must be a power of 2: default to byte boundary	6-14
.entry [*address*]	Initialize the starting address of the SPC when loading a file	6-20

(f) Directives that define symbols at assembly time

Mnemonic and Syntax	Description	Page
.set *value*	Equate a value with a local symbol	6-28
.sdef *value*	Equate a value with a local symbol multiple times	6-26

(g) Miscellaneous Directives

Mnemonic and Syntax	Description	Page
.brstart *"section name", n*	Align the named section to the next 2n address boundary.	6-15
.end	Program end	6-19
.start *"section name", address*	Links the named section to start assembling at the location *address*.	6-30

7.1 Invoking the Debugger

Here's the command for invoking the debugger:

dsk3d [*options*]

dsk3d is the command that invokes the debugger.

options supply the debugger with additional information.

Table 7–1 lists the debugger options; the following subsections describe some of the more commonly used options.

Table 7–1. Summary of Debugger Options

Option	Brief Description
? or HELP	Displays a listing of the available options
AUTO	Automatically detects if the parallel port supports 8- or 4-bit mode
BW = 4, Nibble	Forces communication using the parallel port in standard 4-bit uni-directional mode
BW = 8, Byte	Forces communication using the parallel port in 8-bit bidirectional mode
LPTx, LPT = x	Selects a parallel printer port (LPT1 is default)
PORT = 0x378	Selects any port address
RESET	Resets (cold boots) the DSK
TEST	Searches automatically through LPT1, LPT2, and LPT3 for the presence of a DSK
T = xx	Adds extra *xx* I/O bus cycles to each transfer for long or noisy cables
WIN = 1	Enables Windows Time Slice management
WIN = 0	Disables Windows Time Slice management and enables set or clear interrupt (STI/CLI)

Displaying a list of available options (? or Help option)

You can display the contents of Table 7–1 on your screen by using the ? or Help option. For example, enter:

```
dsk3d ?
```

Selecting the parallel printer port (LPT = 3 or LPT# option)

The LPT option selects a parallel printer port from the host to communicate with the DSK.

Parallel Printer Port	Functions
LPT1 or LPT = 1	Selects printer port hardware at I/O address 0x378.
LPT2 or LPT = 2	Selects printer port hardware at I/O address 0x278.
LPT3 or LPT = 3	Selects printer port hardware at I/O address 0x3BC.

Note:

Some EISA machines and IBM PS/2s use a different naming convention for the LPTx.

AT Convention	EISA and PS/2	I/O Address
LPT1	LPT2	ox378
LPT2	LPT3	0x278
LPT3	LPT1	0x3BC

Select the parallel printer port at a particular address (PORT option)

The port option selects the parallel printer port at the given address. For example:

```
port = 0x378
```

selects the host's parallel port mapped to the address 0x378.

Note:

Use this option with extreme care since any base address can be used.

Automatically search for a printer port (TEST option)

Use the test option to systematically search for a parallel port that has a DSK connected. The search loops through LPT1, LPT2, and LPT3.

Note:

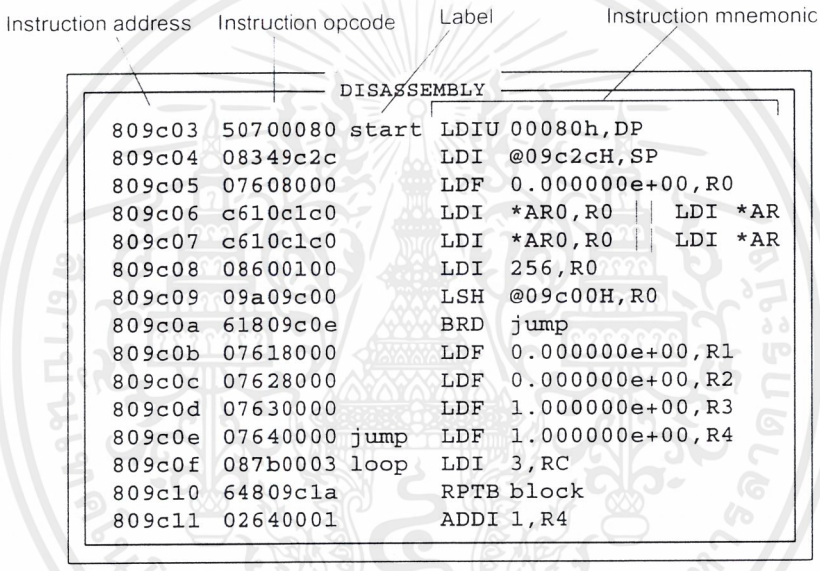
If you have a printer port or other peripheral connected to your PC, turn it off before using the test option.

