เครื่องแปลงสัญญาณมิดี้

MIDI CONVERSION BOX

โดย

นายสันต์ พู่พิทยาสถาพร

นายอภิวัฒน์ วงษ์กำแหง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

เครื่องแปลงสัญญาณมิดี้

MIDI CONVERSION BOX

โดย

นายสันต์ พู่พิทยาสถาพร รหัส 45010814

นายอภิวัฒน์ วงษ์กำแหง รหัส 45010925

อาจารย์ที่ปรึกษา

รศ. จิรวัฒน์ ปานกลาง

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ ปีการศึกษา 2548

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องแปลงสัญญาณมิดี้

ผู้จัดทำ

1. นายสันต์ พู่พิทยาสถาพร รหัส 45010814
2. นายอภิวัฒน์ วงษ์กำแหง รหัส 45010925

อาจารย์ที่ปรึกษา

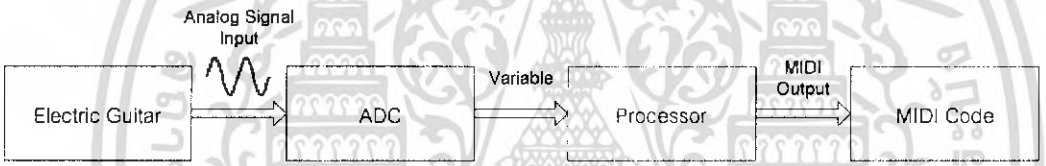
(รศ.จิรวัฒน์ ปานกลาง)

เครื่องส่งคำสั่งมิดี้จากสัญญาณอนาลอก

นายสันต์ พู่พิทยาสถาพร 45010814
นายอภิวัฒน์ วงษ์กำแหง 45010925
รศ.จิรวัฒน์ ปานกลาง (อาจารย์ที่ปรึกษา)
ปีการศึกษา 2548

บทคัดย่อ

โครงงานนี้เป็นอุปกรณ์ใช้รับสัญญาณอนาลอกจากกีต้าร์ไฟฟ้า นำมาประมวลผล และส่งผลลัพธ์ออกมาเป็นชุดคำสั่งดิจิตอล ในรูปแบบมิดี้ ซึ่งชุดคำสั่งมิดี้เป็นการบอกค่าตัวแปรต่างๆของโน้ตดนตรีสากล เช่น ค่าตำแหน่งตัวโน้ต ค่าความดังของโน้ต มีการทำงานตามบลอกไดอแกรมในรูปที่ 1



รูปที่ 1 บลอกไดอแกรม

- รับสัญญาณอนาลอกจากกีต้าร์ไฟฟ้า ที่มีช่วงความถี่ประมาณ 329.6 Hz ถึง 1318.4 Hz
- ในส่วน ADC จะนำสัญญาณอนาลอกมาแปลงเป็นสัญญาณดิจิตอล
- ในส่วนประมวลผลจะนำสัญญาณดิจิตอลมาวิเคราะห์หา ระดับเสียง ของโน้ตดนตรีสากล แล้วนำมาแปลงเป็นรูปแบบมิดี้ แล้วส่งออกมาทาง DIN 5 Port
- สัญญาณมิดี้ที่ได้สามารถนำไปสั่งงานอุปกรณ์ดนตรีที่สามารถรับคำสั่งมิดี้ได้

Midi Conversion Box

Mr. Sant Poopityastaporn 45010814
Mr. Apiwat Wongkmahaeng 45010925
Assoc. Prof. Jirawat Parnklang (Adviser)
Education Year 2005

Abstract

This project is a device that receives analog signal from electric guitar. Process that signal and give us the digital command output in MIDI format. MIDI commands contain many parameters of musical note such as note number and velocity.

A block diagram shown in Figure1

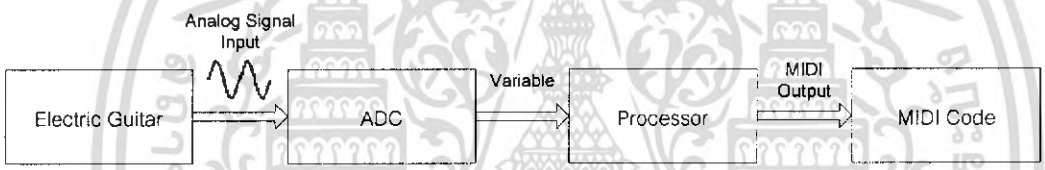


Figure1 Block Diagram

- We receive the analog signal from a electric guitar, which gives us electrical signal, in the range of frequency from 329.6 Hz to 1318.4 Hz.
- In ADC part, we sample the analog signal to digital signal.
- In the processor part , the signal is analyzed, identified pitch of musical note and converted to MIDI code.
- The MIDI output can be used to control other MIDI compatible music instruments.

กิตติกรรมประกาศ

ขอขอบคุณ

รศ.จิรวัฒน์ ปานกลาง และอาจารย์ภาควิชาอิเล็กทรอนิกส์ เป็นอย่างสูงสำหรับคำแนะนำ คำปรึกษา และความรู้ต่างๆ ที่นำมาพัฒนาปริญญานิพนธ์นี้

เพื่อนร่วมห้องโปรเจค และ เพื่อนๆ พี่ๆ น้องๆ ภาควิชาอิเล็กทรอนิกส์ และภาควิชาวัดคุม ที่คอยให้กำลังใจ ให้คำแนะนำดีๆ และให้หยิบยืมกีต้าร์เครื่องมืออุปกรณ์ต่างๆ

อุปกรณ์เครื่องมือที่ห้อง ผศ.พลผดุง ผดุงกุล และชุมนุมอิเล็กทรอนิกส์

และสุดท้ายนี้ขอกราบขอบพระคุณ คุณพ่อคุณแม่ ครอบครัวพู่พิทยาสถาพร และครอบครัววงษ์กำแหง ที่คอยห่วงใยสุขภาพและให้การสนับสนุนทางการศึกษามาโดยตลอด

สันต์ พู่พิทยาสถาพร

(นายสันต์ พู่พิทยาสถาพร)

อภิวัฒน์ วงษ์กำแหง

(นายอภิวัฒน์ วงษ์กำแหง)

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาและแนวคิด

ปัจจุบัน การส่งข้อมูลในรูปแบบมิดี้ไม่ว่าจะเป็นการใช้ส่งคำสั่งให้ซินธีไซเซอร์ซอฟท์แวร์หรือซินธีไซเซอร์ฮาร์ดแวร์สร้างเสียงต่างๆหรือการบันทึกข้อมูลทางดนตรีในรูปแบบมิดี้ นั้นส่วนใหญ่ถูกออกแบบมาให้ใช้กับอุปกรณ์ดนตรีที่มีลักษณะเป็นคีย์บอร์ด ดังนั้นโลกของมิดี้ทางดนตรีจึงจำกัดวงแคบอยู่ที่ผู้ใช้คีย์บอร์ดเท่านั้น จะเป็นไปได้หรือไม่ที่นักดนตรีที่เล่นอุปกรณ์ดนตรีอื่นๆ เช่น กีตาร์จะมีส่วนเข้าไปสู่โลกของมิดี้บ้าง จึงเป็นที่มาของโครงงานนี้ เครื่องส่งคำสั่งมิดี้จากสัญญาณอนาลอก จะทำอย่างไรให้อุปกรณ์ดนตรีที่สร้างสัญญาณอนาลอก เช่น กีต้าร์ไฟฟ้าสามารถส่งข้อมูลในรูปแบบมิดี้ไปยังซินธีไซเซอร์ซอฟท์แวร์หรือซินธีไซเซอร์ฮาร์ดแวร์เพื่อสร้างเสียงต่างๆ หรือการบันทึกข้อมูลทางดนตรีบนซอฟท์แวร์ทางดนตรีได้

1.2 จุดประสงค์

เพื่อเชื่อมต่อกลุ่มเครื่องดนตรีกำเนิดสัญญาณอนาลอก (ซึ่งอาจหมายรวมถึงเครื่องดนตรีอคุสติกในอนาคต) ไปสู่กลุ่มเครื่องดนตรีทางดิจิตอล เช่นเพื่อให้สัญญาณจากกีต้าร์ไฟฟ้าสามารถควบคุมซินธีไซเซอร์ โดยมีความสามารถคล้ายคีบอร์ด

บทที่ 2
ทฤษฎีและหลักการ

2.1 ระบบมิดี้ (MIDI System)

2.1.1 มิดี้คืออะไร? (What is MIDI?)

มิดี้ (Musical Instrument Digital Interface) คือการเชื่อมต่อทางดิจิตอลของอุปกรณ์ทางดนตรี หรือเป็นการควบคุมระยะไกลด้วยดิจิตอลสำหรับระบบทางดนตรี ซึ่งอุปกรณ์ที่ควบคุมด้วยมิดี้โดยทั่วไปจะใช้ไมโครคอนโทรลเลอร์ในการควบคุม ผ่านทางอินพุท/เอาท์พุทพอร์ต มิดี้เป็นที่นิยมและแพร่หลายมากในด้านการควบคุม นอกจากนี้มิดี้ยังสามารถประยุกต์นำไปใช้ในงานประเภทอื่นได้ เช่นงานด้าน ระบบเสียง ระบบแสง แผงควบคุม ควบคุมอุปกรณ์แสง ฯลฯ แม้ว่าคำสั่งมาตรฐานของมิดี้จะเป็นคำสั่งทางด้านดนตรี แต่ก็สามารถนำไปปรับเปลี่ยนใช้กับงานประเภทอื่นได้ไม่ยาก

การที่มิดี้ใช้มาตรฐานการส่งแบบอนุกรมเนื่องจากเหตุผลทางด้านความประหยัด และความเป็นไปได้ในทางปฏิบัติ โดยมีจุดมุ่งหมายเพื่อให้อุปกรณ์ราคาถูก และสามารถรองรับผู้ใช้ได้หลากหลายเท่าที่จะทำได้ ในการส่งแบบขนานอาจจะมีประสิทธิภาพมากกว่าแต่เมื่อรวมถึงค่าผลรวมของการผลิตต่อชิ้น จะมีราคาที่แพงกว่าและมีขนาดใหญ่เทอะทะมากกว่าการที่ใช้การเชื่อมต่อแบบอนุกรม ความง่ายและสะดวกต่อการติดตั้งอุปกรณ์มิดี้ส่งผลให้มีการแพร่หลายอย่างรวดเร็วในฐานะของมาตรฐานนานาชาติ

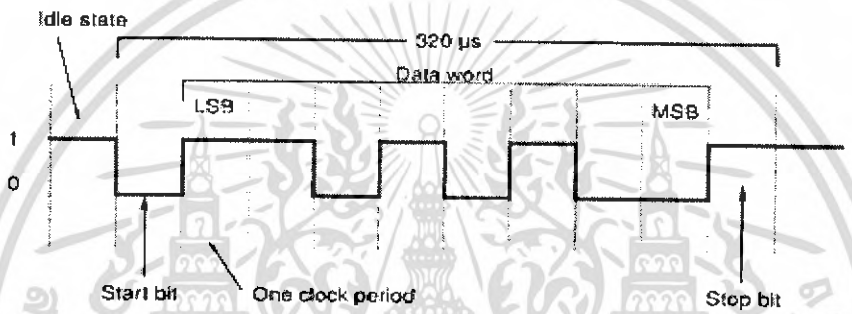
แตกต่างกับอุปกรณ์ในยุคก่อนซึ่งใช้สัญญาณอนาลอก ในการส่งคำสั่งทุกชนิดก็จะอยู่ในรูปแบบเดียวกันบนสายเส้นเดียวกัน มิดี้สามารถควบคุมอุปกรณ์มิดี้ที่เป็นหลายๆเสียงพร้อมกันได้ในเวลาจริงเทียม โดยความเร็วในการส่งจะต้องมีการเหลื่อมของเวลาจะน้อยมากโดยหูของมนุษย์ไม่สามารถแยกแยะความแตกต่างได้ และยังสามารถกำหนดจุดหมายที่ต้องการจะให้แสดงผลโดยการเลือกช่องสัญญาณ

2.1.2 รายละเอียดของระบบ (System specifications)

มารตฐานมิดี้กำหนดว่าเป็นการส่งสัญญาณผ่านการเชื่อมต่อแบบอนุกรมทางเดียว(uni-directional serial interface) โดยทำงานที่ความเร็ว 31.25กิโลบิตต่อวินาที โดยผิดพลาด ±1% ด้วยความเร็วขนาดนี้ถูกกำหนดเมื่อเทียบกับความเร็วของสัญญาณนาฬิกาของไมโครโปรเซสเซอร์ซึ่งโดยทั่วไปจะมีความเร็วของสัญญาณนาฬิกาอย่างน้อย 1 เมกะเฮิรทซ์ หรือ 2 เมกะเฮิรทซ์ จะมี

ความเร็วมากกว่าหลายเท่า โดยความเร็วของการส่งมิดี้ช้าเพียงพอที่จะไม่เกิดการล้นหรือเกินของข้อมูลในสายสัญญาณและอุปกรณ์ทางการเชื่อมต่อ แต่ก็ยังเร็วพอที่จะสามารถส่งข้อมูลทางดนตรีจากเครื่องหนึ่งไปยังเครื่องอื่นๆโดยไม่สามารถสังเกตความแตกต่างของเวลาที่หน่วงได้ ข้อมูลจะถูกส่งไปได้ทางเดียว จากผู้ส่งไปยังผู้รับโดยไม่มีการย้อนกลับ เว้นแต่จะมีการเชื่อต่อกลับจากภายนอก

ข้อความควบคุม (Control messages) จะถูกส่งเป็นกลุ่มของไบท์ โดยเริ่มจากบิทเริ่มต้น 1 บิทและปิดท้ายข้อมูลด้วยบิทสิ้นสุด 1 บิท เพื่อที่เป็นการเทียบเวลาให้ตรงกันของผู้รับสำหรับข้อมูลที่ส่งในรูปแบบอะซิงโครนัส ดังรูปที่ 2.1

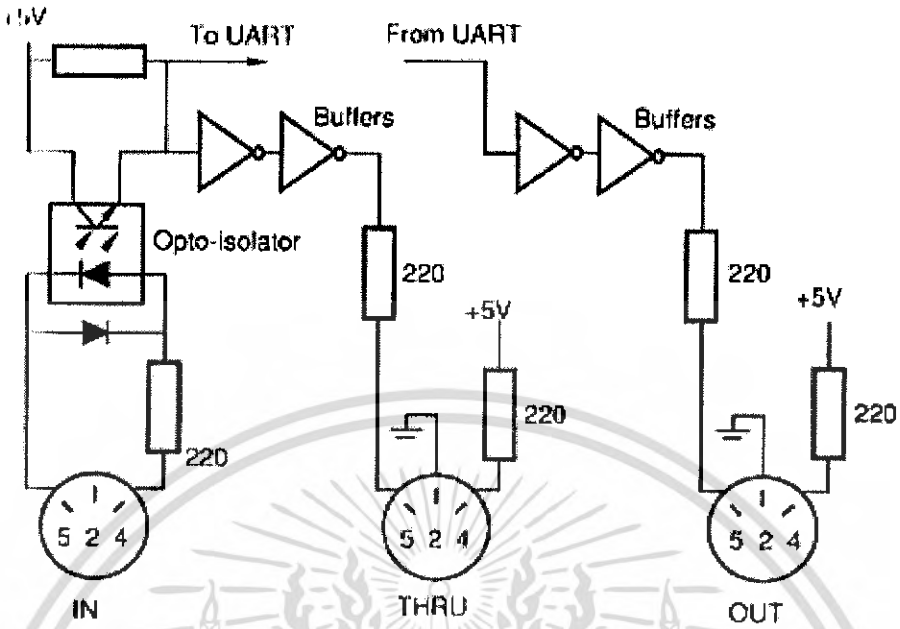


รูปที่ 2.1 การส่งข้อความมิดี้อย่างอนุกรมอะซิงโครนัส ในรูปแบบของ UART โดยจะมีบิตเริ่มต้นและบิตสิ้นสุดเพื่อเป็นการซิงโครไนซ์ให้กับผู้รับ UART โดยที่บอร์ดเรท 31250 จะใช้ระยะเวลาในการส่ง 320us ต่อ 1 ไบท์

มีการเพิ่มบิทเริ่มต้นและบิทสิ้นสุดรวมกับบิทข้อมูล 8 บิท จะกลายเป็นบิทข้อมูล 10 บิทที่จะต้องส่ง โดยรวมเวลาส่งจะได้ประมาณ 320us มาตรฐานของข้อความมิดี้ โดยทั่วไปจะประกอบไปด้วย 1 ไบท์ 2 ไบท์ หรือ 3 ไบท์

2.1.3 การเชื่อมต่อกับอุปกรณ์ (The Hardware Interface)

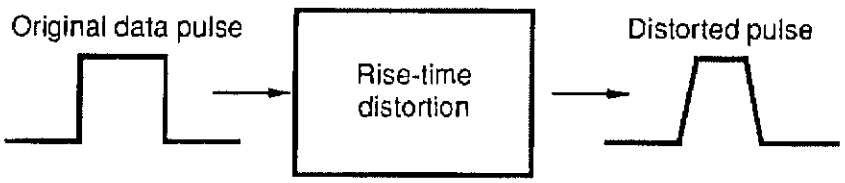
เป็นการเชื่อมต่อกับอุปกรณ์ที่ชัดเจน โดยควรจะใช้งานได้รวมกับอุปกรณ์ทางมิดี้ทุกชนิด และจะต้องมีระบบทางไฟฟ้าที่เข้ากันได้ในระบบ ดังในรูปที่ 2.2



รูปที่ 2.2 การเชื่อมต่อทางไฟฟ้าของมิดี้ที่พอร์ท IN, THRU และ OUT

อุปกรณ์มิดี้ส่วนใหญ่จะมี 3 ช่องคือ IN, OUT และ THRU ที่ช่อง OUT จะเป็นตัวส่งข้อมูลที่ตัวอุปกรณ์นั้นเป็นตัวสร้างขึ้นมาเอง ผ่าน UART ของอุปกรณ์นั้น ที่ช่อง IN ใช้รับข้อมูลที่ส่งมาจากอุปกรณ์อื่นไปยัง UART ส่วนที่ช่อง THRU จะส่งต่อข้อมูลจากช่อง IN ไปยังอุปกรณ์อื่นๆ ได้ อุปกรณ์มิดี้ถูกๆจะไม่มีช่อง THRU แต่สามารถต่ออุปกรณ์ MIDI THRU boxes เพื่อเพิ่มช่อง THRU ได้

ทางช่อง MIDI IN ต้องต่อผ่านทาง Opto-isolator เพื่อไปสู่ส่วนระบบไมโครคอนโทรลเลอร์ของอุปกรณ์ ทั้งนี้เพื่อให้แน่ใจว่าจะไม่มีการเชื่อมต่อทางไฟฟ้าระหว่างอุปกรณ์ และช่วยลดผลของการปัญหาถ้าเกิดอุปกรณ์ตัวหนึ่งมีข้อผิดพลาดทางไฟฟ้า ตัว Opto-isolator เป็นอุปกรณ์ที่ประกอบด้วย LED เพื่อใช้ควบคุมการปิดเปิดของสวิทช์โดยขึ้นอยู่กับศักดาที่ตกคร่อมขั้วทั้ง 2 เพื่อให้แสงกับ Photo-transistor ที่จะขึ้นอยู่กับสถานะของ LED ด้วยเหตุนี้จะส่งข้อมูลผ่านทางแสงด้วย ซึ่งดีกว่าทางไฟฟ้า ในรายละเอียดของมิดี้ ตัว Opto-isolator ใช้เวลาในช่วงขาขึ้นไม่เกิน 2us ซึ่งเวลาในช่วงขาขึ้นจะมีผลกับการตอบสนองของอุปกรณ์เมื่อเปลี่ยนค่า ซึ่งถ้าเวลาในช่วงขาขึ้นใช้เวลานานอาจจะทำให้เกิดรูปผิดเพี้ยนที่ขอบของบิทข้อมูลได้ ดังในรูปที่ 2.3 ซึ่งจะเป็นเหมือนกันสำหรับเวลาในช่วงขาลง



รูปที่ 2.3 ขอบของคลื่นพัลส์สี่เหลี่ยมถูกทำให้เกิด Rise-time distortion

การบิดเบือนของรูปคลื่นในช่วงขาขึ้นนั้นส่งผลให้เกิดความไม่เสถียรทางเวลาของข้อมูล เมื่อกำลังสลับค่าที่ขอบจะไม่สามารถแยกแยะได้ว่าเป็น 0 หรือ 1 และถ้าเวลาในขอบขาขึ้นช้ามากเกินไปค่าของข้อมูลจะผิดพลาดโดยผลลัพธ์ของอุปกรณ์จะยังไม่ขึ้นเต็มค่าก่อนที่ข้อมูลบิตถัดมาจะตามมา ถ้าอุปกรณ์มิดี้หลายๆเครื่องต่อพ่วงกันแบบอนุกรม(เข้าทาง IN และผ่านทางช่อง THRU) ข้อมูลก็จะผ่าน Opto-isolator หลายตัวและจำทำให้เกิดการบิดเบือนไปมาก หรือไม่ก็จะทำให้เกิดการรับข้อมูลที่ผิดพลาดของตัวรับตัวสุดท้าย ดังนั้นจะขึ้นอยู่กับคุณภาพของ Opto-isolator และความสูญเสียของข้อมูลระหว่างทาง

เมื่อคำนึงถึงความเป็นไปได้ของความหน่วงเวลาระหว่างช่อง IN และช่อง THRU จะต้องอยู่ในหน่วยของ us จะไม่มีผลต่อการได้ยินของมนุษย์

2.1.4 อุปกรณ์เชื่อมต่อและสายสัญญาณ (Connectors and Cables)

หัวต่อสำหรับมิดี้เหมือนกับ DIN plugs 5 ขาที่ใช้สำหรับระบบไฮไฟบางระบบ และแม้ว่าสามารถใช้สายของไฮไฟได้ แต่สายที่มีคุณภาพดีกว่าก็ยังจำเป็น ในมาตรฐานก็อนุญาตให้ใช้กับหัว XLR-type ได้แม้ว่าจะใช้ในทางปฏิบัติได้ยาก แต่อุปกรณ์ส่วยใหญ่ก็ใช้เพียงแค่ 3 พินจากทั้งหมด 5 พินของ DIN plug

สายสัญญาณควรจะเป็นสาย Shielded twisted pair และชิลด์จะต่อกับพินที่ 2 ของหัวทั้ง 2 ด้าน แม้ว่าช่อง IN จะไม่ได้ต่อลงกราวด์ที่พินที่ 2 เพื่อที่จะป้องกัน Ground loop และสามารถใช้สายสลับได้ทั้ง 2 ด้าน แนะนำว่าสายไม่ควรยาวเกิน 15เมตร ถ้ายาวเกินจะเกิดข้อผิดพลาดขึ้นได้

2.1.5 รูปแบบข้อความมิดี้ (MIDI Message format)

ข้อความมิดี้จะมีรูปแบบไบท์ข้อความพื้นฐานอยู่ 2 รูปแบบ คือไบท์สถานะ (Status byte) และไบท์ข้อมูล โดยไบท์สถานะนั้นจะขึ้นต้นบิตแรกเป็นไบนารี่ 1 ซึ่งจะตรงข้ามกับไบท์ข้อมูล ที่จะขึ้นต้นบิตแรกเป็น ไบนารี่ 0 ดังนั้นจึงเหลือ 7 บิตที่สามารถใช้งานบอกค่าต่างๆได้ 128 (2^7) ค่า

ข้อความ มิดี้ แต่ละข้อความจะมีไบท์ข้อความอยู่ 1-3 ไบท์ จากรูปที่ 2.4 ในไบท์แรกของข้อความมิดี้จะเป็น ไบท์สถานะ ซึ่งบรรจุไปด้วยข้อมูลเกี่ยวกับมิดี้แชนเนล (MIDI Channel) ซึ่งจะถูกระบุโดย 4 บิตแรก (บิต3 – บิต 0)ซึ่งจะมีทั้งหมด16 แชนเนล ส่วนใน บิต 6- บิต 4 จะทำหน้าที่บอกรูปแบบไบท์ของข้อความมิดี้ชุดนั้นๆ

