ปีการศึกษา 2533

การประมวลผลสัญญาณเชิงเลขบนเครื่องไมโครคอมพิวเตอร์

โดย

กอบกิจ สิมะวานิชกุล

ทัศเวท วีระวัฒน์

อาจารย์ที่ปรึกษา

ผศ.ดร. กอบชัย เดชหาญ

อาจารย์ สมยศ จุณณะปิยะ

ปริญญานิพนธ์ปีการศึกษา 2533

ภาควิชาโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประมวลผลสัญญาณเชิงเลขบนเครื่องไมโครคอมพิวเตอร์

ผู้จัดทำ

1. กอบกิจ สิมะวานิชกุล
2. ทัศเวท วีระวัฒน์

.... อาจารย์ที่ปรึกษา

(ผศ.ดร. กอบชัย เดชหาญ)

.... อาจารย์ที่ปรึกษา

(อ.สมยศ จุณณะปิยะ)

การประมวลผลสัญญาณเชิงเลขบนเครื่องไมโครคอมพิวเตอร์

กอบกิจ สิมะวานิชกุล
ทัศเวท วีระวัฒน์
อาจารย์ที่ปรึกษา
ผศ.ดร.กอบชัย เดชหาญ
อาจารย์ สมยศ จุณณะปิยะ

บทคัดย่อ

เนื่องจากระบบดิจิตอลในปัจจุบันนี้มีการพัฒนามากขึ้นเรื่อยๆ ซึ่งถ้านำระบบดิจิตอลมาใช้ในการประมวลผลสัญญาณแทนวงจรอนาลอก จะทำให้ได้ผลในการโพรเซสที่มีคุณภาพดีกว่าและมีความยืดหยุ่นมากกว่า โครงงานนี้จึงเป็นการศึกษาการออกแบบบอร์ดตัวประมวลผลสัญญาณเชิงเลข ซึ่งใช้ตัวประมวลผลสัญญาณเชิงเลขเบอร์ TMS320C25 และในการทำงานจะใช้เสียบลงบนเครื่องไมโครคอมพิวเตอร์ IBM PC โดยสามารถทำงานเป็นโพรเซสเซอร์ร่วม(co-processor) กับหน่วยประมวลผลกลาง(CPU) ของเครื่องไมโครคอมพิวเตอร์ได้ เพื่อเพิ่มขีดความสามารถในการประมวลผลสัญญาณบนเครื่องไมโครคอมพิวเตอร์

DIGITAL SIGNAL PROCESSING ON MICROCOMPUTER

Kobkij Simavanichkul
Tasawate Weerawat
Advisors
Assistant Professor Dr.Kobchai Dejhan
Mr. Somyot Junnapiyar

Abstract

Recently, the digital system is more and more developed and becomes the system for signal processing in stead of the analog system because the result of digital signal processing is much more efficient and flexible. So the point of the thesis is to study the design of the signal processor. The board is designed to plug into IBM-PC computer's slot. It allows the TMS320C25 working as a co-processor with the CPU of IBM-PC computer to perform high speed signal processing.

สารบัญ

บทที่ 1

บทนำ

การประมวลผลสัญญาณเชิงเลข (Digital Signal Processing) นั้นสามารถให้ความหมายอย่างง่าย ๆ ได้โดย การให้ความหมายเป็นคำ ๆ อันได้แก่

1. สัญญาณ(signal) คือ ปริมาณที่สนใจ เช่น ค่าโวลเตจที่เวลาใดๆ จะเห็นว่า ถ้าสัญญาณของบรรจุสาระ (Information)บางอย่างบางรูปแบบที่เราสนใจหรือ กำลังพิจารณาอยู่

2. การโพรเซส (process) การดำเนินการ (operating) ให้ลักษณะที่เป็นลำดับขั้นต่อสัญญาณเพื่อที่จะให้ได้สาระที่เราสนใจออกมา

3. ดิจิตอล (digital) เราหมายถึง การโพรเซสเราจะกระทำโดยใช้คอมพิวเตอร์ดิจิตอล หรือฮาร์ดแวร์ดิจิตอล ที่ออกแบบพิเศษเฉพาะงาน

การประมวลผลสัญญาณเชิงเลข ในปัจจุบันเริ่มได้รับความสนใจมากขึ้น เนื่องมาจาก เทคโนโลยีในการสร้างวงจรรวม หรือวงจรไอซี (Integrated Circuit) สามารถทำได้อย่างมีประสิทธิภาพดีขึ้นมา วงจรที่เคยยุ่งยากที่มีขนาดใหญ่ สามารถนำมาสร้างเป็นวงจรรวมที่ผลิตบนสารกึ่งตัวนำชิ้นเดียวกันได้ และสามารถผลิตครั้งละจำนวนมากๆได้ จึงทำให้ราคาต่อวงจรถูกลงเป็นอย่างมาก โดยเฉพาะอย่างยิ่งวงจรรวมแบบ แอลเอสไอ (LSI : Large Scale Integrated Circuit) ที่สำคัญๆ เช่น ไมโครโปรเซสเซอร์ (Microprocessor) ซึ่งมีราคาถูกลงมาก จึงมีผู้สนใจนำมาทำเป็นตัวประมวลผลสัญญาณมากขึ้น นอกจากนี้การประมวลผลเชิงตัวเลขยังให้ความเชื่อถือได้ (reliability) และความแม่นยำในการคำนวน (accuracy) ดีกว่าการประมวลผลสัญญาณเชิงอนาลอกมาก

DSP (Digital Signal Processor) เป็นไมโครโปรเซสเซอร์ที่ใช้งานเฉพาะสำหรับการประมวลผลสัญญาณ ทั้งนี้เพราะมันทำงานได้อย่างรวดเร็วมาก DSP เป็นชิพที่เกี่ยวข้องกับการประยุกต์ใช้งานด้านการสังเคราะห์สัญญาณ หรือประมวลผลสัญญาณโดยใช้อัลกอริทึมทางคณิตศาสตร์ ดังนั้น DSP จึงสามารถทำงานในระบบเวลาจริงได้ช่วงเวลาการหน่วงในระหว่างการคำนวนจะต้องน้อยที่สุดอาจจะกล่าวได้ว่า DSP เป็นอนาลอกโปรเซสเซอร์ได้ดีที่เดียว ดังนั้นในปัจจุบันจึงมีการนำมาประยุกต์ใช้กับงานต่าง ๆได้มากมาย

1.1 ข้อดีของการประมวลผลสัญญาณเชิงเลข

1. เหมาะสำหรับอุปกรณ์ที่ข้อมูลอยู่ในรูปแบบสัญญาณเชิงเลขอยู่แล้ว เช่น ผลลัพธ์จากคอมพิวเตอร์ , ไมโครโปรเซสเซอร์ ทั้งนี้เนื่องจากสัญญาณหรือข้อมูลจากอุปกรณ์เหล่านี้ ถ้าหากต้องนำไปประมวลผลในระบบการประมวลผลสัญญาณอนาลอกก็จำเป็นต้องมี วงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก (Digtal to Analog converter หรือ D/A) และหลังจากประมวลผลด้วยระบบประมวลผลเชิงอนาลอกแล้ว ก็ต้องแปลงผลลัพธ์ที่เป็นสัญญาณอนาลอกไปเป็นสัญญาณเชิงเลขด้วย วงจรแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล (Analog to Digital converter หรือ A/D) เพื่อส่งกลับไปยังอุปกรณ์เชิงเลขต่อไป ซึ่งจะทำให้ระบบยุ่งยากขึ้น อีกทั้งในปัจจุบันA/D หรือ D/A ที่มีประสิทธิภาพสูง และมีความเร็วในการแปลงสูงมีราคาแพง

2. อุปกรณ์ทางเชิงเลข หรือ ดิจิตอล มีราคาถูก ขนาดเล็ก มีประสิทธิภาพสูง ความแม่นยำและความแน่นอนสูงเพราะ การประมวลผลสัญญาณเชิงเลขสามารถคำนวนได้อย่างแม่นยำ และต้องการให้มีรายละเอียดเท่าใดก็ได้ แต่สำหรับการประมวลผลเชิงอนาลอก การคำนวนที่ให้ความละเอียดเกินกว่าหนึ่งในพันส่วนทำได้ยากมาก อีกทั้งอุปกรณ์ในการประมวลผลเชิงอนาลอก เช่น ความต้านทาน ตัวขยายสัญญาณ ยังมีคุณสมบัติการแปรค่าไปตามสภาวะแวดล้อม เช่น อุณหภูมิ ความชื้น อายุการใช้งาน ทำให้เกิดความเชื่อถือได้ต่ำ

3. การรับและส่งข้อมูลหรือสัญญาณเชิงเลข ทำได้แน่นอนกว่า ทั้งนี้เนื่องจากสัญญาณเชิงเลข มีแค่ 2 ระดับ คือ 0 กับ 1 เท่านั้น ถ้าหากรูปคลื่นสัญญาณผิดเพี้ยนไปก็สามารถแก้ไข และสร้างขึ้นมาใหม่ให้เหมือนเดิมได้โดยง่าย

4. การประมวลผลสัญญาณเชิงเลขทำได้ง่าย เพราะขั้นตอนวิธีการประมวลผลสัญญาณมักประกอบด้วย การบวก การลบ การคูณ การหาร และการเลื่อนตัวเลขเท่านั้น

5. ระบบการประมวลผลเชิงเลข สามารถทำเป็นแบบระบบแบ่งส่วนเวลา (Time-shared system) ได้ดังนั้นจึงสามารถทำการประมวลผลได้พร้อม ๆ กันหลายช่องสัญญาณได้ และนอกจากนี้ในระบบเดียวกันยังสามารถโปรแกรมให้ทำงานได้หลายรูปแบบด้วย

6. ระบบประมวลผลสัญญาณเชิงเลข มีความคล่องตัวสูง ทั้งนี้เนื่องจากสามารถทำการมัลติเพล็กซ์กับข้อมูล หรือ สัญญาณเสียง หรือ สัญญาณภาพได้ ทั้งยังสามารถทำเป็นการมัลติเพล็กซ์แบบแบ่งเวลา (TDM) หรือการมัลติเพล็กซ์ แบบแบ่งระหัส (CDM)

1.2 ข้อเสียของการประมวลผลสัญญาณเชิงเลข

1. ระบบการประมวลผลสัญญาณเชิงตัวเลขต้องมีสัญญาณสำหรับการชิงค์ (synchronization) การจัดเวลา (timing) การกำหนดขอบ (lining) ของมูลไว้ด้วย ถ้าหากสัญญาณเหล่านี้สูญหายไป หรือผิดพลาดการทำงานของระบบก็จะผิดพลาด

2. มีปัญหาในการเชื่อมต่อ (interfacing) กับระบบการประมวลสัญญาณอนาลอกทั้งนี้เนื่องจากระบบเดิมส่วนใหญ่เป็นแบบอนาลอก ดังนั้นระบบต้องสามารถเชื่อมต่อกับระบบเดิมได้ทำให้การประมวลผลสัญญาณซับซ้อนขึ้น

3. สัญญาณหรือข้อมูลเชิงเลข เป็นสัญญาณที่สร้างขึ้นมา ดังนั้นการส่งสัญญาณชนิดนี้ไปในตัวกลางตามธรรมชาติทั่วไป ซึ่งมักมีแถบความถี่ปฏิบัติงาน (Bandwidth) จำกัด และยังมีผลตอบสนองไม่เป็นเชิงเส้น (Non - Linear phase response) ทำให้สัญญาณเกิดความผิดเพี้ยนได้

4. เนื่องจากเราต้องการออกแบบระบบการประมวลผลสัญญาณเชิงเลขให้มีความคล่องตัวสูงสามารถใช้งานได้หลายรูปแบบดังนั้น ทำให้ระบบซับซ้อนมากซึ่งรวมทั้งด้านการซ่อมแซมบำรุงรักษา

5. ช่วงกว้างความถี่ปฏิบัติงานของระบบเชิงตัวเลข ต่ำกว่าของระบบประมวลผลเชิงอนาลอก ข้อจำกัดนี้เนื่องจากอุปกรณ์ที่ใช้ หรือ ประกอบขึ้นเป็นระบบการประมวลผลสัญญาณเชิงเลข เช่น วงจร D/A, วงจร A/D วงจรเกท เป็นต้น วงจรเหล่านี้ยังมีความเร็วต่ำ ทำให้การประมวลผลเชิงเลขมีความเร็วต่ำ

อย่างไรก็ตามเมื่อพิจารณาข้อดีและข้อเสียต่างๆแล้ว และถ้าเราต้องการระบบประมวลผลสัญญาณที่มีประสิทธิภาพและมีความแม่นยำสูง โดยไม่คำนึงถึงว่าราคาจะสูงเท่าใดแล้ว ระบบการประมวลผลสัญญาณก็ให้ประโยชน์ได้มาก

1.3 การประยุกต์โพรเซสสัญญาณดิจิตอล

งานการโพรเซสสัญญาณดิจิตอลจะสามารถแบ่งย่อยได้ 2 ส่วน คือ การกรองรวมทั้งวงจรกรองดิจิตอลและการวิเคราะห์สเปกตรัมของสัญญาณ ซึ่งทั้งสองส่วนนี้มีความสำคัญต่อกันอย่างใกล้ชิด และปกติในทางปฏิบัติจะต้องทำทั้งสองส่วนนี้ควบคู่กันไป ในปัจจุบันมีการนำการโพรเซสสัญญาณดิจิตอลไปประยุกต์ในงานต่าง ๆ มากมาย ส่วนหนึ่งของการประยุกต์ใช้งานแบ่งตามสาขาได้แก่

ก. การประยุกต์ในงานทั่ว ๆ ไป เช่น การกรองดิจิตอล (digital filtering), การหาคอนโวลูชั่น (convolution), การหาค่าสหสัมพันธ์ร่วม (corelation), การกรองแบบปรับตัวเอง (adaptive filtering), การกำเนิดรูปสัญญาณ (waveform generation)

ข. งานด้านกราฟิคหรือภาพ (graphic/image) เช่น งานด้านการส่งภาพ (image transmission) , การเข้ารหัสภาพ (image coding หรือ compression) , การเพิ่มคุณภาพของภาพ (image enhancement) , การจำ (pattern recognition)

ค. งานด้านการวัดและเครื่องมือวัด เช่น การวิเคราะห์และเครื่องวิเคราะห์สเปกตรัม (spectrum analysis) ,เครื่องกำเนิดสัญญาณรูปต่างๆ (function generation),การวิเคราะห์แผ่นดินไหว (seismic processing)

ง. ทางด้านเสียงพูด (voice/speech) เช่น การจำ (recognition) , การยืนยัน (verification) , การเพิ่มคุณภาพ (enhancement) , การสังเคราะห์ (synthesis) รวมถึงการเปลี่ยนข้อความ (text) เป็นเสียงพูด (speech)

จ. ทางด้านระบบควบคุม (control) เช่น ระบบควบคุมหุ่นยนต์ (robotic),การควบคุมเครื่องจักรกลไฟฟ้าต่าง ๆ ระบบเซอร์โว หรือพวกเฟสล็อคลูพ (PLL)

ฉ. ทางด้านทหาร (military) เช่น การสื่อสารที่เกี่ยวพันกับความมั่นคง (secure communication), ระบบเรดาร์โซน่า ,ระบบนำร่อง (navigation) , ระบบนำวิถีจรวด (missile guidance)

ช. ทางด้านระบบโทรคมนาคม (telecommunication) เช่น การกำจัดเสียงสะท้อน(echo cancellation), ระบบสื่อสารดิจิตอล โมเด็ม(modem) , อิควอไลเซอร์แบบปรับตัวเอง(adaptive equalizer) โทรศัพท์เคลื่อนที่ (cellular telephones)

ซ. ทางด้านการแพทย์(medical) เช่น เครื่องช่วยการได้ยิน เครื่องมือวินิจฉัยต่าง ๆ

ฌ. ทางด้านอุตสาหกรรม(industrial) เช่น เครื่องควบคุมเครื่องยนต์ (engine control),ระบบเสียงหรือโทรทัศน์ดิจิตอล(digital audio / TV),เครื่องมือสังเคราะห์เสียงดนตรี (music synthesizer) เครื่องเจาะหรือตัดแบบควบคุมด้วยตัวเลข (numerical control)

บทที่ 2

ทฤษฎีพื้นฐานของการประมวลผลสัญญาณเชิงเลข

2.1 นิยามของสัญญาณและระบบ

1. สัญญาณอนาลอก (Analog signal) : ใช้กับการกล่าวถึงสัญญาณที่มีรูปคลื่น (Wave form) เป็นไปอย่างต่อเนื่องกับพิสัยเวลา โดยที่แอมพลิจูดหรือค่าของสัญญาณก็มีการแปรไปอย่างต่อเนื่องด้วย เช่น สัญญาณไซน์

2. สัญญาณเชิงเวลาต่อเนื่อง (Continuous-time signal) สัญญาณแบบนี้จะแปรค่าไปอย่างต่อเนื่องกับพิสัยเวลา แต่แอมพลิจูดไม่ได้เจาะจงว่าต้องแปรไปอย่างต่อเนื่อง ซึ่งหมายถึงว่าอาจแปรอย่างไม่ต่อเนื่องก็ได้ จึงอาจกล่าวได้ว่าสัญญาณอนาลอกเป็นชนิดหนึ่งของสัญญาณเวลาต่อเนื่องได้

3. สัญญาณเชิงเวลาเต็มหน่วย (Discrete-time signal) เป็น สัญญาณ x(t) ที่ค่าของฟังก์ชันกำหนดเฉพาะเซตของเวลาที่แน่นอนอันหนึ่งเท่านั้น สัญญาณแบบนี้อาจแบ่งตามลักษณะของแอมพลิจูดได้เป็น 2 แบบ คือ

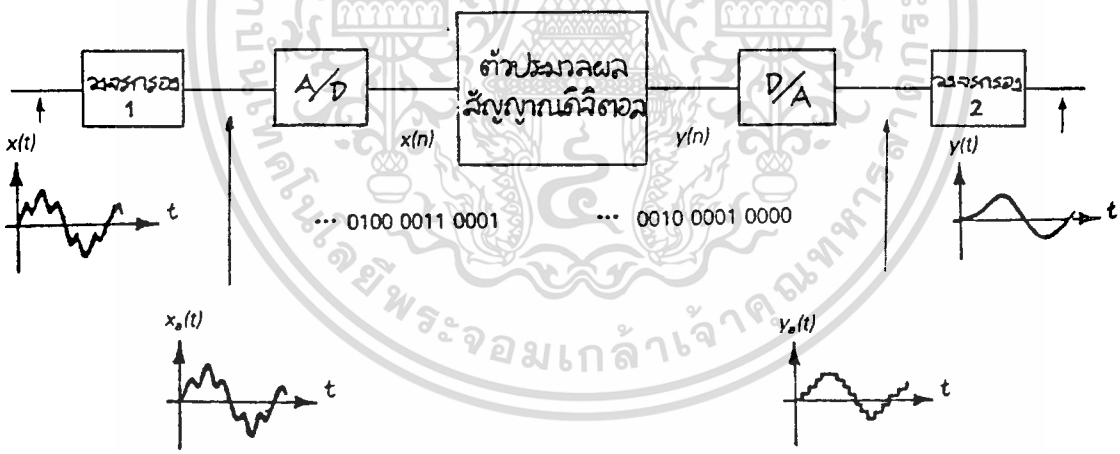
3.1 สัญญาณเชิงข้อมูลเต็มหน่วย (Discrete data signal) สัญญาณแบบนี้แอมพลิจูดจะต่อเนื่องหรือกล่าวได้ว่า แอมพลิจูลมีค่าเท่ากันทุกประการกับสัญญาณอนาลอกที่เป็นตัวต้นแบบในการสุ่มตัวอย่าง (Sampling)

3.2 สัญญาณเชิงเลข (Digital signal) สัญญาณแบบนี้แอมพลิจูดของสัญญาณมีค่าเฉพาะเซตของค่าที่แน่นอนเซตหนึ่งเท่านั้น เช่น สัญญาณที่ออกจากวงจร A/D เป็นต้น

4. ระบบเวลาจริง(Real - time signal) โดยทั่วไปคำนี้ใช้กับ ระบบการประมวลผลสัญญาณที่การคำนวน การประมวลผลทำได้เสร็จสิ้นก่อนที่จะมีการสุ่มตัวอย่างสัญญาณลำดับใหม่

5. ลำดับ (Sequence) หรือ ลำดับข้อมูล หรือลำดับสัญญาณ ซึ่งคำว่าลำดับอาจใช้แทนสัญญาณเชิงเต็มหน่วย หรือ สัญญาณเชิงเลขก็ได้

โดยทั่วไปนั้นรูปแบบของสัญญาณไฟฟ้าส่วนใหญ่ จะอยู่ในรูปของสัญญาณอนาลอก ซึ่งก็เช่นสัญญาณเสียง , คลื่นหัวใจ การนำเอาสัญญาณเหล่านี้ไปทำการประมวลผลในลักษณะการประมวลผลเชิงเลข จะต้องทำการเปลี่ยนรูปแบบของสัญญาณเสียก่อน (สาเหตุที่ทำการประมวลผลในรูปเชิงเลขนั้น เพราะการเก็บการประมวล การสื่อสารและการแสดงผล สามารถทำได้ง่ายและมีประสิทธิภาพมากกว่า) จากสัญญาณอนาลอกเปลี่ยนมาเป็นสัญญาณดิจิตอล นั้นจะทำได้โดยวงจร A/D (Analog to Digital Converter) และทำการประมวลผลโดยตัวประมวลผลทางดิจิตอล เช่น คอมพิวเตอร์ จากนั้นก็จะถูกนำมาแสดงผลเลย หรือ เปลี่ยนกลับมาให้อยู่ในรูปสัญญาณอนาลอกอย่างเดิมโดย วงจร D/A (Analog to Digital Converter) โดยที่สัญญาณก่อนที่จะเข้าวงจร A/D หรือ จาก D/A แล้วนั้น สัญญาณไฟฟ้าอาจจะถูกปรับให้อยู่ในรูปแบบและขนาดที่เหมาะสมก่อนโดย Analog Signal Conditioner เช่น วงจรขยาย หรือ วงจรกรองเป็นต้น ดังรูป



รูปที่ 2.1 : การประมวลผลสัญญาณดิจิตอล

2.2 ทฤษฎีการสุ่มตัวอย่าง (Sampling Theory)

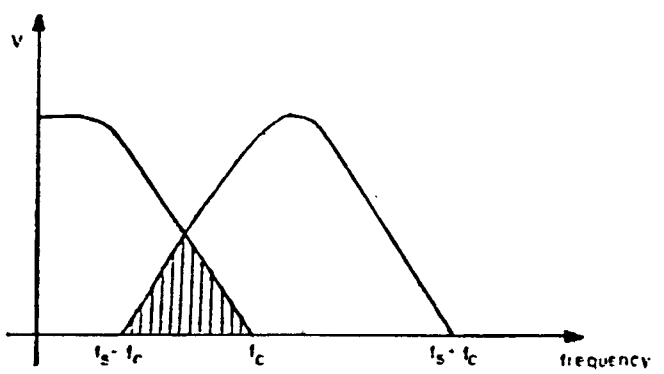
การแปลงสัญญาณอนาลอกมาเป็นสัญญาณเชิงเลข(สัญญาณดิจิตอล) นั้น ในเบื้องแรกต้องมีการสุ่มตัวอย่างก่อนซึ่งความถี่ของการสุ่มตัวอย่างโดยไม่ทำสูญเสียข้อมูลสำคัญไปนั้น ทฤษฎีการสุ่มตัวอย่างของ แชนนอน(Shannon) กล่าวไว้ว่า

ถ้าหากเรามีสัญญาณอนาลอก x(t) ที่ค่าการแปลงฟูเรียร์ หรือสเปกตรัมกำลัง (Power spectrum) ของมันมีความถี่ปฏิบัติงาน (band width) เท่ากับ fo แล้ว เราสามารถสุ่มตัวอย่าง โดยที่สัญญาณไม่สูญเสียเนื้อหาสำคัญ ก็ต่อเมื่อความถี่ในการสุ่มตัวอย่าง f_s มีค่ามากกว่าหรือเท่ากับสองเท่าของความถี่ f_o หรือ

$$f_s > 2f_o$$

โดยทั่วไปเอาอาจสุ่มตัวอย่างที่ความถี่ $f_{sn} = 2f_o$ (ซึ่งค่าความถี่นี้เรียกว่า ความถี่นีควิท(Nyqiust frequency) และ ช่วงเวลา $T_n = 1/2f_o$ นี้เรียกว่า ช่วงการสุ่มตัวอย่าง นีควิท (Nyqiust frequency)

ถ้าความถี่ของสัญญาณสุ่ม f_s ไม่มากกว่า $2f_o$ บางส่วนของ f_s จะมาซ้อนทับกับ สเปกตรัมของสัญญาณ ทำให้เกิดปรากฏการณ์ความผิดพลาดที่ เรียกว่า Frequency Folding เกิดขึ้น ซึ่งจะทำให้เกิดความผิดเพี้ยนแก่สัญญาณอนาลอกจากการซ้อนกันของสเปกตรัมเมื่อสัญญาณถูกเปลี่ยนกลับมาอยู่ในรูปเดิม

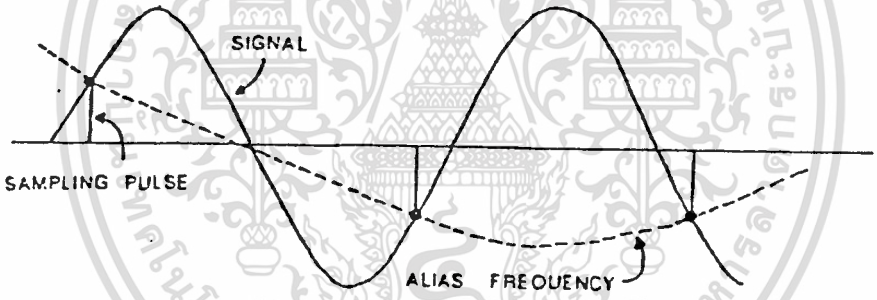


รูปที่ 2.2 : การเกิด frequency folding

ซึ่งวิธีการป้องกันการซ้อนกันของสเปกตรัมนี้ทำได้ โดยการใช้อัตรา ความถี่การสุ่มที่สูงพอ ($f_s > 2f_0$) และ การกรองความถี่สัญญาณอนาลอกก่อนการสุ่ม เพื่อมิให้แบนด์วิดของสัญญาณอนาลอกเกินไปกว่า $f_s/2$