7.2 Understanding the Debugger Windows

DISASSEMBLY window

The DISASSEMBLY window shows the reverse assembly of memory contents. As shown in Figure 7–1, this window displays several lines of code. Each line shows the instruction address, instruction opcode, label, and instruction mnemonic. The highlighted line corresponds to the next instruction to be executed.

Figure 7–1. DISASSEMBLY Window

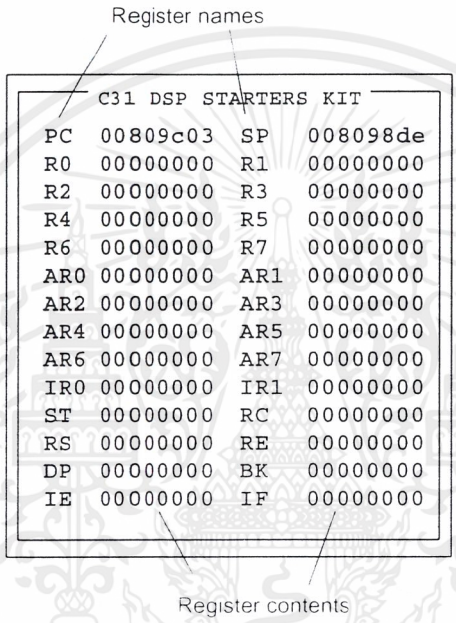


To select the DISASSEMBLY window, press ALT D. While in the DISASSEMBLY window, you can use the cursor to select a line and then use a function key to set or clear a breakpoint. Refer to Table 7–13 for more information about function keys.

CPU REGISTER window

The CPU REGISTER window displays the content of all CPU registers as shown in Figure 7–2. The register's contents are normally displayed in hexadecimal format. You can press F3 to display the extended-precision registers in floating-point decimal format. You can press F2 to display the extended-precision registers in 40-bit hexadecimal format.

Figure 7–2. CPU REGISTER Window



To modify the contents of a register, activate the CPU REGISTER window by pressing ALT C. You can type over the highlighted data and press ENTER to accept the changes when you are satisfied with them. Use the following keys to select the data you want to edit:

→ ↑ ↓ ← PAGE UP PAGE DOWN TAB

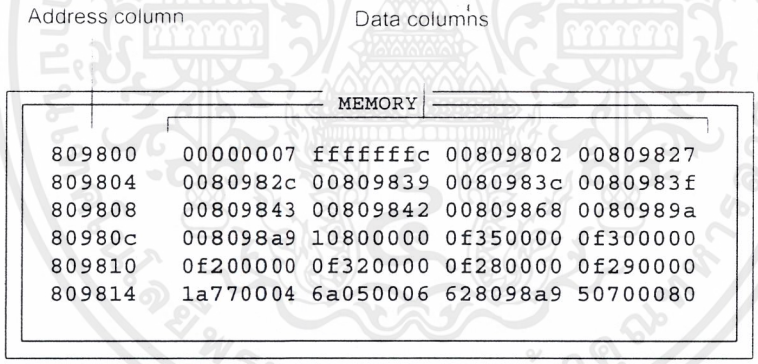
MEMORY window

The MEMORY window shows the contents of a range of memory as shown in Figure 7–3. The MEMORY window has two parts:

- **Addresses**. The first column of numbers identifies the addresses of the first column of display data. No matter how many columns of data you display, only one address column is displayed. Each address in this column identifies the address of the data immediately to its right.
- **Data**. The remaining columns display values at the listed addresses.

