สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การวิเคราะห์สเปกตรัมโดยใช้ CARD TMS320C31

(SPECTRUM ANALYZER BY CARD TMS320C31)

โดย

1. นายจุติโรจน์ เบ็ญจลักษณ์ 38014080 ห้อง 4J
2. นายเจตนา คุณศรีรักษ์สกุล 38014082 ห้อง 4J

อาจารย์ที่ปรึกษา

รศ.ดร.ฟูศักดิ์ ชีวสุวิทย์

เลขหมู่.....
เลขทะเบียน.....33929
วัน, เดือน, ปี 20 ก.ย. 2542

รายงานนี้เป็นส่วนหนึ่งของการศึกษาสตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

เรื่อง **การใช้ CARD TMS320C31 ในการวิเคราะห์สเปกตรัมของสัญญาณ**

(Spectrum Analyzer By CARD TMS320C31)

ผู้จัดทำ

1. นายจุติโรจน์	เบ็ญจลักษณ์	38014080	4J
2. นายเจตนา	คุณศรีรักษ์สกุล	38014082	4J

..... อาจารย์ที่ปรึกษา

(รศ. ดร.ฟูศักดิ์ ชีวสุวิทย์)

หัวข้อปริญญานิพนธ์	การวิเคราะห์สเปกตรัมโดยใช้ CARD TMS320C31	
นักศึกษา	นายจุติโรจน์ เบ็ญจลักษณ์	38014080
	นายเจตนา คุณศรีรักษ์สกุล	38014082
อาจารย์ที่ปรึกษา	รศ. ดร. ฟูศักดิ์ ชีวสุวิทย์	
ระดับการศึกษา	วิศวกรรมศาสตร์บัณฑิตเทคโนโลยีการวัดคุมทางอุตสาหกรรม	
ปีการศึกษา	พ.ศ.2541	

บทคัดย่อ

ปัจจุบันงานในด้านต่างๆได้มีการนำดิจิตอลซิกแนลโปรเซสซิงมาประยุกต์ใช้งานอย่างกว้างขวาง เช่น เครื่องวิเคราะห์สเปกตรัม , การประมวลผลความเร็วสูง , การสื่อสารข้อมูล , ในด้านเครื่องมือวัดทางอุตสาหกรรมและการวัด เป็นต้น

สำหรับปริญญานิพนธ์ฉบับนี้จะใช้ดิจิตอลซิกแนลโปรเซสเซอร์ (Digital Signal Processor) เบอร์ TMS320C31 ในการประมวลผลและวิเคราะห์สัญญาณ ซึ่งการ์ด TMS320C31 จะถูกกำหนดการทำงานด้วยวิธีการวิเคราะห์สัญญาณโดยใช้วิธีการฟาสต์ฟูเรียร์ทรานส์ฟอร์ม Fast Fourier Transform) หรือ บัตเตอร์ ฟลาย(Butterfly)

Thesis Title : SPECTRUM ANALYZER BY CARD TMS320C31

Name : Mr. Jutiroj Benjaluck 38014080

: Mr. Jatetana Kunsriluksakul 38014082

Thesis Advisor : Assoc. Prof. Fusak Cheevasuvit

Level of Study : Engineer of Industrial Technology in Industrial Instrument

Academic Year : 1998

Abstract

Digital signal processing are widely used in many application such as spectrum analyzer , speed processing , data communication , instrumentation and measurement etc.

In this thesis the digital signal processor TMS320C31 will be used as a part of signal analyzer acquisition. The TMS320C31 card will be assigned as a signal analyzer software processor for some application such as spectrum analyzer using Fast Fourier Transform method.

สารบัญ

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

การประมวลผลสัญญาณทางดิจิตอล (DSP) เป็นการนำสัญญาณทางดิจิตอลมากระทำทางคณิตศาสตร์ เช่น บวก,ลบ,คูณ,ถอดรากที่ 2 หรือ การอินทิเกรต ทั้งนี้เพื่อให้ได้ผลลัพธ์ตามชนิดงานในระบบนั้นๆ อาทิ งานควบคุม งานสื่อสาร เป็นต้น

แต่เดิมนั้น ระบบต่างๆใช้สัญญาณอนาลอก คือ สัญญาณจะเข้าหรือออกจากระบบต่างๆอย่างต่อเนื่องตลอดเวลา ระบบดังกล่าวมักมีราคาถูก และ ออกแบบง่าย แต่ข้อเสียคือมักมีขีดจำกัดทางประสิทธิภาพ และความแม่นยำในการประมวลผล ซึ่งข้อเสียนี้กลายเป็นอุปสรรคต่อระบบสมัยใหม่ อันเป็นระบบที่ต้องการความถูกต้องแม่นยำมาก ดังนั้นจึงได้มีการปรับปรุงระบบการประมวลผลสัญญาณให้เป็นระบบดิจิตอล โดยใช้เทคโนโลยีวงจรรวม (Integrated Circuit) ซึ่งได้รับการพัฒนา ให้สามารถทำงานได้อย่างมีประสิทธิภาพสูง และราคา ไม่แพงนัก อีกทั้งยังสามารถใช้งานร่วมกับไมโครโปรเซสเซอร์ คอมพิวเตอร์ และเกทต่าง ๆ ได้

ข้อดีของการประมวลผลสัญญาณดิจิตอลมีดังนี้

1. สัญญาณดิจิตอลเป็นสัญญาณที่มีสถานะตายตัว คือ “0” หรือ “1” ทำให้มีการประมวลผลมีความแน่นอนสูง การใช้งานจึงทำได้อย่างมีประสิทธิภาพ ถูกต้องแม่นยำ มีความละเอียดมากกว่าหนึ่งในพันส่วน ในขณะที่ระบบอนาลอกทำได้ไม่ดีเท่า เนื่องจากอุปกรณ์แบบอนาลอกมีคุณสมบัติ แปรค่าตามสภาพแวดล้อม เช่น อุณหภูมิ จึงทำให้ความแม่นยำและความเชื่อถือได้ของระบบต่ำ

2. มีความยืดหยุ่นมากกว่าการประมวลผลสัญญาณแบบอนาลอก เพราะสามารถเปลี่ยนแปลงเพียงอัลกอริธึม เพื่อให้ฮาร์ดแวร์เดียวกันทำงานหลายๆอย่างได้ จึงใช้งานได้กว้างกว่า ซึ่งจะมีผลทำให้ต้นทุนโดยรวมสำหรับการใช้งานหลายอย่างนั้น มีราคาต่ำกว่าการที่จะต้องสร้างวงจรอนาลอกขึ้นมาหลายๆวงจร เพื่อทำงานทุกอย่างนั้น

อย่างไรก็ตาม การประมวลผลสัญญาณเชิงดิจิตอล ก็มีจุดบกพร่องอยู่เช่นกัน

1. เนื่องสัญญาณต่าง ๆ ทางธรรมชาติเป็นแบบอนาลอก ดังนั้นการใช้งานของระบบสัญญาณดิจิตอลร่วมกับอนาลอก จึงจำเป็นต้องมีอุปกรณ์ประเภทอนาลอกเป็นดิจิตอลและดิจิตอลเป็นอนาลอก เข้ามาช่วยซึ่งอุปกรณ์ดังกล่าว ที่มีประสิทธิภาพและความเร็วสูงนั้นมักมีราคาแพง

2. ระบบสัญญาณดิจิตอลมักมีข้อจำกัดทางด้านแถบความถี่ (Bandwidth) เนื่องจากความเร็วสูงสุดในการทำงานของวงจรดิจิตอลมีค่าจำกัด ซึ่งในปัจจุบันความเร็วดังกล่าวยังมีค่า

ต่ำมาก ทำให้การประมวลผลมีความเร็วต่ำ และใช้ได้กับสัญญาณที่มีแถบความถี่ไม่กว้างมาก เท่าระบบสัญญาณอนาลอก แต่อย่างไรก็ตาม เทคโนโลยี ไอซี ได้ถูกพัฒนาให้มีความเร็วสูงขึ้น จึงเป็นไปได้ว่าในอนาคตอันใกล้นี้ระบบสัญญาณดิจิตอลจะสามารถใช้กับแถบความถี่กว้าง ๆ ได้

ปัจจุบันการใช้การประมวลผลสัญญาณทางดิจิตอลเข้ามามีบทบาทกับงานหลาย ๆ อย่าง เช่น วานควบคุมระบบ , งานกราฟฟิก , ใช้ในเครื่องมือวัดต่าง ๆ เช่น เครื่องวิเคราะห์สเปกตรัม (Spectrum analyzer) , เครื่องสร้างสัญญาณ (Function generator), งานประมวลผลทางเสียง เช่น การจดจำเสียงพูด (Speech recognition), ใช้ในงานทางด้านการสื่อสาร เช่น การกำจัดเสียงสะท้อน (Echo cancellation) , ตู้สาขาโทรศัพท์ดิจิตอล (Digital PBXs) , อีควอไลเซอร์ปรับค่าได้ (Adaptive equalizers) เป็นต้น ในระบบต่าง ๆ ที่กล่าวมานี้จำเป็นอย่างยิ่งที่จะต้องใช้การกรองเชิงเลข (Digital Filtering) ซึ่งเป็นการประมวลสัญญาณทางดิจิตอลชนิดหนึ่ง ที่ถูกนำมาใช้ในเกือบทุกระบบ การกรองเชิงเลขมีหลายประเภท และมักเกี่ยวเนื่องกับการกำจัดสัญญาณรบกวน (noise) และการลดความพร่าเพี้ยน (Distortion) ดังนั้นจึงเป็นการดีสำหรับการเริ่มต้นศึกษาเรื่องการประมวลผลสัญญาณทางดิจิตอล

การประมวลผลสัญญาณทางดิจิตอลมีหลายวิธี วิธีหนึ่งที่นิยมกันก็คือ การใช้ไมโครโปรเซสเซอร์ในตระกูล TMS320 เป็นตัวประมวลผลสัญญาณ ซึ่งไมโครโปรเซสเซอร์ในตระกูล TMS320 นี้มีข้อดีหลายอย่างที่เหมาะจะนำมาใช้ในการประมวลผลสัญญาณดิจิตอล

TMS320 เป็นไมโครโปรเซสเซอร์ประมวลผลสัญญาณทางดิจิตอลยุคต้น มีคุณลักษณะเด่นคือ มีกลุ่มคำสั่งที่มีประสิทธิภาพ มีความยืดหยุ่นในการใช้งาน ทำงานด้วยความเร็วสูง และมีราคาไม่สูงมากแต่ในปัจจุบัน การพัฒนาไมโครโปรเซสเซอร์ตระกูลนี้ก้าวไปไกลมากทั้งด้านประสิทธิ และความเร็วในการคำนวณ ดังนั้น TMS320C31 จึงเหมาะสำหรับเป็นพื้นฐานของการศึกษาอัลกอริธึมในการประมวลผลสัญญาณดิจิตอล อันจะนำไปสู่การพัฒนาอัลกอริธึมให้ซับซ้อนมากขึ้น และมีประสิทธิภาพในการคำนวณมากขึ้น

เนื่องจากการสั่งให้ TMS320C31 ทำงานนั้น จะต้องส่งข้อมูลผ่านทางเครื่องคอมพิวเตอร์ไปเก็บไว้ในหน่วยความจำซึ่งมีขนาด 4 กิโลเวิร์ด แต่ละเวิร์ดมี 16 บิท TMS320C31จะทำงานตามคำสั่งของโปรแกรมภาษาแอสเซมบลี้ที่ได้เขียนขึ้น สำหรับสัญญาณที่จะถูกประมวลผลนั้น อาจเป็นได้ทั้งตัวอย่างสัญญาณส่วนหนึ่งที่ แซมเปิลค่าเก็บไว้ หรือ อาจเป็นสัญญาณจากภายนอกที่ส่งเข้ามาประมวลผลโดยตรงทางการ์ดแปลงสัญญาณ อนาลอกเป็นดิจิตอลที่เรียกว่าการประมวลผลแบบเวลาจริง (Real Time) แต่ในการใช้งานในปริญญานิพนธ์ฉบับนี้ จะเป็นการประมวลผลกลุ่มตัวอย่างของสัญญาณที่เก็บไว้ในหน่วยความจำ และเมื่อ

ประมวลผลเสร็จก็จะส่งผลลัพธ์ไปเก็บไว้ในหน่วยความจำอีกครั้ง ทั้งนี้เพื่อจะนำผลลัพธ์ดังกล่าวไปแสดงผลเปรียบเทียบได้ทางหน้าจอเครื่องคอมพิวเตอร์ โดยการใช้ภาษาซีควบคุม

สำหรับโครงงานนี้ ได้นำบอร์ด DSP สำเร็จรูปของ บริษัท TEXAS INSTRUMENT จำกัด รุ่น TMS320C31 มาประยุกต์การใช้งานทำให้เครื่องคอมพิวเตอร์เป็นเครื่องแสดงผลสเปกตรัม ในช่วงความถี่ต่างๆ

บทที่ 2
การแปลงฟาสท์ฟูเรียร์ (Fast Furier Transform)

2.1 บทนำ (Introduction)

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

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

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

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

2.2 การวิเคราะห์อนุกรมฟูเรียร์ (Furier Series Analysis)

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

$$X(f_n) = \sum_{n=-\infty}^{\infty} X(f_n) e^{-j2\pi f_n t} \qquad (2.1)$$

โดยที่ a_0 เป็นค่าเฉลี่ยของฟังก์ชัน $X(t)$

a_n และ b_n เป็นสัมประสิทธิ์ของอนุกรมของฟังก์ชัน $X(t)$

เมื่อ

$$a_0 = \frac{1}{T} \int_{-T/2}^{T/2} X(t)dt \qquad (2.2)$$

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

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} X(t)\cos(\frac{2\pi fnt}{T})dt \qquad \text{โดยที่ } n = 1 \to \infty \qquad (2.3)$$

และ

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} X(t)\cos(\frac{2\pi fnt}{T})dt \qquad \text{โดยที่ } n = 1 \to \infty \qquad (2.4)$$

นอกจากนั้นเราสามารถเขียนฟังก์ชั่นคาบใดๆให้อยู่ใน อนุกรมฟูเรียร์รูปเชิงซ้อน(Complex Form of the Furier series) ได้ในรูปต่อไปนี้

$$x(t) = A_0 + A_2 \sin(\alpha t + \phi) + A_2 \sin(2\alpha t + \phi) + \ldots \qquad (2.5)$$

โดยแต่ละลำดับของอนุกรมอาจสามารถเขียนได้ดังนี้

$$X(f_n) = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-j2\pi f_n t} dt \qquad \text{เมื่อ} \quad f_n = nf \qquad (2.6)$$

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

$$x(t) = \sum_{n=-\infty}^{\infty} X(f_n) e^{-j2\pi f_n t} \qquad \text{และ} \qquad f_{-n} = -f_n \qquad (2.7)$$

2.3 การแปลงฟูเรียร์ (Furier Transform)

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

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f_n t} dt \qquad (2.8)$$

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

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{-j2\pi f_n t} dt \qquad (2.9)$$

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

$$X(f) = Re\, X(f) + j\, Im\, X(f) \qquad (2.10)$$

ส่วนจริงของ $X(f)$ หาได้จาก

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

$$= \int_{-\infty}^{\infty} x(t) \cos 2\pi f t dt \qquad (2.11)$$

และส่วนจินตภาพของ $X(f)$ หาได้จาก

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

$$= \int_{-\infty}^{\infty} x(t)\sin 2\pi ftdt \qquad (2.12)$$

ดังนั้นสเปกตรัมของแอมปลิจูด (Amplitude spectrum) ของแต่ละความถี่หาได้จาก

$$|X(f)| = \sqrt{(\mathrm{Re}\,X(f))^2 + (\mathrm{Im}\,X(f))^2} \qquad (2.13)$$

และสเปกตรัมของมุมเฟส (Phase spectrum) หาได้จาก

$$\phi(f) = \tan^{-1}\left(\frac{\mathrm{Im}\,X(f)}{\mathrm{Re}\,X(f)}\right) \qquad (2.14)$$

2.4 การแปลงฟูเรียร์แบบเต็มหน่วย (Discrete Furier Transform)

การแปลงฟูเรียร์แบบเต็มหน่วยเป็นการแปลงฟูเรียร์สำหรับข้อมูลในโดเมนเวลาที่ไม่ต่อเนื่อง(ค่าที่ผ่านการสุ่ม) ให้เป็นค่าในโดเมนความถี่ไม่ต่อเนื่องเช่นกัน ผลการแปลงฟูเรียร์ที่ได้จะประกอบด้วยส่วนประกอบย่อยรวมกันดังนี้

$$X(f_k) = \frac{1}{N}\sum_{n=-0}^{N-1} x(t_n)e^{-j2\pi kn/N} \qquad (2.15)$$

เรียกการแปลงฟูเรียร์ลักษณะนี้ว่า *การแปลงฟูเรียร์แบบเต็มหน่วย* (Discrete Furier Transform: DFT)

$$x(t_n) = \sum_{k=0}^{N-1} X(f_n)e^{-j2\pi kn/N} \qquad (2.16)$$

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

คู่การแปลงฟูเรียร์ที่ได้จะอยู่ในรูปเต็มหน่วย (Discrete from)ซึ่งเป็นรูปแบบที่เหมาะสมในการนำมาคำนวณโดยการประมวลผลแบบดิจิตอลต่อไปจากสมการที่ 2.15 เราอาจเขียนผลการแปลงในรูปใหม่ได้ดังนี้

$$X(f_k) = \frac{1}{N}\sum_{n=0}^{N-1} x(t_n) W^{kn} \qquad (2.17)$$

โดยที่

$$W = e^{-j2\pi/N} \qquad (2.18)$$

2.5 การแปลงฟาสท์ฟูเรียร์ (Fast Fourier Transform : FFT)

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

เราจะกล่าวถึงวิธีการซึ่งสามารถลดจำนวนครั้งของการการคูณลงได้เหลือ $M\log_2 N$ ครั้งหรือจำนวนครั้งลดไปถึง $N/(\log_2 N)$ เท่าซึ่งเรียกวิธีการนี้ว่าการแปลงฟาสท์ฟูเรียร์ (Fast Fourier Transform หรือ FFT)

2.5.1 การแปลงฟาสท์ฟูเรียร์แบบฐานสอง (Radix 2 FFT)

2.5.1.1 หลักการเบื้องต้นของ FFT

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

$$X(k) = \sum_{m=0}^{N-1} x(m).W^{mk} \qquad (2.20)$$

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

เราสามารถเขียนสมการ (2.23) ในรูปของสมการเมตริกซ์ได้เป็น

$$\{X\} = \{A\} \cdot \{x\} \tag{2.21a}$$

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

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \tag{2.21b}$$

แต่เนื่องจากคุณสมบัติความเป็นคาบของ W คือ

$$W^{mk} = W^{[mk \bmod(N)]} \tag{2.22}$$

สมการ (2.21b) อาจเขียนได้เป็น

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \tag{2.23}$$

คุณสมบัติความเป็นคาบทำให้เราต้องแยกตัวประกอบของเมตริกซ์ ออกเป็นเมตริกซ์ย่อยหลายเมตริกซ์คูณกัน และสมาชิกภายในเมตริกซ์ย่อยให้มีค่าเป็นศูนย์มากที่สุด วิธีการแยกตัวประกอบนี้จะไม่กระทำโดยตรงจาก แต่จะมีการสลับตำแหน่งหรือจัดกลุ่มของเมตริกซ์ด้วยวิธีการกลับบิต (bit reversal) และเมตริกซ์หลังจัดการสลับแถวแล้วนำมาแยกตัวประกอบอีกครั้ง

วิธีการแยกตัวประกอบอาจทำโดยวิธีการใช้ตัวเลขฐานสอง โดยการแทนดรรชนี k และ m ของสมการ (2.20) ด้วยเลขฐานสอง กรณีที่ ดรรชนี k และ m จะมีค่าได้เพียง 0,1,2, และ 3 เท่านั้น ดังนั้น

$$k = (k_1, k_0) \quad , \quad m = (m_1, m_0) \tag{2.24}$$

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

$$k = 2k_1, k_0 \quad , \quad m = 2m_1, m_0 \tag{2.25}$$

แทนค่า และ ลงในสมการ (2.20)

$$X(k_1, k_0) = x(m_1 m_0)\, W^{2m1+m0).(2k+k0)} \tag{2.26}$$

โดยคุณสมบัติความเป็นคาบของ W และ $W^{4m1k1}=1$ ดังนั้นสมการ (2.26) ได้ใหม่เป็น

$$\begin{aligned} X(k1,k0) &= \sum_{m_0=0}^{1}\left\{\sum_{m_1=1}^{1} x(m_1,m_0)W^{(2m_1k_0)}\right\}W^{(2k_1+k_0)m_0} \\ &= \sum_{m_0=0}^{1}\{x(k_0,m_0)\}W^{(2k_1+k_0)m_0} \end{aligned} \tag{2.27a}$$

โดยสมมติให้ตัวแปร เป็นการคำนวณระหว่างกลาง ผลของสมการ (2.30a) เขียนเป็นเมตริกซ์ใหม่ ๆ ได้เป็น

$$\begin{bmatrix} x(0,0) \\ x(1,0) \\ x(0,1) \\ x(1,1) \end{bmatrix}^{(k_1,k_2)} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^0 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0,.0) \\ x(0,.1) \\ x(1,0) \\ x(1,1) \end{bmatrix}^{(m_1,m_2)} \tag{2.27b}$$

ซึ่งมีผลลัพธ์ระหว่างกลาง และ ผลลัพธ์ เป็น

$$\begin{bmatrix} x(0,0) \\ x(1,0) \\ x(0,1) \\ x(1,1) \end{bmatrix}^{(k_1,k_2)} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0,.0) \\ x(0,.1) \\ x(1,0) \\ x(1,1) \end{bmatrix}^{(m_1,m_2)} \tag{2.28a}$$

และค่า DFT ของสัญญาณเป็น

$$\begin{matrix}(k_1,k_2)\\ \begin{bmatrix} x(0,0)\\ x(1,0)\\ x(0,1)\\ x(1,1)\end{bmatrix}\end{matrix} = \begin{bmatrix} 1 & W^0 & 0 & 0\\ 1 & W^2 & 0 & 0\\ 0 & 0 & 1 & W^1\\ 0 & 0 & 1 & W^0\end{bmatrix} \begin{matrix}(m_1,m_2)\\ \begin{bmatrix} x(0,.0)\\ x(0,.1)\\ x(1,0)\\ x(1,1)\end{bmatrix}\end{matrix} \qquad (2.28b)$$

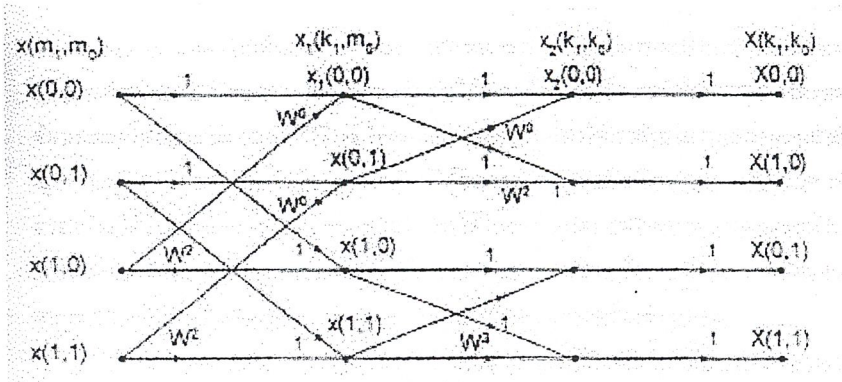
จะเห็นได้ว่าจากผลการแยกตัวประกอบของ $\{A\}$ ได้ว่า สมาชิกตามแนวนอนของตัวประกอบเมตริกซ์ จะมีเพียง 2 ตัวเท่านั้นที่มีค่าไม่เป็นศูนย์ และในสองตัวนี้จะมีสมาชิกหนึ่งตัวมีค่าเป็นศูนย์เสมอ ในขณะที่อีกตัวจะเป็นจำนวนเชิงซ้อน ซึ่งเราต้องการคูณจำนวนเชิงซ้อนเพียง $Mog2N = 8$ ครั้ง และบวกจำนวนเชิงซ้อนอีก 8 ครั้ง ในขณะที่การคำนวณปกติใช้การคูณจำนวนเชิงซ้อน 16 ครั้ง และบวกจำนวนเชิงซ้อน 12 ครั้ง เรายังสามารถลดจำนวนการคูณลงได้อีก จากการคำนวณสัญญาณระหว่างกลาง $x_1(0,0)$ และ $x_1(0,1)$ ซึ่ง

$$x_1(0,0) = x(0,0) + W^0 .x(1,0) = x(0,0) + W^0 .x(1,0) \qquad (2.29)$$

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

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

เราอาจแสดงวิธีการคำนวณ FFT โดยแสดงเป็นกราฟการไหล ดังรูป 2.1 โดยหัวลูกศรชี้ทิศทางการคำนวณ ส่วนอักษรกำกับเป็นค่าตัวคูณค่าของสัญญาณที่ต้นลูกศรนั้น และที่บัพ (Node) เป็นการรวมหรือบวกกันของสัญญาณ ส่วน $x_1(k_0,m_0)$ แทนการคำนวณระหว่างกลางและ $X(k_1,k_0)$ เป็นค่า DFT ของลำดับสัญญาณ



รูปที่ 2.1 กราฟการไหลแสดงถึงวิธีการคำนวณตามสมการ 2.31

2.5.1.2 ขั้นตอนการลดทอนทางเวลา

วิธีการที่เสนอมาแล้วนั้น จะเป็นการจัดกลุ่มลำดับสัญญาณในโดเมนเวลา $x(m)$ ที่มีขนาด N จุด ออกเป็นสองอันดับสัญญาณที่มีความยาว N/2 จุดเท่ากัน โดยเรียกว่าลำดับสัญญาณคู่และลำดับสัญญาณคี่ โดยที่ลำดับสัญญาณคู่เกิดจากการเอาลำดับสัญญาณในตำแหน่งเลขคู่มาเรียงกัน ที่เหลือเป็นลำดับสัญญาณคี่ ดังนั้นถ้าเราให้ $x_E(m)$ เป็นลำดับคู่ และ ลำดับคี่เป็น $x_0(m)$ เพราะฉะนั้น

$$x_E(m) = x(2m) \quad ; \quad m = 0,1...(N/2) - 1 \tag{2.33}$$

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

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

$$\begin{aligned} X(k) &= \sum_{m=0}^{N-1} x_{\varepsilon}(m)(W_N)^{loss} + \sum_{m=0}^{N-1} x_{0}(m)(W)^{loss} \\ &\qquad \text{m=เลขคู่} \qquad\qquad \text{m=เลขคี่} \\ &= \sum_{n=0}^{(N/2)=1} x(2m)(W_N)^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1)(W_N)^{(2m+1)k} \end{aligned} \tag{2.31}$$

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

$$X(k) = \sum_{m=0}^{N/2-1} x_{\varepsilon}(m)(W_{N/2})^{loss} + (W_N)^k \sum_{m=0}^{N/2-1} x_0(m)(W_{N/2})^{loss} \tag{2.35}$$

