ปีการศึกษา 2532

DIGITAL SIGNAL PROCESSING

โดย

นาย ทศทิศ จงจีรกาล 29-1066

อาจารย์ที่ปรึกษา

รศ. มนัส สังวรศิลป์

ปริญญานิพนธ์ ปีการศึกษา 2532 ภาคการศึกษาที่ 2

ภาควิชา ELECTRONICS

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

เรื่อง DIGITAL SIGNAL PROCESSING

ผู้จัดทำ ทศทิศ จงจีรกาล — 29-1066

..... อาจารย์ที่ปรึกษา

(ร.ศ มนัส สังวรศิลป์)

DIGITAL SIGNAL PROCESSING

โดย	นาย ทศทิศ	จงจีรกาล
อาจารย์ที่ปรึกษา	ร.ศ. มนัส	สังวรศิลป์
ปีการศึกษา	2532	

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ จะเป็นการศึกษาถึงการประมวลผลทาง DIGITAL โดยส่วนแรกจะศึกษาถึงกระบวนการแปลงสัญญาณอนาลอกไปเป็นสัญญาณดิจิตอล และการ INTERFACE ระบบดังกล่าวกับระบบ MICROCOMPUTER เพื่อที่จะใช้ MICROCOMPUTER เป็นหน่วยประมวลผลข้อมูลที่ได้จากระบบการแปลงนั้นและในส่วนที่สอง จะเป็นการศึกษาถึงระบบขั้นพื้นฐานที่ใช้ในการประมวลผลข้อมูล เพื่อเป็นแนวทางในการนำไปใช้งานให้เกิดประโยชน์ได้ต่อไป

ช่วงประมาณ 20 ปีที่แล้วจนถึงปัจจุบัน ปริมาณ COMPUTER ได้มีเพิ่มมากขึ้นพร้อมๆกับการพัฒนาทางด้าน DSP (DIGITAL SIGNAL PROCESSING) ซึ่งได้รับการพัฒนาให้มีประสิทธิภาพในการประมวลผลทางตัวเลขให้ดียิ่งขึ้น จนปัจจุบันนี้ถือได้ว่า DSP เป็นพื้นฐานและเป็นการประยุกต์เบื้องต้นทางด้าน MODERN INTEGRATED CIRCUIT TECHNOLOGY เช่น ชิพความเร็วสูง และงานทางด้านอื่นๆ เช่น

- PROCESSING OF SPEECH SIGNAL
- PROCESSING OF SEISMIC SIGNAL
- RADAR SIGNAL PROCESSING
- IMAGE PROCESSING
- ในด้านชีวแพทย์ เช่น เครื่องวัดความดันโลหิต , เครื่องวัดคลื่นหัวใจ เป็นต้น

ซึ่งจะเห็นได้ว่าเราสามารถประยุกต์ให้เกิดประโยชน์ได้มากมายดังนั้นปริญญานิพนธ์ฉบับนี้จึง เป็นการศึกษาถึงส่วนหนึ่งของ DSP ซึ่งเป็นพื้นฐานเบื้องต้นสำหรับศึกษาถึงการใช้งานในขั้นต่อไป

สารบัญ

หน้า

สารบัญ

หน้า

สารบัญ

บทนำ

การนำ DSP ไปใช้งาน

ช่วงประมาณ 20 ปีที่แล้ว ปริมาณ Computer ที่มีขนาดเล็กและมีประสิทธิภาพ ได้มีปริมาณเพิ่มขึ้นพร้อมๆกับการพัฒนาทางด้าน DSP ซึ่งได้รับการพัฒนาให้มีประสิทธิภาพในการประมวลผลทางตัวเลขให้ดียิ่งขึ้น จนในปัจจุบันนี้ถือได้ว่า DSP เป็นการประยุกต์เบื้องต้นและเป็นพื้นฐานทางด้าน Modern Integrated Circuit Technology เช่น พวกชิพความเร็วสูงและงานทางด้านอื่นๆ เช่น Processing of Speech Signal , Processing of Seismic Signals , Radar Signal Processing , Image Processing เป็นต้น ต่อไปจะยกตัวอย่างการนำไปใช้ในงานบางอย่าง ให้เห็นภาพพจน์ได้ชัดเจนยิ่งขึ้น

1) PROCESSING OF SPEECH SIGNAL

จุดมุ่งหมายของการนำ Digital Signal Processing มาใช้ในงานทางด้านนี้ก็คือ

1.1) STORING AND TRANSMITTING SPEECH SIGNAL

เพื่อเก็บและการสื่อสาร SPEECH SIGNAL เป้าหมายเบื้องต้นคือการพยายามลดจำนวนข้อมูลที่ต้องการจะส่งให้น้อยลง และการเปลี่ยนข้อมูลดังกล่าวกลับเป็นเสียงที่ตัวรับ เทคนิคอย่างหนึ่งที่เป็นไปได้ คือ การสุ่มสัญญาณเสียงด้วยค่าที่เหมาะสม และแปลงเป็นตัวเลขที่เหมาะสม นั่นคือการทำ A/D กระบวนการที่เหมาะสมที่ใช้ในการ TRANSMISSION ที่ต้องการความละเอียดน้อย คือการพิจารณากลุ่มของ PARAMETOR ที่ใช้ในการวิเคราะห์ระบบของเสียงนั้น เพื่อบอกถึงคุณลักษณะของกระบวนการพูดหรือ กระบวนการที่ทำให้เกิดเสียงนั้น แทนการ SAMPLING SPEECH SINGNAL

1.2) ENHANCING SPEECH SIGNAL

เพื่อปรับปรุงคุณภาพและทำให้สามารถเข้าใจใน SPEECH SIGNAL นั้น เพื่อปรับปรุงคุณภาพของเสียงพูดที่ถูกรบกวนจาก NOISE ซึ่งในงานบางอย่างจำเป็นต้องใช้ ไม่อย่างนั้นจะไม่สามารถฟังเสียงพูดได้อย่างชัดเจน

1.3) GENERTING OR SYNTHESIZING

เพื่อสร้างหรือสังเคราะห์เสียง จะสนใจการสร้าง WAVEFORM ใหม่ๆที่ใกล้เคียงกับเสียงพูดของคนซึ่งสามารถทำได้โดยการกระตุ้นอุปกรณ์กำเนิดเสียงที่กำหนดพารามิเตอร์ต่างๆไว้ เพื่อให้ได้เสียงต่างๆ ตามที่ต้องการ

1.4) SPEECH VERIFICATION AND IDENTIFICATION SYSTEM

เพื่อตรวจสอบเสียงของผู้พูดในงานบางอย่างที่สำคัญ ทำขึ้นเพื่อใช้ตรวจสอบ

เสียงของผู้พูด ใช้ในการป้องกันการเข้าถึงข้อมูลที่สำคัญใน COMPUTOR

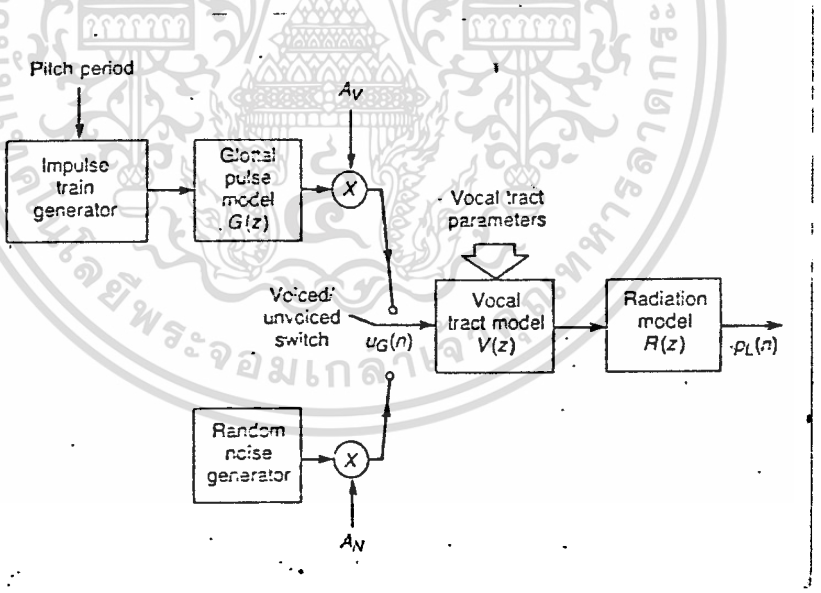
1.5) SPEECH RAGONITION SYSTEM

ให้สามารถจำคำต่างๆจากการป้อน I/P ได้และยังรวมถึงการพัฒนาระบบให้สามารถจำคำต่างๆได้มากๆ และอาจถึงขั้นพูดได้ตามที่ USER สั่ง

1.6) SPEECH MODIFICATION SYSTEM

ปรับรูปของเสียงให้อยู่อีกรูปแบบหนึ่งที่เหมาะสมสำหรับผู้พิการทางตาหรือหู

ในด้านของ Speech Analysis และ Speech Analysis System แบบจำลองที่ใช้ในการสร้างเสียงเลียนแบบ HUMAN สามารถแสดงให้เห็นได้ดังรูปที่ 1.1 ซึ่งเป็น BLOCK DIAGRAM ถ้าหากแบบจำลองนี้ถูกกระตุ้นโดยการรับค่าใดๆ ที่เหมาะสมในช่วงเวลาที่กำหนดก็จะทำให้ VOICE / UNVOICE SWITCH , VOCAL TRACT PARAMETER สร้าง O/P ที่ดีออกมาได้เหมือนกับเสียงที่ออกมาจาก ริมฝีปาก, ลำคอ, เพดานบน และล่างได้ ใน BLOCK ของ IMPLUSE TRAIN GENERATOR และ GLOTTAL PULSE MODEL จะทำหน้าที่ผลิตเสียงที่มีลักษณะของสระหลายๆตัวรวมกัน ส่วน RAMDOM NOISE GENERATOR จะทำหน้าที่ผลิตเสียงของพยัญชนะที่รวมกันอยู่ เช่น Sh, Ch และในส่วนของ VOCAL TRACT PARAMETER จะทำหน้าที่ DELAY หรือพักประมาณ 10-20 mS เพื่อทำการเปลี่ยนแปลงสัญญาณที่เข้ามา ทั้งหมดนี้เป็นพื้นฐานในระบบสังเคราะห์เสียง

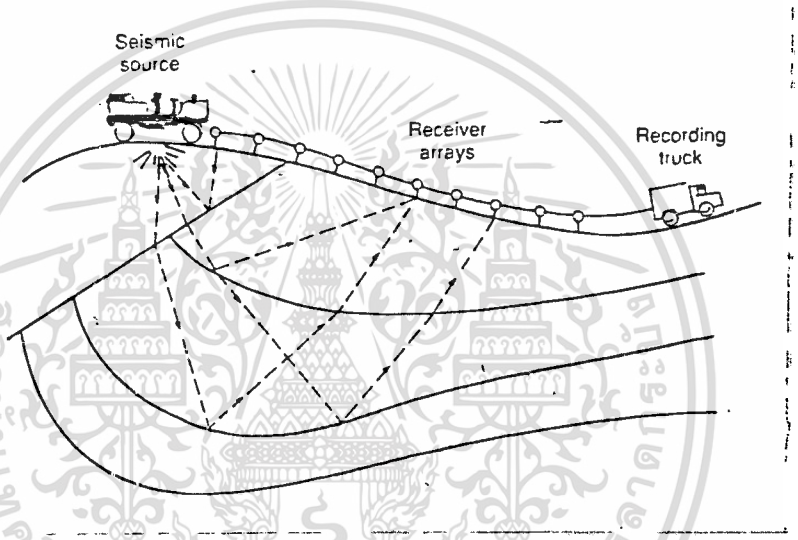


รูปที่ 1.1

2). PROCESSING OF SEISMIC SIGNALS

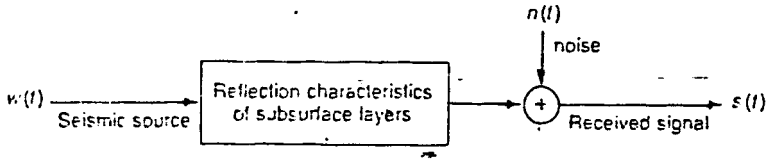
ในการขุดเจาะหาน้ำมันนั้น จำเป็นต้องใช้ค่าใช้จ่ายมากมายรวมทั้งเสียเวลามาก และสิ่งที่สำคัญที่สุดคือ เมื่อทำการขุดเจาะแล้วก็ควรจะเจอน้ำมัน จึงจะคุ้มค่ากับการลงทุน หนทางหนึ่งที่เราสามารถนำมาใช้ได้อย่างมีประสิทธิภาพคือ การนำเอาการประมวลผลสัญญาณมาช่วย เพื่อให้ทราบถึงรายละเอียดของโครงสร้างในชั้นหินและชั้นดิน ในพื้นที่ที่ต้องการจะทำการขุดเจาะ

กระบวนการที่ดีและสามารถนำมาใช้ได้อย่างมีประสิทธิภาพคือ การใช้ข้อมูลจาก Acoustic Sounding ร่วมกับรายละเอียดอื่นๆในการกำหนดและค้นหาอนาเขตของ บ่อน้ำมันและก๊าซธรรมชาติ ซึ่งการสำรวจนั้นสามารถอธิบายได้ดังรูปที่ 2.1



รูปที่ 2.1

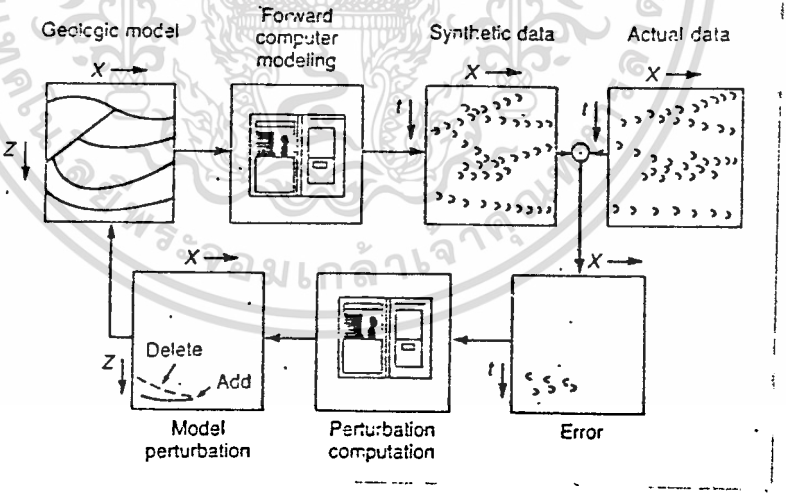
เราสามารถที่จะแทนความสัมพันธ์ระหว่าง Transmitted Seismic Source และ Receive Signal โดย Block Diagram ในรูปที่ 2.2 สัญญาณ S(t) จะถูกนำมาคำนวณตามฟังชั่นต่างๆ เพื่อให้เกิดการประมาณค่าที่ดี ทำให้เราทราบถึงลักษณะของพื้นผิวในชั้นหินและชั้นดินต่างๆจากการสะท้อนกลับของสัญญาณ ผลเหล่านี้จะบอกให้ทราบถึงความเป็นไปได้ ในลักษณะโครงสร้างทางภูมิศาสตร์ว่าจะมีน้ำมันหรือไม่ การพิจารณาผลของสัญญาณที่สะท้อนกลับมานั้นเราใช้กระบวนการ Deconvolution แบบจำลองทางธรณีวิทยาแสดงให้เห็นถึง คุณสมบัติการสะท้อนของพื้นผิวในชั้นต่าง ซึ่งสามารถตัดสินได้โดยใช้กระบวนการในรูป 2.3



รูปที่ 2.2

แบบจำลองลักษณะของชั้นดินและชั้นหินต่างของพื้นผิวโลก สร้างขึ้นมาจากการสังเคราะห์ข้อมูล ซึ่งเปรียบเทียบกับการวัดจริง ถ้าหากค่าที่ได้จากการสังเคราะห์กับค่าที่มาจากการวัดจริง ประมาณแล้วมีค่าใกล้เคียงกัน แบบจำลองนั้นถือว่ายอมรับได้ แต่หากแตกต่างกันเกินค่านัยสำคัญค่าหนึ่ง ค่าความแตกต่างนี้จะถูกนำไปคำนวณเพื่อบอกความคลาดเคลื่อน หลังจากปรับแบบจำลองจนได้ที่แล้วก็จะทำกระบวนการดังกล่าวซ้ำจนกระทั่ง Synthesized และ Measure Data ทั้งสองอย่างนี้ Match กัน

นอกจากนี้ Seismic Signal Processing ยังสามารถใช้ทำนายการเกิดแผ่นดินไหวได้อีกด้วย

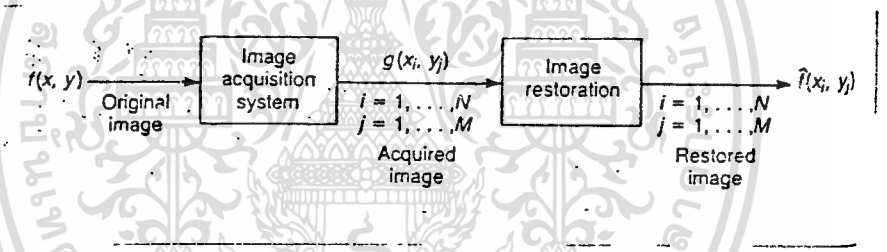


รูปที่ 2.3

3). IMAGE PROCESSING

ตัวอย่างของ Image Processing ที่เราคุ้นเคยก็คือการ Reproduction ของ Image ที่ส่งมาจากยานอวกาศ ซึ่งอาจจะเป็น Image ของ โลก,ดวงจันทร์ ,ดวงดาวที่อยู่ห่างไกล หรือภาพถ่ายทางอากาศจากดาวเทียมที่เราเคยเห็นในข่าว Image เหล่านี้ได้มาจาก X-Ray ,อุลตร้าซาวด์ ซึ่งใช้มากในการตรวจและรักษาอาการเจ็บป่วย , ส่วนในทางดาวเทียมก็จะใช้ใน การสำรวจแร่ธาตุ , การเพาะปลูก , การสำรวจใต้ท้องมหาสมุทร

ก่อนที่จะนำเอา Image เหล่านี้มาดำเนินการตาม Process จำเป็นต้องมีการแปลง Image เหล่านี้ให้อยู่ในรูปแบบ Digital ที่เหมาะสมเพื่อที่จะป้อนเข้าสู่คอมพิวเตอร์ได้ รวมทั้งสามารถเก็บบันทึกไว้ได้ด้วยกระบวนการประมวลผลจะใช้ Image Acquisition System ซึ่งจะใช้วิธีการทำนายและการสุ่มเพื่อลดการบิดเบือนของ Image ที่เก็บหรือรับเข้ามาได้ ซึ่งจำเป็นต้องใช้กรรมวิธีทาง Digital ให้ถูกต้องและเหมาะสมกับ Image แต่ละอย่าง เพื่อที่จะนำไปผ่านขั้นตอนการคำนวณเพื่อให้ได้ค่าข้อมูลที่ใกล้เคียงกับความจริง



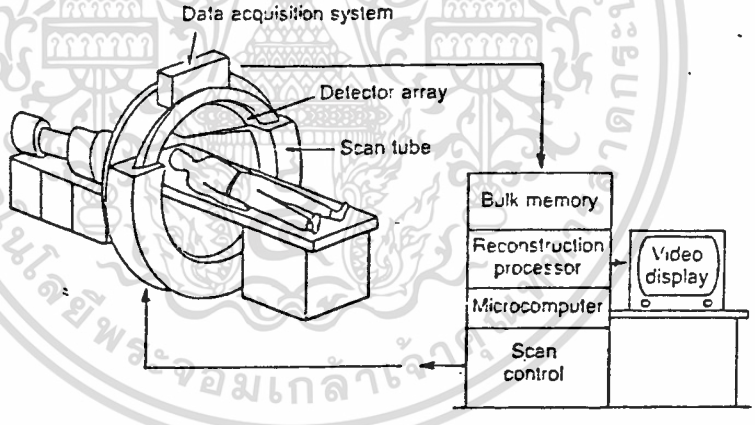
รูปที่ 3.1

วิธีการหนึ่งที่ใช้ในขั้นตอนการคำนวณเพื่อหา Image Restoration ซึ่งเป็นการลดการรบกวนและความบิดเบือน ที่ติดมากับ Original Image โดยใช้ Aquisition Process ผลของกระบวนการนี้จะทำให้ได้ Image ที่ใกล้เคียงกับความเป็นจริงมากที่สุด ซึ่งกระบวนการนี้สามารถแสดงให้เห็นได้ในรูปที่ 3.1 โดยที่ X,Y เป็น Coordinate ของ Image Plane ซึ่ง Image ที่รับเข้ามาจะอยู่ในเซทของ X_i;i=0 to N ,Y_j ; j=0 to M เป้าหมายของ Restoration Process ก็คือการทำให้ $F(X_i,Y_j)$ ใกล้เคียงกับ $f(X_i,Y_j)$ มากกว่า $G(X_i,Y_j)$ ซึ่งจะต้องใช้ขั้นตอนการคำนวณมากมายหลายขั้น

นอกจากนี้ยังมีกระบวนการอื่นๆอีก เพื่อนำเอา Image มาปรับปรุงให้เกิดประโยชน์ ซึ่งเรารู้จัก Process เหล่านี้ในลักษณะของ Image Enhancement ซึ่ง

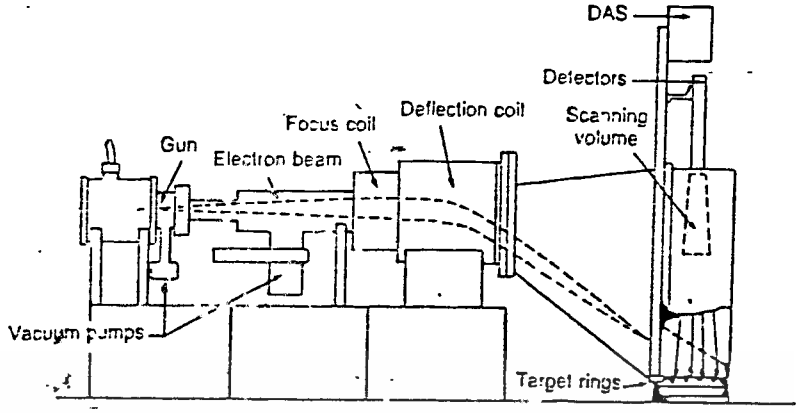
ผลของ Process นี้จะไม่ทำให้ค่าของ Image มาใกล้เคียงกับ Original Image ซึ่งโดยทั่วๆไปแล้วกระบวนการดังกล่าวนี้ได้รับความไว้วางใจในงานหลายอย่าง แต่สิ่งที่ควรพิจารณาคือ เป้าหมายและจุดประสงค์ของ Process นั้นๆ ซึ่งบางอย่างก็สามารถนำไปใช้ได้ดี แต่บางอย่างก็ไม่ดี

กระบวนการสุดท้ายที่จะกล่าวถึงในที่นี้ คือ การ Reproduction Image จาก Projection ซึ่งได้ถูกนำไปใช้ในทางการแพทย์ เพื่อดูโครงสร้างภายในของร่างกายมนุษย์ ซึ่งสามารถแบ่งดูเป็นส่วนๆ (Section) ได้ กระบวนการนี้จะอาศัยการฉายอุลตร้าซาวด์ หรือ X-Ray เช่น การใช้เทคนิคที่เรียกว่า CT (Computerized Tomography) ซึ่งตัวอย่างอยู่ในภาพที่ 3.2 ภาพนี้เป็นระบบ CVCT หรือ Cadiovascular Computerized Tomographic Scaner ซึ่งเครื่องนี้ใช้การเบี่ยงเบนของลำอิเลคตรอนโดยอาศัยสนามแม่เหล็กให้มีการสแกนอย่างรวดเร็ว และมีความสัมพันธ์กันระหว่างการเคลื่อนที่ของลำอิเลคตรอน กับ Machinical Beem Scaning อุปกรณ์นี้จะใช้กับอวัยวะที่ไม่ปกติ (เพื่อตรวจรักษา) หรือใช้ในการวางแผนเพื่อทำการผ่าตัด โดยดูตำแหน่งที่จะทำการผ่าให้แน่นอน



(a) Schematic design of the CVCT scanner system

รูปที่ 3.2



(b) Cross section of the CVCT scan tube indicating the beam focusing and steering method and the target configuration

รูปที่ 3.2 (ต่อ)