แต่ในทางปฏิบัติแล้วจะยังคงเกิด Frequency Folding ได้เสมอจากส่วนฮาร์โมนิคของสัญญาณ รวมทั้งสเปกตรัมของสัญญาณรบกวนที่ยังคงอยู่แม้ว่าจะทำการกรองก่อนหน้ามาแล้ว ดังนั้นการทำการสุ่มสัญญาณควรพยายามทำให้การสุ่มสัญญาณเป็นไปอย่างเร็วมากที่สุด

ผลของการใช้อัตราการสุ่มที่ไม่เหมาะสมอีกประการหนึ่งคือ ความถี่ปลอม (Aliasing frequency) ซึ่งเกิดกับสัญญาณที่เปลี่ยนกลับมาเช่นเดิมหลังจากเกิดการสุ่มแล้ว



รูปที่ 2.3 : การเกิด Alias Frequency จากการสุ่ม
ด้วยความถี่ต่ำกว่า 2 เท่าของสัญญาณ

อนึ่งในการเลือกอัตราการสุ่มให้เหมาะสมตามความถี่สูงสุดของสัญญาณซึ่งจะต้องรู้สเปกตรัมของสัญญาณ ซึ่งต้องใช้การแปลงฟูเรียร์ของสัญญาณ ในการโพรเซสสัญญาณดิจิตอล เราจะใช้ DFT (หรือ FFT) ที่จะกล่าวต่อไป ในการประมาณค่าแปลงฟูเรียร์ของสัญญาณอนาลอก ซึ่งจะทำให้เกิดความผิดพลาดที่เกิดขึ้นระหว่างค่านวน กับ ค่าการแปลงที่ควรจะเป็นปรากฏการณ์นั้นก็คือ

1. ความถี่ปลอม (aliasing frequency) เป็น เช่นเดียวกับการสุ่มตัวอย่าง

2. การรั่ว(leakage) สัญญาณที่มีสเปกตรัมความถี่ที่เป็นแบนด์ลิมิต(bandlimit) เป็นสัญญาณที่มีค่าต่อเนื่องกันไปไม่สิ้นสุด ดังนั้นในการหาค่าที่ถูกต้องของการแปลงฟูเรียร์ จะต้องคำนวนโดยใช้สัญญาณทั้งหมดถึงอนันต์ ซึ่งไม่สามารถทำได้ จะทำได้เพียงช่วงหนึ่งสัญญาณ (ตัดส่วนที่นอกเหนือส่วนสนใจทิ้งไป) ดังนั้นสเปกตรัมความถี่ที่ได้จะมีการกระจายออก หรือเรียกว่าการรั่วของสเปกตรัม

3. ผลกระทบอันเนื่องมาจากการตกหาย(picket fence effect เนื่องจากการใช้ DFT จะทำให้ได้ค่าเป็นช่วงของสเปกตรัม เหมือนกับการมองผ่านรั้วไม้ระแนง (picket fence) ซึ่งส่วนของไม้จะปิดภาพส่วนหนึ่งไว้ จำนวนตัวอย่างที่นำมาคำนวน DFT จะเป็นตัวกำหนดความละเอียดของความถี่ระหว่างค่าของสเปกตรัม

2.3 การแปลงแซด (Z - Transform)

ในสาขาวิชาวิศวกรรมไฟฟ้า มีคณิตศาสตร์ที่เรียกว่า การแปลง หรือ ผลการแปลง (transform) อยู่หลายรูปแบบที่ใช้ช่วยในการวิเคราะห์หรือแก้ปัญหาโดยอ้อม โดยที่การหาคำตอบผ่านการแปลงเหล่านี้สามารถทำได้โดยสะดวกกว่าการแก้ปัญหาโดยตรง ในระบบการประมวลผลสัญญาณเชิงตัวเลข การแปลงที่สำคัญที่ใช้ในการหาคำตอบโดยทางอ้อมคือ การแปลงแซด โดยคุณสมบัติของมันใช้ในการเปลี่ยนแปลงจาก สมการผลต่างเชิงเส้น ไปเป็นสมการพีชคณิต ซึ่งทำให้สามารถวิเคราะห์ระบบได้อย่างมีประสิทธิภาพทั้งด้านการตอบสนอง และ คุณลักษณะ(characteristic) ของระบบได้โดยอาศัยกระบวนการทางคณิตศาสตร์ธรรมดา

ถ้าเรามีสัญญาณในโดเมนเวลา $x(0),x(1),\ldots,x(n)$ เราสามารถแทนสมการนี้

ด้วย $x(n) = \Sigma\, x(m)\delta(n-m)$ ซึ่งเป็นรูปของอัมฟลล์เชิงเลข (ที่อยู่ในพจน์ของอัมพัลส์เชิงเลขที่หน่วยออกไปได้) ถ้าหากเราทำการแปลงแซด อนุกรม $x(n)$ที่ได้ก็คือ

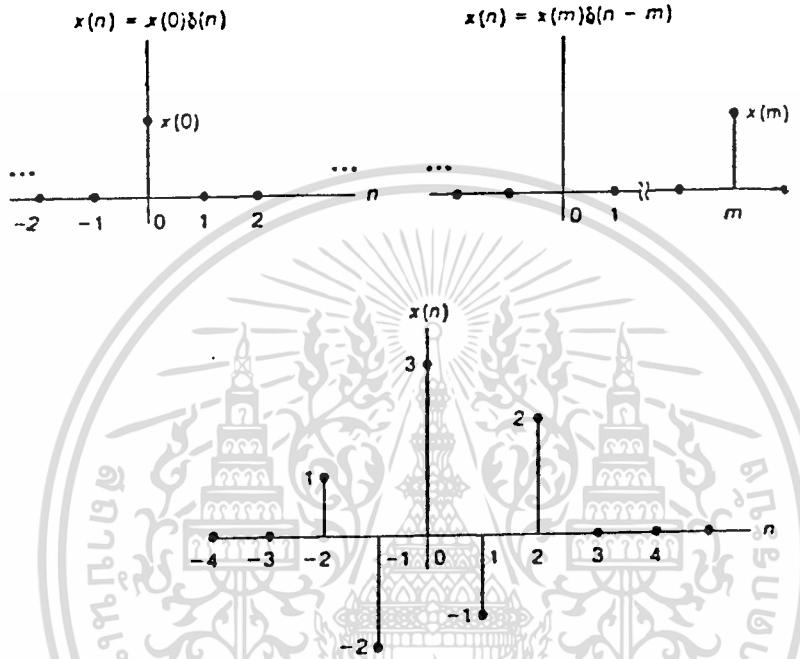
$$x(z) = \mathcal{Z}[x(n)]$$

$$= \sum_{m=-\infty}^{\infty} x(m)z^{-m}$$

เมื่อ Z เป็นจำนวนเชิงซ้อน เนื่องจากดรรชนี m ของการแปลงแซดแปรค่าจาก $-\infty$ ถึง ∞ ดังนั้นการแปลงแซดนี้จึงมีชื่อเรียกอีกอย่างว่าการแปลงแซดสองข้าง (two-side Z-transform)แต่ในทางปฏิบัติเรามักจะมีลำดับ $x(n)$ โดยที่ m มีค่ามากกว่าศูนย์ เพราะฉะนั้นจึงเรียกว่า ผลการแปลงแซดข้างเดียว (One-side transform) ซึ่งนิยมโดย

$$x(z) = \sum_{m=0}^{\infty} x(m)Z^{-m}$$

จากสมการข้างด้านนี้เราเห็นได้ว่า z เป็นการชี้ตัวตำแหน่งของแต่ละแซมเปิล หมายถึง ในสัญญาณที่เป็นแบบเวลาต่อเนื่องเมื่อมาพิจารณาในระบบเต็มหน่วย(Discrete) จะมีการสุ่มสัญญาณดังกล่าวเป็นช่วงๆ ทำให้เป็นอนุกรมแซมเปิลขึ้นมา และสัมประสิทธิ์ของ Z จะบอกถึงขนาดของแต่ละแซมเปิล



รูปที่ 2.4 : ตัวอย่างอิมพัลส์

$\delta(n)$ คือ อิมพัลส์เชิงเลขนิยามได้เป็น

$$\delta(n) = \begin{cases} 1 ; & n = 0 \\ 0 ; & n \neq 0 \end{cases}$$

$\delta(n-n_o)$ คืออิมพัลส์เชิงเลขที่หน่วงไป n_o วินาที นิยามได้เป็น

$$\delta(n-n_o) = \begin{cases} 1 ; & n = n_o \\ 0 ; & n = n_o \end{cases}$$

จากรูป (a) สามารถเขียนสมการได้ดังนี้ $x(n) = x(0)\delta(n)$ และจากสมการ (1)

การแปลงแซด คือ

$$x(z) = x(0)z^{o} = x(0)$$

จากรูป (b) มีการแซมเปิลที่ n=m จะได้ $x(n) = x(m)\delta(n-m)$; $m>o$ และการแปลงแซดคือ

$$x(z) = x(m)\, z^{-m}$$

ในรูป (c) เขียนอนุกรมแทนได้ดังนี้

$$x(n) = [\_\_\_,0,0,0,1,-2,3,-1,2,0,0,0\_\_\_,]$$

สามารถเขียนได้เป็น

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

ทำเป็นการแปลงแซดได้คือ

$$x(z) = \sum_{m=-\infty}^{\infty} [\delta(n+2)-2\delta(n+1)+3\delta(n)-\delta(n-1)+2\delta(n-2)]z^{-m}$$

$$x(z) = z^{2}-2z^{1}+3z^{0}-z^{-1}+2z^{-1}$$

สังเกตดูจะเห็นว่า z^{-1} แทนการหน่วงของแซมเปิลหนึ่ง ๆ และ z แทนการเลื่อนของแซมเปิลนั้นๆ ทำให้ได้สมการที่เกี่ยวกับการเลื่อนน้ำหนักข้อมูล (Shifted Weighted Sample) คือ

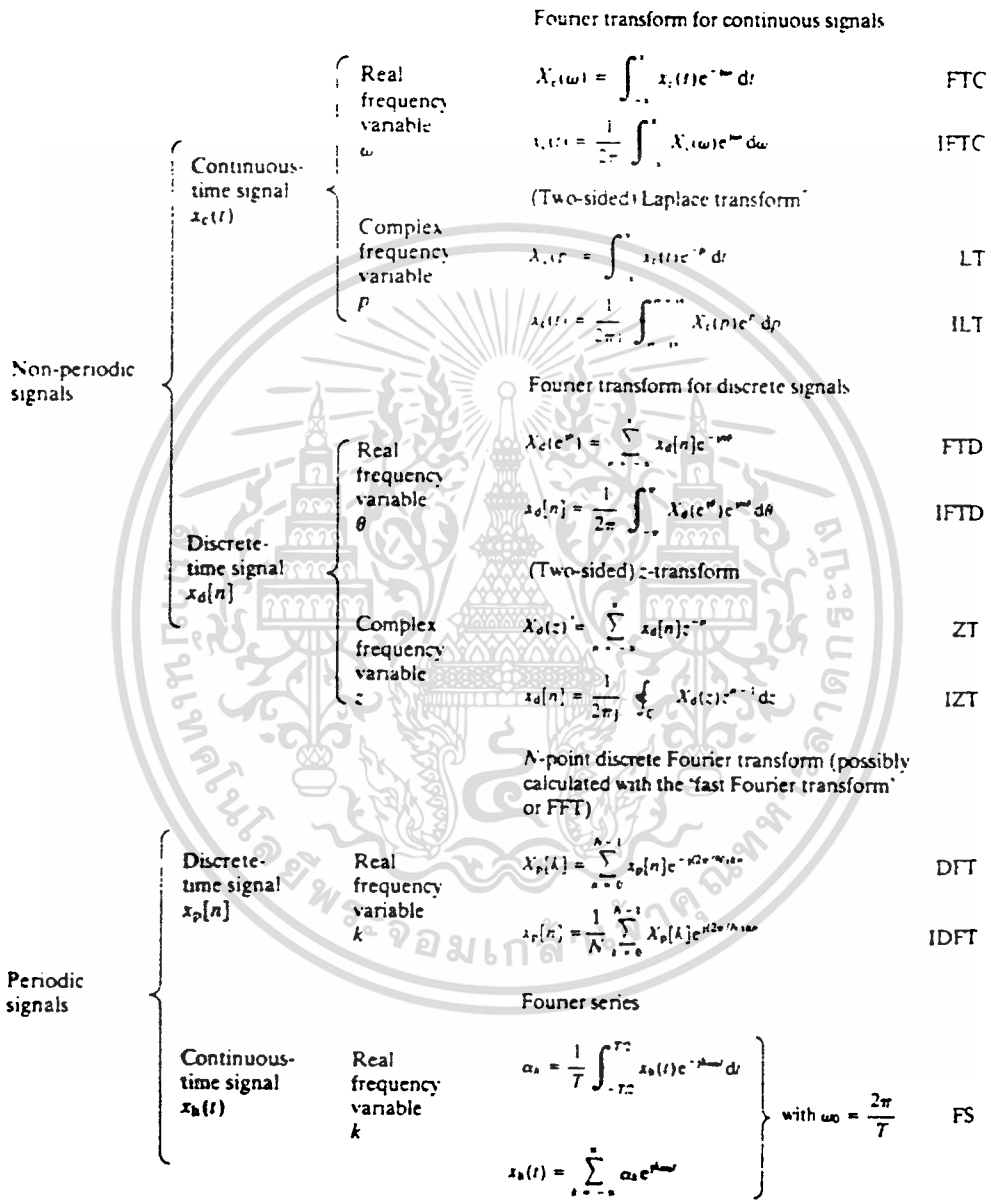
$$\mathcal{Z}[A\delta(n-m)] = Az^{-m}$$

โดยที่ A แทนขนาดหรือน้ำหนักของแซมเปิลนั้น ๆ สามารถสรุปได้เกี่ยวกับการแปลงแซดได้ดังนี้

1. x(z) เป็นโพลิโนเมียล หรือ เป็นล่าดับกำลังในรูปของ z และหาได้จากอนุกรม x(n)

2. ในระบบการแซมเปิล x(z) จะมีรูปแบบเป็นอิสระในการแซมเปิลในคาบเวลา T แต่แฟคเตอร์ z^{-m} จะมีความสัมพันธ์กับ x(n) โดย t = nT ซึ่งโดยลักษณะนี้ z^{-n} จะหมายถึงการหน่วง nT วินาทีจากเวลา t = 0

รูปที่ 2.5 : **ความสัมพันธ์ของการทรานสฟอร์มแบบต่างๆของสัญญาณ**



2.4 สมการผลต่างสืบเนื่อง (Difference Equation)

ในทำนองเดียวกันกับการวิเคราะห์ระบบเชิงอนาลอก ที่คุณสมบัติของระบบในโดเมนเวลา สามารถเขียนอธิบายได้โดยใช้สมการเชิงอนาลอก ส่วนในระบบเชิงเลขเราก็มีสมการผลต่างสืบเนื่อง ใช้ในการอธิบายคุณสมบัติของระบบในโดเมนเวลาซึ่งสมการผลต่างสืบเนื่องอันดับ n อาจเขียนได้เป็น

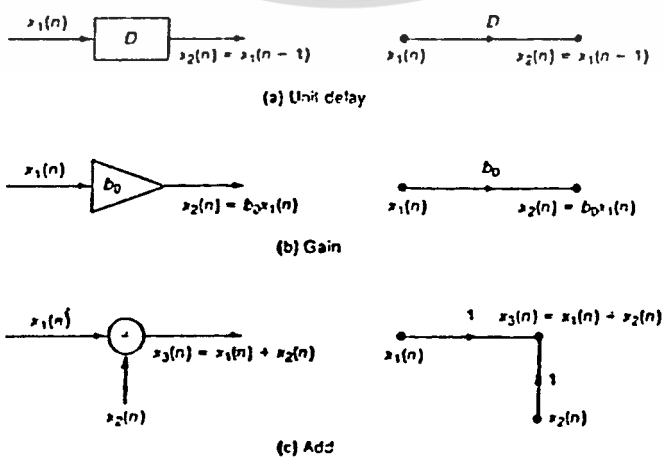
$$y(n) = \sum_{i=0}^{r} a_i x_{(n-i)} - \sum_{i=1}^{m} b_i y_{(n-i)}$$

โดยที่ x(n) เป็นลำดับสัญญาณเข้า y(n) เป็นลำดับสัญญาณออก และ a_i, b_i เป็นค่าสัมประสิทธิ์

เพื่อประโยชน์ในการอธิบาย เราจะพิจารณาในกรณีของสมการผลต่างสืบเนื่องอันดับที่หนึ่ง(1st order difference equation)ซึ่งเขียนได้ คือ

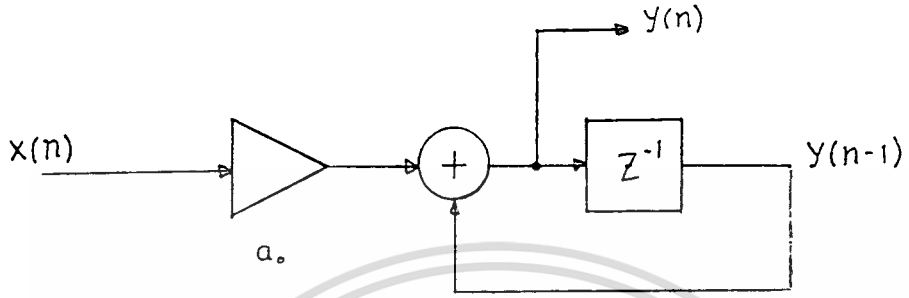
$$y(n) = a_0 x(n) - b_1 y(n-1)$$

ซึ่งสมการนี้เป็นการกรองแบบง่ายที่สุดเห็นได้ว่าถ้าต้องการสร้างตัวประมวลผลสัญญาณนี้ต้องมีอุปกรณ์ 3 ชนิด คือ ตัวคูณเลข (Multiplier) ตัวหน่วงลำดับสัญญาณ (Delay) หรือ ชิฟรีจีสเตอร์ และตัวรวมสัญญาณ (Summer) ซึ่งทั้ง 3 ตัวเขียนสัญญาลักษณ์ได้ตามตัวรูป



รูปที่ 2.6 : สัญญาลักษณ์ของอุปกรณ์ต่างๆ

ดังนั้นจากสมการ(2) เราจะได้แผนภาพของวงจรประมวลผลสัญญาณของสมการผลต่างสืบเนื่องอันดับที่หนึ่ง ดังรูป



รูปที่ 2.7 : แผนภาพแทนสมการผลต่างสืบเนื่องอันดับที่หนึ่ง

การทำวงจรกรองโดยโดยใช้ฮาร์ดแวร์นั้น โครงสร้างฮาร์ดแวร์ของการโพรเซสสัญญาณดิจิตอล จะต้องมีลักษณะพิเศษที่ออกแบบมาเพื่อให้สามารถคำนวนการคูณและการบวกสะสมค่าได้อย่างรวดเร็ว ส่วนอินเตอร์เฟสอินพุทและเอาพุทปกติจะประกอบด้วย วงจรแปลง A/D วงจรแปลง D/A และบัฟเฟอร์ระหว่างสัญญาณอินพุทและเอาท์พุททั้งอนาลอกและดิจิตอล ส่วนความจำ RAM จะเก็บข้อมูลเป็นสองส่วน คือ ส่วนค่าในอดีตและปัจจุบันของอินพุทและส่วนที่เหลือเก็บค่าในอดีตและปัจจุบันของเอาท์พุท ค่าสัมประสิทธิ์ (ค่า a และ b) อาจจะเก็บอยู่ใน ROM หรือ ถ้าหากเป็นข้อมูลที่อแคฟทีฟ ปกติจะเก็บไว้ใน RAM ส่วน multiplier/accumulator (MAC) จะทำการรับค่าอินพุท 2 ค่า เช่น B และ C ทำการคูณค่าทั้งสองทำด้วยกัน หลังบวกผลคูณที่ได้เข้ากับ A (ค่าใน accumulator) หลังค่าผลลัพธ์ จะเก็บไว้ในตัวสะสม (เป็นค่าใหม่) จะเห็นว่าค่า B และ C จะเปลี่ยนไปทุกครั้งของการคูณ ดังนั้นเป็นหน้าที่ของตัวควบคุม (controller) ในการที่ส่งอินพุทที่แตกต่างกันนี้ไปให้ MAC ได้อย่างถูกต้องตามลำดับตัวควบคุมดังประกอบด้วย โปรแกมสำหรับวงจรกรอง ตัวจัดลำดับสำหรับโปรแกรม และหน่วยควบคุมที่จัดหาสัญญาณต่าง ๆ ในการใช้ทำงานได้จริง

การประมวลผลสัญญาณดิจิตอลอาจแบ่งเป็นได้ 2 ส่วนที่มีความสัมพันธ์ต่อกัน คือ

1. การกรองดิจิตอล
2. การวิเคราะห์สเปกตรัม

2.5 การกรองดิจิตอล (Digital Filtering)

จะเกี่ยวข้องกับทั้งการวิเคราะห์ (analysis) และการสังเคราะห์ (synthesis) โดยที่การวิเคราะห์จะเกี่ยวข้องกับการนำความรู้เกี่ยวกับทฤษฎีระบบเป็นช่วง มาศึกษาคุณสมบัติพฤติกรรม หรือ ธรรมชาติของวงจรกรองดิจิตอล ซึ่งถือว่าเป็นระบบเชิงเส้น และไม่แปรตามเวลา (linear time-invarient) การวิเคราะห์ ทำให้ได้ทรานสเฟอร์ฟังก์ชัน ซึ่งเป็นตัวกำหนด หรือ สามารถบ่งบอกคุณสมบัติจำเพาะ(characteristic) ของระบบ ส่วนการสังเคราะห์จะเกี่ยวข้องกับการออกแบบวงจรกรองแบบต่างๆให้มีคุณสมบัติในโดเมนเวลา หรือ โดเมนความถี่ที่ต้องการ เราสามารถแทนระบบเชิงเส้น และไม่แปรตามเวลา ด้วยสมการผลต่างสืบเนื่อง (Difference Equation)

$$y(n) = \sum_{i=0}^{k} a_i x_{(n-i)} - \sum_{i=1}^{k} b_i y_{(n-i)}$$

เมื่อ x(n) และ y(n) คือ อินพุทและเอาท์พุทของระบบตามลำดับ สามารถหาทรานสเฟอร์ฟังก์ชัน H(z) ของระบบโดยการใช้การแปลงของ z ได้

$$H(z) = \left(\sum_{i=0}^{k} a_i z^{-i}\right) \Big/ \left(1 + \sum_{i=1}^{k} b_i z^{-i}\right)$$

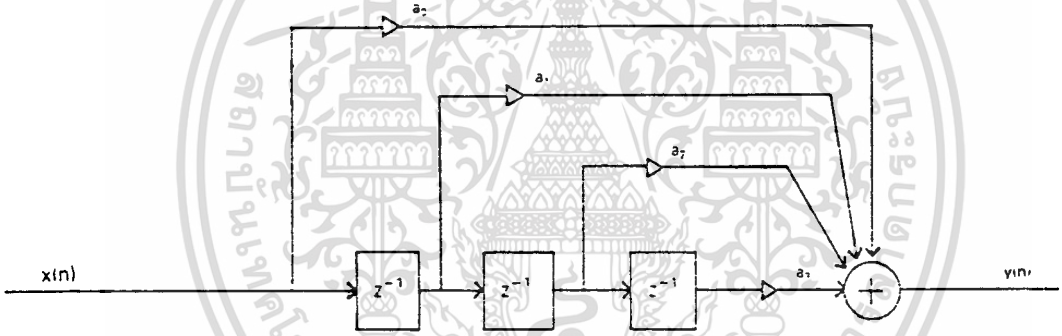
ซึ่งปัญหาการออกแบบวงจรดิจิตอลก็คือ การหาค่าสัมประสิทธิ์ a_i และ b_i ที่ทำให้วงจรกรองมีพฤติกรรมตามต้องการ เราสามารถแบ่งวงจรกรองตามการแบ่งชนิดของการทำให้เป็นจริง (ซึ่งหมายถึงการแทนวงจรกรองในรูปที่เราสามารถคำนวนหาการตอบสนองวงจรกรองโดยตรง) จะแบ่งย่อยเป็น

การทำแบบป้อนกลับเชิงเลข หรือ การทำแบบรีเคอร์ชีฟ (recursive realization) ในแบบนี้ค่าเอาท์พุทที่คำนวนได้ในปัจจุบันจะขึ้นกับอินพุททั้งในอดีตและปัจจุบัน และขึ้นกับเอาท์พุทในอดีต ดังรูป

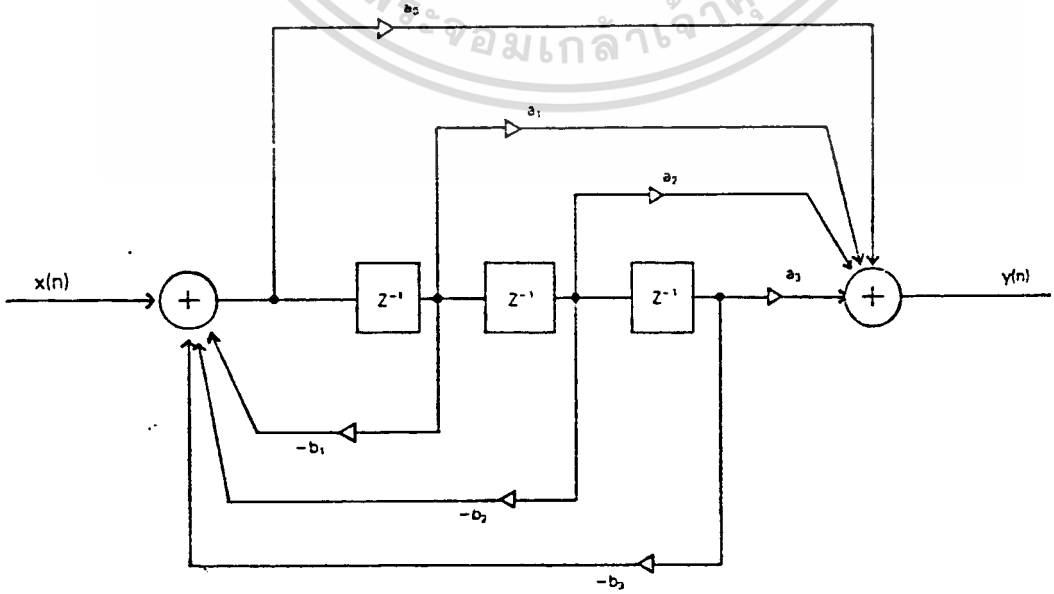
แต่ถ้าแบ่งตามช่วงเวลาที่ระบบตอบสนองอินพุท เมื่ออินพุทเป็นอิมพัลส์ สามารถแบ่งย่อยเป็น