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

โดยที่ $X_1(k)$ และ $X_2(k)$ ผลการแปลง DFT ขนาด $N/2$ จุดของลำดับ $x_E(m)$ และ $x_0(m)$ ตามลำดับ สมการที่ (2.35) แสดงให้เห็นว่าการคำนวณ DFT ขนาด N จุดนั้นสามารถแบ่งการคำนวณย่อยออกเป็นการคำนวณ DFT ขนาด $N/2$ จุด สองอันดับได้ และข้อสำคัญคือ จะทำให้การคูณจำนวนเชิงซ้อนลดลงไปประมาณ 50 เปอร์เซ็นต์ โดยหลักการเดียวกันนี้ทำให้เรายังสามารถแบ่งทอนลำดับ $x_E(m)$ และ $X_0(m)$ ออกเป็นลำดับคู่และลำดับคี่ได้อีก จนในที่สุดเหลือเพียงลำดับขนาด 2 จุด หรืออาจกล่าวได้ว่า การคำนวณ DFT ขนาด N จุด ทำได้โดยการแปลง DFT ขนาด 2 จุด จำนวน $N/2$ ภาคด้วยกัน ข้อสังเกตคือการซอยเพื่อแบ่งลำดับ $x(n)$ ออกเป็นทีละครึ่งจนเหลือการคำนวณ DFT ขนาด 2 จุด นี้ สำหรับสัญญาณ N ลำดับ จะทำการแบ่งได้ $\log_2 N$ ครั้ง (ดังรูป 2.3)

การนำ DFT ขนาด 2 จุด จำนวน $N/2$ ภาคนี้มาประกอบกันเพื่อให้ได้การคำนวณ DFT ขนาด N จุดนั้น จะต้องมีหลักเกณฑ์ในการทำเพื่อไม่ให้ค่าที่ได้ผิดพลาดไป ดังนั้นเราต้องทำการนิยามสมการ (2.32) สำหรับ $k>N/2$ ด้วยซึ่งทำได้โดยการเขียน

$$X(k) = X_1(k) + (W_N)^k X_2(k) \qquad ;0 \le k \le (N/2)-1 \qquad (2.33)$$

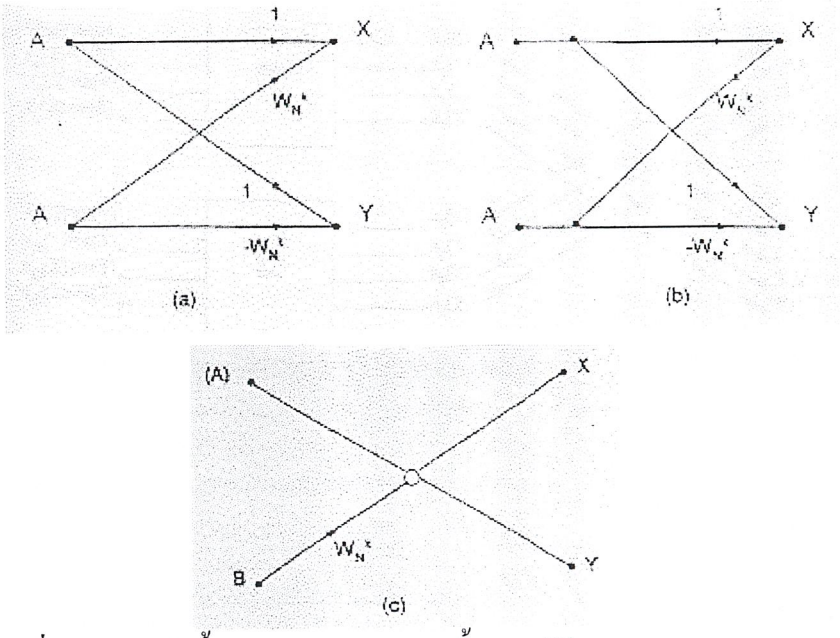
$$= X_1(k - N/2) + (W_N)^k X_2(k - n/2) \qquad ;N/2 \le k \le N/2-1$$

พจน์ $(W_N)^k$ ในสมการ 2.32 เรียกว่าตัวประกอบการหมุน (twiddle factor) ซึ่งมีความสำคัญในการนำ DFT ขนาด 2 จุด หรือ DFT ขนาด $N/2$ จุดมาประกอบกันเป็น DFT ขนาด N จุดได้เหมือนเดิม และจากความสัมพันธ์ $(W_N)^{k-N/2} = -(W_N)^k$ เราจะได้

$$X(k) = X_1(k) + (W_N)^k X_2(k) \qquad ;0 \le k \le (N/2)-1 \qquad (2.34a)$$

$$= X_1(k - N/2) + (W_N)^{k-N/2} X_2(k - N/2) \;;N/2 \le k \le N/2-1 \qquad (2.34b)$$

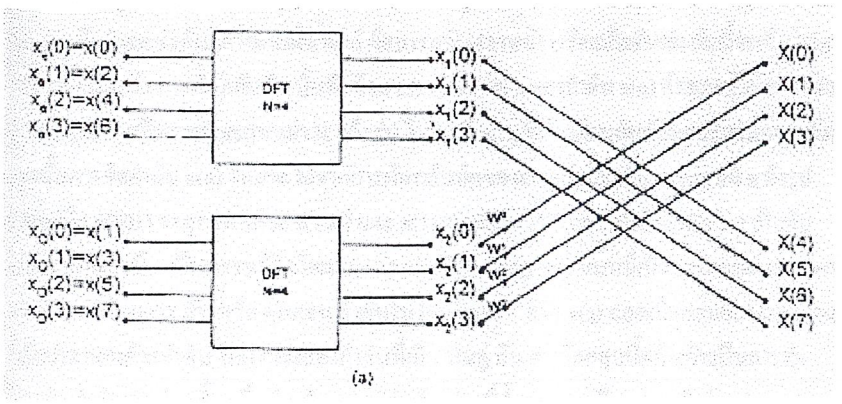
ตามสมการ 2.37 ทำให้เราทราบว่าในการคำนวณหา DFT ของลำดับคู่หนึ่ง จะประกอบด้วยลำดับ $X(k)$ ในสมการ 2.34a และลำดับ $X(k)$ ในสมการ 2.34b ซึ่งจะห่างออกไปจากลำดับ $X(k)$ ในสมการ 2.34a ไป $N/2$ จุดนั้น สามารถคำนวณได้โดยใช้สูตรการคูณจำนวนเชิงเส้นเพียงครั้งเดียวเท่านั้น จากผลอันนี้เราจะนำไปสร้างหน่วยความจำที่มีชื่อว่าหน่วย *ความจำผีเสื้อ (butterfly unit)* โดยหน่วยคำนวณนี้ (อาจอยู่ในรูปแบบของวงจรหรือโปรแกรม) มีข้อมูลเข้าสองข้อมูลคือ A และ B และให้ข้อมูลออกเป็น X และ Y เป็น

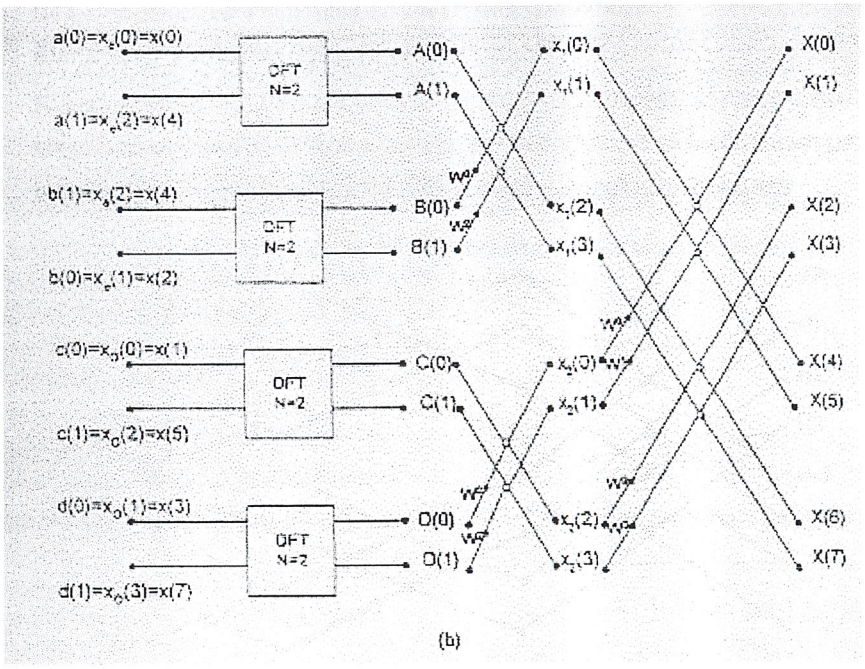


รูปที่ 2.2 หน่วยผีเสื้อของการคำนวณตามขั้นตอนวิธีลดทอนทางเวลา

$$X = A + (W_N)^k.B$$
$$Y = A - (W_N)^k.B \qquad (2.38)$$

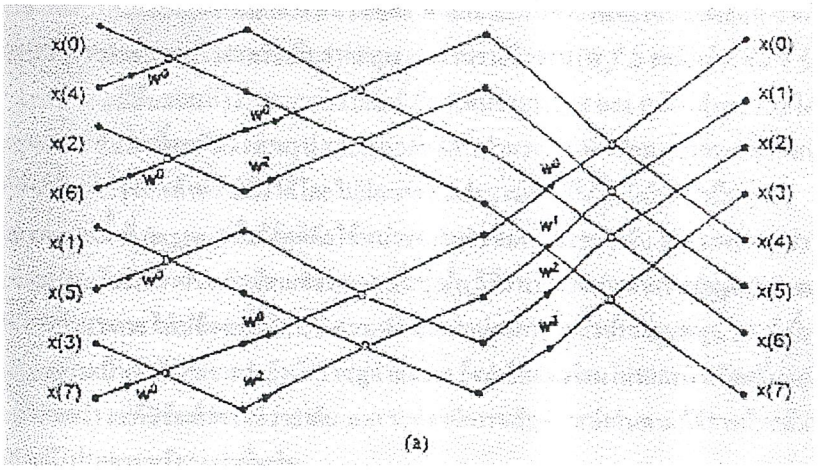
โดยที่การทำงานของหน่วยผีเสื้อ แทนได้ด้วยกราฟการไหล ดังแสดงไว้ในรูป 2.2(a) 2.2(b) หรือ 2.2(c) โดยรูปที่ 2.3 แสดงการคำนวณ DFT แบบ 8 จุด



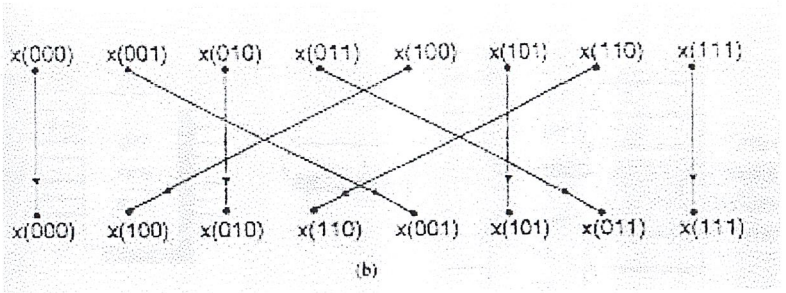


รูปที่ 2.3 (a) และ (b) แสดงขั้นตอนแบบ การลดทอนทางเวลา สำหรับ DFT แบบ 8 จุด

ตามรูป 2.4(a) ลำดับสัญญาณเข้า *x(n)* ไม่ได้ถูกจัดเรียงอย่างต่อเนื่อง หรือ ตามธรรมชาติแต่ไม่ถูก สลับตำแหน่งกันอย่างมีหลักเกณฑ์ คือการสลับตำแหน่ง หรือสลับอันดับ กันนี้จะเป็นไปตามวิธรการที่เรียกว่า การผันกลับบิต นั่นคือถ้าเราแทนดรรชนี n ของลำดับ *x(n)* ด้วยเลขฐานสองโดยที่จำนวนบิตต้องเพียงพอที่จะแทนค่า *N* ได้ เช่น ในกรณี *N=8* ที่ต้องแทนกันด้วยเลขฐานสอง 3 บิต จากนั้นการจัดลำดับ *x(n)* ใหม่จะได้จากการผันกลับบิตของเลขฐานสองที่แทนดรรชนี *n* ดังรูป 2.4 (b) คือ *x(001)* จะถูกแทนด้วย *x(100)* และ *x(110)* และ *x(110)* ถูกแทนด้วย *x(011x)* เป็นต้น เนื่องจากดรรชนี *n* เป็นดรรชนีในโดเมนเวลา และวิธีการของ FFT แบบนี้เป็นการลดทอนเวลาทางการคำนวณโดยการ สับ หรือ ตัดทอน ลำดับในโดเมนเวลา หรือ *x(n)* ออกเป็นกลุ่มย่อยโดยแต่ละกลุ่มประกอบด้วยลำดับ *x(n)* เพียงสองลำดับที่เป็น*ปมคู่*กัน การจัดกลุ่มนี้คล้ายกับเป็นการสุ่มตัวอย่างลำดับเดิมอีกครั้งหนึ่ง ด้วยอัตราการสุ่มตัวอย่างที่ต่ำกว่า และถ้าหากเราถือว่าแต่ละกลุ่มข้อมูลใหม่ที่จัดทำขึ้นมา ต่างเป็นลำดับ ข้อมูลชุดหนึ่งแล้ว ก็เท่ากับว่าเราได้ตัดทอนลำดับในโดเมนเวลาลงไปเป็นกลุ่มลำดับข้อมูลย่อยหลายลำดับ ดังนั้นจึงเรียกวิธีการแบบนี้ว่า การลดทอนทางเวลา ซึ่งแสดงแผนภาพลำดับวิธีการของ การลดทอนทางเวลา ได้ดังรูป 2.4.1.5 โดยที่ DFT ขนาด *N* จุดเดิมถูกแบ่งออกเป็น DFT ขนาด *N/2* จุดจำนวน 2 ภาคนำมารวมกันโดยใช้ตัวประกอบการหมุน และลำดับนี้จะทำจนกระทั่งผลลำดับสุดท้ายเป็นการแปลง DFT ขนาด 2 จุด โดยที่การนำเอา DFT ขนาด *N/2* จุดมาประกอบกัน มีวิธีการที่ขึ้นอยู่กับการจัดเรียงตัวของตัวประกอบการหมุน



รูปที่ 2.4(a) กราฟการไหลสัญญาณแสดงการคำนวณตามรูป 2.3

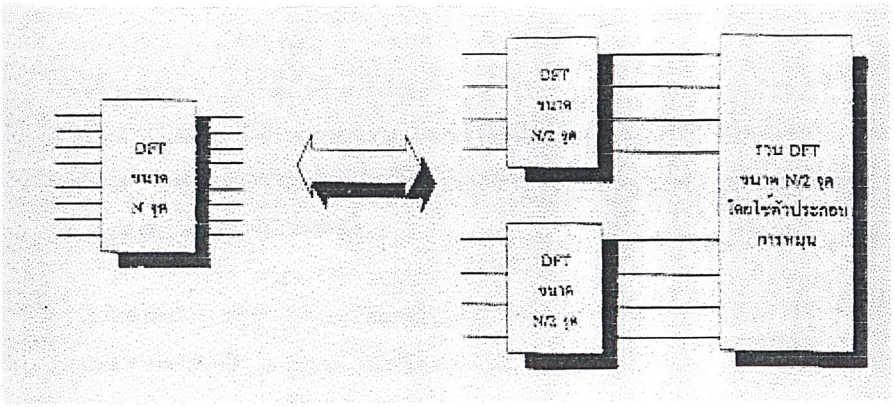


รูปที่ 2.4 (b) แสดงการสลับตำแหน่งของลำดับ x(n) ด้วยการผันกลับบิต

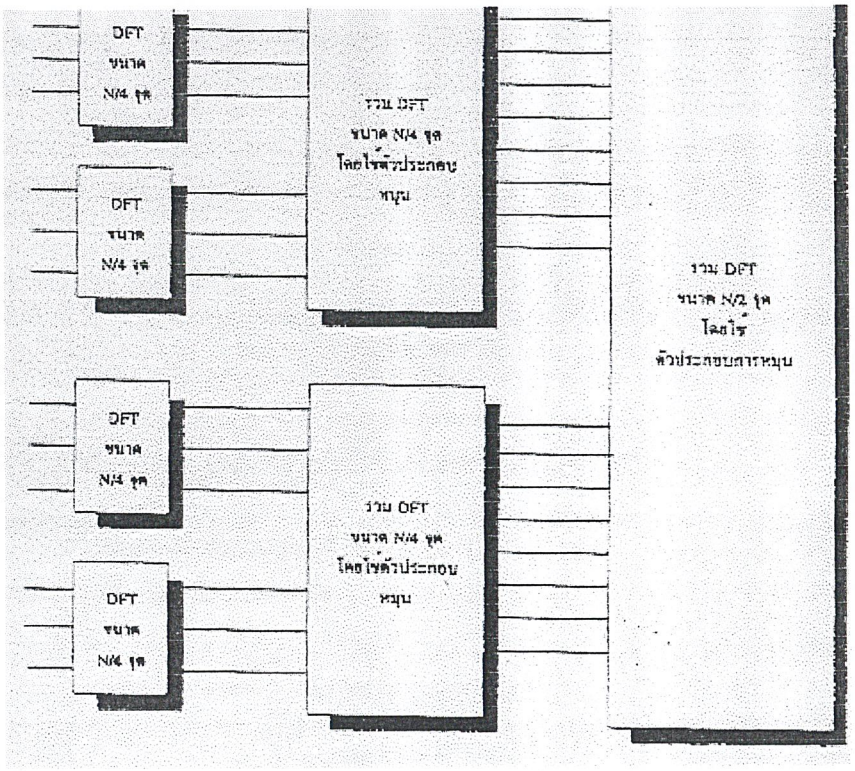
2.5.1.3 การลดทอนทางเวลาโดยวิธีการซ้ำที่

คุณสมบัติที่เป็นข้อดีบางประการของการแปลงฟาสท์ฟูเรียร์ ตามรูป 2.4(a) โดยทั่วไปแล้วอาจกล่าวได้ว่าการคำนวณการแปลง DFT นั้นแท้จริงแล้วก็คือ การนำลำดับจำนวนเชิงซ้อนที่มีอยู่ N ลำดับ มาทำการแปลงเป็นจำนวนเชิงซ้อนอีกกลุ่มหนึ่งที่มีอยู่ N ลำดับเช่นกัน โดยการใช้ FFT การแปลงเช่นนี้จะกระทำ $\log_2 N$ ขั้นตอนด้วยกัน ดังนั้นตามรูป 2.4(a) ซึ่ง $N=8$ ควรต้องมีหน่วยความจำ ($\log_2 N$) หรือ 3 แถวลำดับ (array) ด้วยกันสำหรับเก็บข้อมูลที่ต้องใช้ในการคำนวณโดยที่แถวลำดับแรกไว้เก็บลำดับข้อมูล $x(n)$ สองแถวลำดับ $x_1(k)$ และแถวลำดับสุดท้ายสำหรับผลลัพธ์ $X(k)$ โดยการคำนวณจะประกอบด้วยหน่วยผีเสื้อ ซึ่งลักษณะการคำนวณของหน่วยผีเสื้อนี้ หากเรามีหน่วยความจำต่างหากไว้สำหรับเก็บค่าผลคูณของจำนวนเชิงซ้อน $(W_n)^k.B$ ผลลัพธ์ X กับ Y ที่คำนวณสามารถเก็บแทนที่ไว้ในหน่วยความจำที่เก็บลำดับข้อมูลเข้า A และ B ได้ (ดังรูป 2.2) โดยลักษณะการทำเช่นนี้จะเห็นว่า ตามรูป 2.4(a) ผลลัพธ์การคำนวณทางขวามือสามารถบรรจุแทนที่ในหน่วยความจำทางด้านซ้ายมือได้โดยไม่มีผลต่อการคำนวณส่วนอื่น ๆ ซึ่งเรียกว่าเป็น การคำนวณแบบซ้ำที่ (in place) ซึ่งมีข้อดีคือใช้หน่วยความจำเพียงหนึ่งแถวลำดับ หรือต้องการหน่วยความจำสำหรับเก็บจำนวนเชิงซ้อนเพียง $N+1$ ค่าเท่านั้น วิธีการนี้จะเหมาะสำหรับข้อมูลยาวมาก โดยการคำนวณจะไม่เปลืองเนื้อที่หน่วยความจำ ข้อดีอีกประการคือ ตัวประกอบหมุน $(W_n)^k$ นั้นจะถูกเรียงอย่างเป็นลำดับคือจากกำลังน้อยไปสู่กำลังมาก จึงทำให้การเขียนโปรแกรม หรือสร้างวงจรทำได้ง่ายกว่า แต่ก็มีข้อเสียตรงที่ว่าลำดับ $x(n)$ จะต้องมีการสลับตำแหน่งกันตาม วิธีการผันกลับบิต จึงต้องมีโปรแกรมหรือวงจรเพิ่มเติม

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง



รูปที่ 2.5 ภาพรวมแสดงขั้นตอนวิธีการ DFT ขนาด N จุด แบบลดทอนทางเวลา



รูปที่ 2.5(ต่อ)ภาพรวมแสดงขั้นตอนวิธีการ DFT ขนาด N จุด แบบลดทอนทางเวลา

33929

2.5.1.4 การแปลงฟาสท์ฟูเรียร์ชนิดลดทอนทางความถี่ (Decimation-in-Frequency : DIF)

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

$$\begin{aligned} x_E(m) &= x(m) && ;m=0,1....,(N/2)-1 \\ x_o(m) &= x(m+N/2) && ;m=0,1....,(N/2)-1 \end{aligned} \tag{2.36}$$

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

$$\begin{aligned} X(k) &= \sum_{m=0}^{(N/2)-1} x(m)(W_N)^{mk} + \sum_{m=N/2}^{N-1} x(m)(W_N)^{mk} \\ &= \sum_{m=0}^{(N/2)-1} x_E(m)(W_N)^{mk} + \sum_{m=N/2}^{(N/2)-1} x_0(m)(W_N)^{(m+N/2)k} \\ X(k) &= \sum_{m=0}^{(N/2)-1} x_E(m)(W_N)^{mk} + (W)^{Nk/2} \sum_{m=N/2}^{(N/2)-1} x_0(m)(W_N)^{mk} \end{aligned} \tag{2.37}$$

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

$$X(k) = \sum_{m=0}^{(N/2)-1} \left[x_E(m) + (-1)^k . x_0(m+N/2)\right](W_N)^{mk} \tag{2.38}$$

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

$$\begin{aligned} X(2k) &= \sum_{m=0}^{(N/2)-1} \left[x_E(m) + x_0(m)\right](W_N)^{2mk} \\ &= \sum_{m=0}^{(N/2)-1} f(m)(W_{N/2})^{mk} \end{aligned} \tag{2.42}$$

และ

$$X(2k) = \sum_{m=0}^{(N/2)-1}[x_E(m)+x_0(m)](W_N)^{2mk} = \sum_{m=0}^{(N/2)-1} f(m)(W_{N/2})^{mk} \quad (2.40)$$

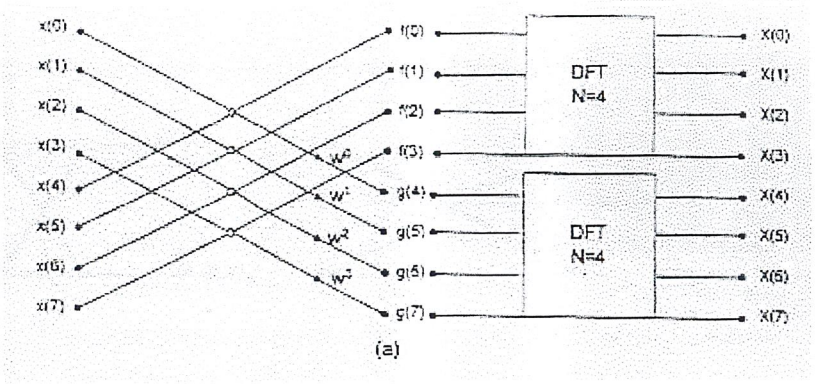
จากผลของ (2.39) และ (2.40) แสดงว่าการคำนวณ DFT ขนาด N จุด ด้วยวิธีนี้จะทำได้โดย ในเบื้องแรกจากลำดับ *x(m)* เราทำการแบ่งครึ่งออกเป็น 2 ลำดับ ดังสมการ (2.39) แล้วนำมาสร้างเป็นลำดับอันใหม่ที่ยาว *N/2* จุด 2 ลำดับ โดยสมมติให้ลำดับใหม่นี้ชื่อ *f(m)* และ *g(m)* ลำดับคู่นี้ สามารถสร้างได้โดยใช้สมการ

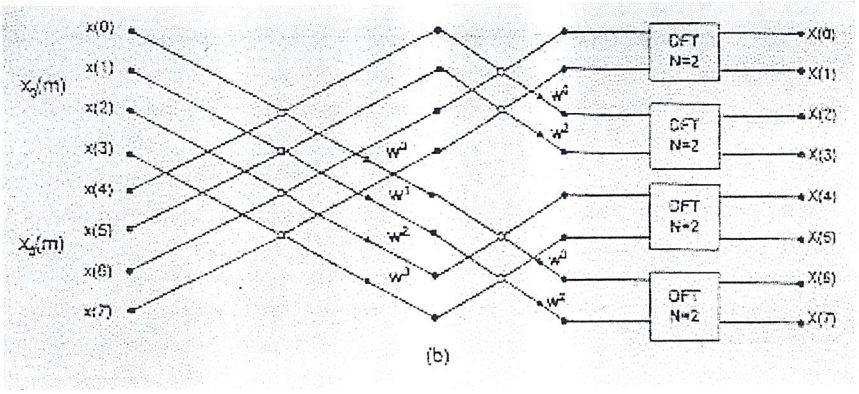
$$f(m) = x_E(m)+x_o(m) \quad ;m=0,1....,(M/2)-1$$
$$g(m) = [x_E(m)-x_o(m)].(W_N)^{mk}; \quad (2.44)$$

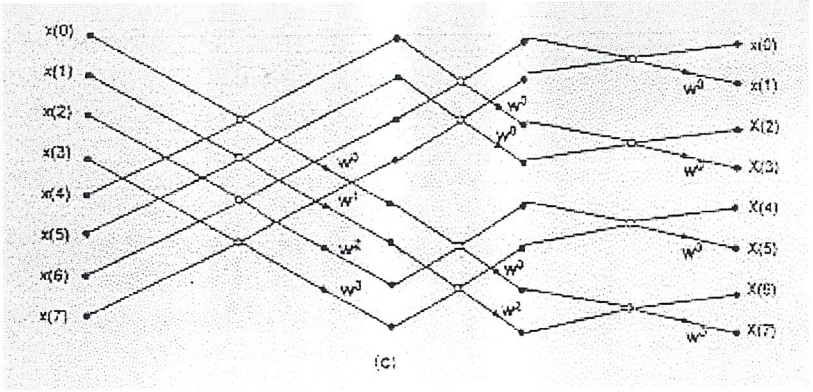
และจากลำดับยาว N/2 จุดในสมการ (2.44) เราก็อาจนำไปคำนวณหา DFT ขนาด N/2 จุด โดยใช้ (2.42) และ (2.43) เพราะฉะนั้น โดยรวมแล้วจะเห็นว่า การคำนวณ DFT ขนาด N จุด ได้ถูกแบ่งเป็นการคำนวณ DFT ขนาด N/2 จุดสองภาคด้วยกัน

จะเห็นว่าเหมือนกับในการลดทอนทางเวลา การแบ่งการคำนวณย่อยออกเป็น DFT ขนาด N/2 จุดนี้สามารถแบ่งย่อยออกไปได้เรื่อย ๆ จนในที่สุดเหลือการคำนวณขนาด 2 จุด ตัวอย่างที่แสดงการคำนวณโดยใช้กราฟการไหลกรณี N=8 แสดงไว้ในรูป 2.6 โดยในรูปแสดงการแบ่งลำดับย่อยออกตามลำดับนั้น จนเหลือการคำนวณ DFT ขนาด 2 จุด และการแบ่งย่อยทำได้ 3 ครั้ง หรือ $\log_2 8$ และจำนวนครั้งในการคูณจำนวนเชิงซ้อนจะประมาณ $N \log_2 N$ เช่นเดียวกัน