รูปที่ 2.4 รูปแบบทั่วไปของข้อความมิดี้ โดย บิต 's s s' ใช้บอกรูปแบบของข้อความ บิต 'n n n n' ใช้บอกเลขมิดี้แชนเนล (MIDI Channel number) ส่วน บิต 'x x x x x x x' และ 'y y y y y y y' ใช้บอกถึงข้อมูลของข้อความนี้ดังจะได้กล่าวต่อไป

ข้อความมิดี้มาตรฐานสามารถมีได้ถึง 3 ไบท์ ต่อหนึ่งข้อความแต่ไม่ใช่ว่าทุกข้อความจะต้องยาว 3 ไบท์
รูปแบบข้อความมิดี้ดูได้จากตารางที่ 2.1

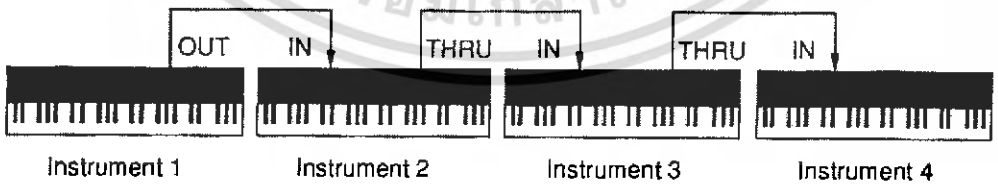
ตารางที่ 2.1 รูปแบบข้อความมิดี้

Message	Status	Data 1	Data 2
Note off	&8n	Note number	Velocity
Note on	&9n	Note number	Velocity
Polyphonic aftertouch	&An	Note number	Pressure
Control change	&Bn	Controller number	Data
Program change	&Cn	Program number	--
Channel aftertouch	&Dn	Pressure	--
Pith wheel	&En	LSbyte	MSbyte
System exclusive			
System exclusive start	&F0	Manufacturer ID	Data (Data)(Data)
End of SysEx	&F7	--	

System common			
Quarter frame	&F1	Data	--
Song pointer	&F2	LSbyte	MSbyte
Song select	&F3	Song number	--
Tune request	&F6	--	
System realtime			
Timing clock	&F8	--	--
Start	&FA	--	--
Continue	&FB	--	--
Stop	&FC	--	--
Active sensing	&FE	--	--
reset	&FF	--	--

2.1.6 มิดี้แชนเนล (MIDI Channel)

มิดี้แชนเนล คือ การประกาศรหัสในการรับส่งข้อมูลมิดี้ เพื่อให้เกิดการตอบสนอง และการละเว้นการตอบสนอง ข้อมูลของอุปกรณ์การรับ จากอุปกรณ์การส่ง เพื่อให้เกิดการตอบสนองข้อมูลของอุปกรณ์รับเฉพาะชิ้นที่ระบุไว้ เช่น จากรูปที่ 2.5 หากอุปกรณ์ ชิ้นที่ 1 เป็นตัวส่ง อุปกรณ์ชิ้นที่ 2,3,4 เป็นตัวรับข้อมูลแล้ว เราสามารถระบุได้ว่าต้องการให้อุปกรณ์ตัวใดรับแล้วตอบสนองข้อมูลบ้าง โดยการกำหนด มิดี้แชนเนล ของอุปกรณ์การรับให้ตรงกับมิดี้แชนเนล ของอุปกรณ์การส่ง ส่วน อุปกรณ์การรับที่กำหนดมิดี้แชนเนลไม่ตรงกันก็จะไม่ตอบสนองต่อข้อมูลนั้นๆ โดยมิดี้แชนเนลทั้งหมดจะมี16 แชนเนล คือ แชนเนล 1 ถึง แชนเนล 16 โดยข้อมูลของมิดี้แชนเนลจะถูกระบุใน บิต3 – บิต 0 ของไบท์สถานะ.ในข้อความมิดี้



รูปที่ 2.5 การส่งข้อมูลมิดี้ แบบลูกโซ่โดยอุปกรณ์ 1 เป็นตัวส่งและอุปกรณ์ 2, 3 และ 4 เป็นตัวรับ

2.1.7 ข้อมูลเกี่ยวกับโน้ตในข้อความมิดี้ (Note information in MIDI message)

ในข้อความมิดี้ มีการบอกค่าต่างๆของโน้ต โดยจะมีการระบุค่าไว้ที่ไบท์ข้อมูลที่ 1 และ 2 (ไบท์ที่ 3 ของข้อความมิดี้)ยกเว้นค่าความยาวเสียงจะไม่มีการระบุโดยตรงแต่จะใช้เป็นการบอกเปิดโน้ต (Note on) และการบอกปิดโน้ต (Note off) แทนโดยจะบอกค่าใน บิต 6- บิต 4 ของไบท์สถานะ โดยการบอกค่าต่างๆของโน้ต มีรายละเอียดดังนี้

1. เปิดโน้ต และ ปิดโน้ต (Note on and note off) ใช้ในการบอกค่าความยาวของตัวโน้ตคือโน้ตจะเริ่มเล่นเมื่อได้รับข้อความเปิดโน้ต และ จะเล่นไปเรื่อยๆจนกว่าจะได้รับข้อความปิดโน้ตจึงทำการหยุดเล่น โดยข้อมูลการเปิดโน้ต และ ปิดโน้ต จะถูกระบุใน บิต 3 – บิต 0 ของไบท์สถานะ. ในข้อความมิดี้ โดยมีค่าดังในตารางที่ 2.2

2. ระดับเสียงของตัวโน้ต (Pitch) จะมีการระบุค่าไว้ทั้งหมด 128 (2^7) ค่า ตั้งแต่ ระดับเสียง C -2 ถึง G8 โดยจะทำการระบุ ค่าไว้ในบิต 6 – บิต 0 ของ ไบท์ข้อมูลที่ 1 โดยมีค่าดังตารางที่ 2.1

ตารางที่ 2.2 การบอกระดับเสียงเป็นตัวเลขโน้ตมิดี้

ระดับเสียง (Pitch)	ตัวเลขโน้ตมิดี้ (MIDI note number)
C -2	0
C -1	12
C 0	24
C 1	36
C 2	48
C 3	60
C 4	72
C 5	84
C 6	96
C 7	108
C 8	120
G 8	127

3. ค่าความดังของโน้ต (Velocity) จะสามารถระบุค่าได้ทั้งหมด 128 (2^7) ความแตกต่างโดยจะทำการระบุค่าไว้ในบิต 6 – บิต 0 ของ ไบท์ข้อมูลที่ 2โดยค่าความดังจะถูกส่งมาพร้อมกับชุดข้อมูลเปิดโน้ตและจะคงค่าความดังนั้นๆไว้จนกว่าจะได้รับข้อมูลปิดโน้ต ส่วนค่าความดังที่มากับชุดข้อมูลปิดโน้ตนั้นจะมีใช้กันบ้างในวงแคบ มีอุกรณ์บางชิ้นที่จะตอบสนองข้อมูลนี้

4. ค่าแรงกดคีย์หลังเปิดโน้ต (Polyphonic aftertouch) เป็นข้อมูลที่มีใช้ในอุปกรณ์ที่มีลักษณะเป็นคีย์บอร์ด โดยจะเป็นข้อมูลที่บอกถึงแรงกดคีย์ที่เปลี่ยนไปหลังจากมีการเปิดโน้ตไปแล้ว โดยจะมีการส่งข้อมูลในบิต 3 – บิต 0 ของไบท์สถานะในข้อความมิดี้ โดยมีค่าดังในตารางที่ 2.1

2.2 การแปลงฟูเรียร์ (Fourier Transforms)

การแปลงฟูเรียร์เป็นคณิตสาสตร์พื้นฐานในการวิเคราะห์สัญญาณในโดเมนความถี่ ในสาขาการ
ประมวลผลสัญญาณจึงนิยมนำการแปลงฟูเรียร์มาใช้งานกันอย่างแพร่หลายทั้งนี้เพราะการแปลงฟูเรียร์จะมีสัญญาณไซน์และโคไซน์เป็นองค์ประกอบ ซึ่งโดยทั่วไปแล้วสัญญาณทั้งสองจะมีความสำคัญมากเพราะเป็นสัญญาณพื้นฐานสำหรับพิจารณาสัญญาณอื่นๆ

ในบทนี้จะได้กล่าวถึงการแปลงฟูเรียร์เต็มหน่วย ซึ่งเป็นการวิเคราะห์สัญญาณในโดเมนความถี่ การแปลงฟูเรียร์แบบเร็ว การแปลงฟูเรียร์ในช่วงเวลาสั้น การหาค่าเฉลี่ยของสัญญาณแบบเคลื่อนที่ และคุณสมบัติของเสียงดนตรี

2.2.1 การแปลงฟูเรียร์เต็มหน่วย (Discrete Fourier Transform: DFT)

การแปลงฟูเรียร์เต็มหน่วยหรือการแปลงฟูเรียร์ไม่ต่อเนื่อง ได้พัฒนาขึ้นมาเพื่อใช้ในการแปลงสัญญาณดิจิตอลซึ่งอยู่ในโดเมนเวลาไปเป็นสัญญาณที่อยู่ในโดเมนความถี่ ดังนั้นจึงกล่าวได้ว่าการแปลงฟูเรียร์เต็มหน่วยเป็นการวิเคราะห์สัญญาณในโดเมนความถี่

2.2.1.1 การคำนวณการแปลงฟูเรียร์เต็มหน่วยโดยตรง (Direct Computation DFT)

พิจารณา x(n) ซึ่งเป็นชุดแถวของสัญญาณเต็มหน่วยอินพุทที่ประกอบด้วย N จุดตัวอย่าง X(k) ก็คือชุดแถวของสัญญาณที่ได้จากการแปลงฟูเรียร์ และมีจำนวน N จุดตัวอย่าง เช่นเดียวกัน

$$\text{DFT}: \quad X(k) = \begin{cases} \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi nk}{N}} & ; \ 0 \le k \le N-1 \\ 0 & ; \ elsewhere \end{cases} \qquad (2.1)$$

และ

$$\text{IDFT}: \quad x(n) = \begin{cases} \sum_{n=0}^{N-1} X(k) e^{\frac{j2\pi nk}{N}} ; & 0 \le n \le N-1 \\ 0 ; & elsewhere \end{cases} \qquad (2.2)$$

สังเกตว่า การคำนวณ x(n) จาก X(k) หรือ Inverse DFT (IDFT) นั้น เป็นวิธีเดียวกันกับการคำนวณ X(k) จาก x(n) สำหรับการคำนวณแต่ละองค์ประกอบความถี่ X_k จะต้องใช้การคูณจำนวนเชิงซ้อน N ครั้ง และใช้การบวกจำนวนเชิงซ้อน (N-1) ครั้ง แต่ละชุดข้อมูลก็จะมี N ตัวอย่าง ดังนั้นออเดอร์ของการคูณและการบวกจึงเขียนได้เป็น

- (Multiplication) = N^2
- (Addition) = $N(N-1) \approx N^2$

เนื่องจากมีการกำหนดมาก่อนแล้วถึงความเป็นวัฏจักร (Cyclic หรือ Periodic) การคำนวณ DFT จึงต้องอาศัยรากเอกภาพอันดับ N (N^{th} root of Unity) ในที่นี้ให้

$$W_N = e^{-j2\pi/N}$$

โดยอาศัยคุณสมบัติสังยุคต์สมมาตร (Complex Conjugate Symmetry) และการสมคาบ (Periodicity) ของ $e^{-j2\pi nk/N} = W_N^{nk}$ สำหรับ n และ k

$$W_N^{k(N-n)} = W_N^{-nk} = \left(W_N^{nk}\right)^*$$

และ

$$W_N^{nk} = W_N^{k(n+N)} = W_N^{n(k+N)}$$

คุณสมบัติดังกล่าวเป็นคุณสมบัติที่มีประโยชน์ในการพัฒนาการแปลงฟูเรียร์แบบเร็ว

การประมวลข้อมูลจำนวนมาก ๆ โดยใช้วิธีคำนวณโดยตรงนั้น มักจะไม่นิยมเพราะสิ้นเปลืองเวลาในการคำนวณมากเกินไป อย่างไรก็ตามการคำนวณโดยตรงมีโครงสร้างที่ไม่ซับซ้อน ทำให้สามารถที่จะสร้าง วงจรคำนวณในแบบหน่วยแถว (Systolic Array) ได้ง่าย หรือสามารถที่จะใช้การประมวลแบบขนาน (Parallel Processing) ได้สะดวก

2.2.2 การแปลงฟูเรียร์แบบเร็ว (Fast Fourier Transform)

ในการคำนวณการแปลงฟูเรียร์เต็มหน่วยโดยตรงนั้นมีการสิ้นเปลืองเวลาในการคำนวณมาก มักไม่ได้รับความนิยม โดยเฉพาะเมื่อ N มีค่ามาก ดังนั้นจึงมีขั้นตอนวิธีจำนวนอื่นอีกมากที่ได้รับการพัฒนาจนใช้งานได้อย่างเกิดประสิทธิภาพเมื่อเทียบกับการคำนวณโดยตรง ขั้นตอนวิธีส่วนใหญ่จะขึ้นอยู่กับจำนวนข้อมูลในชุดแถวหรือขนาดของบล็อค (Block Size, N) เป็นสำคัญ เช่น ขั้นตอนวิธีตัวประกอบปฐม (Prime Factor Algorithm, PFA) เหมาะสมกับ N ที่เป็นจำนวนปฐม (เช่น 2, 3, 5, 7, 11, 13, 17, 19 เป็นต้น) ขั้นตอนวิธีแยกองค์ประกอบมักจะเหมาะสมกับ N ที่แยกตัวประกอบได้ หรือ Composite Block Length (เช่น 4 = 2x2, 6 = 3x2, 12 = 3x2x2, 15 = 5x3, 255 = 3x5x17 เป็นต้น) ขั้นตอนวิธีแยกตัวประกอบที่ลดการคำนวณได้สูงสุดก็คือวิธีของ Cooley & Tukey ที่ถูกคิดค้นขึ้นในปี ค.ศ. 1965 ซึ่งเหมาะสมกับ $N=2^m$ (เช่น N = 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 ... เป็นต้น)

วิธีของ Cooley & Tukey ได้ย่อยการคำนวณเป็นชุดเล็ก ๆ จำนวนมากที่แทบจะไม่ต้องอาศัยการคูณ ความซับซ้อนของการคำนวณจาก N^2 จึงกลายมาเป็น $N\log_2 N$ วิธีดังกล่าวนี้อาศัยการลดทอนทางเวลา (เริ่มการแบ่งจากกลุ่มของข้อมูล, x(n) ก่อน) หรือการลดทอนทางความถี่ (เริ่มการแบ่งจากกลุ่มของผลลัพธ์, X(k) ก่อน) ก็ได้

2.2.2.1 การคำนวณการแปลงฟูเรียร์แบบเร็วโดยการลดทอนทางเวลา (Decimation in Time Fast Fourier Transforms; DIT FFT)

สำหรับชุดข้อมูล $N = 2^m$ เมื่อ m คือเลขจำนวนเต็มบวกใด ๆ x(n) เป็นอินพุทที่ดัชนีต่าง ๆ การแปลง ฟูเรียร์เต็มหน่วย สามารถเขียนได้โดย

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad 0 \le k \le N-1 \qquad (2.3)$$

$$X(k) = \sum_{\substack{n=0 \\ n=even}}^{N-2} x(n) W_N^{nk} + \sum_{\substack{n=1 \\ n=odd}}^{N-1} x(n) W_N^{nk} \qquad (2.4)$$

ให้ $n = 2r$ สำหรับผลบวกชุดแรก และให้ $n = 2r+1$ สำหรับผลบวกชุดหลัง จะได้

$$X(k)=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2rk}+\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_N^{(2r+1)k}$$

จาก $W_N^{(2r+1)k}=W_N^{2rk}\cdot W_N^k$ และ $W_N^{2rk}=\left(W_N^2\right)^{rk}=e^{-\frac{j2\pi}{N}2rk}=e^{-\frac{j2\pi}{N/2}rk}=W_{N/2}^{rk}$ จะได้

$$X(k)=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_{N/2}^{rk}$$

เมื่อให้ $g(r)=x(2r)$ และ $h(r)=x(2r+1)$ จะได้

$$X(k)=\sum_{r=0}^{\frac{N}{2}-1}g(r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}h(r)W_{N/2}^{rk}\quad;\ 0\le k\le N-1 \qquad (2.5)$$

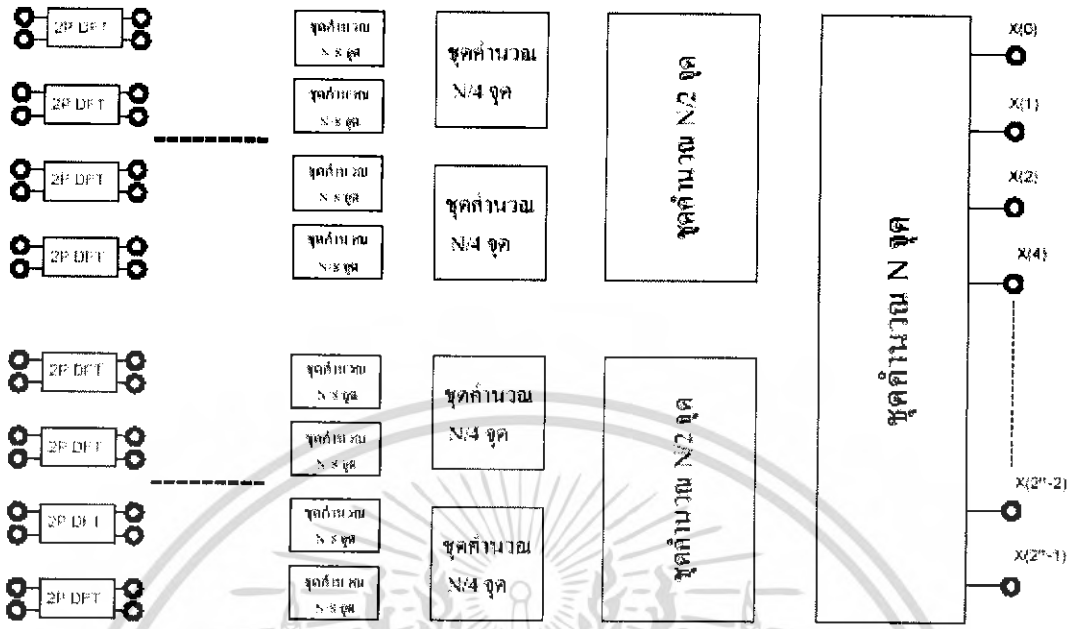
แม้ว่าดัชนี k ในสมการ (2.5) จะมีค่าจาก 0 ถึง N-1 แต่ละส่วนผลบวกก็จะใช้เพียงครึ่งเดียวเท่านั้น และเมื่อให้ G(k) และ H(k) เป็นผลการแปลงฟูเรียร์เต็มหน่วยของ g(n) และ h(n) ตามลำดับ จะได้

$$X(k)=G(k)+W_N^kH(k)\quad;\ 0\le k\le\frac{N}{2} \qquad (2.6)$$

เนื่องจากคาบของ G(k) และ H(k) มีค่าเป็น N/2 ค่าของ X(k) ที่เหลือจึงหาได้จาก

$$X(k)=G(k-\frac{N}{2})+W_N^kH(k-\frac{N}{2})\quad;\ \frac{N}{2}\le k\le N-1 \qquad (2.7)$$

การคำนวณตามสมการที่ (2.7) สามารถที่จะแสดงได้ดังรูปที่ 2.6



รูปที่ 2.6 การคำนวณการแปลงฟูเรียร์แบบเร็วโดยการลดทอนทางเวลา

ในทำนองเดียวกันการคำนวณการแปลง G(k) และ H(k) ก็สามารถที่จะแยกย่อยไปได้อีกเรื่อย ๆ จน มีขนาดน้อยที่สุด คือ 2 จุด แสดงได้โดย

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} + \sum_{n=0}^{1} x(n) W_2^{nk}$$

ซึ่งอาจจะเขียนอยู่ในรูปของเมตริกซ์ ดังนี้

$$\begin{bmatrix} X(0) \\ X(1) \end{bmatrix} = \begin{bmatrix} W_2^0 & W_2^0 \\ W_2^0 & W_2^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \end{bmatrix}$$

เมื่อ $W_2^0 = e^0 = 1$ และ $W_2^1 = e^{-j2\pi/2} = \cos(\pi) - j\sin(\pi) = -1$

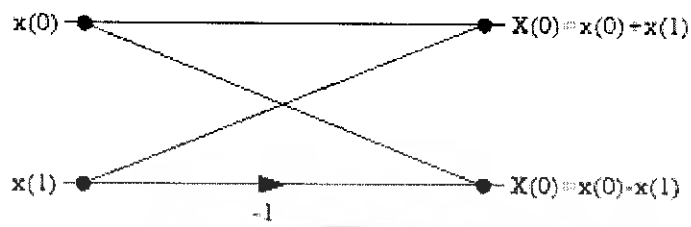
หรือจะได้

$$X(0) = x(0)W_2^0 + x(1)W_2^0 = x(0) + x(1)$$

และ

$$X(1) = x(0)W_2^0 + x(1)W_2^1 = x(0) - x(1)$$

สามารถที่จะเขียนกราฟไหลของสัญญาณ (Signal Flow Graph) ได้ดังรูป

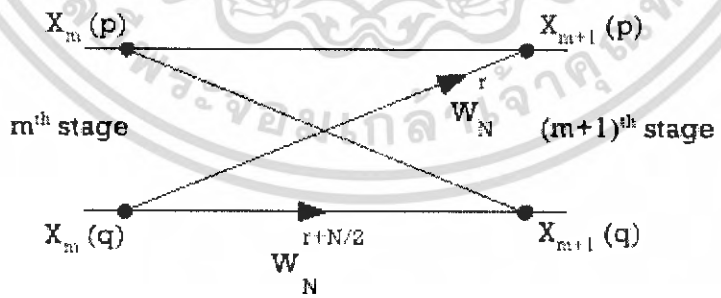


รูปที่ 2.7 กราฟไหลสัญญาณของการแปลง 2 จุด

สังเกตว่าในการลดทอนแต่ละครั้งผ่าน (Pass) ขนาดของบล็อคที่จะต้องทำการแปลงจะลดลงครึ่งหนึ่งเสมอ ดังนั้นจำนวนครั้งผ่านทั้งหมดจึงเป็น $\log_2 N$ ในแต่ละครั้งผ่าน จะมีการคูณ N ครั้ง (คือ คูณด้วย W_N^k) จำนวนการคูณทั้งหมดจึงเป็น Nlog2N ซึ่งเมื่อเทียบกับ N^2 แล้วจะน้อยกว่ามาก

2.2.2.2 แฟกเตอร์ตัวบิดและหน่วยคำนวณผีเสื้อ (Twiddle Factor & Butterfly Unit)

การทำงานขั้นพื้นฐานของการแปลงก็คือการเรียงดัชนีใหม่การคูณด้วย W_N^k และการบวกกันของ ข้อมูลการคูณเสมือนทำให้ตำแหน่งของข้อมูลย้ายไป จึงเรียก W_N^r ว่าเป็นแฟกเตอร์ตัวบิด (Twiddle Factor) การบวกกันเป็นคู่ ๆ มีลักษณะเป็นแบบเดิมเสมอและใช้วงเหมือนตัวผีเสื้อ ดังนี้



รูปที่ 2.8 กราฟไหลสัญญาณในหน่วยคำนวณผีเสื้อ

$$X_{m+1}(p) = X_m(p) + W_N^r X_m(q) \qquad (2.8)$$

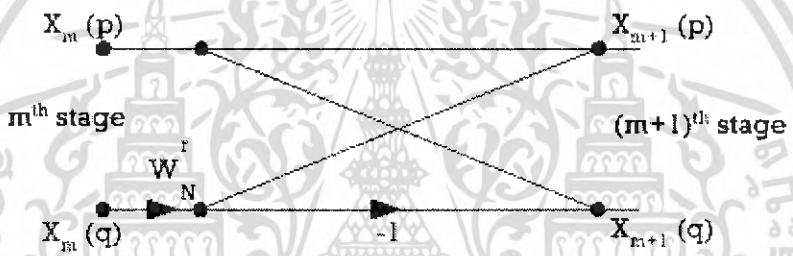
$$X_{m+1}(q) = X_m(p) + W_N^{r+N/2} X_m(q) \qquad (2.9)$$