- การตอบสนองอิมพัลส์ถึงอนันต์ IIR (Infinite Impulse Response) วงจรกรองแบบนี้ จะมีการตอบสนองอิมพัลส์ h(n) ที่ประกอบด้วยตัวอย่าง (sample) จำนวนอนันต์

- การตอบสนองอิมพัลส์ที่จำกัด FIR (Finite Impulse Response) วงจรกรองแบบนี้จะมีการตอบสนองอิมฟัลส์ ที่จำกัดเพียงจำนวนตัวอย่างที่แน่นอนค่าหนึ่ง



รูปที่ 2.8 : การทำแบบไม่ป้อนกลับ



รูปที่ 2.9 : การทำแบบป้อนกลับ

ข้อเปรียบเทียบของตัวกรองแบบไม่ป้อนกลับเชิงเลข กับตัวกรองป้อนกลับเชิงเลข

ตัวกรองแบบป้อนกลับ (แบบรีเคอร์ซีฟ)	ตัวกรองแบบไม่ป้อนกลับ (แบบไม่เป็นรีเคอร์ซีฟ)
1. อาจไม่เสถียรภาพได้	1. มีเสถียรภาพเสมอ
2. ให้ช่วงความถี่ที่ผลตอบสนองความถี่จากผ่านเป็นไม่ผ่านสัญญาณ หรือทรานชิสชันแบนด์ (transistion band) แคบกว่า	2. ให้ช่วงทรานชิสชันแบนด์กว้างกว่า
3. ค่าการหน่วงสัญญาณ ที่จุดสัญญาณออก (overall delay) น้อยกว่าจึงเหมาะสำหรับใช้กับระบบเวลาจริง	3. ค่าการหน่วงออกไปของสัญญาณที่จุดสัญญาณออกมากกว่ามาก
4. เกิดผลของความผิดเพี้ยนของการตอบสนองเฟส(phase distortion)สูง การสร้างวงจรกรองแบบนี้ให้มีผลตอบสนองเฟสเชิงเส้นทำได้ แต่ไม่สามารถใช้กับระบบประมวลผลสัญญาณที่เป็นระบบเวลาจริงได้	4. สามารถออกแบบให้มีผลตอบสนองเฟสเป็นแบบเชิงเส้นได้
5. ผลของสัญญาณรบกวน เนื่องจากการปัดเศษ (round-off error) มีค่ามากเกิดจากเพราะมีการป้อนกลับ (feedback)	5. สัญญาณรบกวนจากการปัดเศษมีค่าน้อย
6. การออกแบบง่าย โดยเฉพาะการออกแบบโดยใช้วิธีการแปลงมาจากตัวกรองอนาลอก	6. การออกแบบให้มีผลตอบสนองดีทำได้ยากกว่า

2.6 การตอบสนองสัญญาณตัวอย่างหนึ่งหน่วย (Unit Sample Response)

การตอบสนองสัญญาณตัวอย่างหนึ่งหน่วย เป็นอนุกรมของเอาท์พุทที่ผลิตออกมา โดยอาศัยการป้อน $x(n) = z(n)$ ให้กับระบบเก่าเรารู้การตอบสนองของระบบที่เกิดขึ้นนี้ได้ ก็จะสามารถรู้ผลตอบสนองอันเนื่องมาจากการป้อนล่าดับอินพุทใด ๆ ก็ได้ วิธีนี้มักถูกใช้เป็นพื้นฐานในการประมวลผลสัญญาณเชิงเลข

พิจารณาระบบไม่เป็นรีเคอร์ซีฟ ดังต่อไปนี้

$$y(n) = \sum_{k=0}^{m} b_k x_{(n-k)} \qquad - (1)$$

และกำหนดสัญญาณตัวอย่างอินพุทหนึ่งหน่วย ดังนี้

$$x(n) = \delta(n) = 1 \quad \text{เมื่อ} \quad n=0 \qquad - (2)$$
$$= 0 \quad \text{เมื่อ} \quad n>0, n<0$$

เมื่อทำการป้อนอินพุทนี้เข้าไปยังระบบ ผลตอบสนองที่ได้จะเรียกว่า การตอบสนองสัญญาณหนึ่งหน่วย $h(n)$ นั่น คือ

$$y(n) = h(n)$$

จากสมการข้างต้นจะได้

$$y(0) = h(0)$$
$$= b_0x(0) + b_1x(-1) + \ldots + b_mx(-m)$$
$$= b_0$$

$$y(1) = h(1)$$
$$= b_0x(1) + b_1x(0) + \ldots + b_mx(1-m)$$
$$= b_1$$

$$y(m) = h(m)$$
$$= b_0x(m) + b_1x(m-1) + \ldots + b_mx(0)$$
$$= b_m$$

ทำให้เราสามารถสรุปได้ว่า

$$y(n) = h(n) = b(n) \quad \text{เมื่อ } n=0,1,2,\ldots,m$$

และ $$y(n) = 0 \quad \text{เมื่อ } n<0$$

เขียนสมการทั่วไปของการตอบสนองตัวอย่างหนึ่งหน่วย ได้ คือ

$$h(n) = \sum_{k=0}^{m} b_k\delta(n-k)$$

$$= \sum_{k=0}^{m} h(k)\delta(n-k) \qquad - (3)$$

หรือเขียนในอนุกรมได้ดังนี้

$$[h(h)] = [b_0, b_1, b_2, \ldots, b_n]$$

ระบบที่เป็นไปตามสมการที่ เรียกว่า ระบบการตอบสนองอิมพัลส์ที่จำกัด (Finite Impulse Response (FIR)System) เนื่องจาก h(n) มีเทอมจำกัดที่แน่นอน คือ ตั้งแต่ 0 ถึง m ดังสมการ (3) โดยที่ ที่ n>m+1 ค่าของ การตอบสนองสัญญาณตัวอย่างหนึ่งหน่วย จะถือเสมือนว่าเป็นศูนย์ จึงเป็นไปได้เสมอที่จะแทนระบบ FIR ด้วยสมการความแตกต่างในรูปแบบของสมการที่ (1)

แต่ถ้าผลการตอบสนองของระบบที่มีต่อสัญญาณตัวอย่างหนึ่งหน่วยไม่มีค่าใดเข้าใกล้ศูนย์เลยระบบดังกล่าวจะเรียกว่าระบบตอบสนองอัมพัลส์ถึงอนันต์ (Infinite Impulse Response (IIR) System)

คุณสมบัติโดยทั่วไปของวงจรกรองดิจิตอล แบบ FIR และ IIR

1. ลักษณะของวงจรแบบ FIR โดยทั่วไปเป็นแบบไม่ป้อนกลับ ส่วนลักษณะของวงจรแบบ IIR โดยทั่วไปเป็นแบบที่มีการป้อนกลับ
2. วงจรกรองแบบIIR สามารถออกแบบให้มีคุณสมบัติของเฟส แบบเชิงเส้นได้ แต่ทำได้ยากใน FIR
3. ทรานสเฟอร์ฟังก์ชันของ FIR จะมี POLE อยู่ที่จุดกำเนิดและ มีเสถียรภาพเสมอ
4. ค่าผิดพลาดเนื่องจากการปัดเศษ และความคลาดคลื่อนของสัมประสิทธิ์ FIR มีค่าน้อยกว่า IIR
5. ในการประมาณค่าพารามิเตอร์ต่างๆ สำหรับวงจรแบบ FIR จะมีปัญหา มากกว่าแบบ IIR จึงต้องใช้คอมพิวเตอร์มาช่วย ในการหาค่าพารามิเตอร์ต่างๆ
6. ช่วงเวลาในการหน่วงของ FIR จะเพิ่มขึ้นตามจำนวนพจน์ และจะมีค่า มากที่วงจรที่มีอันดับสูง

2.7 การวิเคราะห์สเปกตรัม (Spectral Analysis)

2.7.1 การแปลงฟูเรียร์ (Fourier Transform)

ถ้าหากเรานิยามให้ x(t) มีค่าเท่ากับฟังก์ชันบนช่วง $(-T_0, T_0)$ และนิยามให้มันมีคาบ $2T_0$ ณ จุดอื่นๆ จากนั้นเขียนแทนฟังก์ชันนี้ด้วยอนุกรมฟูเรียร์แล้วทำการพิจารณาให้ T_0 เข้าสู่อนันต์ เมื่อ T_0 มากจะทำให้เครื่องหมายผลบวกรวบยอด (summation sign) ก็จะกลายเป็นค่าอินทิกรัล ดังนั้น

$$x(t) = (1/2\pi) \int_{-\infty}^{\infty} x(w)\exp(jwt)\, dw \qquad - (1)$$

และ

$$x(w) = \int_{-\infty}^{\infty} x(t)\exp(-jwt)\, dt \qquad - (2)$$

สมการ (2) เรียกเป็น ฟูเรียร์อินทิกรัล (Fourier integral) ส่วนสมการ (1) เป็นการแปลงฟูเรียร์ผกผัน เงื่อนไขเพียงพอสำหรับ x(t) ที่จะมี x(w) ได้นั้นคือ

$$\int_{-\infty}^{\infty} |x(t)|\, dt < \infty$$

และ x(t) ยังต้องสอดคล้องตามเงื่อนไขที่ว่า x(t) ต้องมีการแปรผันแบบมีขอบเขต นั่นคือต้องมีจำนวนของค่าสูงสุดและต่ำที่สุดเฉพาะที่ และจำนวนจุดที่ไม่ต่อเนื่องที่แน่นอนในช่วงขอบเขต ส่วนการแปลงผกผันจะสู่เข้าหา x(t) แทนจุดทุกจุดที่มีความต่อเนื่อง และสู่เข้าหาจุดกึ่งกลาง ณ จุดที่ไม่ต่อเนื่องสำหรับค่าของการแปลงระหว่าง x(w) อาจเขียนแทนได้เป็น

$$x(t) \quad <===> \quad x(w)$$

หรือ $x(w) = \mathbf{F}[x(t)]$ และ $x(t) = \mathbf{F}^{-1}[x(w)]$

โดยที่ $\mathbf{F}$ แทนการแปลงฟูเรียร์ และ $\mathbf{F}^{-1}$ แทนการแปลงฟูเรียร์ผกผัน

2.7.2 อนุกรมฟูเรียร์และการแปลงฟูเรียร์เต็มหน่วย

(1) อนุกรมฟูเรียร์ (Fourier Series)

ถ้าให้ x(t) เป็นสัญญาณต่อเนื่องในโดเมนเวลา โดยทั่วไปสัญญาณนี้อาจเป็นปริมาณของ ศักดาไฟฟ้า กระแสไฟฟ้า หรือ ประจุไฟฟ้า ถ้า x(t) มีคุณสมบัติเป็นฟังก์ชันเป็นคาบ (Periodic function) ที่มีคาบเป็น T_o กล่าวคือ

$x(t+T_o) = x(t)$ ต่อทุกค่าของ t เราสามารถกระจายหรือเขียนแทนฟังก์ชันเป็นคาบนี้ได้ด้วย อนุกรมคริโกณมิติ คือ

$$x(t) = \sum_{n=-\infty}^{\infty} C_n \exp(j2n\pi t/T_o) \qquad - (1)$$

$$= C_o + \sum_{n=1}^{\infty} [C_n \exp(j2n\pi t/T_o) + C_{-n} \exp(-j2n\pi t/T_o)]$$

โดยที่ n เป็นจำนวนเต็ม

$$C_n = (1/T_o) \int_{o}^{T_o} x(t) \exp(-j2n\pi t/T_o)\, dt \qquad - (2)$$

โดย C_n เป็นจำนวนเชิงซ้อน และมีชื่อเรียกว่าเป็นส่วนประกอบสเปกตรัม (Spectral component) และอนุกรม (1) นี้เรียกว่าเป็นอนุกรมฟูเรียร์ ของฟังก์ชัน x(t)

สังเกตุเห็นว่า สองสมการนี้มีความสัมพันธ์กันในลักษณะเป็นส่วนกลับกันและกัน กล่าวคือ สมการ (1) ให้ฟังก์ชันx(t) ในพจน์ของสัมประสิทธิ์ C_n และสมการ (2) ให้สัมประสิทธิ์ C_n ในพจน์ของฟังก์ชัน x(t) แนวความคิดพื้นฐานนี้สำคัญมาก ในการวิเคราะห์ระบบ วงจรอิเลกทรอนิคส์ เพราะบอกให้เรารู้ว่า สัญญาณในโดเมนเวลา x(t) สามารถแปลงหรือ กระจายให้อยู่ในรูปของ สัญญาณในโดเมนความถี่ โดยจะประกอบด้วย สัญญาณที่มี

ความถี่เป็นค่า พหุนาม (Multiples) และความถี่หลักมูล (Fundamental frequency)จำนวนอนันต์ความถี่มารวมกัน สำหรับที่ n>2 เรียกว่าเป็นความถี่ฮารัมอนิก (Harmonic frequency)

อนุกรมฟูเรียร์สามารถเขียนในรูปตรีโกณมิติ โดยเขียนให้ $a_n = C_n + C_{-n}$ และ $b_n = j(C_n - C_{-n})$ ดังนั้น จะได้อนุกรมฟูเรียร์เป็น

$$x(t) = (a_o/2) + \sum_{n=1}^{\infty} [a_n \cos(2n\pi/T_o) + b_n \sin(2n\pi/T_o)]$$

และค่าสัมประสิทธิ์หาได้จากการอินทกรัล

$$a_n = (2/T_o) \int_o^{T_o} x(t)\cos(2n\pi/T_o)\, dt$$

$$b_n = (2/T_o) \int_o^{T_o} x(t)\sin(2n\pi/T_o)\, dt$$

<u>ข้อสังเกตุ</u> ในทางปฏิบัติ เราเขียนอนุกรมในสมการ (1) ด้วยจำนวนพจน์คงตัวหรือจำกัด $n<N$ อนุกรมฟังก์ชันที่มีจำนวนพจน์คงตัว (Finite Fourier series) นี้จะเป็นค่าโดยประมาณของ x(t) และเป็นการประมาณแบบค่าผิดพลาดกำลังสองเฉลี่ยน้อยที่สุด ของฟังก์ชัน x(t)

(2) <u>การแปลงฟูเรียร์เต็มหน่วย</u> (Discrete Fourier Transform)

เนื่องจากผลการแปลงฟูเรียร์ เป็นเครื่องมือทางคณิตศาสตร์ที่มีประโยชน์มากในการวิเคราะห์ และศึกษาระบบทางอิเลกทรอนิคส์ การมีวิธีคำนวนการแปลงที่มีประสิทธิภาพ และเหมาะสมที่ใช้ในการคำนวนจึงเป็นสิ่งจำเป็น แต่ในการแปลงค่าต่างๆจะเป็นอินทิกรัล ซึ่งเวลาการคำนวนโดยใช้

คอมพิวเตอร์ อาจต้องใช้เวลานาน หรืออาจหาค่าตอบไม่ได้ ถ้าหากเรามีการแปลงที่เหมาะสม และข้อมูลหรือสัญญาณที่รับเข้ามามีความยาวจำกัด ใช้อนุกรมจำกัดที่มีจำนวนพจน์มากพอ ค่าใกล้เคียงกับการแปลงเดิม และเวลาในการคำนวนเร็ว การแปลงใหม่นี้จะเป็นประโยชน์อย่างมากในการประมวลสัญญาณเชิงเลข การแปลงที่จะกล่าวถึงในบทนี้มีชื่อเรียกว่า
การแปลงฟูเรียร์เต็มหน่วย

ถ้าให้ $x_p(t)$ เป็นฟังก์ชันต่อเนื่องและเป็นคาบ (Continuous periodic function) ที่มีคาบเป็น T_p หรือ

$$x_p(t) = x_p(t+mT_p) \qquad - (1)$$

สามารถเขียนแทนได้ด้วยอนุกรมฟูเรียร์ได้ดังนี้

$$x_p(t) = \sum_{k=-\infty}^{\infty} x_p(k)\exp(j2\pi kf_o t) \qquad - (2)$$

โดยที่ $f_o = 1/T_p$ ซึ่งเป็นค่าความถี่หลักมูล หาสัมประสิทธิ์ $x(k)$ ได้จาก

$$x(k) = (1/Tp) \int_{o}^{Tp} x_p(t)\exp(-j2\pi kf_o t)\, dt \qquad - (3)$$

ข้อสังเกตุก็คือ $x_p(t)$ เป็นฟังก์ชันแบบต่อเนื่อง และสเปกตรัมของมันมีค่าเป็นฟังก์ชันเต็มหน่วย (Discrete value) นำสัญญาณ $x_p(t)$ ที่อยู่ในช่วง $0<t<T_p$ มาทำการสุ่ม (ค่าความถี่การสุ่มอย่างน้อยที่สุดต้องเป็น 2 เท่าของความถี่ปฏิบัติงาน) เมื่อถูกสุ่มออกเป็น n สามารถเขียนสมการแทนได้เป็น

$$x(t/T) = \sum_{n=0}^{N-1} x_p(t/T)\delta(t/T - n) \qquad - (4)$$

โดยที่ t/T เป็นเวลานอร์มอลไรซ์ (Nomalized time) หรือ $0 < t/T < n$ ว่าสามารถหาสัมประสิทธิ์ฟูเรียร์

$$x'(k) = (1/N) \sum_{n=0}^{N-1} x_p(n)\exp(-j2\pi kn/N) \quad - (5)$$

และโดยความจริงแล้ว ณ จุดที่ทำการสุ่มสัญญาณ สัญญาณที่สุ่มมา $x(n)$ มีค่าเท่ากับ $x_p(n)$ และโดยทั่วแล้วจะนิยามให้เป็น

$$x(k) = \sum_{n=0}^{N-1} x(n)\exp(-j2\pi kn/N) \quad - (6)$$

สมการนี้มีชื่อเรียกว่า การแปลงฟูเรียร์เต็มหน่วย (Discrete Fourier Transform : DFT) ของ สัญญาณ $x(n)$ สำหรับ การแปลงฟูเรียร์เต็มหน่วยผกผัน (Inverse Discrete Fourier Transform : IDFT) สามารถได้จากสมการที่ (2) โดยการแทนให้ $t = nT$ และ $x_p(t) = x(n)$ และจากสมการ (6) จะได้ว่า

$$x(k) = (1/N) \sum_{k=0}^{N-1} x(k)\exp(-j2\pi kn/N) \quad - (7)$$

กล่าวโดยสรุปคือ จากการที่ทำการสุ่มตัวอย่างฟังก์ชันเป็นคาบและต่อเนื่อง $x_p(t)$ ออกเป็นฟังก์ชันเต็มหน่วยจำนวน N ลำดับด้วยกัน สเปกตรัมของลำดับเต็มหน่วยนี้หาได้โดยสมการ (5) และเมื่อทำการเปรียบเทียบสเปกตรัมของลำดับเต็มหน่วยกับสเปกตรัมของฟังก์ชันต่อเนื่องตัวต้นแบบ ในช่วง $o<k<n-1$ แล้วปรากฏว่าเท่ากันทุกประการ ต่างไปแต่ว่าลำดับเต็มหน่วยมีคุณสมบัติเป็นคาบด้วย และการคำนวนสเปกตรัมโดยใช้สมการ (5) เหมาะเป็นอย่างยิ่งที่ใช้คำนวนโดยคอมพิวเตอร์ เนื่องจากเวลาคำนวนมีค่าแน่นอน

2.8 การแปลงฟาสท์ฟูเรียร์ (Fast Fourier Transform หรือ FFT)

เนื่องจากอุปกรณ์ทางคอมพิวเตอร์ในปัจจุบันมีราคาถูก และ มีประสิทธิภาพสูง อีกทั้งความแม่นย่า และความแน่นอนจากการคำนวนโดยการใช้คอมพิวเตอร์มีมาก จึงทำให้การวิจัยและ การพัฒนาของการประมวลผลสัญญาณเชิงเลขเพื่อนำไปประยุกต์ใช้กับงานในสาขาวิชาต่างๆ โดยเฉพาะทางด้านอิเลกทรอนิคส์มีอย่างกว้างขวาง เช่น การประมวลสัญญาณเสียงเชิงเลข (Digital Speech Processing) การประมวลผลสัญญาณภาพ (Digital Image Processing) การประมวลสัญญาณเรดาร์ เป็นต้น การประมวลสัญญาณดังตัวอย่างที่กล่าวมานี้ มีการคำนวนผลการประสาน (Convolution) เป็นสำคัญ โดยทั่วไปแล้วการแปลงฟูเรียร์แบบเต็มหน่วย (DFT) นั้นสามารถนำมาใช้ในการคำนวนการประสานได้ แต่การคำนวน DFT สำหรับลำดับยาว N ลำดับหรือจุดนั้น คอมพิวเตอร์ต้องทำการคำนวนจำนวนเชิงซ้อนถึง NxN ครั้ง และการบวกจำนวนเชิงซ้อนอีก N(N-1) ครั้ง โดยปกติขบวนการคูณตัวเลขต้องใช้เวลามากกว่าการบวกเลขมาก จึงทำให้ความเร็วในการคำนวน DFT ขึ้นอยู่กับความเร็วและจำนวนครั้งในการคูณเป็นสำคัญ ถ้าทำให้จำนวนครั้งในการคำนวนการคูณน้อยลงแล้วจะมีประโยชน์อย่างมาก

ขั้นตอนและวิธี หรือ ลำดับการในการคำนวน DFT ให้เร็วที่มีชื่อเรียกว่าฟาสท์ฟูเรียร์ (FFT) เป็นผลมาจากผลงานของ คูลีย์ (J.W Cooley) กับ เทอร์คีย์ (J.W. Turkey) ซึ่งทำ ให้เกิดการพัฒนาต่อมาจนทำให้ปัจจุบันในการคำนวน DFT ใช้การคูณเชิงซ้อนเพียง $N\log_2 N$ ครั้งเท่านั้น ผลดีของ FFT อีกประการหนึ่งคือ ทำให้การสร้างวงจรเฉพาะการคำนวน DFT ทำได้ง่าย และเร็วขึ้น เป็นผลให้การประมวลสัญญาณเชิงเลข สามารถนำไปประยุกต์ใช้กับระบบเวลาจริงได้ อย่างไรก็ตาม FFT นั้นไม่ใช่การแปลงฟูเรียร์ แต่เป็นวิธีการที่ช่วยในการคำนวน DFT ซึ่งเป็นการแปลงฟูเรียร์ให้รวดเร็วขึ้น

ตัวอย่างอัลกอริทึม FFT เช่น radix - 2 decimation-in-time และ radix-2 decimation-in-freqvency อัลกอริทึมเหล่านี้ต้องการ N เป็นเลขที่สามารถถูกแทนในรูปเลขกำลังของ 2 ในปัจจุบันจะพบอัลกอริทึม การคำนวน FFT ในรูปโปรแกรมคอมพิวเตอร์ภาษาต่าง ๆ ที่สะดวกต่อการใช้หรืออาจจะเห็นในรูปกล่อง FFT ที่มีประสิทธิภาพในราคาที่ประหยัด กล่อง FFT เหล่านี้เป็นฮาร์ดแวร์ซึ่งออกแบบมาเฉพาะสำหรับการคำนวน FFT ในลักษณะเป็นหน่วยซึ่งสามารถทำงานโดยตัวมันเอง หรือ ในลักษณะที่ใส่เข้าไปกับคอมพิวเตอร์แบบทั่วไป มีการนำเอาซอฟท์แวร์ หรือฮาร์ดแวร์ การคำนวน FFT เหล่านี้ไปประยุกต์ใช้งานในเครื่องวัด เครื่องวิเคราะห์ต่างๆอย่างกว้างขวาง นอกจากนี้ได้ค้นคว้าวิจัยและพัฒนาฮาร์ดแวร์ FFT ที่เพิ่มความสลับซับซ้อนเพื่อใช้ประมวลสัญญาณที่มีความเร็วสูงมาก ๆ

บทที่ 3

การคำนวณและการสร้าง

3.1 สถาปัตยกรรมของ TMS320C25

TMS320C25 เป็นตัวประมวลผลสัญญาณเชิงเลข (Digital Signal Processor) ที่มีขีดความสามารถสูงซึ่งใช้สถาปัตยกรรมในการออกแบบแบบฮาร์วาด (Havard-type architecture) ซึ่งทำการแยกส่วนที่อยู่ของระบบบัสหน่วยความจำของโปรแกรม (program memory) และหน่วยความจำของข้อมูล (data memory)ออกจากกัน ทำให้สามารถเฟทช์คำสั่ง (instruction fetch) และเอ็กซีคิวคำสั่ง (execution) ในเวลาเดียวกันได้ นอกจากนี้ระบบบัสของหน่วยความจำของโปรแกรมและระบบบัสของหน่วยความจำของข้อมูล ถูกมัลติเพลกซ์ (multiplexed) ให้อยู่บนระบบบัสเดียวกัน ทำให้ลดจำนวนขาของอุปกรณ์ลง รวมทั้งการมีหน่วยความจำภายในความเร็วสูงภายในที่อยู่บนชิป จึงทำให้มีความยืดหยุ่นในการใช้งาน