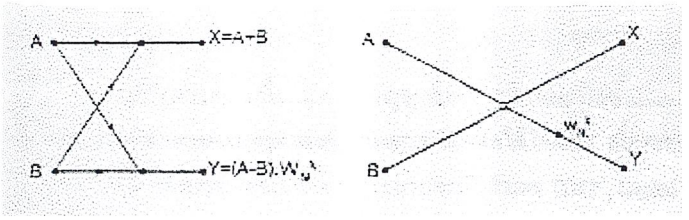
ข้อแตกต่างระหว่างสองวิธีขั้นตอนการคำนวณทั้งสองวิธีคือ ประการแรก การลดทอนทางเวลา *x(n)* จะเรียงตามธรรมชาติ และ *X(k)* เรียงตามธรรมชาติ ส่วนการลดทอนทางความถี่จะตรงข้ามคือ *x(n)* จะเรียงตามธรรมชาติ และ *X(k)* จะถูกเรียงสลับแบบผันกลับบิท ประการที่สองหน่วยผีเสื้อของการลดทอนทางความถี่ต่างไปจากการลดทอนทางเวลาคือเพราะได้จากการเอาลำดับ $x_E(m)$ และ $x_o(m)$ มาบวกและลบกันก่อนแล้วจึงทำการคูณด้วยจำนวนเชิงซ้อน $(W_N)^k$ หน่วยคำนวณผีเสื้อของการลดทอนทางความถี่ แสดงดังรูป 2.7







รูปที่ 2.6 แสดงลำดับขั้นตอนวิธีการของ FFT ชนิดการลดทอนทางความถี่



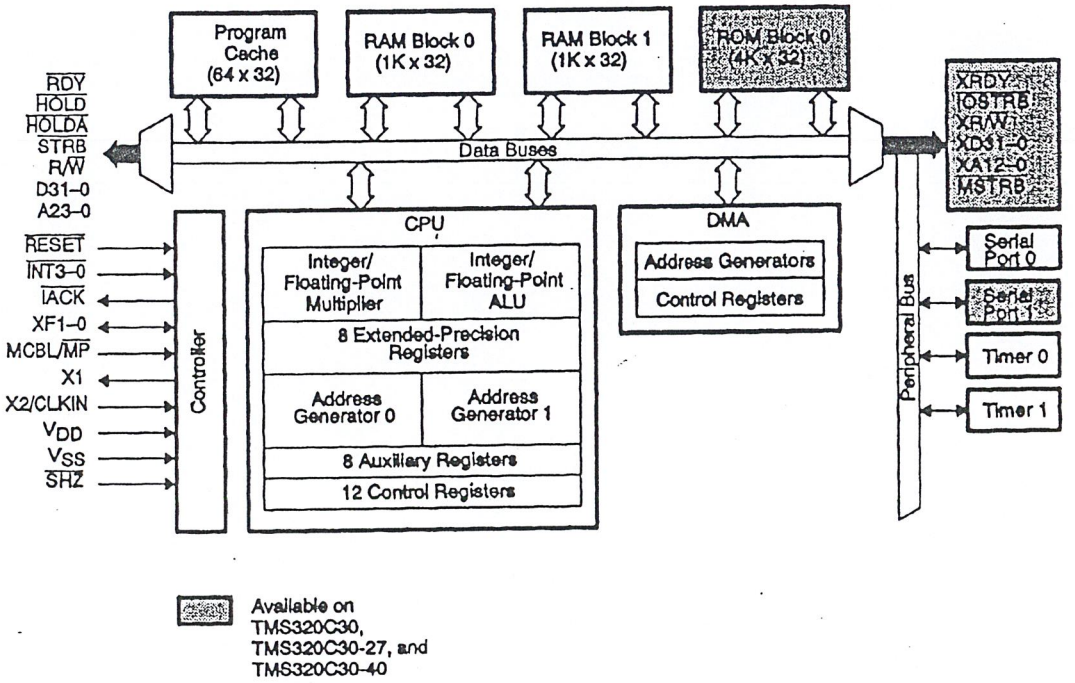
รูปที่ 2.7 แสดงการคำนวณของ หน่วยผีเสื้อ ของขั้นตอนวิธีชนิดลดทอนทางความถี่

บทที่ 3

สถาปัตยกรรมของ TMS320C31

ไอซีเบอร์ TMS320C31 ถูกสร้างและออกแบบมาเพื่อตอบสนองความต้องการทางด้านการประมวลผลบนพื้นฐานของคณิตศาสตร์ชั้นสูงและการแก้ปัญหาทาง hard ware และ soft ware

ประสิทธิภาพอันสูงของ TMS320C31 เกิดจากความแน่นอนของความกว้างทางไดนามิกของส่วนคำนวณทางทศนิยม มีหน่วยความจำขนาดใหญ่ การทำงานแบบขนานและมี DMA Controller เพื่อการทำงานที่มีประสิทธิภาพมากยิ่งขึ้น



รูปที่ 3.1 แสดงสถาปัตยกรรมของไอซีเบอร์ TMS320C3X

สถาปัตยกรรม TMS320C31 ประกอบด้วยส่วนต่างๆ ที่สำคัญดังนี้

3.1 หน่วยประมวลผลกลาง (CPU)

หน่วยประมวลผลกลางจะประกอบไปด้วย

3.1.1 ตัวคูณ (Multiplier)

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

เมื่อ ตัวคูณ ทำการคูณทศนิยม อินพุทที่ใช้มีขนาด 32 บิต และผลลัพธ์ที่ได้จะมีขนาด 40 บิต แต่ถ้า ตัวคูณ ทำการคูณจำนวนเต็มอินพุทจะมีขนาดเพียง 24 บิต และจะได้ผลลัพธ์ขนาด 32 บิต

3.1.2 หน่วยการคำนวณทางคณิตศาสตร์ (Arithmetic Logic Unit (ALU))

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

บัสภายใน (CPU1/CPU2 และ REG1/REG2) จะเป็นตัวพาข้อมูลที่จำนำมาประมวลผล 2 ตัวจากหน่วยความจำ และอีก 2 ตัวจากรีจิสเตอร์ ดังนั้นจึงสามารถทำการคูณ บวก หรือลบแบบขนานได้ ใน 1 รอบการทำงาน

3.1.3 หน่วยช่วยคำนวณทางคณิตศาสตร์ (Auxiliary Register Arithmetic Unit (ARAUs))

หน่วยช่วยคำนวณทางคณิตศาสตร์ เป็นรีจิสเตอร์ที่ช่วยในการทำงานแบบขนานกับตัวคูณ และ หน่วยคำนวณทางคณิตศาสตร์ โดยหน่วยช่วยคำนวณทางคณิตศาสตร์ จะมี 2 ตัว คือ ARAU0 และ ARAU1 หน่วยช่วยคำนวณทางคณิตศาสตร์จะสามารถเก็บแอดเดรสได้ 2 แอดเดรสใน 1 รอบการทำงานเพื่อใช้ในการอ้างตำแหน่งแบบต่างๆ

3.1.4 CPU Register File

ใน TMS320C31 จะมีรีจิสเตอร์อยู่ 28 ตัว ซึ่งตรงกับCPU โดยรีจิสเตอร์ทั้งหลายเหล่านี้จะทำงานกับ ตัวคูณ และ หน่วยคำนวณทางคณิตศาสตร์ และยังสามารถใช้เพื่อวัตถุประสงค์อื่นอีกด้วย อย่างไรก็ตามรีจิสเตอร์เหล่านี้ก็ยังมีหน้าที่พิเศษบางอย่างอีกด้วย เช่น ใช้สำหรับการทำงานเกี่ยวกับจำนวนเลขทศนิยม และรีจิสเตอร์ช่วย(Auxiliary Register) ทั้ง 8 ตัวยังใช้การอ้างแอดเดรสแบบทางอ้อม (indirect addressing register) ด้วยรีจิสเตอร์ที่เหลือจะทำหน้าที่เกี่ยวกับระบบต่างๆ เช่น การอ้างแอดเดรส การจัดการเกี่ยวทับแสตก (stack) การบอกสถานะของโปรเซสเซอร์ การอินเตอร์รัพต์ และการเคลื่อนย้ายข้อมูลเป็นบล็อก

-extend-precision register (R0-R7) : สามารถใช้เก็บค่าจำนวนเต็ม 32 บิต แบะทศนิยม 40 บิต โดยจะเก็บค่าลงในบิตที่ 0-39 แต่ถ้าจำนวนเต็มแบบมีเครื่องหมายก็จะทำการเก็บบิตที่ 0-31 ส่วนบิตที่ 32-39 จะไม่ใช้

-auxiliary register (AR0-AR7) : จะถูกใช้โดยหน่วยประมวลผลหลักและถูกควบคุมโดยหน่วยช่วยคำนวณทางคณิตศาสตร์ ทั้ง 2 ตัว หน้าที่แรกของ AR ยังใช้ให้ส่วนที่เก็นตัวนับในการวนลูปหรือจะใช้เพื่องานอื่นๆ ที่เกี่ยวกับตัวคูณ และ หน่วยคำนวณทางคณิตศาสตร์

-data page pointer (DP) : ส่วนรีจิสเตอร์ขนาด 32 บิต โดยที่ 8 บิตแรก จะถูกใช้ในการอ้างแอดเดรสแบบโดยตรง (direct addressing mode) ซึ่งใช้เป็นตัวชี้ (pointer) ไปยังหน้า (page) ต่างๆ ของข้อมูล

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

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

-system stack pointer (SP) : เป็นรีจิสเตอร์ขนาด 32 บิต ซึ่งเป็นค่าของแอดเดรสของยอดของแสตก โดยแสตก (SP) จะชี้ไปยังค่าสุดท้ายซึ่งถูกเก็บอยู่บนแสตก การ PUSH จะกระทำ ก่อนที่จะเพิ่มค่าของแสตก และการ POP หลังการลดค่าของแสตก คำสั่งที่จะทำงานกับแสตกได้ คือ interrupts, traps, returns, PUSH และ POP

-status register (ST) : เป็นรีจิสเตอรที่เก็บค่าข้อมูลต่างๆที่แสดงถานะของCPU เช่นค่าของ flag ซึ่งจะแสดงถานะของผลลัพธ์ที่ได้จากการคำนวณทางลอจิกด้วย

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

-I/O Flag register (IOF) : เป็นรีจิสเตอร์ในการควบคุมขาXF0 และ XF1

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

-repeat start address register (RS) : รีจิสเตอร์ขนาด 32 บิตจะถูกใช้ในการเก็บแอดเดรสเริ่มต้นของบล็อกข้อมูลที่จะถูกกระทำซ้ำ

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

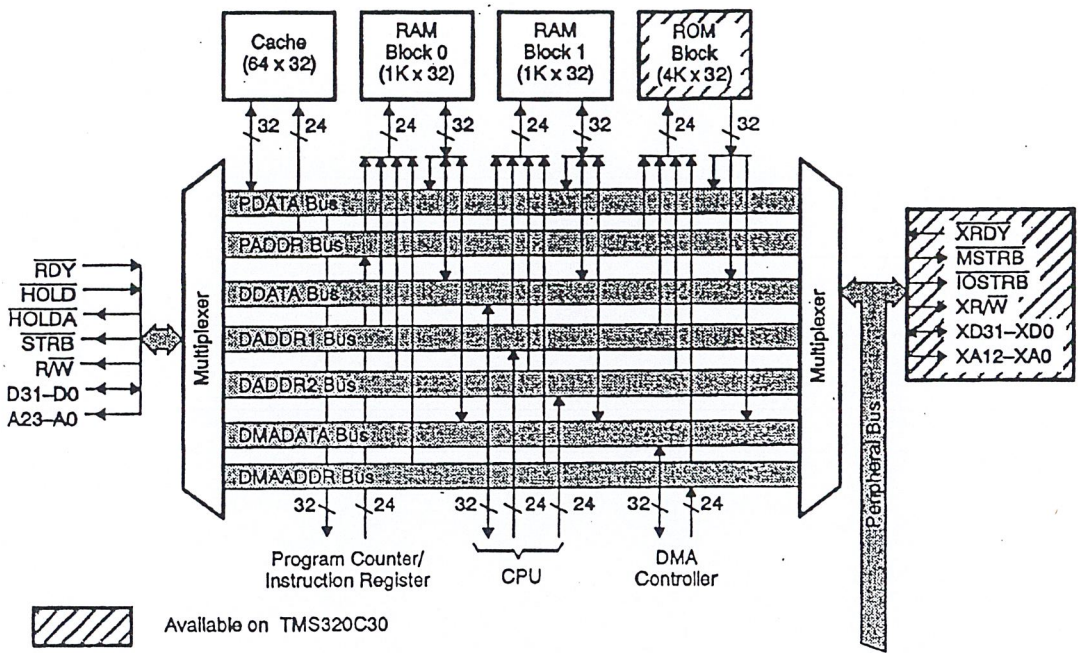
-program counter (PC) : เป็นรีจิสเตอร์ขนาด 32 บิต ซึ่งเก็บแอดเดรสของคำสั่งที่จะถูกนำมาอ่านคำสั่งถัดไป

Register Name	Assigned Function
R0	Extended-precision register 0
R1	Extended-precision register 1
R2	Extended-precision register 2
R3	Extended-precision register 3
R4	Extended-precision register 4
R5	Extended-precision register 5
R6	Extended-precision register 6
R7	Extended-precision register 7
AR0	Auxiliary register 0
AR1	Auxiliary register 1
AR2	Auxiliary register 2
AR3	Auxiliary register 3
AR4	Auxiliary register 4
AR5	Auxiliary register 5
AR6	Auxiliary register 6
AR7	Auxiliary register 7
DP	Data-page pointer
IR0	Index register 0
IR1	Index register 1
BK	Block-size register
SP	System-stack pointer
ST	Status register
IE	CPU/DMA interrupt-enable
IF	CPU interrupt flag
IOF	I/O flag
RS	Repeat start-address
RE	Repeat end-address
RC	Repeat counter
PC	Program counter

ตารางที่ 3.1 ชื่อและหน้าที่ต่างๆของรีจิสเตอร์ในCPU

3.2 หน่วยความจำ (Memory Organization)

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



รูปที่ 3.2 แสดงหน่วยความจำของไอซีเบอร์ TMS320C31

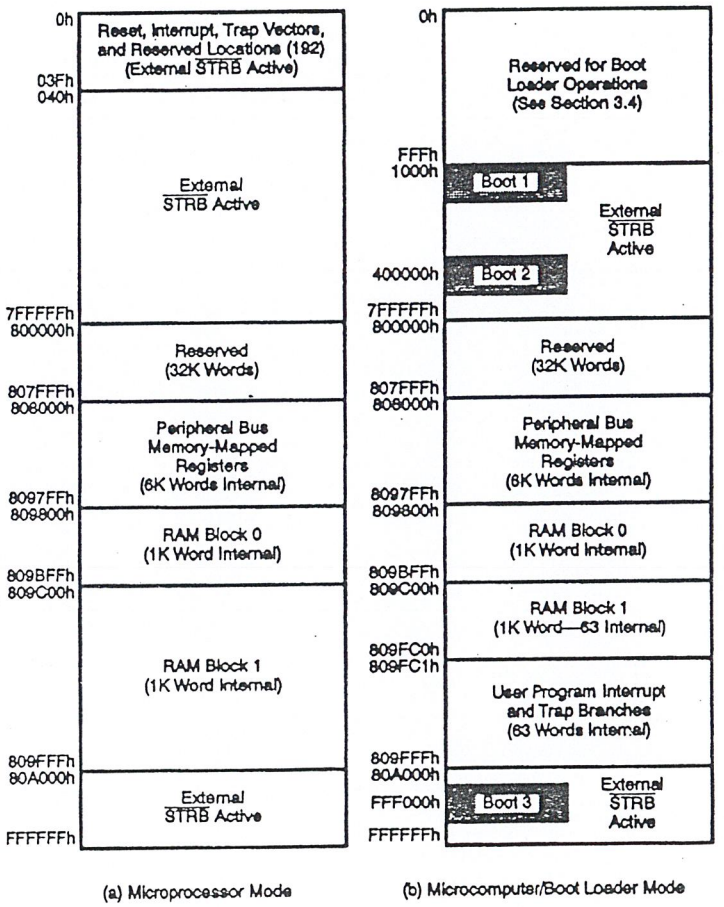
3.2.1 RAM , ROM

รูปที่ 3.2 จะแสดงการจัดหน่วยความจำภายใน TMS320C31 โดยแรมจะแบ่งออกเป็นบล็อก 0 และ บล็อก 1 ซึ่งจะมีขนาดบล็อกละ 1K x 32 และ ROM จะมีขนาด 4K x 32 ทั้ง RAM และ ROM สามารถถูกเข้าใช้ได้โดย CPU 2 ครั้งใน 1 รอบของการทำงานได้ ดังนั้น การที่มีบัสของโปรแกรมแยกกันและบัสของ DMA แยกกัน ทำให้การอ่านโปรแกรม การอ่านและการเขียนข้อมูล หรือ การทำงานของ DMA ที่สามารถทำงานแบบขนานได้

3.2.2 ตารางหน่วยความจำ (Memory Maps)

ตารางหน่วยความจำจะขึ้นอยู่กับว่าจะให้โปรเซสเซอร์ประมวลผลในโหมดของโปรเซสเซอร์ หรือ ไมโครคอมพิวเตอร์ แต่ในโครงงานนี้จะให้โปรเซสเซอร์ประมวลผลในโหมดของไมโครโปรเซสเซอร์ แสดงดังรูปที่ 3.3 แอดเดรสที่ 800000h จนถึง 801FFFh จะถูกใช้สำหรับ expansion bus ซึ่งจะถูกเข้าถึงได้โดยการให้สัญญาณ/ MSTRB แอดทีฟแอดเดรสที่ 80200h จนถึง 803FFFh จะไม่ใช้แอดเดรสที่ 804000h ถึง 805FFFh จะถูกใช้สำหรับ expansion bus ซึ่งจะถูกทำให้เข้าถึงโดยการให้สัญญาณ/ IOSTRB แอดทีฟแอดเดรส 806000h ถึง 807FFFh จะไม่ใช้ ทุกๆ รีจิสเตอร์จะอยู่ทีแอดเดรส 808000h ถึง 8097FFh RAM บล็อก0 จะอยู่ที่ 809800h ถึง 809BFFh และ

บล็อกที่ 1 จะอยู่ที่ 809C00h ถึง 809FFFh แอดเดรส 80A000h จนถึง 0FFFFFFh จะถูกใช้โดยอุปกรณ์ภายนอก (เมื่อสัญญาณ /STRB แอคทีฟ)



รูปที่ 3.3 แสดงการจัดหน่วยความจำของไอซี TMS320C31

3.2.3 โหมดการอ้างอิงหน่วยความจำ (Memory Address Modes)

TMS320C31 จะแบ่งโหมดของการอ้างแอดเดรสออกเป็น 5 กลุ่ม โดยมีโหมดการอ้างแอดเดรสทั้งหมด 6 แบบ ดังนี้

1.General addressing mode

-Register : ค่าที่นำมาคำนวณจะเก็บที่ CPU รีจิสเตอร์

-Short immediate : ค่าที่นำมาคำนวณจะเป็นค่าที่โหลดโดยตรงขนาด 16 บิต จากคำสั่ง

-Direct : ค่าที่นำมาคำนวณจะถูกเก็บอยู่ในตำแหน่ง ซึ่งมีแอดเดรสขนาด 24 บิต แสดงอยู่ในคำสั่ง

-Indirect : รีจิสเตอร์ช่วยจะเก็บแอดเดรสของข้อมูล

2.Three-operand addressing mode:

-Register

-Indirect

3.Parallel addressing mode:

-Register : ข้อมูลจะอยู่ใน extended-precision register

-Indirect

4.Long-immediate addressing mode:

-Long-immediate : ข้อมูลจะเป็นค่าที่โหลดโดยตรงขนาด 24 บิต จากคำสั่ง

5.Conditional branch addressing mode:

-Register

-PC-relative : ค่าที่มีเครื่องหมายขนาด 16 บิต จะถูกบวกค่าใน PC

3.3 Internal Bus Operation

ข้อดีที่สำคัญอีกประการหนึ่งของ TMS320C31 คือการที่มีบัสภายในและสามารถประมวลผลแบบขนานได้ บัสโปรแกรมที่แยกกัน (PADDR และ PDATA) บัสข้อมูลที่แยกกัน (DADDR1 , DADDR2 และ DDATA) และบัส DMA ที่แยกกัน (DMAADDR และ DMADATA) ทำให้สามารถทำการประมวลผลแบบขนานได้ บัสเหล่านี้สามารถต่อได้กับหน่วยความจำภายใน หน่วยความจำภายนอก หรืออุปกรณ์ภายนอกอื่นๆ รูปที่ 3.2 จะแสดงการต่อบัสต่างๆ เหล่านี้

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

DADDR1 และ DADDR2 (ขนาด 24 บิต) และ DDATA (ขนาด 32 บิต) จะใช้ในการติดต่อกับหน่วยความจำภายในทุกๆ รอบของการทำงาน DDATA จะใช้ในการส่งข้อมูลไปยัง CPU โดยผ่านบัส CPU1 และ CPU2 บัส CPU1 และ CPU2 สามารถส่งข้อมูลจากหน่วยความจำ 2 ข้อมูลไปยังตัวคูณ (Multiplier) หน่วยการคำนวณทางคณิตศาสตร์ (ALU) และรีจิสเตอร์ในทุกๆ รอบของการทำงาน นอกจากนี้ภายใน CPU มีบัสรีจิสเตอร์REG1 และ REG2 ซึ่งใช้ในการหาข้อมูล 2 ค่า จากรีจิสเตอร์ไปยังคูณ และหน่วยการคำนวณทางคณิตศาสตร์ ในทุกๆ รอบของการทำงาน รูปที่ 3.2 แสดงบัสภายในต่างๆ DMA Controller จะใช้ DMAADDR (ขนาด 24 บิต) และ DMADATA (ขนาด 32 บิต) โดยบัสเหล่านี้ทำให้ DMA สามารถติดต่อกับหน่วยความจำแบบขนานได้

3.4 External Bus Operation

หน่วยความจำและอุปกรณ์ภายนอกสามารถอินเตอร์เฟสกับ TMS320C31 ได้ 2 ทาง คือ ทาง Primary Bus และ Expansion Bus การอินเตอร์เฟสกับหน่วยความจำหรืออุปกรณ์ภายนอกที่มีความเร็วต่ำกว่า TMS320C31 จะต้องเพิ่ม Wait State เข้าไป โดยอาศัยการควบคุมของ Memory Mapped Control Register และสัญญาณจากภายนอก

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

Register	Peripheral Address
Expansion-Bus Control (see subsection 7.1.2)†	808060h
Reserved	808061h
Reserved	808062h
Reserved	808063h
Primary-Bus Control (see subsection 7.1.1)	808064h
Reserved	808065h
Reserved	808066h
Reserved	808067h
Reserved	808068h
Reserved	808069h
Reserved	80806Ah
Reserved	80806Bh
Reserved	80806Ch
Reserved	80806Dh
Reserved	80806Eh
Reserved	80806Fh

† Reserved on the TMS320C31

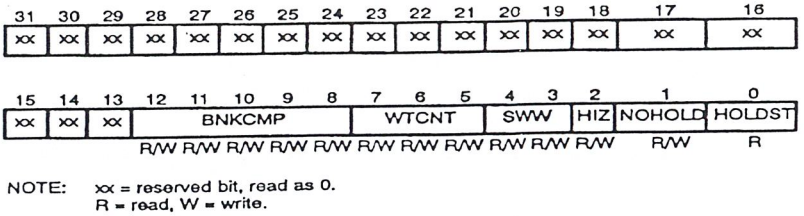
รูปที่ 3.4 memory-mapped external interface control register

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

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

-อุปกรณ์ภายนอกสามารถเข้าใช้ได้โดยให้สัญญาณ /IOSTRB เป็น 0

primary bus และ expansion bus จะมี control register ดังรูปที่ 3.5



รูปที่ 3.5 primary bus control register

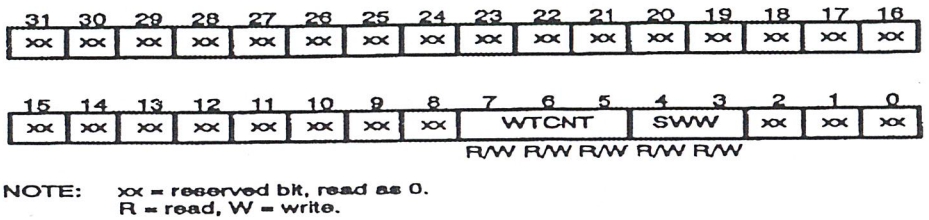
Bit	Name	Reset Value	Function
0	HOLDST	x †	Hold status bit. This bit signals whether the port is being held (HOLDST = 1) or is not being held (HOLDST = 0). This status bit is valid whether the port has been held via hardware or software.
1	NOHOLD	0	Port hold signal. NOHOLD allows or disallows the port to be held by an external $\overline{\text{HOLD}}$ signal. When NOHOLD = 1, the TMS320C3x takes over the external bus and controls it, regardless of serviced or pending requests by external devices. No hold acknowledge ($\overline{\text{HOLDA}}$) is asserted when a $\overline{\text{HOLD}}$ is received. However, it is asserted if an internal hold is generated (HIZ = 1). NOHOLD is set to 0 at reset.
2	HIZ	0	Internal hold. When set (HIZ = 1), the port is put in hold mode. This is equivalent to the external $\overline{\text{HOLD}}$ signal. By forcing a high-impedance condition, the TMS320C3x can relinquish the external memory port through software. $\overline{\text{HOLDA}}$ goes low when the port is placed in the high-impedance state. HIZ is set to 0 at reset.
4–3	SWW	11	Software wait mode. In conjunction with WTCNT, this two-bit field defines the mode of wait-state generation. It is set to 1 1 at reset.
7–5	WTCNT	111	Software wait mode. This three-bit field specifies the number of cycles to use when in software wait mode for the generation of internal wait states. The range is 0 (WTCNT = 0 0 0) to 7 (WTCNT=1 1 1) H1/H3 cycles. It is set to 1 1 1 at reset.
12–8	BNKCMP	10000	Bank compare. This five-bit field specifies the number of MSBs of the address to be used to define the bank size. It is set to 1 0 0 0 0 at reset.
31–13	Reserved	0–0	Read as 0.

† x = 0 or 1

ตารางที่ 3.2 แสดงหน้าที่ต่างๆ ของ primary bus control register

control register ของ primary bus จะมีขนาด 32 บิต ดังรูปที่ 3.5 และ ตารางที่ 3.2

control register ของ expansion bus จะมีขนาด 32 บิต ดังรูปที่ 3.6 และ ตารางที่ 3.3



รูปที่ 3.6 expansion bus control register

Bit	Name	Reset Value	Function
2–0	Reserved	000	Read as 0.
4–3	SWW	11	Software wait-state generation. In conjunction with the WTCNT, this two-bit field defines the mode of wait-state generation. It is set to 1 1 at reset.
7–5	WTCNT	111	Software wait mode. This three-bit field specifies the number of cycles to use when in software wait mode for the generation of internal wait states. The range is 0 (WTCNT = 0 0 0) to 7 (WTCNT = 1 1 1) H1/H3 clock cycles. It is set to 1 1 1 at reset.
31–8	Reserved	0–0	Read as 0.