บทที่ 1

ทฤษฎีของ DATA ACQUISITION AND CONVERTION

1.1) บทนำ

รูปแบบสัญญาณไฟฟ้าที่เราพบเห็นคุ้นเคยในชีวิตประจำวัน จะอยู่ในรูปของสัญญาณ Analog ซึ่งแต่เดิมการนำเอาสัญญาณไฟฟ้าดังกล่าวมาประมวลผลก็จะกระทำในรูปของสัญญาณ Analog นั่นเอง แต่เมื่อเทคนิคการประมวลผลทาง Digital ได้รับการพัฒนาขึ้น ก็พบว่าในรูปของ Digital การเก็บการประมวล การสื่อสาร และการแสดงผล กระทำได้ง่ายและมีประสิทธิภาพมากกว่า ดังนั้นการเปลี่ยนรูปแบบของสัญญาณจึงได้มีความจำเป็นขึ้น จากสัญญาณ Analog ที่มีอยู่ตามธรรมชาติจะถูกเปลี่ยนมาเป็นสัญญาณ Digital โดย Analog to Digital Convertors (ADC) และประมวลโดยตัวประมวลผลทาง Digital เช่น Computer จากนั้นจะถูกนำมาแสดงผลเลยหรือเปลี่ยนกลับมาให้อยู่ในรูปของสัญญาณ Analog ที่คุ้นเคยและใช้งานได้ง่ายกว่าโดย Digital to Analog Convertors (DAC) การเปลี่ยนแปลงทางกายภาพใดๆ ก็ตาม เช่น ความดัน อุณหภูมิ จะถูกเปลี่ยนให้เป็นสัญญาณ Analog โดยทรานสดิวเซอร์ที่มีคุณสมบัติเหมาะสมกับรูปแบบทางกายภาพนั้น สัญญาณไฟฟ้านั้นจะถูกปรับให้อยู่ในรูปแบบ และขนาดที่เหมาะสมก่อนโดย Analog Signal Conditioner เช่นวงจรขยาย หรือ ฟิลเตอร์ เป็นต้น

ในระบบที่ต้องประมวลในเวลาเดียวกันทีหลายข้อมูล แต่เนื่องจาก ADC ทำงานได้เร็วพอ จึงไม่จำเป็นต้องใช้ ADC หลายๆ ตัวทำงานแยกกันแต่ใช้วิธีแบ่งเวลา (Timesharing) โดยวิธี Multiplexing วงจร Sample and Hold จะสุ่ม (Sample) ขนาดของสัญญาณ Analog มาและ Hold ไว้ชั่วขณะเพื่อรอให้ ADC รับไปเปลี่ยนให้เป็นระหัส Digital จนเรียบร้อยก่อน แล้วค่อยรับสัญญาณใหม่ ทั้งนี้เพื่อไม่จำเป็นต้องใช้ ADC ที่ต้องทำงานเร็วมากนัก ข้อมูลที่ทำการแปลงได้จะถูกส่งต่อไปยัง System Bus เพื่อทำการประมวลผลโดย Processor แล้วทำการเปลี่ยนกลับมาเป็นสัญญาณ Analog โดย DAC เพื่อควบคุมกิจการทางกายภาพของระบบผ่าน Analog Actuator

1.1) ทฤษฎีการ Sample

ในการแปลงสัญญาณ Analog เป็น Digital นั้น ADC จะต้องใช้เวลาช่วงหนึ่งในการจัดการ ซึ่งช่วงเวลาดังกล่าวขึ้นอยู่กับหลายแฟคเตอร์ เช่น

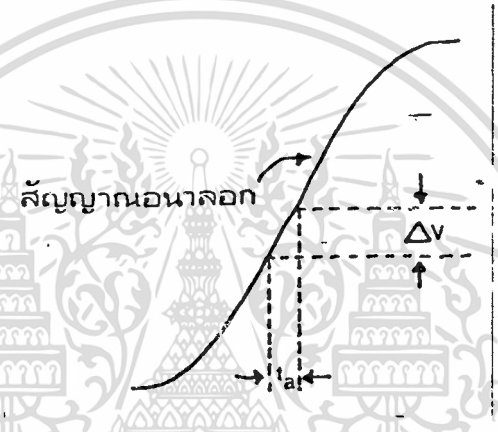
1. ความละเอียดของการเปลี่ยนสัญญาณ
2. เทคนิคของการแปลงสัญญาณ

3. ความเร็วในการทำงานของอุปกรณ์ร่วมอื่นๆ ในวงจรของการแปลงสัญญาณนี้จำเป็นสำหรับการใช้งานเฉพาะอย่างและ ความแม่นยำที่ต้องการ

Aperture Time : คือช่วงเวลาในการแปลงสัญญาณ ซึ่งคำว่า Aperture Time โดยทั่วไปหมายถึงช่วงเวลาที่เกิดความไม่แน่นอนในการวัด และผลก็คือเกิด error ต่อค่าที่วัดได้

ในรูปที่ 1.1 สัญญาณ Analog V(t) มีอัตราการเปลี่ยน dV/dt ในช่วง Aperture Time t_a ดังนั้นช่วงการเปลี่ยนแปลง Analog จะเท่ากับ V โดย

$$\Delta V \leq t_a dV(t)/dt \quad \text{----------(1.1)}$$



รูปที่ 1.1 แสดง error จากการวัดใน Aperture Time

ดังนั้นหากเวลาที่ ADC ใช้ในการเปลี่ยนสัญญาณในเวลา t_a นี้ระหัสของ Digital ที่ได้อาจจะตรงกับขนาดของสัญญาณ Analog ค่าใดค่าหนึ่งในช่วงนี้ และส่วนอื่นๆ ที่เหลือคือ error ที่เกิดขึ้น ซึ่งแน่นอนในบางครั้งเป็นไปได้ที่ระหัส Digital จะตรงกับค่า Analog ที่ถูกต้อง

1.2) Sample and Hold และ Aperture error

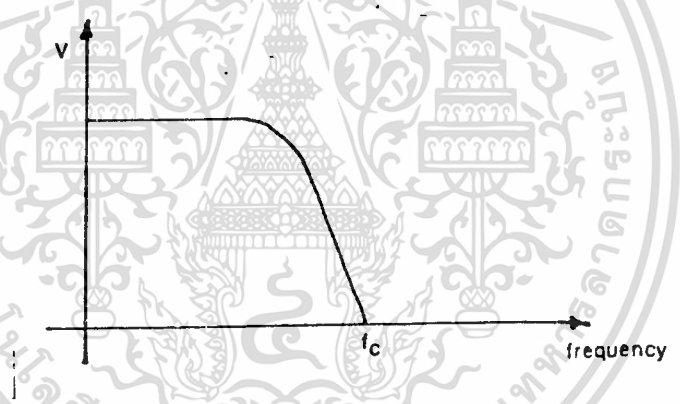
วงจร Sample and Hold จะทำการสุ่มสัญญาณอินพุท และนำสัญญาณนั้นมาเก็บหรือ Hold ไว้ในเวลาหนึ่งได้ ซึ่งส่วนใหญ่จะใช้การประจุแรงดันนั้นในตัวเก็บประจุที่รั่วไหลต่ำ ดังนั้นในเมื่อแรงดันอินพุทสามารถคงอยู่ได้นานพอ ทำให้ ADC ไม่จำเป็นต้องมีเวลาในการแปลง(Convertion Time) อย่างรวดเร็วนัก Aperture Time ของ Sample and Hold คือเวลาตั้งแต่เริ่มสุ่มสัญญาณจนตัวเก็บประจุมีค่าแรงดัน จนถึงค่าที่สุ่ม ซึ่งสำหรับ Sample and Hold แล้ว Aperture time ขึ้นอยู่กับ Band-

width และ Switching Time ของอุปกรณ์แอคทิฟที่ใช้ในวงจร ซึ่งหาและสร้างได้ง่าย และราคาถูกกว่าการสร้าง ADC ความเร็วสูงมีปัญหาว่าอัตราการสุ่มสัญญาณนั้นควรมีขนาดเท่าใด ที่จะไม่ทำให้ข้อมูลสูญเสียไปเมื่อสัญญาณนั้นถูกเปลี่ยนกลับมาเช่นเดิม อันนี้ขึ้นกับความถี่ของสัญญาณ Analog และทฤษฎีการสุ่มกล่าวไว้ว่า

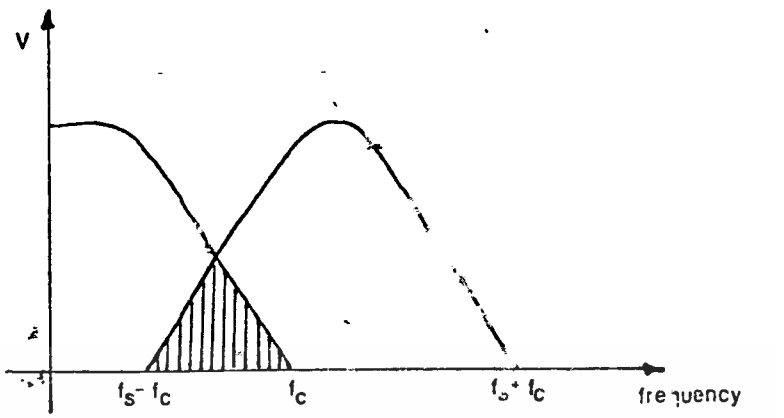
" ถ้าสัญญาณต่อเนื่องซึ่งมีความถี่และฮาร์โมนิคส์ไม่เกิน f_c แล้ว สัญญาณดังกล่าวจะสามารถเปลี่ยนกลับมาได้อย่างเดิม โดยไม่สูญเสียรายละเอียดหรือเพี้ยนไป ถ้าอัตราการสุ่มไม่น้อยกว่า $2f_c$ ต่อ Sec "

1.3) **Frequency folding and Aliasing**

จากทฤษฎีการสุ่มสามารถอธิบายด้วยลักษณะรูป Spectum ของสัญญาณในรูปที่ 1.2 แสดงให้เห็น Spectum ของสัญญาณที่ถูกสุ่มซึ่ง Bandwidth ไม่เกินกว่า f_c ในขณะที่สัญญาณนี้จะถูกสุ่มด้วยความถี่ f_s ขบวนการ Moduration จะทำให้แถบ Spectum ของสัญญาณสุ่มขยายกว้างออกจาก f_s เป็น $2f_s$, $3f_s$,... ได้เป็นดังรูปที่ 1.3



รูปที่ 1.2 แสดง Spectum ของสัญญาณ Analog ที่จะถูกสุ่ม



รูปที่ 1.3 หลังจากการสุ่มเกิด Frequeney folding

ถ้าความถี่ของสัญญาณสุ่ม f_s ไม่สูงพอหลังจากการสุ่ม Spectum บางส่วนของ f_s จะมาซ้อนกับ Spectum ของสัญญาณ ซึ่งเรียกว่า Frequency folding หากเป็นเช่นนี้ก็จะทำให้เกิดความเพี้ยนแก่สัญญาณ Analog จากการซ้อนกันของ Spectum เมื่อสัญญาณถูกเปลี่ยนกลับให้อยู่ในรูปเดิม และถ้าเลื่อนความถี่ของการสุ่มให้สูงขึ้นจนโอกาสการซ้อนของ Spectum หมดไป $(f_s - f_c) = f_c$ จะทำให้การเปลี่ยนกลับของสัญญาณหลังจากถูกสุ่มก็ยังคงเหมือนเดิมได้

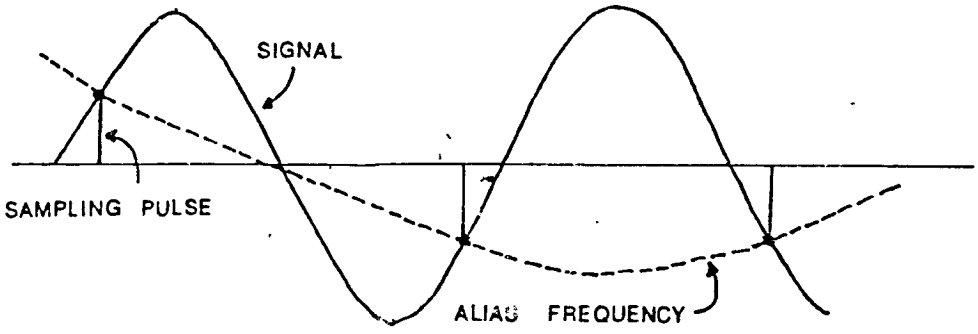
จากที่กล่าวมาแสดงการสนับสนุนทฤษฎีการสุ่มที่ว่าให้ $f_s \geq 2f_c$ นั่นคือการกำจัดการซ้อนกันของ Spectum ได้สองวิธีคือ

1) ใช้อัตราการสุ่มที่สูงพอ

2) การทำ Filter ความถี่ของสัญญาณ Analog ก่อนการสุ่มเพื่อให้ Bandwidth ไม่เกินไปกว่า $f_s/2$

ในทางปฏิบัติแล้วจะยังคงเกิด Frequency Folding ได้เสมอจากส่วนฮาร์โมนิคส์ของสัญญาณ รวมทั้ง Spectum ของสัญญาณรบกวนที่ยังคงอยู่แม้ว่าจะทำการ Filter ก่อนหน้ามาแล้วก็ตามการกำจัดการซ้อนกันของ Spectum นี้วิธีที่ได้ผลคือพยายามให้การสุ่มสัญญาณเป็นไปอย่างรวดเร็วมากที่สุด

ผลของการใช้อัตราการสุ่มที่ไม่เหมาะสมอีกประการหนึ่งที่เกิดขึ้นดังรูปที่ 1.4 เรียกว่า Alias Frequency ซึ่งเกิดกับสัญญาณที่เปลี่ยนกลับมาเช่นเดิมหลังจากถูกสุ่มแล้ว



รูปที่ 1.4 การเกิด Alias Frequency จากการสุ่มด้วยความถี่ต่ำกว่า 2 เท่า ของสัญญาณ Analog

1.4) Analog to Digital Convertor

ลักษณะการจัดวงจร ADC มีหลายแบบ แต่ที่นิยมมีเพียงไม่กี่แบบ และส่วนใหญ่จะอยู่ในรูปของวงจรรวม ในที่จะกล่าวถึงเฉพาะแบบ SAR

SAR (Successive Approximation Register)

วงจร ADC นี้เป็นเทคนิคที่ได้รับความนิยมในงานประยุกต์ ที่ต้องการ ความเร็วสูงและปานกลาง การจัดวงจรจะคล้ายกับ Counter ที่ทำงานในลักษณะป้อนกลับ รูปที่ 1.5 แสดง Block Diagrame ของ SAR คอมพาราเตอร์จะคอยเปรียบเทียบเอาท์พุทจาก ADC กับ Analog อินพุท V_{in} ส่วนเอาท์พุทจะไปควบคุม Successive Approximation Register ซึ่งเป็นไอซี MSI ที่ได้รับการออกแบบเป็นพิเศษเพื่อทำหน้าที่นี้โดยเฉพาะการทำงานของ SAR เป็นดังนี้

ในรูปที่ 1.6 แสดง Timing Diagram ของ ADC ที่มีระดับ Analog 1 และ 2 พิจารณาที่ระดับ 1 เมื่อ Clock เข้าไปหนึ่งลูกจะทำให้ MSB (bit 1) เป็น 1 แต่ทุกบิทยังคงเป็นศูนย์อยู่ DAC จะเปลี่ยนเอาท์พุทของ SAR เป็น Analog เปรียบเทียบกับสัญญาณ Analog อินพุท ถ้าผลการเปรียบเทียบที่คอมพาราเตอร์น้อยกว่าอินพุทให้คงบิทนั้นเป็น 1 ไว้ แต่ถ้ามากกว่าให้บิทนั้นเป็นศูนย์จากนั้นทำการทดสอบบิทต่อไป โดยทำให้เป็น 1 หากผลรวมของสองบิทหรือบิทหลังมากกว่าก็ทำให้บิทนั้นเป็นศูนย์ แต่ถ้าน้อยกว่าให้คง 1 เอาไว้แล้วทดสอบบิทถัดไป กรรมวิธีดังกล่าวจะกระทำไปจนครบทุกบิทหรือจนกว่าเอาท์พุทจะต่างจาก V_{in} ไม่เกิน 1 LSB

มีข้อจำกัดอันหนึ่งสำหรับการ Convert คือสัญญาณ Analog อินพุท จะต้องคงที่ในช่วงเวลาที่ทำการเปลี่ยนแปลงสัญญาณโดยเปลี่ยนได้ไม่เกิน 1/2 LSB และในช่วงสุดท้ายของการเปลี่ยนสัญญาณ Digital output จะออกมาชนานกันทุกบิท แต่บางแบบจะให้เอาท์พุทออกมาในลักษณะอนุกรม

วงจร ADC แบบนี้สามาทำงานได้สองโหมดคือ

1) โหมดที่ทำงานโดยอิสระ (Free run)

2) โหมดที่รอคำสั่ง Start Convertion จากภายนอก

เวลาที่ใช้ในการเปลี่ยนสัญญาณจะใช้ (n+1) ลูกของ Pulse Clock โดย Clock ลูกแรกจะใช้ในการรีเซทรีจิสเตอร์ภายใน

บทที่ 2
ทฤษฎีเบื้องต้นของ DIGITAL SIGNAL PROCESSING

2.1) Z-transform

ในทาง DSP เราใช้คุณสมบัติของ Z-transform ในการเปลี่ยนแปลงจาก Linear Difference Equation ไปเป็น Algebratic Equation ซึ่งทำให้เราสามารถวิเคราะห์ระบบได้อย่างมีประสิทธิภาพทั้งทางด้าน การตอบสนองและคุณลักษณะ (Characteristics) ของระบบได้โดยอาศัยกระบวนการทางคณิตศาสตร์ธรรมดาๆ ในที่นี้จะอธิบายถึงการนำไปใช้งานในด้านการวิเคราะห์ระบบ

Z-transform ของอนุกรม x(n) ถูกกำหนดดังนี้

$$\mathcal{Z}[x(n)] = x(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \qquad \text{----------(2.1)}$$

ซึ่ง Z เป็น Complex variable เราสามารถกระจายเทอมออกมาให้เห็นได้ดังนี้

$$x(z) = \ldots + x(-203)z^{203} + \ldots + x(-1)z + x(0)z^{0} + x(1)z^{-1} + \ldots + x(159)z^{-159} + \ldots \qquad \text{--------(2.2)}$$

จากสมการข้างต้นนี้เราจะเห็นได้ว่า Z เป็นตัวบ่งชี้ถึงตำแหน่งของแต่ละ Sample (หมายถึง ในสัญญาณที่เป็นแบบต่อเนื่องเมื่อมาพิจารณาในระบบ Discrete จะมีการ Sample สัญญาณดังกล่าวเป็นช่วงๆ ทำให้ได้อนุกรมของ Sample ขึ้นมา) และสัมประสิทธิ์ของ Z จะบอกถึง Amplitude ของแต่ละ Sample

หากอนุกรมของ Sample ประกอบด้วย Single Sample ที่ n=0 ดังรูปที่ 2.1(a) สามารถเขียนสมการได้ดังนี้ x(n) = x(0) (n) และจากสมการที่ 2.1 Z-transform คือ

$$\mathcal{Z}[x(0)\delta(n)] = X(0)z^{0} = X(0) \qquad \text{----------(2.3)}$$

จะเห็นได้ว่ามีการ Sample ที่ n=m และจากรูปที่ 2.1(b) เราจะบอกได้ว่า $x(n) = x(m)\delta(n-m)$; $m>0$ และมี Z-transform ดังนี้

$$\mathcal{Z}[x(m)\delta(n-m)] = x(m)z^{-m} \qquad \text{----------(2.4)}$$

และในรูป 2.1(c) สามารถเขียนอนุกรมแทนได้ดังนี้

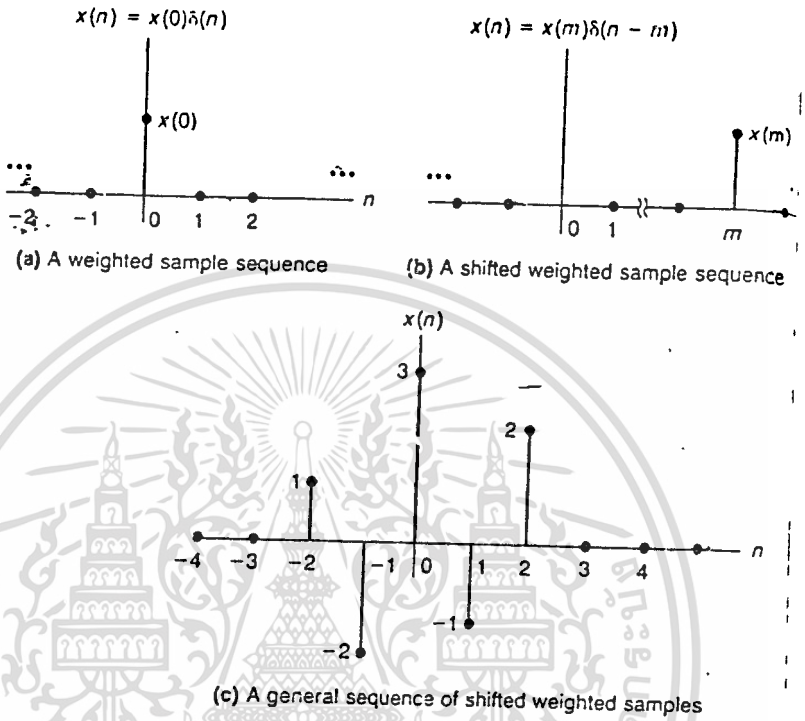
$$x(n) = \{ \ldots,0,0,1,-2,3,-1,2,0,0,\ldots \} \qquad \text{---------(2.5)}$$

สามารถอธิบายในลักษณะของ Shifted Weighted Samples ได้ดังนี้

$$x(n) = \delta(n+2) - 2\delta(n+1) + 3\delta(n) + \delta(n-1) + 2\delta(n-2)$$

ทำเป็น Z-transform ได้ดังนี้คือ

$$X(z) = \sum_{n=-\infty}^{n=+\infty} \{ \delta(n+2) - 2\delta(n+1) + 3\delta(n) - \delta(n-1) + 2\delta(n-2) \} z^{-n} \quad \text{----------(2.6)}$$



รูปที่ 2.1

เรารู้ว่าค่า $A\delta(n-m) = A$ ที่ $n=m$ ดังนั้นสมการที่ 2.6 ก็จะเขียนได้เป็น

$$X(z) = z^{2} - 2z^{1} + 3z^{0} - z^{-1} + 2z^{-2} \quad \text{----------(2.7)}$$

สังเกตดูจะเห็นได้ว่า z^{-1} จะแทนการ Delay ของ Sample หนึ่งๆ และ z^{+1} จะแทนการ Advance ของ Sample นั้นๆเช่นกันทำให้ได้สมการที่เกี่ยวกับการ Shifted Weighted Sample คือ

$$[A\delta(n-m)] = Az^{-m} \quad \text{----------(2.8)}$$

โดยที่ A แทนขนาดหรือน้ำหนักของ Sample นั้นๆ สามารถสรุปได้เกี่ยวกับ Z-transform ได้ดังนี้

1. X(z) เป็นโพลิโนเมียลหรือเป็น Power Series ในรูปของ z และ

หาได้จากอนุกรม {x(n)}

2. ในระบบการ Sample , X(z) จะมีรูปแบบเป็นอิสระในการ Sample ในคาบเวลา T แต่ Factor z^{-n} จะมีความสัมพันธ์กับ X(n) โดย t = nT ซึ่งโดยลักษณะนี้ z^{-n} จะหมายถึงการ Delay nT วินาทีจากเวลา t=0

2.2) Linear Time Invariant System (LTI)

Differrence Equation for Nth-Order Systems

โดยทั่วไปแล้ว LTI System จะสามารถอธิบายได้ในรูปแบบของสมการต่อไปนี้

$$Y_n = a_1y(n-1) + a_2y(n-2) +...+ a_Ny(n-N) + b_ox(n) + b_1x(n-1) +...+ b_Lx(n-L) \quad \text{------(2.9)}$$

ซึ่ง a' กับ b' เป็นค่าคงที่จำนวนจริงใดๆ และค่า N เป็นค่าใด ๆ ที่เป็นไปได้ที่สามารถที่จะนำมาใช้ใน Delay Term Output y(n) , ค่า L เป็นค่าใดๆที่เป็นไปได้ที่สามารถนำมาใช้ใน Delay Term Input x(n) ด้วยคุณสมบัตินี้ ทำให้เรียกสมการนี้ว่า Nth-Order Difference Equation เนื่องจาก O/P y(n) จะขึ้นกับ 1^{st}, 2^{nd} and N^{th} ของ Output จากสมการบนเราสามารถเขียนสมการให้ดูง่ายขึ้นได้ดังนี้

$$y(n) = \sum_{k=1}^{k=N} a_k y(n-k) + \sum_{k=0}^{k=L} b_k x(n-k) \quad \text{--------(2.10)}$$