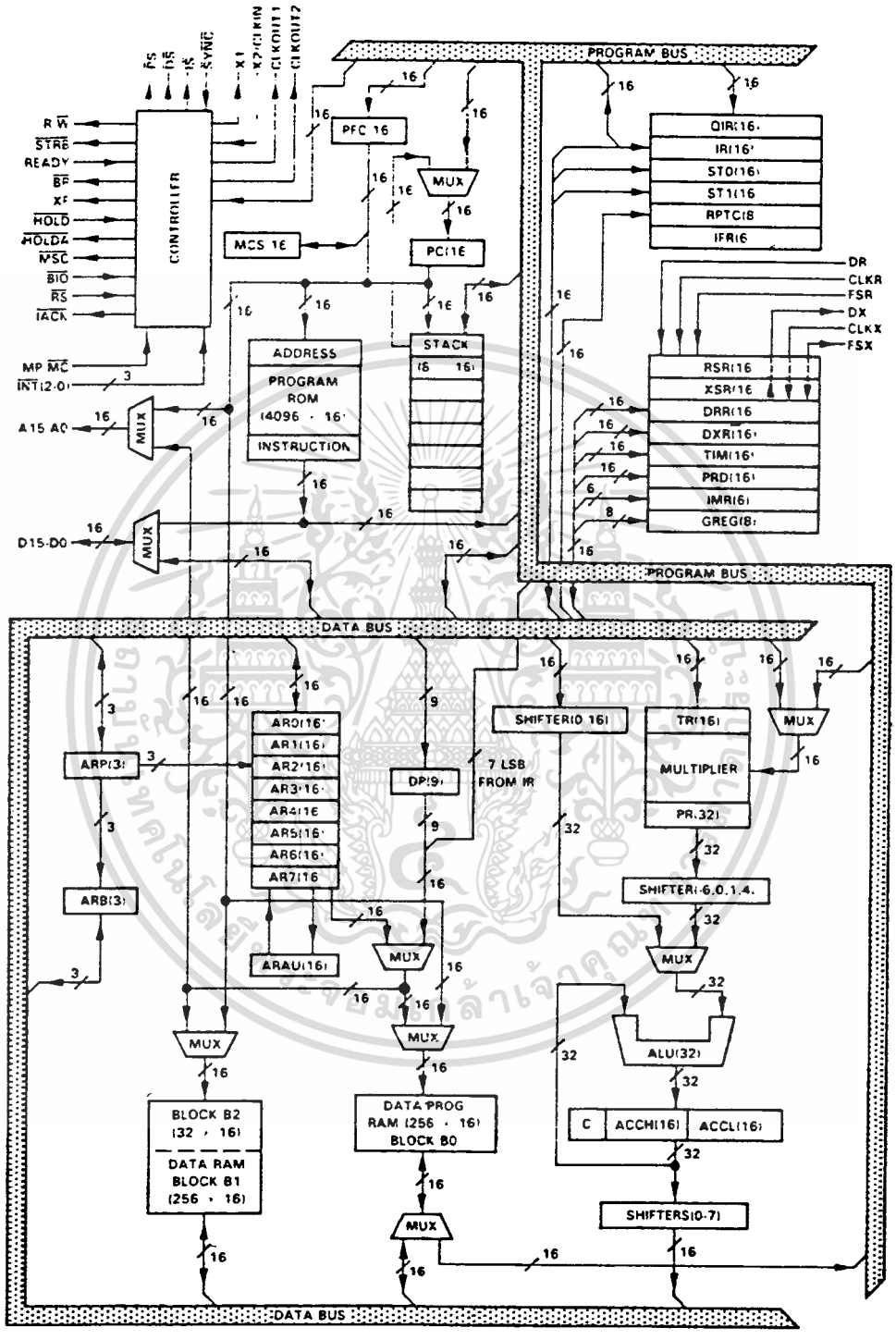
หน่วยคำนวนทางคณิตศาสตร์กลาง (Central Arithmetic Logic Unit:CALU) ของ TMS320C25 ประกอบด้วย

- 16-bit scaling shifter
- 16x16 parellel multiplier
- 32-bit Arithmetic Logic Unit (ALU)
- 32-bit accumulator
- aditional shifters

ขั้นตอนในการทำงานของคำสั่งที่เกี่ยวของกับ ALU คือ

- ข้อมูลจะถูกเฟทช์จากหน่วยความจำ RAM (Random Access Memory) ลงสู่บัส ข้อมูล
- และข้อมูลจะผ่านเข้ามาที่scaling shifter และ ALU ซึ่งจะทำการคำนวน ทางคณิตศาสตร์
- ผลลัพธ์ของการคำนวนจะถูกเคลื่อนมาเก็บไว้ในตัวสะสม (accumulator)

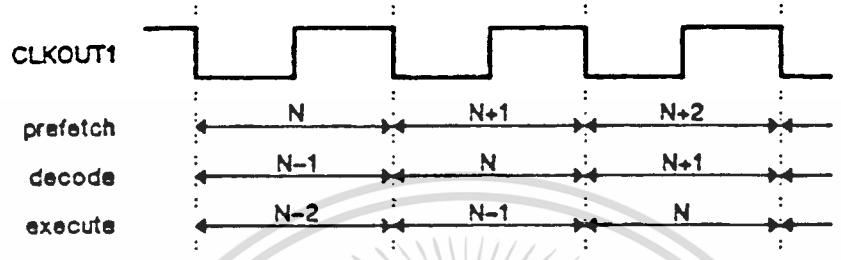
รูปที่ 3.1 : ฟังชันก์บล็อกไดอะแกรมของ TMS320C25



LEGEND:

ACCH	Accumulator high	IFR	Interrupt flag register	PC	Program counter
ACCL	Accumulator low	IMR	Interrupt mask register	PFC	Prefetch counter
ALU	Arithmetic logic unit	IR	Instruction register	RPTC	Repeat instruction counter
ARAU	Auxiliary register arithmetic unit	MCS	Microcall stack	GREG	Global memory allocation register
ARB	Auxiliary register pointer buffer	QIR	Queue instruction register	RSR	Serial port receive shift register
ARP	Auxiliary register pointer	PR	Product register	XSR	Serial port transmit shift register
DP	Data memory page pointer	PRD	Period register for timer	AR0-AR7	Auxiliary registers
DRR	Serial port data receive register	TIM	Timer	ST0,ST1	Status registers
DXR	Serial port data transmit register	TR	Temporary register		

การเอ็กซีคิว (execute) คำสั่งในการทำงาน จะทำงานในลักษณะไปป์ไลน์ (pipeline) ซึ่ง TMS320C25 จัดระบบไปป์ไลน์ ภายในของการเอกซีคิว คำสั่งทั่วไปเป็นสามระดับ ดังรูป ยกเว้นคำสั่งในการบรานช์ (branch) ต่างๆ



รูปที่ 3.2 : ไปป์ลายน์แบบ 3 ระดับ

และเมื่อเอ็กซีคิวคำสั่ง จาก RAM ภายใน การทำงานของไปป์ไลน์ จะลดลงเหลือเพียง 2 ระดับเนื่องจากคำสั่งที่อยู่ใน RAM ภายในสามารถเฟทช์ (fetch) และ ถอดระหัส (decode) ได้ภายในวงรอบ (cycle) เดียว ดังรูป

รูปที่ 3.3 : ไปป์ลายน์แบบ 2 ระดับ

TMS320C25 มีหน่วยความจำภายในทั้งหมด 544 words แบ่งออกเป็น 3 บล็อก คือ 256 words (บล็อก B0) ซึ่งใช้เป็นได้ทั้งหน่วยความจำข้อมูล (data memory) และหน่วยความจำโปรแกรม (program memory) ขึ้นอยู่กับคำสั่ง CNFD และ 228 words ที่เหลือ (บล็อก B1 และ B2) เป็นหน่วยความจำข้อมูล โดย TMS320C25 สามารถอ้างถึงหน่วยความจำข้อมูลได้ 64 kwords

นอกจากนี้ TMS320C25 ยังสามารถบรรจุหน่วยความจำ ROM ภายในที่โปรแกรมมาแล้วโดยการสั่งให้โปรแกรมจากโรงงานได้ 4096 words และ ROM ภายในนี้จะใช้หรือไม่ใช้ก็ได้ขึ้นอยู่กับโมด (mode) ในการทำงาน ถ้าเป็นโมดไมโครคอมพิวเตอร์ ก็สามารถที่จะอ้างถึง ROM ภายในนี้ได้ แต่ถ้าทำงานในโมดไมโครโพรเซสเซอร์ก็จะมองไม่เห็น ROM ส่วนนี้ ซึ่งทำได้โดยการป้อนสัญญาณเลือกให้กับขา MP/MC (MicroProcessor/MicroComputer select)

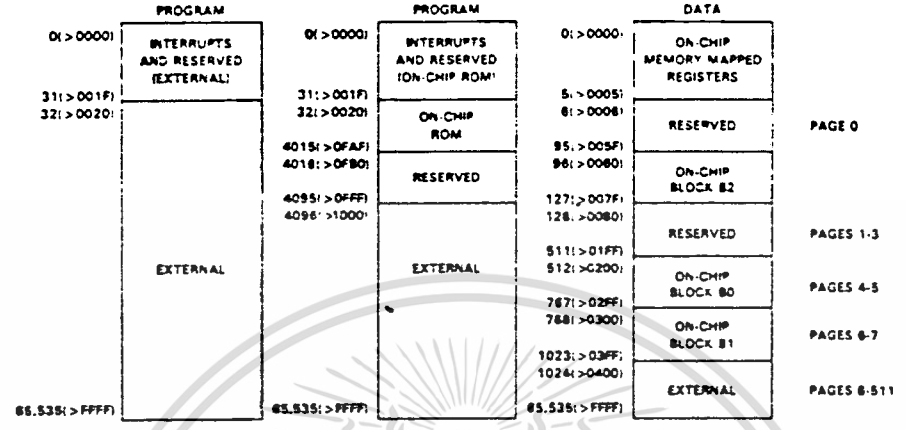
TMS320C25 สามารถอ้างถึงหน่วยความจำโปรแกรมภายนอก (program memory)ได้ 64 kwords โดยสามารถต่อได้ทั้งหน่วยความจำที่มีความเร็วสูงกว่า และหน่วยความจำความที่มีเร็วในการทำงานต่ำกว่าโดยการเพิ่มสภาวะรอ (wait state) เพื่อรอให้หน่วยความจำที่ความเร็วต่ำกว่านั้นทำงานเสร็จทันได้

และในการต่อกับอุปกรณ์อินพุทเอาท์พุท (I/O) TMS320C25 สนับสนุนการต่อกับอุกปรณ์ภายนอกหลายได้อย่าง โดยแบ่งสัญญาณในการติดต่อเป็น 3 ส่วน คือ program memory (PS),data memory (DS) และ I/O (IS) ซึ่งจะใช้แอดเดรส(addressbus) และบัสข้อมูล(data bus) ร่วมกันโดยการทำมัลติเพล็กซ์ (multiplex) และอาศัยสัญญาณ READY ซึ่งเป็นสัญญาณ อินพุทเพื่อรับรู้การติดต่อกับอุปกรณ์ในกรณีที่อุปกรณ์ภายนอกทำงานไม่ทัน TMS320C25ก็จะสร้างสภาวะรอ(wait state)ขึ้นเพื่อรออุปกรณ์เหล่านั้น

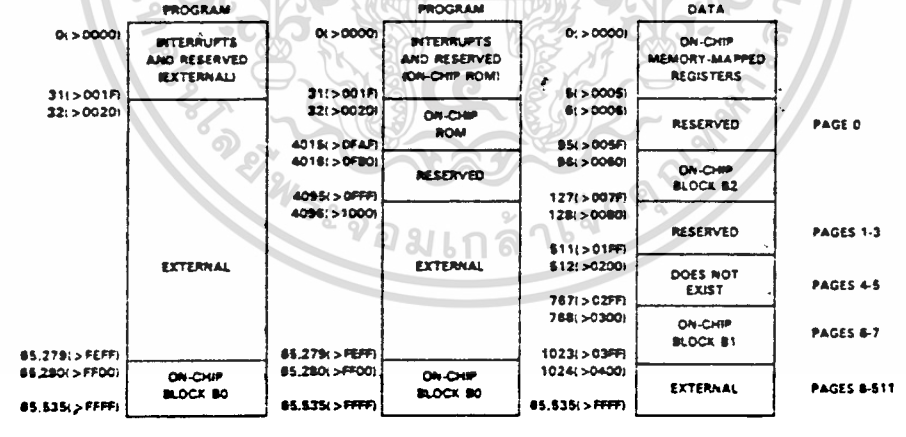
TMS320C25 สามารถต่ออินพุทและเอาท์พุทพอร์ต (I/O ports) ได้ 16 พอร์ท โดยในแต่ละพอร์ท(port) ข้อมูลเป็นแบบขนานจำนวน 16 บิท ซึ่งการสื่อสารอินพุทหรือเอาท์พุทกับอุปกรณ์สื่อสารภายนอกในแต่ละครั้งจะใช้ 2 วงรอบ (cycle) แต่อย่างไรก็ตามถ้าใช้ repeat counter การอินพุท (input) หรือ เอาท์พุท (output) ในแต่ละครั้งก็จะใช้เพียง 1 วงรอบเท่านั้น

นอกจากนี้ TMS320C25 ยังสนับสนุนการต่อแบบอนุกรม (serial) ด้วย โดยมีพอร์ทอนุกรม(serial port) ในการสื่อสารอยู่ภายในตัว ซึ่งสามารถต่อกับ ตัวแปลงระหัส (codec) ,A/D แบบอนุกรม (serial A/D) หรืออุปกรณ์อนุกรมอื่นๆได้ สามารถที่จะทำงานได้ทั้งโมด 8 บิท และ 16 บิท โดยสามารถ สร้างสัญญาณเฟรมซิงโครไนซ์ (frame syschronized)ได้ทั้งภายในและภายนอก โดยมีความเร็วสูงสุดในการสื่อสาร 5 Mhz

รูปที่ 3.4 : การจัดหน่วยความจำของ TMS320C25



(a) MEMORY MAPS AFTER A CNFD INSTRUCTION



(b) MEMORY MAPS AFTER A CNFP INSTRUCTION

3.2 คุณสมบัติของบอร์ดที่ออกแบบ

เป็นบอร์ดที่เสียบลงบนสล็อต (slot) ของเครื่องไมโครคอมพิวเตอร์ IBM PC/AT หรือเครื่องที่คล้ายกัน(compatible) โดยที่ เครื่อง IBM PC สามารถที่จะควบคุม (control) การทำ งานของบอร์ดนี้ได้

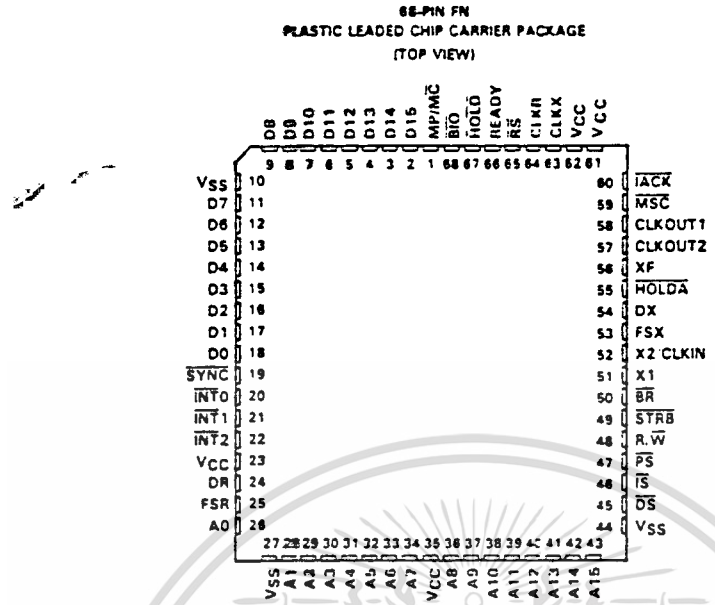
บอร์ดที่ออกแบบใช้ตัวประมวลผลสัญญาณเชิงเลข (Digital Signal Processor) ของบริษัทเทกซัสอินสตรูเมนท์ (Texas Instruments) เบอร์ TMS320C25 ซึ่งมีคุณสมบัติเด่นดังนี้

- มีช่วงเวลาของวงรอบ (cycle) ในการทำงานเพียง 100 nS (10 ล้านคำสั่ง/วินาที)
- มีหน่วยความจำ ROM (Read Only Memery) บนชิป (on-chip) 4 kwords
- มีหน่วยความจำ RAM (Random Access Memory) ภายในบนชิป 544 words
- ต่อกับหน่วยความจำภายนอกได้ 128 kwords โดยแบ่งเป็น หน่วยความจำโปรแกรม(Program Memery) 64 kwords และ หน่วยความจำข้อมูล (Data Memory) 64 kwords
- มีรีจิสเตอร์(register) ช่วยในการคำนวณ 8 ตัว
- สามารถสร้างสภาวะรอ(wait state) เพื่อรอหน่วยความจำและอุปกรณ์ภายนอกที่มีความเร็วน้อยกว่าได้
- มีโมดการอ้างแอดเดรส (addressing mode) แบบ Bit-reverred ซึ่งมีประโยชน์มากในการทำ FFT (Fast Fourier Transform)
- ใช้เทคโนโลยีซีมอส (CMOS) 1.8 um
- สามารถทำกระบวนการ DMA (Direct Memory Access) ได้

DSP ที่อยู่บนบอร์ดสามารถทำงานเป็น Co-Processor ของ CPU (Central Processing Unit) หลัก ของเครื่องคอมพิวเตอร์(IBM PC) ได้ ซึ่งสามารถทำ การประมวล ผลแบบขนาน (Parallel Processing) กับ CPU หลักได้ โดย TMS320C25 มีหน่วยความจำเป็นของตัวเอง

มีการรับส่งสัญญาณอินเทอรรัพระหว่าง CPU หลัก และTMS320C25 ได้

ติดต่อสื่อสารกับเครื่องคอมพิวเตอร์ โดยใช้วิธี DMA (Direct Memory Access)

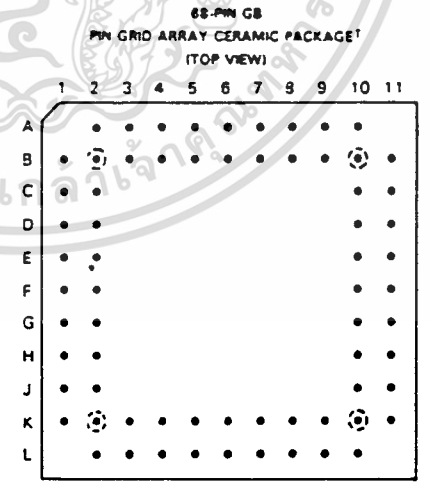


รูปที่ 3.5 : TMS320C25 68 ขา แบบ FN

Plastic Leaded Chip Package

PIN ASSIGNMENTS

PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION
A2	D8	C11	CLKOUT1	J10	PS
A3	D10	D1	D4	J11	IS
A4	D12	D2	D3	K1	A0
A5	D14	D10	CLKOUT2	K2	A1
A6	VCC	D11	XF	K3	A3
A7	HOLD	E1	D2	K4	A5
A8	RS	E2	D1	K5	A7
A9	CLKX	E10	HOLDA	K8	A8
A10	VCC	E11	DX	K7	A10
B1	VSS	F1	D0	K8	A12
B2	D7	F2	SYNC	K9	A14
B3	D9	F10	FSX	K10	DS
B4	D11	F11	X2/CLKIN	K11	VSS
B5	D13	G1	INT0	L1	VSS
B6	D15	G2	INT1	L2	A2
B7	BIO	G10	X1	L3	A4
B8	READY	G11	BR	L4	A6
B9	CLKR	H1	INT2	L5	VCC
B10	VCC	H2	VCC	L6	A9
B11	IACK	H10	STRB	L7	A11
C1	D6	H11	R/W	L8	A13
C2	D5	J1	DR	L10	A15
C10	MSC	J2	FSR		



† See Pin Assignments Table (Page 1) and Pin Nomenclature Table (Page 2) for location and description of all pins

รูปที่ 3.6 :TMS320C25 แบบ GB Pin Grid Array Package

และตารางเทียบขาสัญญาณของ TMS320C25 แบบ FN และแบบ GB

3.3 รายละเอียดของขาสัญญาณต่างๆของ TMS320C25

สัญญาณ	ขา	สถานะ	รายละเอียด
บัสแอดเดรส/บัสข้อมูล			
A15 MSB A14-A8 A7-A1 A0 LSB	43 42-36 34-28 26	เอาท์พุท/ อิมพีแดนซ์สูง	บัสแอดเดรสแบบขนานโดยมี A15 เป็นบิทที่มีนัยสำคัญสูงสุด (MSB) และ A0 เป็นบิทที่มีนัยสำคัญต่ำสุด (LSB) มัลติเพล็กซ์เป็นแอดเดรสของข้อมูลภายนอก /หน่วยความจำโปรแกรม/พอร์ทอินพุทเอาท์พุท และจะเป็นอิมพีแดนซ์สูงเมื่ออยู่ในโมดโฮลด์(HOLD)
D15 MSB D14-D8 D7-D1 D0 LSB	2 3-9 11-17 18	อินพุท/ เอาท์พุท/ อิมพีแดนซ์สูง	บัสข้อมูลแบบขนานจำนวน 16 บิท D15 เป็น MSB และ D0 เป็น LSB มัลติเพล็กซ์ในการรับส่งข้อมูลระหว่าง TMS320C25 และ ข้อมูลภายนอก/ข้อมูลโปรแกรม/ข้อมูลอินพุทเอาท์พุทพอร์ท และเป็นอิมพีแดนซ์สูงเมื่อไม่มีการเอาท์พุท หรือ เมื่อสัญญาณ RS หรือ HOLD แอคทีฟ
สัญญาณควบคุม			
DS IS PS	45 46 47	เอาท์พุท/ อิมพีแดนซ์สูง	โดยปกติสัญญาณอยู่ที่ระดับสูง(high) และจะแอคทีฟที่ระดับต่ำ(active low) เมื่อมีการติดต่อกับอุปกรณ์ภายนอกและจะเป็นอิมพีแดนซ์สูงเมื่ออยู่ในโมดโฮลด์

สัญญาณ	ขา	สถานะ	รายละเอียด
READY	66	อินพุท	เป็นสัญญาณอินพุทที่อุปกรณ์ภายนอกใช้บอก TMS320C25 ให้รอ โดยถ้า TMS320C25 ได้รับสัญญาณจากอุปกรณ์ว่าไม่พร้อม (READY =0)TMS320C25 จะทำการอเป็นช่วงเวลาหนึ่งไซเคิลและจะตรวจสัญญาณนี้ใหม่
R/W	48	เอาท์พุท/ อิมพีแดนซ์สูง	เป็นสัญญาณที่ใช้บอกทิศทางในการติดต่อของ TMS320C25 กับอุปกรณ์ภายนอก และเป็นอิมพีแดนซ์สูงเมื่ออยู่ในโมดโฮลด์
STRB	49	เอาท์พุท/ อิมพีแดนซ์สูง	สัญญาณสโตป โดยปกติจะเป็นระดับสูง และจะอยู่ในระดับต่ำเพื่อบอกว่าอยู่ในไซเคิลบัสภายนอกและเป็นอิมพีแดนซ์สูงเมื่ออยู่ในโฮลด์โมด
สัญญาณมัลติโปรเซสซิ่ง			
BR	50	เอาท์พุท	สัญญาณขอใช้บัส เมื่อ TMS320C25 ต้องการติดต่อกับ หน่วยความจำข้อมูลโกลบอล (gobal)ภายนอก และสัญญาณ READY จะถูกส่งมาเพื่อบอกอนุญาติให้ใช้บัส
HOLD	67	อินพุท	เมื่อมีอินพุทแอคทีฟ TMS320C25 จะทำให้บัสแอดเดรส,บัสข้อมูล และ บัสควบคุม อยู่ในสภาวะอิมพีแดนซ์สูง

สัญญาณ	ขา	สถานะ	รายละเอียด
HOLDA	55	เอาท์พุท	เป็นเอาท์พุทที่บอกให้รู้ว่า TMS320C25กำลังอยู่ในโมดโฮลด์อยู่ดังนั้นตัวโปรเซสเซอร์อื่นก็สามารถที่จะติดต่อกับหน่วยความจำภายนอกของ TMS320C25 ได้
SYNC	19	อินพุท	ใช้ในการซิงโครไนซ์ของ TMS320C25ที่มากกว่า 1 ตัว
สัญญาณอินเทอรัพท์และสัญญาณพิเศษ			
BIO	68	อินพุท	สัญญาณอินพุทที่ใช้ในการควบคุม การบรานช์ (branch) ซึ่งจะถูกโพล(poll)โดยคำสั่ง BIOZ ถ้าสัญญาณนี้แอคทีฟ (BIO=0) เมื่อ TMS320C25ทำคำสั่งBIOZ จะทำการบรานช์
IACK	60	เอาท์พุท	สัญญาณตอบรับการขออินเทอร์รัพท์
INT0 INT1 INT2	20 21 22	อินพุท	สัญญาณที่อุปกรณ์ภายนอก ใช้ในการขออิน-เทอร์รัพท์จาก TMS320C25 ลำดับการตอบสนองก่อนหลัง ถูกมาร์ค(mark)โดยอินเทอร์-รัพมาร์ครีจีสเตอร์และอินเทอร์รัพโมดบิท์
MP/MC	1	อินพุท	ใช้บอกโมดการทำงานของ TMS320C25 ให้เป็นแบบไมโครคอมพิวเตอร์หรือแบบไมโครโปรเซสเซอร์ ถ้าสัญญาณเป็นระดับต่ำจะทำให้ROM ภายในถูกแทนที่(map)กับหน่วยความจำโปรแกรมภายนอก 4 Kwords แรก

สัญญาณ	ขา	สถานะ	รายละเอียด
MSC	59	เอาท์พุท	จะถูกใช้เป็นระดับต่ำในช่วง CLKOUT1 เป็นระดับต่ำ ในขณะเมื่อTMS320C25 ปฏิบัติการกับหน่วยความจำอย่างสมบูรณ์ และ MSC สามารถสร้างสัญญาณ READY 1 สภาวะการรอ(1 wait-state)สำหรับหน่วยความจำที่ทำงานช้ากว่าได้
RS	65	อินพุท	เมื่อแอคทีฟ จะทำให้ TMS320C25 สิ้นสุดการปฏิบัติคำสั่ง และบังคับให้โปรแกรมเคาท์เตอร์ (program counter) เป็นศูนย์
XF	56	เอาท์พุท	เป็นเอาท์พุทแฟลก(flag)ภายนอกใช้เป็นขาเอาท์พุทเอนกประสงค์
สัญญาณออสซิสเลเตอร์และไฟเลี้ยง			
CLKOUT1	58	เอาท์พุท	สัญญาณคล็อกเอาท์พุทหลัก (CLKIN / 4)
CLKOUT2	57	เอาท์พุท	สัญญาณคล็อกเอาท์พุทรอง (CLKIN / 4)
V_{cc}	23,35 61,62	อินพุท	ไฟเลี้ยง 5 โวลท์
V_{ss}	10,27 44	อินพุท	ขากราวน์ (ground)
X1	51	เอาท์พุท	ขาเอาท์พุทจากออสซิลเลเตอร์ภายในสำหรับคริสตอล ถ้าไม่ใช้คริสตอล ควรจะปล่อยว่าง