ตารางที่ 3.3 แสดงหน้าที่ต่างๆ ของ expansion bus control register

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

ถ้าการเขียนข้อมูลเป็นการกระทำภายในโดยไม่มีการอินเตอณีเฟสกับอุปกรณีภายนอก (คือการใช้จาก CPU และ DMA) วงรอบของการเขียนข้อมูลจะใช้เพียง 1 รอบของ H1 เท่านั้น

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

1./MSTRB และ /STRB จะเป็นสถานะ high

2.(X)R/W จะเปลี่ยนสถานะที่ขอบขาขึ้นของ H1

3.ถ้าเป็นสถานะของการเขียนข้อมูล address จะเปลี่ยนสถนะที่ขอบขาขึ้นของ H1 และถ้าเป็นสถานะของการอ่านข้อมูล

3.5 คำสั่งของไอซีเบอร์ TMS320C31

คำสั่งของไอซีเบอร์ TMS320C31 เป็นคำสั่งของภาษาแอสเซมบลี้ ซึ่งเป็นคำสั่งเฉพาะของไอซีตระกูล TMS320C3x ซึ่งประกอบไปด้วยคำสั่งแบบธรรมดากับคำสั่งแบบขนาน ที่จะทำให้มีการประมวลผลที่เร็วขึ้น ซึ่งคำสั่งต่างๆจะแสดงดังตารางที่ 3.4

Mnemonic	Description	Operation
ABSF	Absolute value of a floating-point number	|*src*| → Rn
ABSI	Absolute value of an integer	|*src*| → Dreg
ADDC	Add integers with carry	*src* + Dreg + C → Dreg
ADDC3	Add integers with carry (3 operand)	*src*1 + *src*2 + C → Dreg
ADDF	Add floating-point values	*src* + Rn → Rn
ADDF3	Add floating-point values (3 operand)	*src*1 + *src*2 → Rn
ADDI	Add integers	*src* + Dreg → Dreg
ADDI3	Add integers (3 operand)	*src*1 + *src*2 + → Dreg
AND	Bitwise logical AND	Dreg AND *src* → Dreg
AND3	Bitwise logical AND (3 operand)	*src*1 AND *src*2 → Dreg
ANDN	Bitwise logical AND with complement	Dreg AND $\overline{src}$ → Dreg
ANDN3	Bitwise logical ANDN (3 operand)	*src*1 AND $\overline{src2}$ → Dreg
ASH	Arithmetic shift	If count ≥ 0: (Shifted Dreg left by count) → Dreg Else: (Shifted Dreg right by |count|) → Dreg
ASH3	Arithmetic shift (3 operand)	If count ≥ 0: (Shifted *src* left by count) → Dreg Else: (Shifted *src* right by |count|) → Dreg
B*cond*	Branch conditionally (standard)	If *cond* = true: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC → PC Else, PC + 1 → PC
B*condD*	Branch conditionally (delayed)	If *cond* = true: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC + 3 → PC Else, PC + 1 → PC
BR	Branch unconditionally (standard)	Value → PC
BRD	Branch unconditionally (delayed)	Value → PC
CALL	Call subroutine	PC + 1 → TOS Value → PC

Legend:				
	C	carry bit	C*src*	conditional-branch addressing modes
	cond	condition code	count	shift value (general addressing modes)
	Dreg	register address (any register)	PC	program counter
	Rn	register address (R7–R0)	*src*	general addressing modes
	*src*1	three-operand addressing modes	*src*2	three-operand addressing modes

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31

Mnemonic	Description	Operation
CALL*cond*	Call subroutine conditionally	If *cond* = true: PC + 1 → TOS If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC → PC Else, PC + 1 → PC
CMPF	Compare floating-point values	Set flags on Rn – *src*
CMPF3	Compare floating-point values (3 operand)	Set flags on *src*1 – *src*2
CMPI	Compare integers	Set flags on Dreg – *src*
CMPI3	Compare integers (3 operand)	Set flags on *src*1 – *src*2
DB*cond*	Decrement and branch conditionally (standard)	ARn – 1 → ARn If *cond* = true and ARn ≥ 0: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC + 1 → PC Else, PC + 1 → PC
DB*condD*	Decrement and branch conditionally (delayed)	ARn – 1 → ARn If *cond* = true and ARn ≥ 0: If C*src* is a register, C*src* → PC If C*src* is a value, C*src* + PC + 3 → PC Else, PC + 1 → PC
FIX	Convert floating-point value to integer	Fix (*src*) → Dreg
FLOAT	Convert integer to floating-point value	Float(*src*) → Rn
IACK	Interrupt acknowledge	Dummy read of *src* IACK toggled low, then high
IDLE	Idle until interrupt	PC + 1 → PC Idle until next interrupt
LDE	Load floating-point exponent	*src*(exponent) → Rn(exponent)
LDF	Load floating-point value	*src* → Rn
LDF*cond*	Load floating-point value conditionally	If *cond* = true, *src* → Rn Else, Rn is not changed
LDFI	Load floating-point value, interlocked	Signal interlocked operation *src* → Rn
LDI	Load integer	*src* → Dreg
LDI*cond*	Load integer conditionally	If *cond* = true, *src* → Dreg Else, Dreg is not changed

Legend:

ARn	auxiliary register n (AR7–AR0	Rn	register address (R7 — R0)
C*src*	conditional-branch addressing modes	*src*	general addressing modes
cond	condition code	*src*1	three-operand addressing modes
Dreg	register address (any register)	*src*2	three-operand addressing modes
PC	program counter	TOS	top of stack

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
LDII	Load integer, interlocked	Signal interlocked operation *src* → Dreg
LDM	Load floating-point mantissa	*src* (mantissa) → Rn (mantissa)
LSH	Logical shift	If count ≥ 0: (Dreg left-shifted by count) → Dreg Else: (Dreg right-shifted by |count|) → Dreg
LSH3	Logical shift (3-operand)	If count ≥ 0: (*src* left-shifted by count) → Dreg Else: (*src* right-shifted by |count|) → Dreg
MPYF	Multiply floating-point values	*src* × Rn → Rn
MPYF3	Multiply floating-point value (3 operand)	*src*1 × *src*2 → Rn
MPYI	Multiply integers	*src* × Dreg → Dreg
MPYI3	Multiply integers (3 operand)	*src*1 × *src*2 → Dreg
NEGB	Negate integer with borrow	0 – *src* – C → Dreg
NEGF	Negate floating-point value	0 – *src* → Rn
NEGI	Negate integer	0 – *src* → Dreg
NOP	No operation	Modify ARn if specified
NORM	Normalize floating-point value	Normalize (*src*) → Rn
NOT	Bitwise logical complement	$\overline{src}$ → Dreg
OR	Bitwise logical OR	Dreg OR *src* → Dreg
OR3	Bitwise logical OR (3 operand)	*src*1 OR *src*2 → Dreg
POP	Pop integer from stack	*SP–– → Dreg
POPF	Pop floating-point value from stack	*SP–– → Rn
PUSH	Push integer on stack	Sreg → *++ SP
PUSHF	Push floating-point value on stack	Rn → *++ SP

Legend:

ARn	auxiliary register n (AR7–AR0)	SP	stack pointer
C	carry bit	Sreg	register address (any register)
Dreg	register address (any register)	*src*	general addressing modes
PC	program counter	*src*1	3-operand addressing modes
Rn	register address (R7–R0)	*src*2	3-operand addressing modes

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
RETI*cond*	Return from interrupt conditionally	If *cond* = true or missing: *SP–– → PC 1 → ST (GIE) Else, continue
RETS*cond*	Return from subroutine conditionally	If *cond* = true or missing: *SP–– → PC Else, continue
RND	Round floating-point value	Round (*src*) → Rn
ROL	Rotate left	Dreg rotated left 1 bit → Dreg
ROLC	Rotate left through carry	Dreg rotated left 1 bit through carry → Dreg
ROR	Rotate right	Dreg rotated right 1 bit → Dreg
RORC	Rotate right through carry	Dreg rotated right 1 bit through carry → Dreg
RPTB	Repeat block of instructions	*src* → RE 1 → ST (RM) Next PC → RS
RPTS	Repeat single instruction	*src* → RC 1 → ST (RM) Next PC → RS Next PC → RE
SIGI	Signal, interlocked	Signal interlocked operation Wait for interlock acknowledge Clear interlock
STF	Store floating-point value	Rn → Daddr
STFI	Store floating-point value, interlocked	Rn → Daddr Signal end of interlocked operation
STI	Store integer	Sreg → Daddr
STII	Store integer, interlocked	Sreg → Daddr Signal end of interlocked operation
SUBB	Subtract integers with borrow	Dreg – *src* – C → Dreg

Legend:

C	carry bit	RM	repeat mode bit
cond	condition code	RS	repeat start register
Daddr	destination memory address	Rn	register address (R7–R0)
Dreg	register address (any register)	SP	stack pointer
GIE	global interrupt enable register	ST	status register
PC	program counter	Sreg	register address (any register)
RC	repeat counter register	*src*	general addressing modes
RE	repeat interrupt register		

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
SUBB3	Subtract integers with borrow (3 operand)	*src*1 – *src*2 – C → Dreg
SUBC	Subtract integers conditionally	If Dreg – *src* ≥ 0: [(Dreg – *src*) << 1] OR 1 → Dreg Else, Dreg << 1 → Dreg
SUBF	Subtract floating-point values	Rn – *src* → Rn
SUBF3	Subtract floating-point values (3 operand)	*src*1 – *src*2 → Rn
SUBI	Subtract integers	Dreg – *src* → Dreg
SUBI3	Subtract integers (3 operand)	*src*1 – *src*2 → Dreg
SUBRB	Subtract reverse integer with borrow	*src* – Dreg – C → Dreg
SUBRF	Subtract reverse floating-point value	*src* – Rn → Rn
SUBRI	Subtract reverse integer	*src* – Dreg → Dreg
SWI	Software interrupt	Perform emulator interrupt sequence
TRAP*cond*	Trap conditionally	If *cond* = true or missing: Next PC → * ++ SP Trap vector N → PC 0 → ST (GIE) Else, continue
TSTB	Test bit fields	Dreg AND *src*
TSTB3	Test bit fields (3 operand)	*src*1 AND *src*2
XOR	Bitwise exclusive OR	Dreg XOR *src* → Dreg
XOR3	Bitwise exclusive OR (3 operand)	*src*1 XOR *src*2 → Dreg

Legend:					
	C	carry bit	Rn	register address (R7–R0)
	cond	condition code	SP	stack pointer
	Dreg	register address (any register)	*src*	general addressing modes
	GIE	global interrupt enable register	*src*1	3-operand addressing modes
	N	any trap vector 0–27	*src*2	3-operand addressing modes
	PC	program counter	ST	status register

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
	Parallel Arithmetic With Store Instructions	
ABSF || STF	Absolute value of a floating point	|*src2*| → *dst1* || *src3* → *dst2*
ABSI || STI	Absolute value of an integer	|*src2*| → *dst1* || *src3* → *dst2*
ADDF3 || STF	Add floating point	*src1* + *src2* → *dst1* || *src3* → *dst2*
ADDI3 || STI	Add integer	*src1* + *src2* → *dst1* || *src3* → *dst2*
AND3 || STI	Bitwise logical AND	*src1* AND *src2* → *dst1* || *src3* → *dst2*
ASH3 || STI	Arithmetic shift	If count ≥ 0: *src2* << count → *dst1* || *src3* → *dst2* Else: *src2* >> |count| → *dst1* || *src3* → *dst2*
FIX || STI	Convert floating point to integer	Fix(*src2*) → *dst1* || *src3* → *dst2*
FLOAT || STF	Convert integer to floating point	Float(*src2*) → *dst1* || *src3* → *dst2*
LDF || STF	Load floating point	*src2* → *dst1* || *src3* → *dst2*
LDI || STI	Load integer	*src2* → *dst1* || *src3* → *dst2*
LSH3 || STI	Logical shift	If count ≥ 0: *src2* << count → *dst1* || *src3* → *dst2* Else: *src2* >> |count| → *dst1* || *src3* → *dst2*
MPYF3 || STF	Multiply floating point	*src1* x *src2* → *dst1* || *src3* → *dst2*
MPYI3 || STI	Multiply integer	*src1* x *src2* → *dst1* || *src3* → *dst2*

Legend: count register addr (R7–R0); *dst1* register addr (R7–R0); *dst2* indirect addr (disp = 0, 1, IR0, IR1); *src1* register addr (R7–R0); *src2* indirect addr (disp = 0, 1, IR0, IR1); *src3* register addr (R7–R0)

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

Mnemonic	Description	Operation
	Parallel Arithmetic With Store Instructions (Concluded)	
NEGF || STF	Negate floating point	0– *src2* → *dst1* || *src3* → *dst2*
NEGI || STI	Negate integer	0 – *src2* → *dst1* || *src3* → *dst2*
NOT || STI	Complement	$\overline{src1}$ → *dst1* || *src3* → *dst2*
OR3 || STI	Bitwise logical OR	*src1* OR *src2* → *dst1* || *src3* → *dst2*
STF || STF	Store floating point	*src1* → *dst1* || *src3* → *dst2*
STI || STI	Store integer	*src1* → *dst1* || *src3* → *dst2*
SUBF3 || STF	Subtract floating point	*src1* – *src2* → *dst1* || *src3* → *dst2*
SUBI3 || STI	Subtract integer	*src1* – *src2* → *dst1* || *src3* → *dst2*
XOR3 || STI	Bitwise exclusive OR	*src1* XOR *src2* → *dst1* || *src3* → *dst2*
	Parallel Load Instructions	
LDF || LDF	Load floating point	*src2* → *dst1* || *src4* → *dst2*
LDI || LDI	Load integer	*src2* → *dst1* || *src4* → *dst2*
	Parallel Multiply And Add/Subtract Instructions	
MPYF3 || ADDF3	Multiply and add floating point	op1 x op2 → op3 || op4 + op5 → op6
MPYF3 || SUBF3	Multiply and subtract floating point	op1 x op2 → op3 || op4 – op5 → op6
MPYI3 || ADDI3	Multiply and add integer	op1 x op2 → op3 || op4 + op5 → op6
MPYI3 || SUBI3	Multiply and subtract integer	op1 x op2 → op3 || op4 – op5 → op6

Legend:

dst1 register addr (R7–R0)
dst2 indirect addr (disp = 0, 1, IR0, IR1)
op1, op2, op4, and op5 Any two of these operands must be specified using register addr; the remaining two must be specified using indirect.
op3 register addr (R0 or R1)
op6 register addr (R2 or R3)
src1 register addr (R7–R0)
src2 indirect addr (disp = 0, 1, IR0, IR1)
src3 register addr (R7–R0)

ตารางที่ 3.4 แสดงคำสั่งของไอซี TMS320C31(ต่อ)

บทที่ 4
การเชื่อมต่อระหว่าง TMS320C31 กับ TLC32040 AIC

4.1 ลักษณะของ TLC32040 AIC

-ใช้ในการแปลง อนาลอกเป็นดิจิตอล และ ดิจิตอลเป็นอนาลอก ขนาด 14 บิต

-สามารถเปลี่ยนแปลงอัตราการแซมปลิ้ง (Sampling) ทั้งของอนาลอกเป็นดิจิตอล และดิจิตอลเป็น อนาลอก ด้วยความแม่นยำที่ 20,000 ตัวอย่างต่อวินาที

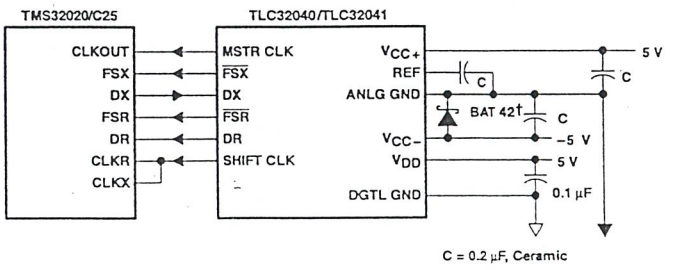
-การติดต่อระหว่าง TLC32040 AIC กับ TMS320C31 จะมีการติดต่อผ่านทาง พอร์ตอนุกรม (Serial Port)

-ใช้เทคโนโลยี ซีมอส (CMOS)

-สามารถเลือกใช้ ช่องอนาลอกอินพุทช่วยเหลือ (auxiliary analog input channel) ได้

บนบอร์ดการเชื่อมต่อทาง Serial port ระหว่าง TMS320C31 กับ TLC32040 AIC จะมีตัวหัวไว้เชื่อมตัว (เป็นลักษณะคล้ายกับ Jumper) เมื่อไม่มีการเชื่อมต่อก็คือการไม่ใช้ TLC32040 AIC แต่ถ้ามีการต่อก็จะเป็นการเชื่อมต่อและใช้งานระหว่าง TMS320C31 กับ TLC32040 AIC

จากตัว TLC32040 AIC จะเชื่อมต่อ analog input และ analog output กับหัวต่อแบบ RCA โดยสัญญาณที่เข้าและออกผ่านทางหัวต่อ RCA นี้จะมีระดับแรงดัน + / - 3 V peak ที่หัวต่อเอาท์พุทสามารถเชื่อมต่อโดยตรงกับ ลำโพงก็ได้ แต่อาจไม่มีระดับสูงพอ จึงอาจจะต้องมีวงจรขยายสัญญาณภายนอก หรือต่อกับลำโพงแบบ high impedance



รูปที่ 4.1 แสดงการเชื่อมต่อระหว่าง TMS320C31 กับ TLC32040

4.2 รายละเอียดของแต่ละขาของ TLC32040 AIC ที่ใช้เชื่อมต่อกับ TMS320C31

RESET ที่ขานี้จะเป็นขาอินพุทที่ใช้ในการรีเซ็ทตัว TLC32040

SCLK (shift clock) เป็นคล็อกที่ใช้ในการส่งข้อมูลแบบอนุกรม (ผ่านทาง serial port) ความถี่นี้จะเกิดจาก Master Clock Signal Frequency หรือ MCLK หารด้วย 4

FSR (Frame Sync receive) ในหมวดการส่งข้อมูลแบบอนุกรม เมื่อขา FSR เป็น 0 จะทำให้พอร์ตอนุกรมของ TMS320C31 เริ่มรับค่าจากขา DR ของตัว TLC32040 AIC ค่าของบิต DR จะแสดงผลบนขา DR ก่อนที่ขา FSR จะเป็น 0

DR (Data Receive) ใช้สำหรับการส่งข้อมูลจากอนาลอกเป็นดิจิตอล จาก TLC32040 AIC ไปทางพอร์ตอนุกรมของ TMS320C31 การส่งข้อมูลที่ขานี้ต้องทำการซิงโครไนซ์ (Synchronized) กับ SCLK (Shift Clock Signal)

FSX (Frame Sync Transmit) เหมือน FSR แต่กลับกันเป็นการส่งค่า จากพอร์ตอนุกรมของ TMS320C31 ไปยังขา DX ของ TLC32040 AIC

DX (Data Transmit) เหมือน DR แต่กลับกันตรงที่เป็นตัวรับค่าดิจิตอลเป็นอนาลอก จากพอร์ตอนุกรมของ TMS320C31 โดยการส่งข้อมูลที่ขานี้ต้องทำการซิงโครไนซ์ (Synchronized) กับ SCLK (Shift Clock Signal)

MCLK(Master clock frequency) ขาที่ 6 ถูกใช้เป็นสัญญาณทางลอจิกที่สำคัญแก่ TLC32040 AIC โดยสัญญาณคล็อกที่ป้อนเข้ามาที่ขานี้ใช้สำหรับการแปลงเป็นสัญญาณต่างๆที่ใช้ใน TLC32040 AIC เช่น Shift Clock Signal , Switch Capacitor Filter Clock และ สัญญาณการแปลงอนาลอกเป็นดิจิตอล และ ดิจิตอลเป็นอนาลอก

IN+ (Non-inverting Input) เป็นอินพุทของสัญญาณอนาลอกที่ไม่กลับเฟส

IN- (Inverting Input) เป็นอินพุทของสัญญาณอนาลอกที่กลับเฟส (ในที่นี้จะไม่ใช้อินพุทตัวนี้ เลยทำการต่อกับกราวนด์

OUT+ (Non-inverting Output) เป็นเอาท์พุทที่ไม่กลับเฟสของสัญญาณอนาลอก

OUT- (Inverting Output) เป็นเอาท์พุทที่กลับเฟสของสัญญาณอนาลอก

VCC (Voltage Supply) เป็นสัญญาณไฟบวกซึ่งป้อนให้แก่ TLC32040 AIC มีขนาดแรงดัน 5 โวลต์

4.3 รายละเอียดของแต่ละขาของ TMS320C31 ที่ใช้เชื่อมต่อกับ TLC32040

XF0 เป็นขาที่ใช้ส่งสัญญาณในการที่จะทำการรีเซ็ท ตัว TLC32040 AIC

CLKR0 เป็นขาที่ใช้ในการรับคล็อกที่ใช้ในการรับมูลแบบอนุกรม โดยเชื่อมต่อกับ SCLK(Shift Clock) ของ TLC32040 เพื่อให้ซิงโครไนซ์(Synchronized) กัน

CLKX0 เป็นขาที่ใช้ในการรับคล็อกที่ใช้ในการส่งมูลแบบอนุกรม โดยเชื่อมต่อกับ SCLK(Shift Clock) ของ TLC32040 เพื่อให้ซิงโครไนซ์(Synchronized) กัน

FSR0 เป็นการรับสัญญาณว่าจะมีการส่งข้อมูลมาจาก TLC32040 AIC ในการแปลงอนาลอกเป็นดิจิตอล

DR0 เป็นขาที่ใช้สำหรับรับข้อมูลจาก TLC32040 AIC

FSX0 เป็นขาที่ใช้ในการรับสัญญาณจาก TLC32040 เมื่อพร้อมที่จะรับค่าจาก TMS320C31 ในการแปลงจากดิจิตอลเป็นอนาลอก

DX0 เป็นขาที่ใช้ในการส่งข้อมูลจาก TMS320C31 ไปยัง TLC32040 AIC

TCLK0 ขาที่ใช้ในการส่งสัญญาณนาฬิกาจาก TMS320C31 เพื่อให้ TLC32040 AIC ไปใช้เป็น MCLK (Master Clock Frequency)

บทที่ 5
การเชื่อมต่อกันระหว่างคอมพิวเตอร์และ TMS320C31

การทำงานของโปรแกรมนั้นจะต้องมีการทำงานร่วมกันระหว่างโปรแกรม 2 ส่วน คือ ส่วนรับค่าจากสัญญาณแล้วมาคำนวณ และ อีกส่วนหนึ่งก็คือส่วนที่จะทำการแสดงผล

ซึ่งจะประกอบไปด้วย

1. โปรแกรมในส่วนของ TMS320C31 ซึ่งจะทำหน้าที่ รับค่าจากสัญญาณและคำนวณค่า
2. โปรแกรมในส่วนของคอมพิวเตอร์ซึ่งจะทำหน้าที่รับค่าจาก TMS320C31 มาทำการแสดงผล

ซึ่งทั้งสองส่วนนี้จะต้องทำการส่งข้อมูลระหว่างกันเกือบตลอดเวลา การเชื่อมต่อกันระหว่าง 2 ตัวนี้จะเชื่อมต่อกันโดยใช้ พอร์ทขนาน ของ TMS320C31 โดยจะเขียนหรืออ่านข้อมูลจากรีจิสเตอร์ควบคุมพอร์ทขนานและรีจิสเตอร์สถานะของคอมพิวเตอร์ซึ่งจะควบคุมการทำงานด้วยโปรแกรมบนคอมพิวเตอร์

7	6	5	4	3	2	1	0
DIR0	X	DIR1	INT	$\overline{\text{SLCTIN}}$	INIT	$\overline{\text{AUTOFD}}$	$\overline{\text{PSTROBE}}$
					RESET		HPSTB
		W	R/W	W	R/W	W	R/W

รูปที่ 5.1 แสดงบิทของรีจิสเตอร์ควบคุมพอร์ตขนาน

7	6	5	4	3	2	1	0
$\overline{\text{BUSY}}$	ACK	PAPER	SELECT	ERROR	ACK	X	X
D3	D2	D1	D0	HPACK			

รูปที่ 5.2 แสดงบิทของรีจิสเตอร์สถานะของพอร์ตขนาน

การส่งข้อมูลจากคอมพิวเตอร์ไป TMS320C31 สามารถใช้งานได้หลายแบบดังนี้

1. คอมพิวเตอร์เขียนข้อมูลลงในพื้นที่ในการอินพุทและเอาท์พุทของพอร์ทขนาน

2. คอมพิวเตอร์ให้สัญญาณ HPSTB เป็น 0 และรอสัญญาณเพื่ออนุญาตให้มีการอินเตอร์รัพโดยจะส่งมาทาง INT2 ซึ่งสัญญาณนี้จะใช้สำหรับการกำหนดค่าเริ่มต้นของระบบ นอกนั้นการทำงานของสัญญาณ INT2 จะไม่การนำมาใช้ในการทำงานอื่นๆ
3. TMS320C31 เริ่มมีสภาวะการคอยที่ตำแหน่ง 0xFFF000 ซึ่งจะสามารถถอดรหัสได้เป็นตำแหน่ง ของ HPACK ซึ่ง คอมพิวเตอร์จะใช้ สัญญาณ HPACK เพื่อที่ทำให้ TMS320C31 ทำการหยุดรอรับข้อมูล
4. คอมพิวเตอร์จะทำการส่ง HPSTB เพื่อบอกให้ TMS320C31 พร้อมที่จะเริ่มทำงานได้