สมการนี้จะแทน Discrete System ที่เรียกว่า Recursive System ทั้งนี้เนื่องจาก Output ของมันจะขึ้นกับ Output ก่อนหน้านี้

และสมการหนึ่งที่สำคัญคือ

$$y(n) = \sum_{k=0}^{k=L} b_k x(n-k) \quad \text{----------(2.11)}$$

สมการนี้เป็นสมการ Nonrecursive เนื่องจากมิได้มีการนำค่า O/P ก่อนหน้านี้เข้ามาใช้ในการคำนวนด้วย

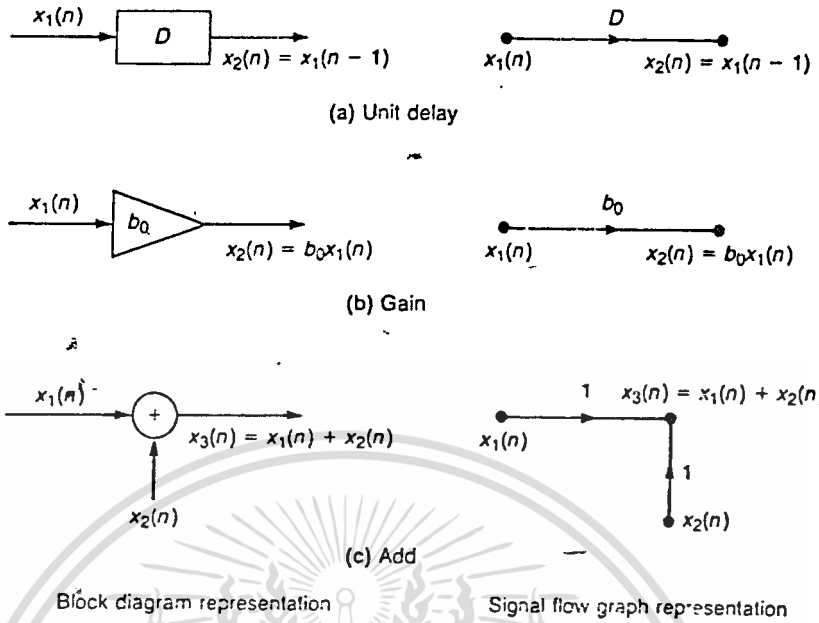
System Diagram

ในการพิจารณาระบบ วิธีการหนึ่งที่นิยมนำมาใช้เพื่อทำให้ระบบดังกล่าวดูง่ายและเห็นภาพพจน์ได้ดีขึ้นมากกว่าการพิจารณาเพียงสมการอย่างเดียว วิธีการดังกล่าวคือ System Diagram ซึ่งเป็นการแทนสมการด้วยสัญลักษณ์ที่กำหนดไว้ และมีพฤติกรรมเป็นไปตามรูปแบบของสมการนั้นๆ

หากเรามีสมการหนึ่งดังต่อไปนี้

$$y(n) = \sum_{k=1}^{k=N} a_k y(n-k) + \sum_{k=0}^{k=L} b_k x(n-k) \quad \text{----------(2.12)}$$

เราสามารถที่จะใช้สัญลักษณ์ดังรูปเหล่านี้แทนได้

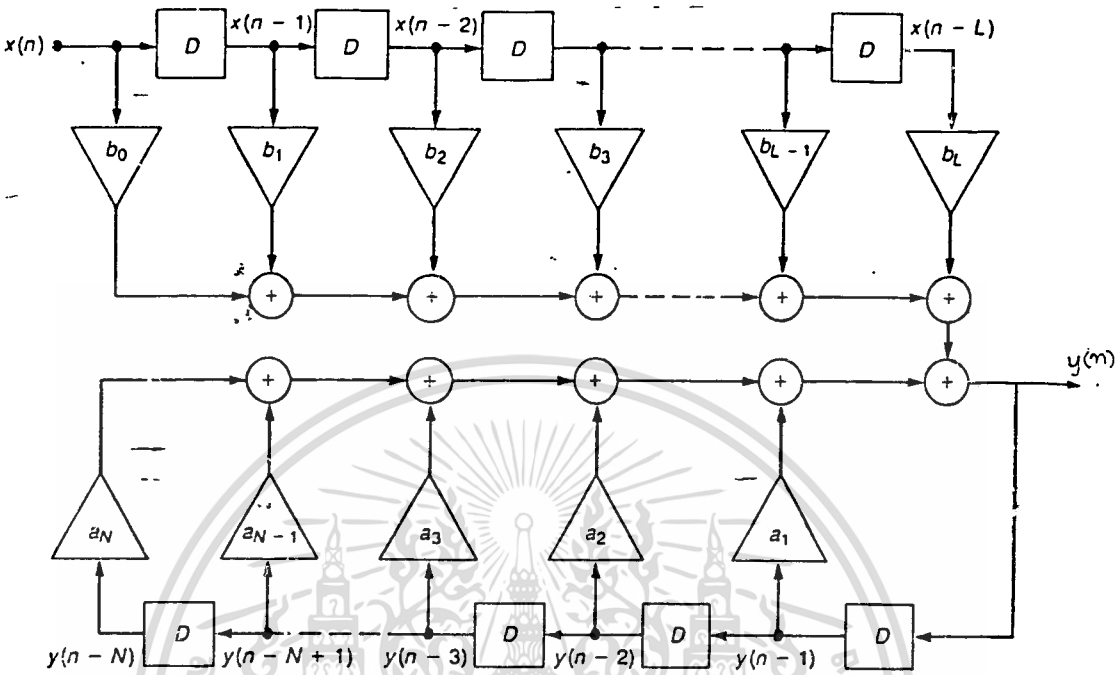


รูปที่ 2.2 System Diagram Block

เมื่อนำมาสร้างเป็น System Diagram ตามสมการ 2.12 จะได้ระบบดังรูป 2.3

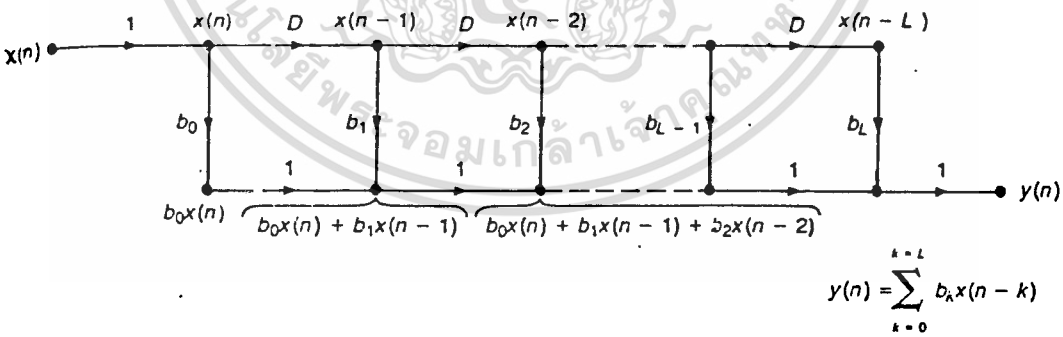
ในการนำไปใช้งานทาง Digital Filter นั้น ระบบหนึ่งเมื่อสร้างเป็น System Diagram แล้ว จะแตกต่างกันไปตามสถานะของระบบที่ใช้ในการพิจารณา ในรูป 2.3 ครึ่งบนจะแสดงเทอมของ Input และครึ่งล่างจะแสดงเทอมของ Output ในสมการเดียวกัน ซึ่งเป็นสมการในระบบ Recursive

ในระบบ Nonrecursive ไม่มีความจำเป็นที่จะต้องนำ Output มาป้อนกลับเข้าในระบบ ดังนั้นจะได้ System Diagram ดังในรูปที่ 2.4 ซึ่งแสดงเป็นแบบ Signal Flow Graph



Realization of the general difference Equation

รูปที่ 2.3



Realization of a nonrecursive system signal flow graph

รูปที่ 2.4

Unit Sample Response

Unit Sample Response เป็นอนุกรมของ O/P ที่ผลิตออกมา โดยอาศัยการป้อน $x(n) = \delta(n)$ ให้กับระบบ ถ้าเรารู้ Unit Sample Response ก็จะสามารถรู้ผลตอบสนองอันเนื่องมาจากการป้อน Input Sequence ใดๆได้ ซึ่งกรรมวิธีนี้มักจะใช้เป็นพื้นฐานใน Digital Signal Processing

พิจารณา Nonrecursive System ดังต่อไปนี้

$$y(n) = \sum_{k=0}^{k=L} b_k x(n-k) \quad \text{----------(2.13)}$$

และกำหนด Unit Sample Input ดังนี้

$$x(n) = \delta(n) = 1 \; ; \; n = 0$$
$$= 0 \; ; \; n \neq 0 \quad \text{----------(2.14)}$$

เมื่อทำการป้อน Input นี้เข้าไปยังระบบที่เราสนใจพิจารณา ผลตอบสนองที่ได้รับจากระบบจะเรียกว่า Unit Sample Response h(n) นั่นคือ $y(n)=h(n)$ ถ้า $x(n) = \delta(n)$ จากสมการข้างต้นจะพบว่า

$$y(0) = h(0) = b_0x(0) + b_1x(-1) + \ldots + b_Lx(-L)$$
$$= b_0$$
$$y(1) = h(1) = b_0x(1) + b_1x(0) + \ldots + b_Lx(1-L)$$
$$= b_1$$
$$\vdots$$
$$y(L) = h(L) = b_0x(L) + b_1x(L-1) + \ldots + b_Lx(0)$$
$$= b_L$$

จากผลดังกล่าวข้างต้นสรุปได้ว่า

$$y(n) = h(n) = b_n \; ; \; n = 0,1,2,\ldots,L$$
$$= 0 \; ; \; n = < 0 \quad \text{----------(2.15)}$$

ทำให้ได้ลักษณะสมการทั่วไปของ Unit Sample Response คือ

$$h(n) = \sum_{k=0}^{L} b_k\delta(n-k) = \sum_{k=0}^{L} h(k)\delta(n-k) \quad \text{----------(2.16)}$$

หรือแทนเป็นอนุกรมได้ดังนี้

$$\{ h(n)\} = \{ b_0, b_2, \ldots, b_n\} \quad \text{----------(2.17)}$$

ระบบที่เป็นไปตามสมการ 2.17 Unit Sample Response เรียกว่า FINITE IMPULSE RESPONSE (FIR) SYSTEM เนื่องจาก h(n) มีเทอมจำกัดแน่นอน คือตั้งแต่ 0 ถึง L ดังสมการ 2.16 เรารู้ว่าที่ $n >= L+1$ ค่าของ Unit Sample Re-

sponse จะถือเสมือนว่าเป็นศูนย์ ดังนั้นจึงเป็นไปได้เสมอที่จะแทน FIR System ด้วย Difference Equation ที่อยู่ในรูปแบบของสมการ 2.13

ถ้าผลตอบสนองของระบบที่มีต่อ Unit Sample Response ไม่มีค่าใดเข้าใกล้ศูนย์เลย ระบบดังกล่าวจะเรียดว่า INFINITE IMPULSE RESPONSE (IIR)

คุณสมบัติโดยทั่วไปของ Digital Filter แบบ FIR และ IIR

1. ลักษณะของวงจรเป็นแบบ Nonecursive และ เป็น Convolution โดยตรง ไม่มีการป้อนกลับ

2. ค่าผิดพลาดเนื่องจากการปัด และความคลาดเคลื่อนของสัมประสิทธิ์ของ FIR มีน้อยกว่าแบบ IIR

3. ทรานสเฟอร์ฟังค์ชั่นของ FIR แบบ Nonrecursive จะมี Pole อยู่ที่จุดกำเนิดและมีเสถียรภาพเสมอ

4. วงจรฟิลเตอร์แบบ IIR สามารถออกแบบให้มีคุณสมบัติของเฟสแบบเชิงเส้นได้ แต่ทำได้ยากใน FIR

5. ช่วงเวลาในการหน่วงของ FIR จะเพิ่มขึ้นตามจำนวนพจน์ และค่อนข้างจะมีค่ามากที่วงจรที่มีอันดับสูง

6. โดยปกติแล้วปัญหาการประมาณค่าสำหรับวงจรแบบ FIR ค่อนข้างจะมีปัญหามากกว่าวิธีการของ IIR จำเป็นต้องใช้ Computer มาช่วยในการหาค่าพารามิเตอร์ต่างๆในแบบ FIR

Discrete Fourier Transform

เป็นการแทนอนุกรมหนึ่งๆด้วยรูปแบบของผลรวมในเทอม Complex Exponentials ดังเช่นสมการต่อไปนี้

$$y(\alpha) = a_0/2 + (a_1\cos\alpha + b_1\sin\alpha) + (a_2\cos\alpha + b_2\sin\alpha) + \ldots \quad \text{----------(2.18)}$$

ใช้แทนสมการของ Wave form หนึ่งๆ เทอมต่างๆในสมการ 2.18 สามารถเปลี่ยนให้อยู่ในรูปแบบของ Exponential ได้ดังนี้

$$y(\alpha) = \sum_{n=-\infty}^{+\infty} y(n)e^{-jn\alpha} \quad \text{----------(2.19)}$$

โดยอาศัยความสัมพันธ์ของ Euler ในสมการ 2.18 จะพบว่า

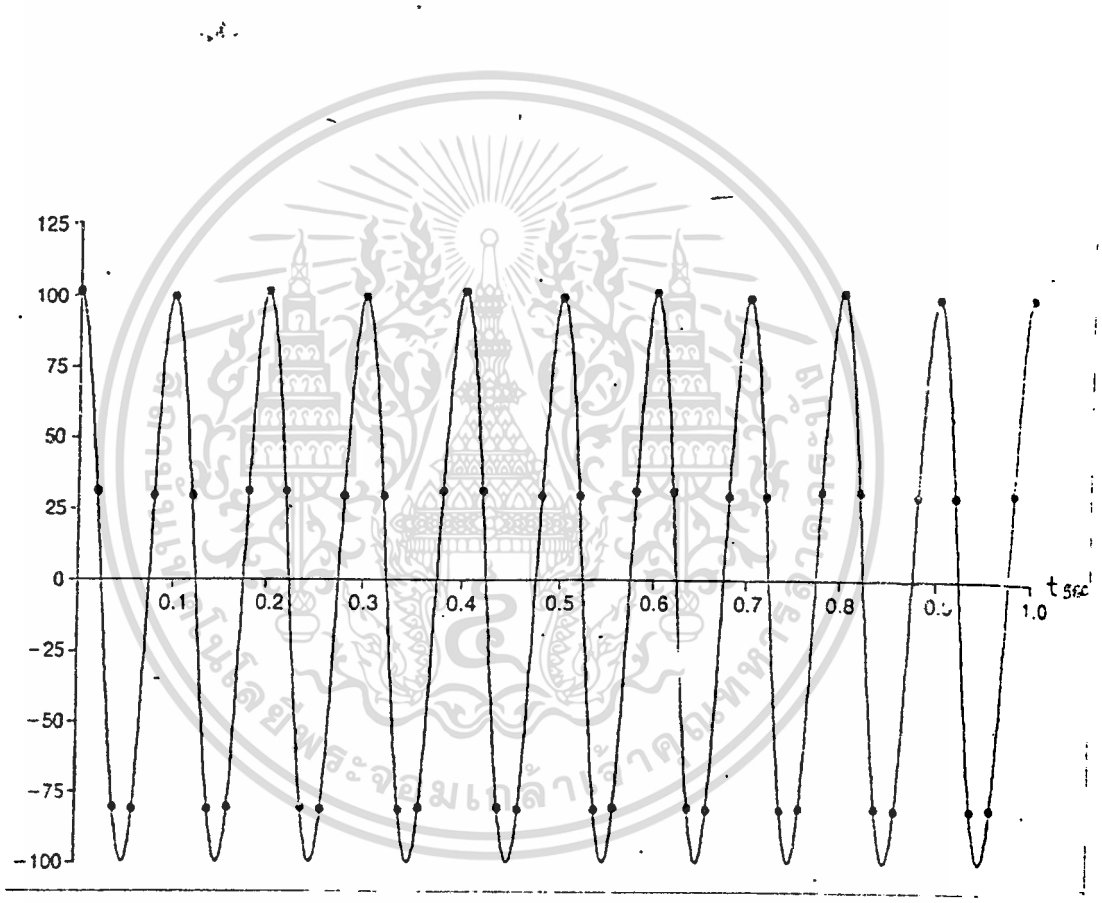
$$y(n) = 1/2(a_n \pm jb_n) \; ; \; n= \pm1,\pm2,\ldots$$

และ $y_o = a_o/2$ ซึ่งสมการ 2.19 นี้จะแทน Periodic Waveform ตลอดทั้ง (Infinite series) อนุกรม Complex Exponential แต่ในที่นี้จะเป็น

การใช้ในแบบ Finite Series

เหตุผลในการแทนอนุกรมที่สนใจด้วย Finite Sum ของ Complex Exponential (หรือ Sinusoids) เหตุผลหนึ่งที่สำคัญคือ หากเราพิจารณา Wave Form หนึ่งๆใน Time Domain ดังรูปที่ 2.4 จะเห็นได้ว่า มีลักษณะเป็น Sinusoids ที่เป็น Single Sinusoid จริงๆแล้วไม่ใช่ แต่หากเป็นผลรวมของ Sinnusoid ถึง 3 เทอม ดังสมการต่อไปนี้

$$x(t) = 1.0\cos(2\pi.5t) + 100\cos(2\pi.10t) + 0.5\cos(2\pi.20t)$$



Periodic signal and its sample value

รูปที่ 2.4

ดังนั้นในการพิจารณาจึงจำเป็นต้องอาศัยการพิจารณาใน Frequency Domain จึงจะสามารถแยกออกได้ว่า Wave form นี้ประกอบด้วยความถี่อะไรบ้าง และแต่ละความถี่นั้นมีขนาดของ Amplitude เท่าใด ซึ่งค่าเหล่านี้มีความสำคัญมากในการที่จะบอกให้เราทราบถึง คุณสมบัติของระบบที่เราตรวจวัด Wave form ออกมา เช่น ความผิดปกติของ Mechanism ในระบบกลไกต่างๆ , ความผิดปกติของการทำงานของหัวใจ ซึ่งกระบวนการทาง DFT สามารถนำมาใช้ได้อย่างมีประสิทธิ์ภาพ

บทที่ 3
ฮาร์ดแวร์ของระบบ

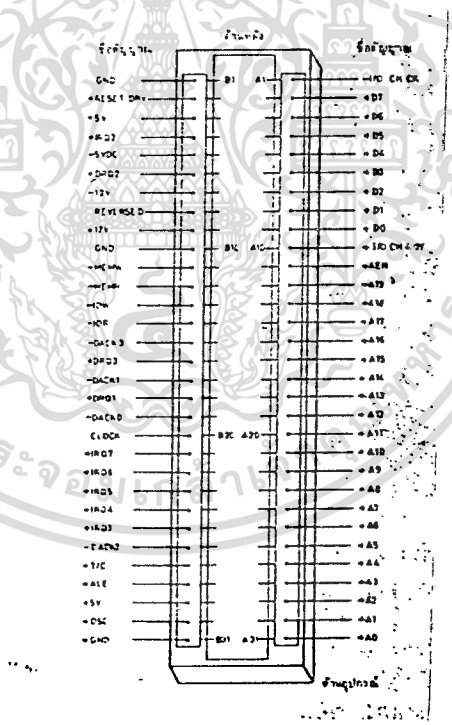
แบ่งได้เป็น 2 ส่วนคือ

1. การติดต่อ I/O PORT และการ DECODE PORT
2. ANALOG TO DIGITAL CONVERTER

ซึ่งกระบวนการของทั้งสองส่วนจะทำงานและถูกควบคุมโดย PROGRAME ที่เขียนโดยใช้ภาษา C แต่ละส่วนสามารถอธิบายได้ดังนี้

การติดต่อ I/O PORT และการ DECODE PORT

ในการ INTERFACE กับ MICRO COMPUTER โครงสร้างของ MAIN BOARD ที่ช่วยในการขยายขีดความสามารถในการ INTERFACE กับอุปกรณ์ภายนอกคือ สล็อต(SLOT)ซึ่งมีรายละเอียดของขาต่างๆ ดัง ในรูปที่ 3.1

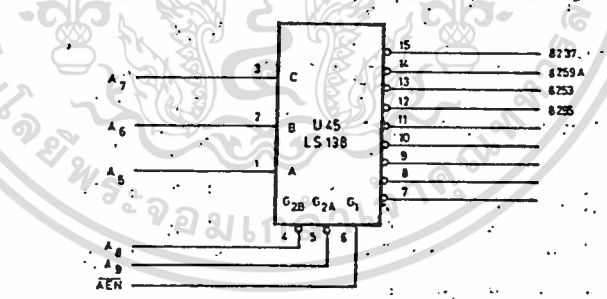


รูปที่ 3.1 แสดงสายสัญญาณเชื่อมต่อที่สล็อต

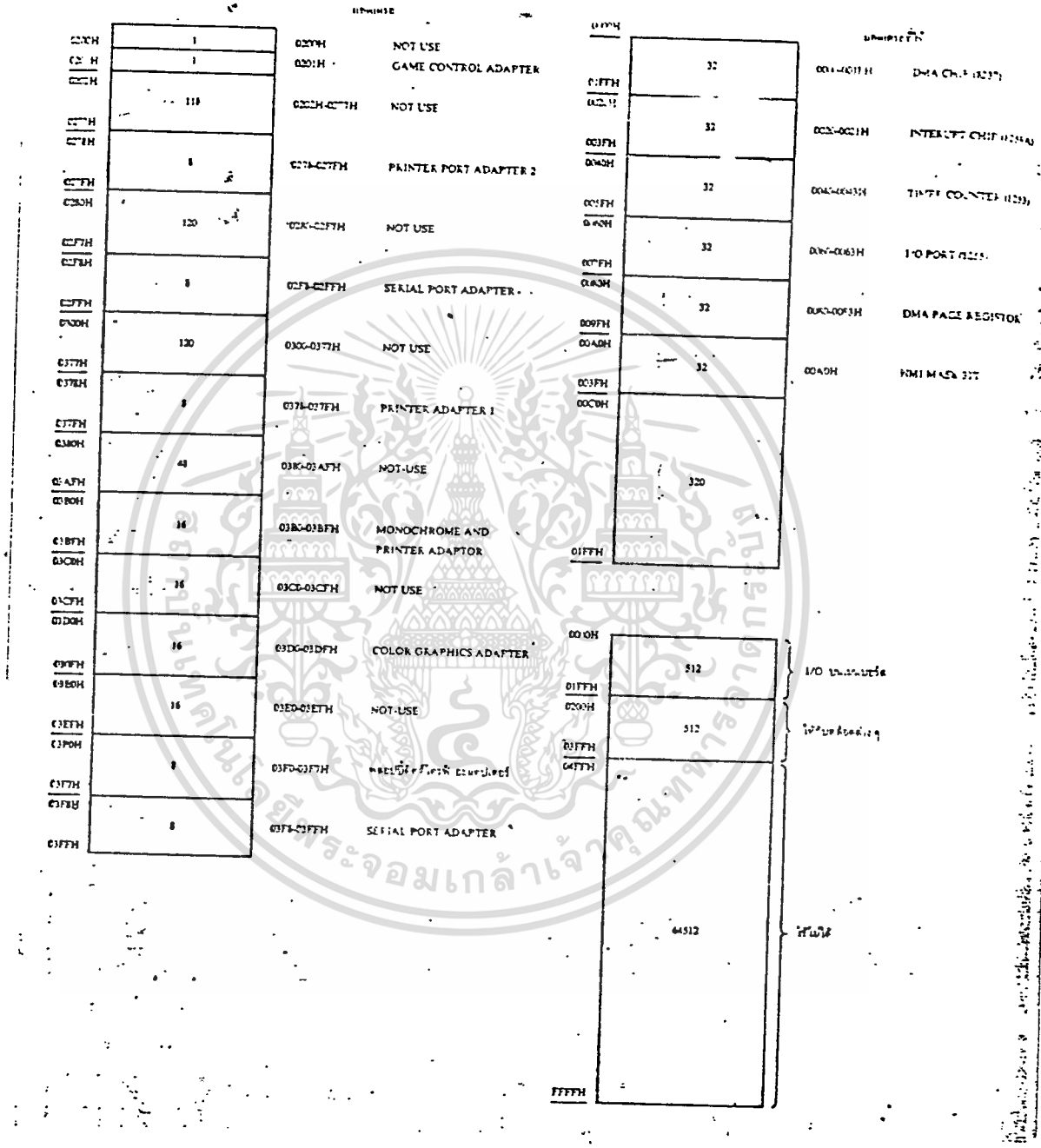
สำหรับใน IBM PC/XT นั้นจะมีสล็อตเชื่อมต่อกับอุปกรณ์ภายนอกได้มากขึ้นคือใน IBM PC/XT จะทำการเพิ่มสล็อตบน MAIN BOARD ขึ้นเป็น 8 สล็อตจากเดิมที่มีอยู่เพียง 5 สล็อต บน IBM PC โดยการจัดสัญญาณต่างๆ ใน 8 สล็อตจะยังคงเหมือนใน IBM PC เพียงแต่สัญญาณต่างๆที่ถูกส่งออกมายังขาของสล็อตที่ 8 นั้นจะถูกต่อผ่านวงจรขับกระแส(BUFFER)ก่อน และในสล็อตที่ 8 นี้ขา B8 จะถูกใช้งานด้วยโดยจะถูกใช้เป็นขา Card Select ซึ่งขาสัญญาณนี้จะเป็นสัญญาณอินพุทจากวงจรภายนอกที่เสียบอยู่บนสล็อตที่ 8 เพื่อให้วงจรบน MAIN BOARD ทราบว่า CARD ที่อยู่บนสล็อตนี้ถูกเลือกใช้งานอยู่ซึ่งจะทำให้ DRIVER บน MAIN BOARD ทำการอ่านข้อมูลหรือส่งข้อมูลไปยังสล็อตที่ 8