จากสมการ (2.9)

$$X_{m+1}(q) = X_m(p) + W_N^r W_N^{N/2} X_m(q)$$

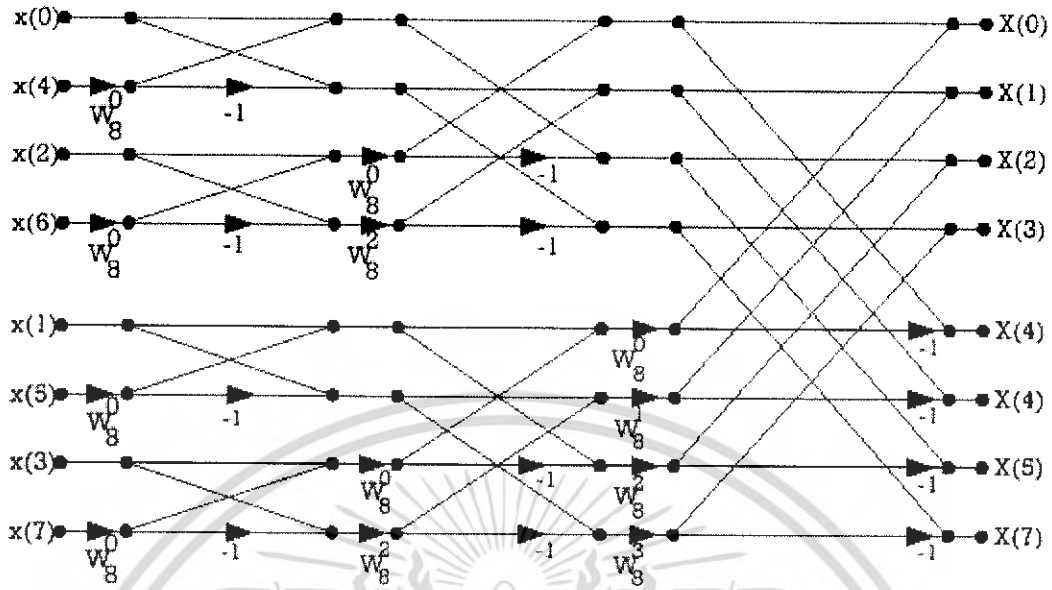
$$= X_m(p) - W_N^r X_m(q) \qquad (2.10)$$

จากสมการ (2.8) และ (2.10) จะเขียนรูปหน่วยผีเสื้อได้ใหม่เป็น



รูปที่ 2.9 หน่วยคำนวณผีเสื้อที่ลดรูปแล้ว

จะสังเกตเห็นว่าหน่วยคำนวณนี้ใช้การคูณ(จำนวนเชิงซ้อน)เพียง 1 ครั้ง จากตัวอย่างการแปลง 8 จุด จะเขียนกราฟไหลของสัญญาณได้เป็นดังรูปที่ 2.10 การคูณด้วย –1 เป็นการทำงานที่ง่ายเพราะเพียงแค่กลับเครื่องหมายเท่านั้น และยังมีการคูณอีกมาก (=2^m-1 ครั้ง) ที่คูณด้วย 1 (เพราะว่า $W_N^0 = 1$)



รูปที่ 2.10 การแปลง 8 จุดโดยใช้หน่วยคำนวณผีเสื้อที่ลดรูปแล้ว (DIT)

2.2.2.3 การกลับบิตและภาระการคำนวณ (Bit Reversing & Computation Load)

การแปลงฟูเรียร์เต็มหน่วยแบบเร็วโดยอาศัยการลดทอนทางเวลา ตำแหน่งข้อมูลทางด้านเอ้าท์พุทจะเรียงกันตามลักษณะปกติ แต่ตำแหน่งข้อมูลทางด้านอินพุทจะบิดเบือนไป (เช่น 0,4,2,6,1,5,3,7 ในกรณีการแปลง 8 จุด) ก่อนที่จะเริ่มการคำนวณ จะต้องจัดวางข้อมูลในตำแหน่งที่เหมาะสมก่อน การจัดตำแหน่งนี้มีกฎเกณฑ์ที่เรียกว่าการกลับบิต ดังจะยกตัวอย่างตามตารางข้างล่างนี้

ตารางที่ 2.3 การกลับบิตดัชนีข้อมูล

ตำแหน่ง	รหัสฐานสอง	รหัสที่กลับบิต	ดัชนีข้อมูล
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

ในการแปลงฟูเรียร์เต็มหน่วยแบบเร็ว N จุด หากใช้ไมโครโปรเซสเซอร์จะต้องเสียส่วนเก็บข้อมูล 5N ตำแหน่ง (2N สำหรับข้อมูลอินพุท 2N สำหรับข้อมูลเอ้าท์พุท และ N สำหรับแฟกเตอร์ตัวบิด) และสามารถลดเหลือ 3N ตำแหน่ง หากเป็นการคำนวณแบบแทนที่ (Implace Computation) เพราะว่าบางครั้งหากไม่ได้นำข้อมูลอินพุทไปใช้ต่อ ตำแหน่งนั้นก็จะใช้เก็บผลการคำนวณในแต่ละรอบจนได้เอ้าท์พุท สังเกตว่าแม้ข้อมูลอินพุทจะเป็นจำนวนจริง แต่ในระหว่างการคำนวณจะทำให้ได้ผลลัพท์เป็นจำนวนเชิงซ้อนเสมอ สำหรับ N ตำแหน่งส่วนของส่วนเก็บข้อมูลแฟกเตอร์ตัวบิด N/2 ใช้สำหรับเก็บ $\cos n\theta$ และ N/2 ใช้สำหรับเก็บ $\sin n\theta$ เมื่อ $\theta = 2\pi/N$ และ $n = 0.....N/2-1$

การแปลงฟูเรียร์เต็มหน่วยแบบเร็ว N จุด จะมีครั้งผ่านทั้งหมด $\log_2 N$ ครั้งผ่าน มีหน่วยคำนวณผีเสื้อทั้งหมด $N/2 \log_2 N$ หน่วย อาศัยการคูณจำนวนเชิงซ้อนทั้งหมด $N\log_2 N$ ครั้ง ตารางข้างล่างนี้แสดงให้เห็นภาระการคูณเมื่อเทียบกับการคำนวณโดยตรงซึ่งใช้การคูณ N^2 ครั้ง

จำนวนจุด (N)	N^2	$N \log_2 N$
8	64	24
16	256	64
64	4,096	384
256	65,536	2,048
512	262,144	4,608
1,024	1,048,576	10,240

ตารางที่ 2.4 จำนวนการคูณจำนวนเชิงซ้อนในการแปลงฟูเรียร์เต็มหน่วย

2.2.2.4 การคำนวณการแปลงฟูเรียร์แบบเร็วโดยการลดทอนทางความถี่

(Decimation in Frequency Fast Fourier Transform : DIF FFT)

ชุดข้อมูล $N = 2^m$ เมื่อ m คือเลขจำนวนเต็มบวกใด ๆ x(n) เป็นอินพุทที่ดัชนีต่าง ๆ การแปลงฟูเรียร์เต็มหน่วย สามารถเขียนได้โดย

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad ; \ 0 \le k \le N-1 \qquad (2.11)$$

$$= \sum_{n=0}^{\frac{N}{2}-1} X(n) W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n) W_N^{nk}$$

เมื่อให้ $r = n - \frac{N}{2}$ จะได้ $n = r + \frac{N}{2}$

ดังนั้น

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{r=0}^{\frac{N}{2}-1} x\left(r + \frac{N}{2}\right) W_N^{(r+\frac{N}{2})k}$$

เพราะว่า $W_N^{\frac{N}{2}k} = e^{-j\pi k} = (-1)^k$ โดยที่ทั้ง r และ n เป็นดัชนีอันดับเดียวกัน จึงได้

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + (-1)^k x\left(n + \frac{N}{2}\right) \right\} W_N^{nk}$$

จัดเป็นกลุ่มเทอมคู่และเทอมคี่ ในช่วง $0 \le r \le \frac{N}{2} - 1$

สำหรับเทอมคู่ $k = 2r$

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + (1)^{2r} x\left(n + \frac{N}{2}\right) \right\} W_N^{2nr}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + x\left(n + \frac{N}{2}\right) \right\} W_{N/2}^{nr}$$

สำหรับเทอมคี่ $k = 2r + 1$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + (1)^{2r+1} x\left(n + \frac{N}{2}\right) \right\} W_N^{n(2r+1)}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) - x\left(n+\frac{N}{2}\right) \right\} W_N^n W_{N/2}^{nr}$$

โดยกำหนดให้

$$g(n) = x(n) + x(n+\frac{N}{2})$$

$$h(n) = x(n) - x(n+\frac{N}{2})$$

ดังนั้น

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} g(n) W_{N/2}^{nr} \quad ; \quad 0 \le r \le \frac{N}{2}$$

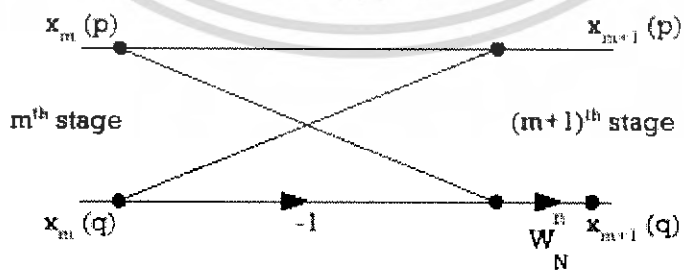
$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} h(n) W_N^n W_{N/2}^{nr} \quad ; \quad 0 \le r \le \frac{N}{2}$$

จะเห็นว่า X(k) ซึ่งเป็นการแปลง N จุด ได้จากการแปลง N/2 จุดจากตัวอย่าง $g(n)$ และ $h(n)W_N^n$ การกระทำสามารถที่จะแบ่งครึ่งออกไปได้อีกเรื่อย ๆ จนเหลือ 1 คู่ ในขั้นตอนสุดท้าย แต่ละคู่ของการแปลงสามารถที่จะเขียนเป็นหน่วยคำนวณผีเสื้อได้เช่นเดียวกัน

$$x_{m+1}(p) = x_m(p) + x_m(q) \qquad (2.12)$$

$$x_{m+1}(q) = \{x_m(p) - x_m(q)\} W_N^n \qquad (2.13)$$

ซึ่งสามารถที่จะเขียนกราฟการไหลของสัญญาณได้ดังรูป



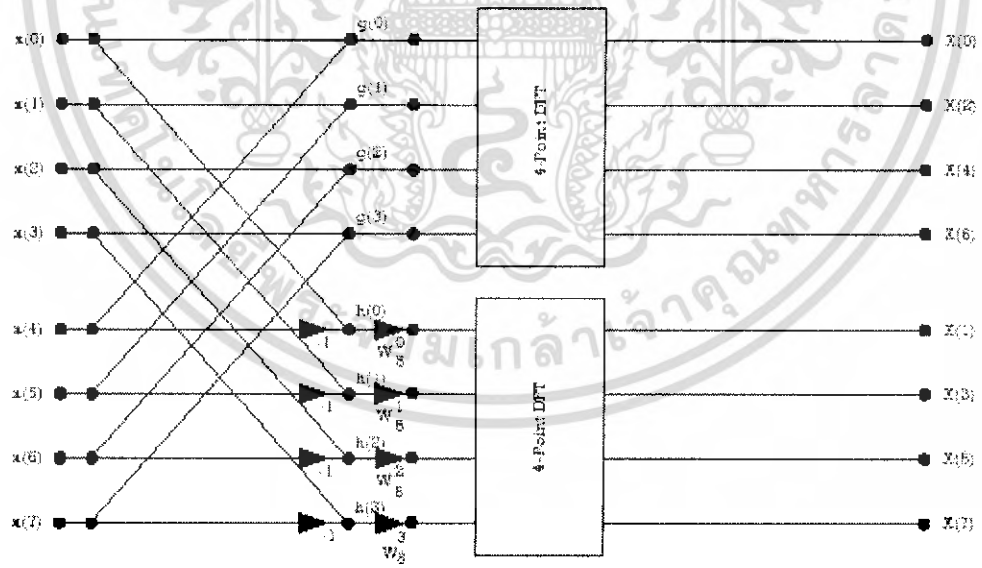
รูปที่ 2.11 หน่วยคำนวณผีเสื้อของการแปลงฟูเรียร์โดยการลดทอนทางความถี่

จำนวนครั้งผ่านที่ต้องการคือ $\log_2 N$ โดยแต่ละครั้งผ่านจะมีการคำนวณในผีเสื้อ N/2 หน่วย ดังนั้นจะสังเกตเห็นได้ว่าการแปลงฟูเรียร์ไม่ว่าจะอาศัยการลดทอนทางเวลาหรือการลดทอนทางความถี่ โครงสร้างการคำนวณจะใกล้เคียงกันมาก

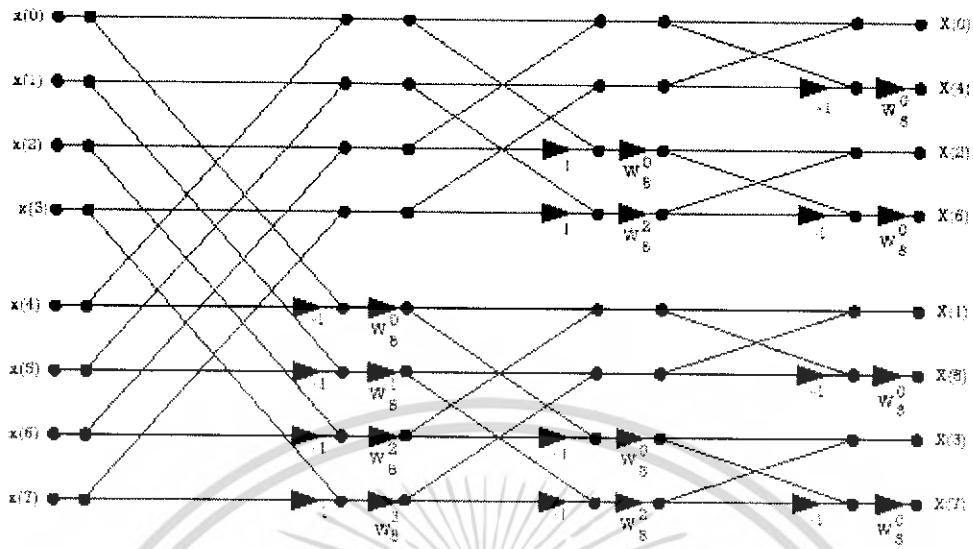
ยกตัวอย่างการแปลงฟูเรียร์เต็มหน่วย 8 จุด โดยการลดทอนทางความถี่ จะสามารถแบ่งครั้งผ่านได้เป็น 3 ครั้งผ่าน ครั้งผ่านแรกจะทำงานกับข้อมูลทั้งหมด แฟกเตอร์ตัวบิดที่ใช้คือ W_8^0, W_8^1, W_8^2 และ W_8^3 โดยที่ข้อมูลที่เข้ามานั้นจัดลำดับอย่างปกติ ในครั้งผ่านที่ 2 จะแบ่งผลที่ได้มาเป็น 2 กลุ่ม กลุ่มละ 4 จุดข้อมูล

การแปลงฟูเรียร์เต็มหน่วย 4 จุดนั้น สามารถแยกย่อยเป็นการแปลงฟูเรียร์เต็มหน่วย 2 จุด 2 ชุด แฟกเตอร์ตัวบิดที่ใช้คือ W_4^0 และ W_4^1 ที่เหลือจึงทำการคำนวณการแปลง 2 ชุด ทั้งหมด 4 ชุด เมื่อใช้ความสัมพันธ์ $W_4^1 = W_8^2, W_4^0 = W_8^0$ และ $W_2^0 = W_8^0$ กราฟไหลสัญญาณของการแปลงฟูเรียร์เต็มหน่วย 8 จุด โดยการลดทอนทางความถี่จะแสดงได้ดังรูป

สังเกตว่าในการแปลงฟูเรียร์เต็มหน่วยเร็วโดยการลดทอนทางความถี่นั้น แม้ว่าข้อมูลทางด้านอินพุทจะเรียงมาอย่างปกติ แต่การแยกดัชนีคู่และดัชนีคี่ในแต่ละครั้งผ่านจะทำให้เกิดการสลับตำแหน่ง สุดท้ายแล้วข้อมูลที่ได้ทางเอ้าท์พุทจะเรียงกันในลักษณะกลับบิทซึ่งจะต้องมีการสลับคืนจึงจะได้การเรียงแบบธรรมชาติ



รูปที่ 2.12 กราฟไหลสัญญาณของการคำนวณโดยการลดทอนทางความถี่ขนาด 8 จุดครั้งผ่านแรก



รูปที่ 2.13 กราฟไหลสัญญาณแปลงฟูเรียร์เต็มหน่วยเร็วโดยการลดทอนทางความถี่ขนาด 8 จุด

2.2.2.5 การแปลงฟูเรียร์แบบเร็วเมื่อค่า N แยกเป็นตัวประกอบได้

(Composite Block Length Algorithm FFT)

วิธีการของ Cooley & Tukey เป็นวิธีการที่มีประสิทธิภาพในการคำนวณที่ดีมาก แต่ชุดของข้อมูลต้องมีค่า $N = 2^m$ หาก $N \neq 2^m$ แต่สามารถที่จะแยกเป็นตัวประกอบปฐมได้ เช่น 12=3×2×2,15=5×3,21=3×7 เป็นต้น ก็สามารถที่จะใช้ขั้นตอนวิธีแยกองค์ประกอบ (Prime Factor Algorithm , PFA) นี้ได้ โดยหลักการก็คือข้อมูลทั้งหมดจะถูกแบ่งเป็นชุดย่อย ๆ สั้น ๆ หลายชุดจำนวนเท่า ๆ กัน ทำการแปลงแต่ละชุดแล้วนำมารวมกัน แต่เนื่องจากการแปลงฟูเรียร์ในแต่ละชุดจะทำให้เกิดการสลับตำแหน่ง ก่อนการแปลงและหลังการแปลง จึงต้องทำการเรียงตำแหน่ง (Index mapping) การแปลงฟูเรียร์ในชุดสั้น ๆ นั้นอาจใช้การคำนวณโดยตรง หรืออาจใช้ขั้นตอนวิธีสำหรับชุดสั้น ๆ (Short Length Algorithm , SLA) โดยเฉพาะ เช่น วิธีการของวินโนกราด (Winograd's Fourier Transform Algorithm , WFTA)

พิจารณา $N = N_1 \times N_2$ โดยที่ N_1 และ N_2 ไม่มีตัวประกอบร่วม หรือ $(N_1, N_2) = 1$ ให้ i และ kเป็นดัชนีทางอินพุทและเอ้าท์พุทตามลำดับ การแปลงฟูเรียร์เต็มหน่วยเขียนได้โดย

$$X(k) = \sum_{i=0}^{N-1} X(i) W_N^{ik}, \qquad W_N = e^{\frac{-2j\pi}{N}}$$

ให้ A_1, A_2, A_3และ A_4 เป็นจำนวนเต็มใด ๆ (A_N หมายถึง A มอดูโล ด้วย N) i_1, i_2 และ k_1, k_2 เป็นดัชนีในแต่ละมิติของ N_1 และ N_2 ตามลำดับ จะได้

$$i = \left(A_1 i_1 + A_2 i_2\right)_N$$

$$k = \left(A_3 k_1 + A_4 k_2\right)_N$$

ดังนั้น

$$x(i_1, i_2) = x\left(A_1 i_1 + A_2 i_2\right)_N$$

$$X(k_1, k_2) = X\left(A_3 k_1 + A_4 k_2\right)_N$$

จะได้

$$X(k_1, k_2) = \sum_{i_1=0}^{N_1-1} \sum_{i_2=0}^{N_2-1} x\left(i_1, i_2\right) W_N^{A_1 A_3 i_1 k_1} W_N^{A_1 A_4 i_1 k_2} W_N^{A_2 A_3 i_2 k_1} W_N^{A_2 A_4 i_2 k_2}$$

มีวิธีการเลือกค่าที่เหมาะสมของ A_1, A_2, A_3และ A_4 อยู่หลายวิธี แต่วิธีที่นิยมคือการกำหนดให้ $(A_1 A_3)_N = N_2, (A_2 A_4)_N = N_1$ และ $(A_1 A_4)_N = (A_2 A_3)_N = 0$ ซึ่งยังผลให้การแปลงฟูเรียร์ข้างบนนั้น เป็นการแปลงใน 2 มิติ คือ

$$X(k_1, k_2) = \sum_{i_2=0}^{N_2-1} \sum_{i_1=0}^{N_1-1} x\left(i_1, i_2\right) W_{N_1}^{i_1 k_1} W_{N_2}^{i_2 k_2}$$

ทำโดยการคำนวณแบบแทนที่ การเลือกค่าตัวคงที่ กำหนดดังนี้

$$A_1 = N_2 \quad \text{และ} \quad A_2 = N_1$$

$$A_3 = \left(N_2 \left(N_2\right)_{N_1}^{-1}\right)_N \quad \text{และ} \quad A_4 = \left(N_1 \left(N_1\right)_{N_2}^{-1}\right)_N$$

เมื่อ $\left(\_\right)_N^{-1}$ คือ ส่วนกลับการคูณมอดูโล N (Multiplicative inverse modulo N) เช่น $\left(5\right)_3^{-1}$=2 คือ (5×2)/3 จะเหลือเศษ 1 ดังนั้น การจัดตำแหน่งทางอินพุท จะกำหนดได้โดย

$$i = (N_2 i_1 + N_1 i_2)_N$$

และการจัดตำแหน่งทางเอ้าท์พุท จะกำหนดได้โดย

$$k = \left(\left(N_2 (N_2)_{N_1}^{-1}\right)_N k_1 + \left(N_1 (N_1)_{N_2}^{-1}\right)_N k_2\right)_N$$

2.3 คุณสมบัติของเสียงดนตรี

คุณสมบัติเฉพาะตัวของโน้ตแต่ละตัวนั้นมีอยู่ 2 คุณสมบัติด้วยกัน คือ ระดับเสียง (Pitch) และระยะเวลา (Duration) ดังนั้นการวิเคราะห์อย่างเหมาะสมจะทำให้สามารถระบุระดับเสียงและระยะเวลาของโน้ตแต่ละตัวได้

2.3.1 ระดับเสียง (Pitch)

ระดับเสียง ก็คือลักษณะของตัวโน้ตที่สามารถจัดเรียงเป็นระดับของเสียงดนตรีได้ ซึ่งคำจำกัดความที่องค์กร America National Standards ให้ไว้ก็คือ "ระดับเสียงคือ คุณสมบัติของการได้ยินในส่วนของเสียงซึ่งสามารถจัดแยกออกเป็นระดับจากต่ำไปสูง"

แนวคิดในการทำความรู้จักกับเสียงก็คือ อันดับแรกจะแนะนำให้รู้จักกับเสียงดนตรี ซึ่งความสามารถในการได้ยินของคนเรานั้นจะสามารถแยกระดับเสียงที่ต่างกันได้ค่อนข้างที่จะสูง (ประมาณ 1,500 ระดับเสียง) การได้ยินของคนธรรมดานั้นจะสามารถรับรู้ความแตกต่างระหว่างเสียงที่มีความถี่ 440 Hz และ 441 Hz ได้ ซึ่งเสียงที่ความถี่ 440 Hz นี้รู้จักกันดีคือเสียง A4 หรือ ลา และโน้ตตัวถัดไปของเปียโน (A#4) มีความถี่ 466.2 Hz ทำให้สามารถสรุปได้อย่างคล่าว ๆ ว่า การได้ยินของคนสามารถจะรับรู้ความแตกต่างของโน้ต 2 ตัวที่มีระดับเสียงที่ต่างกัน 25 ระดับเสียงได้ ซึ่งคนส่วนใหญ่มีความสามารถนี้

ในทางวิทยาศาสตร์ แนวความคิดเกี่ยวกับระดับเสียง คือ ความสัมพันธ์เกี่ยวกับคุณสมบัติของเสียงทางกายภาพที่เรียกว่า "ความถี่" ซึ่งตามทฤษฎีนั้นคนเราสามารถได้ยินเสียงในช่วงความถี่ 20 Hz ถึง 20,000 Hz แต่ในความจริงแล้วใช้ได้กับเฉพาะเด็กและวัยรุ่นเท่านั้น เพราะเมื่อคนเรามีอายุมากขึ้น ความสามารถในการได้ยินก็จะลดน้อยลงไปด้วย ดังนั้นในช่วงอายุ 20 ปี ความถี่สูงสุดที่เราสามารถรับฟังได้จะตกลงมา เหลือเพียง 16,000 Hz และในผู้สูงอายุก็จะมีความสามารถในการได้ยินอยู่ระหว่าง 20 Hz ถึง 8,000 Hz เท่านั้น