For example, the MEMORY window below has four columns of data, so each new address is incremented by 4. Although the window shows four columns of data, there is still only one column of addresses; address 0x0080 9800 contains 0x0000 0007, address 0x0080 9801 contains 0xFFFF FFFC, address 0x0080 9804 (the first value in the second row) contains 0x0080 982C, address 0x0080 9805 contains 0x0080 9839, etc.

Figure 7–3. MEMORY Window



To modify the contents of the MEMORY window, press (ALT) (M) to activate the window and then type over the data. To select a cell, you can use the following keys:

(→) (↑) (↓) (←) (PAGE UP) (PAGE DOWN) (TAB)

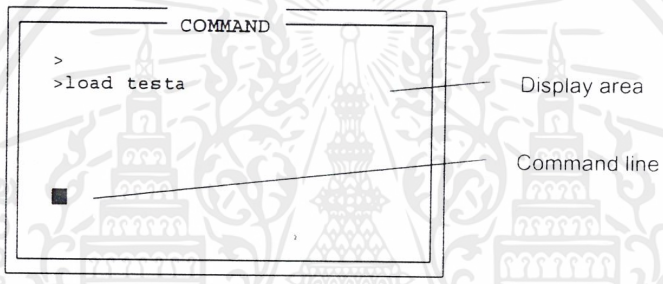
COMMAND window

The COMMAND window provides an area for entering commands, echoing commands, and displaying command output errors and messages. The COMMAND window has two parts:

- **Command line**. This is the area where you enter commands. When you want to enter a command, just type — no matter which window is active.
- **Display area**. This area echoes the commands that you enter, shows any output from your commands, and displays debugger error messages.

Figure 7–4 shows the window command line and display area.

Figure 7–4. COMMAND Window



You can use the ↑ and ↓ keys to select a previously entered command from the buffer (a > is used to indicate the buffer). The editing command keys are shown in Table 7–2.

Table 7–2. Editing Command Keys

To do this	Use this command
Move through the command	← →
Toggle the insert and type over mode	INS
Delete the character at the cursor	DEL
Move to the beginning of the line	HOME
Move to the end of the line	END
Clear the command	ESC
Select a command from the buffer	↑ ↓