สัญญาณ	ขา	สถานะ	รายละเอียด
X2/ CLKIN	52	อินพุท	ขาอินพุทต่อกับออสซิลเลเตอร์ภายในสำหรับคริสตอล ถ้าไม่ใช้คริสตอล สามารถต่อคล็อกอินพุท เข้าที่ขานี้
สัญญาณพอร์ทอนุกรม			
CLKR	64	อินพุท	คล็อกอินพุท ที่ใช้ในการรับข้อมูลจากพอร์ทอนุกรม
CLKX	63	อินพุท	คล็อกอินพุท ที่ใช้ในการส่งข้อมูลจากพอร์ทอนุกรม
สัญญาณอินเทอรัพท์และสัญญาณพิเศษ			
DR	24	อินพุท	อินพุทรับข้อมูลแบบอนุกรม โดยข้อมูลจะถูกรับเข้าไปใน RSR (serial port receive shift register)
DX	54	เอาท์พุท อิมพีแดนซ์สูง	เอาท์พุทส่งข้อมูลแบบอนุกรม โดยข้อมูลจะถูกส่งจาก XSR (serial port transmit shift register)และจะเป็นอิมพีแดนซ์สูงเมื่อไม่มีการส่งข้อมูล
FSR	25	อินพุท	เฟรมชิงโครไนซ์พัลซ์สำหรับรับอินพุท
FSX	53	อินพุท/ เอาท์พุท	เฟรมชิงโครไนซ์พัลซ์สำหรับส่งอินพุท/ เอาท์พุท

รูปที่ 3.7 : สล็อตของเครื่องไมโครคอมพิวเตอร์ IBM PC/AT

REAR PANEL

GND	B1	A1	–I/O CH CK
RESET DRV	B2	A2	SD07
+5V	B3	A3	SD06
IR 9	B4	A4	SD05
–5V	B5	A5	SD04
DRQ 2	B6	A6	SD03
–12V	B7	A7	SD02
0WS	B8	A8	SD01
+12V	B9	A9	SD00
GND	B10	A10	–I/O CH RDY
–S MEMW	B11	A11	AEN
–S MEMR	B12	A12	SA19
–IOW	B13	A13	SA18
–IOR	B14	A14	SA17
–DACK3	B15	A15	SA16
DRQ3	B16	A16	SA15
–DACK1	B17	A17	SA14
DRQ1	B18	A18	SA13
–REFRESH	B19	A19	SA12
SYSCLK	B20	A20	SA11
IR 7	B21	A21	SA10
IR 6	B22	A22	SA09
IR 5	B23	A23	SA08
IR 4	B24	A24	SA07
IR 3	B25	A25	SA06
–DACK2	B26	A26	SA05
T/C	B27	A27	SA04
BALE	B28	A28	SA03
+5V	B29	A29	SA02
OSC	B30	A30	SA01
GND	B31	A31	SA00

–MEM CS16	D1	C1	SBHE
–I/O CS16	D2	C2	LA23
IR 10	D3	C3	LA22
IR 11	D4	C4	LA21
IR 12	D5	C5	LA20
IR 15	D6	C6	LA19
IR 14	D7	C7	LA18
–DACK0	D8	C8	LA17
DRQ0	D9	C9	–MEMR
–DACK5	D10	C10	–MEMW
DRQ5	D11	C11	SD08
–DACK6	D12	C12	SD09
DRQ6	D13	C13	SD10
–DACK7	D14	C14	SD11
DRQ7	D15	C15	SD12
+5V	D16	C16	SD13
–MASTER	D17	C17	SD14
GND	D18	C18	SD15

I/O CHANNEL PIN NUMBERING

3.4 รายละเอียดเกี่ยวกับสัญญาณต่างๆบนสล็อตของเครื่องไอบีเอ็มพีซี

OSC (Oscillator; ขา B30) :

ขานี้เป็นเอาท์พุทที่เชื่อมกับสัญญาณคล็อก (clock) ที่มีค่าความถี่ คือ 14.31818 MHz ซึ่งมี Duty Cycle (ช่วงเวลาใน 1 คาบที่สัญญาณคล็อกมีลอจิกเป็น "1" หารด้วยคาบเวลาทั้งหมด)ประมาณ 50% สัญญาณคล็อกอื่นๆ ของระบบ เช่น คล็อกที่ป้อนให้กับ CPU หรือ ชิพซับพอร์ทต่างๆนั้นจะถูกสร้างขึ้นโดยการหารสัญญาณคล็อกนี้

CLK (Clock; B20) :

ขานี้เป็นเอาท์พุท ซึ่งต่ออยู่กับสัญญาณคล็อกที่มีความถี่ขึ้นอยู่กับความถี่ที่เครื่องแต่ละเครื่องใช้ และมีค่า Duty Cycle ของสัญญาณมีค่าประมาณ 1/3 คือใน 1 คาบจะมีช่วงเวลาที่เป็นลอจิก "1" เท่ากับ 1/3 ของคาบเวลาทั้งหมด สัญญาณนี้เป็นสัญญาณที่ถูกใช้เป็นคล็อกของระบบ

A0-A19 (Address Bus; ขา A31-A12) :

ขาสัญญาณทั้ง 20 ขานี้เป็นเอาท์พุท ซึ่งใช้สำหรับกำหนดแอดเดรสของหน่วยความจำ หรืออุปกรณ์ I/O ที่ CPU ต้องการติดต่อด้วย โดยที่สัญญาณ A0 จะมีนัยสำคัญต่ำสุด (Least Significant Bit) และ A19 จะมีนัยสำคัญสูงสุด (Most Significant Bit) สำหรับค่าแอดเดรสบนบัสแอดเดรส A0-A19 นี้ จะถูกกำหนดโดย CPU ในระหว่างขบวนการอ่าน/เขียนข้อมูลลงในหน่วยความจำหรืออุปกรณ์ I/O แต่ในช่วงของขบวนการ DMA นั้นตัวควบคุม DMA จะเป็นผู้กำหนดค่าแอดเดรสบนบัสแอดเดรสเอง (ในระหว่างนี้ CPU จะถูกตัดออกจากระบบ)

จะเห็นได้ว่าจำนวนเส้นแอดเดรสนี้จะมีอยู่ 20 เส้น ซึ่งสามารถที่จะอ้างแอดเดรสของหน่วยความจำได้ถึง 1Mbyte แต่อย่างไรก็ตามจะมีแอดเดรสบางแอดเดรสที่ถูกใช้งานโดย IBM/PC อยู่ก่อนแล้ว คือแอดเดรสของหน่วยความจำ RAM บนเมนบอร์ดที่ถูกใช้โดยระบบ จำนวน 64Kbyte และแอดเดรสสำหรับหน่วยความจำ ROM อีก 48Kbyte ซึ่งถูกจัดในช่วงของแอดเดรสบนสุดใน 1Mbyte คือ OFCOOH จนถึง OFFFFFH (สำหรับ IBM PC /XT จะเป็น 64Kbyte)

สำหรับการอ้างแอดเดรสของพอร์ท I/O นั้น จะใช้เส้นแอดเดรสเพียง 16 เส้น คือ A0-A15 ซึ่งจะทำให้อ้างแอดเดรสที่เหลือคือ A16-A19 นั้นจะไม่ถูกใช้งาน อย่างไรก็ตามภายใน IBM/PC จะใช้เส้นแอดเดรสในการอ้างแอดเดรสของพอร์ทเพียง 10 เส้น คือจาก A0-A9 และค่าแอดเดรสที่ใช้งานจะต้องอยู่ในช่วง 0200H จนถึง 03FFH เท่านั้น

D0-D7 (Data Bus; ขา A9-A2) :

ขาสัญญาณนี้จะเป็นแบบสองทิศทาง(Bi-Directional) ซึ่งต่อกับบัสข้อมูลของระบบ เพื่อทำหน้าที่ในการส่งผ่านข้อมูลระหว่างพอร์ทI/Oกับ IBM/PC โดยบิท D0 จะมีนัยสำคัญต่ำสุดและบิท D7 จะมีนัยสำคัญสูงสุด

สำหรับในบัสไซเคิลของการเขียนข้อมูลที่สร้างขึ้นโดย CPU นั้น ข้อมูลจะถูกส่งออกมาบนบัสข้อมูล ก่อนที่สัญญาณ IOW (ในกรณีที่ต้องการส่งข้อมูลให้กับพอร์ท)หรือ SMEMW (ในกรณีที่ต้องการส่งข้อมูลให้กับหน่วยความจำ) จะเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น) ซึ่งโดยทั่วไปขอบขาขึ้นของสัญญาณ IOW หรือ SMEMW นี้ จะถูกใช้เพื่อสั่งให้พอร์ท I/O หรือหน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้นรับข้อมูลไปเก็บไว้

สำหรับในบัสไซเคิลของการอ่านข้อมูลที่สร้างขึ้นโดย CPU นั้น พอร์ท I/O หรือหน่วยความจำที่ถูกอ้างถึงจะต้องส่งข้อมูลออกมาบนบัสข้อมูล ก่อนที่สัญญาณ IOR (ในกรณีที่ต้องการอ่านข้อมูลจากพอร์ท) หรือ SMEMR (ในกรณีที่ต้องอ่านข้อมูลจากหน่วยความจำ) จะเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น)

ALE (Address Latch Enable; ขา B28) :

ขาสัญญาณนี้เป็นสัญญาณเอาท์พุทที่ ตัวควบคุมบัส (Bus Controller)' สร้างขึ้นเพื่อใช้สำหรับแสดงการเริ่มต้นของบัสไซเคิล และแสดงให้อุปกรณ์ภายนอกทราบว่าแอดเดรสที่ CPU ต้องการจะติดต่อด้วยนั้นถูกส่งออกมาบนบัสแอดเดรสแล้ว โดยที่สัญญาณ ALE นี้จะเปลี่ยนจากลอจิก "1" เป็น "0" เมื่อแอดเดรสที่ถูกต้องถูกส่งออกมาบนบัสข้อมูลเรียบร้อยแล้ว ดังนั้นขอบขาลงของสัญญาณ ALE นี้จะถูกใช้ในการแลทช์ค่าแอดเดรสจากบัสแอดเดรส/ข้อมูล (Address/Data Bus;AD0-AD7) ของ CPU ทำให้สามารถแยกค่าแอด

เดรส (A0-A19) และข้อมูล (A0-A7) ออกจากกันได้ อย่างไรก็ตามสัญญาณ ALE จะแอคทีฟเฉพาะในบัสไซเคิลที่สร้างขึ้นโดย CPU เท่านั้น โดยจะไม่แอคทีฟในระหว่างขบวนการ DMA

I/O CHCK (I/O Channel Check; ขา A1) :

ขาสัญญาณนี้เป็นอินพุทที่ใช้ในการแสดงความผิดพลาดเกี่ยวกับพาริตี้ ที่เกิดขึ้นในการทำงานของวงจรอินเตอร์เฟสหรืออุปกรณ์ I/O เมื่อขาสัญญาณได้รับลอจิก "0" จะทำให้ CPU ถูกอินเทอร์รัพท์แบบ Non-Maskable (NMI) อย่างไรก็ตามเราสามารถที่จะกำหนดให้วงจรภายในของ IBM/PC ทำการขออินเทอร์รัพท์ (เมื่อได้รับสัญญาณ I/O CHCK) หรือไม่ก็ได้โดยการกำหนดลอจิกของบิทข้อมูลของพอร์ทที่ควบคุมการขออินเทอร์รัพท์แบบ NMI คือบิท D7 ของพอร์ท OOAOH ในกรณีที่บิท D7 ของพอร์ท AOH ถูกเซ็ทเป็น "1" ก็จะทำให้วงจรภายนอกอินเทอร์รัพท์แบบ NMIได้ (Enable) แต่ถ้าบิท D7 ของพอร์ท AOH ถูกเซ็ทเป็น "0" ก็จะเป็นการดีสเอเบิล (Disable) การขออินเทอร์รัพท์แบบ NMI ดังนี้

Enable : ใช้คำสั่ง OUT ส่งข้อมูล 80H ไปยังพอร์ท AOH

Disable : ใช้คำสั่ง OUT ส่งข้อมูล OOH ไปยังพอร์ท AOH

และเนื่องจากยังมีอุปกรณ์อื่นที่สามารถขออินเทอร์รัพท์แบบNMIได้อีก ดังนั้นซอฟท์แวร์ที่ใช้งานจะต้องสามารถตรวจสอบว่าการขออินเทอร์รัพท์นั้นเกิดขึ้นจากแหล่งใดได้ด้วย

I/O CHRDY (I/O Channel Ready; ขา A10) :

ขาสัญญาณนี้เป็นอินพุทที่ใช้เพิ่มช่วงเวลาในบัสไซเคิลในกรณีที่มีอุปกรณ์ I/O หรือหน่วยความจำที่เกี่ยวข้องกับขบวนการในบัสไซเคิลที่เกิดขึ้นนั้นไม่สามารถทำงานทันตามช่วงเวลาปกติของบัสไซเคิลนั้นๆได้ (ช่วงเวลาของบัสไซเคิลที่เกี่ยวกับหน่วยความจำใช้ช่วงเวลาเท่ากับช่วงเวลาของคล็อก 4 ลูก ในขณะบัสไซเคิลที่เกี่ยวกับ I/O โดยใช้เวลาเท่ากับช่วงเวลาของคล็อก 5 ลูก)

เมื่ออุปกรณ์ I/O หรือหน่วยความจำต้องการที่จะเพิ่มช่วงเวลาในบัสไซเคิลให้นานขึ้นอีกนั้น จะสามารถทำได้โดยการป้อนลอจิก "0" ให้กับขา I/O CHRDY ในช่วงเวลาที่ I/O หรือ หน่วยความจำที่ถูกกำหนดนั้น ได้รับสัญญาณจากการดีโค้ดแอดเดรส และสัญญาณ SMEMR,SMEMW,IOR หรือ IOW แอคทีฟ

IRQ2-IRQ7 (Interrupt Request 2-7; ขา B4 และ B25-21) :

ขาสัญญาณทั้ง 6 นี้เป็นขาอินพุทที่ใช้สำหรับการขออินเทอร์รัพท์จาก CPU โดยสัญญาณเหล่านี้จะต่อเข้ากับ ตัวควบคุมการอินเทอร์รัพ บนเมนบอร์ดโดยตรง โปรแกรมในส่วน BIOS ของ IBM/PC จะทำการโปรแกรม ตัวควบคุมการอินเทอร์รัพ ให้ IRQ2 มีลำดับความสำคัญสูงสุด (Highest Priority) และ IRQ7 มีระดับความสำคัญต่ำสุด ในกรณีที่มีการขออินเทอร์รัพท์เกิดขึ้นคือระดับของลอจิกที่ขา IRQ ขาใดขาหนึ่งถูกเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น) ตัวควบคุมการอินเทอร์รัพก็จะทำการส่งสัญญาณ INT ให้กับ CPU เพื่อทำการขออินเทอร์รัพท์

สิ่งสำคัญในการขออินเทอร์รัพท์โดยผ่านทาง IRQ2-IRQ7 นี้ ก็คืออุปกรณ์ที่ทำการขออินเทอร์รัพท์โดยผ่านทางขา IRQ ขาใดก็จะต้องรักษาระดับสัญญาณที่ขาIRQนั้น ให้แอคทีฟ (ลอจิก "1") อยู่จนกว่าจะได้รับสัญญาณ INTA (Interrupt Acknowledge) จาก CPU เสียก่อน ถ้าไม่เช่นนั้นการขออินเทอร์รัพท์จะถูกยกเลิก และอินเทอร์รัพท์ Level 7 (IRQ7) ก็จะถูกสร้างขึ้นโดยอัตโนมัติ ไม่ว่าการขออินเทอร์รัพท์ที่ถูกยกเลิกนั้นจะเป็นการขออินเทอร์รัพท์ใน Level หรือขาใด

แต่อย่างไรก็ตามสัญญาณ INTA นี้จะไม่ถูกต่อออกมาที่ขาของสล็อตด้วย ดังนั้นโปรแกรมที่ทำการตอบสนองต่อการขออินเทอร์รัพท์ (Interrupt Service Routine) จะต้องทำการรีเซ็ทสัญญาณ IRQ เอง โดยใช้คำสั่ง OUT ไปยังพอร์ท I/O ที่เกี่ยวข้อง

IOR (I/O Read; ขา B14) :

ขาสัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ที่สร้างขึ้นโดย ตัวควบคุมบัส เพื่อใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นนี้ เป็นบัสไซเคิลของการ

อ่านข้อมูลจากพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้นส่งข้อมูลออกมาบนบัสข้อมูล โดยข้อมูลจะต้องถูกส่งออกมาบนบัสข้อมูลก่อนขอบขาขึ้นของสัญญาณ IOR ประมาณ 30 nanosec. เพื่อให้มั่นใจได้ว่า CPU สามารถรับข้อมูลได้ถูกต้อง สำหรับในขบวนการ DMA ตัวควบคุม DMA (DMA Controller) จะทำการสร้างสัญญาณ IOR เอง โดยที่ค่าแอดเดรสที่อยู่บนบัสแอดเดรสจะเป็นค่าแอดเดรสของหน่วยความจำ (แทนที่จะเป็นแอดเดรสของพอร์ท I/O) ที่พอร์ท I/O ที่ขอ DMA ต้องการจะนำข้อมูลไปเก็บ การที่พอร์ทใดจะส่งข้อมูลออกมาบนบัสข้อมูลนั้น จะอาศัยสัญญาณ DACK จากตัวควบคุม DMA เป็นตัวกำหนด เช่นกรณีที่สัญญาณ DACK 1 แอคทีฟก็จะแสดงว่าพอร์ท I/O ที่จะส่งข้อมูลออกมาบนบัสของข้อมูลก็คือพอร์ท I/O ที่ขอ DMA ผ่านทางแชนแนลที่ 1 (DRQ 1) เป็นต้น

IOW (I/O Write; ขา B13) :

ขาสัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก"0" ซึ่งถูกสร้างขึ้นโดยตัวควบคุมบัส (Bus Controller) เพื่อใช้แสดงว่าบัสไซเคิลที่เกิดขึ้นนี้เป็นบัสไซเคิลของการเขียนข้อมูลลงบนพอร์ท I/O เพื่อให้พอร์ท I/O นี้มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้น รับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ อย่างไรก็ตามเนื่องจากในช่วงเวลาที่สัญญาณ IOW นี้แอคทีฟ (ลอจิก "0") นั้นข้อมูลบนบัสข้อมูลอาจจะยังไม่สมบูรณ์ ดังนั้นในการออกแบบจึงควรใช้ขอบขาขึ้นของสัญญาณ IOW แทนขอบขาลงในการทำให้พอร์ท I/O ที่เกี่ยวข้องรับข้อมูลไปเก็บไว้ เพื่อให้ข้อมูลบนบัสข้อมูลสมบูรณ์เสียก่อน สำหรับในขบวนการ DMA นั้นตัวควบคุม DMA จะทำการสร้างสัญญาณ IOW เอง โดยที่ค่าแอดเดรสที่อยู่บนบัสแอดเดรสจะเป็นค่าแอดเดรสของหน่วยความจำที่พอร์ท I/O ที่ขอ DMA ต้องการจะอ่านข้อมูล

SMEMW (System Memory Write; ขา B11) :

ขานี้เป็นเอาท์พุทแอคทีฟที่ลอจิก"0" ซึ่ง ตัวควบคุมบัส สร้างขึ้นในระหว่างบัสไซเคิลในการเขียนข้อมูลลงในหน่วยความจำของ CPU สัญญาณ SMEMW นี้จะถูกส่งออกมาเพื่อให้หน่วยความจำที่แอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการรับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ โดยทั่วไปหน่วยความจำจะรับข้อมูลในช่วงขอบขาขึ้นของสัญญาณSMEMW

สำหรับในขบวนการ DMA นั้น ตัวควบคุม DMA (DMA Controller) จะทำการควบคุมบัสต่างๆ ของระบบแทน CPU และสัญญาณSMEMWจะถูกใช้ในบัสไซเคิลของการเขียนข้อมูลลงในหน่วยความจำ

SMEMR (System Memory Read; ขา B12) :

ขานี้เป็นเอาท์พุทจาก ตัวควบคุมบัส ซึ่งสัญญาณนี้จะแอคทีฟ (ลอจิก "0") ในระหว่างบัสไซเคิลของการอ่านข้อมูลจากหน่วยความจำของ CPU เพื่อให้หน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสนั้น ทำการส่งข้อมูลออกมาบยบัสข้อมูล โดยหน่วยความจำนั้นจะต้องส่งข้อมูลออกมาในช่วงเวลา 30 nanosec. ก่อนที่สัญญาณSMEMWจะกลับเป็นลอจิก "1" ทั้งนี้ก็เพื่อให้ CPU ได้รับข้อมูลที่ถูกต้อง

สำหรับในระหว่างขบวน การตัวควบคุม DMA จะควบคุมบัสต่างๆ ของระบบแทน CPU และสัญญาณ SMEMR จะถูกใช้ในบัสไซเคิลในการอ่านข้อมูลของหน่วยความจำ (ข้อมูลถูกส่งจากหน่วยความจำไปให้กับอุปกรณ์ I/O)

DRQ1-DRQ3 (DMA Request 1-3; ขา B16,B6 และ ขา B16) :

ขาสัญญาณทั้งสามนี้เป็นสัญญาณอินพุทแอคทีฟที่ลอจิก "1" ซึ่งอุปกรณ์ภายนอกสามารถใช้ในการขอ DMA จากระบบ โดยการป้อนระดับสัญญาณลอจิก "1" ให้กับขา DRQ ขาใดขาหนึ่ง (ขา DRQ ทั้งสามนี้จะต่อเข้ากับ DRQ1-DRQ3 ของ ตัวควบคุม DMA)

เมื่อตัวควบคุม DMA ได้รับสัญญาณนี้แล้วก็จะตรวจสอบว่ามีการขอ DMA ในแชนแนลที่มีลำดับความสำคัญ (Priority) สูงกว่าหรือไม่ ถ้าไม่มีการขอ DMA จาก CPU และ ตอบรับการขอ DMA จากอุปกรณ์ภายนอก (สัญญาณ DACK ของแชนแนลที่ขอ DMA จะแอคทีฟ) แต่ถ้ามี ตัวควบคุม DMA ก็ขอ DMA ให้กับแชนแนลที่มีลำดับความสำคัญสูงกว่าก่อนแล้วจึงทำการขอ DMA ให้กับแชนแนลที่มีลำดับความสำคัญต่ำกว่า ภายใน ROM BIOS ของ IBM/PC จะโปรแกรมตัวควบคุม DMA ให้ DRQ1 มีลำดับความสำคัญสูงสุด และ DRQ3 มีลำดับความสำคัญต่ำสุด ดังนั้นถ้ามีการขอ DMA ของอุกกรณภายนอก ผ่านทางแชนแนลที่ 1 (DRQ1) และแชนแนลที่ 2 (DRQ2) ตัวควบคุม DMA ก็จะทำการขอ DMA ให้กับแชนแนลที่ 1 ก่อน จากนั้นเมื่อเสร็จจากขบวนการ DMA ของแชนแนลที่ 1 แล้ว จึงจะทำการขอ DMA ให้กับแชนแนลที่ 2

ในการขอ DMA นั้นสัญญาณ DRQ นี้จะต้องแอคทีฟอยู่ในช่วงระยะเวลาหนึ่งเท่านั้น ถ้าสัญญาณนี้แอคทีฟอยู่นานเกินไป จะทำให้เกิดขบวนการ DMA มากกว่า 1 ขบวนการได้ สำหรับวงจรที่ขอ DMA โดยทั่วไปแล้วจะใช้สัญญาณตอบรับการขอ DMA หรือสัญญาณ DACK ของแชนแนลที่ขอ DMA นั้น ในการรีเซ็ทสัญญาณ DRQ เช่นอุปกรณ์ภายนอกที่ขอ DMA ผ่านทางแชนแนลที่ 1 (DACK1) เมื่อได้รับสัญญาณจาก DACK1 แล้วก็จะรีเซ็ทสัญญาณ DRQ1 (เปลี่ยนจากลอจิก "1" เป็น "0")

DACK0-DACK3 (DMA Acknowledge 0-3;ขา B19,B17,B26 และ B15)

สัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ซึ่งตัวควบคุม DMA สร้างขึ้นเพื่อแสดงให้วงจรภายนอกที่ขอ DMA ทราบว่าการขอ DMA นั้นได้รับการตอบสนองแล้ว และตัวควบคุม DMA จะเข้าสู่ขบวนการ DMA เพื่อให้การส่งผ่านข้อมูลระหว่างอุปกรณ์ I/O ที่ขอ DMA กับหน่วยความจำเกิดขึ้นได้โดยตรง (คือไม่ต้องผ่าน CPU) โดยสัญญาณ DACK นี้จะแอคทีฟในแชนแนลใดก็ขึ้นอยู่กับว่าขบวนการ DMA ที่จะเกิดขึ้นนั้น เป็นการตอบสนองต่อการขอ DMA ในแชนแนลใด เช่นถ้าขบวนการ DMA ที่จะเกิดขึ้นนั้นเป็นการตอบสนองต่อการขอ DMA ในแชนแนลที่ 2 (DRQ2) สัญญาณ DACK2 ก็จะแอคทีฟ เป็นต้น

ดังที่ได้กล่าวแล้วว่าสัญญาณ DRQ0 นั้น จะไม่ถูกต่อออกมายังขาของสล็อต ดังนั้นวงจรอินเทอร์เฟสจึงไม่สามารถจะขอ DMA ผ่านทางแชนแนล 0 ได้ แต่สัญญาณ DACK0 จะถูกต่อออกมายังสล็อตด้วย (ขา B19) ทั้งนี้ก็เพื่อที่จะแสดงให้วงจรอินเทอร์เฟสต่างๆ ทราบว่าขบวนการ DMA ที่เกิดขึ้นในช่วงเวลาที่ DACK0 แอคทีฟนั้น เป็นขบวนการที่ใช้สำหรับการรีเฟรชหน่วยความจำที่เป็นไดนามิกแรม (Dynamic RAM) ซึ่งวงจรอินเทอร์เฟสที่ใช้หน่วยความจำประเภทนี้สามารถจะนำไปใช้ในการรีเฟรชไดนามิก(Dynamic RAM) ที่อยู่ในวงจรได้