ระดับเสียงนั้นจะมีความสัมพันธ์กันกับความดังของเสียง เช่น ระดับเสียงสูง (มากกว่า 2,000 Hz) จะสูงขึ้นเมื่อความดังของมันเพิ่มขึ้น และในทำนองเดียวกัน ระดับเสียงต่ำ (น้อยกว่า 2,000 Hz) จะต่ำลงอีกเมื่อลดความดังของมันลง ซึ่งปรากฏการณ์นี้ รู้จักกันในชื่อของ "Stevens's rule" หากเสียงมีความถี่น้อยกว่า 1,000 Hz หรือเสียงมีความถี่ สูงกว่า 2,000 Hz ก็จะเห็นปรากฏการณ์นี้ได้อย่างชัดเจน แต่หากใช้ความถี่อยู่ในช่วงระหว่าง 1,000 Hz กับ 2,000 Hz จะไม่สามารถสังเกตเห็นความแตกต่างได้

ความน่าสนใจอื่น ๆ เกี่ยวกับปรากฏการณ์ของเสียง เช่น ปรากฏการณ์ Doppler Effect ที่ว่าหากระยะทางระหว่างแหล่งกำเนิดเสียงกับผู้ฟังนั้นเปลี่ยน ความสามารถในการรับรู้ระดับเสียงของ

ผู้ฟังก็จะเปลี่ยนไปด้วย เช่น ระยะทางระหว่างแหล่งกำเนิดเสียงกับผู้ฟังนั้นเพิ่มขึ้น ความสามารถในการรับรู้ระดับเสียงของผู้ฟังก็จะลดลง

2.3.2 ความสัมพันธ์ระหว่างตัวโน้ตและความถี่

ได้มีการจัดแบ่งกลุ่มของตัวโน้ต 12 ตัวเป็นกลุ่มขึ้นมา เรียกว่า หนึ่งออคเทฟ (Octave) ซึ่งประกอบไปด้วยโน้ต A, A#, B, C, C#, D, D#, E, F, F#, G, และ G# โน้ตเหล่านี้จะวนซ้ำกันไป และจะระบุตัวเลขเฉพาะของแต่ละทบเสียงซึ่งในช่วงของเสียงดนตรีมีทั้งหมด 10 ออคเทฟด้วยกัน จาก C0 (ประมาณ 16 Hz) ถึง C10 (ประมาณ 16 KHz)

เราสามารถแยกระดับเสียงดนตรีได้เนื่องจากแต่ละระดับเสียงจะมีความถี่ที่ต่างกันออกไป ดังนั้นจึงได้มีการคำนวณค่าความถี่ของแต่ละระดับเสียงไว้ ตามมาตรฐานของดนตรีตะวันตกถือว่าโน้ตตัวลา (A) ในออคเทฟที่ 4 ซึ่งมีความถี่ 440 Hz เป็นโน้ตพื้นฐาน และโน้ตตัวอื่น ๆ ก็จะมีความสัมพันธ์กับความถี่ 440 Hz นี้ ความถี่ของโน้ตตัวลาในแต่ละออคเทฟจะเป็นจำนวนเท่าของความถี่ 440 Hz เช่น โน้ตตัวลาในออคเทฟที่ 5 จะมีความถี่เป็นสองเท่าของโน้ตตัวลาในออคเทฟที่ 4 ส่วนโน้ตอื่น ๆ ในออคเทฟที่ 4 ก็จะมีความสัมพันธ์กับโน้ตตัวลาในออคเทฟที่ 4 ในเชิงเอ็กโพเนนเชียล (Exponential) ดังสมการที่ 2.17

$$f_i = 440 \times 2^{\frac{i}{12}} \qquad (2.17)$$

โดย f_i คือ ความถี่ของตัวโน้ต

i คือ หมายเลขของตัวโน้ต

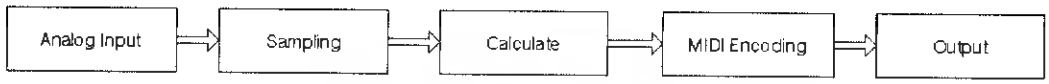
เช่น โน้ต A4 ค่า i จะเท่ากับ 0 โดยค่า i จะเพิ่มขึ้นหนึ่งค่า เมื่อโน้ตสูงขึ้นครึ่งเสียง และ i จะมีค่าลดลงหนึ่งค่า เมื่อโน้ตต่ำลงครึ่งเสียง

ตารางที่ 2.5 ความถี่ของโน้ตต่าง ๆ ในออคเทฟที่ 4

Musical Note	Frequency	
	Equation	Hz
A4	440	440.0000000000
A#4	$440\times2^{\frac{1}{12}}$	466.1637615181
B4	$440\times2^{\frac{2}{12}}$	493.8833012561
C5	$440\times2^{\frac{3}{12}}$	523.2511306012
C#5	$440\times2^{\frac{4}{12}}$	554.3652619537
D5	$440\times2^{\frac{5}{12}}$	587.3295358348
D#5	$440\times2^{\frac{6}{12}}$	622.2539674442
E5	$440\times2^{\frac{7}{12}}$	659.2551138257
F5	$440\times2^{\frac{8}{12}}$	698.4564628660
F#5	$440\times2^{\frac{9}{12}}$	739.9888454233
G5	$440\times2^{\frac{10}{12}}$	783.9908719635
G#5	$440\times2^{\frac{11}{12}}$	830.6093951599
A5	$440\times2^{\frac{12}{12}}$	880.0000000000

บทที่ 3
รายละเอียดในการออกแบบและการสร้าง

3.1 หลักการ



รูปที่ 3.1 บลอกไดอแกรมหลักการทำงาน

จากรูปจะทำการรับค่าจากสัญญาณอนาลอกโดยโมดูลแปลงสัญญาณอนาลอกเป็นดิจิตอล ใช้กระบวนการแปลงฟูเรียร์อย่างเร็วเพื่อทำการแยกองค์ประกอบทางความถี่ของสัญญาณ และจับค่าของความถี่ที่มีแอมปลิจูดสูงที่สุด และนำไปเปรียบเทียบว่าตรงกับโน้ตดนตรีตัวไหน เมื่อได้ค่าแล้วจะส่งสัญญาณของโน้ตตัวนั้นส่งไปยังพอร์ตเอาท์พุท ในรูปแบบของมิดี้

3.2 รายละเอียดการออกแบบ

ความถี่อินพุทจะรับตั้งแต่ค่า 329.6275 ~ 1318.5102 Hz ซึ่งเป็นโน้ตระหว่าง E3 – E5 ตามมาตรฐานดนตรีสากล ซึ่งเป็น pitch ที่ค่อนข้างสูงสำหรับเครื่องดนตรี เนื่องจากที่ Pitch สูง จะใช้เวลาในการ Sampling เก็บข้อมูลน้อยกว่าเวลาในการ Sampling ที่ความถี่ต่ำๆ

ใช้ Fast Fourier Transform ในการคำนวณหาความถี่ที่มีกำลังสูงที่สุด เนื่องจาก Fast Fourier Transform เป็น Algorithm ที่มีขั้นตอนในการคำนวณน้อยกว่าวิธีอื่น ในความเป็นจริง ถ้าต้องการค่าที่แม่นยำจะต้องคำนวณค่าในช่วง 1 ลูกคลื่นเท่านั้น แต่การจะทำเช่นนั้นจะต้องมีหน่วยประมวลผลที่ความเร็วมากๆ และใช้หน่วยความจำมาก จึงยอมให้ประมวลผลค่าจากหลายลูกคลื่นจะเกิดความผิดเพี้ยนไปบ้าง โดยเฉพาะที่ความถี่ต่ำๆ โดยใช้ N = 128 เพื่อให้เมื่อสเกลอยู่ในแกนของความถี่จะยังละเอียดเพียงพอที่จะแยกแยะและประมาณค่าความถี่ในช่วงที่ต้องการได้

ที่ทำในช่วง 2 Octaves เนื่องจากข้อจำกัดของความเร็วในการคำนวณและหน่วยความจำของหน่วยประมวลผล เนื่องจาก Fast Fourier Transform เป็น Algorithm ที่มีใช้ตัวแปรทศนิยมในการคำนวณ ซึ่งตัวแปร Float ในภาษา C ใช้หน่วยความจำถึง 32 บิต หรือ 4 ไบท์ และเนื่องจากบางตัวแปรเป็นค่าเลขเชิงซ้อนจึงต้องใช้ Float 2 ตัวต่อ 1 ตัวแปร และการที่ใช้ Fast Fourier Transform ที่ N = 128 จะต้องใช้หน่วยความจำอย่างน้อยเท่ากับ 128 x 2 x 4 = 1024 bytes เป็นอย่างต่ำ ซึ่งราคา

ของคอนโทรลเลอร์ที่มีหน่วยความจำสูงจะมีราคาแพงมากเนื่องจากเป็นหน่วยความจำชนิด SRAM ที่มีโครงสร้างซับซ้อน

ใช้ความถี่ในการ Sampling = 2666.666 Hz ซึ่งเป็นค่าประมาณ 2 เท่าของความถี่สูงสุดจากอินพุท เนื่องจากเมื่อคำนวณโดยใช้ Fast Fourier Transform แล้วจะทำจะอ่านค่าได้ตั้งแต่ 0Hz ~ 1333.333Hz โดยจะ scale เป็น 64 ช่องมีความห่างระหว่างช่องเท่ากับ 1333.333/64 = 20.833 และเมื่อนำความห่างของแต่ละช่องมาพลอทเทียบกับความถี่ของตัวโน้ต จะได้ค่าที่ใกล้เคียงตัวโน้ตดังตารางที่ 3.1 จากตารางจะเห็นได้ว่าในช่วงความถี่ต่ำๆจะมีจุดที่ซ้ำกันบ้างในแต่ละตัวโน้ต แต่ก็ยังอยู่ในขอบเขตที่จะแยกแยะและประมาณระดับเสียงของตัวโน้ตได้ เนื่องจากไม่มีค่าความถี่ที่อยู่ในช่องเดียวกัน จากค่าที่ได้จะเห็นว่าช่วงห่างของโน้ตแต่ละตัวกับช่องที่ได้จากการแปลงจะห่างจากขอบบนขอบล่างของช่องไม่เท่ากันจึงต้องมีการทดลองเพื่อหาค่าความแตกต่างของแอมปลิจูดระหว่างตัวโน้ตที่ยอมรับได้ เพื่อที่จะสามารถระบุโน้ตได้แม่นยำขึ้น

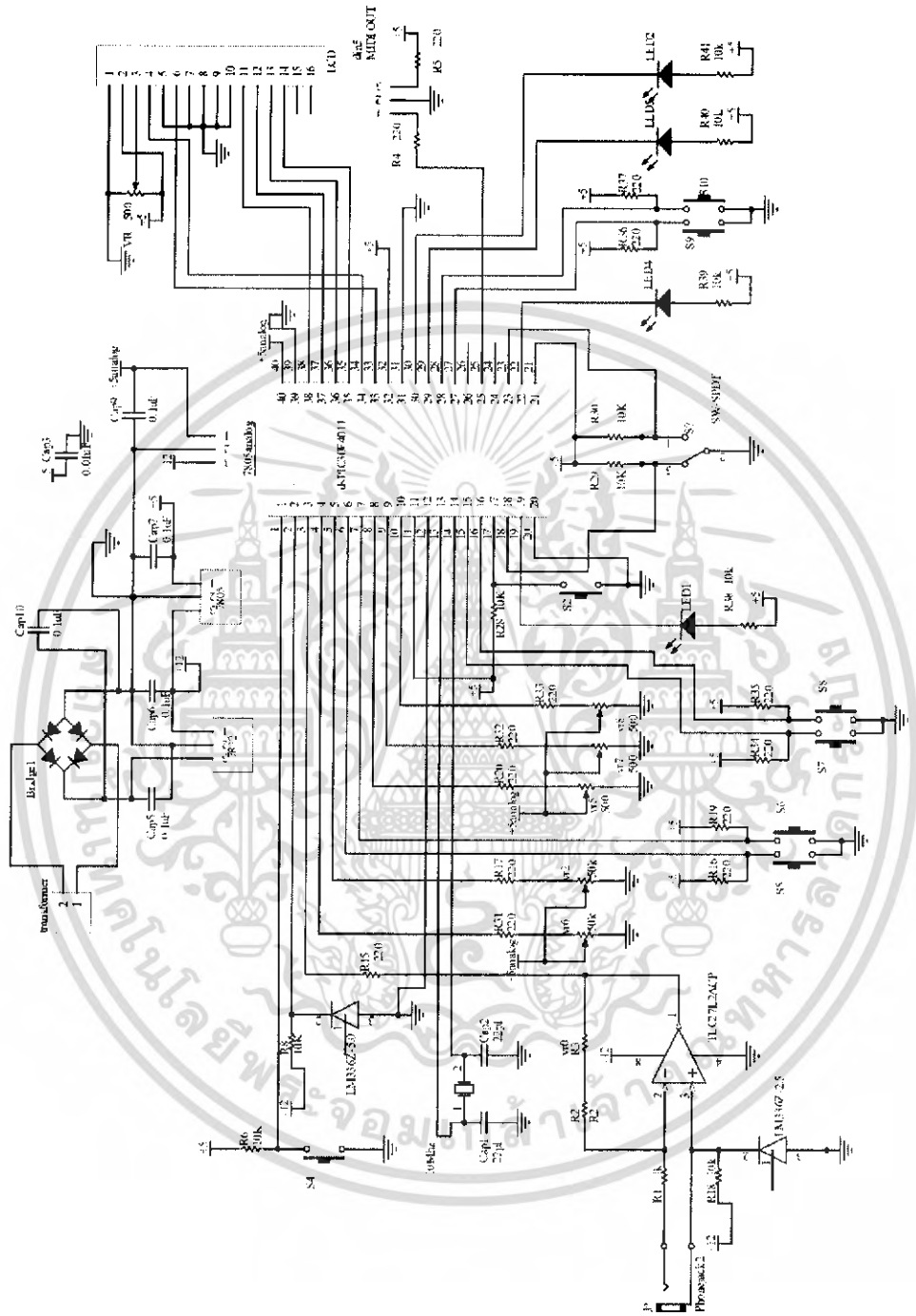
ตารางที่ 3.1 ตารางเปรียบเทียบช่วงความถี่

โน้ต	ความถี่ของตัวโน้ต	ความถี่ขอบล่างของช่อง	ความถี่ขอบบนของช่อง
E4	329.6275	312.50(15)	333.33(16)
F4	349.2282	333.33(16)	354.17(17)
F#4	369.9944	354.17(17)	375.00(18)
G4	391.9954	375.00(18)	395.83(19)
G#4	415.3046	395.83(19)	416.67(20)
A4	440.0000	437.50(21)	458.33(22)
A#4	466.1637	458.33(22)	479.17(23)
B4	493.8833	479.17(23)	500.00(24)
C5	523.2511	520.83(25)	541.67(26)
C#5	554.3652	541.67(26)	562.50(27)
D5	587.3295	583.33(28)	604.17(29)
D#5	622.2539	604.17(29)	625.00(30)
E5	659.2551	645.83(31)	666.67(32)
F5	698.4564	687.50(33)	708.33(34)

F#5	739.9888	729.17(35)	750.00(36)
G5	783.9908	770.83(37)	791.67(38)
G#5	830.6093	812.50(39)	833.33(40)
A5	880.0000	875.00(42)	895.83(43)
A#5	932.3275	916.67(44)	937.50(45)
B5	987.7666	979.17(47)	1000.00(48)
C6	1046.5022	1041.70(50)	1062.50(51)
C#6	1108.7305	1104.20(53)	1125.00(54)
D6	1174.6590	1166.70(56)	1187.50(57)
D#6	1244.5079	1229.20(59)	1250.00(60)
E6	1318.5102	1312.50(63)	1333.30(64)

ในการออกแบบได้เลือกใช้ไมโครคอนโทรลเลอร์ชนิด dsPIC30F4011 เนื่องจากเป็นไมโครคอนโทรลเลอร์ 16 บิตที่มีความเร็วสูงถึง 20 MIPS (Million Instruction per Second) และมีหน่วยความจำโปรแกรมแบบ Flash ขนาด 48 กิโลไบท์ และมีหน่วยความจำ SRAM ที่มีความเร็วสูง ขนาด 2048 ไบท์ ซึ่งเพียงพอต่อการประมวลผล Fast Fourier Transform ที่ N = 128 และมีความสามารถในการติดต่ออุปกรณ์ภายนอก ต่างๆ ที่จำเป็นต่อการส่งข้อมูลมิดี้ เช่น UART (Universal Asynchronous Receiver Transmitter) ซึ่งเป็นมาตรฐานการส่งข้อมูลแบบอนุกรมตามรูปแบบของมิดี้ แต่เนื่องจากมิดี้เป็นใช้การส่งสัญญาณควบคุมทางเดียว จึงใช้แค่ตัวส่งเพียงอย่างเดียวที่บอร์ดเรท 31250

มีการต่อ LCD เพื่อเป็น User Interface ให้กับผู้ใช้ได้ใช้งานได้สะดวกขึ้นและเพื่อลดจำนวนปุ่มลง โดยสามารถกำหนดค่าตัวแปรต่างๆของตัวโน้ต โดยใช้เมนู ซึ่งจะสามารถลดปุ่มลงได้เหลือเพียง 1 ปุ่ม คือปุ่ม Enter และลูกบิดซ้ายขวา สามารถเขียนเป็นรูปวงจรได้ดังรูปที่ 3.2



รูปที่ 3.2 รูปวงจรเครื่องแปลงสัญญาณมิดี้

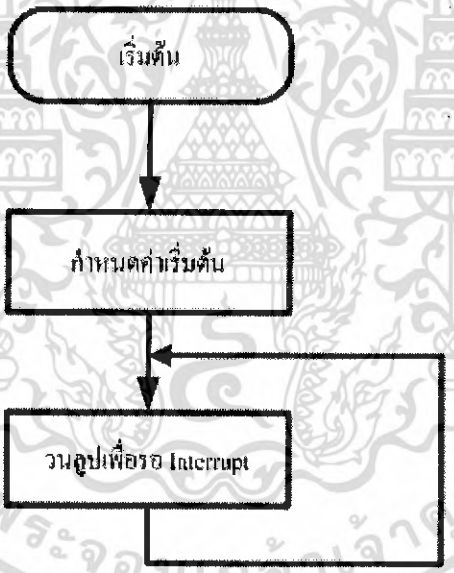
จากรูปวงจรใช้คริสตอลความถี่ 10 MHz เป็นตัวสร้างสัญญาณนาฬิกา แล้วภายในตัวไมโครคอนโทรเลอร์จะมีเฟสล็อคลูป x8 ทำให้ได้สัญญาณนาฬิกาของระบบ 80 MHz เนื่องจากไมโครคอนโทรเลอร์ชนิดนี้ต้องการ 4 รอบสัญญาณนาฬิกาต่อ 1 คำสั่ง ดังนั้นจะมีความเร็วในการประมวลผล 80 / 4 = 20 MIPS

ทางด้านอินพุทจะรับสัญญาณอนาลอกด้วย Connecter ชนิด TRS 1/4 นิ้วตัวเมีย พอร์ตมิดี้เอาท์จะใช้หัวต่อแบบ DIN-5 ซึ่งเป็นหัวต่อที่ใช้สำหรับสัญญาณมิดี้ตามมาตรฐาน

ภาคไฟเลี้ยงจะแบ่งออกเป็น 3 ส่วนคือ +12 V , +5 V ที่ใช้เลี้ยงในส่วนอุปกรณ์ดิจิตอล และ+5 V ที่ใช้เลี้ยงในส่วนอุปกรณ์อนาลอก

3.3 การออกแบบซอฟท์แวร์

เป็นส่วนของอัลกอริทึ่มในตัวไมโครคอนโทรเลอร์จะดำเนินตามโฟลว์ชาร์ตตามรูปข้างล่าง



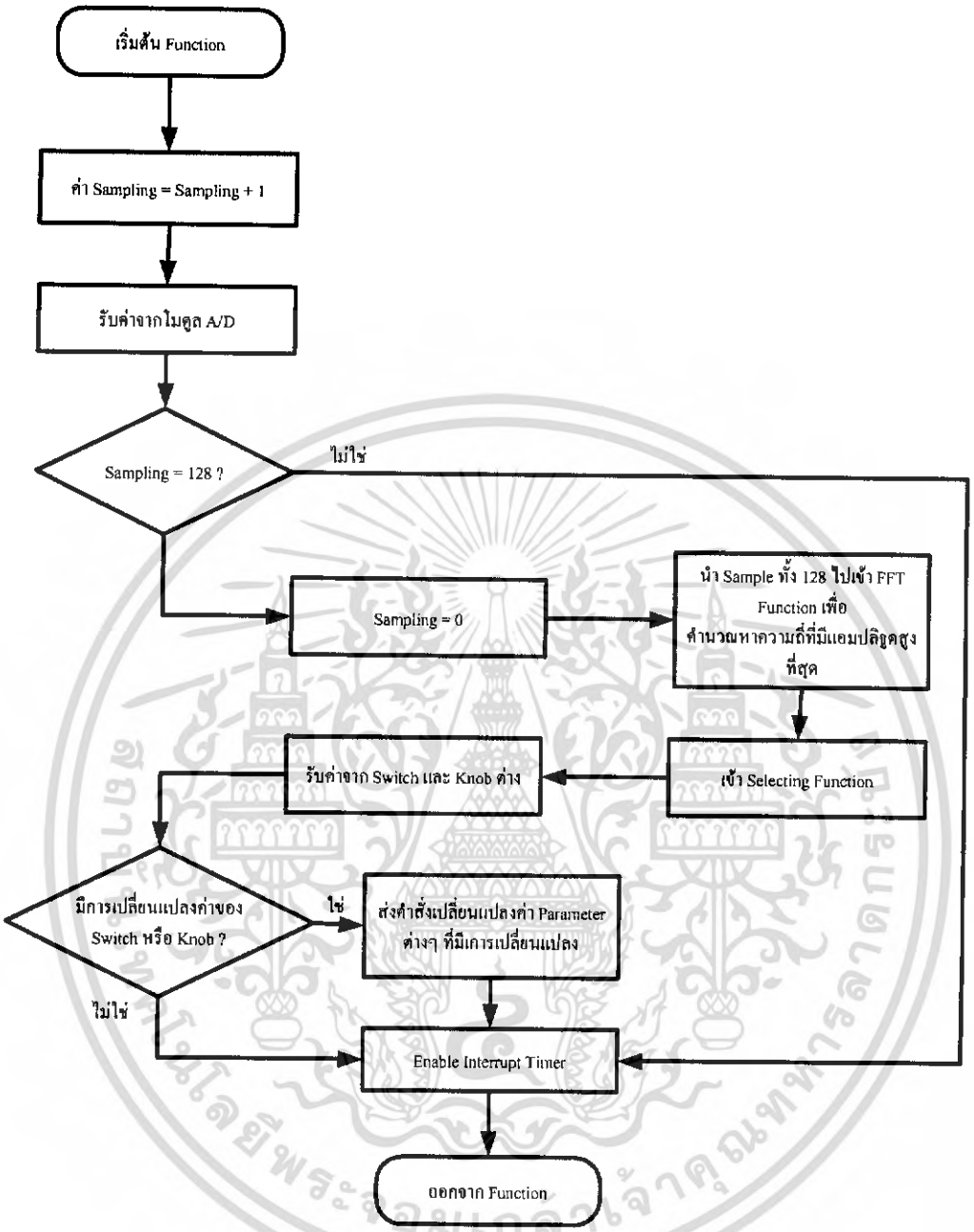
รูปที่ 3.3 โฟลวชาร์ตของโปรแกรมหลัก

จะเห็นว่าตัวโปรแกรมเมื่อกำหนดค่าเริ่มต้นเสร็จจะไม่มีการคำนวณตลอดเวลา จะรออินเตอร์รัปจาก Timer และจากภายนอกเท่านั้น เพื่อลดการสูญเสียพลังงานขณะสวิทช์ทำงาน และ