ภาคผนวก

ค. แสดง LIST PROGRAMS

```
//PROJECT: Spectrum Analyser

/*This project presents the adaptive signal analysis and its application
  which is designed and controlled by Texas Instrument's DSP board
  TMDS3200031. In this DSP board is programmed for analysis all input
  signal which are changed from time domain to frequency domain by using
  Fourier Transform Theory and display result via the monitor which have
  function is programmed in operation with DOS for illustration about
  input signal in band-low frequency spectrum.According to the real-time
  processing.it is easy to compare between input and output signals. */

// The project file (link list) should include the following
//
//   SPECTRUM.CPP file
//   DRIVER.CPP   Low level printer port drivers
//   TARGET.CPP   DSK Command level
//   OBJECT.CPP   Application setup routines
//   DSK_COFF.CPP DSK and COFF file loaders and other utils
//   ERRORMSG.CPP Messages used for most function returns
//   SYMBOLS.CPP  Symbol tables (needed to link DSK_COFF)
//   TEXTWIN.CPP  DOS level text window functions
//   HARDWARE.CPP Command line help message
//              (Built from HARDWARE.HLP source)
// * EGAVGA.OBJ   EGA and VGA graphics driver

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

#define MSG_BOX    0x809E00L
#define TRG_BOX    0x809E01L
#define A_BOX      0x809E02L
#define B_BOX      0x809E03L
#define C_BOX      0x809E04L
#define EDGESEL    0x809E05L
#define SAMPLES    0x809E06L
#define LOAD_BOX   0x809E07L
#define Samples 512 // FFT size
#define THx 40e-9  // DSP cycle time H1/H3 clock rate
#define DATABLOCK (0x809800L + Samples)
#define menu_vwport() setviewport(0, 262, 639, 340, 1)
#define graph_vwport() setviewport( 50.0, 639, 340, 1)
#define A_REG  ((TA <<9)+(RA <<2)+0) // A divisors.. set SCF rate
#define AP_REG ((TAP<<9)+(RAP<<2)+1) // TA RA prime registers (not used)
#define B_REG  ((TB <<9)+(RB <<2)+2) // B divisors

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

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

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

float Hz_per_div = 0.0;
float Fsr=1000.0, Fsx=1000.0;
int   TA =  10;    // DAC divisors
int   TB =  14;    //
```

```
int  RA = 10;    // ADC divisors
int  RB = 14;    //
int  TAP= 1;     // TA' and RA' are not used in this application
int  RAP= 1;     //
int C_REG=0x03;    // AIC control register bits

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

void binsprintf(char *s,int val);
void init_graphics(void);
int  check_key(void);
void out_TEXT(void);
int  avg_on = 0;



int shift=0;
int X=256;      //marker
float scale= 256;

//-------------------------------------------------
// draw the red cross bars at the top of each data column
//-------------------------------------------------
#define pw 1 // define peak indicator draw width
void draw_peak(void)
{
 int  x, y, *y_pk;
 char *pk_t, *ptr0, *tmp0;
 pk_t = buf_pk_t;
 tmp0 = buf_0;
 y_pk = buf_y_pk;
 ptr0 = tmp0;                   // Set buffer pointers
 setcolor(LIGHTRED);
 x=0;

 while(x<Samples)
 {
  y = 128 - *ptr0++;              present Y to display
  *pk_t += 1;                  // Inc times peak has been displayed
  if((*pk_t > 8) || (y < *y_pk))   // If vmag>last redraw new peak
  {
   line(x-pw,*y_pk,x+pw,*y_pk  ); // undraw old hsorizontal peak

   if(y <= *y_pk)                 // if vmag>last freshen peak hold
   {
          *y_pk = y;
          *pk_t = 0;
   }
   else
   {                        // else decay the peak
         if(y > (*y_pk+6))
           *y_pk += 6;
         else
           *y_pk += 1;
   }
   line(x-pw,*y_pk,x+pw, *y_pk  ); // draw new horizontal peak

 }
 y_pk++;
 pk_t++;
 x=x+2;
```

```
  }
 }
 //-------------------------------------------------------
 // draw the data data columns using an XOR line draw of
 // only the pixels that change for maximum speed.
 //-------------------------------------------------------
 void draw_vect()
 {
  int x, y, *old;
  char *ptr0, *tmp0;
  tmp0 = buf_0;
  ptr0 = tmp0;                    // Set buffer pointers
  setcolor(WHITE);
  old = oldbuf;
  setwritemode(1);
  x=0;

  while(x<Samples)
   {
    y = 128 - *ptr0++;             // present Y to display
    if(y  > *old) line(x,*old+1,x,y  );
    if(y  < *old) line(x,*old  ,x,y+1);
    *old++ = y;
    x=x+2;
   }
 }

 //-------------------------------------------------------------------//

 void main(void)
 {
  int New_Params = 0, reset_flag = 0;
  ulong MSG=START, aic;
  MSGS err;
  char *ptr1, *ptr2, *ptr3, *ptr4;
  char *ptr5, *ptr6, *ptr7, *ptr8, *ptr0;
  char *tmp1, *tmp2, *tmp3, *tmp4;
  char *tmp5, *tmp6, *tmp7, *tmp8, *tmp0;
  int x;
  clrscr();
  Scan_Command_line(DSK_EXE_APP);
  clrscr();
  //*********************************************************//
  // The outer loop initializes the DSK on entry
  // or if the application fails
  //*********************************************************//
  for(;;)
  { for(;;)
    { if(Init_Communication(10000) == NO_ERR) break;
      if(kbhit()) exit(0);
    }
    HALT_CPU (); // Halt previously running apps code
    if((err=Load_File(DSK_APP,LOAD))!=NO_ERR)
    { printf("%s %s\n",DSK_APP,Error_Strg(err));
      exit(0);
    }
    RUN_CPU ();
    init_graphics();

    // Init buff ptrs, Temps are rotated

    tmp0 = buf_0;
    tmp1 = buf_1; tmp2 = buf_2; tmp3 = buf_3; tmp4 = buf_4;
    tmp5 = buf_5; tmp6 = buf_6; tmp7 = buf_7; tmp8 = buf_8;
    ptr0 = tmp0;
    for(x=0;x<Samples;x++)                 // Clear all buffersq
    {
     buf_1[x] = 0; buf_2[x] = 0; buf_3[x] = 0; buf_4[x] = 0;
     buf_5[x] = 0; buf_6[x] = 0; buf_7[x] = 0; buf_8[x] = 0;

     buf_0[x]    = 0;
     buf_y_pk[x] = 255;
```

```
  buf_pk_t[x] = 8;
  oldbuf[x]   = 255;
 }

 check_key();
 draw_vect(); //
 draw_peak(); // Draw peaks and vector displays using XOR write
 //*************************************************************//
 // The inner loop is repeated until a keyboard hit exits the //
 // application or the application reports an error and needs //
 // to be reinitialized                                       //
 //*************************************************************//
 for(;;)
 { // The inner loop draws the display using an XOR line draw
  // to overdraw existing vectors
  ptr0 = tmp0;                          // Set buffer pointers
  ptr1 = tmp1; ptr2 = tmp2; ptr3 = tmp3; ptr4 = tmp4;
  ptr5 = tmp5; ptr6 = tmp6; ptr7 = tmp7; ptr8 = tmp8;
  for(x=0;x<Samples/2;x+=1)
  { switch(avg_on)
         { case  1: *ptr0=(*ptr1+*ptr2)          >> 1; break; // avg 2
           case  2: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4) >> 2; break; // avg 4
           case  3: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4+          // avg 8
                           *ptr5+*ptr6+*ptr7+*ptr8) >> 3; break;
           default: *ptr0= *ptr1;                 break; // avg 1
         }
         ptr0++;
         ptr1++; ptr2++; ptr3++; ptr4++;                  // next data
         ptr5++; ptr6++; ptr7++; ptr8++;
  }
  draw_vect();
  draw_peak();


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

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