โดยที่การรีเฟรชหน่วยความจำนั้นจะเกิดขึ้นในทุกๆ 15.12 usec. ดังนั้นสัญญาณ DACK0 นี้ก็จะเกิดแอคทีฟในทุกๆ 15.12 usec. ด้วย

AEN (Address Enable; ขา A11) :

สัญญาณนี้เป็นเอาท์พุทที่ใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นในช่วงเวลาที่สัญญาณ AEN แอคทีฟ (ลอจิก"1") นั้น เป็นบัสไซเคิลของขบวนการ DMA

สำหรับบนเมนบอร์ดของ IBM/PC นั้นจะใช้สัญญาณนี้ในการดิสเอเบิล (Disable) ตัวควบคุมบัส(Bus Controller) และจะใช้ดิสเอเบิลพอร์ท I/O ต่างๆ ที่ไม่เกี่ยวข้องกับขบวนการ DMA ที่เกิดขึ้นนี้ ที่จำเป็นต้องทำเช่นนี้ก็เพราะในช่วงเวลาขบวนการ DMA นั้น ตัวควบคุม DMA จะส่งแอดเดรสของหน่วยความจำออกมาบนบัสแอดเดรส และจะทำให้สัญญาณIOR หรือ IOW แอคทีฟด้วย ดังนั้นถ้าไม่ทำการดิสเอเบิลพอร์ท I/O ที่ไม่เกี่ยวข้องไว้ก็อาจจะทำให้พอร์ท I/O ที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรส (ซึ่งเป็นแอดเดรสของหน่วยความจำ) นั้น ทำการอ่านหรือส่งข้อมูลออกมาบนบัสข้อมูลทำให้เกิดความผิดพลาดขึ้นได้

T/C (Terminal Count; ขา B27) :

สัญญาณที่ถูกสร้างขึ้นจากการนำเอาสัญญาณเอาท์พุทที่ขา EOP ของตัวควบคุม DMA มากับลอจิก (โดยใช้เกท inverter) ทำให้สัญญาณ T/C นี้แอคทีฟที่ลอจิก "1" สำหรับสัญญาณนี้จะแอคทีฟเมื่อจำนวนไบท์ในการส่งผ่านข้อมูลของขบวนการ DMA ในแชนแนลใดแชนแนลหนึ่งครบตามจำนวนที่กำหนดไว้ โดยทั่วไปแล้วการรับสัญญาณที่จะถูกใช้ในการสิ้นสุดขบวนการ DMA ที่ทำการส่งผ่านข้อมูลเป็นบล็อก เนื่องจากสัญญาณที่จะแอคทีฟโดยไม่แสดงว่าเป็นสัญญาณของแชนแนลใดดังนั้นจึงต้องทำการนำสัญญาณ T/C นี้ผ่านเกท inverter แล้วนำไป OR กับสัญญาณ DACK เพื่อให้สามารถทราบได้ว่าสัญญาณ T/C ที่เกิดขึ้นนั้นเป็นสัญญาณของแชนแนลใด สำหรับในแชนแนลที่ 0 นั้น สัญญาณ T/C จะแอคทีฟในช่วงเวลาที่คงที่คือ ทุกๆ 990.804 msec. ซึ่งก็คือช่วงเวลาที่ใช้ในการรีเฟรชหน่วยความจำขนาด 64 Kbyte นั่นเอง

บัสของแหล่งจ่ายไฟของระบบ

+5Vdc (ขา B3 และ B29) :

ขาทั้งสองนี้ต่อกับแหล่งจ่ายไฟ DC +5V ของระบบ โดยจะมีค่าความเที่ยงตรง (Regulated) +/- 5% คืออยู่ในช่วง +4.75 ถึง +5.25 Vdc

+12 Vdc (ขา B9) :

ขานี้จะต่อกับแหล่งจ่ายไฟ DC +12V ของระบบโดยจะมีค่าความเที่ยงตรง (Regulated) +/- 5% คืออยู่ในช่วง +11.4 ถึง +12.6 Vdc

-5Vdc (ขา B5) :

ขาทั้งสองนี้ต่อกับแหล่งจ่ายไฟ DC -5V ของระบบ โดยจะมีค่าความเที่ยงตรง (Regulated) +/- 10% คืออยู่ในช่วง -5.50 ถึง -4.50 Vdc

-12 Vdc (ขา B7) :

ขานี้จะต่อกับแหล่งจ่ายไฟDC-12Vของระบบโดยจะมีค่าความเที่ยงตรง (Regulated) +/- 10% คืออยู่ในช่วง -13.2 ถึง -10.8 Vdc

GND (ขา B1, B10 และ B31) :

ขาทั้งสามนี้จะต่อเข้ากับกราวด์ (Ground) ของระบบ

การจัดสัญญาณบนสล็อตของ IBM/PC

สำหรับใน IBM PC/XT นั้นจะมีสล็อตสำหรับเชื่อมต่อกับวงจรภายนอกได้มากชิ้นคือ ใน IBM/PC จะทำการเพิ่มจำนวนสล็อตบนเมนบอร์ดขึ้นเป็น 8 สล็อต จากเดิมที่มีอยู่เพียง 5 สล็อตบน IBM/PC เพียงแต่สัญญาณต่างๆ ที่จะถูกส่งออกมายังขาของสล็อตที่ 8 นั้น จะถูกต่อผ่านวงจรขับกระแส (Buffer) ก่อน และในสล็อตที่ 8 นี้ขา B8 จะถูกใช้งานด้วย โดยจะถูกใช้เป็นขา CARD SLCTD (หรือ Card Selected) ซึ่งให้วงจรบนเมนบอร์ดทราบว่าคาร์ดที่อยู่บนสล็อตนี้ถูกเลือกใช้งานอยู่ ซึ่งจะทำให้ Driver บนเมนบอร์ดทำการอ่านหรือส่งข้อมูลไปยังสล็อตที่ 8

3.5 การออกแบบสร้าง

ใช้ 74HCT85 เป็นตัวเปรียบเทียบแอดเดรส จากสล็อตของเครื่อง IBM PC ถ้าช่วงแอดเดรสตรงกับค่าที่ตั้งไว้บนดิพสวิทช์ ก็จะหมายถึงมีการติดต่อกับบอร์ด DSP โดยเครื่อง IBM PC จะมองเห็นหน่วยความจำ RAM บนบอร์ด DSP ที่ละ 64kbyte (ใช้บัสแอดเดรส 16 เส้น) ซึ่งการเลือกว่าจะติดต่อกับ 64 kbyte ไหน ใน 256 kbyte ที่อยู่บนบอร์ด DSP สามารถเลือกได้โดยทำการเอาท์พอร์ท (outport) ออกไปตั้งค่ารีจีสเตอร์สถานะ (status register) บนบอร์ด DSP ในบิทที่ 0 และ 1 โดย

00 จะเป็น 64kbyte แรก
01 จะเป็น 64kbyte ที่สอง
10 จะเป็น 64kbyte ที่สาม
11 จะเป็น 64kbyte สุดท้าย

รวมเป็น 64 Kbytes x 4 = 256kbyte ซึ่งใช้ 74LS138 เป็นตัวถอดระหัสเลือกชุดที่ของหน่วยความจำ RAM บนบอร์ด และชิปไอซี (Integrated Circuit : IC) หน่วยความจำเป็นแบบสแตติก (Static Random Access Memory : SRAM) ซึ่งแต่ละตัวที่ใช้มีความจุหน่วยความจำข้อมูล 32kbyte (บัสแอดเดรส 15 เส้น) รวมทั้งหมด 8 ตัว ดังนั้นจึงใช้บัสแอดเดรสที่มาจากสล็อตของ IBM PC (16 เส้น) เส้นสุดท้ายเป็นตัวมัลติเพล็กซ์เลือกข้อมูลจาก SRAM 2 ตัว (SRAM 2 ตัว = 64 Kbytes)

เครื่องไมโครคอมพิวเตอร์ IBM PC สามารถติดต่อกับหน่วยความจำบนบอร์ด DSP ได้ 2 โมดคือ โมดไบต์(Byte) และโมดเวอร์ด(Word) โดยการเลือกเซ็ทที่หัวต่อแอดเดรส(header) คือ หน่วยความจำบนบอร์ดที่ทางไมโครคอมพิวเตอร์มองเห็น 64 Kbytes จะแบ่งออกได้เป็น 2 กลุ่ม คือ 32 Kbytes 8 บิทล่าง และ 32 Kbytes 8 บิทบน เมื่อเทียบกับที่ TMS320C25 มองเห็น โดยการอ่านและเขียนแบบไบต์ คือการอ่านและเขียนอย่างเรียงหน่วยความจำลงมาจาก 8 บิทล่างก่อน แล้วจึงอ่าน 8 บิทบนที่เหลือต่อกันไป ส่วนการอ่านและเขียนแบบเวอร์ด คือการอ่านข้อมูล 8 บิทล่างครั้งหนึ่ง แล้วจึงอ่าน 8 บิทบนอีกครั้งหนึ่งจนกระทั่งหมด

วิธีที่เครื่องไมโครคอมพิวเตอร์ IBM PC ใช้ในการติดต่อกับหน่วยความจำบนบอร์ด DSP ทำได้ทั้งวิธีการแม็พหน่วยความจำ (Memory Map) และ การใช้วิธี DMA (Direct Memory Access) ซึ่งทำได้โดยการตั้งค่าในบิทที่ 6 ของรีจีสเตอร์สถานะ(Status Register) โดยบิทที่ 6 นี้ถูกต่อออกไปที่ขาเลือกชุดสัญญาณของ 74LS157 ที่จะเลือกสัญญาณระหว่างสัญญาณ IOR กับ SMEMR และสัญญาณ IOW กับ SMEMW โดยช่วงของหน่วยความจำของเครื่องไมโครคอมพิวเตอร์ IBM PC ที่จะแม็พ(map) กับหน่วยความจำบนบอร์ด DSP สามารถเลือกได้โดยการตั้งค่าที่ดิพสวิทช์ SW001

การใช้พอร์ทของบอร์ด DSP จะใช้พอร์ตทั้งหมด 8 พอร์ท คือ

พอร์ท 0 : สงวนไว้
พอร์ท 1 : ใช้ในการส่งข้อมูลไปให้ TMS320C25
พอร์ท 2 : ใช้ในการเป็นรีจิสเตอร์สถานะ 8 บิทล่าง
พอร์ท 3 : ใช้ในการเป็นรีจีสเตอร์สถานะ 8 บิทบน
พอร์ท 4 : ใช้ในการควบคุม 8255
พอร์ท 5 : ใช้ในการรับข้อมูลจาก TMS320C25
พอร์ท 6 : ใช้ในการอ่านรีจิสเตอร์สถานะ 8 บิทล่าง
พอร์ท 7 : ใช้ในการอ่านรีจิสเตอร์สถานะ 8 บิทบน

โดยที่ทั้ง 8 พอร์ทนี้สามารถตั้งหมายเลขพอร์ทได้โดยการตั้งดิพสวิทช์ SW081 โดยจะต้องตั้งให้ไม่ตรงกับพอร์ทที่เครื่อง IBM PC ใช้งานอยู่

ความหมายของบิทต่างๆในรีจีสเตอร์สถานะ(Status Register)

บิท 0 : เลือกชุดของหน่วยความจำที่จะทำการติดต่อ
บิท 1 : เลือกชุดของหน่วยความจำที่จะทำการติดต่อ
บิท 2 : สงวนไว้
บิท 3 : สงวนไว้
บิท 4 : สงวนไว้
บิท 5 : สงวนไว้
บิท 6 : เลือกโมดการติดต่อ(Byte/Word)
บิท 7 : ใช้ในการรีเซ็ท(Reset) TMS320C25

บิท 8 : สงวนไว้

บิท 9 : เลือก CPU ที่จะทำการติดต่อกับหน่วยความจำบนบอร์ด DSP

บิท 10 : ใช้ในการขออินเทอร์รัพจาก TMS320C25

บิท 11 : ใช้ในการเคลียร์(clear) อินเทอร์รัพที่มาจาก TMS320C25

บิท 12 : สงวนไว้

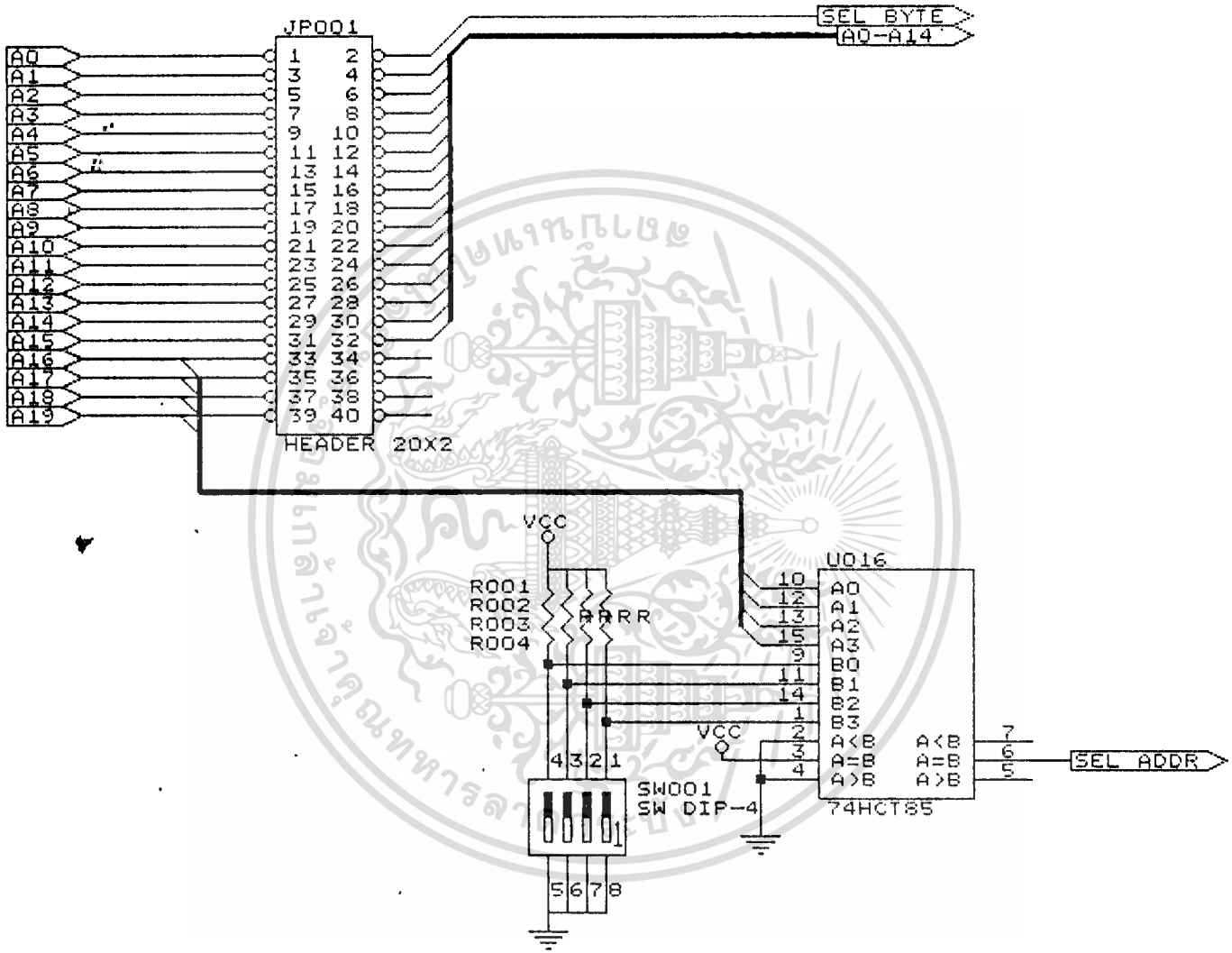
บิท 13 : สถานะของสัญญาณ HOLDA

บิท 14 : สถานะอินเทอร์รัพของ TMS320C25

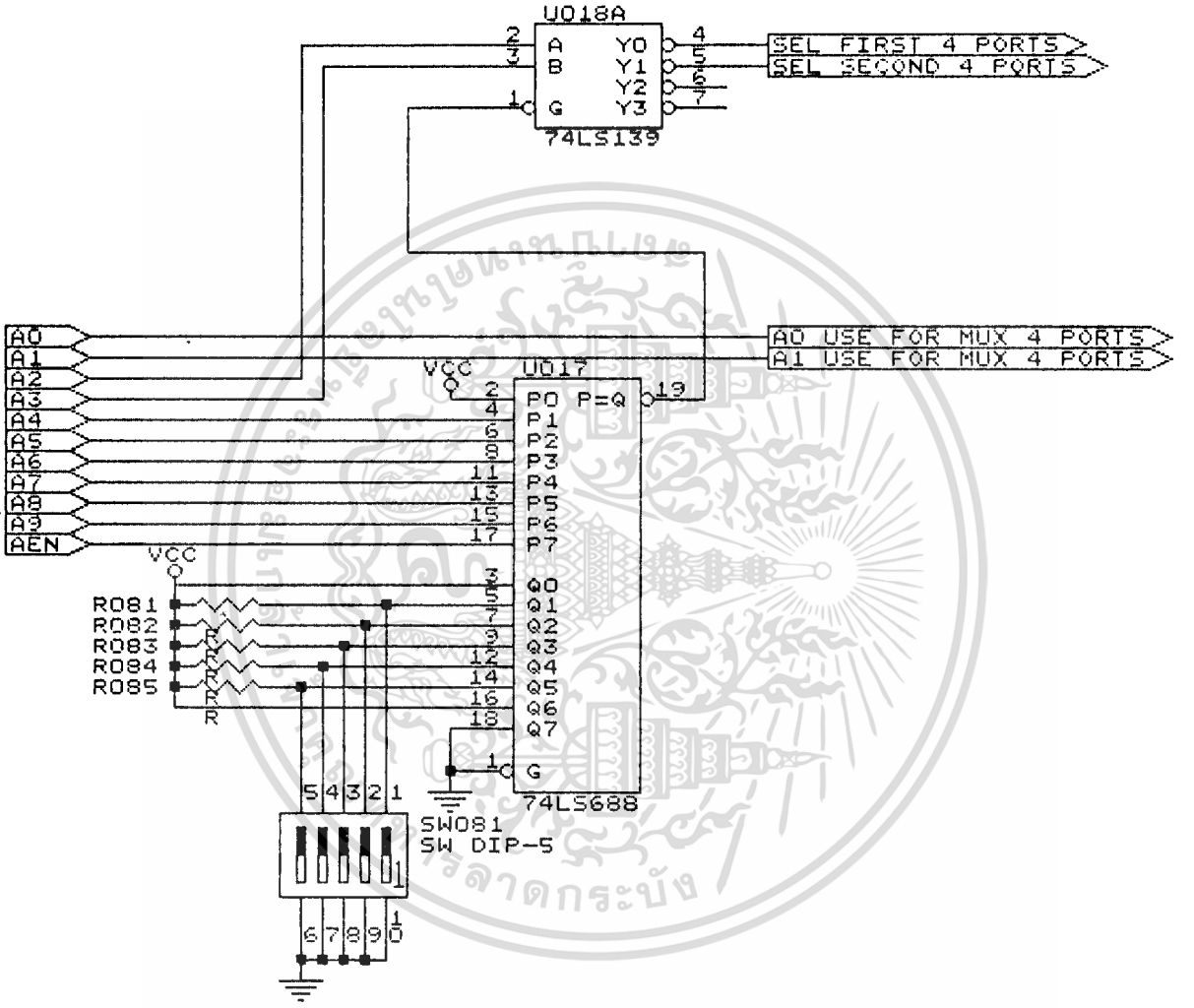
บิท 15 : สถานะอินเทอร์รัพของ IBM PC

การที่เครื่องไมโครคอมพิวเตอร์ IBM PC จะติดต่อกับหน่วยความจำบนบอร์ด DSP ทำได้โดยทำการตั้งค่าในบิทที่ 9 ของรีจีสเตอร์สถานะ โดยบิทที่ 9 นี้ถูกต่ออยู่กับขา HOLD ของ TMS320C25 เพื่อบอกให้ TMS320C25 รับรู้ว่าเครื่อง IBM PC จะทำการติดต่อกับหน่วยความจำบนบอร์ด จากนั้นเมื่อ TMS320C25 ได้รับสัญญาณHOLD TMS320C25 จะทำให้บัสแอดเดรส, บัสข้อมูล และ บัสควบคุมเป็นอิมพีแดนซ์สูง และจะเอาท์พุทสัญญาณ HOLDA ออกมา สัญญาณ HOLDA ที่ถูกเอาท์พุทออกมานี้จะถูกนำไปใช้ในการเลือก CPU ที่จะติดต่อกับหน่วยความจำบนบอร์ด DSP โดยจะไปทำการเลือกบัสแอดเดรส,บัสข้อมูล และบัสควบคุมที่ใช้ในการติดต่อกับหน่วยความจำบนบอร์ด DSP

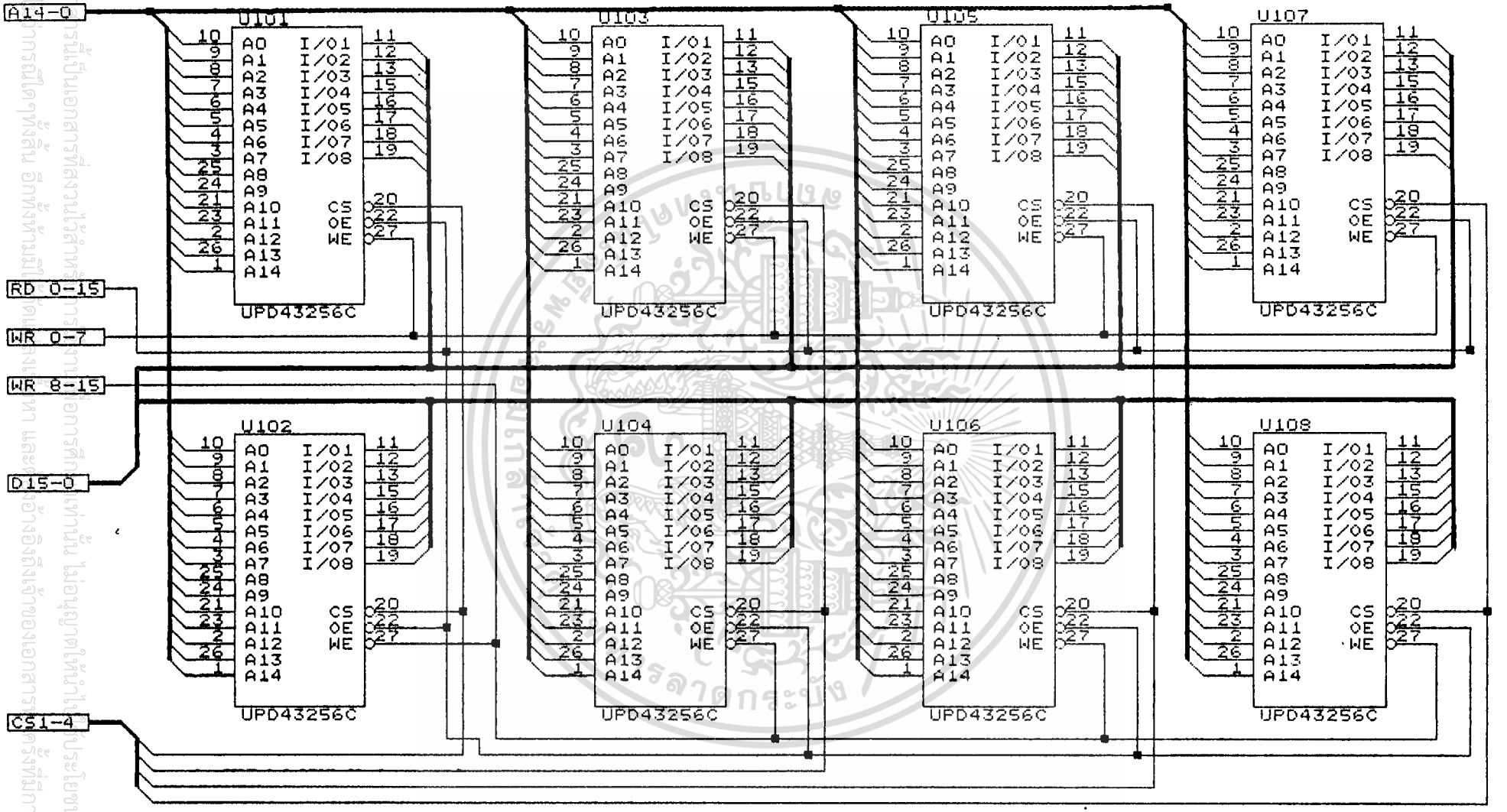
ส่วนในการเริ่มต้นใช้งานบอร์ด DSP นั้นจะต้องมีการตั่งค่าเริ่มต้น (initial value) ให้กับบอร์ดก่อน โดยจะต้องตั้งค่าให้พอร์ท a พอร์ท b และ พอร์ท c ของ 8522 บนบอร์ด DSP เป็นอินพุททั้งหมด นอกจากนี้ยังต้องตั้งค่าในรีจีสเตอร์สถานะให้เป็นไปตามโมดและวิธการในการติดต่อที่ต้องการกับบอร์ด DSP ก่อน จึงจะเริ่มใช้งานบอรดนี้ได้ และ ต้องทำการตั่งค่าเริ่มต้นนี้ทุกครั้งที่มีการรีเซ็ทเครื่อง IBM PC หรือเมื่อมีการเปิดเครื่องใหม่