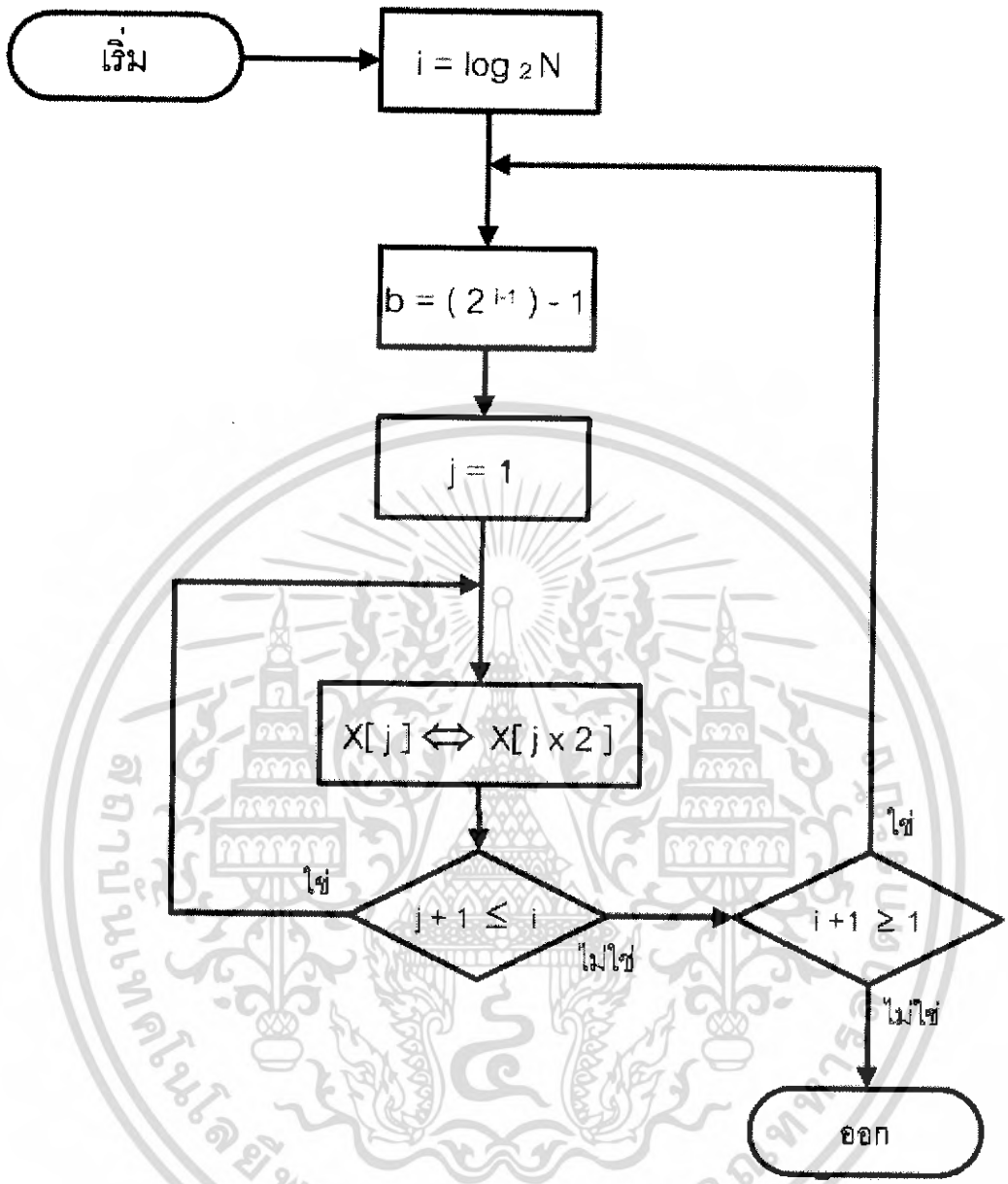
ง่ายต่อการออกแบบ โดยการตั้งค่า Sampling จะตั้งจาก Timer โดยให้คาบของการแซมปลิ้งเท่ากับ 0.375 ms โดยการตั้งให้เกิด Interrupt ทุกๆ 0.375 ms เพื่อจะไปเข้าฟังก์ชั่น

3.3.1ไทม์เมอร์อินเตอร์รัปต์

รับค่าจาก A/D มา เมื่อครบ 128 ค่าจะไปทำ Fast Fourier Transform เพื่อแยกหาองค์ประกอบความถี่ของสัญญาณที่แซมปลิ้งมาได้ เพื่อดูว่าความถี่มีแอมปลิจูดสูงที่สุดไปตรงกับโน้ตตัวไหน และจะส่งค่าโน้ตนั้นออกทางมิดี้เอาท์ และทำการตรวจความเปลี่ยนแปลงของสวิทช์และลูกบิดปรับค่าต่างๆ ถ้ามีการเปลี่ยนแปลง โปรแกรมจะทำการส่งค่าไปควบคุมทางมิดี้เอาท์อีกที จากนั้นทำการเคลียร์ค่าตัวแปรที่ใช้เก็บข้อมูลจากการแซมปลิ้ง และวงจรจะทำการแซมปลิ้งต่อ เมื่อสิ้นสุดการส่งและวนทำไปเรื่อยๆ ซึ่งขั้นตอนการทำงานจะอยู่ในโฟลวชาร์ตรูปที่ 3.4



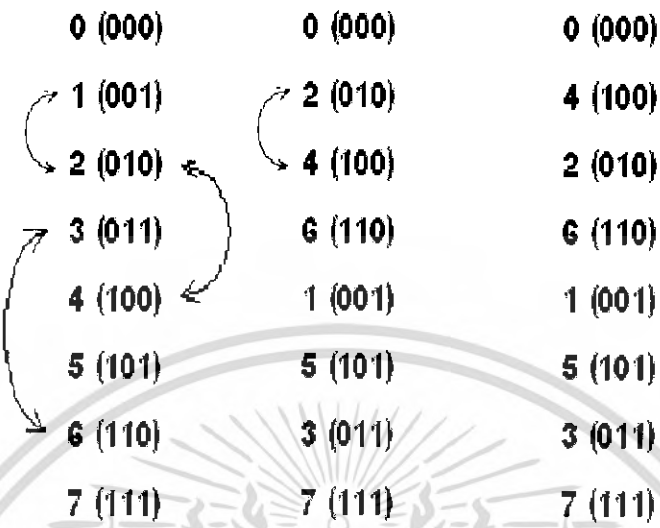
รูปที่ 3.4 โฟลวชาร์ตของฟังก์ชั่นอินเตอร์รัปของ Timer ใช้ในการ Sampling สัญญาณ



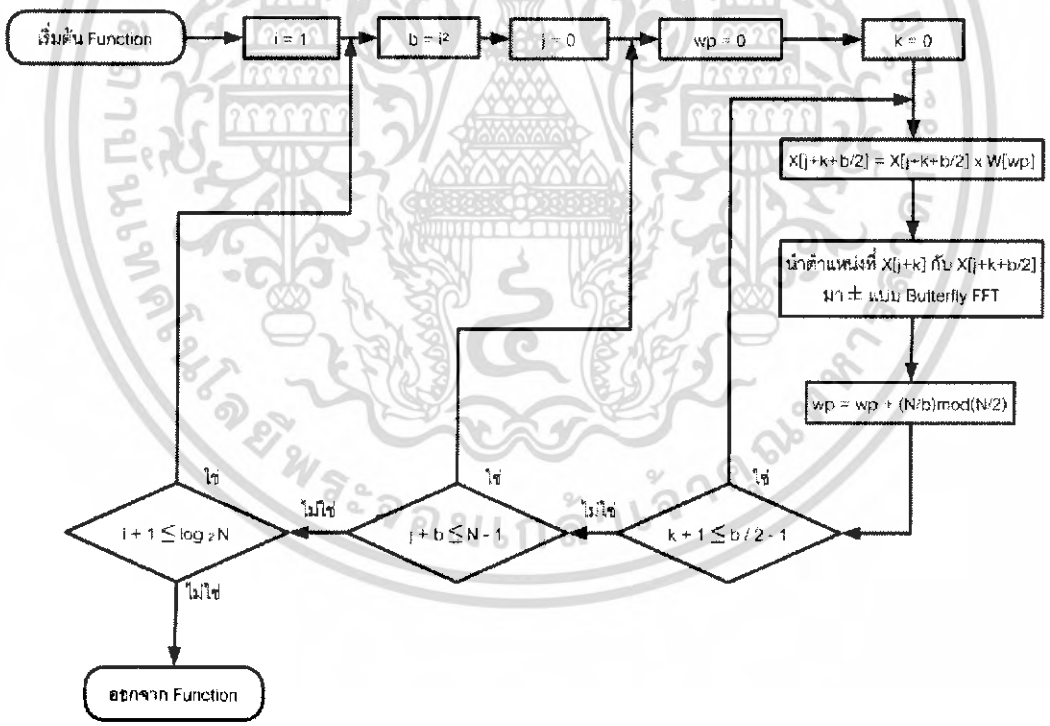
รูปที่ 3.5 โฟลวชาร์ตของฟังก์ชั่นกลับบิต

3.3.2 ฟังก์ชั่นการกลับบิต

ส่วนนี้เป็นส่วนฟังก์ชั่นกลับบิต โดยที่อินพุทของฟาส์ทฟูเรียร์ทรานสฟอร์มจะต้องเรียงลำดับแบบกลับบิตจึงต้องมีการเรียงลำดับใหม่ด้วยฟังก์ชั่นนี้ โดยใช้หลักการสลับตำแหน่งค่าที่ x และค่าที่ 2x โดย x จะมีค่าตั้งแต่ 0 ~ N/2 จนหมด และจะทำต่ออีกรอบในช่วง 0 ~ N/4 ไปเรื่อยๆ จนเหลือ 0 ~ 0 จะเรียงผลเป็นแบบกลับบิต ดังตัวอย่างในรูปที่ 3.6



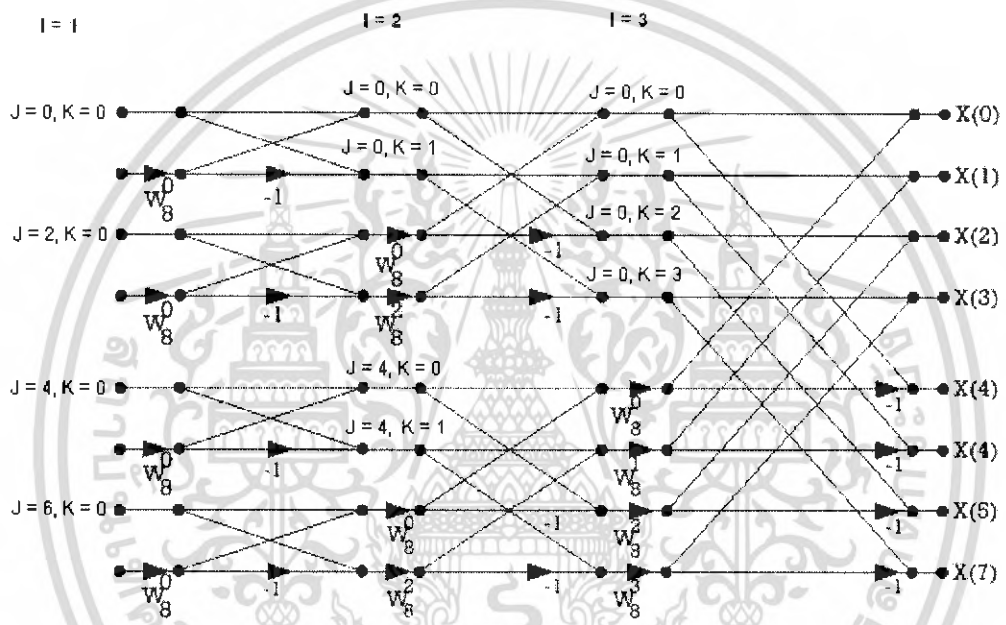
รูปที่ 3.6 ตัวอย่างการกลับบิต



รูปที่ 3.7 โฟลวชาร์ตของฟังก์ชั่นฟาส์ทฟูเรียร์

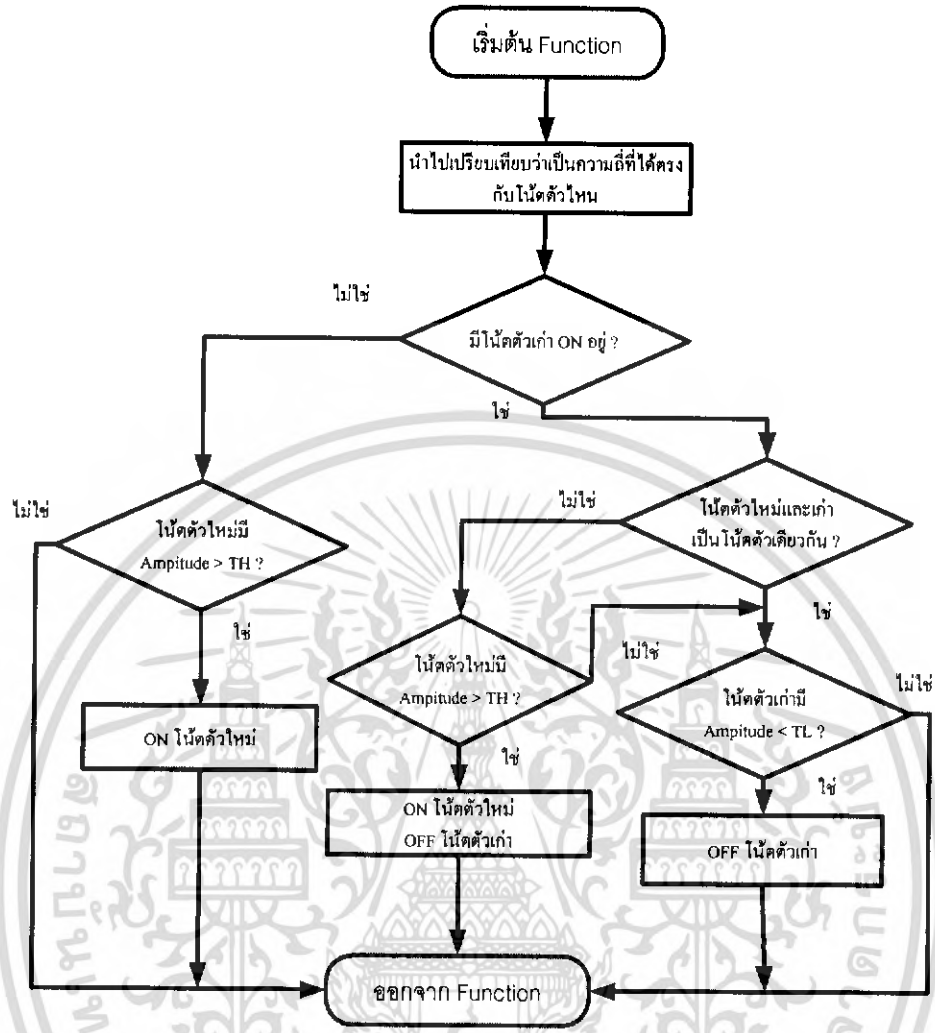
3.3.3 ฟังก์ชันฟาส์ทฟูเรียร์ทรานสฟอร์ม

เราจะแบ่งลูปการกระทำเป็น 3 ลูปที่ซ้อนกัน คือลูป I, J และ K โดย I จะเป็นจำนวนครั้งในการคูณแต่ละชุด โดย I จะมีค่าตั้งแต่ 0 เพิ่มขึ้นครั้งละ 1 จนถึง $\log_2 N$ เมื่อ N เป็นจำนวนข้อมูล และ J จะแทนจุดเริ่มต้นของแต่ละชุด โดยจะมีค่าตั้งแต่ 0 ไปจนถึงไม่เกิน N – 1 และเพิ่มขึ้นครั้งละ 2^I ส่วน K จะเป็นจำนวนครั้งของการคำนวนใน J หนึ่งครั้ง และค่า wp เป็นตัวระบุกำลังของ Twiddle Factor ที่ใช้ในแต่ละครั้ง ซึ่งจะมีค่าเพิ่มขึ้นครั้งละ (N / 2^I) % (N / 2) โดย % คือหารเอาแต่เศษ ดังรูปที่ 3.8



รูปที่ 3.8 Signal Flow Graph ของฟาส์ทฟูเรียร์ทรานสฟอร์มแบบลดทอนทางเวลา

โดยการคำนวณหนึ่งครั้งของ Fast Fourier Transform จะทำเป็นคู่กันสมมุติว่า A ทำการคำนวณกับ B Output ของ A = A + BW และ Output ของ B จะเท่ากับ A – BW



รูปที่ 3.9 โฟลวชาร์ตของฟังก์ชันการคัดเลือกตัวโน้ต

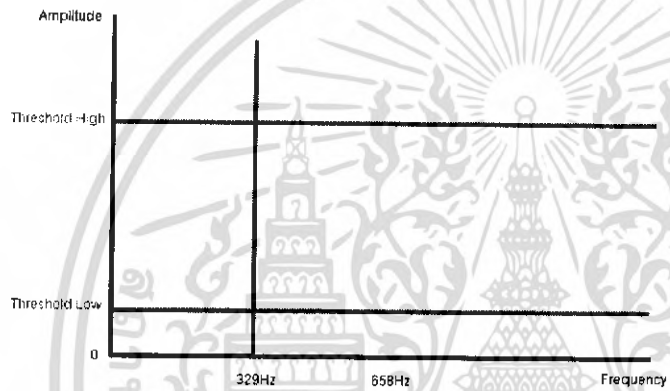
3.3.4 การคัดเลือกตัวโน้ต

เมื่อผ่านการคำนวณ Fast Fourier Transform มาแล้ว เราจะได้ค่าแอมปลิจูดของแต่ละความถี่มา ซึ่งเมื่อเทียบกับความถี่ของตัวโน้ตแล้ว จะมี Grid ที่มีค่าความถี่ใกล้เคียงกัน เราจะใช้แอมปลิจูดของ Grid ที่มีค่าใกล้เคียงแทนแอมปลิจูดของตัวโน้ตนั้นๆ เช่น โน้ต E3 มีความถี่ใกล้กับ Grid ที่ 16 เราก็จะใช้แอมปลิจูดที่ Grid 16 เป็นแอมปลิจูดของโน้ต E3 แต่บางช่วงอาจจะห่างจาก Grid ซึ่งเป็นค่าที่ไม่เท่ากัน ซึ่งแต่ละตัวโน้ตอาจมีค่าไม่เท่ากัน เลยได้ใช้หลักทางสถิติในการหาแอมปลิจูดที่แน่นอนของแต่ละความถี่ โดยการใช้การทดลองหาค่าที่สูงสุดของแต่ละตัวโน้ต นำเก็บไว้เป็นตัวหารของแต่ละ Grid ซึ่งเมื่อมีข้อมูลเข้ามา จะนำมาหารด้วยค่าสูงสุดของแต่ละตัวโน้ต เพื่อให้

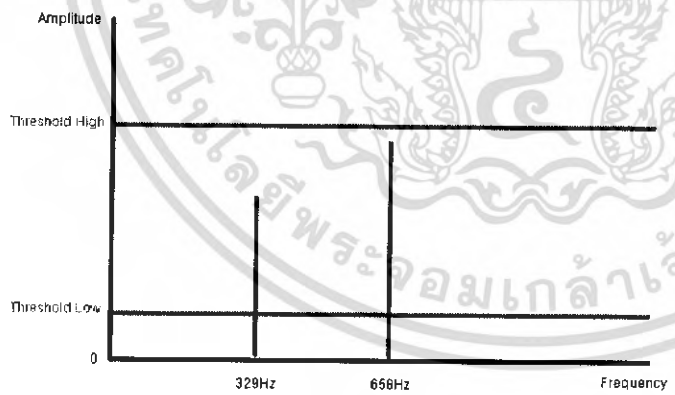
แอมปลิจูดของโน้ตทุกตัวอยู่ใน Scale เดียวกัน ซึ่งจะสามารถทำให้เราหาค่าโน้ตที่มีแอมปลิจูดสูงสุดได้

ในการเลือกตัวโน้ต เราจะเลือกจากความถี่ของตัวโน้ตที่มีแอมปลิจูดสูงที่สุด โดยเมื่อแอมปลิจูดของตัวโน้ตนั้นสูงเกินกว่า Threshold High ก็จะถือว่าโน้ตตัวนั้นอยู่ในสถานะ On แต่เนื่องจากแอมปลิจูดนั้นลดลงเร็วมาก จึงต้องตั้ง Threshold Low เอาไว้ให้ต่ำ และห่างจาก Threshold High เพื่อไม่ทำให้เสียงของโน้ตนั้นสั้นเกินไป จึงกำหนดให้เมื่อแอมปลิจูดของโน้ตตัวนั้น ลดต่ำลงจนต่ำกว่า Threshold Low แล้ว จะสั่งให้โน้ตตัวนั้นอยู่ในสถานะ Off

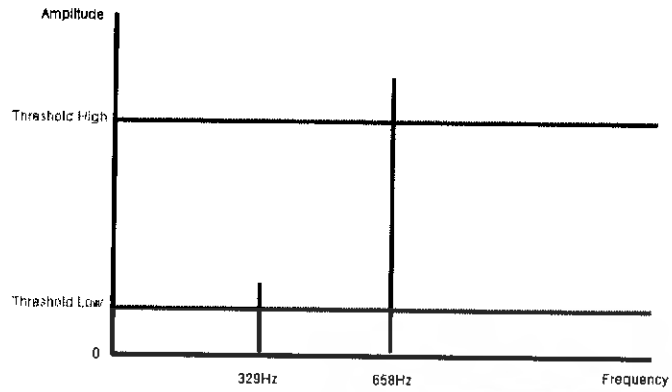
ในอีกกรณีหนึ่ง คือเมื่อมีโน้ตที่ On อยู่ แต่มีโน้ตตัวอื่นที่ Amplitude สูงสุดแทน ถ้าโน้ตตัวใหม่มีแอมปลิจูดสูงกว่า Threshold High จะสั่ง Off โน้ตตัวเก่าและ On โน้ตตัวใหม่แทนทันที



(ก)



(ข)



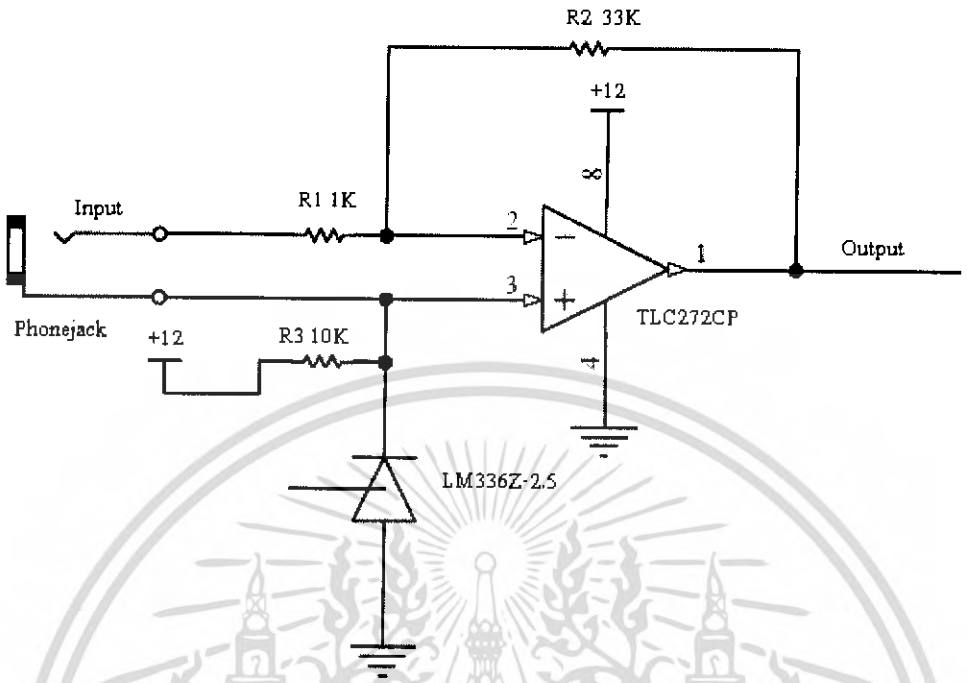
(ค)