```
      if(getmem(DATABLOCK,Samples/8,(ulong *)ptr1)!=NO_ERR) break;//128 char
      putmem(MSG_BOX,1,&MSG);
    }
    closegraph();  // Shutdown graphics before re-initializing
    printf("%s: %s\n",DSK_APP, Error_Strg(err));
    printf("Communications are being reinitialized");
    delay(1000);
    DSK_reset();
  }
 }
//---------------------------------------------------
// clip() clips a value to min and max
//---------------------------------------------------
long inline clip(long x, int min, int max)
{
 if(x<min) return min;
 if(x>max) return max;
 return x;
}
//---------------------------------------------------
// check_key() is the main keytrap routine.
//---------------------------------------------------
int check_key(void)
{
 int key=0;
 char buf[80];

 static int old_key=0;  // If the same key is hit repeatedly, speed up
 static int accel=1;

 if(kbhit()) key = bioskey(0) & 0xFF00;
 if(old_key==key) accel = accel + 1;
 else             accel = accel / 4;
 accel = clip(accel,1,200); old_key = key;
 if(key)

 {
  switch(key)
  {

   case _R  : return 1;  // Return reset (break) flag
   case _Q  : closegraph();
                      _setcursortype(_NORMALCURSOR);
                      exit(0);
                      break;

   case _F1 : TA++; break;
   case _F2 : TA--; break;
   case _F3 : TB++; break;
   case _F4 : TB--; break;
   case _F5 : RA++; break;
   case _F6 : RA--; break;
   case _F7 : RB++; break;
   case _F8 : RB--; break;

   case _Rt  :if(X<512)scale = X+=1 ; break;
   case _Lt  :if(X>0)  scale = X-=1 ; break;

   case _A  :
                      switch(avg_on)
                      {
                       case  1: avg_on=2; break;
                       case  2: avg_on=3; break;
                       case  3: avg_on=4; break;
                       default: avg_on=1; break;
                      }
                      break;