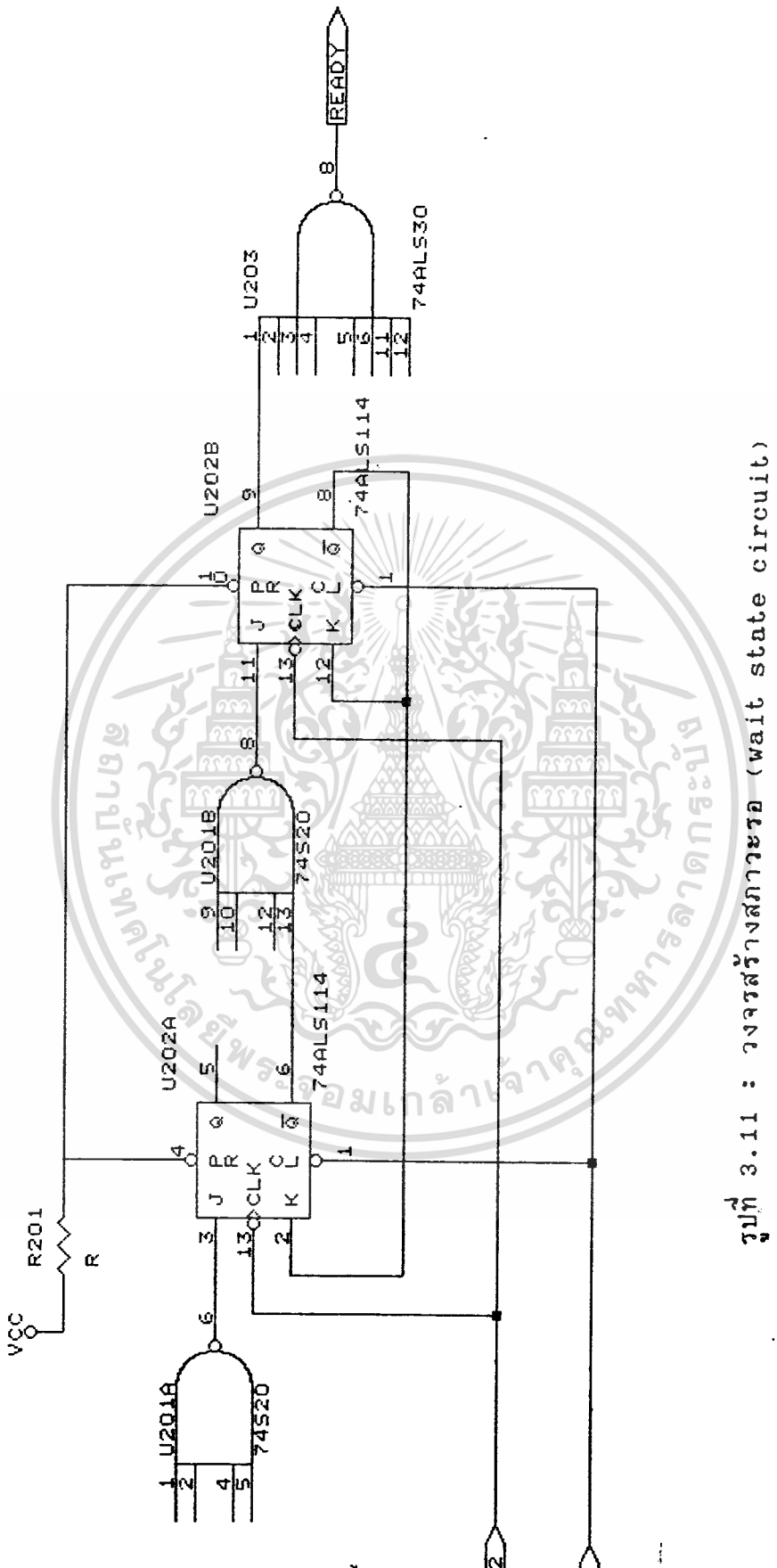
รูปที่ 3.8 : การดีโค้ดแอดเดรสโดยใช้ดิพสวิทช์



รูปที่ 3.9 : การดีโค้ดพอร์ทโดยใช้ดิพสวิทช์

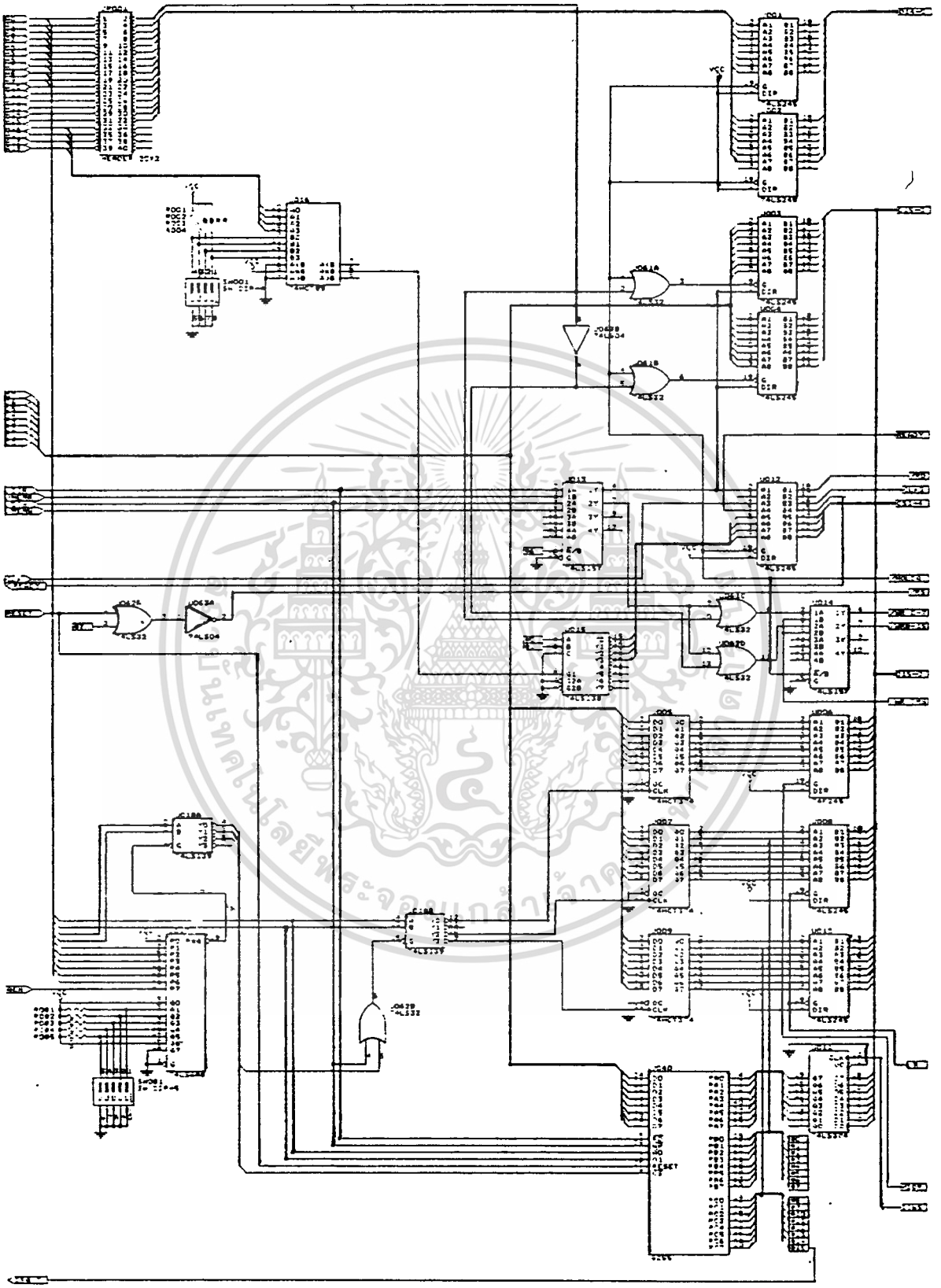


รูปที่ 3.10 : โมดูลหน่วยความจำสแตติก

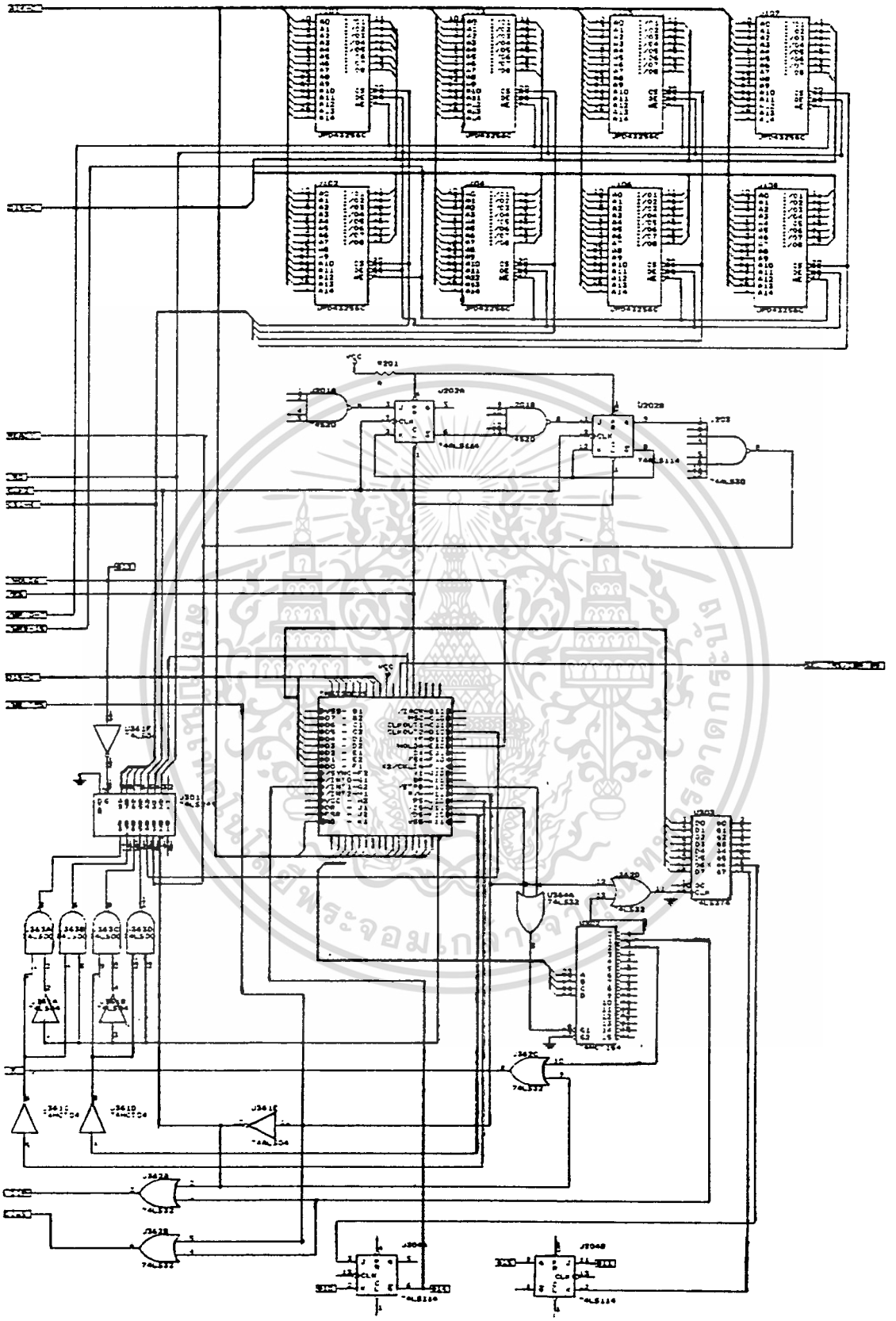


รูปที่ 3.11 : วงจรสร้างสภาวะรอ (wait state circuit)