รูปที่ 3.10 ตัวอย่างของการคัดเลือกตัวโน้ต

ดังตัวอย่างในรูปที่ 3.10ก ในรูปแรกเมื่อแอมปลิจูดของความถี่ 329Hz ซึ่งเป็นค่าที่สูงที่สุด มีค่ามากกว่า Threshold High เราจะสั่งให้ 329Hz อยู่ในสถานะ On ในรูปที่ 3.10ข จะเห็นว่าความถี่ 329Hz ไม่ได้เป็นความถี่ที่มีค่าแอมปลิจูดสูงที่สุดแล้ว แต่ค่าของความถี่ 658Hz ที่มีค่าสูงสุด ยังไม่เกิน Threshold High และความถี่ 329Hz ยังไม่ต่ำกว่า Threshold Low จะทำให้ยังไม่มีการเปลี่ยนแปลงเกินขึ้นซึ่งสถานะของ 329Hz จะยัง On อยู่ แต่เมื่อแอมปลิจูด ของ 658Hz เป็นความถี่ที่มีแอมปลิจูดสูงสุด และ มีค่ามากกว่า Threshold High ดังรูปที่ 3.10ค จะทำการ Off ความถี่ 329Hz ทันที และสั่งให้ 658Hz On

3.4 วงจรขยาย กลับเฟสแบบป้อนกลับทางลบ

ในส่วนนี้จะทำหน้าที่ขยายสัญญาณอินพุทให้มีขนาดของแอมปลิจูดที่เหมาะสมโดยใช้ไฟเลี้ยงแบบ Single Supply (0-12 V) และทำการยกระดับสัญญาณเปรียบเทียบเป็น 2.5 V โดยจะมีGain 33 เท่า ดังรูปที่ 3.11



รูปที่ 3.11 วงจรขยายกลับเฟสแบบป้อนกลับทางลบ

3.5 ลูกบิด และสวิตช์ปรับค่า ที่สามารถควบคุมตัวแปรทางมิดี้

3.5.1ลูกบิดปรับค่า

โดยลูกบิดจะทำการปรับค่าตัวแปรต่างๆของมิดี้ในส่วนของ Control Changeที่เป็น 7 Bit Controllers โดยจะมี VR เพื่อปรับค่าแรงดันแล้วส่งค่าไปยังโมดูลแปลงสัญญาณอนาลอกเป็นดิจิตอลตรวจสอบค่าแรงดันขนาด 10 บิต แล้วประมวลผลออกมาเป็นสัญญาณดิจิตอลขนาด 7 บิต (0-127) โดยการหารด้วย 8 และแปลงให้อยู่ในรูปแบบมิดี้โค้ด โดยค่าที่ได้จากการปรับ VR จะถูกนำมาใช้เป็นค่าตัวแปร ใน Byte ที่ 3 ของข้อความมิดี้ โดยเราจะสามารถกำหนดค่า Controller Number (ข้อมูลใน Byte ที่ 2) ให้กับลูกบิดต่างๆได้ โดยจะมีตัวแปรที่ใช้และโค้ดดังนี้

ตารางที่ 3.2 ตัวแปรที่สามารถปรับค่าได้จากลูกบิด

Controller Number	Function	Abbreviation
46	Sound variation	SVA

47	Timbre/harmonic content	THC
48	Release time	RET
49	Attack time	ATT
4A	Brightness	BRN
5B	External Effects Depth	EED
5C	Tremolo Depth	TRD
5D	Chorus Depth	CHD
5E	Celeste (Detune) Depth	CED
5F	Phaser Depth	PHD

3.5.2 สวิตช์

โดยสวิตช์จะทำหน้าที่ส่ง มิดี้โค้ด ในส่วนของ Control Change ที่เป็น 7 Bit Swithces โดยเราจะสามารถกำหนดค่า Controller Number (ข้อมูลใน Byte ที่ 2) ซึ่งจะมีแค่ 2 สถานะคือ On และ Off ซึ่งจะต้องกำหนดค่า 0x00 เป็น Off และ 0x7F เป็น On โดยจะมีตัวแปรที่ใช้และโค้ดดังนี้

ตารางที่ 3.3 ตัวแปรที่สามารถปรับค่าได้จากสวิตช์

Controller Number	Function
40	Sustain pedal
41	Portamento on/off
42	Sostenuto pedal
43	Soft pedal
44	Legato footswitch
45	Hold 2

บทที่ 4

การทดลอง

การทดลองที่ 1 การหาค่าและเปรียบเทียบแอมปลิจูดองค์ประกอบทางความถี่

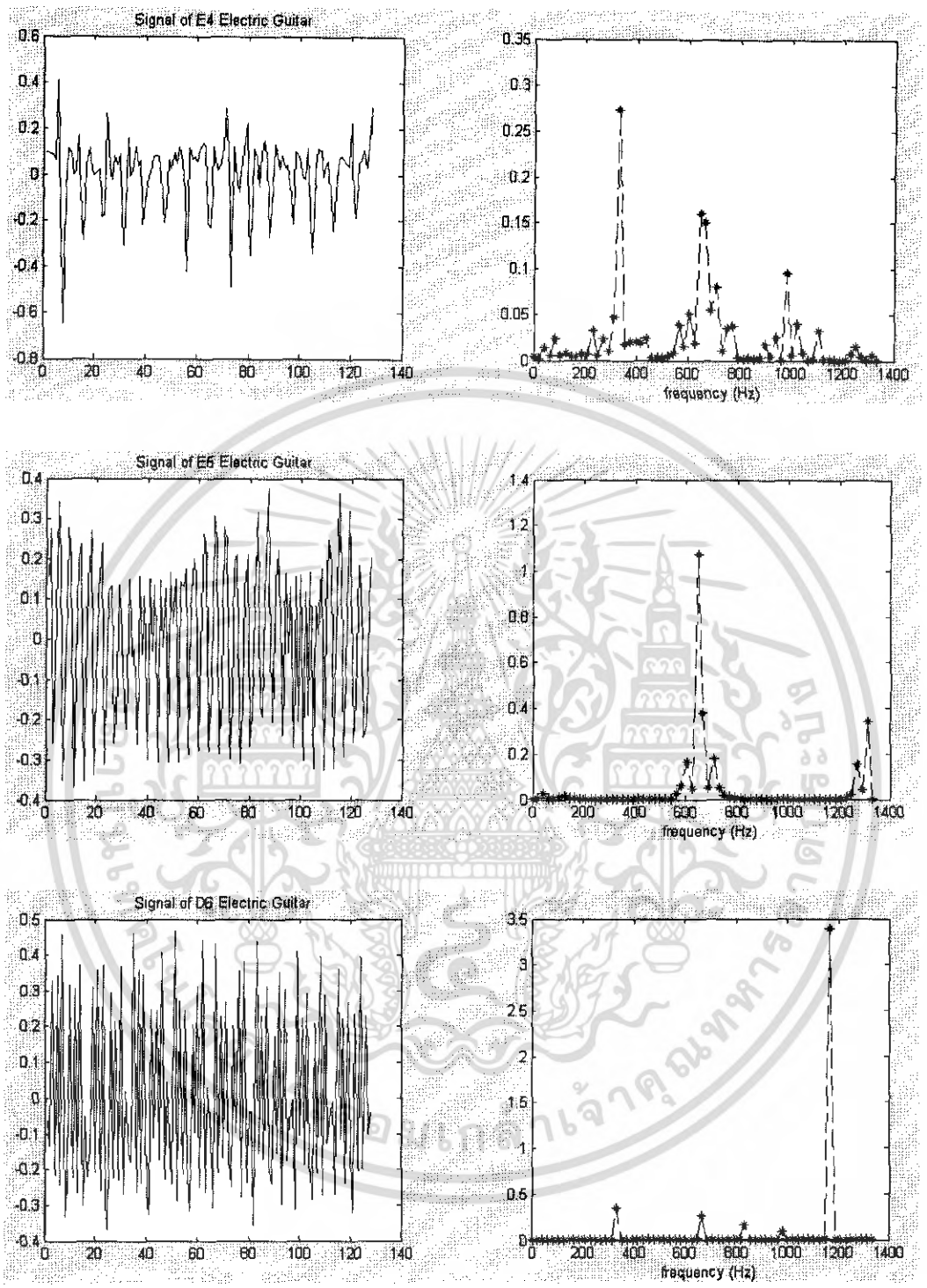
เราใช้เสียงกีต้าร์ในการทดลอง โดยทดลองดีดตั้งแต่ตัวโน้ต E3 ~ D5 คือสายกีต้าร์สาย 1 เนื่องจากสายที่เฟรท 0 ถึงเฟรท 22 ที่แอมปลิจูดเดียวกัน โดยใช้ Fast Fourier Transform Function ของ MATLAB ในการหาค่าแอมปลิจูดของแต่ละค่าความถี่ โดยเก็บค่า Sampling 128 ค่าแรกที่ความถี่การแซมปลิ้ง 2666.66Hz ซึ่งเป็นช่วงเวลา 48ms แรกหลังจากที่เริ่มมีเสียงออกจากสายกีต้าร์ โดยใช้ Source code

```
tt = 1
for ii=1:tt
    t = 0:0.000375:0.6
    kk = 4608*(ii-1)
    f = 1+kk
    t = 4608+kk
    y = wavread('e4',[f,t]);
    for i=1:128
        x(i) = y(36*i);
    end

    YY = fft(x,128);
    Pyy = YY.* conj(YY) / 128;
    f = 2666.666*(0:64)/128;
    figure(1);
    subplot(tt,1,ii);
    plot(f,Pyy(1:65),'b--*')
    title('Frequency content of signal')
    xlabel('frequency (Hz)')
end
```

ตารางที่ 4.1 ตารางการหาค่าแอมปลิจูดองค์ประกอบทางความถี่ของสัญญาณ

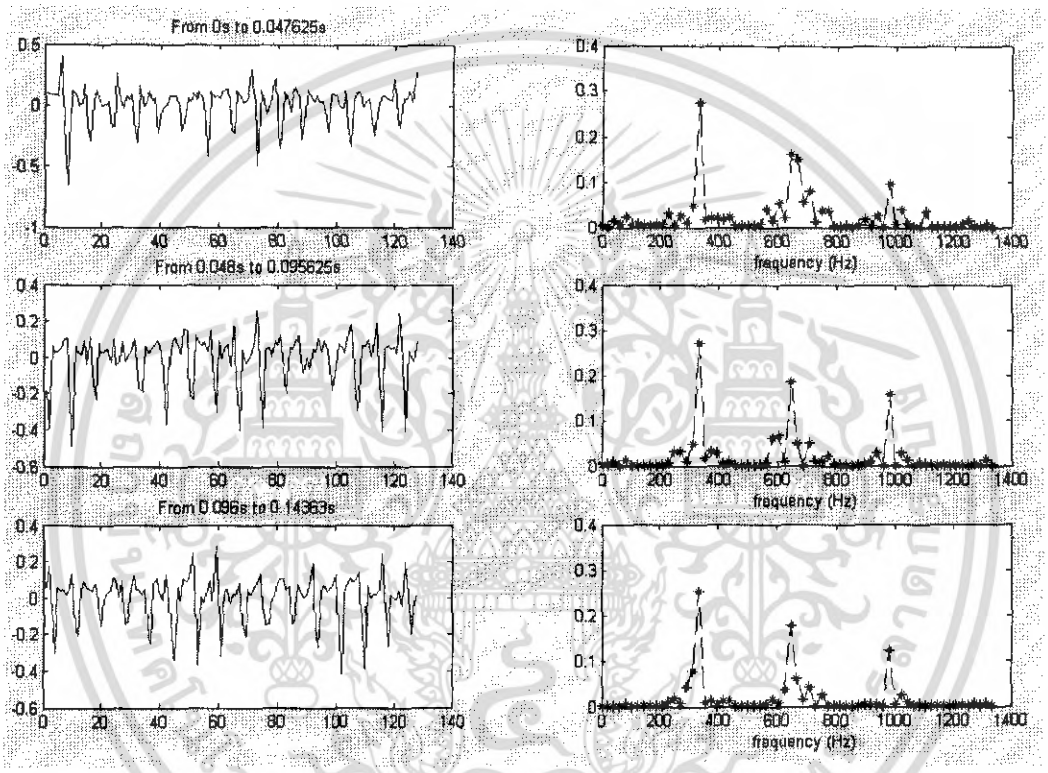
โน้ต	ความถี่ของตัวโน้ต (Hz)	ค่าความถี่ที่วัดได้ (Hz)	ความถี่ขอบล่างของช่อง (Hz)	แอมปลิจูดของขอบล่าง	ความถี่ขอบบนของช่อง (Hz)	แอมปลิจูดของขอบล่าง
E4	329.6275	327.45	312.50(15)	0.047191	333.33(16)	0.27344
F4	349.2282	346.78	333.33(16)	0.088144	354.17(17)	0.33045
F#4	369.9944	367.32	354.17(17)	0.23236	375.00(18)	0.72679
G4	391.9954	389.49	375.00(18)	0.13675	395.83(19)	0.71287
G#4	415.3046	412.32	395.83(19)	0.056515	416.67(20)	0.810290
A4	440.0000	437.26	437.50(21)	0.987790	458.33(22)	0.002112
A#4	466.1637	462.82	458.33(22)	0.927320	479.17(23)	0.083373
B4	493.8833	489.85	479.17(23)	0.390340	500.00(24)	0.440230
C5	523.2511	518.24	520.83(25)	2.766500	541.67(26)	0.043639
C#5	554.3652	549.45	541.67(26)	1.048300	562.50(27)	0.384660
D5	587.3295	582.69	583.33(28)	2.038400	604.17(29)	0.005846
D#5	622.2539	617.20	604.17(29)	0.510560	625.00(30)	1.398300
E5	659.2551	653.90	645.83(31)	1.073900	666.67(32)	0.378930
F5	698.4564	693.41	687.50(33)	2.096900	708.33(34)	0.351290
F#5	739.9888	735.00	729.17(35)	2.579200	750.00(36)	0.350460
G5	783.9908	777.95	770.83(37)	1.946400	791.67(38)	0.506310
G#5	830.6093	822.90	812.50(39)	1.649700	833.33(40)	1.646200
A5	880.0000	873.71	875.00(42)	2.221200	895.83(43)	0.018325
A#5	932.3275	924.25	916.67(44)	2.073000	937.50(45)	0.661660
B5	987.7666	980.81	979.17(47)	3.0063	1000.00(48)	0.023076
C6	1046.5022	1037.61	1041.70(50)	3.3417	1062.50(51)	0.099007
C#6	1108.7305	1098.61	1104.20(53)	3.6099	1125.00(54)	0.19215
D6	1174.6590	1165.75	1166.70(56)	3.3958	1187.50(57)	0.010436



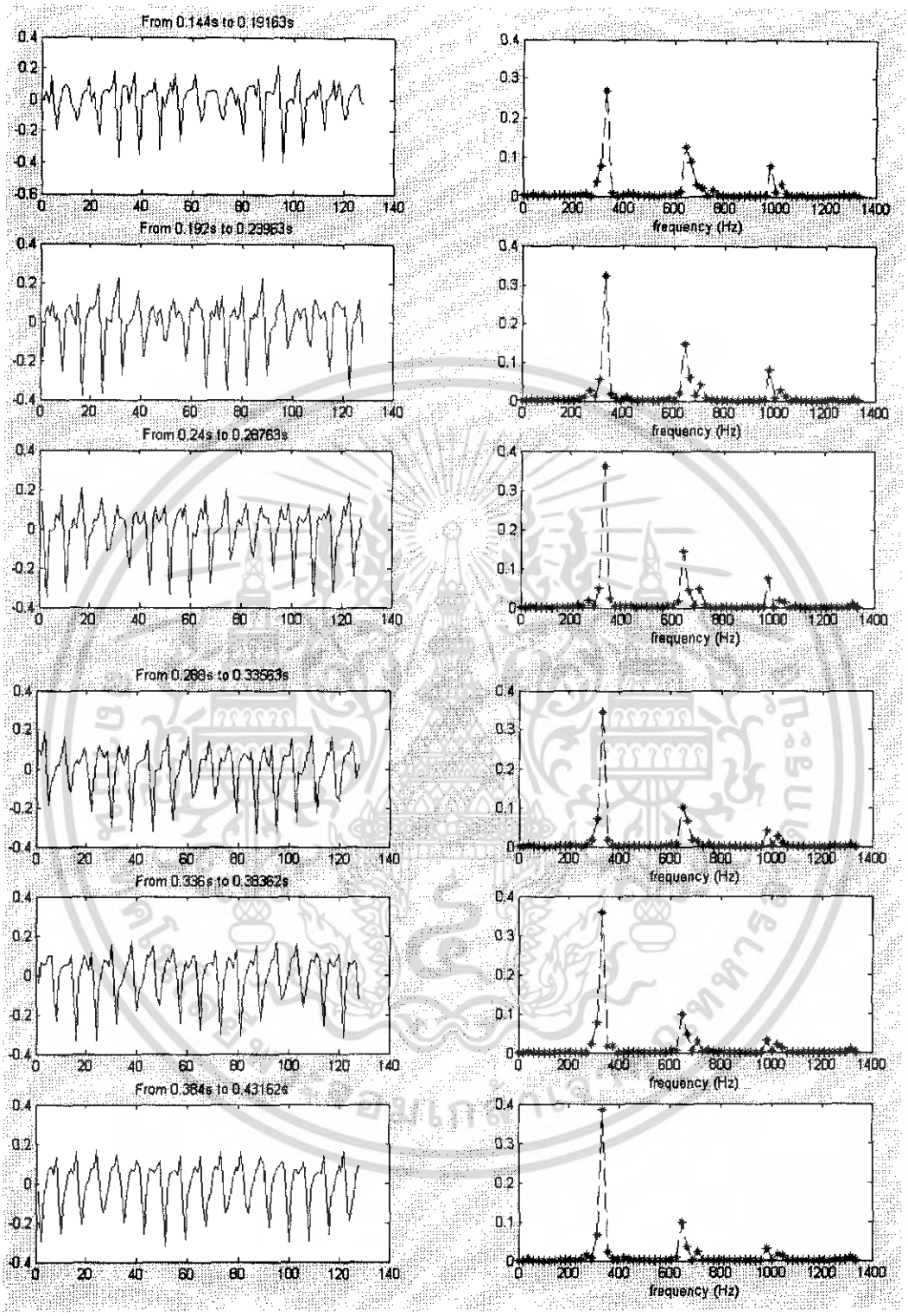
รูปที่ 4.1 กราฟเปรียบเทียบรูปสัญญาณและค่าแอมปลิจูดของแต่ละความถี่ ที่ตัวโน้ต E3, E4 และ D5 ของกีต้าร์ไฟฟ้า

การทดลองที่ 2 แอมปลิจูดขององค์ประกอบทางความถี่ของสัญญาณที่เวลาต่างๆ

เป็นการทดลองเปรียบเทียบ Fundamental Frequency กับ Harmonic Frequency เมื่อเวลาผ่านไป ตั้งแต่ 0-0.43162 วินาที



รูปที่ 4.2 กราฟความถี่ที่เวลาต่างๆในการแซมปลิ้งในช่วง 0 - 0.43162 วินาที



รูปที่ 4.2 (ต่อ) กราฟความถี่ที่เวลาต่างๆในการแซมปลิ้งในช่วง 0 - 0.43162 วินาที

การทดลองที่ 3 การหาค่าแอมปลิจูดสูงสุดในแต่ละ Grid

ในการทดลองนี้จะทำการเก็บข้อมูลของแอมปลิจูดสูงสุดในแต่ละ Grid เพื่อนำมาใช้ในการปรับค่าแอมปลิจูดของสัญญาณในแต่ละ Grid ให้อยู่ในเสกลเดียวกันด้วยการนำแอมปลิจูดของสัญญาณที่ได้มาหารด้วยแอมปลิจูดสูงสุดในแต่ละ Grid ทุกๆ Grid โดยในการเก็บค่าแอมปลิจูดสูงสุดจะทำการเก็บค่า 4 ครั้งแล้วจึงนำมาหาค่าเฉลี่ยจะได้ผลดังตารางที่ 4.2

ตารางที่ 4.2 ค่าแอมปลิจูดเฉลี่ยสูงสุดของ Grid ที่ตรงกับตัวโน้ต และ Grid ใกล้เคียง

Note	Number	Grid #1	Grid #2	Grid #3	Grid #4
		14	15	16	17
E	1	1319	1526	7113	1458
	2	1649	1408	5272	1323
	3	1838	1568	8355	1941
	4	1895	1609	7367	1786
	Average	1675.25	1527.75	7026.75	1627
		15	16	17	18
F	1	1060	2379	4993	2731
	2	1083	2003	5089	2561
	3	1253	2435	7475	2160
	4	1280	2711	6113	2859
	Average	1169	2382	5917.5	2577.75
		16	17	18	19
F#	1	2033	1505	6238	5771
	2	1522	2190	4440	3487
	3	1943	2568	5726	4974
	4	1895	3086	4155	7786
	Average	1848.25	2337.25	5139.75	5504.5
		17	18	19	20
G	1	2292	4355	8611	2266
	2	2235	2876	8137	2665

	3	1664	4904	8762	1724
	4	1756	6407	8323	1369
	Average	1986.75	4635.5	8458.25	2006
		18	19	20	21
G#	1	1857	2565	7059	3819
	2	2193	6028	5993	2886
	3	2092	3442	7739	3104
	4	1873	4110	5291	2228
	Average	2003.75	4036.25	6520.5	3009.25
		20	21	22	23
A	1	3357	28179	8876	2867
	2	2689	26899	7393	2664
	3	3834	28057	6046	3713
	4	3992	25986	6913	2089
	Average	3468	27280.25	7307	2833.25
		21	22	23	24
A#	1	9086	14769	4540	2442
	2	9388	12891	4720	1899
	3	10018	12396	3775	1941
	4	8822	11769	3125	1927
	Average	9328.5	12956.25	4040	2052.25
		22	23	24	25
B	1	3554	4602	6021	3495
	2	2369	4521	6256	3616
	3	2896	5349	4959	3397
	4	2645	4521	5341	3275
	Average	2866	4748.25	5644.25	3445.75
		24	25	26	27
C	1	3943	10186	4786	2481

	2	4564	12393	5107	3388
	3	2727	12291	2806	2281
	4	4447	13177	4312	1911
	Average	3920.25	12011.75	4252.75	2515.25
		25	26	27	28
C#	1	3158	14540	7259	2796
	2	2768	11512	9369	3106
	3	2214	10530	9527	3183
	4	3917	13293	7330	2346
	Average	3014.25	12468.75	8371.25	2857.75
		27	28	29	30
D	1	4943	17125	7700	1746
	2	5370	16567	7817	3013
	3	5160	18585	6364	1777
	4	6187	19113	6036	2250
	Average	5415	17847.5	6979.25	2196.5
		28	29	30	31
D#	1	3254	5817	18973	7412
	2	3609	6655	22691	6656
	3	2262	7572	20740	6071
	4	2725	8430	19529	8900
	Average	2962.5	7118.5	20483.25	7259.75
		30	31	32	33
E	1	3326	17603	45676	12204
	2	4534	14714	43179	13420
	3	4897	20284	44012	12493
	4	5778	14004	45771	10038
	Average	4633.75	16651.25	44659.5	12038.75
		32	33	34	35

F	1	6771	31251	20064	2967
	2	9799	31261	18890	5162
	3	11148	28715	18076	2903
	4	12869	28884	18699	2483
	Average	10146.75	30027.75	18932.25	3378.75
		34	35	36	37
F#	1	5516	16740	14786	3466
	2	4502	17205	15464	2937
	3	5593	18416	15352	2911
	4	4196	17289	12915	3440
	Average	4951.75	17412.5	14629.25	3188.5
		36	37	38	39
G	1	2847	8875	19316	4753
	2	2922	9225	20910	4384
	3	4256	10277	19324	3506
	4	2587	7772	18956	5016
	Average	3153	9037.25	19626.5	4414.75
		38	39	40	41
G#	1	4080	8345	12316	6223
	2	4133	6335	12306	6023
	3	3919	4808	11982	5279
	4	3801	5334	13238	6617
	Average	3983.25	6205.5	12460.5	6035.5
		41	42	43	44
A	1	7458	31251	23129	7300
	2	6309	33471	28267	8512
	3	6194	36123	20817	8131
	4	8049	34528	25107	7882
	Average	7002.5	33843.25	24330	7956.25