การส่งข้อมูลจาก TMS320C31 ไปยัง คอมพิวเตอร์ สามารถใช้งานได้หลายแบบดังนี้

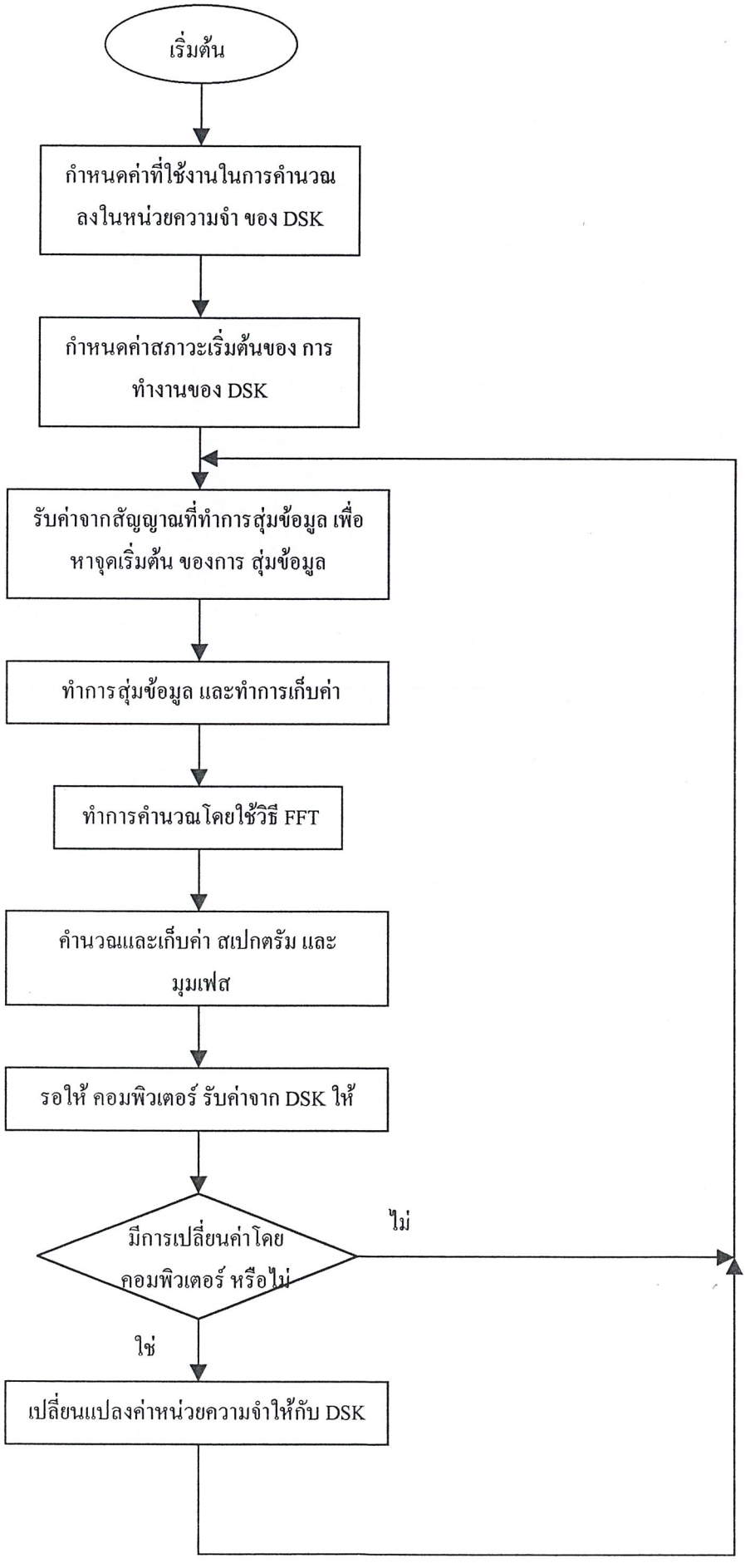
1. คอมพิวเตอร์จะทำการรอรับค่าจาก สัญญาณ HPACK เนื่องจาก TMS320C31 ไม่เข้าใจการสั่งการทำงานของคอมพิวเตอร์
2. TMS320C31 เริ่มมีสภาวะการคอยที่ตำแหน่ง 0xFFF000 ซึ่งจะสามารถถอดรหัสได้เป็นตำแหน่ง ของ HPACK ซึ่ง คอมพิวเตอร์จะใช้ สัญญาณ HPACK เพื่อที่ทำให้ TMS320C31 ทำการพร้อมที่จะส่งข้อมูล
3. ขณะที่คอมพิวเตอร์รับสัญญาณ HPIA คอมพิวเตอร์จะให้สัญญาณ PSTROBE เป็น 0 และจะอ่านค่า 4 บิท ซึ่งจะผ่านทางพอร์ทขนาน
4. คอมพิวเตอร์จะทำการส่ง HPSTB เพื่อบอกว่าได้อ่านค่าจาก TMS320C31 เสร็จสิ้นการทำงานแล้ว

5.1ขั้นตอนในการทำงานของโปรแกรมบน TMS320C31

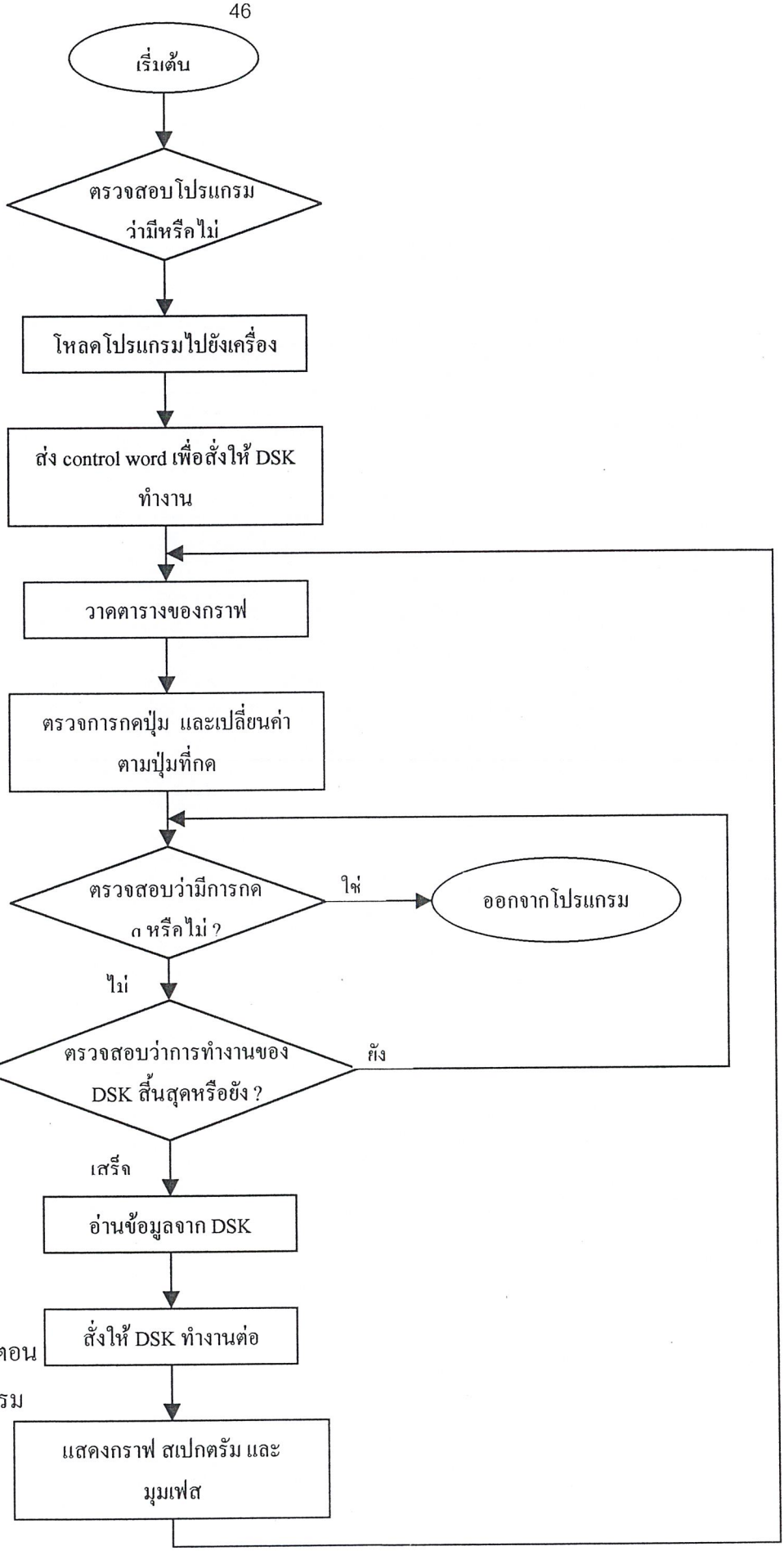
เริ่มต้นจะกำหนดค่าบนเมมโมรี่ บน DSK และ เซตค่าต่างๆ เพื่อทำการรับค่า หลังจากนั้นจะทำการรับค่าและเช็คว่าจะเริ่มต้นรับค่าที่จุดใดและจะเริ่มรับค่าไปเท่ากับจำนวนที่กำหนดไว้ แล้วนำค่าที่ได้ไปทำการคำนวณหาค่าตามทฤษฎี FFT ในรูปที่ 5.3 เป็นรูปขั้นตอนการทำงานของการคำนวณ FFT แล้วนำค่าที่ได้นั้นไปทำการคำนวณหาขนาดและมุมเฟสของสเปกตรัมและนำไปจัดลำดับการเก็บค่าใหม่ รอจนกว่า COMPUTER จะรับค่าจาก DSK ทั้งหมดก่อนแล้วเช็คค่าว่า COMPUTER สั่งให้มีการเปลี่ยนแปลงค่าหรือไม่ถ้ามีก็เปลี่ยนแปลงค่าให้กับเมมโมรี่ แต่ถ้าไม่มีก็ให้ไปรับค่าต่อไปได้เลยโดยรูปที่ 5.1 จะแสดงขั้นตอนการทำงานของโปรแกรมทั้งหมดของ TMS320C31

5.2 ขั้นตอนในการทำงานของโปรแกรมบน COMPUTER

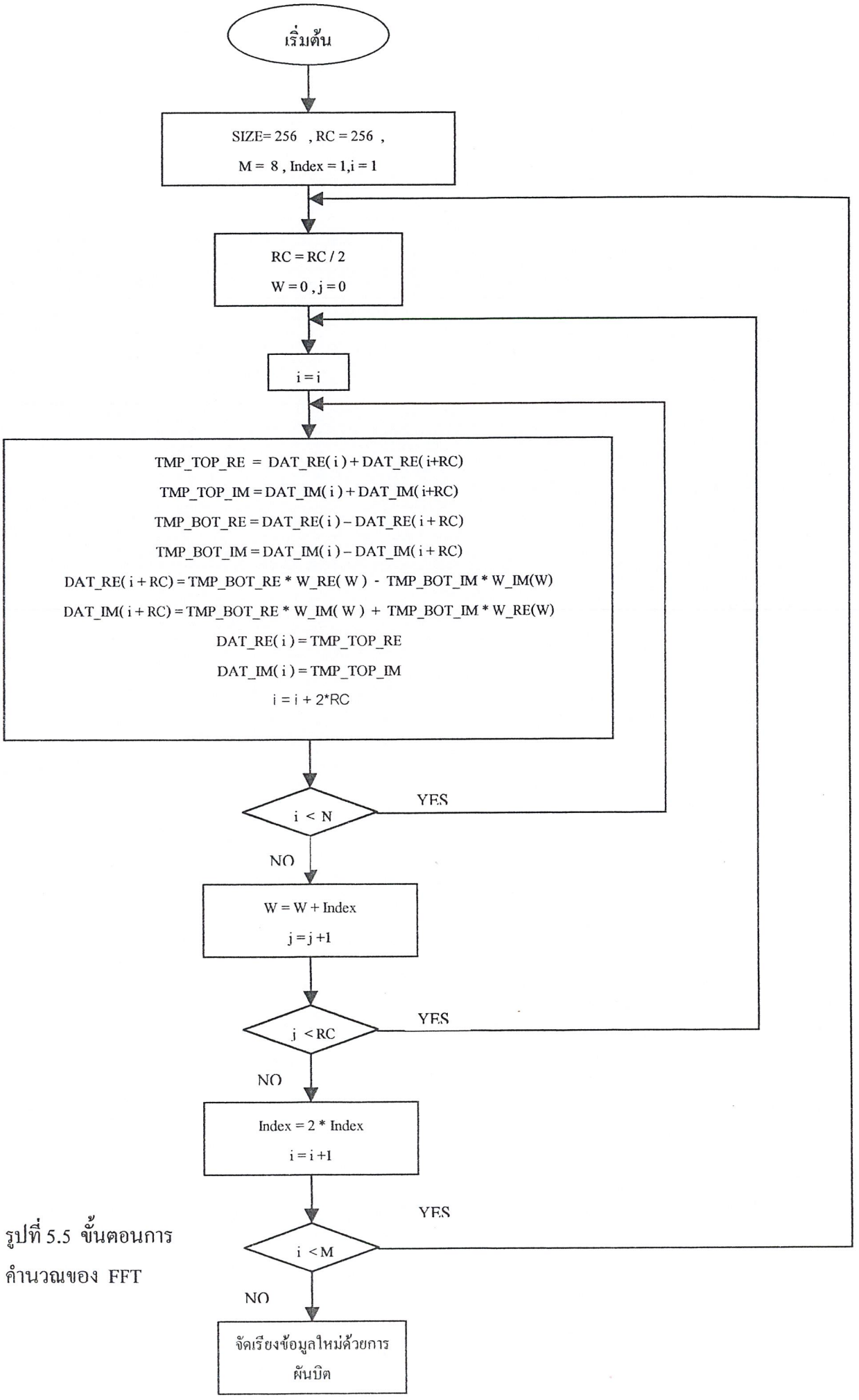
เริ่มต้นจะเช็คว่ามีโปรแกรมที่ต้องการหรือไม่แล้วจึง load โปรแกรมลงไปยัง DSK แล้วทำการสั่งให้ ตัว DSK ทำงาน แล้วจะทำการวาดกราฟเพื่อรอค่าจาก DSK หลังจากนั้นจะทำการรับค่า KEY ที่กดว่ามีการกดอะไรบ้าง และถ้ามีการกด Q ก็จะออกจากโปรแกรมทันที แต่ถ้าไม่ก็จะทำงานต่อ โดยจะไปตรวจสอบว่า DSK ทำงานเสร็จหรือยังเมื่อเสร็จแล้วก็จะทำการรับค่าจาก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



รูปที่ 5.3 แสดงขั้นตอนการทำงานของโปรแกรมบน DSK



รูปที่ 5.4 รูปแสดงขั้นตอนการทำงานของโปรแกรมบนคอมพิวเตอร์



รูปที่ 5.5 ขั้นตอนการคำนวณของ FFT

บทที่ 6
การทดลองและผล

จากสัญญาณต่างๆทั่วไปจะประกอบไปด้วยสัญญาณที่มีความถี่ต่างๆมาประกอบกัน ซึ่งเมื่อสัญญาณนั้นผ่านกระบวนการวิเคราะห์ฟาสต์ฟูเรียร์ (Fast Fourier Transform) แล้ว จะแสดงผลออกมาเป็นแถบของความถี่หรือสเปกตรัมของสัญญาณ

จาก TMS320C31 ได้ทำการเขียนโปรแกรมภาษาแอซเซมบลี้ของเครื่อง ในการติดต่อกับตัวเปลี่ยนสัญญาณอนาลอกเป็นดิจิตอล TLC32040 AIC เพื่อทำการสุ่มข้อมูล (Sampling) จากสัญญาณที่เราต้องการดูแถบความถี่หรือสเปกตรัมของสัญญาณ และนำค่าที่ได้จากการสุ่มข้อมูลนี้มาทำการประมวลผล โดยวิธีการวิเคราะห์ฟาสต์ฟูเรียร์ จากนั้น นำค่าผลลัพธ์ที่ได้จากการคำนวณนี้ไปเก็บไว้ที่หน่วยความจำ แล้วคอมพิวเตอร์จะทำการนำค่าที่ได้ไปแสดงผลทางจอคอมพิวเตอร์

ในการทดลองเราได้ทำการป้อนสัญญาณรูปต่างๆให้กับ TLC32040 AIC ทำการสุ่มข้อมูล จากนั้นก็ทำการประมวลผลโดย TMS320C31 ซึ่งค่าที่ผ่านการประมวลผลจะนำมาเขียนเป็นลักษณะสเปกตรัมของสัญญาณ ที่ป้อนเข้าสู่เครื่อง โดยจะจัดทำการแสดงผลออกมาเป็น 2 ส่วน คือ ส่วนแรกเป็นสเปกตรัม โดยมีแกน Y เป็นแอมปลิจูด หรือ ขนาดของสัญญาณ และแกน X เป็นความถี่ของสัญญาณ และ ส่วนที่สองคือ มุมเฟสของสเปกตรัม โดยมีแกน Y เป็นมุมเฟส (องศา) ส่วนแกน X เป็นตำแหน่งความถี่ ตามที่กล่าวมาแล้วในบทที่ 2 ในสมการ 2.13 และสมการ 2.14

ในโปรแกรมเพิ่มส่วนในการเลือกจุดเริ่มต้นของการแซมปลิ้งสัญญาณ โดยเริ่มที่จุดที่มีการเปลี่ยนแปลงค่าจากค่าลบเป็นค่าบวก และจะเริ่มทำการแซมปลิ้งต่อจนครบ 256 จุด เช่น สัญญาณคลื่นสี่เหลี่ยมก็จะเริ่มต้นแซมปลิ้งสัญญาณที่ ค่าบวก เพื่อที่จะทำให้การทำงานรอบต่อ ๆ ไปมีการแซมปลิ้งสัญญาณที่จุดเริ่มต้นเดียวกัน ทำให้มุมเฟสที่ได้มีค่าคงที่มากขึ้น

ในโปรแกรมการแสดงผลของสเปกตรัมนี้ จะแสดงเป็นลักษณะของ Hz/div คือ ต่อจำนวนช่อง คล้ายกับของออสซิลโลสโคป ส่วนขนาด หรือ Amplitude ก็เช่นกัน จะแสดงเป็น Volts/div คือ ต่อช่องเช่นเดียวกับความถี่

ในโปรแกรมที่ใช้วิเคราะห์นี้สามารถเปลี่ยนแปลงความถี่ในการสุ่มข้อมูลได้ โดยอาศัยตัวแปร 3 ตัว คือ

RA (receive register A) เป็นรีจิสเตอร์ที่ใช้สำหรับในการเปลี่ยนแปลงค่าความถี่ของการสุ่มข้อมูล หรือการแปลงอนาลอกเป็นดิจิตอล โดยค่า RA นี้จะมีค่าอยู่ในช่วง 4 ถึง 31

RB (receive register B) เป็นรีจิสเตอร์ที่ใช้สำหรับในการเปลี่ยนแปลงค่าความถี่ของการสุ่มข้อมูลหรือการแปลงอนาลอกเป็นดิจิตอลเช่นเดียวกับ RA โดยที่ RBนี้จะมีค่าอยู่ในช่วง 12 ถึง 61

T0_prdv (Time Period) เป็นตัวเปลี่ยนแปลงคาบเวลาของสัญญาณนาฬิกา ทำให้สัญญาณนาฬิการที่ TMS320C31 ส่งมีการเปลี่ยนแปลงคาบเวลา ทำให้ความถี่ลดลง โดยค่า T0_prdv นี้จะมีค่าอยู่แค่ 1 กับ 2 เท่านั้น ดังในบทที่ 4

จากตัวแปรทั้ง 3 ตัวนี้จะทำการเปลี่ยนแปลงค่าความถี่ในการสุ่มข้อมูลตามสมการ (6.1)

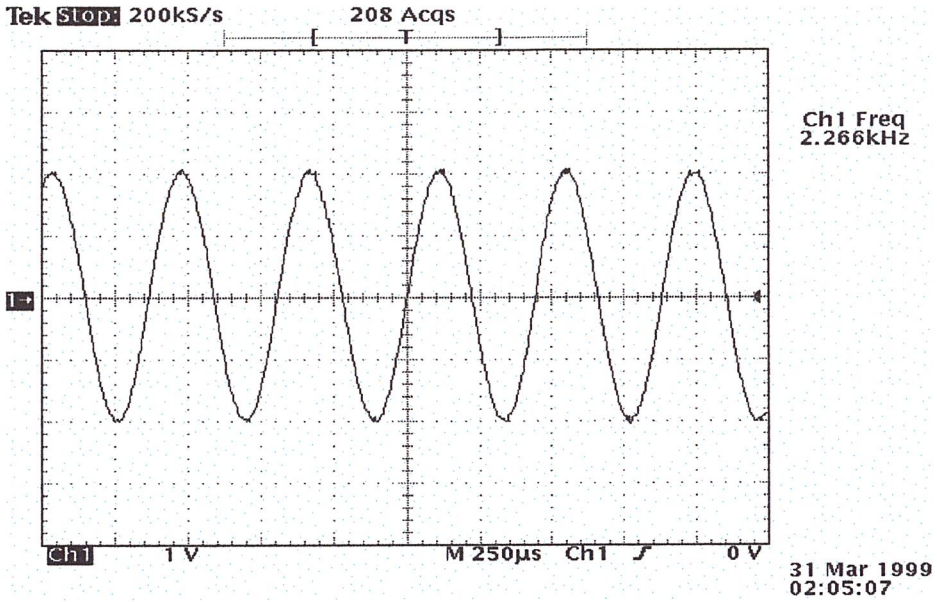
$$\text{conversion\_frequency} = \text{MCLK} / (2 \times \text{RA} \times \text{RB} \times \text{T0\_prdv}) \qquad (6.1)$$

โดยที่ MCLK (Master Clock Frequency) นี้จะมีค่าเท่ากับ 12.5 MHz ค่าเริ่มต้นของโปรแกรม RA=10 ,RB=14 และ T0_prdv=1 ทำให้ได้ค่า conversion frequency = 44642.85 Hz จากโปรแกรมเป็นการวิเคราะห์ทั้งหมด 256 จุด การแสดงค่านั้นจะแสดงได้เพียงแค่ครึ่งหนึ่งของความถี่การสุ่มข้อมูล ดังนั้นจะวัดสัญญาณได้ที่ความถี่ 22321.42 Hz เมื่อมาเทียบสเกลเป็นช่อง ช่องละ13 แถบความถี่ มีทั้งหมด 10 ช่อง จะได้เป็นความถี่ต่อช่อง หรือ Hz/div เป็น 2267.02 โดย Hz/div ที่ได้นี้เป็นความถี่ที่ใช้ดูว่า แถบความถี่ที่ปรากฏมีความถี่เท่าไหร่

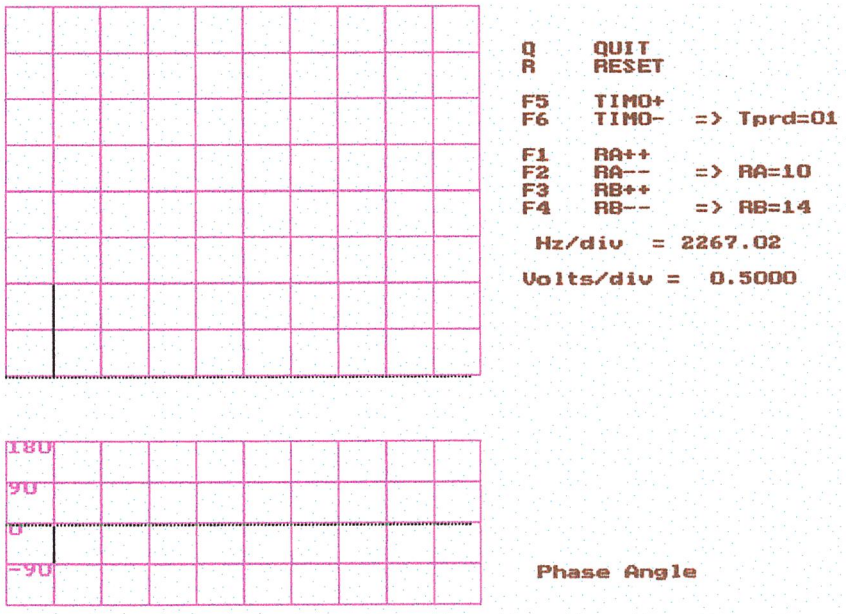
โดยในการเปลี่ยนแปลงค่าต่างๆ จากข้างต้น ทำได้โดยการกดปุ่มตามเมนูทางด้านขวาของโปแรแกรมเพื่อเปลี่ยนค่าตัวแปรตามต้องการทำให้ค่าความถี่ต่อช่องเปลี่ยนแปลง ซึ่งเป็นผลมาจากความถี่ในการสุ่มข้อมูลเปลี่ยนแปลง จะทำให้ดูสัญญาณตามช่วงความถี่ได้ง่าย ดังผลการทดลองเกี่ยวกับการทดลองกับสัญญาณโทรศัพท์ จะมี Hz/div ลดลงเป็น 179.62 Hz สามารถทำได้โดยการกดปุ่ม F1 เพื่อทำการเพิ่มค่า RA ลงทีจนถึง 31 และกดปุ่ม F3 เพื่อทำการเพิ่มค่า RB จนเท่ากับ 57 และเมื่อต้องการลดค่า RA และ RB ก็ทำโดยกด F2 และ F4 ตามลำดับ ส่วนของค่า T0_prdv ก็ทำได้โดยการกด F5 และ F6

จากโปรแกรมเมื่อขนาดของสัญญาณมีขนาดเล็ก ขนาดของแถบความถี่จะมีขนาดเล็กด้วย จึงได้เพิ่มปุ่ม G โดยเมื่อกดปุ่ม G จะเป็นการเปลี่ยนค่า Volts/div โดยจะลดลงจาก 1 โวลต์ต่อช่องเป็น 0.5 โวลต์ต่อช่อง และจะลดลงอีกเป็น 0.25 และ 0.2 โวลต์ต่อช่อง เมื่อกดอีกทีก็จะกลับเป็น 1 โวลต์ตามเดิม เพื่อใช้ในการแสดงค่าของสเปกตรัมที่มีค่าขนาดของสัญญาณต่ำให้แสดงออกมาเห็นได้ชัด ดังในผลการทดลองต่างๆ ที่ค่า Volt/div เปลี่ยนแปลงเพื่อให้สามารถดูได้ง่าย

จากรูปที่ 6.1 เป็นสัญญาณที่ป้อนให้กับ TLC32040 ทำการสุ่มข้อมูลมาทำการประมวลผลโดยเป็นสัญญาณซายน์ที่ความถี่ 2.266 kHz ที่มีขนาดของสัญญาณ 2 โวลต์ และรูปที่ 6.2 แสดงสเปกตรัมของสัญญาณซายน์ที่ป้อนเข้าไปทำการวิเคราะห์

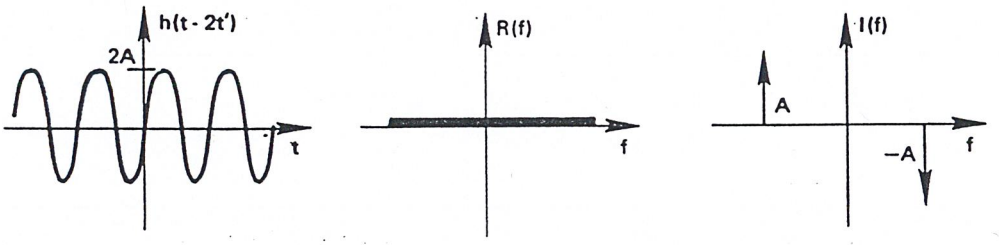


รูปที่ 6.1 สัญญาณซายน์ ความถี่ 2.266 kHz มีขนาด Vp-p 4 Volts



รูปที่ 6.2 สเปกตรัมของสัญญาณคลื่นรูปซายน์ที่มีความถี่ 2.266 kHz มี ขนาดของสัญญาณ 2 โวลท์

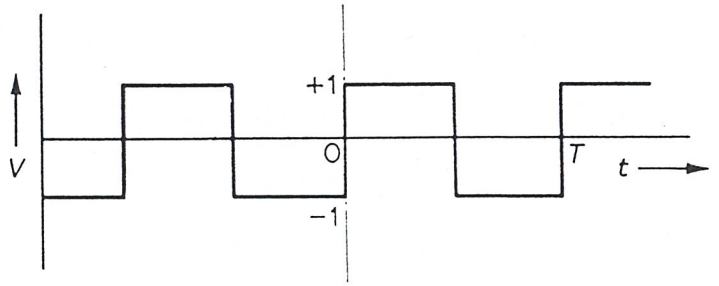
จากการวิเคราะห์ฟาสต์ฟูเรียร์ทรานสฟอร์ม ค่าสเปกตรัมของสัญญาณซายน์จะมีขนาดของค่าจริงและค่าจินตภาพดังรูปที่ 6.3