การจัด Address สำหรับ I/O PORT

ในการอ่านและเขียนพอร์ต CPU จะใช้แอดเดรสเพียง 16 เส้นจาก 20 เส้นสำหรับอ้างอิงแอดเดรสให้พอร์ต ซึ่งจะทำให้สามารถต่อพอร์ตออกมาได้มากถึง 65,536 พอร์ตหรือ 64K แต่ว่าเครื่อง IBM PC จะจัดกรอดระหัสแอดเดรสเพียง 10 เส้น ทำให้อ้างแอดเดรสได้เพียง 1024 พอร์ตจาก ในรูปที่ 3.2 ใช้ A9 เข้าที่ขา G2A ของ 74LS138 ซึ่งเป็นตัวกอดระหัสแอดเดรสให้กับ I/O PORT บน MAIN BOARD ทั้งหมด ดังนั้น I/O PORT จึงมีแอดเดรสพอร์ตอยู่ในช่วง 0000H-01FFH ซึ่งเท่ากับ 512 พอร์ต อันนี้เป็นส่วนของ MAIN BOARD ดังนั้นจึงเหลือพอร์ตสำหรับสล็อตอีกเพียง 512 พอร์ต คือในช่วงแอดเดรส 0200H-03FFH หรือเมื่อ A9 เป็น "1" ในแอดเดรสช่วงนี้ได้มี CARD สนับสนุนของ IBM ใช้ไปบ้างแล้วดังแผนผังในรูปที่ 3.3



รูป 3.2



รูปที่ 3.3

การ DECODE PORT

ในการใช้งานไมโครคอมพิวเตอร์ส่วนใหญ่ จะต้องเชื่อมโยงกับอุปกรณ์ภายนอก เช่น สวิตช์, รีเลย์ หรือตัวตรวจจับอื่นๆ การเชื่อมต่อในลักษณะดังกล่าวจะเชื่อมกับพอร์ตอินพุท-เอาท์พุทเพื่อให้ MICROPROCESSOR ส่งสัญญาณควบคุมไปยังอุปกรณ์ต่างๆ ตามเงื่อนไขที่เกิดขึ้นและตรวจสอบได้โดย MICROPROCESSOR เอง

1. การเชื่อมต่อพอร์ตอินพุท ในลักษณะที่ง่ายที่สุด คือการเชื่อมโดยใช้เกทลอจิก 3สถานะ โดยสัญญาณควบคุมพอร์ตอินพุทจะเป็นตัวไปเปิดเกทให้ข้อมูลเข้าสู่บัสและ MICROPROCESSOR จะอ่านเข้าไปเพื่อประมวลผล

2. สำหรับพอร์ตเอาท์พุท จะใช้แลตช์ฟลิปฟลอปทำหน้าที่รับสัญญาณข้อมูลจาก MICROPROCESSOR ที่ส่งเข้าไปในบัสและได้รับการ LATCH ไว้ที่พอร์ตในขณะที่มีสัญญาณควบคุมพอร์ตมาทริกที่ขา LATCH

แต่พอร์ตอินพุท-เอาท์พุท ที่ใช้เกทขนาดเล็กดังกล่าวยังมีจุดอ่อนในเรื่องของจำนวนไอซี ซึ่งอาจต้องใช้หลายชิพในกรณีที่ต้องการหลายพอร์ต และยากที่จะกำหนดลักษณะการทำงานให้แตกต่างไปจากวงจรเดิมที่ออกแบบไว้ บริษัทผู้ออกแบบ MICROPROCESSOR ส่วนใหญ่จึงออกแบบ LSI ชิพเพื่อทำหน้าที่เป็น พอร์ตอินพุท-เอาท์พุทของระบบซึ่งมีข้อดีในเรื่องของการใช้งาน ซึ่งสามารถทำได้ง่ายและสะดวกในการเปลี่ยนแปลงรูปแบบ

ไอซีที่ทำหน้าที่เป็นพอร์ตอินพุท-เอาท์พุท ที่นำมาใช้ใน PROJECT นี้คือ 8255 เป็นไอซี LSI ของบริษัท INTEL เป็นไอซีในตระกูลของ 8080 เพราะอินเทลได้ออกแบบมาให้ใช้งานร่วมกับ 8080 แต่อย่างไรก็ตามเราสามารถที่จะนำมาประยุกต์ใช้งานกับ MICROCOMPUTER ได้

8255 เป็นไอซีที่ต่อเป็นพอร์ตให้ MICROPROCESSOR ได้ 3 พอร์ต มีพอร์ต A,B,C โดยพอร์ต C แยกเป็นสองส่วนคือ PC_0-PC_3 เรียกว่าพอร์ต C ล่างจำนวน 4 บิทและ PC_4-PC_7 เรียกว่าพอร์ต C บน ทุกพอร์ตสามารถเป็นได้ทั้งพอร์ตอินพุทและพอร์ตเอาท์พุท โดยการส่งระหัสควบคุมของ MICROPROCESSOR มากำหนดรูปแบบพอร์ต

จากการที่ 8255 มีพอร์ตที่ CPU มองเห็นได้ 4 พอร์ตและแต่ละพอร์ตจะเห็นเสมือนเป็น REGISTER ที่เขียน/อ่านได้ REGISTER แต่ละตัวนี้จึงถูกกำหนดด้วยแอดเดรสตามที่ตั้งไว้ ดังนั้นสัญญาณของขาควบคุมที่ประกอบกัน จะแสดงความหมายดังตารางที่ 3.1 โดย A1,A0 เป็นแอดเดรสบัส

การใช้งาน 8255 เพื่อให้แต่ละเป็นพอร์ตอินพุทหรือเอาท์พุท ต้องให้ CONTROL WORD ก่อนเริ่มทำงาน ซึ่งสามารถกำหนดได้ 3 โหมด คือ

1. MODE 0 เป็นอินพุท - เอาท์พุทแบบพื้นฐาน

2. MODE 1 เป็นอินพุท - เอาท์พุทที่มีการตรวจสอบสัญญาณ(Hand-shaking) โดยใช้ อินพุท - เอาท์พุท ของพอร์ต A,พอร์ต B เป็นหลัก และใช้พอร์ต C บนเป็นสัญญาณ Handshake ของพอร์ต A และใช้พอร์ต C ล่างเป็นสัญญาณ Handshake ของพอร์ต B

3. MODE 2 ทำได้เฉพาะพอร์ต A คือทำหน้าที่เป็นพอร์ตสองทิศทาง สามารถเป็นได้ทั้งอินพุทและเอาท์พุท

RD	WR	A1	A0	ความหมาย
1	0	0	0	เขียนพอร์ต A ซึ่งเป็นข้อมูล
0	1	0	0	อ่านพอร์ต A ซึ่งเป็นข้อมูล
1	0	0	1	เขียนพอร์ต B ซึ่งเป็นข้อมูล
0	1	0	1	อ่านพอร์ต B ซึ่งเป็นข้อมูล
1	0	1	0	เขียนพอร์ต C ซึ่งเป็นข้อมูล
0	1	1	0	อ่านพอร์ต C ซึ่งเป็นข้อมูล
1	0	1	1	เขียนข้อมูล ซึ่งเป็นระหัสควบคุม
0	1	1	1	อ่านเข้ามาซึ่งไม่มีความหมายใด

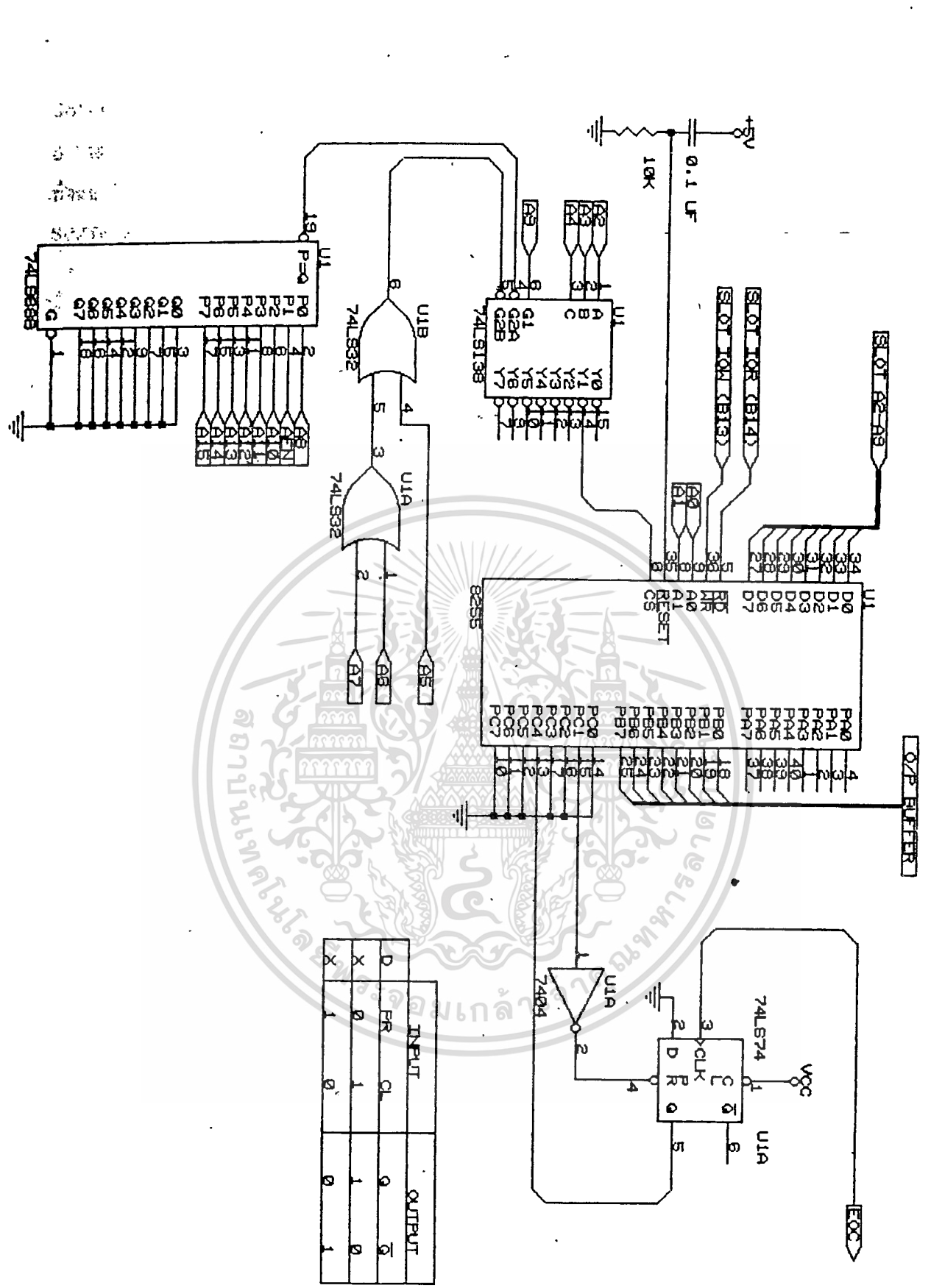
ตารางที่ 3.1 สัญญาณควบคุมการทำงานของ 8255

ใน PROJECT นี้อาศัยการทำงานใน MODE 0 ซึ่งระหัสควบคุมที่ใช้แสดงได้ดังต่อไปนี้

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	1	0

เป็นเลขฐานสิบหกคือ 8AH มีความหมายดังนี้

1. พอร์ต B ทำหน้าที่เป็นอินพุท



รูปที่ 3.4

2. พอร์ต C บนทำหน้าที่เป็นอินพุท

3. พอร์ต C ล่างทำหน้าที่เป็นเอาท์พุท

แอดเดรสที่ใช้สำหรับระหัสควบคุม คือ 0207H

แอดเดรสที่ใช้สำหรับถอดระหัสพอร์ต B คือ 0205H

แอดเดรสที่ใช้สำหรับถอดระหัสพอร์ต C คือ 0206H

<u>การทำงานของวงจรในรูปที่ 3.4 สามารถอธิบายได้ดังนี้</u>

ไอซี 74LS668 จะให้เอาท์พุทที่ขา 19 เป็น LOW ก็ต่อเมื่อ P0 - P7 มีสภาพเป็น LOW ทั้งหมด นั่นคือมีสภาพตรงกับ Q0 - Q7 ซึ่งจะทำการถอดระหัสร่วมกับ 74LS138 โดยใช้ A0,A1 ต่อกับ 8255 เพื่อเลือกพอร์ตที่ต้องการ และที่ขา RESET ของ 8255 ต่อ LOW ไว้ตลอดนั่นคือไม่มีการเซท 8255 ให้กลับไปอยู่ในสถานะเริ่มต้นหรือทุกพอร์ตจะเป็นอินพุทหมด ในการทำงานของพอร์ต C บน จะทำหน้าที่ตรวจสอบสัญญาณจาก D-Flip Flop โดยอาศัย Software อ่านเป็น-Loop ไปเรื่อยๆจนกว่า Data ที่พอร์ตนี้จะเท่ากับ 00 จึงจะหยุดอ่านและกลับไปอ่านพอร์ต B นั่นคือใช้พอร์ต C บนเป็นตัวตรวจสอบว่าข้อมูลมาพร้อมที่จะอ่านหรือยังและใช้พอร์ต C ล่าง ทำหน้าที่ Preset D-FF ให้พร้อมที่จะตรวจสอบการรับข้อมูลตัวต่อไป ซึ่งจะอธิบายโดยละเอียดในหัวข้อ SOFTWARE ต่อไป

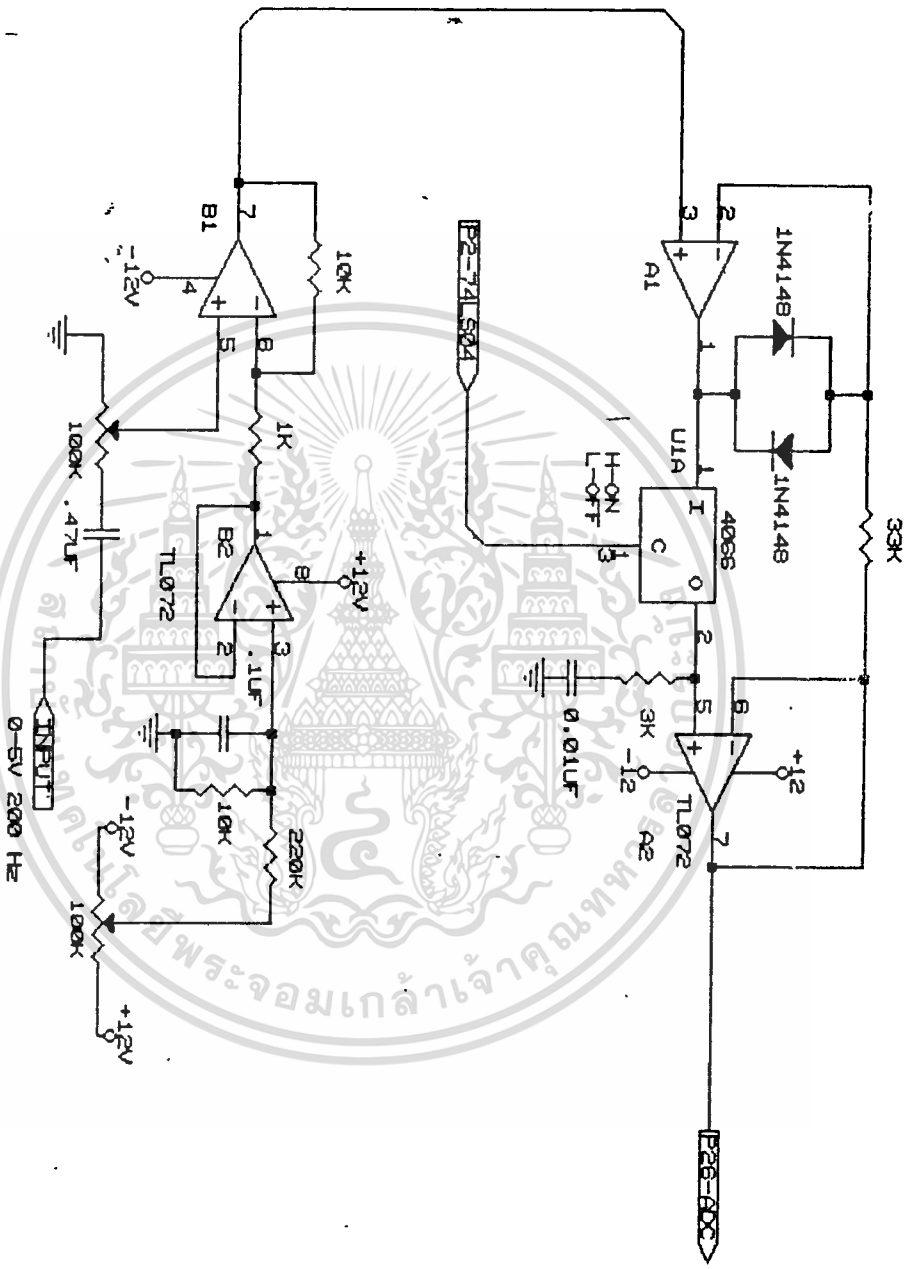
<u>ANALOG TO DIGITAL CONVERTER</u>

ในส่วนนี้จะอธิบายแยกเป็น 2 ส่วน

1. SAMPLE / HOLD และวงจรยกระดับแรงดัน
2. A/D และ MONOSTABLE MULTIVIBRATOR

<u>SAMPLE /HOLD</u>

ในรูปที่2.5 ในวงจรนี้ OPamp A1 ทำหน้าที่เป็น BUFFER และ ERROR AMPLIFIER ในตัว ซึ่งจะทำหน้าที่เปรียบเทียบแรงดันเอาท์พุทจากนั้นก็จะ Charge ตัวเก็บประจุจน ERROR เท่ากับศูนย์ ในการทำงานจริงของวงจรค่าของตัวเก็บประจุ(C)และค่าความต้าน(R) จะสามารถหาได้จากการทดลอง เพื่อให้ได้เอาท์พุทที่เหมาะสม ในวงจรนี้ใช้ C = 0.01 F , R = 3K ส่วนในเรื่องของการ SAMPLE จะอธิบายในส่วนของ MONOSTABLE MULTIVIBRATOR ในวงจรนี้มีอินพุทอิมพีแดนซ์สูงนอกจากนี้การป้อนกลับใน OPamp A1 ด้วยไดโอด ทำให้ A1 ไม่จำเป็นต้องเป็น OPamp ที่มีคุณภาพดีนัก ความต้านทาน R ค่า 33K จ ะทำหน้าที่แยกอินพุท A1 และ เอาท์พุท A2 ออกจากกันในช่วง Hold



รูปที่ 3.5

Time

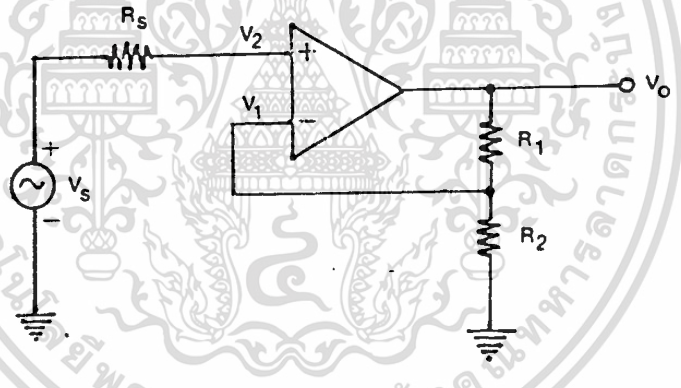
ข้อดีของวงจรนี้คือ ทำงานได้รวดเร็วแม่นยำและความเร็วในการประจุขึ้นกับความเร็วของ OPamp A1 และความสามารถในการจ่ายกระแสของมัน ไดโอด 2 ตัวทำหน้าที่ Clamp สัญญาณเอาท์พุทไปที่อินพุท Inverting ของ A1 เพื่อจะยังทำให้วงจรมีเสถียรภาพดีเมื่อ Switch Sample เปิดวงจร ซึ่งในการใช้งานจริงเราสามารถใช้ IC LF398 แทนได้ เพราะมีลักษณะการทำงานพื้นฐานอันเดียวกัน

ในวงจรรูปที่ 3.5 เมื่อพิจารณาการทำงานของส่วนยกระดับแรงดันสามารถอธิบายได้ดังนี้เริ่มที่ Opamp B2 ซึ่งต่ออยู่ในลักษณะ Voltage Follower การพิจารณาสามารถทำได้จากรูปแบบพื้นฐาน ของวงจร Opamp ที่ต่อแบบขยายไม่กลับชั้วดังรูที่ 2.6

เนื่องจาก $i_2 = 0$ เรารู้ว่า $V_2 = V_s$

และ $i_1 = 0$ จะได้

$$V_1 = R_2 V_o / (R_1 + R_2) \quad \text{----------(1)}$$

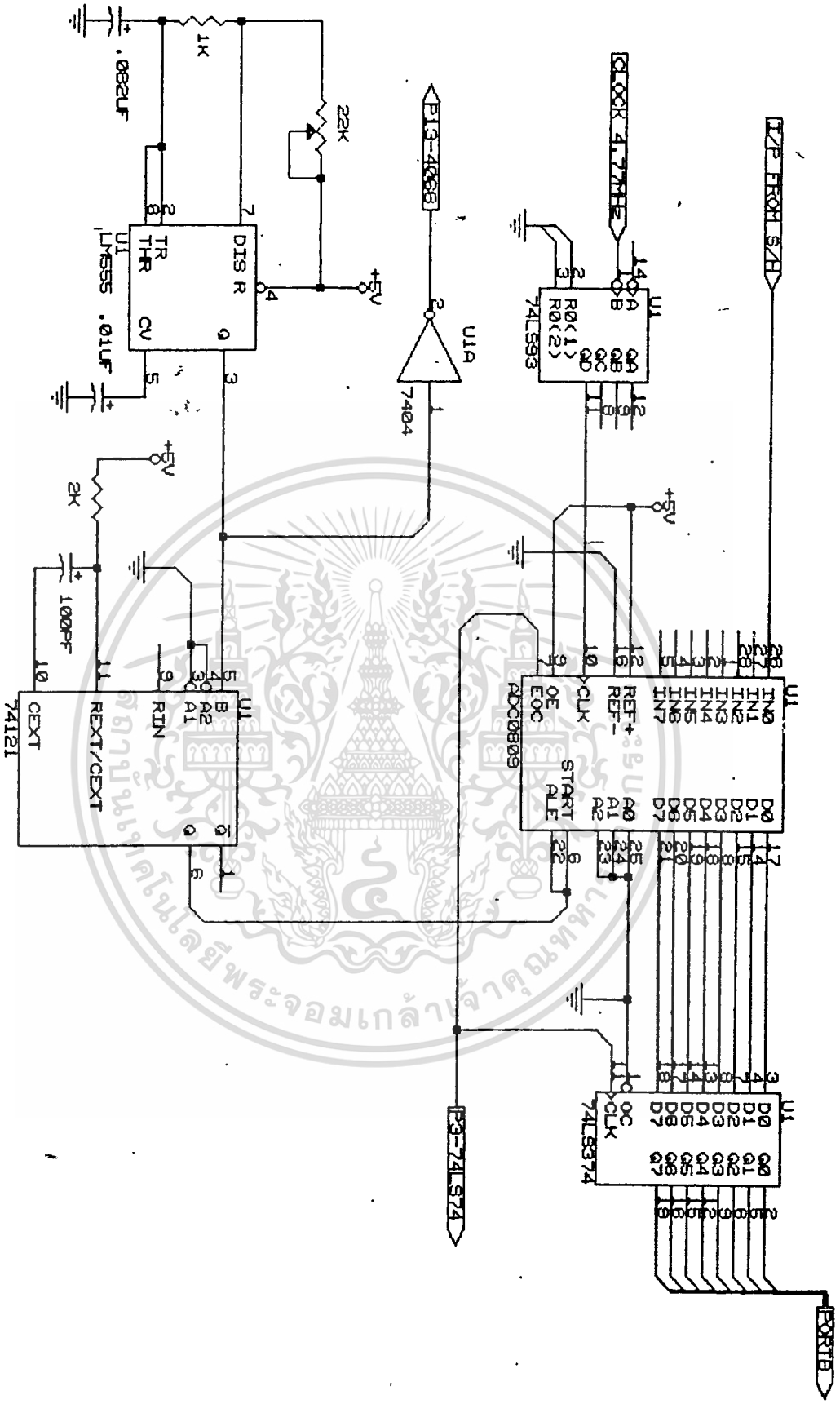


รูปที่ 3.6 แสดงการต่อ Opamp แบบขยายไม่กลับชั้ว

โดยที่ A (Gain) มีค่าใหญ่มากๆ ทำให้ $V_1 = V_2 = V_s$ ดังนั้นจากสมการที่ (1) จะได้

$$V_o / V_s = (R_1 + R_2) / R_2 \quad \text{----------(2)}$$

ถ้าเราให้ R_2 มีค่าสูงขึ้นเรื่อยๆ อัตราขยายจะลู่เข้าหาค่า 1 ถ้าหากเราให้ R_2 = INFINITY , $R_1 = 0$