		43	44	45	46
A#	1	4076	8784	18001	5419
	2	7706	9058	17081	5688
	3	7926	10326	19904	5668
	4	7463	10130	15936	5110
	Average	6792.75	9574.5	17730.5	5471.25
		46	47	48	49
B	1	3935	10836	9505	4416
	2	3635	10627	7851	5227
	3	4536	11610	8586	4214
	4	4158	12418	6243	5111
	Average	4066	11372.75	8046.25	4742
		49	50	51	52
C	1	4392	18219	4630	3662
	2	4919	15705	7115	3538
	3	5544	17449	5623	3279
	4	4848	9138	5257	3823
	Average	4925.75	15127.75	5656.25	3575.5
		52	53	54	55
C#	1	3939	16386	7983	1689
	2	3167	15909	7414	3469
	3	3251	13124	10850	2687
	4	2589	15097	6995	2918
	Average	3236.5	15129	8310.5	2690.75
		55	56	57	58
D	1	3245	7189	10719	2465
	2	3739	9208	11675	2594
	3	2871	8631	12363	3841
	4	5726	9252	12674	3006

	Average	3895.25	8570	11857.75	2976.5

การทดลองที่ 4 การหาค่าเฉลี่ยแอมปลิจูดของทุก Grid เพื่อใช้กำหนดค่า threshold

ในการทดลองนี้จะทำการเก็บค่าแอมปลิจูดของสัญญาณที่มีค่าน้อยที่สุด 10 ครั้งในแต่ละ Grid ทุกๆ Grid มาหาค่าเฉลี่ยจะได้ค่าดังตารางที่ 4.2 แล้วนำค่าเฉลี่ยของแอมปริจูดในแต่ละ Grid มาหารค่าแอมปลิจูดสูงสุดที่ได้จากการทดลองที่ 3 เพื่อปรับให้ แอมปลิจูดน้อยที่สุดเฉลี่ยในแต่ละ Grid อยู่บนเสกลเดียวกันก็จะได้ค่า Threshold ของแต่ละ Grid แล้วจึงนำ Threshold ที่ได้มาหาค่าเฉลี่ยก็จะได้ Threshold high กลางที่ใช้เป็น Threshold ในการสั่ง On โน้ต ดังตารางที่ 4.3

ตารางที่ 4.2 ค่าเฉลี่ยแอมปลิจูดของสัญญาณที่มีค่าน้อยที่สุด

Note(Grid)	1	2	3	4	5	6	7	8	9	10	**Average**
E(16)	1422	779	769	1019	932	1659	1032	953	1016	841	**1042.2**
F(17)	1091	2958	906	711	769	903	1282	1974	1970	1660	**1422.4**
F#(18)	1131	1257	576	1011	418	428	1069	697	1464	1499	**955**
G(19)	984	1283	1937	1058	997	649	892	1702	969	911	**1138.2**
G#(20)	1119	682	1597	933	1941	1207	2198	1550	925	1097	**1324.9**
A(21)	2012	1184	2981	6231	4954	859	2403	3081	4995	1330	**3003**
A#(22)	1181	2414	1100	1171	1481	1584	931	1362	2274	1375	**1487.3**
B(24)	1670	2244	756	693	1007	762	1263	1533	625	907	**1146**
C(25)	1325	1423	1263	1272	2717	2225	3342	1776	2413	1741	**1949.7**
C#(26)	2383	1814	1031	869	1922	2168	1111	1549	1828	1863	**1653.8**
D(28)	4689	1378	1876	1123	1936	2633	1516	2203	1479	2836	**2166.9**
D#(30)	4873	3415	2506	3283	3415	3065	2298	2287	1571	4524	**3123.7**
E(32)	2619	6288	4870	5665	3441	2431	2388	2669	6235	4337	**4094.3**
F(33)	4940	3078	6793	4234	4552	3322	6643	7426	3319	2189	**4649.6**
F#(35)	1978	3887	4090	1630	3091	4530	3474	2292	1325	3089	**2938.6**
G(38)	5330	2391	1329	2039	1329	1475	1416	1423	1785	2304	**2082.1**

G#(40)	3725	3641	3960	1344	1977	2248	2705	3825	1236	4585	**2924.6**
A(42)	6491	13759	6696	6802	6676	4999	9230	6633	2214	8181	**7168.1**
A#(45)	3842	2819	4478	3654	3129	2178	5754	3345	2671	2705	**3457.5**
B(47)	1894	2754	1735	1295	3405	1076	1007	2227	1308	1053	**1775.4**
C(50)	6243	3091	2635	9846	1086	2686	1482	4885	4873	6519	**4334.6**
C#(53)	4878	2195	2060	4658	5536	2634	1421	3024	954	2058	**2941.8**
D(57)	2682	2925	1058	7545	2966	3199	3558	1530	2068	2898	**3042.9**

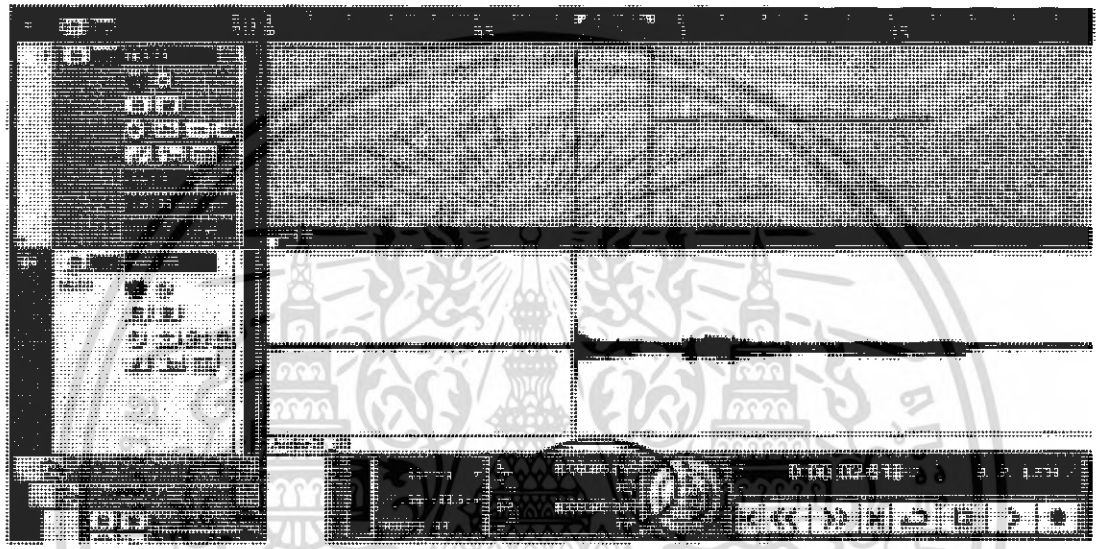
ตารางที่ 4.3 การหาThreshold high

Note(Grid)	Min	Peak	Min/Peak
E(16)	1042.2	7026.75	**0.148319**
F(17)	1422.4	5917.5	**0.240372**
F#(18)	955	5139.75	**0.185807**
G(19)	1138.2	8458.25	**0.134567**
G#(20)	1324.9	6520.5	**0.20319**
A(21)	3003	27280.25	**0.11008**
A#(22)	1487.3	12956.25	**0.114794**
B(24)	1146	5644.25	**0.203038**
C(25)	1949.7	12011.75	**0.162316**
C#(26)	1653.8	12468.75	**0.132636**
D(28)	2166.9	17847.5	**0.121412**
D#(30)	3123.7	20483.25	**0.1525**
E(32)	4094.3	44659.5	**0.091678**
F(33)	4649.6	30027.75	**0.154843**
F#(35)	2938.6	17412.5	**0.168764**
G(38)	2082.1	19626.5	**0.106086**

G#(40)	2924.6	12460.5	**0.23471**
A(42)	7168.1	33843.25	**0.211803**
A#(45)	3457.5	17730.5	**0.195003**
B(47)	1775.4	11372.75	**0.15611**
C(50)	4334.6	15127.75	**0.286533**
C#(53)	2941.8	15129	**0.194448**
D(57)	3042.9	11857.75	**0.256617**

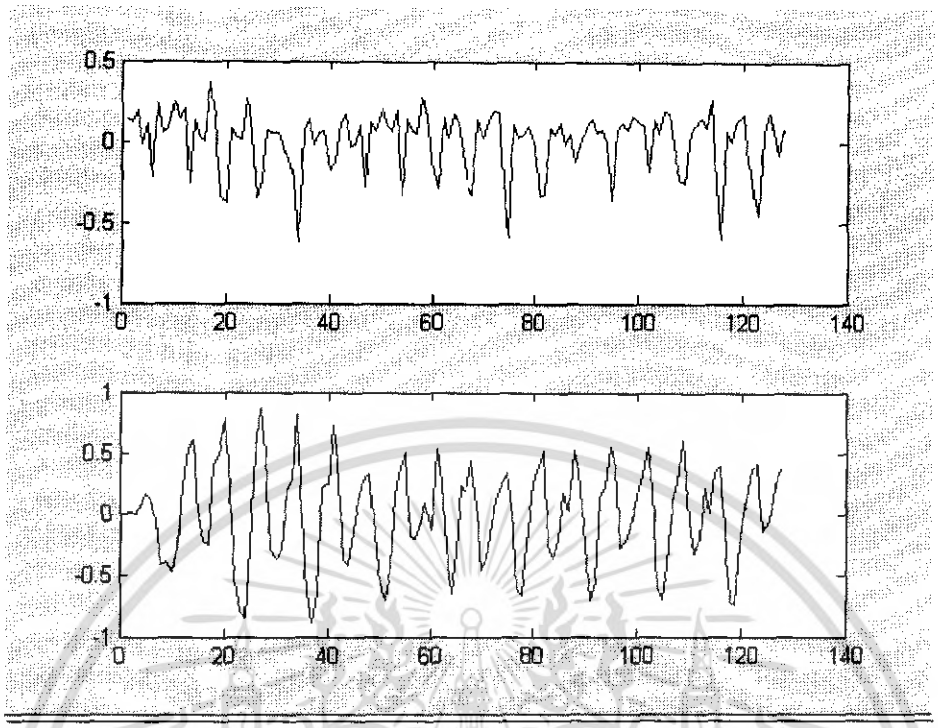
การทดลองที่ 5 การหาค่า Delay Time และเปรียบเทียบสัญญาณ Input กับ Output

ทำการป้อนสัญญาณ Input ด้วยการเล่นกีต้าร์ไฟฟ้าตั้งแต่โน้ต E4 – D5 เข้าสู่วงจรแล้วป้อน Output ในรูปแบบมิดี้ไปบันทึกเป็นรูปแบบมิดี้และบันทึกสัญญาณอนาลอกในรูปแบบเวฟไฟล์ ไปพร้อมๆกันในซอฟท์แวร์บันทึกเสียงโดยในโปรแกรมที่ใช้จะมีเสกลเวลาบอกทำให้สามารถหาค่า Delay Time ระหว่างสัญญาณอนาลอก กับ ค่ามิดี้ที่บันทึกได้ดังรูปที่ 4.3
โดยจะมีค่า Delay Time ประมาณ 0.929 – 0.750 = 0.179 วินาที

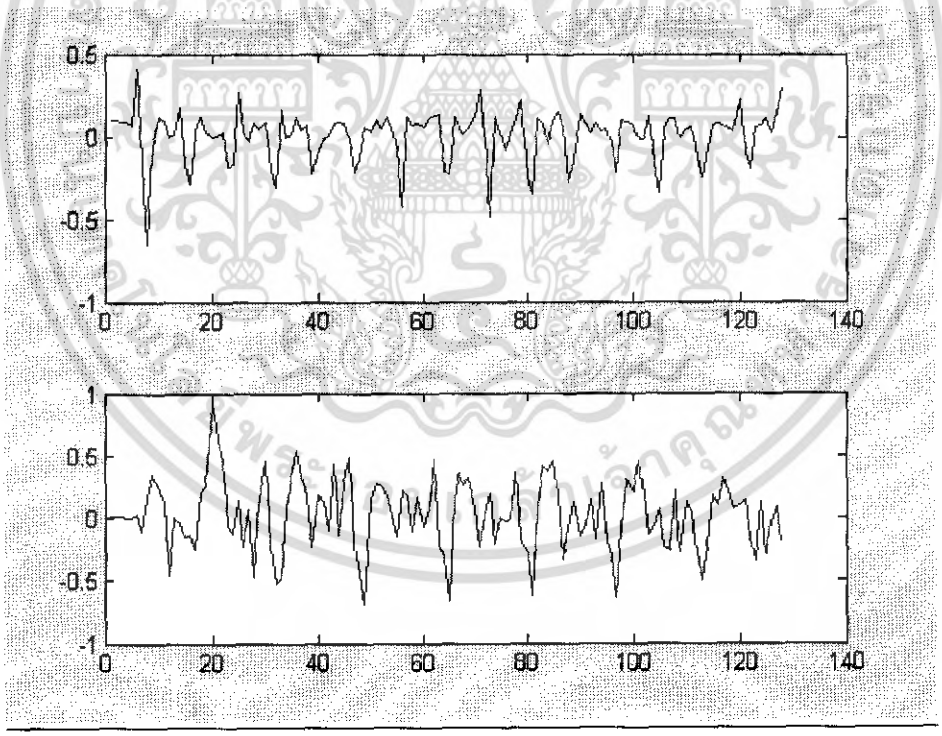


รูปที่ 4.3 แสดงระยะห่างของเวลาในการบันทึกสัญญาณอนาลอกกับการบันทึกมิดี้

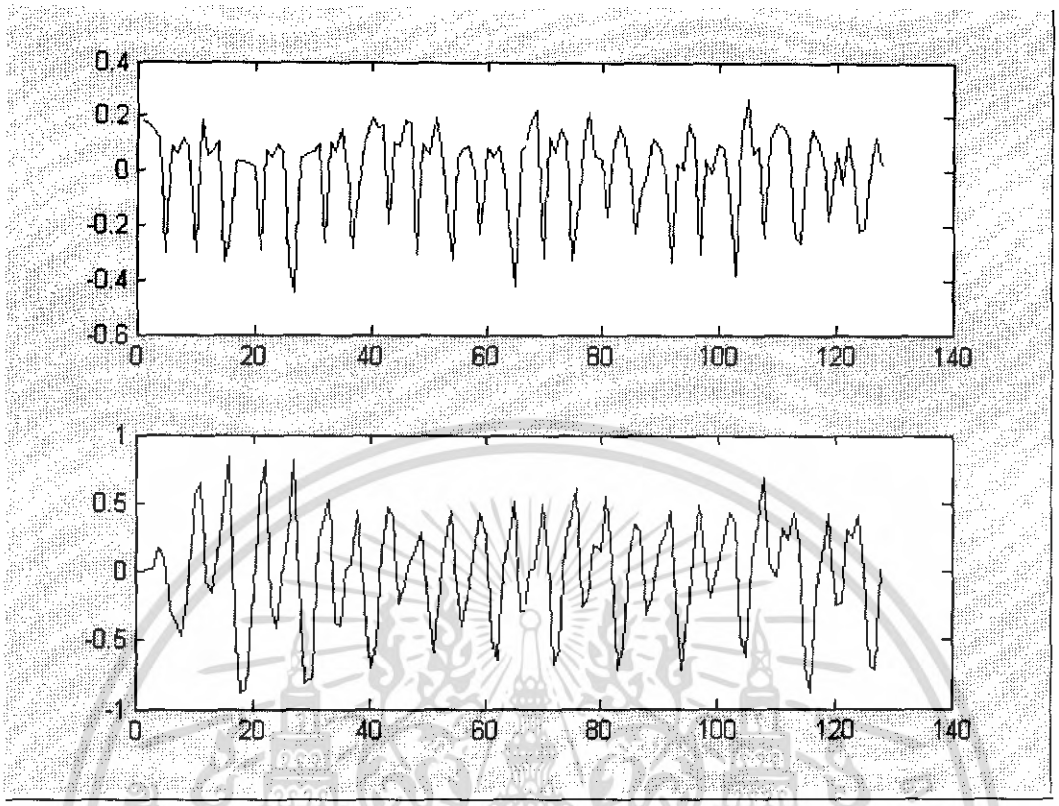
จากนั้นนำโน้ตมิดี้ที่ได้มาสังเคราะห์เสียงด้วยซินธีไซเซอร์ซอฟท์แวร์ออกมาเป็นสัญญาณอนาลอกแล้วทำการบันทึกเป็นรูปแบบเวฟไฟล์แล้วนำรูปสัญญาณที่ได้ไปเปรียบเทียบกับสัญญาณอนาลอกเดิมจากกีตาร์ไฟฟ้า ปรากฏว่ามีความถี่เท่ากันดังแสดงในรูปที่ 4.4



(ก)



(ข)



(ค)

รูปที่ 4.4 เปรียบเทียบรูปสัญญาณอนาลอกที่ได้จากโน้ตมิดี้ที่สังเคราะห์เสียงด้วยซินธีไซเซอร์ซอฟท์แวร์กับรูปสัญญาณอนาลอกเดิมจากกีต้าร์ไฟฟ้า (ก) โน้ต E ความถี่ 329.627Hz (ข) โน้ต G ความถี่ 391.995Hz (ค) โน้ต B ความถี่ 493.883Hz

บทที่ 5
สรุปผล

จากการทดลองที่ 1 การหาค่าแอมปลิจูดองค์ประกอบทางความถี่ของสัญญาณ ในการใช้ Fast Fourier Transform Function ของ MATLAB ในการคำนวน โดยใช้จำนวนครั้งการสุ่มค่า 128 ครั้งได้แอมปลิจูดของความถี่ห่างกัน 20.83 Hz ซึ่งสามารถใช้ระบุแยกความแตกต่างระดับเสียงของตัวโน้ต E3 ~ D5 ในสายกีต้าร์สาย 1 ได้ (ความถี่ของระดับเสียงอยู่ต่างช่วงความถี่ที่แบ่ง) แต่จะมีความผิดเพี้ยนในเรื่องของแอมปลิจูดของโน้ต ในช่วงความต่ำ เพราะระดับเสียงของโน้ตจะมีความต่างของความถี่น้อย โน้ตบางตัวจะอยู่หางจากจุดความถี่ที่ได้จากการคำนวณทำให้ไม่ได้แอมปลิจูดที่แท้จริง แต่ในช่วงความถี่สูงจะมีความถูกต้องมากขึ้นเพราะระยะห่างความถี่ระหว่างโน้ตแต่ละตัวมีความห่างกันมากขึ้นทำให้แอมปลิจูของแต่ละความถี่ที่ได้จากการคำนวณอยู่ตรงกับความถี่ระดับเสียงมากขึ้น

จากผลการทดลองที่ 2 แอมปลิจูดขององค์ประกอบทางความถี่ของสัญญาณที่เวลาต่างๆในการแซมปลิ้ง ในการเลือกช่วงเวลา แซมปลิ้ง ที่แตกต่างกันนั้นปรากฏว่าเมื่อเวลาผ่านไปคลื่นที่เป็นความถี่ฮาร์โมนิกส์จะมีพลังงานลดลงเรื่อยๆ แต่ในขณะที่คลื่นที่เป็นความถี่มูลฐานจะมีพลังงานค่อนข้างคงที่ นั้นคือ หากมีการเลือกช่วงเวลาในการแซมปลิ้งแล้วการเลือกช่วงเวลาที่ผ่านมาแล้วเล็กน้อยก็จะสามารถแก้ปัญหาความผิดพลาดในสัญญาณที่คลื่นที่เป็นความถี่ฮาร์โมนิกส์มีพลังงานมีพลังงานสูงกว่าคลื่นที่เป็นความถี่มูลฐาน

จากการทดลองที่ 3 จะได้ค่า Amplitude Scale ของแต่ละตัวโน้ตที่เท่ากันทำให้สามารถแยกแยะโน้ตที่มีแอมปลิจูดสูงสุดได้โดยง่าย

จากการทดลองที่ 4 นำค่าเฉลี่ย Amplitude ของสัญญาณในแต่ละโน้ตที่ได้จากการดีดน้ำหนักธรรมดามาหาร Max Amplitude จะได้ค่า Threshold High ของแต่ละตัวโน้ต

จากการทดลองที่ 5 ใช้เวลาในการประมวลผลทั้งหมดประมาณ 179 ms ซึ่งเกิดจากการ Sampling ด้วยเวลา 0.375 x 128 = 48 ms และการส่งสัญญาณมิดี้อีกประมาณ 1 ms นอกจากนั้นจะเป็นเวลาในการคำนวณฟังก์ชั้นฟาส์ทฟูเรียร์ทรานสฟอร์ม

บรรณานุกรม

1. ศาสตราจารย์ ดร.วิวัฒน์ กิรานนท์, "วิศวกรรมการสื่อสาร", สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, พิมพ์ครั้งที่ 4 พ.ศ.2540
2. ศาสตราจารย์ ดร.วัลลภ สุระกำพลธร, "การประมวลผลสัญญาณเชิงตัวเลข", สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,
3. Francis Rumsey, "MIDI Systems and Control", Butterworth-Heinemann Ltd, Second Edition, 1994
4. Michael Talbot-Smith, "Audio Engineer's Reference Book", Focal Press, Second Edition, 1999
5. dsPIC30F4011,4012 Data Sheet