รูปที่ 6.3 รูปสเปกตรัมของสัญญาณซายน์ตามทฤษฎีของ Fast Fourier Transform

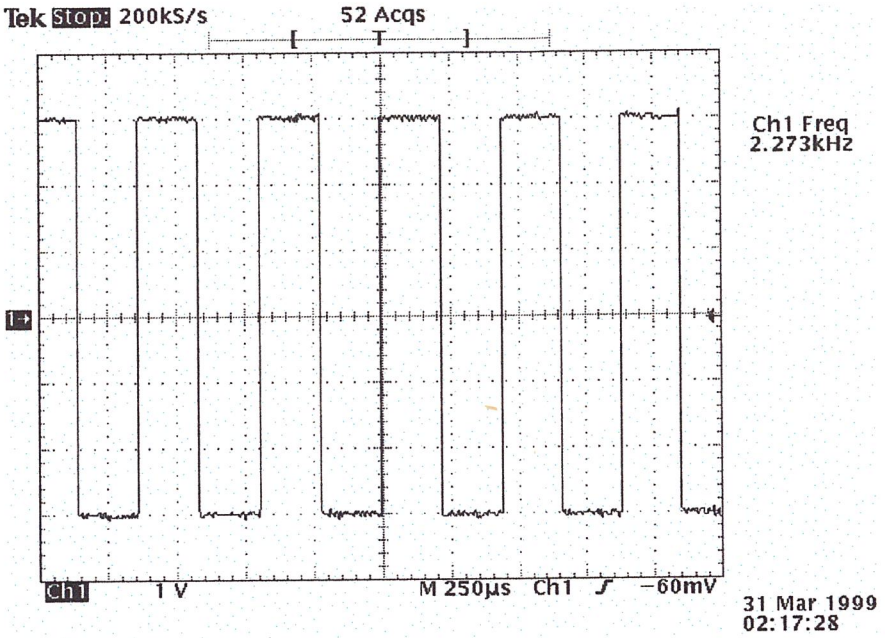
จากรูปที่ 6.3 จะเห็นได้ว่าค่าสเปกตรัมของสัญญาณซายน์ทางด้านบวกจะมีสเปกตรัมเพียงค่าเดียวในทางด้านค่าจินตภาพส่วนทางด้านจริงเป็นศูนย์ และมีขนาดของสัญญาณเป็นครึ่งหนึ่งของขนาดของสัญญาณซายน์ โดยเมื่อเปรียบเทียบผลที่ได้จากรูปที่ 6.2 จะเห็นว่าค่าสเปกตรัม จะตรงที่ตำแหน่งช่อแรงแรก เพราะความถี่ของสัญญาณตรงกับ Hz/div พอดี โดยลักษณะของซายน์จะมีแค่ความถี่เดียว และขนาดของสเปกตรัมเป็นครึ่งหนึ่งของขนาดของสัญญาณตามทฤษฎีในรูปที่ 6.3 ส่วนมุมเฟสเป็น –90 องศา ซึ่งเป็นไปตามทฤษฎีที่มีแต่ค่าจินตภาพ ส่วนค่าจริงไม่มี

รูปที่ 6.4 จากการวิเคราะห์ฟูเรียร์ทรานสฟอร์ม สมการของสัญญาณรูปคลื่นสี่เหลี่ยมจะมีค่าตามรูปดังนี้

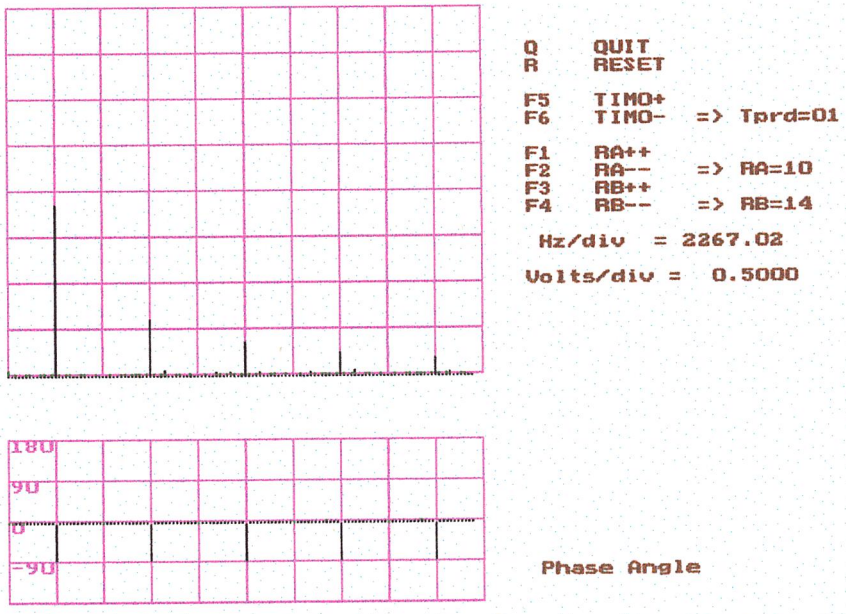


$$f(t) = \frac{4}{\pi}\left[\sin \omega t + \tfrac{1}{3}\sin 3\omega t + \tfrac{1}{5}\sin 5\omega t \cdots\right]$$

รูปที่ 6.4 สมการของสัญญาณรูปคลื่นสี่เหลี่ยมตามทฤษฎีฟูเรียร์ทรานส์ฟอร์ม



รูปที่ 6.5 สัญญาณรูปสี่เหลี่ยมที่ความถี่ 2.273 kHz ที่มี Vp-p 6 โวลต์

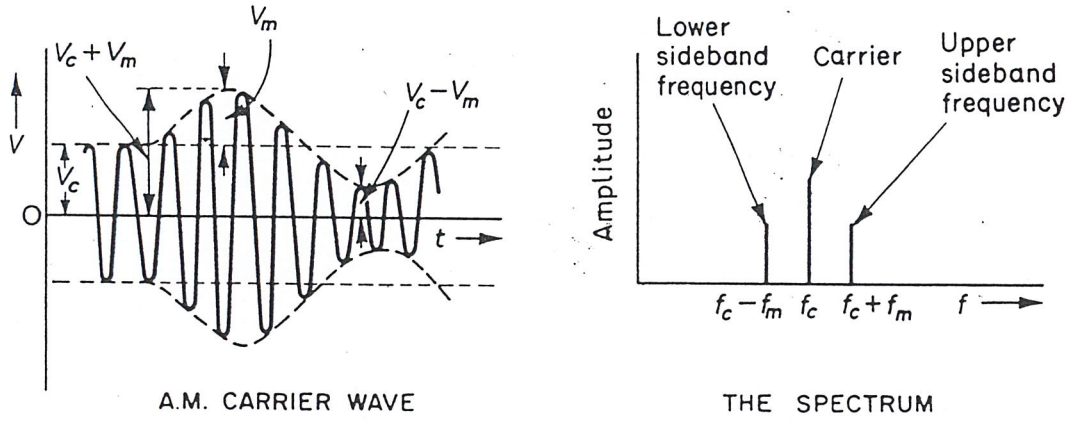


รูปที่ 6.6 รูปสเปกตรัมของสัญญาณรูปคลื่นสี่เหลี่ยมที่มีความถี่ 2.267 kHz และมีขนาด 3 โวลท์

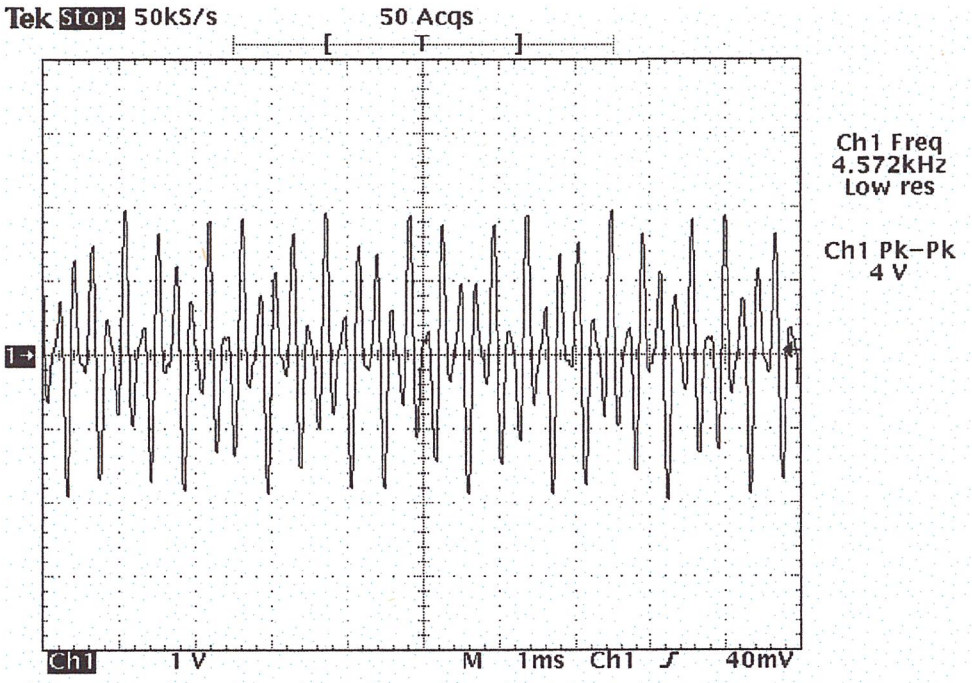
จากสมการตามทฤษฎีในรูปที่ 6.4 สัญญาณคลื่นสี่เหลี่ยมจะมีความถี่ของฮาร์มอนิคที่ 1 และจะมีความถี่ที่ฮาร์มอนิคที่ 3 , 5 , 7 ,... ต่อๆ ไป มาประกอบกันเป็นรูปคลื่นสี่เหลี่ยม ส่วนขนาดของแต่ละตำแหน่งที่ได้จากทฤษฎีจะเห็นว่า มีการลดลงเป็นอัตราส่วน คือ ที่ฮาร์มอนิคที่ 3 จะมีขนาดลดลงเป็น 3 เท่าของฮาร์โมนิคที่ 1 และ ฮาร์มอนิคที่ 5 จะมีขนาดลดลงเป็นอัตรา 5 เท่าของฮาร์โมนิคที่ 1 และที่ลำดับต่อๆ ไปก็จะลดลงเป็น 7เท่า , 9 เท่า ,...ตามลำดับของฮาร์โมนิคที่ 1

จากรูปที่ 6.6 จะแสดงค่าสเปกตรัมของสัญญาณคลื่นรูปสี่เหลี่ยม โดยจะปรากฏเป็นขนาดสูงสุดที่ความถี่หลักของสัญญาณ คือ เป็นฮาร์โมนิคที่ 1 และจะมีแถบความถี่ต่อไปที่เพิ่มขึ้นจากตำแหน่งของความถี่หลัก นั่นคือ เป็นฮาร์โมนิคที่ 3 และลำดับต่อๆไปก็เช่นกัน โดยจะมีความถี่เรียงลำดับเป็นฮาร์มอนิคที่ 1 , 3 , 5 , 7 , 9 ,...ไปเรื่อยๆ โดยในรูปที่ 6.6 ฮาร์โมนิคที่ 1 เกิดที่ช่องที่ 1 คือมีความถี่ประมาณเท่ากับ 2267.2 Hz ใกล้เคียงกับสัญญาณที่ป้อนเข้าตามรูปที่ 6.5 และลำดับต่อไปจะเป็นช่องที่ 3 และลำดับต่อไปเป็นช่องที่ 5 ก็จะเกิดในช่องที่ 5 ต่อๆ ไป

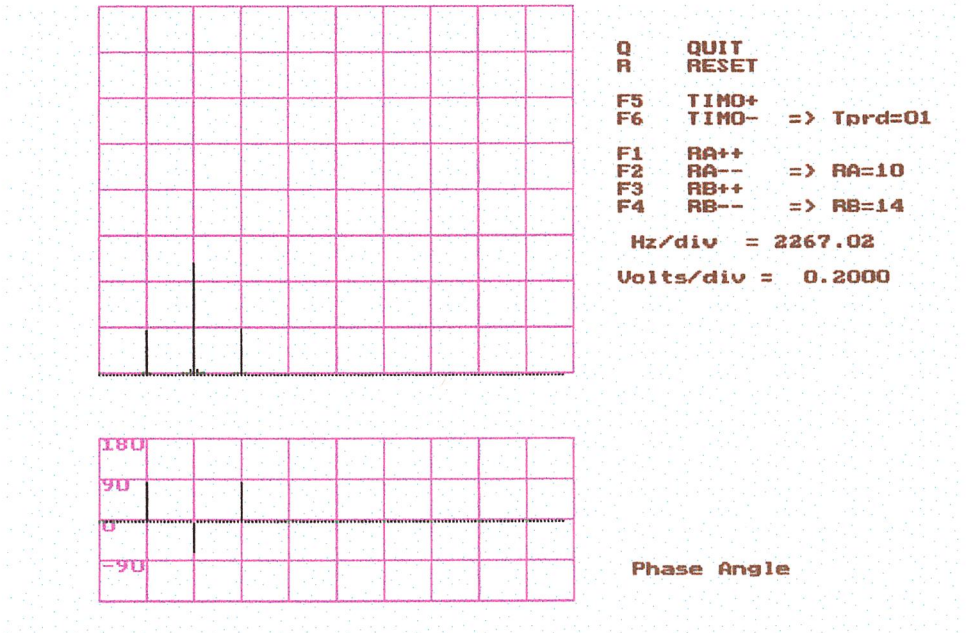
ดังนั้นถ้าเป็นออสซิลโลสโคปก็จะเห็นเป็นรูปของสัญญาณสี่เหลี่ยม โดยเมื่อคำนวณหาค่าความถี่กับ Time/div ของออสซิลสโคปจะพบว่าเป็นสัญญาณสี่เหลี่ยมที่มีความถี่เดียวกับที่ฮาร์โมนิคที่ 1 เท่านั้น ซึ่งในความเป็นจริงแล้ว จะมีความถี่หลายความถี่มารวมกันเป็นสัญญาณรูปคลื่นสี่เหลี่ยมที่ความถี่หลักนั้น



รูปที่ 6.7 แสดงสเปกตรัมที่ได้จากสัญญาณที่เกิดเป็น AM (Amplitude Modulatuion)ตามทฤษฎี

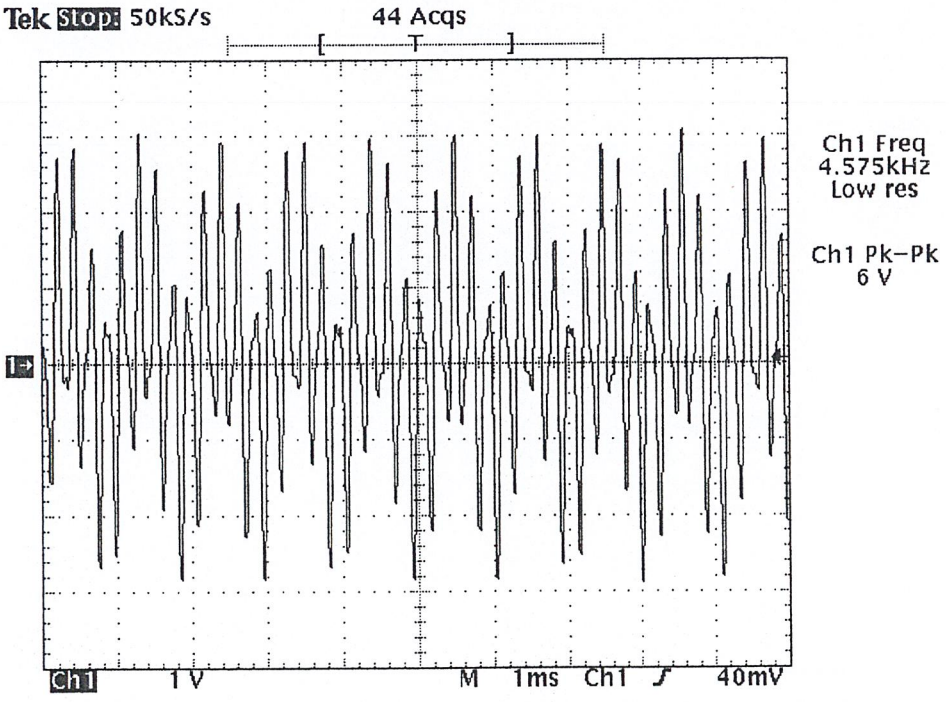


รูปที่ 6.8 สัญญาณมอดดูเลทที่ป้อนเข้า โดยเป็นสัญญาณซายน์ 4.57 kHz มี Vp-p 4 โวลต์ และสัญญาณที่มามอดดูเลทเป็นสัญญาณซายน์ 2.266 kHz มี Vp-p 4 โวลต์

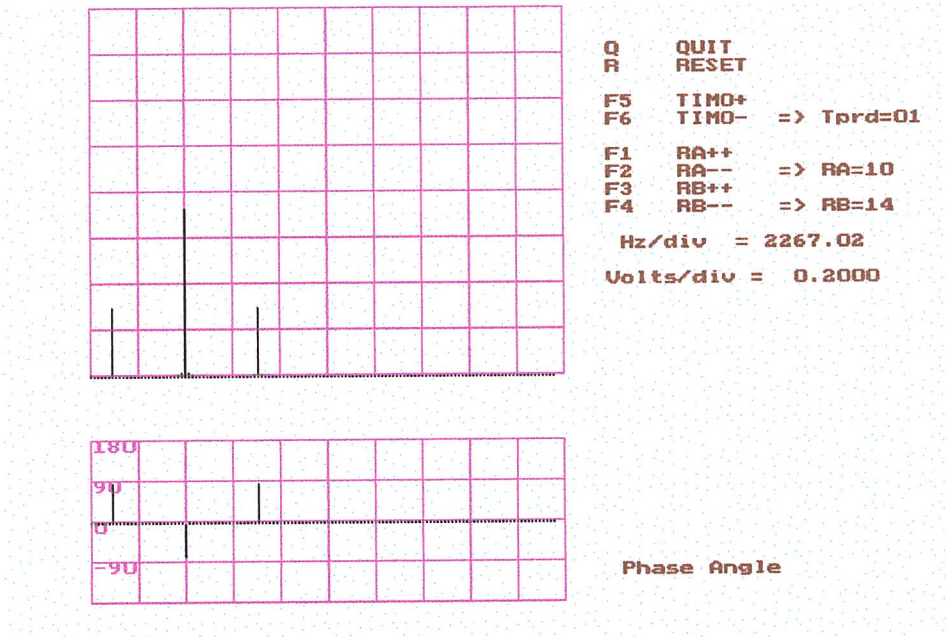


รูปที่ 6.9 แสดงสเปกตรัมที่ได้จากสัญญาณ AM (Amplitude Modulation) ของสัญญาณซายน์ที่ความถี่ 4.54 kHz ขนาด 2 โวลต์มอดดูเลทกับสัญญาณซายน์ที่ความถี่ 2.266 kHz ขนาด 2 โวลต์

จากรูปที่ 6.7 ตามทฤษฎีจะพบว่าสัญญาณที่ได้มาจากการ AM(Amplitude Modulation) ของ 2 ความถี่ของสัญญาณรูปซายน์ จะมีความถี่หลักเกิดเป็นสเปกตรัมที่ตำแหน่งความถี่หลักของสัญญาณ และความถี่ของสัญญาณที่มาทำการมอดดูเลท จะเกิดเป็นสเปกตรัมที่ตำแหน่งทั้ง 2 ข้างของความถี่หลัก โดยจะบวกลบความถี่หลักด้วยความถี่ที่มาทำการมอดดูเลท จะเห็นว่ารูปที่ 6.9 นี้จะแสดงแถบความถี่ของสัญญาณที่มอดดูเลทป้อนเข้าดังรูปที่ 6.8 แล้วเป็นไปตามทฤษฎี โดยที่แถบที่ความถี่สูงสุดอยู่ที่ความถี่หลักที่ช่องที่ 2 เท่ากับ 4534.4 Hz และมีความถี่ด้านข้างห่างจากความถี่หลักเท่ากับ 1 ช่องเป็น 2267.2 Hz เท่ากับความถี่ที่นำมามอดดูเลท และเมื่อทำการเพิ่มความถี่ของสัญญาณที่เป็นตัวมามอดดูเลทเป็น 3.57 kHz จะแสดงรูปสัญญาณให้เห็นดังรูป 6.10 และสเปกตรัมของสัญญาณในรูปที่ 6.11 ซึ่งจะปรากฏว่าแถบความถี่ด้านข้างขยายออกเพิ่มขึ้น ซึ่งเป็นไปตามทฤษฎี



รูปที่ 6.10 สัญญาณมอดดูเลทที่ป้อนเข้า โดยเป็นสัญญาณซายน์ 4.57 kHz มี Vp-p 6 โวลต์ โดยสัญญาณที่มามอดดูเลทเป็นสัญญาณซายน์ 3.54 kHz มี Vp-p 4 โวลต์

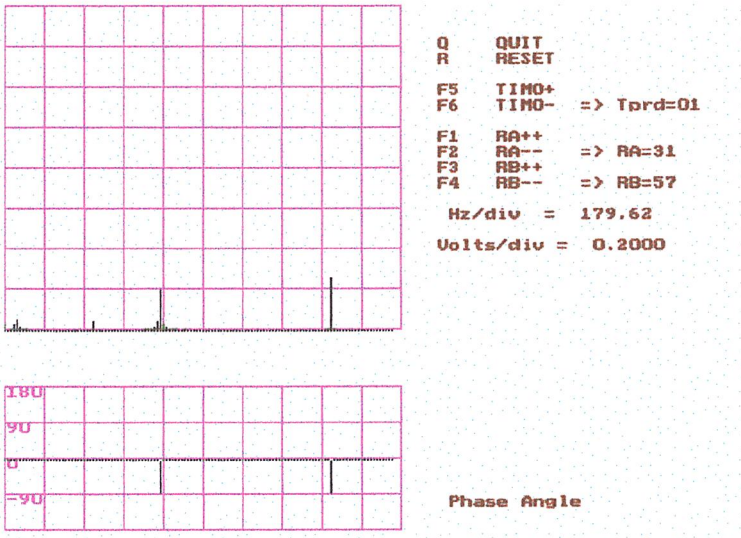


รูปที่ 6.11 แสดงสเปกตรัมที่ได้จากสัญญาณ AM (Amplitude Modulation)
ของสัญญาณซายน์ที่ความถี่ 4.54 kHz ขนาด 3 โวลต์
มอดดูเลทกับสัญญาณซายน์ที่ความถี่ 3.57 kHz ขนาด 2 โวลต์

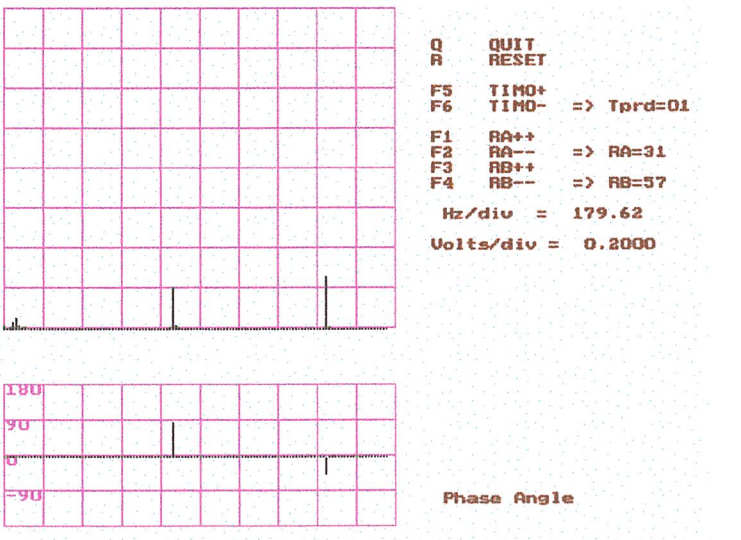
จากการวิเคราะห์สเปกตรัมทำให้สามารถมองเห็นความถี่ที่มามอดดูเลทได้ เพราะถ้ารูปสัญญาณที่มีมอดดูเลทมาวัดดูได้ในออสซิลโลสโคป โดยการที่จะอ่านค่าความถี่ของสัญญาณจะไม่ได้ หรือไม่ก็ยากมาก จากรูปที่ 6.11 ก็จะแสดงสเปกตรัมโดยที่ความถี่สัญญาณที่มามอดดูเลทเพิ่มขึ้นความห่างของแถบความถี่ข้างๆ ก็จะเพิ่มขึ้นด้วย ซึ่งเป็นไปตามทฤษฎีตามรูปที่ 6.7

ในการทดลองได้มีการทดลองนำไปวัดสัญญาณของการกดปุ่มของเครื่องโทรศัพท์ โดยปกติที่สายสัญญาณโทรศัพท์มีแรงดันอยู่ประมาณ 50 โวลต์ เมื่อยกหูแล้วแรงดันตกเหลือประมาณ 10 โวลต์ จึงได้นำ ตัวเก็บประจุมาต่อเพื่อลดแรงดัน เมื่อยกหูแล้วกดปุ่มเครื่องโทรศัพท์จะส่งสัญญาณความถี่กลับไปตามสาย จึงได้นำเอาสัญญาณนี้มาทดลองวัด ในปุ่มของโทรศัพท์จะแบ่งออกเป็น 4 แถว 3 คอลัมภ์ ในแต่ละแถวก็จะมีความถี่ของตัวเอง แต่ละคอลัมภ์ก็เช่นกัน เมื่อกดปุ่มแล้วสัญญาณที่ได้ก็จะมีความถี่ตามที่ แถวและคอลัมภ์ของปุ่มนั้นรวมกันมาเป็นสัญญาณ

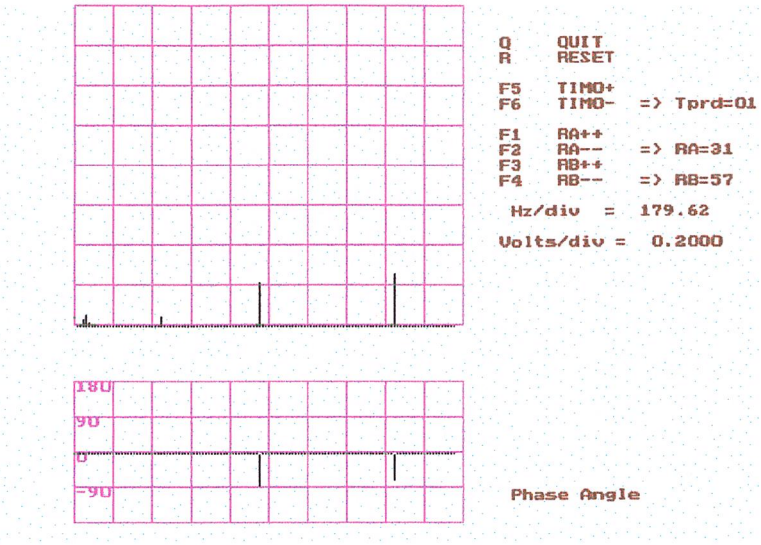
ในการทดลองวัดสัญญาณนี้ ได้เลือกวัดสัญญาณดูที่ปุ่ม 3 ,6 ,9 ,7 และ 8 ได้ผลการทดลองดังแสดงในรูปต่อไปนี้



รูปที่ 6.12 แสดงแถบความถี่ของสัญญาณโทรศัพท์เมื่อกดปุ่ม 3

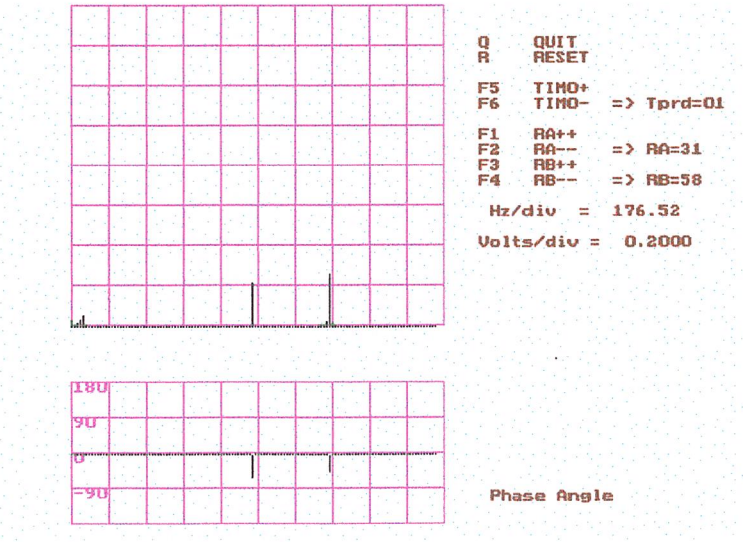


รูปที่ 6.13 แสดงแถบความถี่ของสัญญาณโทรศัพท์เมื่อกดปุ่ม 6



รูปที่ 6.14 แสดงแถบความถี่ของสัญญาณโทรศัพท์เมื่อกดปุ่ม 9

จากรูปที่ 6.12 ,6.13 และ 6.14 ในแต่สัญญาณของแต่ละปุ่มจะเกิดความถี่ขึ้น 2 ความถี่ และทั้ง 3 รูปนี้ จะพบว่า ความถี่ที่เปลี่ยนแปลงตามการกดปุ่ม คือที่แถบความถี่อันแรก แสดงว่า แถบความถี่ด้านหลัง (ขวามือ) ซึ่งคงที่ นั้นเป็นแถบความถี่ของสัญญาณที่คอลัมภ์ 3 นั่นเอง