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

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

  graph_vwport();
  Fsx    = 1/(2*TA*TB*(2*THx * T0_prdv));
  if(C_REG & 0x20) Fsr = Fsx;
  else             Fsr = 1/(2*RA*RB*(2*THx * T0_prdv));
  Hz_per_div = (520.0/Samples)*Fsr/(2.0*10);

  setfillstyle(SOLID_FILL, BLACK);
  bar(0,261,639,310);
  setcolor(14);
  outtextxy (X,260,"  ");
  setcolor(LIGHTCYAN);

  sprintf(buf, "Hz/div= %7.2f",Hz_per_div/2); outtextxy(10,275,buf);
  sprintf(buf, "  Fdac=%7.2f",Fsx); outtextxy(10,305,buf);

  if(C_REG&0x20)
  sprintf(buf," Fadc=Fdac");
  else
  sprintf(buf," Fadc=%7.2f",Fsr);
  outtextxy(10,315,buf);

  sprintf(buf," X=%7.2f",scale   ); outtextxy(16,330,buf);
  sprintf(buf,"Freq  = %7.2f",scale*Hz_per_div/(1000*51.2)); outtextxy(10,290,buf);
  outtextxy(135,290,"kHz");

  setcolor(LIGHTGREEN);
  outtextxy(190,275,"Control");
  outtextxy(265,275,"Adjust Output");
  outtextxy(390,275,"Adjust Scale");

  setcolor(LIGHTGRAY);
  outtextxy(190,290,"< Left");
  outtextxy(190,300,"> Right");
  outtextxy(190,310,"R Reset");
  outtextxy(190,320,"Q Quit");
  outtextxy(190,330,"Capture spectrum Press |Break key|");

  sprintf(buf,"F1 TA++");                 outtextxy(265,290,buf);
  sprintf(buf,"F2 TA-- TA=%02d",TA);      outtextxy(265,300,buf);
  sprintf(buf,"F3 TB++");                 outtextxy(265,310,buf);
  sprintf(buf,"F4 TB-- TB=%02d",TB);      outtextxy(265,320,buf);

  sprintf(buf,"F5 RA++");                 outtextxy(390,290,buf);
  sprintf(buf,"F6 RA-- RA=%02d",RA);      outtextxy(390,300,buf);
  sprintf(buf,"F7 RB++");                 outtextxy(390,310,buf);
  sprintf(buf,"F8 RB-- RB=%02d",RB);      outtextxy(390,320,buf);

  setcolor(LIGHTBLUE);
  line(0,270,520,270);
  line(0,270,0,400);
  line(520,270,520,400);
  line(0,340,520,340);
  line(165,270,165,400);

  return 0;
}
//---------------------------------------------------------------
// init_graphics() is responsible for initializing the graphics
// and then filling in the display lines and text
//---------------------------------------------------------------
void init_graphics(void)
{
  int gdriver = EGA, gmode = EGAHI, errorcode, Y, X;
```

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

 initgraph(&gdriver, &gmode, "");        // if possible open EGA mode
 errorcode = graphresult();

 setbkcolor(BLACK);

 if (errorcode != grOk)
 { printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);
 }

 clearviewport();
 graph_vwport();
 setcolor(GREEN);
 for(Y=0;Y<=520;Y+=52/2) line(0,Y,520,Y);  // draw reticle
 for(X=0;X<=520;X+=52/2) line(X,0,X,520);  //
 Y = 2;
 X = 2;
 outtextxy(X,Y    ,"+20");
 outtextxy(X,Y+=26,"+10");
 outtextxy(X,Y+=26," +0");
 outtextxy(X,Y+=26,"-10");
 outtextxy(X,Y+=26,"-20");
 outtextxy(X,Y+=26,"-30");
 outtextxy(X,Y+=26,"-40");
 outtextxy(X,Y+=26,"-50");
 outtextxy(X,Y+=26,"-60");
 outtextxy(X,Y+=26,"-70");

}
```