ภาคผนวก

โปรแกรมควบคุมการทำงาน

```
#include <p30f4011.h>
#include <ports.h>
#include <lcd.h>
#include <uart.h>
#include <adc10.h>
#include <timer.h>
#include <stdio.h>

//Constant Declare
#define upmsg1 "Main Page"
#define upmsg2 "MENU"
#define upmsg3 "Channel"
#define upmsg4 "Instrument"
#define upmsg5 "Knob1"
#define upmsg6 "Knob2"
#define upmsg7 "Knob3"
#define upmsg8 "Switch Pedal"

#define dnmsg1 "          Channel"
#define dnmsg2 "       Instrument"
#define dnmsg3 "            Knob1"
#define dnmsg4 "            Knob2"
#define dnmsg5 "            Knob3"
#define dnmsg6 "     Switch Pedal"

#define knmsg0 "         Disabled"
#define knmsg1 "  Sound Variation"
#define knmsg2 "Tim/Harm Content"
#define knmsg3 "     Release Time"
#define knmsg4 "      Attack Time"
#define knmsg5 "       Brightness"
#define knmsg6 "Ext Effect Depth"
#define knmsg7 "    Tremolo Depth"
#define knmsg8 "     Chorus Depth"
#define knmsg9 "    Celeste Depth"
#define knmsgA "     Phaser Depth"

#define pdmsg1 "    Sustain Pedal"
#define pdmsg2 "Portamento Pedal"
#define pdmsg3 "  Sostenuto Pedal"
#define pdmsg4 "       Soft Pedal"
#define pdmsg5 " Legato Ftswitch"
#define pdmsg6 "            Hold 2"

#define knabb0 "DIS"
#define knabb1 "SVA"
#define knabb2 "THC"
#define knabb3 "RET"
#define knabb4 "ATT"
#define knabb5 "BRN"
#define knabb6 "EED"
#define knabb7 "TRD"
#define knabb8 "CHD"
#define knabb9 "CED"
#define knabbA "PHD"

#define statb 0x80
#define datab 0x00
#define noteof 0x00
#define noteon 0x01
#define aftert 0x02
#define contro 0x03
```

```
#define pressu 0x05
#define pitchw 0x06

//FFT Constant
#define npoint 128
#define l2n 7

//Variable Declaration
char lcdbuf[20];
unsigned char mst, smst, octoct;
unsigned char value[7] = {0, 1, 0, 0, 0, 0, 0};
unsigned char vol, sus, kn[3];
unsigned char lastvol, lastsus, lastkn[3];
unsigned char rn, ln;          //Recent Note, Last Note
unsigned int scount, grid;
char octaveshift = 0;
float rna, lna;                        //Recent Note Amplitude, Last
Note Amplitude
float xr[128];
float xi[128];

//Code Function
void SdCode(unsigned char dat1, unsigned char dat2, unsigned char
dat3)
{
      WriteUART1(dat1+value[0]);
      while(BusyUART1());
      WriteUART1(dat2);
      while(BusyUART1());
      if (dat3 < 0x80)
      {
            WriteUART1(dat3);
            while(BusyUART1());
      }
}

void lcd_display(char *p)
{
      while(*p)
      {
            lcd_text(*p);
            p++;
      }
}

float minus(float fv2)
{
      while (fv2 >= 1000000000) fv2 = fv2 - 1000000000;
      while (fv2 >= 100000000) fv2 = fv2 - 100000000;
      while (fv2 >= 10000000) fv2 = fv2 - 10000000;
      while (fv2 >= 1000000) fv2 = fv2 - 1000000;
      while (fv2 >= 100000) fv2 = fv2 - 100000;
      while (fv2 >= 10000) fv2 = fv2 - 10000;
      while (fv2 >= 1000) fv2 = fv2 - 1000;
      while (fv2 >= 100) fv2 = fv2 - 100;
      while (fv2 >= 10) fv2 = fv2 - 10;
      return (fv2);
}

char ftcc(float fv2)
{
      char cv;
      if (fv2 >= 0 && fv2 < 1) cv = '0';
      else if (fv2 >= 1 && fv2 < 2) cv = '1';
      else if (fv2 >= 2 && fv2 < 3) cv = '2';
      else if (fv2 >= 3 && fv2 < 4) cv = '3';
```

```
        else if (fv2 >= 4 && fv2 < 5) cv = '4';
        else if (fv2 >= 5 && fv2 < 6) cv = '5';
        else if (fv2 >= 6 && fv2 < 7) cv = '6';
        else if (fv2 >= 7 && fv2 < 8) cv = '7';
        else if (fv2 >= 8 && fv2 < 9) cv = '8';
        else if (fv2 >= 9 && fv2 < 10) cv = '9';
        return (cv);
}

char ftc(float fv, float power)
{
        char cv;
        if (fv < 0) fv = fv * (-1);
        fv = fv / power;
        fv = minus(fv);
        cv = ftcc(fv);
        return (cv);
}

char ftcs(float fv)
{
        char cv;
        if (fv >=0) cv = ' ';
        else cv = '-';
}

void ftl(unsigned char addr, float from, float to, float value, char
sign)
{
        lcd_command(addr);
        float i;
        if (sign == 's') lcd_text(ftcs(value));
        for (i = from;i >= to;i = i / 10)
        {
            if (i == 0.1) lcd_text('.');
            lcd_text(ftc(value,i));
        }
}

char itcx(int iv2)
{
        char cv;
        if (iv2 == 0) cv = '0';
        else if (iv2 == 1) cv = '1';
        else if (iv2 == 2) cv = '2';
        else if (iv2 == 3) cv = '3';
        else if (iv2 == 4) cv = '4';
        else if (iv2 == 5) cv = '5';
        else if (iv2 == 6) cv = '6';
        else if (iv2 == 7) cv = '7';
        else if (iv2 == 8) cv = '8';
        else if (iv2 == 9) cv = '9';
        return (cv);
}

char itc(int iv, unsigned int power)
{
        char cv;
        iv = iv / power;
        iv = iv % 10;
        cv = itcx(iv);
        return (cv);
}

char itcs(int iv)
{
```

```
        char cv;
        if (iv >= 0) cv = '+';
        else cv = '-';
}

void itl(unsigned char addr, unsigned int from, unsigned int to, int
value, char sign)
{
        lcd_command(addr);
        unsigned int i;
        if (sign == 's') lcd_text(itcs(value));
        if (value < 0) value = -value;
        for (i = from; i >= to; i = i / 10)
        {
                lcd_text(itc(value,i));
        }
}

void showoc()
{
        itl(0xC2, 1, 1, octaveshift / 12, 's');
}

void showpd()
{
        lcd_command(0xC0);
        if (sus == 0x7F)
                lcd_text('P');
        else
                lcd_text(' ');
}

void showvol()
{
        itl(0x81, 100, 1, vol, 'u');
}

void showkn(unsigned char k)
{
        itl(0xC5 + (4 * k), 100, 1, kn[k], 'u');
}

void showall()
{
        unsigned char i;
        unsigned char addrbus = 0x85;

        lcd_command(0x80);
        lcd_text('V');
        showvol();

        lcd_command(0x84);
        lcd_text('|');
        lcd_command(0x88);
        lcd_text('|');
        lcd_command(0x8C);
        lcd_text('|');

        for (i = 0; i < 3; i++)
        {
                lcd_command(addrbus);
                switch (value[3+i])
                {
                        case 0 : lcd_display(knabb0);
                        break;
                        case 1 : lcd_display(knabb1);
```

```
                        break;
                        case 2 : lcd_display(knabb2);
                        break;
                        case 3 : lcd_display(knabb3);
                        break;
                        case 4 : lcd_display(knabb4);
                        break;
                        case 5 : lcd_display(knabb5);
                        break;
                        case 6 : lcd_display(knabb6);
                        break;
                        case 7 : lcd_display(knabb7);
                        break;
                        case 8 : lcd_display(knabb8);
                        break;
                        case 9 : lcd_display(knabb9);
                        break;
                        case 10 : lcd_display(knabbA);
                        break;
                }
                addrbus += 0x04;
        showkn(i);
        }

        showpd();
        showoc();

        lcd_command(0xC4);
        lcd_text('|');
        lcd_command(0xC8);
        lcd_text('|');
        lcd_command(0xCC);
        lcd_text('|');
}

void Display(unsigned char Swb)
{
        if ((Swb == 1) | (mst == 1))
        {
                lcd_command(0x01);
                lcd_command(0x80);
                switch (mst)
                {
                        case 1 : showall();
                        break;
                        case 2 : lcd_display(upmsg2);
                        break;
                        case 3 : lcd_display(upmsg3);
                        break;
                        case 4 : lcd_display(upmsg4);
                        break;
                        case 5 : lcd_display(upmsg5);
                        break;
                        case 6 : lcd_display(upmsg6);
                        break;
                        case 7 : lcd_display(upmsg7);
                        break;
                        case 8 : lcd_display(upmsg8);
                        break;
                }
        }
        if (mst == 2)
        {
                lcd_command(0xC0);
                switch (smst)
                {
```

```
                case 1 : lcd_display(dnmsg1);
                break;
                case 2 : lcd_display(dnmsg2);
                break;
                case 3 : lcd_display(dnmsg3);
                break;
                case 4 : lcd_display(dnmsg4);
                break;
                case 5 : lcd_display(dnmsg5);
                break;
                case 6 : lcd_display(dnmsg6);
                break;
        }
}
else if ((mst == 3) | (mst == 4))
{
        sprintf(lcdbuf, "%3d", value[smst]);
        lcd_command(0xCD);
        lcd_display(lcdbuf);
}
else if ((mst >= 5) & (mst <= 7))
{
        lcd_command(0xC0);
        switch (value[smst])
        {
                case 0 : lcd_display(knmsg0);
                break;
                case 1 : lcd_display(knmsg1);
                break;
                case 2 : lcd_display(knmsg2);
                break;
                case 3 : lcd_display(knmsg3);
                break;
                case 4 : lcd_display(knmsg4);
                break;
                case 5 : lcd_display(knmsg5);
                break;
                case 6 : lcd_display(knmsg6);
                break;
                case 7 : lcd_display(knmsg7);
                break;
                case 8 : lcd_display(knmsg8);
                break;
                case 9 : lcd_display(knmsg9);
                break;
                case 10 : lcd_display(knmsgA);
                break;
        }
}
else if (mst == 8)
{
        lcd_command(0xC0);
        switch (value[smst])
        {
                case 0 : lcd_display(pdmsg1);
                break;
                case 1 : lcd_display(pdmsg2);
                break;
                case 2 : lcd_display(pdmsg3);
                break;
                case 3 : lcd_display(pdmsg4);
                break;
                case 4 : lcd_display(pdmsg5);
                break;
                case 5 : lcd_display(pdmsg6);
                break;
```

```
            }
        }
}

//Menu Function
void maindisplay(unsigned char Swb)
{
        if (Swb == 1)
        {
                mst = 2;
                smst = 1;
        }
}

void mainmenu(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = smst + 2;
                break;
                case 2 : if (smst == 1) mst = 1;
                                else if (smst == 3)
                                        smst = smst - 2;
                                else
                                        smst--;
                break;
                case 3 : if (smst == 6) mst = 1;
                                else if (smst == 1)
                                        smst = smst + 2;
                                else
                                        smst++;
                break;
        }
}

void channel(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = 2;
                break;
                case 2 : if (value[smst] != 1) value[smst]--;
                break;
                case 3 : if (value[smst] != 16) value[smst]++;
                break;
        }
}

void instrument(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = 2;
                break;
                case 2 : if (value[smst] != 0) value[smst]--;
                break;
                case 3 : if (value[smst] != 127) value[smst]++;
                break;
        }
}

void knob1(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = 2;
```

```
                break;
                case 2 : if (value[smst] != 0) value[smst]--;
                break;
                case 3 : if (value[smst] != 10) value[smst]++;
                break;
        }
}

void knob2(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = 2;
                break;
                case 2 : if (value[smst] != 0) value[smst]--;
                break;
                case 3 : if (value[smst] != 10) value[smst]++;
                break;
        }
}

void knob3(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = 2;
                break;
                case 2 : if (value[smst] != 0) value[smst]--;
                break;
                case 3 : if (value[smst] != 10) value[smst]++;
                break;
        }
}

void swped(unsigned char Swb)
{
        switch (Swb)
        {
                case 1 : mst = 2;
                break;
                case 2 : if (value[smst] != 0) value[smst]--;
                break;
                case 3 : if (value[smst] != 5) value[smst]++;
                break;
        }
}

void Bsensor(unsigned char Swb)
{
        if (Swb != 0)
        {
                switch (mst)
                {
                        case 1 : maindisplay(Swb);
                        break;
                        case 2 : mainmenu(Swb);
                        break;
                        case 3 : channel(Swb);
                        break;
                        case 4 : instrument(Swb);
                        break;
                        case 5 : knob1(Swb);
                        break;
                        case 6 : knob2(Swb);
                        break;
                        case 7 : knob3(Swb);
```

```
                        break;
                        case 8 : swped(Swb);
                        break;
                        default : mst = 0;
                        break;
                }
                Display(Swb);
        }
        if (mst == 1)
        {
                scount = 0;
                EnableIntT1;
        }
}

//Bit Reverasl Function
void bitreversal()
{
        unsigned char ii, jj, kk, bb, bb2;
        float xt;

        for (ii = l2n; ii >= 1; ii--)
        {
                bb = 1;
                for(jj = 1;jj <= ii; jj++)
                {
                        bb = bb * 2;
                }
                bb2 = bb/2-1;
                for (jj = 1; jj <= bb2; jj++)
                {
                        kk = jj * 2;
                        xt = xr[jj];
                        xr[jj] = xr[kk];
                        xr[kk] = xt;
                }
        }
}

//Fast Fourier Transformer
void fft()
{
        float wr[64] = { 1.000, 0.999, 0.995, 0.989, 0.981, 0.970,
0.957, 0.942, 0.924, 0.904, 0.882, 0.858, 0.831, 0.803, 0.773,
0.741, 0.707, 0.672, 0.634, 0.596, 0.556, 0.514, 0.471, 0.428,
0.383, 0.337, 0.290, 0.243, 0.195, 0.147, 0.098, 0.049, 0.000, -
0.049, -0.098, -0.147, -0.195, -0.243, -0.290, -0.337, -0.383, -
0.428, -0.471, -0.514, -0.556, -0.596, -0.634, -0.672, -0.707, -
0.741, -0.773, -0.803, -0.831, -0.858, -0.882, -0.904, -0.924, -
0.942, -0.957, -0.970, -0.981, -0.989, -0.995, -0.999};
        float wi[64] = { -0.000, -0.049, -0.098, -0.147, -0.195, -
0.243, -0.290, -0.337, -0.383, -0.428, -0.471, -0.514, -0.556, -
0.596, -0.634, -0.672, -0.707, -0.741, -0.773, -0.803, -0.831, -
0.858, -0.882, -0.904, -0.924, -0.942, -0.957, -0.970, -0.981, -
0.989, -0.995, -0.999, -1.000, -0.999, -0.995, -0.989, -0.981, -
0.970, -0.957, -0.942, -0.924, -0.904, -0.882, -0.858, -0.831, -
0.803, -0.773, -0.741, -0.707, -0.672, -0.634, -0.596, -0.556, -
0.514, -0.471, -0.428, -0.383, -0.337, -0.290, -0.243, -0.095, -
0.147, -0.098, -0.049};
        unsigned char b, b2, i, k, km, wp, wpp, j, jm, counter;
        float xar, xai, xbr, xbi;

        jm = npoint - 1;
        for (i = 1; i <= l2n; i++)
        {
                b = 1;
```

```
            for (j = 1; j <= i; j++)
            {
                  b = b * 2;
            }
            b2 = b / 2;
          km = b2 - 1;
            wpp = (npoint / b) % (npoint / 2);
            for (j = 0; j <= jm; j += b)
            {
                  wp = 0;
                  for (k = 0; k <= km; k++)
                  {
                        xar = (xr[j+k+b2] * wr[wp]) - (xi[j+k+b2] *
wi[wp]);
                        xai = (xr[j+k+b2] * wi[wp]) + (xi[j+k+b2] *
wr[wp]);
                        xr[j+k+b2] = xar;
                        xi[j+k+b2] = xai;
                        xar = xr[j+k] + xr[j+k+b2];
                        xai = xi[j+k] + xi[j+k+b2];
                        xbr = xr[j+k] - xr[j+k+b2];
                        xbi = xi[j+k] - xi[j+k+b2];
                        xr[j+k] = xar;
                        xi[j+k] = xai;
                        xr[j+k+b2] = xbr;
                        xi[j+k+b2] = xbi;
                        wp = wp + wpp;
                  }
            }
      }
}

//Complex Conjugate Multiplier
void conjugatemultiplier()
{
      unsigned char i;
      float xar, xai, xbr, xbi;

      for (i = 0; i <= (npoint - 1); i++)
      {
            xar = xr[i];
            xai = - xi[i];
            xbr = (xr[i] * xar) - (xi[i] * xai);
            xbi = (xr[i] * xai) + (xi[i] * xar);
            xr[i] = xbr;
            xi[i] = xbi;
      }
}

//Peak & Amplitude Devider
void peakandamp()
{
      float peak[23] = {6000, 7000, 8000, 8458.25, 6520.5, 27280.25,
12956.25, 5644.25, 12011.75, 12468.75, 17847.5, 20483.25, 44659.5,
45000, 17412.5, 19626.5, 12460.5, 33843.25, 17730.5, 11372.75,
15127.75, 15129, 11857.75};
      unsigned char notegrid[23] = {16, 17, 18, 19, 20, 21, 22, 24,
25, 26, 28, 30, 32, 33, 35, 38, 40, 42, 45, 47, 50, 53, 57};
      unsigned char i;

      rn = 0xFF;
      rna = 0;
      if (ln != 0xFF) lna = xr[ln];

      for (i = 0; i < 23; i++)
      {
```

```
                xr[notegrid[i]] = xr[notegrid[i]] / peak[i];
                if (xr[notegrid[i]] > rna)
                {
                        rna = xr[notegrid[i]];
                        rn = i;
                }
        }
}

//Decision & Control
void notesend()
{
        float th[23] = {0.180, 0.240, 0.185, 0.170, 0.203, 0.110,
0.114, 0.150, 0.162, 0.132, 0.121, 0.152, 0.120, 0.154, 0.168,
0.106, 0.234, 0.211, 0.195, 0.156, 0.286, 0.194, 0.256};
        if (rna >= th[rn])
        {
                if (rn != ln)
                {
                        if (ln != 0xFF)
                        {
                                SdCode(0x80, ln + 0x40 + octaveshift, 0x64);
                        }
                        SdCode(0x90, rn + 0x40 + octaveshift, 0x64);
                        ln = rn;
                }
        }
        else
        {
                if (ln != 0xFF)
                {
                        if (lna <= 5) // 5
                        {
                                SdCode(0x80, ln + 0x40 + octaveshift, 0x64);
                                ln = 0xFF;
                        }
                }
        }
}

void volped()
{
        if (lastvol != vol)
        {
                lastvol = vol;
                SdCode(0xB0, 0x07, vol);
                showvol();
        }
}

void askn(unsigned char var, unsigned char val)
{
        unsigned char con[10] = {0x46, 0x47, 0x48, 0x49, 0x4A, 0x5B,
0x5C, 0x5D, 0x5E, 0x5F};
        kn[val] = var;
        if ((lastkn[val] != kn[val]) & (value[3 + val] != 0))
        {
                lastkn[val] = kn[val];
                SdCode(0xB0, con[value[3 + val] - 1], kn[val]);
                showkn(val);
        }
}

void suspd()
{
        unsigned char con[6] = {0x40, 0x41, 0x42, 0x43, 0x44, 0x45};
```

```
        if (lastsus != sus)
        {
                lastsus = sus;
                SdCode(0xB0, con[value[6]], sus);
                showpd();
        }
}

//Interrupt
void _ISR _INT0Interrupt(void)
{
        unsigned char Swb = 1;
        DisableIntT1;
        Bsensor(Swb);
        delay_ms(200);
        IFS0bits.INT0IF = 0;
}

void _ISR _INT1Interrupt(void)
{
        unsigned char Swb = 2;
        if (mst == 1)
        {
                if (octaveshift >= -48)
                {
                        octaveshift -= 12;
                        if (ln != 0xFF) octoct = 1;
                        delay_ms(200);
                }
        }
        else
        {
                DisableIntT1;
                Bsensor(Swb);
        }
        IFS1bits.INT1IF = 0;
}

void _ISR _INT2Interrupt(void)
{
        unsigned char Swb = 3;
        if (mst == 1)
        {
                if (octaveshift <= 48)
                {
                        octaveshift += 12;
                        if (ln != 0xFF) octoct = 2;
                        delay_ms(200);
                }
        }
        else
        {
                DisableIntT1;
                Bsensor(Swb);
        }
        IFS1bits.INT2IF = 0;
}

void _ISR _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;
}

void _ISR _U1RXInterrupt(void)
{
        IFS0bits.U1RXIF = 0;
```

```
}

void _ISR _T1Interrupt(void)
{
      unsigned char i;
      int adcbuf;

      ConvertADC10();
      while(ADCON1bits.SAMP);
      while(BusyADC10());
      adcbuf = ReadADC10(1);

      if ((scount % 8) == 0)
      {
            LATDbits.LATD3 = !LATDbits.LATD3;
            xr[scount / 8] = adcbuf;
      }

      if (scount == 1016)
      {
            for (i = 0;i < 128;i++)
            {
                  xr[i] = xr[i] / 64;
            }

            if (octoct == 1)
            {
                  SdCode(0x80, ln + 0x40 + octaveshift + 12, 0x64);
                  SdCode(0x90, ln + 0x40 + octaveshift, 0x64);
                  octoct = 0;
            }
            else if (octoct == 2)
            {
                  SdCode(0x80, ln + 0x40 + octaveshift - 12, 0x64);
                  SdCode(0x90, ln + 0x40 + octaveshift, 0x64);
                  octoct = 0;
            }

//Calculation
            bitreversal();
            fft();
            conjugatemultiplier();

//Selection & Control
            peakandamp();
            notesend();

            adcbuf = ReadADC10(2);
            vol = adcbuf / 8;
            volped();

            adcbuf = ReadADC10(3);
            askn(adcbuf / 8, 0);

            adcbuf = ReadADC10(6);
            askn(adcbuf / 8, 1);

            adcbuf = ReadADC10(7);
            askn(adcbuf / 8, 2);

            showoc();
            suspd();

            scount = 0;
      }
      else scount++;
```

```
        IFS0bits.T1IF = 0;
}

//ADC Initialize
void adc_init()
{
        unsigned int Channel, PinConfig, Scanselect, Adcon3_reg,
Adcon2_reg, Adcon1_reg;

        ADCON1bits.ADON = 0;            // Turn off ADC

        Channel      =      ADC_CH0_POS_SAMPLEA_AN1 &
                            ADC_CH0_POS_SAMPLEA_AN2 &
                            ADC_CH0_POS_SAMPLEA_AN3 &
                            ADC_CH0_POS_SAMPLEA_AN4 &
                            ADC_CH0_POS_SAMPLEA_AN5 &
                            ADC_CH0_POS_SAMPLEA_AN6 &
                            ADC_CH0_POS_SAMPLEA_AN7 &
                            ADC_CH0_NEG_SAMPLEA_NVREF;

        SetChanADC10(Channel);
        ConfigIntADC10(ADC_INT_DISABLE);

        PinConfig    =      ENABLE_AN1_ANA &
                            ENABLE_AN2_ANA &
                            ENABLE_AN3_ANA &
                            ENABLE_AN4_ANA &
                            ENABLE_AN5_ANA &
                            ENABLE_AN6_ANA &
                            ENABLE_AN7_ANA ;

        Scanselect   =      SKIP_SCAN_AN8;

        Adcon3_reg   =      ADC_SAMPLE_TIME_10 &
                            ADC_CONV_CLK_INTERNAL_RC &
                            ADC_CONV_CLK_13Tcy;

        Adcon2_reg   =      ADC_VREF_EXT_AVSS &
                            ADC_SCAN_ON &
                            ADC_ALT_BUF_OFF &
                            ADC_ALT_INPUT_OFF &
                            ADC_CONVERT_CH0 &
                            ADC_SAMPLES_PER_INT_8;

        Adcon1_reg   =      ADC_MODULE_ON &
                            ADC_IDLE_CONTINUE &
                            ADC_FORMAT_INTG &
                            ADC_CLK_MANUAL &
                            ADC_SAMPLE_SIMULTANEOUS &
                            ADC_AUTO_SAMPLING_ON;

        OpenADC10(Adcon1_reg, Adcon2_reg,   Adcon3_reg,PinConfig,
Scanselect);
}

void timer_init()
{
        unsigned int match_value;

        ConfigIntTimer1(  T1_INT_PRIOR_7 &
                          T1_INT_ON);

        WriteTimer1(0);
```

```
        match_value = 937;              //Sampling rate setting (2666.66
x 8 Hz)

        OpenTimer1( T1_ON &
                    T1_GATE_OFF &
                    T1_IDLE_CON &
                    T1_PS_1_1 &
                    T1_SYNC_EXT_OFF &
                    T1_SOURCE_INT, match_value);
}

//UART Initialize
void uart1_init()
{
        unsigned int baudvalue;
        unsigned int U1MODEvalue;
        unsigned int U1STAvalue;

        CloseUART1();
        ConfigIntUART1(   UART_RX_INT_EN &
                          UART_RX_INT_PR1 &
                          UART_TX_INT_EN &
                          UART_TX_INT_PR1);

        baudvalue = 39;   // Baud rate 31250 bps at 10MHz PLLx8

        U1MODEvalue =     UART_EN &
                          UART_IDLE_CON &
                          UART_RX_TX&
                          UART_DIS_WAKE &
                          UART_DIS_LOOPBACK &
                          UART_DIS_ABAUD &
                          UART_NO_PAR_8BIT &
                          UART_1STOPBIT;

        U1STAvalue  =     UART_INT_TX_BUF_EMPTY &
                          UART_TX_PIN_NORMAL &
                          UART_TX_ENABLE &
                          UART_INT_RX_3_4_FUL&
                          UART_ADR_DETECT_DIS &
                          UART_RX_OVERRUN_CLEAR;

        OpenUART1(U1MODEvalue, U1STAvalue, baudvalue);
}

//System Initialize
void system_int(void)
{
        mst = 1;
        smst = 1;
        scount = 0;
        grid = 14;
        rn = 0x00;
        ln = 0x00;
        rna = 0;
        lna = 0;

        adc_init();
        lcd_init();

        uart1_init();
        TRISDbits.TRISD3 = 0;
        TRISEbits.TRISE8 = 1;

        delay_ms(200);
```

```
        ConfigINT0(RISING_EDGE_INT & EXT_INT_ENABLE & EXT_INT_PRI_4);
        ConfigINT1(RISING_EDGE_INT & EXT_INT_ENABLE & EXT_INT_PRI_3);
        ConfigINT2(RISING_EDGE_INT & EXT_INT_ENABLE & EXT_INT_PRI_2);

        Display(1);

        timer_init();
}

//Main
int main(void)
{
        system_int();
        while(1)
        {
                if(PORTCbits.RC14 == 1) sus = 0x00;
                else sus = 0x7F;
        }
}
```