รูปที่ 3.7

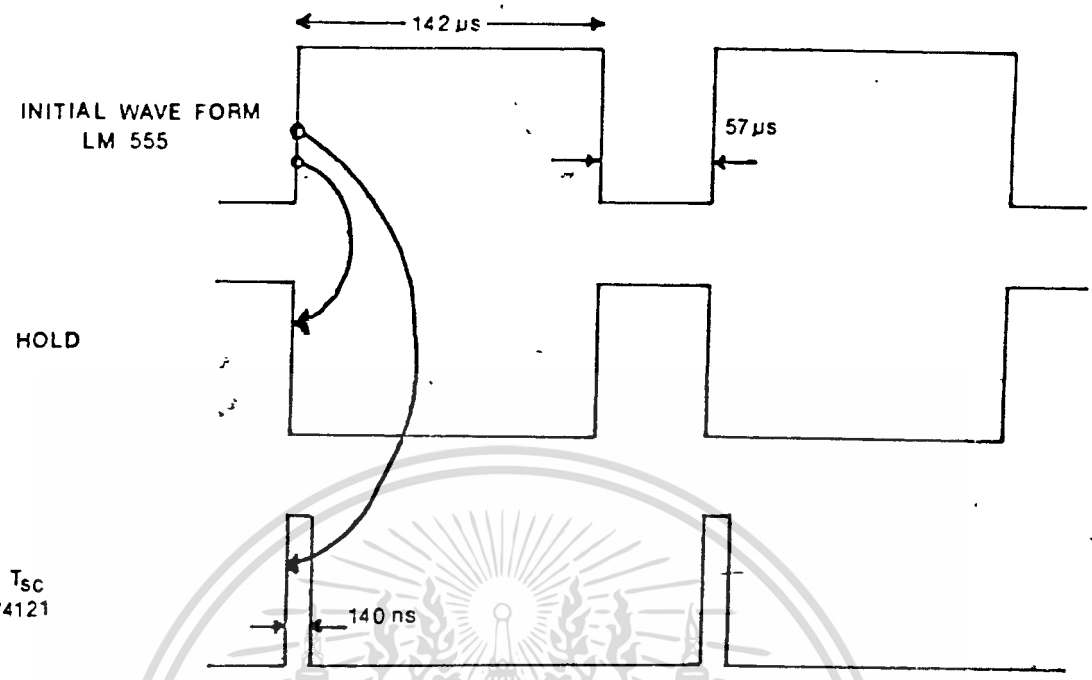
ก็จะได้วงจร Voltage Follower นั่นคือเราจะได้แรงดันเอาท์พุทตามแรงดันอินพุทอยู่ตลอดเวลาถึงแม้ว่าขา Non inverting ของ Opamp จะไม่มีกระแสไหลเข้าก็ตาม ขาเอาท์พุทของ Opamp ก็สามารถจ่ายกระแสให้ Load ได้

สำหรับวงจรในส่วน Opamp B1 เป็นวงจรรวมแรงดันโดยจะนำแรงดัน DC ที่สามารถเลือกปรับค่าได้ ไปรวมกับอินพุททำให้เอาท์พุทมีระดับ Voltage สูงกว่าศูนย์ Volt ตลอดทั้ง Wave Form ทั้งนี้เพราะ ADC0809 จะทำการแปลงสัญญาณอนาลอกในช่วง Voltage 0-5 Volt ตามค่า $+V_{ref}$ ซึ่งตั้งไว้ที่ +5 Volt และ $-V_{ref}$ ที่มีค่าเท่ากับ 0 Volt นอกจากนี้ที่ขา Non inverting ของ B_1 จะต่อกับ R ปรับค่าได้ 100K สัญญาณอินพุทที่เข้ามาจะถูกลดทอนให้มีขนาดที่เหมาะสม ซึ่งสามารถปรับค่าได้จากการทดลอง จะทำให้เราได้เอาท์พุทตามที่เราต้องการและมีการ Convert จาก Analog เป็น Digital ที่ถูกต้อง

A/D และ MONOSTABLE MULTIVIBRATOR

ในวงจรรูปที่ 3.7 ใช้ 74LS121 เพื่อสร้าง Pulse ที่มีความกว้างประมาณ 140 nS ใช้สำหรับการ Start Convert เพราะเนื่องจากความกว้างของ Pulse ที่ถูกกำหนดไว้ในการทำงานของ ADC0809 ความกว้างของ Start Pulse จะมีค่าอยู่ในช่วง 100 nS ถึง 200 nS ซึ่งเราไม่สามารถใช้ Timer IC LM555 สร้างขึ้นมาได้ ดังนั้นในการทำงานจึงต้องใช้ LM555 ร่วมกับ 74LS121 โดย LM555 จะต่อแบบ Astable Multivibrator และปรับค่าความถี่ให้เท่ากับ 5 KHz ทำได้โดยปรับ R ปรับค่า 50K

เมื่อพิจารณาการทำงานรวมทั้งหมด สามารถเขียน TIMING DIAGRAME แสดงให้เห็นขั้นตอนการทำงานได้ดังต่อไปนี้



รูปแสดง TIMING DIAGRAME

จาก TIMING DIAGRAME สามารถอธิบายได้ดังนี้

ในช่วงเวลา t_1 สัญญาณจาก Timer IC LM555 เมื่อมีการเปลี่ยนแปลงของขอบระดับแรงดันจาก LOW เป็น HIGH จะทำให้เกิดเหตุการณ์ 2 อย่างคือ

1. เกิดการ Hold ขึ้นจากการกลับสัญญาณของ 74LS04 ทำให้อินพุท Voltage อยู่ในสภาวะ Stable ชั่วระยะเวลาหนึ่ง นั่นคือ Switch Sample Open นั่นเอง

2. พร้อมๆ กันนั้นก็จะเกิดการ Start Convert โดย 74LS121 จะผลิต Pulse ที่มีความกว้าง 140 nS ไปบอก ADC0809 ให้เริ่มการ Convert ซึ่งจะเริ่มเมื่อขอบของระดับสัญญาณ Start จาก 74LS121 เปลี่ยนจาก HIGH เป็น LOW การ Convert จะใช้เวลาประมาณ 90 - 116 S

ในการออกแบบวงจร ต้องคำนวณเวลาเผื่อสำหรับการ Convert และ เวลาที่ CPU ใช้ในการอ่านเมื่อเสร็จสิ้นการ Convert ดังนั้นช่วงเวลาของ Timer IC LM555 ไม่ควรน้อยกว่า 200 S นั่นคือการ Start Convert จะเกิดขึ้นทุกๆ ช่วง

เวลาประมาณ 200 S

ต่อมาในช่วง t_2 ของ Timer IC LM555 ขอบของระดับสัญญาณเปลี่ยนจาก HIGH ลงมาเป็น LOW ตอนนี้ Switch Sample จะ ON ทำให้ระดับแรงดัน Flow ตามสัญญาณอินพุท ในช่วงนี้การ Start Convert จะยังไม่เกิดจนกว่าจะถึงช่วง t_1 ของ Cycle ถัดไป สิ่งที่สำคัญในการพิจารณา การทำงานคือ Start Pulse จะผลิตออกมาในช่วงที่กำลัง Convert ไม่ได้เ พราะจะทำให้ระบบการทำงานผิดพลาดทั้งหมด เนื่องจากในระหว่างที่กำลัง Convert อยู่นั้นถ้ามี Start Pulse เข้ามาจะทำให้กระบวนการ Convert หยุดลงและจะเริ่ม Convert ใหม่ตาม Pulse ที่ได้รับเข้ามา

ในการใช้งาน 74LS121 จะต่ออยู่ในลักษณะที่จะผลิต Pulse ได้ก็ต่อเมื่อขา 5ได้รับสัญญาณขอบขาขึ้นและขา 3,4 เป็น LOW สำหรับความกว้างของ Pulse สามารถคำนวนได้จาก

$$T = 0.7\, R_{ext} C_{ext} \qquad \text{----------(3)}$$

ภายใต้เงื่อนไขดังต่อไปนี้

$$1.4\ K < R_{ext} < 40\ K$$
$$0 < C_{ext} < 1000\ F$$

ในวงจรใช้ $R_{ext} = 2\ K$, $C_{ext} = 100\ pF$

ดังนั้นจะได้ $T = 140\ nS$

สำหรับ Timer IC LM555 เมื่อต่อแบบ Astable Multivibrator สามารถคำนวณหาช่วงเวลาต่างๆ ได้ดังนี้

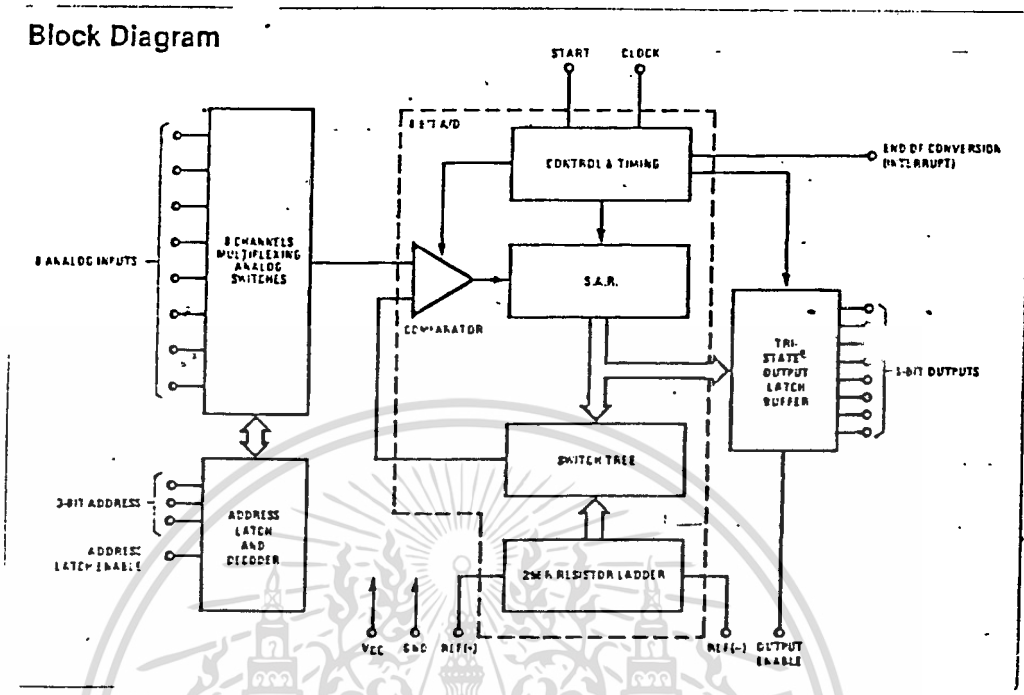
$$t_1 = 0.693(R_a + R_b)C \qquad \text{----------(4)}$$
$$t_2 = 0.693\, R_b C \qquad \text{----------(5)}$$
$$T = t_1 + t_2$$
$$= 0.693(R_a + 2R_b)C \qquad \text{----------(6)}$$

ส่วนประกอบที่สำคัญอีกประการหนึ่งในการ Convert คือ Clock ที่ใช้ในการทำงานของ ADC0809 ซึ่งจะมีค่าการใช้งานอยู่ในช่วงประมาณ 10K - 1280K Hz ดังนั้นใน Project เลือกใช้ค่าเท่ากับ 596 KHz โดยใช้สัญญาณนาฬิกาจาก Main

board ซึ่งมีค่า = 4.77 MHz นำมาหาร 8 โดยอาศัย IC 74LS93 เป็น 4 Bit Binary Counter ทำหน้าที่หารความถี่ให้ได้ตามต้องการ

การทำงานของ ADC0809 จะมี 8 Channels Multiplexing Analog Switches สามารถที่จะต่ออินพุทได้ 8 อินพุท แต่ในการทำงานจะถูกเลือกทีละอินพุท เราสามารถออกแบบวงจรให้ทำหน้าที่ Scan Input ไปตามลำดับได้ แต่ใน Project นี้จะใช้เพียง Channel เดียวเท่านั้นดังนั้นขา A0,A1,A2 จึงต่อลงกราวด์ทั้งหมดส่วน OE (Output Enable) ต่อ HIGH เพื่อ Enable ไว้ตลอดเวลา มี 74LS374 เป็น Output Latch ซึ่งถูกควบคุมโดยสัญญาณ EOC (End Of Convert) จาก ADC-0809 โดยปกติแล้ว EOC เป็น HIGH อยู่ตลอดเวลาและจะเปลี่ยนแปลงจาก HIGH เป็น LOW ภายหลังจากเริ่มมีการ Convert โดยจะ Delay ไปประมาณ 1-2 s และจะรักษาสถานะ LOW ไว้ตลอดจนกว่าการ Convert เสร็จสิ้น แล้วจะกลับขึ้นมาเป็น HIGH อีก สัญญาณนี้เป็นตัวไปบอก CPU ว่า DATA พร้อมแล้วสามารถมาอ่านไปได้ โดยจะเป็น Clock ไปเซท D-FF ทำให้ Q มีค่าเท่ากับ 0 ตามค่า D ที่ต่อลงกราวด์ นั่นคือทำให้ค่าที่พอร์ต C บนเป็น 0 เมื่อ Software Check เจอก็จะไปอ่านพอร์ต B ซึ่งเป็น Data นั่นเอง

ADC0809 เป็นIC A/D ที่ทำงานแบบ SAR (Successive Approximation Register) โดยจะทำการ Set หรือ Reset Bit ต่างๆทีละ Bit ทั้ง 8 Bit ตามการเปรียบเทียบระดับแรงดันอินพุทกับแรงดันที่มาจากการ Convert Digital ให้เป็น Analog โดยอาศัย Comparator เป็นตัวเปรียบเทียบเพื่อตรวจว่ามากกว่าหรือน้อยกว่าแล้วทำการแปลง Bit ต่างๆ ต่อไป ทำเช่นนี้จนกว่าจะเสร็จการแปลง โดยบล็อกไดอะแกรมแสดงลักษณะภายใน ADC0809 แสดงไว้ใน รูป 3.8



รูป 3.8 แสดง Block Diagram ADC0809

บทที่ 4

SOFTWARE

ในการทำงานของ Hardware จะสามารถใช้งานได้ก็ต่อเมื่อมีการเรียกใช้ Software ซึ่งใน Project นี้เขียนโดยใช้ภาษา C การทำงานจะแบ่งเป็น 2 ส่วนใหญ่ๆ คือ ส่วนแรกเป็นการแสดง Wave form จากข้อมูลที่อ่านเข้ามาจาก Port หรือจาก File ซึ่งจะแสดงเป็นกราฟนิ่ง และส่วนที่ 2 จะเป็นการ Display Wave แบบต่อเนื่อง คือถ้ามีการเปลี่ยนแปลงไม่ว่าจะเป็น Amplitude หรือ ความถี่ ในขณะใดก็จะสามารถสังเกตเห็นได้ในขณะนั้นบนจอภาพ

ซึ่งรายละเอียดของการทำงานจะอธิบายได้ใน Flow Chart ซึ่งแบ่งเป็น 6 รูป โดยแต่ละรูปจะอธิบายการทำงานของแต่ละส่วนดังต่อไปนี้

รูปที่ 4.1 อธิบายการทำงานในส่วน Main Menu

รูปที่ 4.2 อธิบายตัวเลือกแรกใน Main Menu คือการอ่านข้อมูลจาก Port มาเก็บไว้ในหน่วยความจำ

รูปที่ 4.3 อธิบายกระบวนการเก็บข้อมูลที่อ่านเข้ามาจาก Port มาเก็บไว้ใน File ที่ต้องการ

รูปที่ 4.4 อธิบายกระบวนการการอ่านข้อมูลจาก Fileไว้ในหน่วยความจำเพื่อเตรียม Display

รูปที่ 4.5 อธิบายกระบวนการนำข้อมูลมาประมวลผลและแสดง Wave Form ให้เห็นทางหน้าจอภาพ โดยจะแบ่งเป็นการเลือกข้อมูลจาก Port หรือ ข้อมูลจาก File ที่อ่านเข้ามา

รูปที่ 4.6 อธิบายการ Display ต่อเนื่องให้เห็นลักษณะการเปลี่ยนแปลงของสัญญาณอินพุทตลอดเวลา

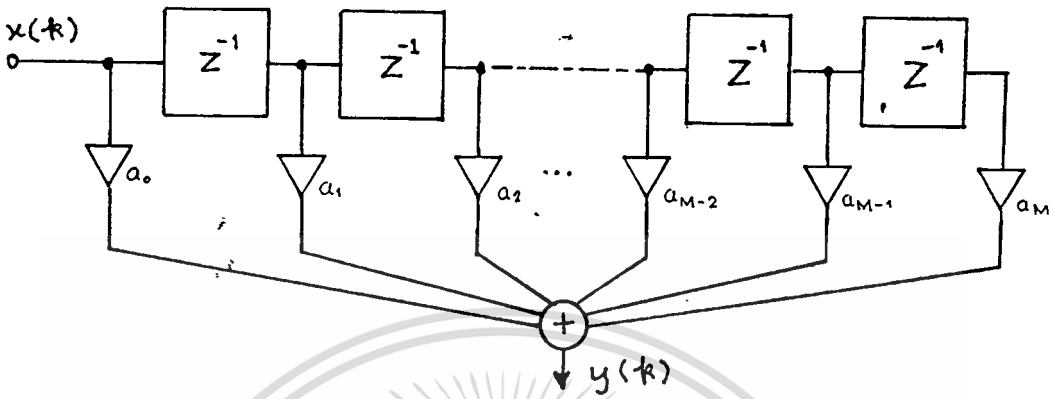
โปรแกรมส่วน Interface กับ Hardware จะแสดงเฉพาะ Flow chart เท่านั้นตัว Main Program จริงๆจะอยู่ในรายงานโปรเจคเทอมแรก

การประมวลผลทาง DIGITAL

โปรแกรมที่ใช้ในการศึกษาจะแบ่งเป็น 4 ส่วน

1) โปรแกรมแสดงการทำงานของระบบในแบบ FIR.DF หรือ Finite Impulse Response Digital Filter ซึ่งเป็นแบบ Nonrecursive สามารถเขียน

อธิบาย Process ของระบบได้โดยอาศัย System Diagram ดังนี้

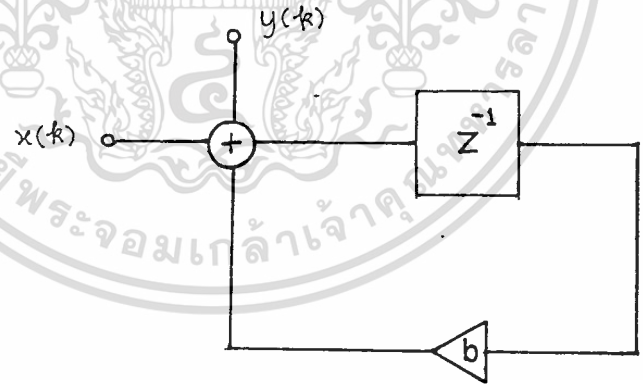


ซึ่งเป็นระบบที่ใช้ในโปรแกรม PRO21.C , PRO22.C

สมการของระบบคือ $y(k) = a_m x(k-m)$

การทำงานของโปรแกรมสามารถอธิบายได้ใน Flow Chart

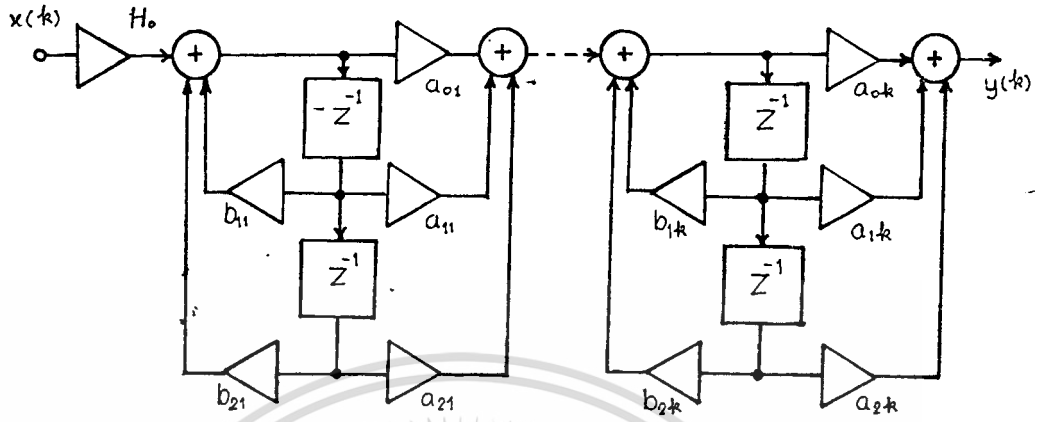
2) โปรแกรมแสดงการทำงานของระบบ IIR.DF หรือ Infinite Impulse Response Digital Filter ซึ่งระบบนี้เป็นแบบ Recursive สามารถเขียน System Diagram ได้ดังนี้



สมการของระบบคือ $y(k) = b*y(k-1) + x(k)$

การทำงานของโปรแกรม PRO23.C สามารถอธิบายได้ตาม Flow Chart ส่วนโปรแกรมต่อมาคือ PRO24C.C ก็เป็นการทำงานในแบบ IIR และเป็นการ