รูปที่ 6.15 แสดงแถบความถี่ของสัญญาณโทรศัพท์เมื่อกดปุ่ม 7

รูปที่ 6.16 แสดงแถบความถี่ของสัญญาณโทรศัพท์เมื่อกดปุ่ม 8

จากรูปที่ 6.14 ,6.15 และ 6.16 จะเห็นว่าแถบความถี่ที่เปลี่ยนแปลง คือ แถบความถี่ด้านหลัง (ทางขวามือ) ส่วนแถบความถี่ทางด้านหน้า(แถบความถี่ทางด้านซ้ายมือ)จะอยู่ตรงตำแหน่งเดิม ดังนั้นแถบความถี่ด้านหน้านั้นเป็นแถบความถี่หลักของแถวที่ 3 นั่นเอง

จากตำแหน่งที่แถบความถี่ด้านหน้าอยู่ประมาน 4.8 ของช่อง เมื่อคูณกับค่าของ Hz/div แล้ว จะได้ความถี่ของแถวที่ 3 ของแป้นปุ่มกด เท่ากับ 862.176 Hz และจากรูปที่ 6.12 ,6.13 และ 6.14 จะเห็นแถบความถี่ด้านหลังอยู่ตรงตำแหน่งประมาณ 8.2 ของช่อง ซึ่งเป็นแถบความถี่ของหลักที่ 3 ของแป้นปุ่มกด เมื่อทำการคูณกับ Hz/div แล้ว จะได้ค่าความถี่ของหลักที่ 3 ของแป้นปุ่มกด เท่ากับ 1472.884 Hz ส่วนในค่าอื่นๆ ก็คิดหาแบบเดียวกัน

การวิเคราะห์สเปกตรัมนี้สามารถแยกสัญญาณที่มีความถี่หลายความถี่มารวมกัน ให้ทราบถึงความถี่ที่ประกอบกันเป็นรูปสัญญาณได้ ระบบของโปรแกรมยังมีการทำงานแบบต่อเนื่อง(Real Time) ดังนั้น เมื่อมีการเปลี่ยนแปลงค่าความถี่ก็จะสามารถเปลี่ยนแปลงตามความถี่ที่เปลี่ยนแปลงได้ทันที จึงสามารถเห็นได้โดยง่าย

สรุปผลการทดลอง

จากการทดลองจะเป็นได้ว่าสัญญาณที่ป้อนเข้าไปจะออกมาเป็นรูปของสเปกตรัมของแต่ละรูปคลื่นนั้น ๆ ตามวิธีการแปลงฟาสต์ฟูเรียร์ทรานส์ฟอร์มโดยลักษณะเฉพาะของสเปกตรัมของแต่ละสัญญาณนี้ เป็นสิ่งที่สามารถนำไปใช้งานในด้านต่าง ๆ ได้มากมาย เช่น การหาความเร็วของมอเตอร์โดยวิธีการหาความถี่ที่เปลี่ยนแปลงไปตามความเร็วของมอเตอร์ จากสัญญาณที่นำมาใช้ในการแปลงฟูเรียร์นี้จะเป็นกระแสไฟฟ้า โดยปกติความเร็วของมอเตอร์จะมีค่าเปลี่ยนแปลงไปตามกระแสที่ป้อนเข้ามอเตอร์ ถ้ามอเตอร์ความเร็วลดลงจะทำให้กระแสสูงขึ้น หรือ การดูสเปกตรัมของสัญญาณที่ได้จากการกดปุ่มโทรศัพท์ ว่าเบอร์ที่กดมีความถี่แตกต่างกันอย่างไร เป็นต้น

ข้อจำกัดของระบบ

สามารถรับสัญญาณอินพุทที่มีแรงดัน ไม่เกิน +/- 3 โวลต์ ในความถี่ที่จะทำการวัดนี้ควรมีความถี่อยู่ในช่วงไม่เกิน 4 เท่าของ Hz/div จะทำให้มีการประมวลผลได้ค่าที่ดี ถ้ามากกว่าในบางสัญญาณอาจจะไม่สามารถแสดงความถี่ได้หมด เช่น สัญญาณรูปคลื่นสี่เหลี่ยม ที่มีลำดับความถี่เป็น 1 ,3 ,5 ,... จะไม่เห็นความถี่ที่ลำดับที่ 5 ในโปรแกรมมีความถี่ในการสุ่มข้อมูลสูงสุด 66 kHz ในกรณีที่ความถี่สูงกว่า 26.4 kHz ก็จะไม่สามารถวัดค่าได้

ข้อเสีย

สเปกตรัมอนาไลเซอร์นี้ยังเป็นซอฟแวร์ที่ปฏิบัติการบนดอส มีข้อจำกัดในการทำงานอยู่ทำให้ในการทำงานของสเปกตรัมอนาไลเซอร์ไม่สามารถเรียกใช้โปรแกรมอื่นที่ใช้วานร่วมกันหรือต้องการแสดงผลพร้อมกันได้

ในงานของสเปกตรัมอนาไลเซอร์ จะใช้ที่ความถี่สูง ซึ่งในการทำงานนี้ไม่สามารถวัดที่ความถี่สูงมากๆ ได้

แนวทางในการพัฒนา

การเปลี่ยนการปฏิบัติงานจากระบบดอสไปเป็นวินโดว์ เพื่อการแสดงผลที่สมบูรณ์กว่าในด้านกราฟฟิกและง่ายต่อการใช้งาน

ควรมีการพัฒนาในการคำนวณให้มีความละเอียดมากยิ่งขึ้นโดยการเพิ่มจำนวนจุดในการคำนวณ โดยเพิ่มเป็น 512 จุด หรือ 1024 จุด

เพิ่มอัตราการสุ่มข้อมูล เพื่อสามารถใช้งานในที่ความถี่สูงมากได้ หรือเพื่อการสุ่มข้อมูลที่ละเอียดมากขึ้น

ภาคผนวก

; **c3xmmrs.asm**

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

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

```
;                                SPECTRUM.ASM

                .include "C3XMMRS.ASM"
TA              .set    10              ; Register to changed digital to analog frequency
TB              .set    14              ; Register to changed digital to analog frequency
RA              .set    10              ; Register to changed analog to digital frequency
RB              .set    14              ; Register to changed analog to digital frequency
N               .set    256             ; Size of Butterfly
N2              .set    N/2
PI              .set    3.1415926       ; Pine value
PI2             .set    2*PI
PI2N            .set    2.0*PI/N
PIN             .set    PI/N

                .start  "TWIDDLES",0X809800
                .sect   "TWIDDLES"
TR                                      ; 0x809800
                .loop   N2
                .float  cos(br($-TR,N)*PIN)
                .endloop
TI
                .loop   N2              ; 0x809880
                .float  -1*sin(br($-TI,N)*PIN)
                .endloop
DR              .set    TI+N2           ; 0x809900
DI              .set    DR+N            ; 0x809A00
BF              .set    DI+N            ; 0x809B00
REAL            .set    BF+N2           ; 0x809B80
IMAG            .set    REAL+N          ; 0x809C80
```

```
                .start "FFTCODE",0x809D80
                .sect  "FFTCODE"        ; Start of program when run initialized
_STOP           .set    1
_START          .set    2
MSG_BOX         .word   _STOP
TLVL            .word   1000
A_REG           .word   (TA<<9)+(RA<<2)+0 ; Address to indirect address memory of TA,RA
B_REG           .word   (TB<<9)+(RB<<2)+2  ; Address to indirect address memory of TB,RB
C_REG           .word   00000011b       ; Address to indirect address memory data to control
GAIN_A          .word   1               ; TLC32040
EDGESEL         .word   1
SIZE            .word   N
MASK            .word   0xFFFFFFFF
A_REGOLD        .word   0
B_REGOLD        .word   0
C_REGOLD        .word   0
S0_gctrl_val    .word   0x0E970300
S0_xctrl_val    .word   0x00000111
S0_rctrl_val    .word   0x00000111
FFTSIZE         .word   N           ; SIZE OF FAST FOURIER
TR_ADDR         .word   TR          ; ADDRESS OF TWIDDLE REAL
TI_ADDR         .word   TI          ; ADDRESS OF TEIDDLE IMAG
DR_ADDR         .word   DR          ; ADDRESS OF DATA REAL BEFORE FFT
DI_ADDR         .word   DI          ; ADDRESS OF DATA IMAG BEFORE FFT
BF_ADDR         .word   BF          ; ADDRESS OF REAL^2+IMAG^2
REAL_A          .word   REAL        ; ADDRESS OF DATA REAL AFTER FFT
IMAG_A          .word   IMAG        ; ADDRESS OF DATA IMAG AFTER FFT
TEMP            .word   0
test            .word   0
```

```
main        ldi     0x30,IE
            ldi     @S0_rdata,R0
            ldi     0,R0
            sti     R0,@S0_xdata
            ldi     @S0_rdata,R0
            ldi     0,R0
            sti     R0,@S0_xdata
            sti     R0,@RAMP
            ldi     25,RC
            rptb    preload
preload     call    GETADC
            ldi     @DR_ADDR,AR0    ; To prepare and start to sampling from signal
            ldi     @DI_ADDR,AR1
            ldi     @REAL_A,AR2
            ldi     @SIZE,RC
            subi    1,RC
CHECKSINE                   ; To define start of sampling
            ldi     0,R5
            cmpi    32,R5
            bge     SAMP
            call    GETADC
            addi    1,R5
            float   R0,R1
            cmpf    0.000,R1
            bge     CHECKSINE
            call    GETADC
            float   R0,R1
            cmpf    0.000,R1
            blt     CHECKSINE
            subi    1,RC
```

```
                stf     R1,*AR0++

SAMP            rptb    samples     ; put data from sampling to memory at DR_ADDR
                call    GETADC      ; or 0x809900
                float   R0,R0
                stf     R0,*AR0++
                ldf     0,R0
samples         stf     R0,*AR1++

                ldi     @test,R0
                bz      s_signal
                stf     R0,*AR1++
s_signal        stf     R0,*AR1++
                ldi     0,R0
                sti     R0,@S0_xdata

FFT:            ldi     @FFTSIZE,IR0    ; Start of Butterfly program
                ldi     @FFTSIZE,IR1
                lsh     -1,IR0
New_Stg         ldi     @FFTSIZE,RC
                ldi     @DR_ADDR,AR0
                ldi     @DI_ADDR,AR1
                ldi     @TR_ADDR,AR2
                ldi     @TI_ADDR,AR3
                lsh     -1,RC
                subi    1,RC
                lsh     -1,IR0
                lsh     -1,IR1
                ldi     IR1,R0
                bz      FFT_END
```

```
Blk_Top     rptb    B_Fly
            ldf     *+AR0(IR1) ,R0
          || ldf    *AR0       ,R1
            ldf     *+AR1(IR1) ,R2
          || ldf    *AR1       ,R3
            ldf     *AR2++(IR0)B,R4
          || ldf    *AR3++(IR0)B,R5
            and     @MASK,R0
            and     @MASK,R1
            and     @MASK,R2
            and     @MASK,R3
            and     @MASK,R4
            and     @MASK,R5
            addf3   R0,R1,R6
            addf3   *+AR1(IR1),R3,R7
        ||  stf     R6,*AR0
            subf3   R0,R1,R6
            stf     R7,*AR1
            subf3   R2,R3,R7
            mpyf3   R6,R4,R1
            mpyf3   R7,R5,R3
            subf    R3,R1
            stf     R1,*+AR0(IR1)
            nop     *++AR0
            mpyf3   R6,R5,R1
            mpyf3   R7,R4,R3
            addf    R3,R1
            stf     R1,*+AR1(IR1)
            nop     *++AR1
ident       ldi     @TR_ADDR,R7
```

```
                subi    AR2,R7
                ldiz    IR1,R7
                ldinz   0,R7
                addi    R7,AR0
B_Fly           addi    R7,AR1
                b       New_Stg

FFT_END         ldi     @DR_ADDR,AR0    ; Butterfly is end and start to reverse bit
                ldi     @DI_ADDR,AR2    ; to put data to memory for computer get
                ldi     @REAL_A,AR1     ; data to show on monitor
                ldi     @IMAG_A,AR3
                ldi     @SIZE,IR0
                lsh     -1,IR0
                ldi     @SIZE,RC
                subi    1,RC
                rptb    reverse
                ldf     *AR0++(IR0)B,R0
                ldf     *AR2++(IR0)B,R1
                mpyf    @VU_scale,R0
                mpyf    @VU_scale,R1
                absf    R0,R4
                cmpf    6.5,R4
                ldfle   0,R0
                absf    R1,R5
                cmpf    6.5,R5
                ldfle   0,R1
                fix     R0,R2
                fix     R1,R3
                sti     R2,*AR1++
reverse         sti     R3,*AR3++
```

```
        ldi     @GAIN_A,R0          ; To compare gain from computer is equal
        ldf     0.5333333,R6        ; or not equal . If gain in computer change
        cmpi    2,R0                ; then not equal and change it equal.
        beq     MAG
        ldf     2.1333333,R6
        cmpi    4,R0
        beq     MAG
        ldf     0.15,R6

MAG     ldi     @BF_ADDR,AR0    ; To calculated REAL^2 + IMAG^2
        ldi     @DR_ADDR,AR1
        ldi     @DI_ADDR,AR4
        ldi     @SIZE,IR0
        lsh     -1,IR0
        ldi     @SIZE,RC
        lsh     -2,RC
        subi    1,RC
        rptb    WINDOW
        ldi     0,R7
        call    STORE
        lsh     -24,R0
        or      R0,R7
        call    STORE
        lsh     -16,R0
        or      R0,R7
        call    STORE
        lsh     -8 ,R0
        or      R0,R7
        call    STORE
```

```
            lsh     0 ,R0
            or      R0,R7
WINDOW      sti     R7,*AR0++

            ldi     0x4,IE
            ldi     START,R0        ; Inner loop is check data in @MSG_Box equal to
NO_START    cmpi    @MSG_BOX,R0     ;  2 then DSK continued. If not equal to 2 then
            bnz     NO_START        ; inner loop to check until equal.
            ldi     _STOP,R0        ; After check, this below is changed value at
            sti     R0,@MSG_BOX     ; user change on Computer to program in DSK
            ldi     0,R2            ; to calculate in change value
            ldi     @A_REG  ,R0
            cmpi    @A_REGOLD,R0
            ldinz   1,R2
            sti     R0,@A_REGOLD
            ldi     @B_REG  ,R0
            cmpi    @B_REGOLD,R0
            ldinz   1,R2
            sti     R0,@B_REGOLD
            ldi     @C_REG  ,R0
            cmpi    @C_REGOLD,R0
            ldinz   1,R2
            sti     R0,@C_REGOLD
            cmpi    0,R2
            bz      main
            call    AIC_INIT
            b       main                    ; go to main

VU_scale    .float  6.00e-5
```

```
STORE   ldf     *AR1++(IR0)B,R2
        mpyf    @VU_scale,R2
        mpyf    R2,R2
        ldf     *AR4++(IR0)B,R4
        mpyf    @VU_scale,R4
        mpyf    R4,R4
        addf    R4,R2
        mpyf    R6,R2
        mpyf    0.01,R2
        fix     R2,R0
        lsh     20,R0
        lsh     -20,R0
        cmpi    127,R0
        ldige   127,R0
        lsh     24,R0
        rets

RAMP    .word   0
FLAGS   .word   0
GETADC  ldi     0x30,IE         ; loop to receive data of samling from TLC32040
        IDLE
        ldi     @FLAGS,R0
        tstb    0x20,R0
        bz      $-3
        andn    0x20,R0
        sti     R0,@FLAGS
        ldi     @S0_rdata,R0    ; in control bit is set to receive data in 16 bit
        lsh     16,R0
        ash     -16,R0
        rets
```

```
ADC     push    ST
        push    R0
        ldi     @S0_rdata,R0
        ldi     @FLAGS,R0
        or      0x20,R0
        sti     R0,@FLAGS
        pop     R0
        pop     ST
        reti
DAC     push    ST
        push    R1
        ldi     @RAMP,R1
        subi    1024,R1
        lsh     17,R1
        ash     -17,R1
        andn    3,R1
        sti     R1,@RAMP
        sti     R1,@S0_xdata
        pop     R1
        pop     ST
        reti
prog_AIC push   R1
        push    IE
        ldi     0x10,IE
        andn    0x30,IF
        ldi     @S0_xdata,R1
        sti     R1,@S0_xdata
        idle
```

```
        ldi     @S0_xdata,R1
        or      3,R1
        sti     R1,@S0_xdata
        idle
        sti     R0,@S0_xdata
        idle
        andn    3,R1
        sti     R1,@S0_xdata
        pop     IE
        pop     R1
        rets

AIC_INIT    push    R0          ; to initialized TLC32040 before use
        LDI     0x10,IE
        andn    0x34,IF
AIC_reset
        ldi     0,R0
        sti     R0,@S0_xdata
        RPTS    0x040
        LDI     2,IOF
        rpts    0x40
        LDI     6,IOF
        ldi     @S0_rdata,R0
        ldi     0,R0
        sti     R0,@S0_xdata
        ldi     @C_REG,R0
        call    prog_AIC
        ldi     0xfffc ,R0
        call    prog_AIC
        ldi     0xfffc|2,R0
```

```
        call    prog_AIC
        ldi     @B_REG,R0
        call    prog_AIC
        ldi     @A_REG,R0
        call    prog_AIC
        pop     R0
        ldi     0,R0
        sti     R0,@S0_xdata
        ldi     @S0_rdata,R0
        rets

        .start  "STUB",BF
        .sect   "STUB"
        .entry  ST_STUB         ; use this to initialized at start of program first
ST_STUB ldp     T0_ctrl
        ldi     0,R0
        sti     R0,@T0_ctrl
        sti     R0,@T1_ctrl
        sti     R0,@T0_count
        sti     R0,@T1_count
        ldi     1,R0
        sti     R0,@T0_prd
        sti     R0,@T1_prd
        ldi     0x2C1,R0
        sti     R0,@T0_ctrl
        sti     R0,@T1_ctrl
        ldi     @S0_xctrl_val,R0
        sti     R0,@S0_xctrl
        ldi     @S0_rctrl_val,R0
        sti     R0,@S0_rctrl
```

```
ldi     0,R0
sti     R0,@S0_xdata
ldi     @S0_gctrl_val,R0
sti     R0,@S0_gctrl
call    AIC_INIT
ldi     0x30,IE
ldi     @S0_rdata,R0
b       main

.start  "SP0VECTS",0x809FC5
.sect   "SP0VECTS"
B       DAC     ; XINT0   ; to set when XINT0 occurred, jump to DAC
B       ADC     ; RINT0   ;  to set when RINT0 occurred, jump to ADC

.start  "ROMVECTS",0x000000
.sect   "ROMVECTS"          ; To set address of interrupts
.word   0x809FC0  ; RESET
.word   0x809FC1  ; INT0
.word   0x809FC2  ; INT1
.word   0x809FC3  ; INT2
.word   0x809FC4  ; INT3
.word   0x809FC5  ; XINT0
.word   0x809FC6  ; RINT0
.word   0x809FC7  ; XINT1  NA
.word   0x809FC8  ; RINT1  NA
.word   0x809FC9  ; TINT0
.word   0x809FCA  ; TINT1
.word   0x809FCB  ; DINT
```

```
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -DSK.H- - - - - - - - - - - - - - - - - - - - - - -
#define XWRIT    1
#define XREAD    2
#define XCTXT    3
#define XRUNF    4
#define XSSTEP   5
#define XHALT    6
#define XW_HOST  7
#define XR_HOST  8
#define XSPARE   9
MSGS putmem  (ulong addr, ulong length, ulong *data);
MSGS getmem  (ulong addr, ulong length, ulong *data);
MSGS getmemlong(ulong addr,ulong length,long *data);
MSGS RUN_CPU      (void);
MSGS HALT_CPU     (void);
MSGS GET_DEBUG_CTXT(void);
extern uint  port  ;
extern uint  status;
extern uint  ctrl  ;
extern ulong WSHIFT ;
extern ulong WSCOUNT;
extern int   timeout;
extern int   test_flag;
extern char far Help_Msg[];
extern char  LO_PWR;
void   HPI_STRB(int val);
char   HPI_ACK(void);
MSGS   DSK_reset(void);
MSGS   recv_long(ulong *);
MSGS   recv_longo(long *);
MSGS   xmit_byte(char);
MSGS   xmit_long(ulong);
```

```
extern int _DIR;
#define PPSTRB_LO 0x5
#define PPSTRB_HI 0x4
#define RESET_LO 0x0
#define RESET_HI 0x4
#define outctrl(val)   outportb(ctrl,val)
#define instat        inportb(status)
#define inbyte        inportb(port)
#define innible       ((inportb(status) >> 4) ^ 0x8)
#define outbyte(val)   outportb(port,val)
#define inctrl        inportb(ctrl)
typedef enum { R0F, R1F, R2F, R3F, R4F, R5F, R6F, R7F, // R8i-11i  0- 7
               R0 , R1 , R2 , R3 , R4 , R5 , R6 , R7 , // R8F-11F  8-15
               AR0, AR1, AR2, AR3, AR4, AR5, AR6, AR7, //        16-24
               DP , IR0, IR1, BK , SP , ST , IE , IF , //        25-31
               IOF, RS ,  RE,  RC, PC , FREERUN, T1DIF, CNTXT //  32-38
             } TMS_REGS;
#define CTXTSIZE 39
extern ulong CTXT[];
extern ulong PC_Appli;
extern ulong DEBUG_CTXT;
extern int   WINDOWS ;
extern int   timeout ;
extern int   BW_force;
extern int   Windows_Detected;
extern int   DSK3D;
int  ports       (int base);
MSGS Scan_Command_line(char *app_name);
MSGS Init_Communication(int loops);
MSGS get_buswidth  (void);
MSGS set_buswidth  (void);
#endif
```

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

#define MSG_BOX   0x809D80L
#define TRG_BOX   0x809D81L
#define A_BOX     0x809D82L
#define B_BOX     0x809D83L
#define C_BOX     0x809D84L
#define SAMPLES   0x809D86L
#define DATABLOCK 0x809D00L
#define REALBLOCK 0x809B00L
#define IMAGBLOCK 0x809C00L
#define PINE      3.141592
#define THx 40e-9
#define Samples 256

#define graph_vwport() setviewport( 50, 0, 563, 340,   1)
#define menu_vwport()  setviewport(334, 0, 639, 340,   1)
```

```
#define A_REG  ((TA <<9)+(RA <<2)+0)
#define B_REG  ((TB <<9)+(RB <<2)+2)
typedef enum messages
{
 STOP =1,
 START=2
}message;
char DSK_APP[]   ="SPECTRUM.DSK";
char DSK_EXE_APP[]="SPECTRUM.EXE";
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[512];
int  oldbufp[512];
char buf_0[512];              // Keep past data history for
char buf_1[512];              // time averaging of signals
char buf_2[512];
char buf_3[512];
char buf_4[512];
char buf_5[512];
char buf_6[512];
```

```
char buf_7[512];
char buf_8[512];
int  avg_on = 0;
void  init_graphics(void);
int  check_key (void);
void  binsprintf(char *s,int val);
//-------------------------------------------------------------------------//
// draw_phase() is receive data from DSK to arctan and plot it
//-------------------------------------------------------------------------//
void draw_phase()
{
  int  a,b,*oldpp;
  int  z,plot;
  long  *real,*imag;
  double xx,yy,radius,phase;
  oldpp=oldbufp;
  setcolor(WHITE);
  setwritemode(1);
  for(a=0;a<256;a+=2)
            { xx=(double) *real++;
              yy=(double) *imag++;
              if(xx!=0||yy!=0) radius = atan2(yy,xx);
              if(xx==0&yy==0) radius=0;
              phase = ((180*radius)/PINE)/4;
              plot=(int) phase;
              b = 291 - plot;
              if(b>*oldpp) {line(a,*oldpp+1,a,b );}
              if(b<*oldpp) {line(a,*oldpp ,a,b+1);}
              *oldpp++ = b;
            }
```

```
}
//-------------------------------------------------------------------//
//  draw_vect() is plot spectrum from  received data DSK
//-------------------------------------------------------------------//
void draw_vect()
{
 int  x,*old;
 unsigned int y,s;
 char *ptr0, *tmp0;
 float root,data;
 tmp0 = buf_0;
 ptr0 = tmp0;
 setcolor(WHITE);
 old = oldbuf;
 setwritemode(1);
 for(x=0;x<256;x+=2)
  {
  if((data=(float)*ptr0++)<0) { data=0; }
  root = 19.2 * (sqrt(data));
  s = (int) root;
  y = 234 - s ;
  if(y  > *old) line(x,*old+1,x,y  );
  if(y  < *old) line(x,*old  ,x,y+1);
  *old++ = y;
 }
}
//*********************************************************//
//    main of program
//*********************************************************/
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, *tmp0;
 int  x;
 long *real, *imag;
 clrscr();
 Scan_Command_line(DSK_EXE_APP);
 clrscr();
 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();

   tmp0 = buf_0;
   tmp1 = buf_1; tmp2 = buf_2; tmp3 = buf_3; tmp4 = buf_4;
   ptr0 = tmp0;
   for(x=0;x<256;x++)              // Clear all buffersq
   {
```

```
    buf_1[x] = 0; buf_2[x] = 0; buf_3[x] = 0; buf_4[x] = 0;
    buf_0[x]   = 0;
    oldbuf[x]  = 235;
    oldbufp[x]=292;
   }
   check_key();
   draw_vect(); //
for(;;)
   {
    ptr0 = tmp0;
    ptr1 = tmp1; ptr2 = tmp2; ptr3 = tmp3; ptr4 = tmp4;
    for(x=0;x<256;x+=2)
    { switch(avg_on)
        { case  1: *ptr0=(*ptr1+*ptr2)          >> 1; break; // avg 2
          case  2: *ptr0=(*ptr1+*ptr2+*ptr3+*ptr4) >> 2; break; // avg 4
          default: *ptr0= *ptr1;                break; // avg 1
        }
        ptr0++;
        ptr1++; ptr2++; ptr3++; ptr4++;          // next data
    }
    draw_vect();
    draw_phase();
     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 = tmp4;                    // Rotate the buffer pointers
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(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;
  }
  New_Params = 0;
  if(getmemlong(REALBLOCK,Samples,(long *)real)!=NO_ERR)   break;
  if(getmemlong(IMAGBLOCK,Samples,(long *)imag)!=NO_ERR)   break;
  if(getmem(DATABLOCK,Samples/8,(ulong *)ptr1)!=NO_ERR) break;
  putmem(MSG_BOX,1,&MSG);
 }
```