รูปที่ 3.12 : วงจรส่วนที่ติดต่อกับ IBM PC



รูปที่ 3.13 : วงจรส่วนที่ติดต่อกับ TMS320C25



บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดสอบการทำงานของบอร์ด DSP

การทดลองทำโดยการดาวน์โหลด (down load) โปรแกรมทดสอบไปไว้บนหน่วยความจำบนบอร์ด DSP จากนั้นเอาท์พอร์ทสัญญาณไปทำการรีเซ็ท (Reset) TMS320C25 โดยการตั้งค่าในรีจีสเตอร์สถานะบทที่ 7 โดยโปรแกรมทดสอบจะให้ TMS320C25 ทำการแปลงฟาสท์ฟูเรียร์ทรานสฟอร์ม (Fast Fourier Transform : FFT) ขนาด 128 จุด และขนาด 256 จุด เปรียบเทียบกับการแปลงฟาสท์ฟูเรียร์ทรานสฟอร์มขนาด 128 จุด และขนาด 256 จุด บนเครื่องไมโครคอมพิวเตอร์ IBM PC/XT/AT ซึ่งผลการทดลองที่ได้จากการแปลงฟาสท์ฟูเรียร์ทรานสฟอร์มของทั้งเครื่องไมโครคอมพิวเตอร์ IBM PC/XT/AT และที่ได้จากการแปลงโดย TMS320C25 ได้ผลการทดลองมีค่าเท่ากัน แต่ TMS320C25 ทำงานได้เร็วกว่ามาก ในการทดสอบใช้ภาษาซีในการเขียนโปรแกรมเพื่อทดลองการแปลงฟาสท์ฟูเรียร์ทรานสฟอร์ม (FFT) บนเครื่องไมโครคอมพิวเตอร์ IBM PC/XT/AT โดยโปรแกรมสามารถรับอินพุทจำนวนจุด(เป็นค่ายกกำลังของ 2)ในการแปลงได้ ส่วนในการทดลองกับ TMS320C25 ใช้โปรแกรมที่เขียนเป็นภาษาแอสเซมบลี้(assembly) ของ TMS320C25 ซึ่งมีจำนวนจุดในการแปลงฟาสท์ฟูเรียร์ทรานสฟอร์มคงที่ คือการแปลงขนาด 128 จุด และขนาด 256 จุด โดยโปรแกรมเฉพาะที่ใช้ในการทดสอบการแปลงฟาสท์ฟูเรียร์ทรานสฟอร์มของทั้งเครื่องไมโครคอมพิวเตอร์ IBM PC/XT/AT และของ TMS320C25 มีรายละเอียดอยู่ในหน้าถัดไป

```
/*************************************************************************/
/*                                                                       */
/*                    Program test one-dimensional                       */
/*                    Fast Fourier Transform (FFT)                       */
/*                                                                       */
/*************************************************************************/

/*
-------------------------------------------------------------------------
Module Name: One-dimensional, radix 2 FFT test program in C
*/
#define NBITS       32 /* NBITs is maximum bits in bit-reversal
                       (adjust to suit computer addressing) */
struct complx {        /* definition of complex data type: real, imaginary */
    float real;
    float imag;
};

/*
-------------------------------------------------------------------------
Purpose:
To test one-dimensional FFT in C
(Note; input data in array x is replaced by its DFT or inverse DFT).

Method:
Loads a "chirp" waveform (cosine, with linearly increasinng frequency) into test
array, computes Forward FFT, then inverse FFt.
Forward FFT results in spectrum with terms of same order of magnitude, while
inverse FFT priduces original chirp wave.
*/

main()
{
    int             j,m,n,nhlf,is;
    double          twopi,arg,cos();
    float           an,t,freq,scale;
    union {
        float           x[2000];
        struct complx    cx[1000];
    }       u;
    /* real array x and complex array cx share same space */
    printf("input m = ");
    scanf("%d",&m);
    n    = power2(m);
    nhlf = n/2+1;
    an   = n;
    twopi= 6.283185307179586232;
    /* set constants */

    for (j=0;j<n;j++){
        t    = j;
        freq = t*0.25;
        arg  = twopi*freq*t/an;
```

```
        u.x[j]= cos(arg);
    }
    /* load chirped-simusoidal test signal into real array x */

    is    = -1;
    scale = 1.01;
    printf("\nStart calculate FFT\n");
    fftr(u.x, m, is, scale);

    /* compute "forward"FFT */

    printf(" Complex result of Forward FFT of chirp wave\n");
    for (j=0;j<nhlf;j++)
        printf(" j%7d, cx %14g %14g\n", j, u.cx[j].real, u.cx[j].imag);
    printf("End calculate FFT\n\n");

    is    = 1;
    scale = 1.0/an;
    printf("Start calculate IFFT\n");
    fftr(u.x, m, is, scale);

    /* compute "inverse" FFT */

    printf("\n Real result of Inverse FFT:chirp wave\n");
    for (j=0;j<n;j++)
        printf(" j%7d, x %14g\n", j, u.x[j]);
    printf("End calculate IFFT\n\n");

    exit(0);
}

/*
-------------------------------------------------------------------------------

Function Name: power2 (n)

Purpose:
To compute 2 raised to the power n
*/
power2 (n)
int n;
{
    int  i,p;

    p=1;
    for (i=0; i<n; ++i)
        p=p*2;
        return (p);
}

/*
-------------------------------------------------------------------------------

Function Name: fft (x, m, is, scale)
```

```
Purposr:
Computes DFT of complex array x, by the FFT algorithm
(note: input data in array x is replaced by its DFT or inverse DFT).

Arguments:
m specifies array size, thus: 2* *m elements,
is= -1 or +1 is sign of complex exponential in DFT (forward, or inverse),
scale is a real value to scale the result — e.g. scale=1.0 or 1.0/2.0**m.

Method:
First part is bit-reversed permutation using recursive algorithm, which
increments a reversed index corresponding to each bit position.
Second part is radix 2 FFT computation, which
optimizes use of weighting factor w, generated recursively.
*/

fft (x, m, is, scale)
struct complx x[];
int m, is;
float scale;
{
    int             n,ifwd,irev,pass,seg,j,k,
                    span,step,ira[NBITS], nr[NBITS];
    double          pi,fract,sin(),cos();
    struct complx   d,w,temp;

    if (m>NBITS)
        return (NBITS); /* error return, if m too large */

    pi = 3.1415926535897931;

    n  = power2 (m);
    for (j=0;j<m;j++){
        ira[j] = 0;
        nr[j]  = power2(j);
    }
    /*
     * revered index sets, one for each bit-position, initialized
     */
    for (ifwd=0;ifwd<n;ifwd++){
        irev=ira[m-1];
        if (irev>ifwd){
                temp.real    = x[ifwd].real;
                temp.imag    = x[ifwd].imag;
                x[ifwd].real = x[irev].real;
                x[ifwd].imag = x[irev].imag;
                x[irev].real = temp.real;
                x[irev].imag = temp.imag;
        }

        /* reversed index pair swapped */

        j=m-1;
```

```
    while ((j>0)&&(ira[j]>=nr[j]))
        j— ;
    /*
     * alternate increment of ira [j], must go back one
     * bit-position
     */
    ira[j] = ira[j]+nr[j];

    if (j<m-1){
    for (k=j;k<m-1;k++) {
        ira[k+1]=ira[k];
    }
    /* work forward through revesed index bit set */
    }
}
/*
 * array x is now in bit-reversed order; m computing passes
 * follow
 */
for (pass=0;pass<m;pass++){
    span=power2 (pass);
    step=2*span;
    /*
     * span between elements in pair, step to next pair with
     * same w
     */
    w.real = 1.0;
    w.imag = 0.0;
    fract  = pi/span;
    d.real = cos(fract);
    d.imag = sin(fract);
    if (is<0)
        d.imag=-d.imag;
    /* starting phase-adjusting weight w,modifier d */

    for (seg=0;seg<span;seg++){
          for (j=seg;j<n;){
               k=j+span;
               temp.real = x[k].real*w.real - x[k].imag*w.imag;
               temp.imag = x[k].real*w.imag + x[k].imag*w.real;
               x[k].real = x[j].real - temp.real;
               x[k].imag = x[j].imag - temp.imag;
               x[j].real = x[j].real + temp.real;
               x[j].imag = x[j].imag + temp.imag;
               /*
                *inner loop arithmetic—"butterfly"
                *two-point transforms
                */
               j=j+step;
          }
          temp.real = w.real*d.real - w.imag*d.imag;
          w.imag    = w.real*d.imag + w.imag*d.real;
          w.real    = temp.real;
```

```
            /*
             *recursive modification of phase adjusting weight w
             */
        }
    }
    if (scale !=1.0){
       for (j=0;j<n;j++){
          x[j].real = x[j].real*scale;
          x[j].imag = x[j].imag*scale;
       }
    }
    return (0);
}

/*
-------------------------------------------------------------------------

Function Name: fftr (x, m, is, scale)

Purpose:
Computes half-length complex DFT of real array x, by the FFT algorithm or
real inverse DFT of half-length complex array x
(note: input data in array x is replaced by its DFT or inverse DFT).

Arguments:
m specifies array size, thus : 2* *m "notional" elements,
Real data assumed packed as alternate real and imaginary velues of an
overlayed complex array; number of real elements is 2* *m (+2dummies);
number of complex elements is 2* *(m-1)+1.
is = -1 or +1 is sign of comples exponential in DFT (forward, of inverse),and
direction (-1=real-to-complex, +1=complex-to-real).
scale is a real value to scale the result -- e.g.scale=1.0 or 1.0/2.0* *m.

Method:
Uses "odd-even" unscramble/scramble algorithm, and calls to complex FFT.
*/

fftr (x, m, is, scale)
struct complx x[];
int m, is;
float scale;
{
    int             n,j,k,incnt,mh;
    double          pi,fract,sin(),cos();
    struct complx    d,w,tempa,tempb,tempc;

    if (m>NBITS)
          return(NBITS); /* error return, if m too large */

    pi = 3.1415926535897931;
    mh = m-1;
    n  = power2(mh);
    incnt  = n/2+1;
    w.real = 1.0;
```

```
w.imag = 0.0;
fract  = pi/n;
d.real = cos(fract);
d.imag = sin(fract);
/* starting phase-adjusting weight w, modifier d, for
   (un)scramble */

if (is<0) {
/* (real-to-complex FFT follows—do half-length complex FFT first) */
    d.imag=-d.imag;
    fft(x,mh,is,scale);
    x[n].real = x[0].real;
    x[n].imag = x[0].imag;
    for (j=0;j<incnt;j++){
        k=n-j;
        tempa.real = (x[j].real+x[k].real)*0.5;
        tempa.imag = (x[j].imag-x[k].imag)*0.5;

        tempb.real = x[j].imag+x[k].imag;
        tempb.imag =-x[j].real+x[k].real;
        tempc.real = (tempb.real*w.real-tempb.imag*w.imag)*0.5;
        tempc.imag = (tempb.real*w.imag+tempb.imag*w.real)*0.5;

        x[j].real  = tempa.real+tempc.real;
        x[j].imag  = tempa.imag+tempc.imag;

        x[k].real  = tempa.real-tempc.real;
        x[k].imag  =-(tempa.imag-tempc.imag);

        tempa.real = w.real*d.real-w.imag*d.imag;
        w.imag     = w.real*d.imag+w.imag*d.real;
        w.real     = tempa.real;
        /*
         *recursive modification of phase adjusting weight w,
         *elements unscrambled
         */
    }
}
else
{
  /* complex-to-real FFT follows */
    for (j=0; j<incnt;j++){
    k=n-j;
    tempa.real = x[j].real + x[k].real;
    tempa.imag = x[j].imag - x[k].imag;

    tempb.real = x[j].real - x[k].real;
    tempb.imag = x[j].imag + x[k].imag;

    tempc.real = tempb.real*w.imag + tempb.imag*w.real;
    tempc.imag = tempb.real*w.real - tempb.imag*w.imag;

     x[j].real = tempa.real - tempc.real;
```

```
            x[j].imag = tempa.imag + tempc.imag;

            x[k].real = tempa.real + tempc.real;
            x[k].imag =-tempa.imag + tempc.imag;

               tempa.real = w.real*d.real - w.imag*d.imag;
               w.imag     = w.real*d.imag + w.imag*d.real;
               w.real     = tempa.real;
               /*
                *recursive modification of phase adjusting weight w,
                *elements unscrambled
                */
            }
            fft (x, mh, is, scale);
            /*half-length complex fft finishes complex-to-real fft */
        }
        return(0);
}
```

```
        IDT 'FFT2'
*
*    COOLEY-TUKEY 128-POINT, RADIX-2, DIF FFT PROGRAM FOR THE TMS32025.
*
*       SINGLE FFT BUTTERFLY.
*       REAL INPUT DATA, STORED IN PAGE 4 OF BLOCK B0.
*       FFT COMPUTATION IS DONE IN PAGES 6, 7 OF BLOCK B1 (COMPLEX NUMBERS).
*       USES TABLE LOOKUP (FROM PROGRAM MEMORY) OF THE TWIDDLE FACTORS.
*       INTERMEDIATE VALUES ARE SCALED BY .5 AT EACH STAGE SO AS TO PREVENT
*        THE POSSIBILITY OF OVERFLOW.
*       NO EXTERNAL RAM IS USED.
*       THE MAGNITUDE OF THE FFT IS COMPUTED AND NORMALIZED SO THAT ITS MAXIMUM
*       VALUE HAS A NONZERO BIT AFTER THE BINARY POINT.
*
*******************************************************************************
*
* N IS THE SIZE OF THE TRANSFORM.  N = 2**M.
*
N EQU 128
M EQU 7
*
XIN     EQU 512         * LOCATION OF REAL INPUT DATA
XOUT    EQU 640         * LOCATION OF REAL OUTPUT DATA
XFFT    EQU 768         * LOCATION OF COMPLEX DATA FOR THE FFT
*
* BLOCK B2 DATA MEMORY ALLOCATION (DP = 0 WILL ALWAYS POINT TO B2).
*
XT      EQU 96          * TEMPORARY - REAL PART
YT      EQU 97          * TEMPORARY - IMAGINARY PART
I       EQU 98          * 1ST INDEX
IA      EQU 99          * INDEX TO TWIDDLE FACTORS
IE      EQU 100         * INCREMENT TO IA
HOLDN   EQU 101         * CONTAINS VALUE N
QUARTN  EQU 102         * CONTAINS VALUE N/4
N1      EQU 103         * INCREMENT TO I.
N2      EQU 104         * SEPARATION OF I AND L
J       EQU 105         * LOOP COUNTER
K       EQU 106         * BIT REVERSAL INDEX COUNTER
ONE     EQU 107         * CONTAINS VALUE 1
IADDR   EQU 108         * CONTAINS INPUT
SINTBL  EQU 109         * SINE TABLE POINTER
SIN     EQU 110         * SINE LOCATION
COS     EQU 111         * COSINE LOCATION
MAX     EQU 112         * MAX VALUE OF THE SQR MAGNITUDE OF FFT
*
*
* BEGIN PROGRAM MEMORY SECTION.
*
*******************************************************************************
*
        AORG 0
RSVECT  B   32
        AORG 32
```

```
INIT    LDPK 0              * ALWAYS POINT TO B2 FOR TEMP STORAGE
        CNFD
        SOVM
        SSXM                * 32010 ARITHMETIC
        SPM  1              * SHIFT PRODUCT LEFT BY 1
        LACK 1
        SACL ONE
        SACL IE             * INITIALIZE IE = 1
        LALK N
        SACL HOLDN          * HOLDN = N
        SACL N2             * INITIALIZE N2 = N
        LAC  HOLDN,14
        SACH QUARTN         * QUARTN = N/4
        LRLK AR3,XFFT       * ADDRESS OF COMPLEX INPUT DATA
        SAR  AR3,IADDR      * STORE ON PAGE 0
        LARP 2
        LARK AR1,M-1        * AR1 CONTAINS K COUNTER
*
*   READ IN 128 REAL POINTS
*
        LRLK    AR2,XIN
        RPTK    127
        IN      *+,PA0
*
*   INITIALIZE FFT RAM
*
        LRLK    AR2,XFFT
        ZAC
        SACL    MAX
        RPTK    255
        SACL    *+
*
*   MOVE REAL DATA FROM INPUT LOCATIONS TO COMPLEX FFT LOCATIONS
*
        LRLK    AR2,XFFT
        LARK    AR0,2
        RPTK    127
        BLKD    XIN,*0+
*
*   MOVE SQUARED MAGNITUDE OF PREVIOUS COMPUTATION TO OUTPUT LOCATIONS
*
        LRLK    AR2,XOUT
        RPTK    127
        BLKD    XFFT,*+
*
*   FFT COMPUTATION
*
KLOOP       LAC  N2,15
            SACH N1,1       * N1 = N2
            SACH N2         * N2 = N2/2
            ZAC
            SACL IA         * IA = 0
            SACL J          * J = 0
```

```
            LAR  AR2,N2       * AR2 CONTAINS J VALUE
            MAR  *-,AR3       * START AT N2-1
*
           LALK   SINE
           SACL   SINTBL      * SINE TABLE BASE ADDRESS
*
JLOOP         LAC  J,1
              SACL I           * I = J  (DATA ORGANIZED AS REAL VALUE FOLLOWED
*                                BY IMAGINARY SO THAT ADDRESS I IS 2 TIMES J).
*
ILOOP            LAR  AR0,I
                 LAR  AR3,IADDR        * LOAD INPUT BASE ADDRESS
                 MAR  *0+              * AR3 = I + IADDR
                 LAR  AR0,N1           * ADD N2*2 (N1 = N2*2)
                 LAC  *0+,15           * LOAD (1/2)XI, POINT TO XL
*                                        (L = I + N2)
                 SUB  *,15             * XT = (1/2)(XI - XL)
                 SACH XT               * STORE XT ON PAGE 0
                 ADDH *0-              * XI = (1/2)(XI + XL), POINT TO XI
                 SACH *+               * STORE XI, POINT TO YI
                 LAC  *0+,15           * LOAD (1/2)YI, POINT TO YL
                 SUB  *,15             * YT = (1/2)(YI - YL)
                 SACH YT
                 ADDH *0-              * YI = (1/2)(YI + YL), POINT TO YI
                 SACH *0+              * STORE YI, POINT TO YL
*
                LAC    SINTBL
                TBLR   SIN             * READ IN SINE
                ADD    QUARTR
                TBLR   COS             * READ IN COSINE
*
                LT     COS
                MPY    YT              * YT*COS--->P
                LTP    SIN
                MPY    XT              * XT*SIN--->P
                SPAC
                SACH   *-              * YL = XT*COS - XT*SIN
                MPY    YT              * YT*SIN--->P
                LTP    COS
                MPY    XT              * XT*COS--->P
                APAC
                SACH   *               * XL = XT*COS + YT*SIN
*
*
* ADD INCREMENT FOR NEXT LOOP.
*
                 LAC  I
                 ADD  N1,1             * I = I + N1
                 SACL I
                 SUB  HOLDN,1          * WHILE I < N
                 BLZ  ILOOP
*
               LAC  SINTBL           * INDEX SINTBL POINTER BY IE
```

```
                ADD  IE
                SACL SINTBL
*
                LAC  J
                ADD  ONE                * J = J + 1
                SACL J
                LARP 2
                BANZ JLOOP,*-,AR3
*
            LAC  IE,1
            SACL IE                     * IE = 2 * IE
            LARP 1
            BANZ KLOOP,*-,AR2
*
* DIGIT REVERSE COUNTER FOR RADIX-2 FFT COMPUTATION.
*
DRC2    ZAC
        SACL I
        SACL J
        LAR  AR0,IADDR
        LARP 4
        LAR  AR4,HOLDN                  * FOR I = 0 TO N-2
        MAR  *-
        MAR  *-,AR2
DRLOOP          SUB  J                  * IF I < J, THEN SWAP
                BGEZ NOSWAP
*
                        LAR  AR2,J
                        MAR  *0+,AR3    * J = J + IADDR
                        LAR  AR3,I
                        MAR  *0+,AR2    * I = I + IADDR
* SWAP I AND L VALUES.
                        ZALH *,AR3
                        ADDS *
                        SACH *+,0,AR2   * X(I) = X(J)
                        SACL *+         * X(J) = X(I)
*
                        ZALH *,AR3
                        ADDS *
                        SACH *,0,AR2    * Y(I) = Y(J)
                        SACL *,0,AR3    * Y(J) = Y(I)
NOSWAP          LAC  HOLDN
                SACL K                          * K = N
INLOOP                  LAC  J
                        SUB  K                  * IF J >= K THEN
                        BLZ  OUTLOP,*,AR4
                                SACL J          * J = J - K
                                LAC  K,15
                                SACH K          * K = K/2.
                                B    INLOOP
OUTLOP          ADD  K,1
                SACL J                          * L = L + J
                LAC  I
```

```
                ADD  ONE,1
                SACL I                          * INCREMENT I
                BANZ DRLOOP,*-,AR2
*
*
*   COMPUTE THE MAXIMUM VALUE OF THE SQUARED MAGNITUDE OF FFT
*
        LRLK    AR2,XFFT
        LARK    AR3,127
*
LOOP1   ZAC
        MPYK    0
        SQRA    *+                      * X(I)**2
        SQRA    *+,AR3                  * Y(I)**2
        APAC
        SACH    IADDR                   * IADDR = X(I)**2 + Y(I)**2
        SUBH    MAX
        BLEZ    CONT
        LAC     IADDR
        SACL    MAX
CONT    BANZ    LOOP1,*-,AR2
*
*   NORMALIZE THE MAX VALUE TO FIND EXPONENT
*
        LARK    AR2,0
        ZALH    MAX
        RPTK    14
        NORM                            * AR2 CONTAINS EXPONENT
        SAR     AR2,I                   * STORE EXPONENT IN I
*
*   COMPUTE SQUARED MAGNITUDE OF FFT
*
        LRLK    AR2,XFFT
        LRLK    AR1,XFFT
        LARK    AR3,127
*
LOOP2   ZAC
        MPYK    0
        SQRA    *+                      * X(I)**2
        SQRA    *+,AR1                  * Y(I)**2
        APAC
        SFR
        RPT     I
        SFL                             * NORMALIZE RESULT
        SACH    *+,0,AR3                * XOUT(I) = X(I)**2 + Y(I)**2
        BANZ    LOOP2,*-,AR2
*
*   OUTPUT SQUARED MAGNITUDE
*
        LRLK    AR2,XFFT
        RPTK    127
        OUT     *+,PA2
*
```

```
*  FFT COMPLETE.
*
WHOA    B WHOA
*
* COEFFICIENT TABLE (SIZE OF TABLE IS 3N/4).
*
SINE EQU $
      DATA >0
      DATA >648
      DATA >C8C
      DATA >12C8
      DATA >18F9
      DATA >1F1A
      DATA >2528
      DATA >2B1F
      DATA >30FC
      DATA >36BA
      DATA >3C57
      DATA >41CE
      DATA >471D
      DATA >4C40
      DATA >5134
      DATA >55F6
      DATA >5A82
      DATA >5ED7
      DATA >62F2
      DATA >66D0
      DATA >6A6E
      DATA >6DCA
      DATA >70E3
      DATA >73B6
      DATA >7642
      DATA >7885
      DATA >7A7D
      DATA >7C2A
      DATA >7D8A
      DATA >7E9D
      DATA >7F62
      DATA >7FD9
COSINE EQU $
      DATA >7FFF
      DATA >7FD9
      DATA >7F62
      DATA >7E9D
      DATA >7D8A
      DATA >7C2A
      DATA >7A7D
      DATA >7885
      DATA >7642
      DATA >73B6
      DATA >70E3
      DATA >6DCA
      DATA >6A6E
```

```
DATA >66D0
DATA >62F2
DATA >5ED7
DATA >5A82
DATA >55F6
DATA >5134
DATA >4C40
DATA >471D
DATA >41CE
DATA >3C57
DATA >36BA
DATA >30FC
DATA >2B1F
DATA >2528
DATA >1F1A
DATA >18F9
DATA >12C8
DATA >C8C
DATA >648
DATA >0
DATA >F9B8
DATA >F374
DATA >ED38
DATA >E707
DATA >E0E6
DATA >DAD8
DATA >D4E1
DATA >CF04
DATA >C946
DATA >C3A9
DATA >BE32
DATA >B8E3
DATA >B3C0
DATA >AECC
DATA >AA0A
DATA >A57E
DATA >A129
DATA >9D0E
DATA >9930
DATA >9592
DATA >9236
DATA >8F1D
DATA >8C4A
DATA >89BE
DATA >877B
DATA >8583
DATA >83D6
DATA >8276
DATA >8163
DATA >809E
DATA >8027
END
```

```
        IDT 'FFT2'
*
*    COOLEY-TUKEY 256-POINT, RADIX-2, DIF FFT PROGRAM FOR THE TMS32025.
*
*       SINGLE FFT BUTTERFLY.
*       COMPLEX INPUT DATA, STORED AS X(I), Y(I), X(I+1), Y(I+1), ...
*       USES TABLE LOOKUP (FROM EXTERNAL DATA MEMORY) OF THE TWIDDLE FACTORS.
*       INTERMEDIATE VALUES ARE SCALED BY .5 AT EACH STAGE SO AS TO PREVENT
*        THE POSSIBILITY OF OVERFLOW.
*
*******************************************************************************
*
* N IS THE SIZE OF THE TRANSFORM.  N = 2**M.
*
N EQU 256
M EQU 8
*
INPUT   EQU 512         * LOCATION OF COMPLEX INPUT DATA IN INTERNAL DATA MEM
TABLE   EQU 1024        * LOCATION OF COEFFICIENT TABLE IN EXTERNAL DATA MEM
*
* BLOCK B2 DATA MEMORY ALLOCATION (DP = 0 WILL ALWAYS POINT TO B2).
*
XT      EQU 96          * TEMPORARY - REAL PART
YT      EQU 97          * TEMPORARY - IMAGINARY PART
I       EQU 98          * 1ST INDEX
IA      EQU 99          * INDEX TO TWIDDLE FACTORS
IE      EQU 100         * INCREMENT TO IA
HOLDN   EQU 101         * CONTAINS VALUE N
QUARTN  EQU 102         * CONTAINS VALUE N/4
N1      EQU 103         * INCREMENT TO I.
N2      EQU 104         * SEPARATION OF I AND L
J       EQU 105         * LOOP COUNTER
K       EQU 106         * BIT REVERSAL INDEX COUNTER
ONE     EQU 107         * CONTAINS VALUE 1
IADDR   EQU 108         * CONTAINS INPUT
COSTBL  EQU 109         * COSTBL = TABLE + N/4
*
*
* BEGIN PROGRAM MEMORY SECTION.
*
*******************************************************************************
*
        AORG 0
RSVECT  B   32
        AORG 32
INIT    LDPK 0              * ALWAYS POINT TO B2 FOR TEMP STORAGE
        SOVM
        SSXM                * 32010 ARITHMETIC
        SPM  1              * SHIFT PRODUCT LEFT BY 1
        LACK 1
        SACL ONE
        SACL IE             * INITIALIZE IE = 1
```

```
          LALK N
          SACL HOLDN          * HOLDN = N
          SACL N2             * INITIALIZE N2 = N
          LAC  HOLDN,14
          SACH QUARTN         * QUARTN = N/4
          LRLK AR3,INPUT      * ADDRESS OF COMPLEX INPUT DATA
          SAR  AR3,IADDR      * STORE ON PAGE 0
          LRLK AR4,TABLE      * ADDRESS OF SINE TABLE
          LARP 4
          RPTK 191            * MOVE 192 COEFFICIENTS
          BLKP SINE,*+
          LRLK AR4,TABLE      * ADDRESS OF SINE TABLE
          LAR  AR0,QUARTN
          MAR  *0+,AR2        * COSTBL = TABLE + N/4, point to J counter
          SAR  AR4,COSTBL
          LARK AR1,M-1        * AR1 CONTAINS K COUNTER
*
*   READ IN 256 COMPLEX POINTS
*
        LRLK    AR2,INPUT
        RPTK    255
        IN      *+,PA0
        RPTK    255
        IN      *+,PA0
*
*   FFT COMPUTATION
*
KLOOP       LAC  N2,15
            SACH N1,1       * N1 = N2
            SACH N2         * N2 = N2/2
            ZAC
            SACL IA         * IA = 0
            SACL J          * J = 0
            LAR  AR2,N2     * AR2 CONTAINS J VALUE
            MAR  *-,AR3     * START AT N2-1
*
            LAR  AR4,COSTBL * COSINE TABLE BASE ADDRESS
*
JLOOP          LAC  J,1
               SACL I       * I = J  (DATA ORGANIZED AS REAL VALUE FOLLOWED
*                             BY IMAGINARY SO THAT ADDRESS I IS 2 TIMES J).
*
ILOOP             LAR  AR0,I
                  LAR  AR3,IADDR      * LOAD INPUT BASE ADDRESS
                  MAR  *0+            * AR3 = I + IADDR
                  LAR  AR0,N1         * ADD N2*2 (N1 = N2*2)
                  LAC  *0+,15         * LOAD (1/2)XI, POINT TO XL
*                                       (L = I + N2)
                  SUB  *,15           * XT = (1/2)(XI - XL)
                  SACH XT             * STORE XT ON PAGE 0
                  ADDH *0-            * XI = (1/2)(XI + XL), POINT TO XI
                  SACH *+             * STORE XI, POINT TO YI
                  LAC  *0+,15         * LOAD (1/2)YI, POINT TO YL
```

```
                SUB  *,15          * YT = (1/2)(YI - YL)
                SACH YT
                ADDH *0-           * YI = (1/2)(YI + YL), POINT TO YI
                SACH *0+,0,AR4     * STORE YI, POINT TO YL
*                                    AR4 POINTS TO COSTBL
                LAR  AR0,QUARTR
                LT   *0-           * LOAD T WITH COS, POINT TO SIN
                MPY  YT
                LTP  *0+,AR3       * ACC <-- C*YT, POINT TO YL
                MPY  XT
                SPAC               * YL = C*YT - S*XT
                SACH *-,0,AR4      * STORE YL
                MPY  YT
                LTP  *,AR3         * ACC <-- S*YT, POINT TO XL
                MPY  XT
                APAC               * XL = C*XT + S*YT
                SACH *             * STORE XL
*
*
* ADD INCREMENT FOR NEXT LOOP.
*
                LAC  I
                ADD  N1,1          * I = I + N1
                SACL I
                SUB  HOLDN,1       * WHILE I < N
                BLZ  ILOOP
*
              LAR  AR0,IE          * INDEX COSTBL POINTER BY IE
              LARP 4
              MAR  *0+,AR2
              LAC  J
              ADD  ONE             * J = J + 1
              SACL J
              BANZ JLOOP,*-,AR3
*
           LAC  IE,1
           SACL IE                 * IE = 2 * IE
           LARP 1
           BANZ KLOOP,*-,AR2
*
* DIGIT REVERSE COUNTER FOR RADIX-2 FFT COMPUTATION.
*
DRC2    ZAC
        SACL I
        SACL J
        LAR  AR0,IADDR
        LARP 4
        LAR  AR4,HOLDN             * FOR I = 0 TO N-2
        MAR  *-
        MAR  *-,AR2
DRLOOP          SUB  J             * IF I < J, THEN SWAP
                BGEZ NOSWAP
*
```

```
                    LAR  AR2,J
                    MAR  *0+,AR3    * J = J + IADDR
                    LAR  AR3,I
                    MAR  *0+,AR2    * I = I + IADDR
* SWAP I AND L VALUES.
                    ZALH *,AR3
                    ADDS *
                    SACH *+,0,AR2   * X(I) = X(J)
                    SACL *+         * X(J) = X(I)
*
                    ZALH *,AR3
                    ADDS *
                    SACH *,0,AR2    * Y(I) = Y(J)
                    SACL *,0,AR3    * Y(J) = Y(I)
NOSWAP      LAC  HOLDN
            SACL K                  * K = N
INLOOP              LAC  J
                    SUB  K          * IF J >= K THEN
                    BLZ  OUTLOP,*,AR4
                         SACL J     * J = J - K
                         LAC  K,15
                         SACH K     * K = K/2.
                         B    INLOOP
OUTLOP      ADD  K,1
            SACL J                  * L = L + J
            LAC  I
            ADD  ONE,1
            SACL I                  * INCREMENT I
            BANZ DRLOOP,*-,AR2
*
*   OUTPUT FFT VALUES
*
       LRLK   AR2,INPUT
       RPTK   255
       OUT    *+,PA1
       RPTK   255
       OUT    *+,PA1
*
*  FFT COMPLETE.
*
WHOA   B WHOA
*
* COEFFICIENT TABLE (SIZE OF TABLE IS 3N/4).
*
SINE EQU $
     DATA >0
     DATA >324
     DATA >648
     DATA >96B
     DATA >C8C
     DATA >FAB
     DATA >12C8
     DATA >15E2
```

```
DATA >18F9
DATA >1C0C
DATA >1F1A
DATA >2224
DATA >2528
DATA >2827
DATA >2B1F
DATA >2E11
DATA >30FC
DATA >33DF
DATA >36BA
DATA >398D
DATA >3C57
DATA >3F17
DATA >41CE
DATA >447B
DATA >471D
DATA >49B4
DATA >4C40
DATA >4EC0
DATA >5134
DATA >539B
DATA >55F6
DATA >5843
DATA >5A82
DATA >5CB4
DATA >5ED7
DATA >60EC
DATA >62F2
DATA >64E9
DATA >66D0
DATA >68A7
DATA >6A6E
DATA >6C24
DATA >6DCA
DATA >6F5F
DATA >70E3
DATA >7255
DATA >73B6
DATA >7505
DATA >7642
DATA >776C
DATA >7885
DATA >798A
DATA >7A7D
DATA >7B5D
DATA >7C2A
DATA >7CE4
DATA >7D8A
DATA >7E1E
DATA >7E9D
DATA >7F0A
DATA >7F62
```

```
        DATA >7FA7
        DATA >7ED9
        DATA >7FF6
COSINE  EQU $
        DATA >7FFF
        DATA >7FF6
        DATA >7ED9
        DATA >7FA7
        DATA >7F62
        DATA >7F0A
        DATA >7E9D
        DATA >7E1E
        DATA >7D8A
        DATA >7CE4
        DATA >7C2A
        DATA >7B5D
        DATA >7A7D
        DATA >798A
        DATA >7885
        DATA >776C
        DATA >7642
        DATA >7505
        DATA >73B6
        DATA >7255
        DATA >70E3
        DATA >6F5F
        DATA >6DCA
        DATA >6C24
        DATA >6A6E
        DATA >68A7
        DATA >66D0
        DATA >64E9
        DATA >62F2
        DATA >60EC
        DATA >5ED7
        DATA >5CB4
        DATA >5A82
        DATA >5843
        DATA >55F6
        DATA >539B
        DATA >5134
        DATA >4EC0
        DATA >4C40
        DATA >49B4
        DATA >471D
        DATA >447B
        DATA >41CE
        DATA >3F17
        DATA >3C57
        DATA >398D
        DATA >36BA
        DATA >33DF
        DATA >30FC
```

```
DATA >2E11
DATA >2B1F
DATA >2827
DATA >2528
DATA >2224
DATA >1F1A
DATA >1C0C
DATA >18F9
DATA >15E2
DATA >12C8
DATA >FAB
DATA >C8C
DATA >96B
DATA >648
DATA >324
DATA >0
DATA >FCDC
DATA >F9B8
DATA >F695
DATA >F374
DATA >F055
DATA >ED38
DATA >EA1E
DATA >E707
DATA >E3F4
DATA >E0E6
DATA >DDDC
DATA >DAD8
DATA >D7D9
DATA >D4E1
DATA >D1EF
DATA >CF04
DATA >CC21
DATA >C946
DATA >C673
DATA >C3A9
DATA >C0E9
DATA >BE32
DATA >BB85
DATA >B8E3
DATA >B64C
DATA >B3C0
DATA >B140
DATA >AECC
DATA >AC65
DATA >AA0A
DATA >A7BD
DATA >A57E
DATA >A34C
DATA >A129
DATA >9F14
DATA >9D0E
DATA >9B17
```

```
        DATA >9930
        DATA >9759
        DATA >9592
        DATA >93DC
        DATA >9236
        DATA >90A1
        DATA >8F1D
        DATA >8DAB
        DATA >8C4A
        DATA >8AFB
        DATA >89BE
        DATA >8894
        DATA >877B
        DATA >8676
        DATA >8583
        DATA >84A3
        DATA >83D6
        DATA >831C
        DATA >8276
        DATA >81E2
        DATA >8163
        DATA >80F6
        DATA >809E
        DATA >8059
        DATA >8027
        DATA >800A
        END
$
```

บทที่ 5

บทวิจารณ์และสรุป

5.1 บทสรุป

การประมวลผลสัญญาณเชิงเลขโดยใช้ตัวประมวลผลสัญญาณเชิงเลข (Digital Signal Processor) สามารถกระทำได้รวดเร็วกว่าการใช้ไมโครคอมพิวเตอร์ปกติ เพราะตัวประมวลผลสัญญาณเชิงเลขถูกออกแบบสร้างมาเพื่อใช้ในการคำนวณทางด้านการประมวลผลสัญญาณเชิงเลขโดยเฉพาะ แต่หน่วยประมวลผลกลาง(CPU) ของเครื่องไมโครคอมพิวเตอร์ถูกออกแบบสร้างมาเพื่อใช้ในงานทั่วๆไป และโดยทั่วๆไปจะมีคำสั่งในการทำงานต่างๆ มากกว่าเพราะต้องรองรับงานหลายประเภท ทำให้การสร้างไม่อาจค่านึงแต่เฉพาะความเร็วในการประมวลผลด้านใดด้านหนึ่งโดยเฉพาะได้ แต่ในตัวประมวลผลสัญญาณเชิงเลขถูกออกแบบมาเพื่อรองรับงานทางด้านการประมวลผลสัญญาณเชิงเลขโดยเฉพาะทำให้การสร้างสามารถเน้นไปทางด้านนี้ได้ โดยจะมีคำสั่งที่เอื้ออำนวยต่อการเขียนโปรแกรมทางด้านการประมวลผลสัญญาณเชิงเลข และมีช่วงเวลาในการทำสั่งได้รวดเร็วกว่า

ดังนั้นจึงอาจใช้ตัวประมวลผลสัญญาณเชิงเลขเป็นโปรเซสเซอร์ร่วม (co-processor)ในการประมวลผลร่วมกับ CPU ของเครื่องไมโครคอมพิวเตอร์ได้ โดยทำหน้าที่ช่วยในการคำนวณทางด้านการประมวลผลสัญญาณเชิงเลข เช่น การประมวลผลสัญญาณภาพ(Digital Image Processing), ฟาสท์ฟูเรียร์ทรานสฟอร์ม (Fast Fourier Transform), ดีสครีตโคซายน์ทรานสฟอร์ม(Discrete Cosine Transform) เป็นต้น

5.2 แนวทางในการพัฒนาต่อ

ในการรับข้อมูลเพื่อนำมาประมวลผลจะต้องทำการรับอินพุทและเอาท์พุทจากเครื่องไมโครคอมพิวเตอร์เท่านั้น ดังนั้นจะเป็นการสะดวกกว่าถ้าสามารถมีส่วนที่เป็น A/D (Analog to Digital) และ D/A (DigitaltoAnalog) เพิ่มขึ้นเพื่อให้ TMS320C25 สามารถรับอินพุทและเอาท์พุทค่าต่างๆ จากการทดลองได้โดยตรง เพื่อให้สามารถประมวลผลสัญญาณอินพุทที่เป็นอนาลอก(analog) และให้เอาท์พุทอนาลอกแบบเวลาจริง (real time) เช่น ใช้เป็นดิจิตอลฟิลเตอร์ (Digital Filter), การวิเคราะห์และสังเคราะห์เสียงพูด เป็นต้น

กิตติกรรมประกาศ

ขอกราบขอบพระคุณอาจารย์ที่ปรึกษา ดร.กอบชัย เดชหาญ และ อ. สมยศ จุณณะปิยะ เป็นอย่างมากที่ได้ช่วยให้ความรู้ และคำแนะนำในการทำปริญญานิพนธ์ฉบับนี้ ขอขอบคุณทางสำนักวิจัยและบริการคอมพิวเตอร์ที่ได้ให้ยืมเครื่องมือและอุปกรณ์ในการศึกษาทดลองต่างๆ และภาควิชาโทรคมนาคมที่ได้เอื้อเฟื้ออุปกรณ์ในการทำปริญญานิพนธ์ตลอดจนเพื่อนๆ ที่ได้ให้ความช่วยเหลือเป็นอย่างดี ทางผู้จัดทำจึงขอขอบพระคุณมา ณ โอกาสนี้เป็นอย่างยิ่ง

หนังสืออ้างอิง

1. ธานินทร์ ถาวรศาสนวงศ์ และ ทินกร ดุ๊ก,"การอินเทอร์เฟส IBM PC", ฟิสิกเซ็นเตอร์การพิมพ์.
2. ยืน ภู่วรวรรณ,"เทคโนโลยี ฮาร์ดแวร์ IBMPC", บริษัท ซีเอ็ดยูเคชั่น จำกัด, 2533.
3. A.W.M. Vanden Enden and N.A.M. Verhoechx, "Discrete-Time Signal Processing and Introduction", Prentice Hall, 1989.
4. Alan V. Oppenhein ,and Ranald W. Schafer,"Discrete-Time Signal Processing", Prentice Hall, 1989.
5. "IBM PC/XT Technical Refference", Interational Bussiness Machine, 1983.
6. "IBM PC/AT Technical Refference", Interational Bussiness Machine, 1984.
7. Kun-Shan (Editor), "Digital Signal Processing Application with The TMS320C25 Family", Prentice Hall, 1987.
8. Kun-Shan Lin, Gene A. Frantz and Ray Simar, "The TMS320C25 Family of Digital Signal Processors", Proceeding of IEEE, Vol. 75,No. 9, September 1987
9. "TMS320C10 User' Guide", Texas Instruments, 1985
10. "TMS320C25 User's Guide Digital Signal Processor Products", Texas Instruments, 1986