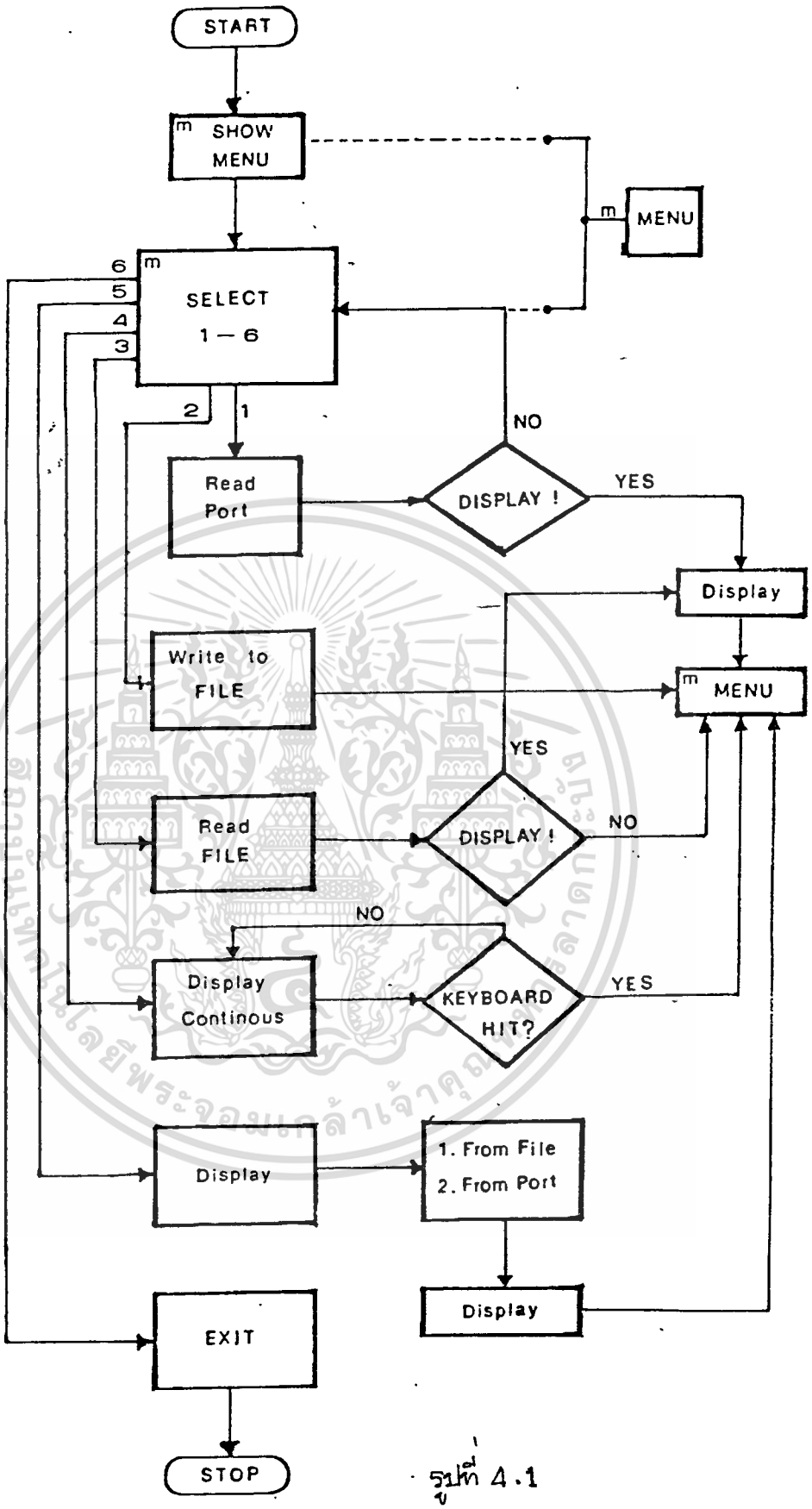
นำระบบ 2 ระบบ มาต่อ Casecade กันดัง System Diagram ดังต่อไปนี้



การทำงานของโปรแกรมสามารถอธิบายได้ใน Flow Chart

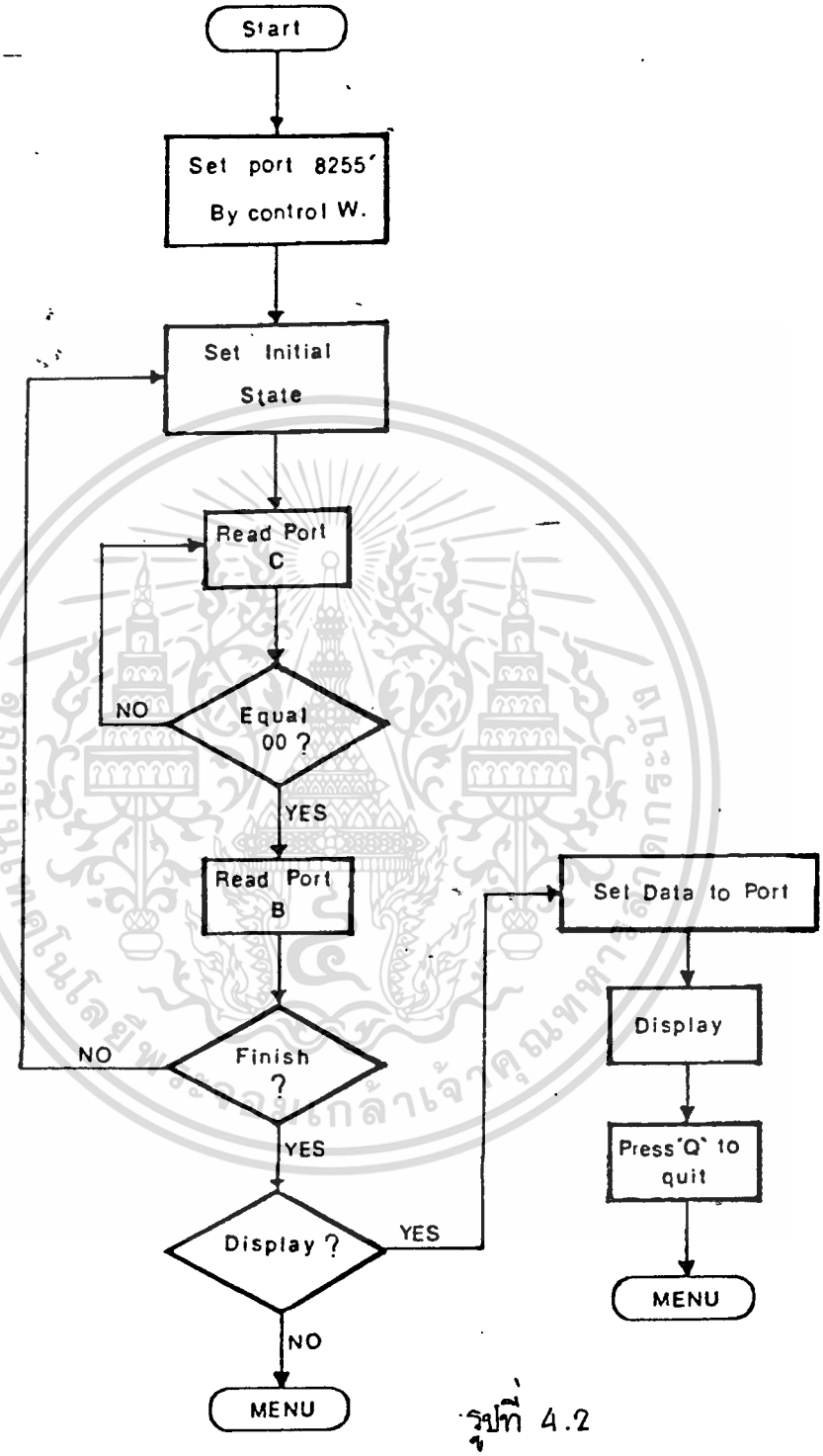
3) ในส่วนนี้จะเป็นการศึกษาถึงการนำ DFT มาใช้ในการทดสอบ Spectum ของ Wave Form หนึ่งๆเพื่อพิจารณาดูว่ามี Harmonics ใดประกอบอยู่บ้าง ซึ่งถ้าหากพิจารณาใน Time Domain ก็จะไม่สามารถเห็นองค์ประกอบย่อยๆเหล่านี้ ดังนั้นจึงต้องพิจารณาใน Frequency Domain ซึ่ง DFT สามารถช่วยได้