```
  closegraph();  // Shutdown graphics before re-initializing
  printf("%s: %s\n",DSK_APP, Error_Strg(err));
  printf("Communications are being reinitialized");
  delay(1000);
  DSK_reset();
 }
}
//*************************************************//
//-----------------------------------------------------------------
// clip() is used to clip the upper and lower bounds of a number
//-----------------------------------------------------------------
long inline clip(long x, int min, int max)
{
 if(x<min) return min;
 if(x>max) return max;
 return x;
}
//-------------------------------------------------------------
// check_key() is used to associate keystrokes with actions
//-------------------------------------------------------------
int check_key(void)
{
 int  key=0;
 int  Yt=0;
 char buf[80];
 static int old_key=0; // accelerate action if same key is hit again
 static int accel  =1;
 if(kbhit()) key = bioskey(0) & 0xFF00;
 if(old_key==key) accel = accel + 1;
 else          accel = accel / 4;
```

```
accel = clip(accel,1,200); old_key = key;
if(key)
{
 switch(key)
 {
  case _Ins: T0_prdv -=accel; break;
  case _Del: T0_prdv +=accel; break;
  case _R  : return 1;  // Return reset (break) flag
  case _Q  :closegraph();
                _setcursortype(_NORMALCURSOR);
                exit(0);
                break;
  case _F5 : RA++; break;
  case _F6 : RA--; break;
  case _F7 : RB++; break;
  case _F8 : RB--; 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;
 }
 RA       = clip(RA      ,   3,      31);
 RB       = clip(RB      ,  12,      63);
 T0_prdv = clip(T0_prdv,    1,       3;
}
 menu_vwport();
clearviewport();
```

```
 setcolor(11);
 Yt=1;
 #define C1 1
 outtextxy(C1,Yt   , "Q   quit" );
 outtextxy(C1,Yt+=10,  "R   reset" );
switch(avg_on)
 { case  1: sprintf(buf,"A   avg   => 2 frames");break;
  case  2: sprintf(buf,"A   avg   => 4 frames");break;
  case  3: sprintf(buf,"A   avg   => 8 frames");break;
  default: sprintf(buf,"A   avg   => 1 frame");break;
 }
 outtextxy(C1,Yt+=10,buf);
 Yt+=10;
 sprintf(buf,"Ins  TIM0+"); outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"Del  TIM0-  => Tprd=%02ld",T0_prdv);  outtextxy(C1,Yt+=10,buf);
 Yt+=10;
 sprintf(buf,"F5   RA++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F6   RA--   => RA=%02d",RA);    outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F7   RB++");                outtextxy(C1,Yt+=10,buf);
 sprintf(buf,"F8   RB--   => RB=%02d",RB);    outtextxy(C1,Yt+=10,buf);
 Yt+=10;
 sprintf(buf,  "Hz/div=%7.2f",Hz_per_div); outtextxy(10,270,buf);
sprintf(buf,  "  Fdac=%7.2f",Fsx); outtextxy(10,285,buf);
 return 0;
}


//-------------------------------------------------------------
// init_graphics() initializes the host display for graphics
// output and then draws the lines and text for the display graph
```

```
//-------------------------------------------------------------
void init_graphics(void)
{
int gdriver = EGA, gmode = EGAHI, errorcode, Y, X;
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);
}
  initgraph(&gdriver, &gmode, "");       // if possible open EGA mode
  errorcode = graphresult();
  if (errorcode != grOk)
  { printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  clearviewport();                    //
  graph_vwport();
  setcolor(GREEN);
  for(Y=245;Y<=337;Y+=23) line(0,Y,260,Y);
  for(X=0;X<=260;X+=26) line(X,245,X,337);
  for(Y=0;Y<=234;Y+=26) line(0,Y,260,Y);  // draw reticle
  for(X=0;X<=260;X+=26) line(X,0,X,234);  //
  setwritemode(1);                    // display lines are XOR drawn
  setcolor(15);                       // light gray
}
```

```
//-------------------------------------------------------------
// Load_File() is used  to load  file.dsk to DSK to used
// to run program
//-------------------------------------------------------------
MSGS Load_File(char *file,TASK task)
{
 char filename[120], *p;
 FILE *DASM_FILE;
 MSGS err;
 if(strlen(file) > 115) return FILE_LEN;
 strcpy(filename,file);
 if(strstr(filename,".")==NULL)
 { strcat(filename,DSKEXT);
  if(access(filename,0)!=0)
  {
   strcpy(strstr(filename,"."),".out");
   if(access(filename,0)!=0)
        strcpy(strstr(filename,"."),".hex");
  }
 }
 //
 if(access(filename,0)!=0)
 {
  return ACCESS_ERR;
 }
 if(strstr(filename,".out"))
 {
  DASM_FILE = fopen("DASMFILE.FIL","wt");
  if(DASM_FILE==NULL) return ACCESS_ERR;
  fprintf(DASM_FILE,"%s   Last COFF file loaded into DSK3D\n",filename);
```

```
    fclose(DASM_FILE);
  }
  strupr(filename);
  *DASM_file = 0;
 if((strstr(filename,".HEX"))&&(task==LOAD))
  return(Load_Hex_File(filename, task));
}

//-------------------------------------------------------------
// Init_Communication() checks the current LPT connection for a
// currently active DSK kernel.  If active, the existing kernel
// is used.  If not, the DSK kernel is bootloaded into the device.
// To invalidate an existing kernel, issue a DSK_reset()
//-------------------------------------------------------------
MSGS Init_Communication(int loops)
{
  char *p1,*p2,*p;
  MSGS err;
  ulong temp, orig;
  ulong testval=0x00320C31L;
for(;;)
  {
   strcpy(PATH,KernelName);
   if(access(PATH,0)==0) break;
   strcpy(PATH,_argv[0]); // get the startup path environment
   p = EMSG;
   p+=sprintf(p,"Application startup path = %s\n\n",PATH);
   p1 = PATH;
   for(;;)
   { p2 = p1;
```

```
    p1 = strstr(p1,"\\");
    if(p1==NULL) break;
    p1++;
  }
  *p2=0;
  strcat(PATH,KernelName);
  p+=sprintf(p,"Kernel search path = %s\n\n",PATH);
  if(access(PATH,0)==0) break; // If file exists exit
  p+=sprintf(p,
  "%s was not found in the current directory or startup path\n"
    "\nC:\PROJECT %s\n",KernelName,KernelSrce,KernelSrce);
  clrscr();
  printf(EMSG);
  exit(0);
}
 for(;loops>0;loops--)
 {
  get_buswidth();
  getmem(TESTLOC,1,&orig);
  putmem(TESTLOC,1,&testval);
  getmem(TESTLOC,1,&temp);
  if(putmem(TESTLOC,1,&orig) == NO_ERR)
  if(temp == testval)
  if(set_buswidth() == NO_ERR) break;
  if((err=Load_File(PATH,BOOT))!=NO_ERR)
  enable_Mtask();
  asm sti
  if(kbhit()) if(toupper(getch()) == 'Q') exit(0);
 }
 asm sti
```

```
  if(loops==0) return TIMEOUT;
  if((err=Load_File(PATH,SLOAD))!=NO_ERR) return err;
  ref_add("R0" ,0,1,0); ref_add("R1" ,0,1,0);
  ref_add("R2" ,0,1,0); ref_add("R3" ,0,1,0);
  ref_add("R4" ,0,1,0); ref_add("R5" ,0,1,0);
  ref_add("R6" ,0,1,0); ref_add("R7" ,0,1,0);
  ref_add("F0" ,0,2,0); ref_add("F1" ,0,2,0);
  ref_add("F2" ,0,2,0); ref_add("F3" ,0,2,0);
  ref_add("F4" ,0,2,0); ref_add("F5" ,0,2,0);
  ref_add("F6" ,0,2,0); ref_add("F7" ,0,2,0);
  ref_add("AR0",0,1,0); ref_add("AR1",0,1,0);
  ref_add("AR2",0,1,0); ref_add("AR3",0,1,0);
  ref_add("AR4",0,1,0); ref_add("AR5",0,1,0);
  ref_add("AR6",0,1,0); ref_add("AR7",0,1,0);
  ref_add("DP" ,0,1,0); ref_add("IR0",0,1,0);
  ref_add("IR1",0,1,0); ref_add("BK" ,0,1,0);
  ref_add("SP" ,0,1,0); ref_add("ST" ,0,1,0);
  ref_add("IE" ,0,1,0); ref_add("IF" ,0,1,0); ref_add("IOF" ,0,1,0);
  ref_add("RS" ,0,1,0); ref_add("RE" ,0,1,0);
  ref_add("RC" ,0,1,0); ref_add("PC" ,0,1,0);
  ref_add(">> End of Kernel<<",0,1,0);
  post_boot_sym     = last_ref;
  post_boot_symbols = symbol_ptr;
  err = GET_DEBUG_CTXT();
  return err;
}
```

```
//----------------------------------------
// ports() converts the port, status, ctrl
// values from the base port value.
//----------------------------------------
int ports(int base)
{
  port   = base+0;
  status = base+1;
  ctrl   = base+2;
  switch(port)
  {
    case 0x378: return 1;
    case 0x278: return 2;
    case 0x3BC: return 3;
    default   : return 99;
  }
}
//------------------------------------------------------------
// Scan_Command_Line() looks for standard DSK command line
// arguments which are common to all C3X DSK apps and sets
// the appropriate global variables.  If help is requested
// the standard help message is displayed followed by an exit
//------------------------------------------------------------
MSGS Scan_Command_line(char *app_name)
{
  char *p;
  char MSG[80], *eptr;
  char reset_flag = 0;
  for(int i=1; i<_argc;i++)
  {
```

```
 p = _argv[i];
 if(*p=='-') p++;  // Ignore '-' if it preceeds an argument
 if(*p=='/') p++;  // Ignore '/' if it preceeds an argument
 strcpy(MSG,p);
 strupr(MSG);
 if(strexact(MSG,"R"     )) reset_flag=1;
 if(strexact(MSG,"RESET" )) reset_flag=1;
 if(strexact(MSG,"WIN=0" )) WINDOWS  = 0;
 if(strexact(MSG,"WIN=1" )) WINDOWS  = 1;
 if(strexact(MSG,"AUTO"  )) BW_force = 0;
 if(strexact(MSG,"BW=8"  )) BW_force = 8;
 if(strexact(MSG,"BW=4"  )) BW_force = 4;
 if(strexact(MSG,"BYTE"  )) BW_force = 8;
 if(strexact(MSG,"NIBBLE")) BW_force = 4;
 if(strexact(MSG,"MTASK=0")) No_MTask = 1;
 if(strexact(MSG,"NOMTASK")) No_MTask = 1;
 if(strexact(MSG,"LO_PWR")) LO_PWR   = 1;
 if(strexact(MSG,"T="    )) timeout = (int) atol(&MSG[2]);
 if(strexact(MSG,"LPT=1" )) port=0x378;
 if(strexact(MSG,"LPT=2" )) port=0x278;
 if(strexact(MSG,"LPT=3" )) port=0x3BC;
 if(strexact(MSG,"LPT1"  )) port=0x378;
 if(strexact(MSG,"LPT2"  )) port=0x278;
 if(strexact(MSG,"LPT3"  )) port=0x3BC;
 if(strstr (MSG,"PORT="  )) port = (int)strtol(MSG+5,&eptr,0);
 if(strexact(MSG,"TEST" )) test_flag=1;
switch(port)                  // Use the ports() function within a
{                           // switch to make sure port address is
 case 0x278:                  // a valid PP
 case 0x378:
```

```
    case 0x3BC: break;
    default: port = 0x378;
  }
  ports(port);
  if(reset_flag)              // Kill the communications kernel at
  { outctrl(RESET_LO); delay(1); // this port by pulsing the INIT line
    outctrl(RESET_HI); delay(1);
  }
  return NO_ERR;
}
//*********************************************************************
// DSK_reset () is subprogram to call for restart program or
//  initialized DSK and all to new start.
//*********************************************************************
MSGS DSK_reset()
{
  char MSG[160], *p;
  if(port!=oldport) bk = 1; // If the port number changes or a
  oldport=port;             // key is hit update the display.
  for(;;)
  { whirly(1,1);
    p = MSG;
    if(bk)
    {
      clrscr();
      textattr((BLACK<<4) | LIGHTGRAY);
      p+=sprintf(p,
            "\r\n"
            ,port,ports(port));
      clrscr();
```

```
    cprintf(MSG); // Print only on first pass or keystroke
   }
   p=MSG;
   outctrl(RESET_LO); delay(20);  // 1/60 s catches 1/2 cycle of bad pwr
   if((instat & 0x8)==0)
   {
    p+=sprintf(p,     "\r\n");
   }
   else
   { outctrl(RESET_HI); delay(20); // 1/60 s catches 1/2 cycle of bad pwr
    if(instat & 0x8)
    {
         outctrl(PPSTRB_LO); delay(1);
         if((instat & 0x8) == 0)
         {
          clrscr();
          printf(" NOW    LOAD  FILE    \r\n"
                 " AND    RUN   DSK     \r\n"
                 " >>>> PLEASE  WAIT <<<<\r\n");
          break;
         }
}
    else // an error has occurred
    p+=sprintf(p,  ARE  YOU CONNECTED DSK ???"  \r\n");
   }
   if(bk)
   { textattr((BLACK <<4) | WHITE | BLINK);
    cprintf(MSG); // Print only on first pass or keystroke
   }
   p=MSG;
```

```
    uint far *ip;
    ip = (uint far *)MK_FP(0,0x40C); //0x408 0x40A 0x40C

    if(bk)
    { textattr((BLACK<<4) | LIGHTGRAY);
     cprintf("                                \r\n");
}
    p=MSG;
    if(test_flag)
    {
     switch(port)
     {
          case 0x378: port = 0x278; break;
          case 0x278: port = 0x3BC; break;
          case 0x3BC:
          default:   port = 0x378; break;
     }
    }
    if((bk=bioskey(1))!=0)
    { switch(bk=(bioskey(0)&0xFF00))
     { case _1: ports(0x378); break;
          case _2: ports(0x278); break;
          case _3: ports(0x3BC); break;
          case _H: Display_manual("DSK3D.EXE"); break;
          default: fcloseall();
                   exit(0);
     }
    }
}
outctrl(RESET_LO); delay(1);
```

```
 outctrl(RESET_HI); delay(1);
 return NO_ERR;
}
//*********************************************************************
// HPI_STRB() drives the strobe line high or low.  Use this
// function with HPI_ACK() to create a user defined interlock
//*********************************************************************
void HPI_STRB(int val)
{
 if(val==0) outctrl(PPSTRB_LO|_DIR);
 else      outctrl(PPSTRB_HI|_DIR);
}
char HPI_ACK(void)
{
 if((instat & 0x8) == 0) return 1;
                         return 0;
}
//*********************************************************************
//  recv_longo() is received  data from printer port from DSK
//*********************************************************************
MSGS recv_longo(long *x)
{
 long ul=0, n;
 int t=3000;
 char shift=0;
 //---------------------------------------
 for(shift=0;shift<32;shift-=WSHIFT)
 {
  // Handshake for 1st xfer and slow down for long cables
  for(;t>0;t--)
```

```
    { if((instat & 0x8) == 0) break;
     if(t<=1)
     {
          //asm sti;
          enable_Mtask();
          return(RECV_ERR);
     }
    }
    t = timeout;
   outctrl(PPSTRB_LO | _DIR);
    if(WSHIFT == -8)  n = inbyte & 0xff;
    else              n = innible & 0xf;
    ul = ul | (n << shift);
    outctrl(PPSTRB_HI | _DIR);
   }
   if(LO_PWR)
   {
    outbyte(0xff);
    outctrl(PPSTRB_HI);
   }
   *x = ul;
   return NO_ERR;
}
//*********************************************************
// xmit_long() sends the 32 bit value 'snd' to the
// printer port using the DSK interface protocol
//*********************************************************
MSGS xmit_long(ulong snd)
{
  int t = 3000, b;
```

```
  for(b=0;b<4;b++)
  {
   outbyte(snd);
   outctrl(PPSTRB_LO);
   for(;t>0;t--)              // Loop will not execute for t<=0
   {
    if((instat&0x8)==0) break; // exit on good handshake
    if(t<=1)
    { outctrl(PPSTRB_HI|_DIR);
          //asm sti;
          enable_Mtask();
          return(XMIT_ERR);
    }
   }
   t = timeout;
   if(b==3) outctrl(PPSTRB_HI | _DIR);
   else    outctrl(PPSTRB_HI);
   snd = snd>>8;
  }
//outctrl(PPSTRB_HI | _DIR);
  if(LO_PWR)
  {
   outbyte(0xff);
   outctrl(PPSTRB_HI);
  }
  return NO_ERR;
}
//*******************************************************
// xmit_byte() sends a single 8 bit value 'snd' to the
// printer port using the DSK interface protocol
```

```
//*******************************************************
MSGS xmit_byte(char snd)
{
 int t;
 outbyte(snd);
 outctrl(PPSTRB_LO);  // Signal data XMIT
 for(t=50; t>0; t--)  // wait no more than 50 ms for handshake
 {
  if((instat & 0x8) == 0) break;
  delay(1);
 }
 if(t==0)
 {
  outctrl(PPSTRB_HI|_DIR);
  //asm sti;
  enable_Mtask();
  return(XMIT_ERR);
 }
 if(LO_PWR)
 {
  outctrl(PPSTRB_HI);      // Finish transfer
  outbyte(0xff);           // Drive bus high
 }
 else
  outctrl(PPSTRB_HI|_DIR);  // Complete byte transfer
 return NO_ERR;
}
//*******************************************************
// get_buswidth() is a special purpose function that is
// used to determine the printer port mode which a
```

```
// potentialy active kernel is using.  This function is
// what enables a users application to reliably start
// and interact with other applications when no command
// line arguments (causing a reset for example) are given.
//*********************************************************
MSGS get_buswidth(void)
{
 MSGS err;
 uchar n;
 int t,i;
 long shift;
 ulong CMD[4];
#define SIZELOC 0x809FFEL
 _DIR = 0;          // Use 'standard' nibble mode for partial readback
 WSCOUNT = 7;       //
 WSHIFT  =-4;       //
 CMD[0] = XREAD;    // Read command
 CMD[1] = 1;        // return data packet length is 1
 CMD[2] = SIZELOC;  // address of WSCOUNT
 CMD[3] = 1;        // srce indx
 disable_Mtask();
 for(i=0;i<4;i++)
 if((err = xmit_long(CMD[i]))!=NO_ERR)
 {
  enable_Mtask();
  return err;
 }
 WSCOUNT = 0;    //+--- partial receive of 16 bits using nibble mode
 for(shift=0;shift<16;shift-=WSHIFT)
 {
```

```
 for(t=10;t>0;t--)
 { if((instat & 0x8) == 0) break;
  if(t<=1)
  {
       enable_Mtask();
    WSCOUNT= 7;
    WSHIFT =-4;
    return RECV_ERR;
  }
 }
 outctrl(PPSTRB_LO | _DIR);
 if(WSHIFT == -8)  n = inbyte & 0xff; // Never executed
 else              n = innible & 0xf;
 WSCOUNT = WSCOUNT | (n << shift);
 outctrl(PPSTRB_HI | _DIR);
}
switch((uint)WSCOUNT)
{
 case 3: WSHIFT = -8;
       _DIR = 0xA0;
       break;          // good kernel, byte mode
 case 7: WSHIFT = -4;     // good kernel, nibble mode
                  //+---Finish receive in nibble mode
          _DIR=0;
      for(shift=16;shift<32;shift-=WSHIFT)
      {
       for(t=10;t>0;t--)
       { if((instat & 0x8) == 0) break;
        if(t<=1)
        {
```

```
      enable_Mtask();
      WSCOUNT=7;
      return RECV_ERR;
    }
   }
   outctrl(PPSTRB_LO | _DIR);
   if(WSHIFT == -8)  n = inbyte & 0xff; // Never executed
   else              n = innible & 0xf;
   WSCOUNT = WSCOUNT | (n << shift);
   outctrl(PPSTRB_HI | _DIR);
 }
 if(WSCOUNT==7) break;
     WSCOUNT = 7;
 WSHIFT = -4;
 enable_Mtask();
 return COM_ERR;
default:                // bad or incomplete kernel, nibble assumed
 WSCOUNT = 7;
 WSHIFT = -4;    //+---Finish receive in nibble mode
 _DIR = 0;
for(shift=16;shift<32;shift-=WSHIFT)
{
  for(t=10;t>0;t--)
  { if((instat & 0x8) == 0) break;
    if(t<=1)
    {
      enable_Mtask();
      WSCOUNT=7;
     return RECV_ERR;
    }
```

```
        }
            outctrl(PPSTRB_LO | _DIR);
        if(WSHIFT == -8)  n = inbyte & 0xff; // Never executed
        else              n = innible & 0xf;
        WSCOUNT = WSCOUNT | (n << shift);
        outctrl(PPSTRB_HI | _DIR);
       }
       WSCOUNT = 7;
       WSHIFT = -4;
       break;
}
if((err=getmem(SIZELOC+1,1,&WSHIFT ))==NO_ERR)
{
 if((WSCOUNT == 7) && (WSHIFT == -4))
 {
  _DIR = 0;
  enable_Mtask();
  return NO_ERR;
 }
 if((WSCOUNT == 3) && (WSHIFT == -8))
 {
  _DIR = 0xA0;
  enable_Mtask();
  return NO_ERR;
 }
 err = COM_ERR;
}
_DIR = 0;
WSCOUNT =  7;
WSHIFT  = -4;  // MUST return with a valid mode
```

```
 enable_Mtask();
 return err;
}
//*********************************************************************
// set_buswidth() is used to force the buswidth which the kernel
// will operate to a particular mode.  This function should not
// be called unless the kernel is known to be operational and not
// in use by other applications
//*********************************************************************
MSGS set_buswidth(void)
{
  MSGS err;
//if((BW_force == 8) && (WSCOUNT==7)) BW_force = 4;
  switch(BW_force)
  {
 case 0: WSCOUNT=3; WSHIFT=-8;
         _DIR = 0xA0;
         if((err=putmem(SIZELOC  ,1,&WSCOUNT))!=NO_ERR) return err;
         if((err=putmem(SIZELOC+1,1,&WSHIFT ))!=NO_ERR) return err;
         if((err=get_buswidth())==NO_ERR) return NO_ERR;
  case 4: WSCOUNT=7; WSHIFT=-4;       // Uni-dir-port
         _DIR = 0;
         if((err=putmem(SIZELOC  ,1,&WSCOUNT))!=NO_ERR) return err;
         if((err=putmem(SIZELOC+1,1,&WSHIFT ))!=NO_ERR) return err;
         if((err=get_buswidth())==NO_ERR) return NO_ERR;
         return err;
  case 8: WSCOUNT=3; WSHIFT=-8;       // Bi -dir-port
         _DIR = 0xA0;
         if((err=putmem(SIZELOC  ,1,&WSCOUNT))!=NO_ERR) return err;
         if((err=putmem(SIZELOC+1,1,&WSHIFT ))!=NO_ERR) return err;
```

```
        if((err=get_buswidth())==NO_ERR) return NO_ERR;
        return err;
case -1: return NO_ERR; // Do not modify buswidth
 }
 return COM_ERR;
}
//*********************************************************************
// getmem() reads a block of size 'length' from the DSK memory 'addr'
// to the hosts local memory 'data'
//*********************************************************************/
MSGS getmem(ulong addr, ulong length,ulong *data)
{
  int i;
  ulong CMD[4];
  MSGS err;
  if(( addr       & 0xF00000L) == 0xF00000L) return HOST_PORT;
  if(((addr+length) & 0xF00000L) == 0xF00000L) return HOST_PORT;
  CMD[0] = XREAD;    // Read command
  CMD[1] = length;   // data packet length
  CMD[2] = addr;     // srce addr
  CMD[3] = 1;        // srce indx
  disable_Mtask();
  for(i=0;i<4;i++)
   if((err = xmit_long(CMD[i]))!=NO_ERR)
   {
     enable_Mtask();
    return err;
   }
  for(;length>0;length--)
   if((err = recv_long(data++))!=NO_ERR)
```

```
  {
   enable_Mtask();
   return err;
  }
 enable_Mtask();
 return NO_ERR;
}
//*********************************************************************
// getmemlong() reads a block of size 'length' from the DSK memory 'addr'
// to the hosts local memory 'data' in 'long' variable.
//*********************************************************************
MSGS getmemlong(ulong addr, ulong length,long *data)
{
 int i;
 ulong CMD[4];
 MSGS err;
 if(( addr       & 0xF00000L) == 0xF00000L) return HOST_PORT;
 if(((addr+length) & 0xF00000L) == 0xF00000L) return HOST_PORT;
 CMD[0] = XREAD;    // Read command
 CMD[1] = length;   // data packet length
 CMD[2] = addr;     // srce addr
 CMD[3] = 1;        // srce indx
 disable_Mtask();
 for(i=0;i<4;i++)
  if((err = xmit_long(CMD[i]))!=NO_ERR)
  {
   enable_Mtask();
   return err;
  }
 for(;length>0;length--)
```

```
  if((err = recv_longo(data++))!=NO_ERR)
  {
   enable_Mtask();
   return err;
  }
 enable_Mtask();
 return NO_ERR;
}
//*************************************************************************
// putmem() writes a block of size 'length' to the DSK memory 'addr'
// from the hosts local memory 'data'
//*************************************************************************/
MSGS putmem(ulong addr,ulong length,ulong *data)
{
 int i;
 ulong CMD[4];
 MSGS err;
 if(( addr       & 0xF00000L) == 0xF00000L) return HOST_PORT;
 if(((addr+length) & 0xF00000L) == 0xF00000L) return HOST_PORT;
 CMD[0] = XWRIT;  // write command
 CMD[1] = length; // data packet length
 CMD[2] = addr;   // dest addr
 CMD[3] = 1;      // dest indx
 disable_Mtask();
 for(i=0;i<4;i++)
  if((err=xmit_long(CMD[i])) !=NO_ERR)
  {
   enable_Mtask();
   return err;
  }
```

```
  for(;length>0;length--)
   if((err=xmit_long(*data++))!=NO_ERR)
   {
    enable_Mtask();
    return err;
   }
  enable_Mtask();
  return NO_ERR;
}
MSGS RUN_CPU(void)
{
  ulong CMD;
  MSGS err;
  CMD = XRUNF;
  disable_Mtask();
  err = xmit_long(CMD);
  delay(10);
  enable_Mtask();
  return err;
}
//*********************************************************************
// HALT_CPU() Halts the DSK and then waits for a synchronization
// word.  This command places the CPU in a spin loop where it
// is waiting for a command.
//*********************************************************************
MSGS HALT_CPU(void)
{
  int t;
  ulong CMD;
  MSGS err;
```

```
 CMD = XHALT;              // Write link command
 disable_Mtask();
 if((err=xmit_long(CMD))!=NO_ERR)
 {
  enable_Mtask();
  return RECV_ERR;
 }
 for(t=500;t>0;t--)    // Allow time for context save in slow DSK
 { if(HPI_ACK()) break;
  delay(1);
 }
 if(t==0)
 {
  enable_Mtask();
  return RECV_ERR;
 }
 err = recv_long(&CMD);
 enable_Mtask();
 return err;
}
```

หนังสืออ้างอิง

1.การประมวลผลสัญญาณเชิงเลข (Digital Signal Processing) ,ศ.ดร.วัลลภ สุระกำพลธร, สำนักพิมพ์ไดนาพริ้นท์ จำกัด กรุงเทพฯ

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

3.TMS320C31 DSP Starter Kit User's Guide ,Texas Instrument Incorporated, USA

4.THE FAST FOURIER TRANSFORM AND ITS APPLICATIONS, E. ORAN BRIGHAM, Prentice-Hall International

5.Introductory Topics in Electronics and Telecommunication SIGNALS , F.R. Connor , Edward Arnold (Publishers)