MAIN PROGRAME



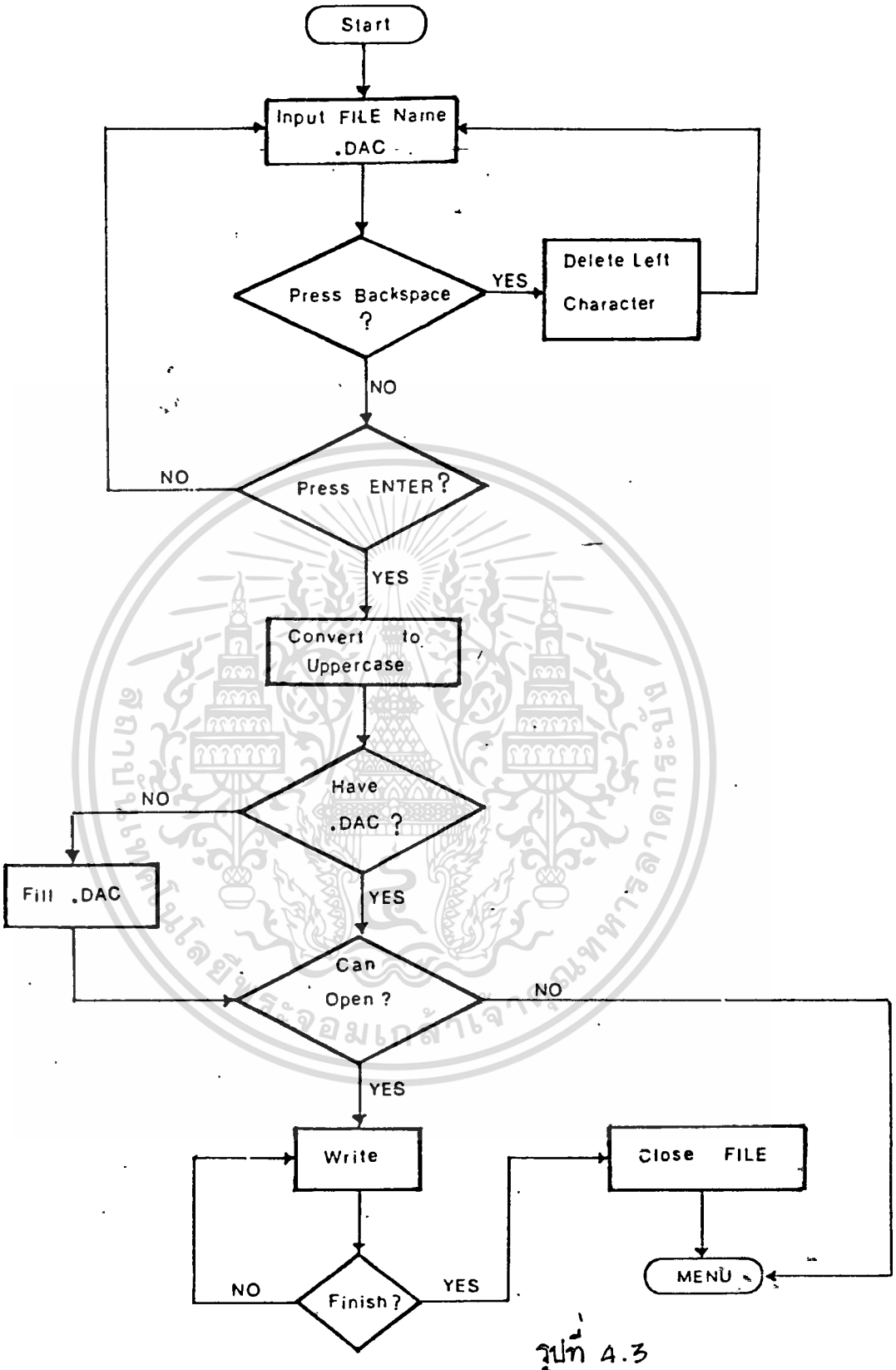
รูปที่ 4.1

READ PORT

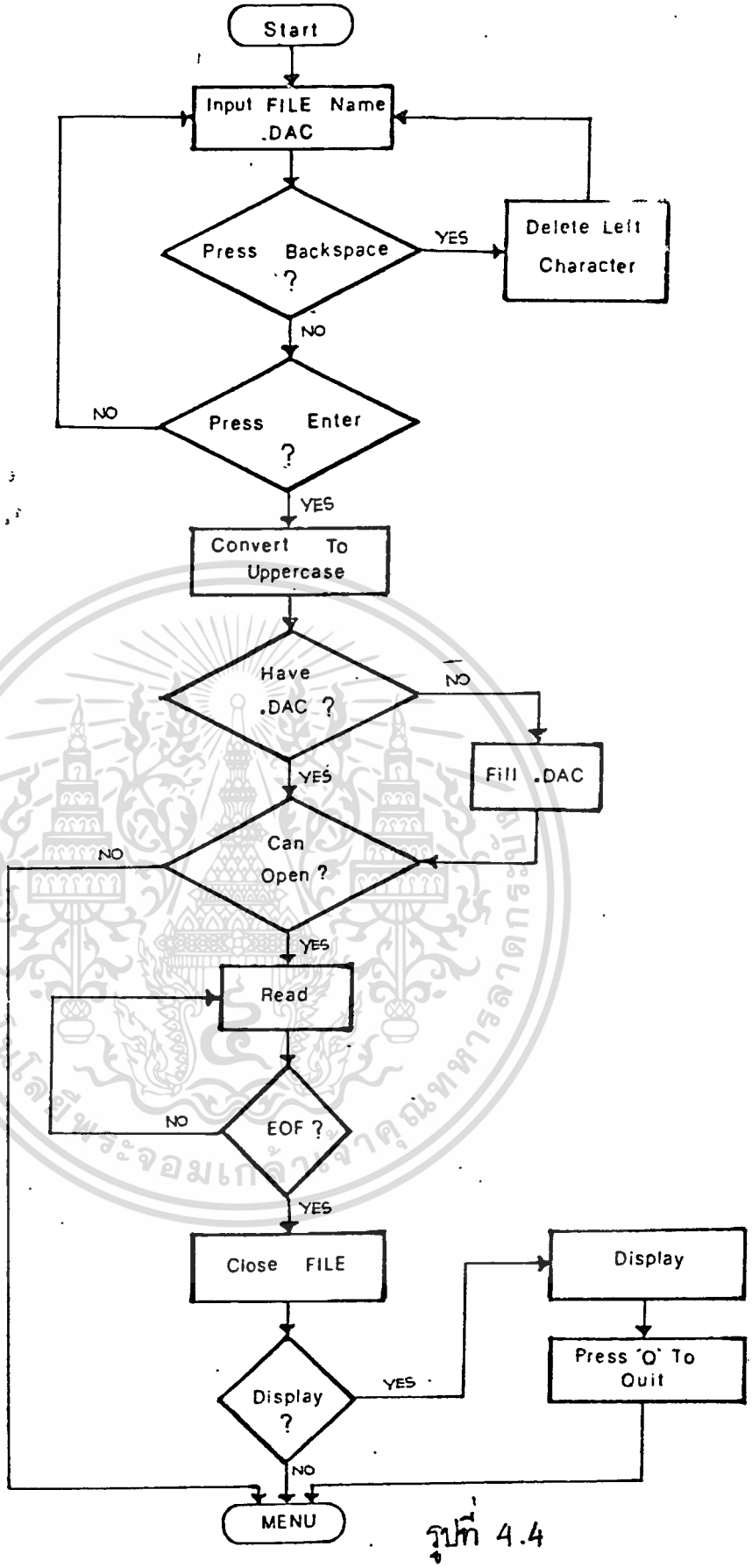


รูปที่ 4.2

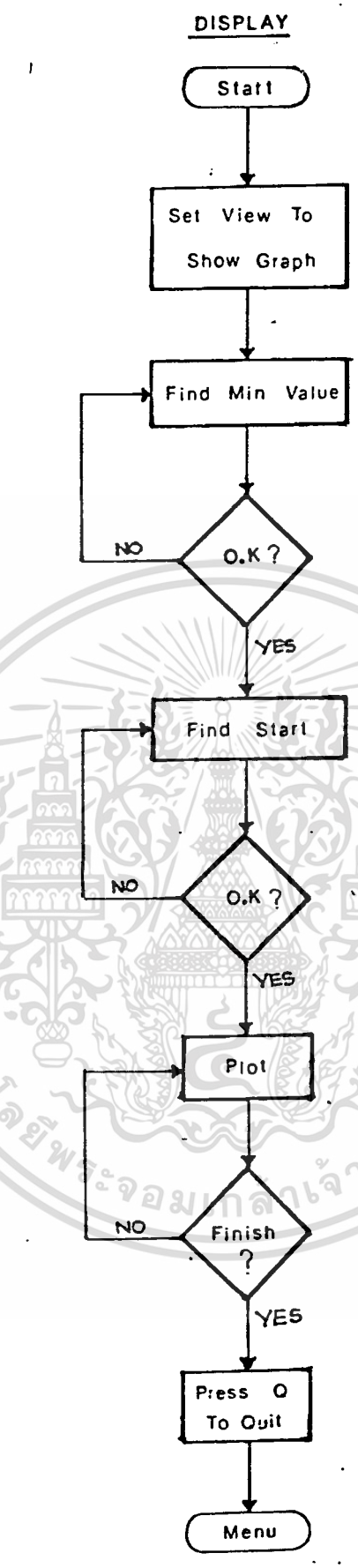
WRITE TO FILE



รูปที่ 4.3

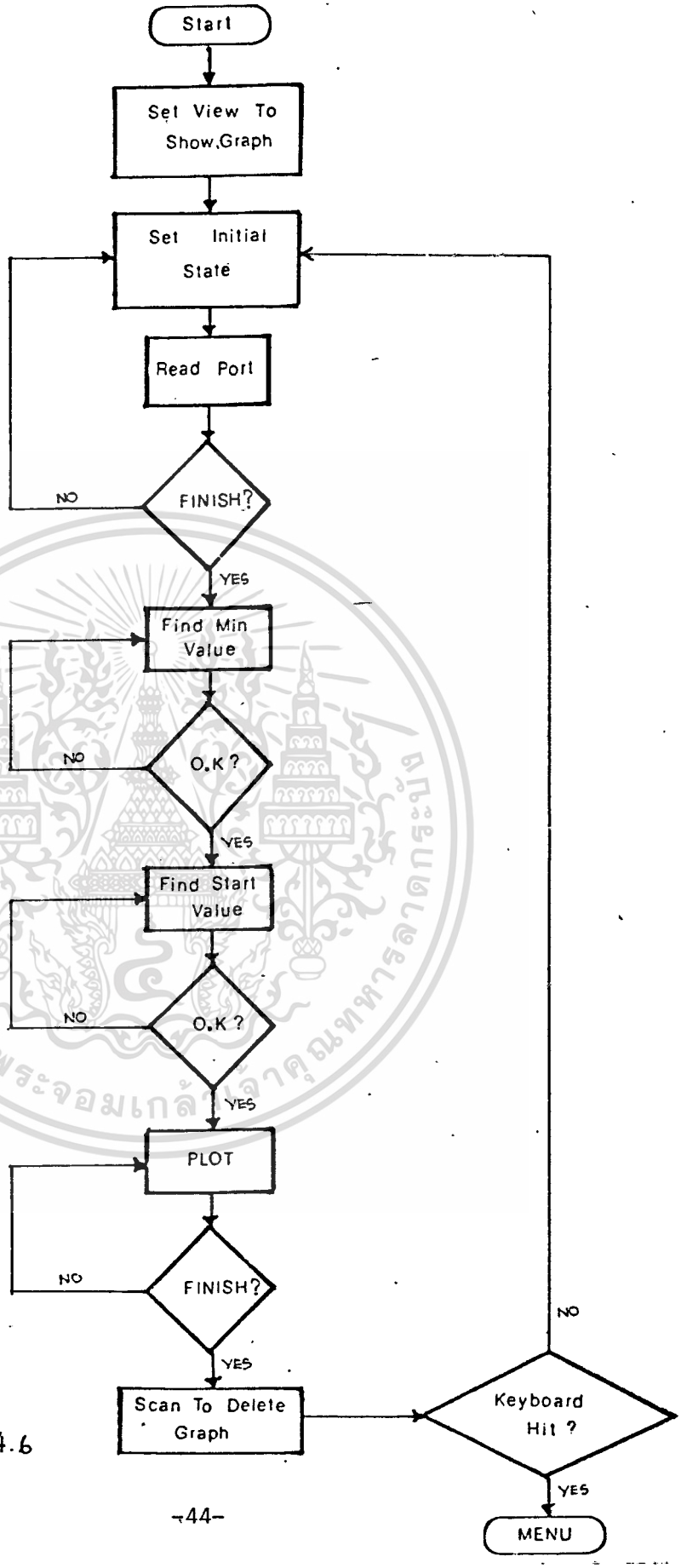


รูปที่ 4.4



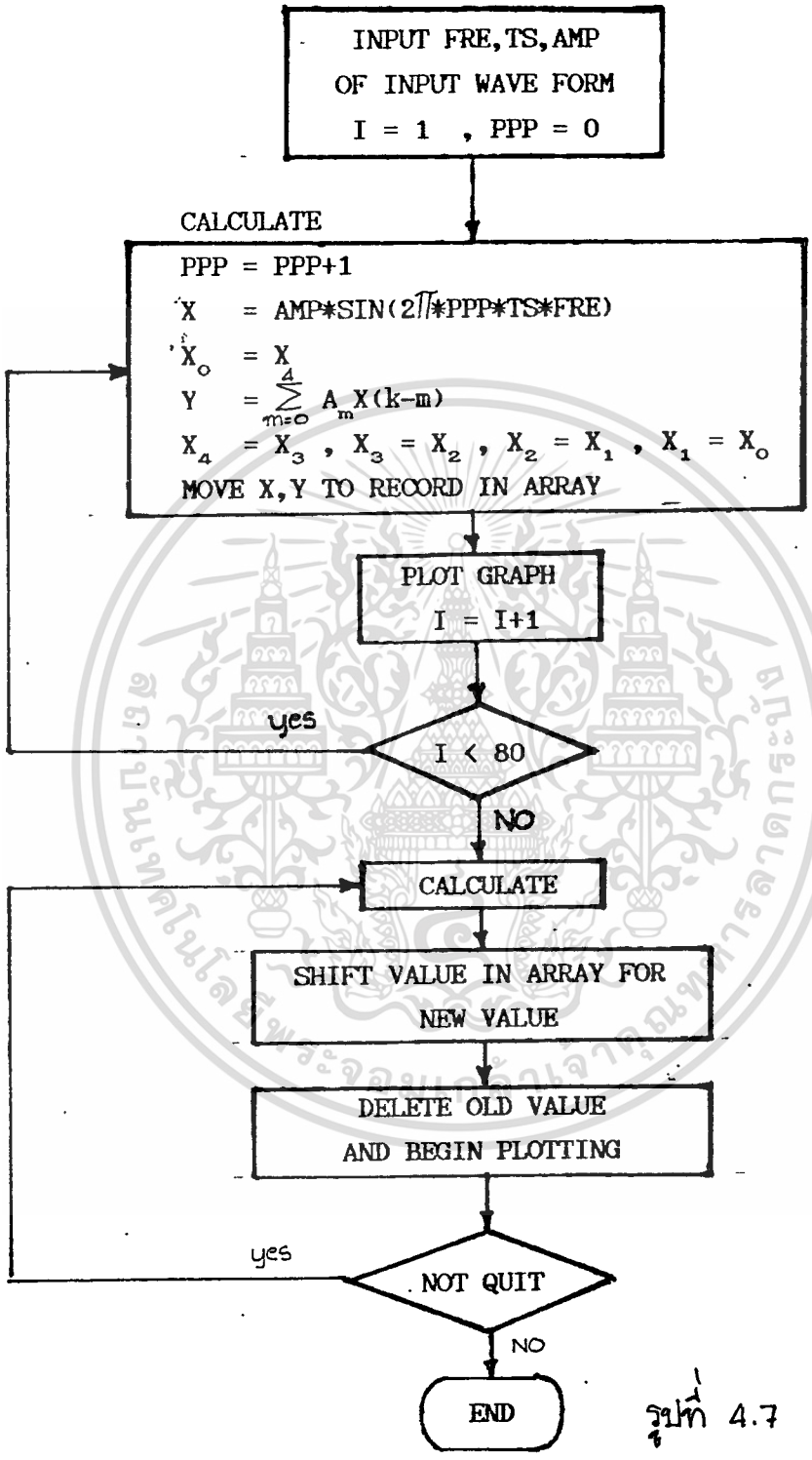
รูปที่ 4.5

DISPLAY CONTINUOUS



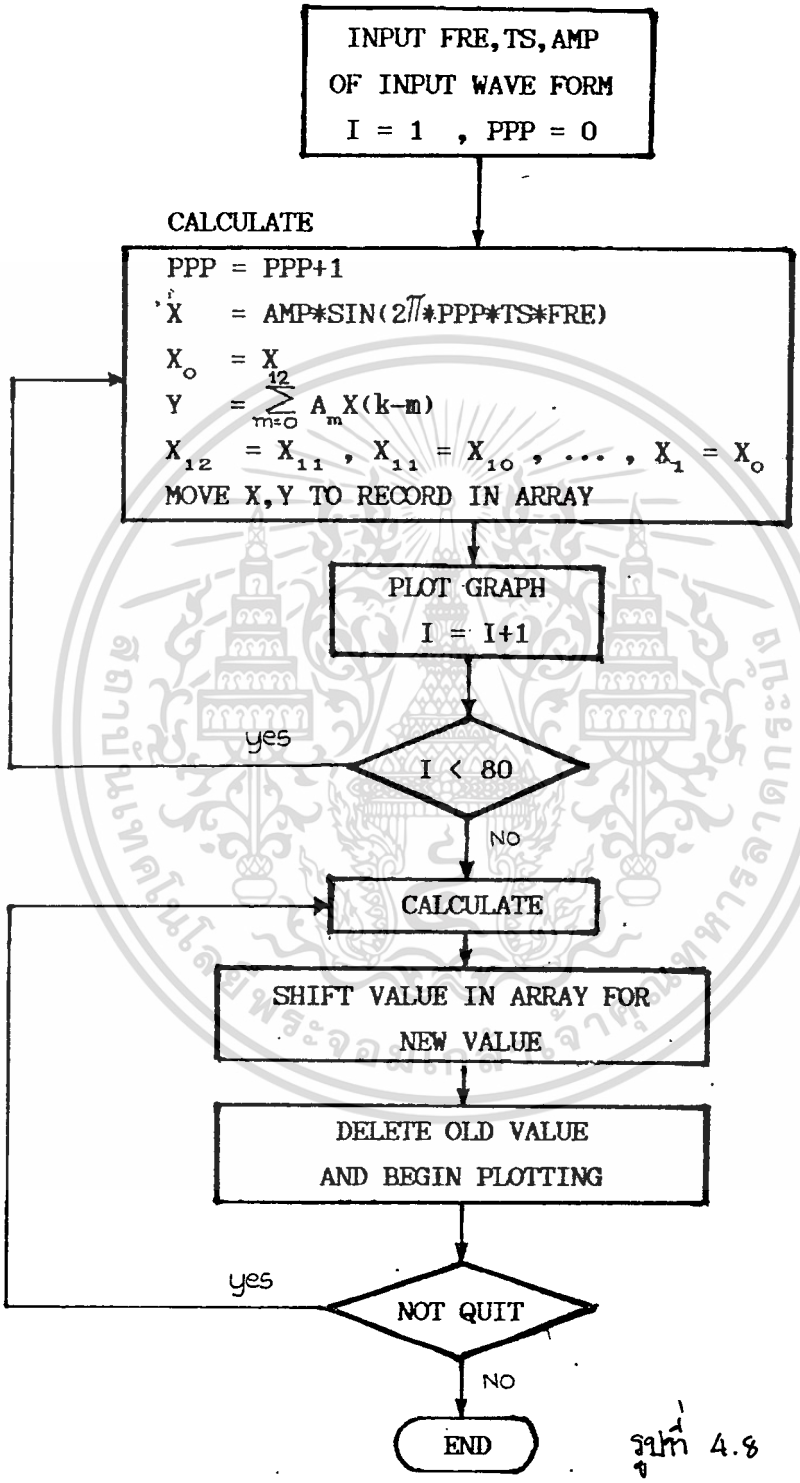
รูปที่ 4.6

PROGRAM PRO21.C



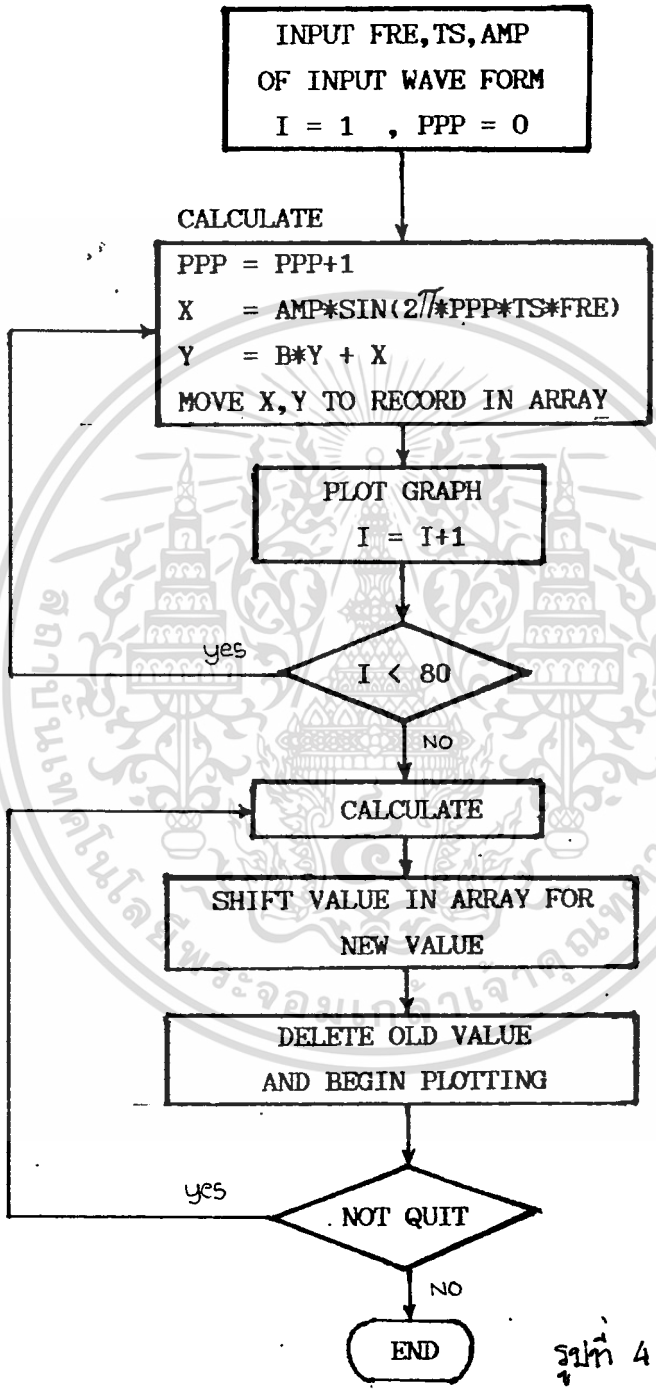
รูปที่ 4.7

PROGRAM PRO22.C



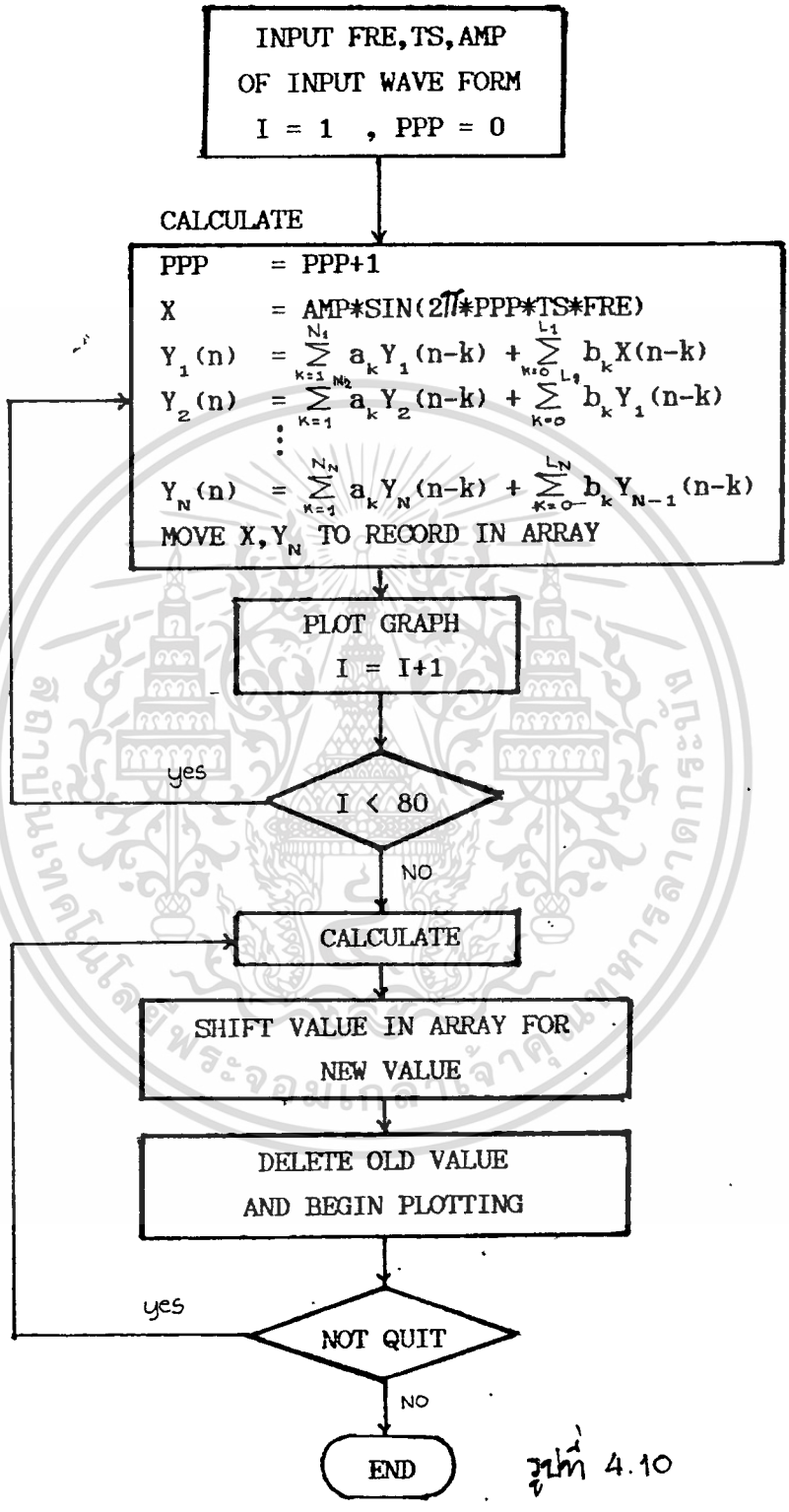
รูปที่ 4.8

PROGRAM PRO23.C



รูปที่ 4.9

PROGRAM PRO24C.C



รูปที่ 4.10

```
/*-------------------------------------------------------*/
/*                  PROGRAM PRO21.C                      */
/*   FIR.DF - Finite Impulse Response Digital Filter.    */
/*        Simulate O/P  from input DATA program.         */
/*             To show Processing of FIR.                */
/*-------------------------------------------------------*/

#include<graphics.h>
#include<math.h>
#define  HONSU   80
#define  AIDA    4
#define  KANKAK  2
#define  S       10
int      SDATA[161],TDATA[161],GX[161],GY[161],ppp=0,pc=0,
         ssdata[1],ttdata[1];
float    XAMP,YAMP,X0,X1,X2,X3,X4,Y;
float    F,T,A;
int keisan(amp,fre,ts)
float amp,fre,ts;
{
    float A0=-1.4, A1=11.2, A2=0, A3=-11.2, A4=1.4;
    ppp    = ppp+1;
       XAMP   = amp*sin(6.28318*ppp*ts*fre);
       X0     = XAMP;
       Y      = A0*X0 + A1*X1 + A2*X2 + A3*X3 + A4*X4 ;
       YAMP   = Y;
       X4     = X3;
       X3     = X2;
       X2     = X1;
       X1     = X0;
    ssdata[0] = 160-S*XAMP;
    ttdata[0] = 160-S*YAMP;
    if(ssdata[0]>=320)
       ssdata[0] = 320;
    else
    {  if(ssdata[0]<=1)
          ssdata[0] = 1;
    }
    if(ttdata[0]>=320)
       ttdata[0] = 320;
    else
    {  if(ttdata[0]<=1)
          ttdata[0] = 1;
    }
}

int graph(I)
int I;
{
   GX[I] = I*AIDA +150;
   GY[I] = I*AIDA +150 + 0.5*AIDA;
   pc    = pc+1;
   if(pc<=HONSU)
   {
```

```
      SDATA[pc] = ssdata[0];
      TDATA[pc] = ttdata[0];
      line(GX[I],160,GX[I],SDATA[I]);
      line(GY[I],160,GY[I],TDATA[I]);
   }
   else;

}

void gamen()
{
   rectangle(149,0,476,321);
   line(150,160,475,160);
}

main()
{
   int i,n,k,g_mode,g_driver=DETECT,quit=0;
   float F,T,A;
   clrscr();
   /*-------------------------------------*/
   /*            INPUT  DATA              */
   /*           Frequency     :           */
   /*           Time Sampling :           */
   /*           Amplitude     :           */
   /*-------------------------------------*/
   gotoxy(25,10);printf(" Frequency     : ");
   scanf("%f",&F);
   gotoxy(25,11);printf(" Time Sampling : ");
   scanf("%f",&T);
   gotoxy(25,12);printf(" Amplitude     : ");
   scanf("%f",&A);
   /*-------------------------------------*/
   /*    Calculate and Show Graph O/P     */
   /*-------------------------------------*/
   initgraph(&g_driver,&g_mode,"bgi");
   gamen();
   for(i=1 ; i<=HONSU ; i++)
   {
       keisan(A,F,T);
       graph(i);
   }
   while(quit!=17)
   {
       keisan(A,F,T);
       for(n=0 ; n<=HONSU-1 ; n++)
       {   SDATA[n] = SDATA[n+1];
           TDATA[n] = TDATA[n+1];
       }

       SDATA[HONSU] = ssdata[0];
       TDATA[HONSU] = ttdata[0];
       for(k=1 ; k<=HONSU ; k++)
       {
```

```
            setcolor(0);
            line(GX[k],160,GX[k],SDATA[k-1]);
            line(GY[k],160,GY[k],TDATA[k-1]);
            setcolor(1);
            line(GX[k],160,GX[k],SDATA[k]);
            line(GY[k],160,GY[k],TDATA[k]);
         }
       quit++ ;
   }
   getch();
   closegraph();
   /*-------------------------------------*/
   /*        END OF PROGRAM PRO21.C       */
   /*-------------------------------------*/

}
```

```
/*-------------------------------------------------------------*/
/*                    PROGRAM PRO22.C                          */
/*     FIR.DF - Finite Impulse Response Digital Filter         */
/*            - Show another processing of FIR                 */
/*-------------------------------------------------------------*/

#include <graphics.h>
#include <math.h>
#define   HONSU   80
#define   AIDA    4
#define   KANKAK  2
#define   S       10
int       SDATA[161],TDATA[161],GX[161],GY[161],ppp=0,pp,
          pc=0,ssdata[1],ttdata[1];
float     XAMP,YAMP,X0,X1,X2,X3,X4,Y;
float     A[13]={0,0.1,0.2,0.3,0.4,0.5,0.6,-0.5,-0.4,-0.3,
                 -0.2,-0.1,0},
          W[512];

int keisan(amp,fre,ts)
float amp,fre,ts;
{
   int m;
       ppp    = ppp+1;
       XAMP   = amp*sin(6.28318*ppp*ts*fre);
       W[0]   = XAMP;
       Y      = A[0]*W[0] ;
       for(m=1 ; m<=12 ; m++)
       {
          Y    = Y + A[m]*W[m];
          W[m] = W[m-1];
       }
    YAMP    = Y;
    ssdata[0] = 160-S*XAMP;
    ttdata[0] = 160-S*YAMP;

    if(ssdata[0]>=320)
       ssdata[0] = 320;
    else
    {  if(ssdata[0]<=1)
          ssdata[0] = 1;
    }
    if(ttdata[0]>=320)
       ttdata[0] = 320;
    else
    {  if(ttdata[0]<=1)
          ttdata[0] = 1;
    }
}

int graph(I)
int I;
{
   GX[I] = I*AIDA +150;
```

```
    GY[I] = I*AIDA +150 + 0.5*AIDA;
    pc    = pc+1;
    if(pc<=HONSU)
    {
       SDATA[pc] = ssdata[0];
       TDATA[pc] = ttdata[0];
       line(GX[I],160,GX[I],SDATA[I]);
       line(GY[I],160,GY[I],TDATA[I]);
    }
    else;

}

void gamen()
{
    rectangle(149,0,476,321);
    line(150,160,475,160);
}

main()
{
    int i,n,k,g_mode,g_driver=DETECT,quit =0;
    float F,T,A;
    clrscr();
    /*-------------------------------------*/
    /*            INPUT DATA               */
    /*           Frequency     :           */
    /*           Time Sampling :           */
    /*           Amplitude     :           */
    /*-------------------------------------*/
    gotoxy(25,10);printf(" Frequency     : ");
    scanf("%f",&F);
    gotoxy(25,11);printf(" Time Sampling : ");
    scanf("%f",&T);
    gotoxy(25,12);printf(" Amplitude     : ");
    scanf("%f",&A);
    /*-------------------------------------*/
    /*     Calculate and Show graph O/P    */
    /*-------------------------------------*/
    initgraph(&g_driver,&g_mode,"bgi");
    gamen();
    for(i=1 ; i<=HONSU ; i++)
    {
        keisan(A,F,T);
        graph(i);
    }
    while(quit!=17)
    {
        keisan(A,F,T);
        for(n=0 ; n<=HONSU-1 ; n++)
        {   SDATA[n] = SDATA[n+1];
            TDATA[n] = TDATA[n+1];
        }
```

```
            SDATA[HONSU] = ssdata[0];
            TDATA[HONSU] = ttdata[0];
            for(k=1 ; k<=HONSU ; k++)
            {
                setcolor(0);
                line(GX[k],160,GX[k],SDATA[k-1]);
                line(GY[k],160,GY[k],TDATA[k-1]);
                setcolor(1);
                line(GX[k],160,GX[k],SDATA[k]);
                line(GY[k],160,GY[k],TDATA[k]);
             }
            quit++;
        }
        getch();
        closegraph();
        /*--------------------------------------*/
        /*      END OF PROGRAM PRO22.C          */
        /*--------------------------------------*/
}
```

```
/*-------------------------------------------------------------*/
/*                    PROGRAM PRO23.C                          */
/*   IIR.DF - Infiinite Impulse Response Digital Filter        */
/*          - Show processing of IIR.                          */
/*          - Simulate O/P from DATA program.                  */
/*-------------------------------------------------------------*/

#include <graphics.h>
#include <math.h>
#define   HONSU   80
#define   AIDA    4
#define   KANKAK  2
#define   S       10
#define   B       0.8
int       SDATA[161],TDATA[161],GX[161],GY[161],ppp=0,pc=0,
          ssdata[1],ttdata[1];
float     XAMP,YAMP,X0,X1,X2,X3,X4,Y;

int keisan(amp,fre,ts)
float amp,fre,ts;
{
   int m;
       ppp    = ppp+1;
       XAMP   = amp*sin(6.28318*ppp*ts*fre);
       YAMP   = B*YAMP+XAMP;
    ssdata[0] = 160-S*XAMP;
    ttdata[0] = 160-S*YAMP;

    if(ssdata[0]>=320)
       ssdata[0] = 320;
    else
    {  if(ssdata[0]<=1)
          ssdata[0] = 1;
    }
    if(ttdata[0]>=320)
       ttdata[0] = 320;
    else
    {  if(ttdata[0]<=1)
          ttdata[0] = 1;
    }
}

int graph(I)
int I;
{
   GX[I] = I*AIDA +150;
   GY[I] = I*AIDA +150 + 0.5*AIDA;
   pc    = pc+1;
   if(pc<=HONSU)
   {
      SDATA[pc] = ssdata[0];
      TDATA[pc] = ttdata[0];
      line(GX[I],160,GX[I],SDATA[I]);
      line(GY[I],160,GY[I],TDATA[I]);
```

```
    }
    else;

}

void gamen()
{
    rectangle(149,0,476,321);
    line(150,160,475,160);
}

main()
{
    int i,n,k,g_mode,g_driver=DETECT,quit =0;
    float F,T,A;
    clrscr();
    /*-------------------------------------*/
    /*            INPUT DATA               */
    /*           Frequency      :          */
    /*           Time Sampling  :          */
    /*           Amplitude      :          */
    /*-------------------------------------*/
    gotoxy(25,10);printf(" Frequency     : ");
    scanf("%f",&F);
    gotoxy(25,11);printf(" Time Sampling : ");
    scanf("%f",&T);
    gotoxy(25,12);printf(" Amplitude     : ");
    scanf("%f",&A);
    initgraph(&g_driver,&g_mode,"bgi");
    gamen();
    for(i=1 ; i<=HONSU ; i++)
    {
        keisan(A,F,T);
        graph(i);
    }
    while(quit!=17)
    {
        keisan(A,F,T);
        for(n=0 ; n<=HONSU-1 ; n++)
        {   SDATA[n] = SDATA[n+1];
            TDATA[n] = TDATA[n+1];
        }

        SDATA[HONSU] = ssdata[0];
        TDATA[HONSU] = ttdata[0];
        for(k=1 ; k<=HONSU ; k++)
        {
            setcolor(0);
            line(GX[k],160,GX[k],SDATA[k-1]);
            line(GY[k],160,GY[k],TDATA[k-1]);
            setcolor(1);
            line(GX[k],160,GX[k],SDATA[k]);
            line(GY[k],160,GY[k],TDATA[k]);
         }
```

```
        quit++;
    }
    getch();
    closegraph();
    /*--------------------------------------*/
    /*       END OF PROGRAM PRO23.C         */
    /*--------------------------------------*/
}
```

```
/*-------------------------------------------------------*/
/*                PROGRAM PRO24C.C                       */
/*   IIR.DF - Infinite Impluse Response Digital Filter   */
/*          - Cascade Connection                         */
/*          - Show I/P and O/P                           */
/*          - O/P simulate from I/P DATA Program.        */
/*-------------------------------------------------------*/

#include <graphics.h>
#include <math.h>
#include <stdlib.h>
#define   HONSU   80
#define   AIDA    4
#define   S       10
int     dec,sign;
int     GX[161],GY[161],pc=0,ppp=0,Am;
float   XAMP,YAMP,T,F,Y,W[2][7],WO,AW,BW,X,HO= 0.025809,ssdata[1],
        ttdata[1],SDATA[161],TDATA[161];
float   A[2][7]=
       {{-2.63093,-2.08901,-1.13278,0.03758,1.15455,1.80805,1.96354},
       {{1.81674,1.79040,1.76033,1.1009100,1.60487,1.00000,1.00000}};
float   B[2][7]=
       {{-1.44816,-1.16679,-0.64350,0.02198,0.71940,0.72212,1.34416},
       {{0.55043,0.55853,0.56808,0.58490,0.62310,0.23602,0.80027}};

int keisan()
{
   int m,t;
       ppp    = ppp +1;
       XAMP   = Am*sin(6.28318*ppp*T*F);
       X      = HO*XAMP;
       for(t=0 ; t<=6 ;t++)
       {
          AW  = A[0][t]*W[0][t] + A[1][t]*W[1][t];
          BW  = B[0][t]*W[0][t] + B[1][t]*W[1][t];
          WO  = X-BW;
          X   = WO+AW;
          W[1][t] = W[0][t];
          W[0][t] = WO;
       }
    YAMP      = X;
    ssdata[0] = 90-S*XAMP;
    ttdata[0] = 260-S*YAMP;

    if(ssdata[0]>=180)
       ssdata[0] = 180;
    else
       {  if(ssdata[0]<=1)
             ssdata[0] = 1;
       }
    if(ttdata[0]>=340)
       ttdata[0] = 340;
    else
       {  if(ttdata[0]<=1)
```

```
            ttdata[0] = 1;
        }
}

int graph(i)
{
   GX[i] = i*AIDA +150;
   GY[i] = i*AIDA +150 + 0.5*AIDA;
   pc    = pc+1;
   if(pc<=HONSU)
   {
      SDATA[pc] = ssdata[0];
      TDATA[pc] = ttdata[0];
      line(GX[i],90,GX[i],(int)SDATA[i]);
      line(GY[i],260,GY[i],(int)TDATA[i]);
   }
   else;

}

void gamen()
{
   int d;
   rectangle(149,0,476,173);
   rectangle(149,177,476,340);
   line(150,90,475,90);
   line(150,260,475,260);
   outtextxy(50,82,"INPUT X(m)");
   outtextxy(50,252,"OUTPUT Y(n)");
}

main()
{
   int i,n,k,g_mode,g_driver=DETECT,ST=0;

   clrscr();
   gotoxy(25,10);printf("Frequency    : ");
   scanf("%f",&F);
   gotoxy(25,11);printf("Time Sampling : ");
   scanf("%f",&T);
   gotoxy(25,12);printf("Amplitude     : ");
   scanf("%d",&Am);
   initgraph(&g_driver,&g_mode,"bgi");
   gamen();
   for(i=1 ; i<=HONSU ; i++)
   {
       keisan();
       graph(i);
   }
   while(ST!=21)
   {
       keisan();
       for(n=0 ; n<=HONSU-1 ; n++)
       {
```

```
            SDATA[n] = SDATA[n+1];
            TDATA[n] = TDATA[n+1];
        }

        SDATA[HONSU] = ssdata[0];
        TDATA[HONSU] = ttdata[0];
        for(k=1 ; k<=HONSU ; k++)
        {
            setcolor(0);
            line(GX[k],90,GX[k],(int)SDATA[k-1]);
            line(GY[k],260,GY[k],(int)TDATA[k-1]);
            setcolor(1);
            line(GX[k],90,GX[k],(int)SDATA[k]);
            line(GY[k],260,GY[k],(int)TDATA[k]);
         }
        ST++;
    }
    getch();
    closegraph();
}
```

```
/*-------------------------------------------------------------*/
/*                    PROGRAM PR033.C                          */
/*   FFT    - Fast Fourier Transform                           */
/*          - Show processing of FFT.                          */
/*          - Simulate O/P from DATA program.                  */
/*-------------------------------------------------------------*/

#include <graphics.h>
#include <math.h>
int     IM[30];
double  CC[30],SS[30],DR[30],DI[16],RE[30],GAIN[512];

Push(x)
float x;
{  int res;
   double pos,comp;
   res = x;
   if(x>=0)
   {   comp = x-res;
       if(comp>=0.5)
          pos = ceil((double)x);
       else
          pos = floor((double)x);
   }
   else
      if(x<0)
      {   comp = res-x;
          if(comp>=0.5)
             pos = floor((double)x);
          else
             pos = ceil((double)x);
      }
   return(pos);
}

void inputdata(n)
float n;
{ int i;
  for(i=0; i<n ;i++)
  {
        DR[i] = -pow((double)(8-i),(double)2)/64 + 1;
        DI[i] = 0;
        RE[i] = DR[i];
        IM[i] = DI[i];
  }
}

void compute(n,n2,lvalue)
float  n,n2;
double lvalue;
{
  int      LI,ARG,SCL,J1,J2,C0,S0,L0,LM,LIX,num2;
  double   LMX,C,S,T1,T2;
  LMX     = Push(n);
```

```
    SCL     = 1;
    for(LO=1; LO<=lvalue ; LO++)
    {
          LIX = LMX;
          LMX = Push((float)LMX/2);
          ARG = 0;
          for(LM=1; LM<=LMX ; LM++)
          {
              CO  = ARG%(int)n2;
              C   = CC[CO];
              SO  = ARG%(int)n2;
              S   = SS[SO];
              ARG = ARG + SCL;
              for(LI=LIX; LI<=n ;LI = LI+LIX)
              {
                  J1      = LI-LIX+LM-1;
                  J2      = J1+LMX;
                  T1      = RE[J1] - RE[J2];
                  T2      = IM[J1] - IM[J2];
                  RE[J1] = RE[J1] + RE[J2];
                  IM[J1] = IM[J1] + IM[J2];
                  RE[J2] = C*T1  + S*T2;
                  IM[J2] = Push((float)(C*T2  - S*T1));
              }
          }
    SCL = 2*SCL;
    }
}

void func(n)
float  n;
{
  int j;
  float p2,w;
  p2 = n/2;
  for(j=0 ; j<p2 ;j++)
  {
      w     = 6.28318/n*j;
      CC[j] = cos((double)w);
      SS[j] = sin((double)w);
  }
}

Bitrev(n,n2)
float n,n2;
{
   int J,I,n3,X2;
   float K,MED;
   double X1;
   J  = Push(n2+1);
   n3 = Push(n-2);
   for( I=2 ; I<=n3 ; I++)
   {
        if(I>=J) K = Push(n2);
```

```
            else
            {
               X1      = RE[I-1];
               RE[I-1] = RE[J-1];
               RE[J-1] = X1;

               X2      = IM[I-1];
               IM[I-1] = IM[J-1];
               IM[J-1] = X2;
               K       = Push(n2);
            }
            while(K<J)
            {
                 J = J-K;
                 K = Push(K/2);
            }
            J=J+K;
      }
   }

   Spec(num,num2,z_log)
   float num,num2,z_log;
   {
      int i;
      func(num);
      inputdata(num);
      compute(num,num2,z_log);
      Bitrev(num,num2);
      for(i=0;i<num-1;i++)
          GAIN[i] =  sqrt(pow(RE[i],2)+pow(IM[i],2));
    }

    int Display_viewport(m,g)
    int  m;
    double g;
    {
        int i;
        char str[10];
        rectangle(100,40,620,328);
        for(i=88;i<=328;i=i+48)
            line(99,i,101,i);
        for(i=204;i<=620;i=i+104)
            line(i,327,i,329);
        for(i=32;i<=320;i=i+48)
        {
            outtextxy(85,i,(char*)itoa(m,str,10));
            m = m-g;
        }
        outtextxy(100,330,"0.00");
        outtextxy(620,330,"0.50");
        return(m);
    }

    Show_bar(amax,amin,n2)
```

```
int   amax,amin;
float n2;
{   int    i,Y;
    float  NK;
    NK  =_ 512/n2;
    for(i=0;i<=n2;i++)
    {    Y = (-GAIN[i]+amax)/(amax-amin)*288;
         if((Y<0) || (Y>288));
         else
             line(i*(int)NK+100,Y+39,i*(int)NK+100,327);
    }
}

main()
{
   double GMAX=-1e+10,GMIN=1e+10,G;
   float  N,N2,Z;
   int    AMAX,AMIN,M,g_driver=DETECT,g_mode,i;
       clrscr();
       printf("Input N Value : ");scanf("%f",&N);
       Z  = log((double)N)/log((double)2);
       N2 = N/2;
       for(i=0;i<=N2;i++)
       {   if(GMAX<GAIN[i])  GMAX=GAIN[i];
           if(GMIN>GAIN[i])  GMIN=GAIN[i];
       }
       if(fabs(GMAX-GMIN)<0.00001)
       {  GMAX = GMAX*1.2;
          GMIN = 0;
       }
       G    = GMAX - GMIN;
       G    = Push((float)G/6)+1;
       AMAX = Push((float)GMIN) + 6*G;
       M    = AMAX;
       Spec(N,N2,Z);
       initgraph(&g_driver,&g_mode,"bgi");
       M = Display_viewport(M,G);
       AMIN = M+G;
       Show_bar(AMAX,AMIN,N2);
       getch();
       closegraph();
}
```

บทที่ 5
ผลการทดลอง

ผลการทดลองต่อไปนี้เป็นการจำลองระบบขึ้นมา INPUT ที่ป้อนให้กับระบบเป็นอนุกรมของข้อมูลที่ได้จากการคำนวน ดังนั้นจึงยังไม่ได้นำเอาอนุกรมข้อมูลที่ทำการอ่านเข้ามาจากการแปลงสัญญาณของส่วน Hardware มาใช้เป็น INPUT จริงๆ โดย INPUT ที่ใช้อยู่ในลักษณะ Sinusoids และ ในระบบที่จำลองขึ้นมาจะทำการสุ่มสัญญาณด้วยคาบเวลา Ts (Time Sampling) ซึ่งจะนำมาเปรียบเทียบกับค่าของ Input Frequency ที่ป้อนเข้าไป ในการทดลองนี้จะใช้ Frequency = 1 Hz และกำหนดให้พารามิเตอร์ต่างๆดังนี้

FRE : Frequency of Input Signal.

Ts : Time Sampling.

Amp : Amplitude of input Signal.

ผลการทดลองนี้จะแสดง OUTPUT ที่เกิดจากการตอบสนองของระบบดังกล่าวเทียบกับ INPUT SIGNAL โดย

5.1) PROGRAM PRO21.C (FIR)

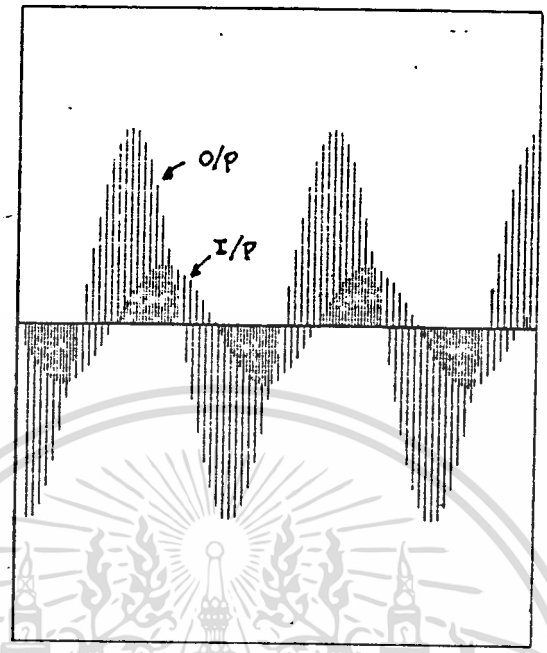
5.2) PROGRAM PRO22.C (FIR)

5.3) PROGRAM PRO23.C (IIR)

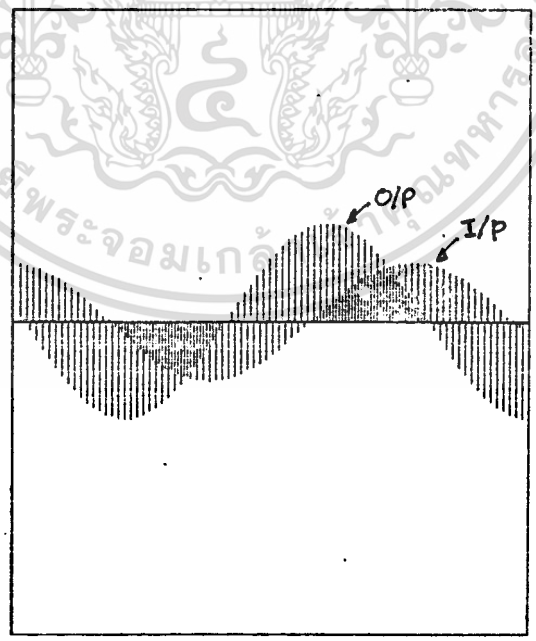
5.4) PROGRAM PRO24C.C (IIR)

5.5) PROGRAM PRO33.C (DFT)

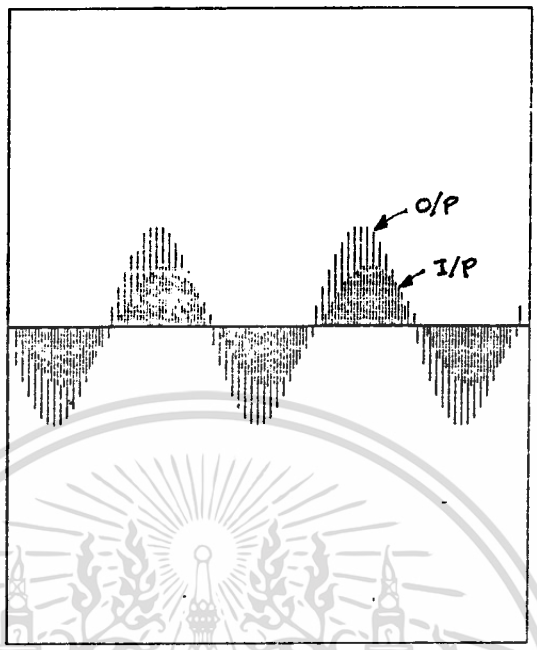
5.1) FIR : FRE = 1 Hz , Ts = .03125 s , AMP = 3



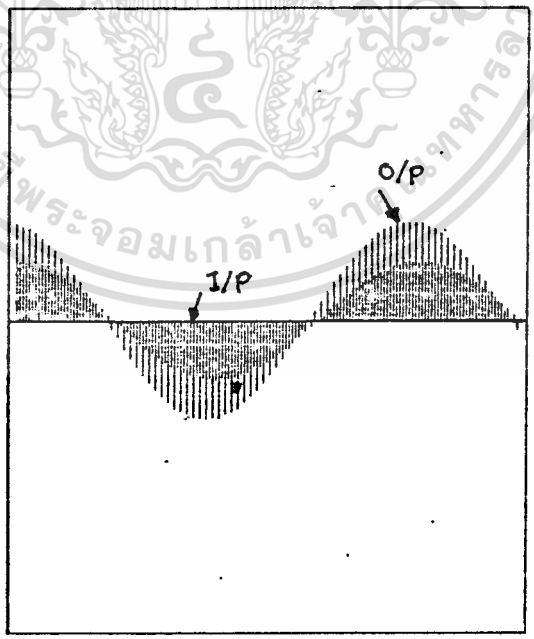
5.1) FIR : FRE = 1 Hz , Ts = .015625 s , AMP = 3



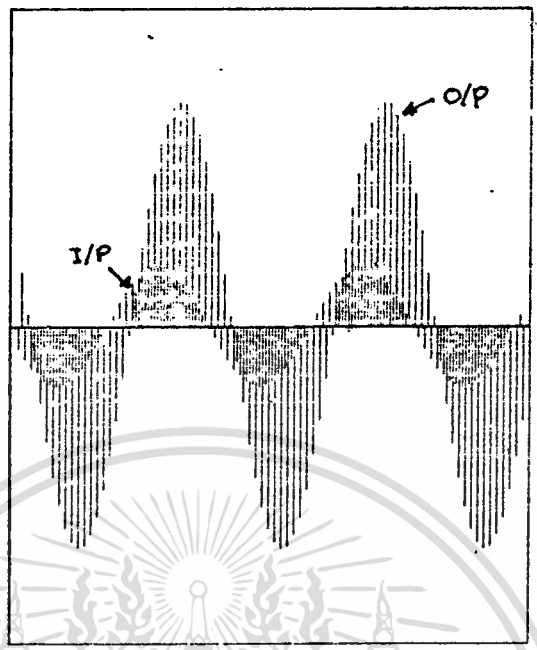
5.2) FIR : FRE = 1 Hz , Ts = .03125 s , AMP = 5



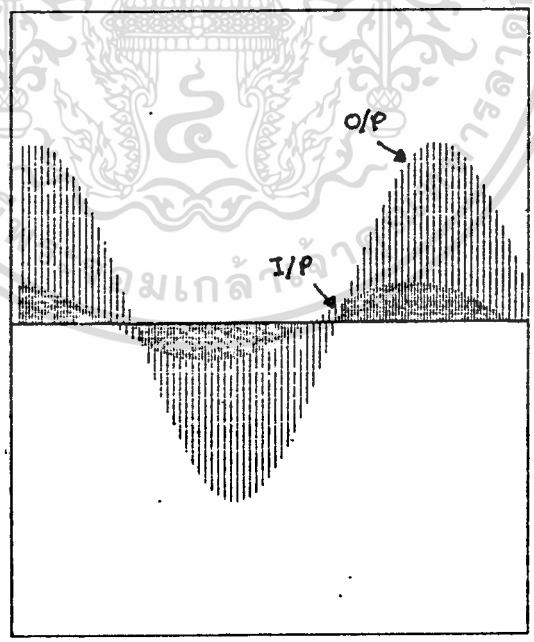
5.2) FIR : FRE = 1 Hz , Ts = .015625 s , AMP = 5



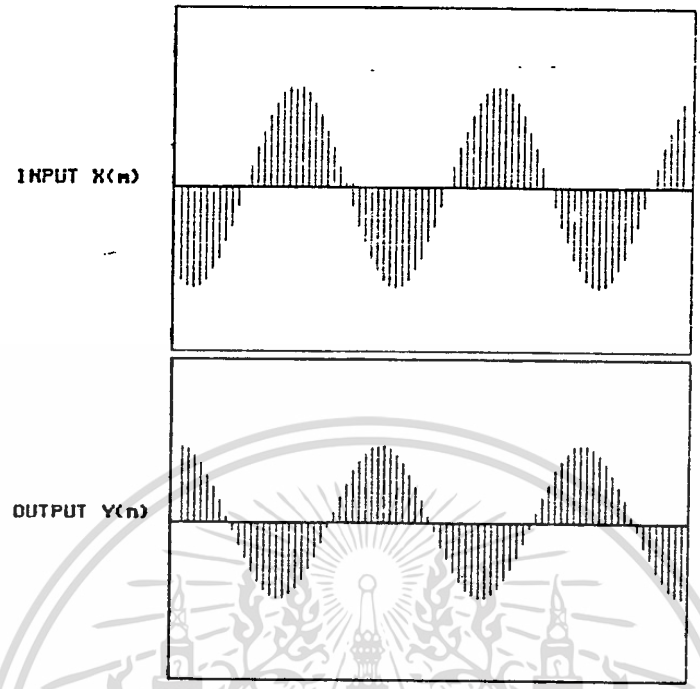
5.3) IIR : FRE = 1 Hz , Ts = .03125 s , AMP = 3



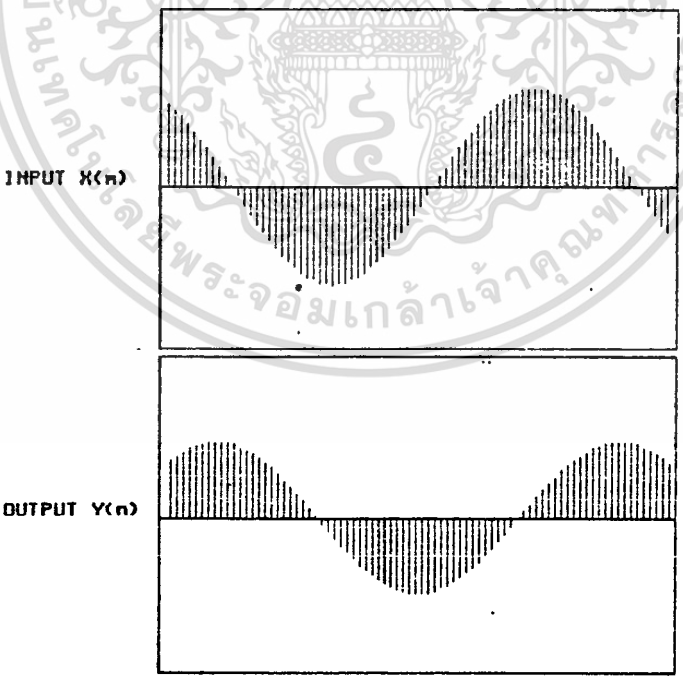
5.3) IIR : FRE = 1 Hz , Ts = .015625 s , AMP = 3



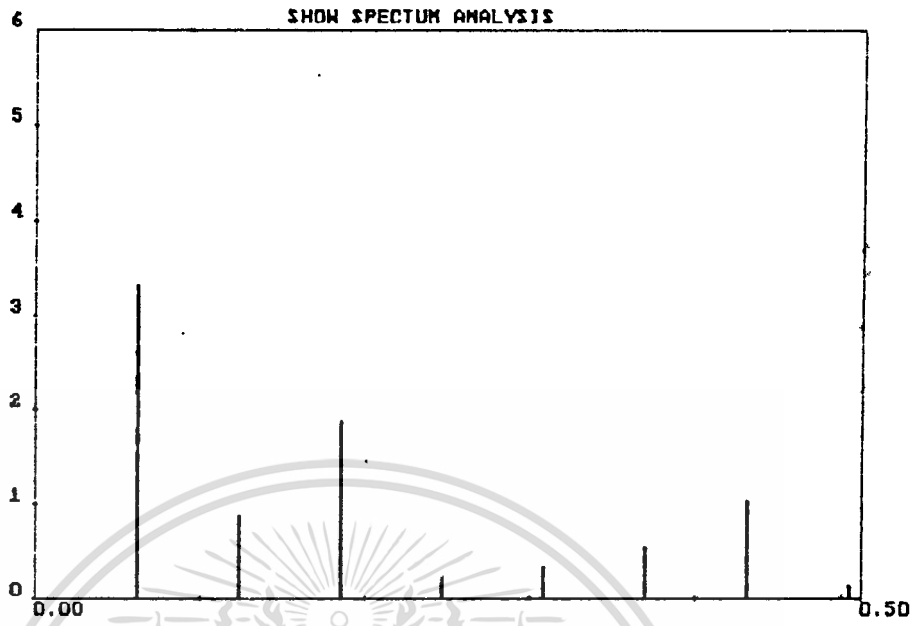
5.4) IIR : FRE = 1 Hz , Ts = .03125 s , AMP = 5



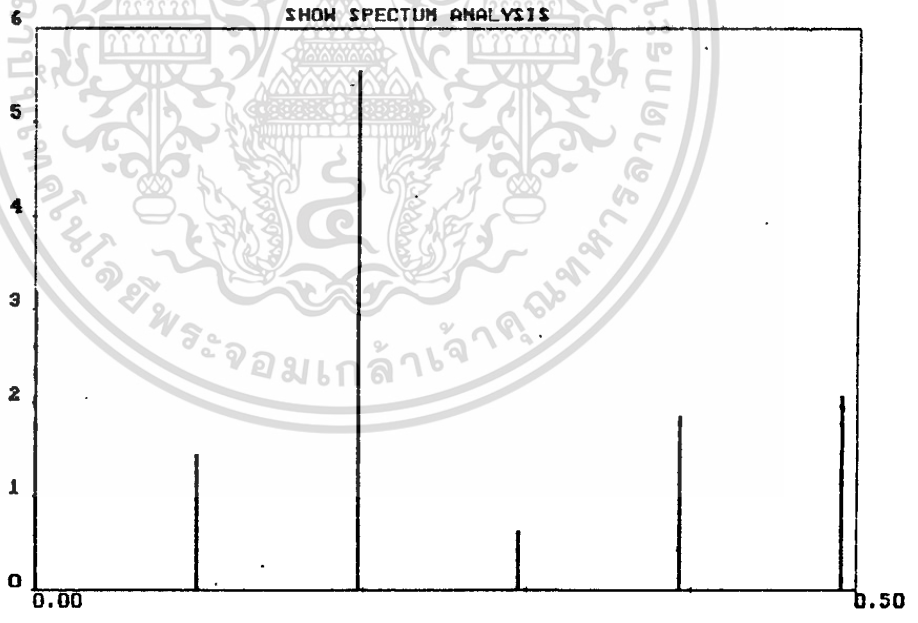
5.4) IIR : FRE = 1 Hz , Ts = .015625 s , AMP = 5



5.5) DFT N = 16 ; N = NUMBER OF SAMPLE POINT IN ONE PERIOD



5.5) DFT N = 10 ; N = NUMBER OF SAMPLE POINT IN ONE PERIOD



บทที่ 6
สรุปและวิจารณ์

จากผลการทดลอง ทำให้เห็นแนวทางการนำไปใช้งานของ DSP เพราะเราสามารถที่จะคำนวนค่าหาพารามิเตอร์เพื่อให้ระบบทำงานตามที่เราต้องการได้

ในโปรเจคนี้เป็นกรณีศึกษาเพื่อให้เข้าใจ ในกระบวนการขั้นพื้นฐานของ DIGITAL SIGNAL PROCESSING ส่วนหนึ่งอันได้แก่ การแปลงสัญญาน ANALOG ไปเป็น DIGITAL และทำการเก็บข้อมูลที่ได้จากการแปลงไว้โดยอาศัยการ INTERFACE กับ COMPUTER และทดลองนำเอาข้อมูลนั้นมาแสดงผลออกจอ เพื่อดูลักษณะของ WAVE FORM ซึ่งใช้กับสัญญานที่มีความต่างศักย์ 0-5 Volt ความถี่ 1-100 Hz นี่เป็นตัวอย่างหนึ่งในการนำเอาข้อมูลที่เก็บได้มาประมวลผลให้เห็นลักษณะของสัญญาน

จากนั้นได้ทำการศึกษาถึงกระบวนการทาง Mathermatic ที่จะนำมาใช้ในการประมวลผลข้อมูล เพื่อให้ข้อมูลนั้นอยู่ในรูปแบบที่เหมาะสม พร้อมที่จะแยกแยะและวิเคราะห์ในรายละเอียดต่อไป ในโปรเจคนี้ยังขาดการดึงเอาข้อมูลที่อ่านเข้ามาจริงมาใช้ในการคำนวน ดังนั้นจึงใช้การจำลองสัญญาน INPUT เพื่อให้เห็นการทำงานของระบบแทน

ถ้าสามารถนำเอาข้อมูลที่ได้จากการแปลงสัญญานจริงๆ เข้ามาใช้ในการคำนวนได้ก็จะทำให้เข้าใจในด้าน APPLICATION ได้ดียิ่งขึ้น ซึ่งก็จะต้องทำการศึกษาต่อไป

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปด้วยดี เพราะได้รับคำแนะนำจากอาจารย์ที่ปรึกษา รศ. มนัส สังวรศิลป์ และนักศึกษาปริญญาโทแผนกชีวอิเลคทรอนิคส์ รวมทั้งผู้ที่มิได้กล่าวถึงไว้ในที่นี้ ที่ให้ความสะดวกในด้านอุปกรณ์และการทดลองทุกอย่าง

ผู้จัดทำจึงขอขอบพระคุณทุกท่านมา ณ. โอกาสนี้

เอกสารอ้างอิง

1.) สานนท์ แก้วอบเชย , "SLOT IBM PC กับการต่อออก", วรสารเซมิ-คอนดักเตอร์ อิเลคทรอนิคส์ ฉบับที่ 87 ,2531 หน้า 186-196

2.) ROBERT D. STRUM , DONALD E, KIRK ,"FIRST PRINCIPAL OF DISCRETE SYSTEM AND DIGITAL SIGNAL PROCESSING"