การประมวลผลสัญญาณเสียงดิจิตอลสำหรับเอฟเฟคกีตาร์ไฟฟ้า

DIGITAL SOUND EFFECT PROCESSING FOR ELECTRIC GUITAR

โดย

นายเสฏฐนันท์ แกระวงค์

นายอุกฤษฏ์ ทะริยะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2555

การประมวลผลสัญญาณเสียงดิจิตอลสำหรับเอฟเฟคกีตาร์ไฟฟ้า

DIGITAL SOUND EFFECT PROCESSING FOR ELECTRIC GUITAR

โดย

นายเสฏฐนันท์	แกระวงค์	52011353
นายอุกฤษฏ์	ทะริยะ	52011468

อาจารย์ที่ปรึกษา

ผศ.ดร.ศรวัฒน์ ชิวปรีชา

ผศ.อัครพล ตรีรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2555

ผ่านการตรวจรูปเล่มแล้ว

(.....)

อาจารย์ที่ปรึกษา

8 พ.ค. 2556

ผ่านการตรวจชิ้นงานแล้ว

(.....)

กรรมการผู้ตรวจชิ้นงาน

29 / พ.ค. / 56

ปริญญานิพนธ์ปีการศึกษา 2555

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประมวลผลสัญญาณเสียงดิจิตอลสำหรับเอฟเฟคกีตาร์ไฟฟ้า

DIGITAL SOUND EFFECT PROCESSING FOR ELECTRIC GUITAR

ผู้จัดทำ

1. นายเสฏฐนันท์ แกระวงค์ 52011353
2. นายอุกฤษฎ์ ทะริยะ 52011468

..... **อาจารย์ที่ปรึกษา**

(ผศ.ดร.ศรวัฒน์ ชิวปรีชา)

..... **อาจารย์ที่ปรึกษา**

(ผศ.อัครพล ตรีรัตน์)

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้ ได้สำเร็จลุล่วงไปได้ด้วยดี อาจารย์ที่ปรึกษาให้คำแนะนำและให้ความช่วยเหลือในการทำงาน จึงขอขอบพระคุณอาจารย์ที่ปรึกษา คือ ผศ.ดร.ศรวัฒน์ ชิวปรีชา และ ผศ. อัครพล ตรีรัตน์ และขอขอบคุณเพื่อนๆ และพี่ๆ ทุกคนที่คอยให้คำแนะนำ ให้คำปรึกษาเกี่ยวกับความรู้ต่างๆ ทางคณะผู้จัดทำจึงหวังเป็นอย่างยิ่งว่าโครงงานนี้ จะสามารถนำไปเป็นแนวทางในการประยุกต์ใช้เพื่อให้เกิดประโยชน์ต่อๆ ไปได้

นายเสฏฐนันท์ แกระวงค์
นายอุกฤษฏ์ ทะริยะ
ผู้จัดทำ

การประมวลผลสัญญาณเสียงดิจิตอลสำหรับเอฟเฟคกีตาร์ไฟฟ้า
DIGITAL SOUND EFFECT PROCESSING FOR ELECTRIC GUITAR

โดย นายเสฏฐนันท์ แกระวงค์ 52011353
นายอุกฤษฏ์ ทะริยะ 52011468

อาจารย์ที่ปรึกษา ผศ.ดร.ศรวัฒน์ ชิวปรีชา
ผศ.อัครพล ตรีรัตน์

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการศึกษาการออกแบบโปรแกรมประมวลผลสัญญาณเสียงดิจิตอล เพื่อนำไปประยุกต์ใช้งานในการปรับแต่งเสียงกีตาร์ไฟฟ้าให้มีลักษณะการทำงานเสมือน MULTI-EFFECT โดยการใช้โปรแกรม MATLAB ในการจำลองการทำงานของการประมวลผลสัญญาณเสียงดิจิตอล และจากนั้นทำการออกแบบให้เป็นระบบสำเร็จรูปที่ประกอบด้วยซอฟต์แวร์และแผงควบคุมด้วยเท้า เพื่อให้ได้เป็นโปรแกรมต้นแบบรวมทั้งนำมาทดสอบในการปรับแต่งเสียงกีตาร์ไฟฟ้าให้มีลักษณะเสียงที่ต้องการ

ABSTRACT

This project is to design a digital sound effect processing system similar to a Multi-effect box for electric guitar. The digital sound effect processing is simulated using MATLAB. A system consists of an executable software and a foot switch for sound to synthesis will be developed and tested on an electric guitar.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในโลกยุคปัจจุบันที่มีเครื่องดนตรีหลายๆชนิดแต่ละชนิดสามารถก่อกำเนิดเสียงที่ไพเราะให้กับผู้ฟังที่ได้รับฟังและชื่นชมกับสุนทรีของเสียงที่ถูกบรรเลงออกมาจากเครื่องดนตรีผ่านทางโสตประสาทและรับรู้เสียงของตัวโน๊ตแต่ละตัวที่ได้ผสมกลมกลืนจนเป็นจังหวะทำนองของบทเพลงได้อย่างไพเราะลงตัว ซึ่งในปัจจุบันนี้เทคโนโลยีได้เข้ามามีบทบาทกับดนตรีเพิ่มมากขึ้น ทั้งทางด้านเสียงดนตรีและทางเครื่องดนตรี โดยการนำเทคโนโลยีมาใช้กับดนตรีนั้นจะช่วยให้เกิดความสะดวกสะบายสำหรับนักดนตรี และช่วยให้เสียงดนตรีนั้นเกิดความไพเราะขึ้นด้วย ซึ่งการนำเทคโนโลยีมาใช้กับดนตรีนั้นเรียกว่าสนเทศศาสตร์การดนตรี (Music Informatics) เป็นการนาเอาศาสตร์หลายๆ แขนงมารวมเข้าด้วยกัน เช่น วิทยาศาสตร์ วิศวกรรมศาสตร์ ศิลปะการดนตรี การวิเคราะห์เสียงดนตรี อะคูสติก (Acoustics) เป็นต้น เพื่อยกระดับการดนตรีให้ดีขึ้นทั้งทางด้านความสะดวกสบาย ความถูกต้องและความไพเราะของเสียงดนตรี

ในปัจจุบันนี้ หลักการของการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) ได้เข้ามาบทบาทสำคัญในการดำรงชีวิตสามารถนำหลักการนี้ไปประยุกต์ใช้ในด้านต่างๆ เช่น คอมพิวเตอร์ ด้านโทรคมนาคม เป็นต้น นอกจากนี้การนำเอาหลักการของการประมวลสัญญาณดิจิตอลมาใช้ในสามารถทำให้เกิดการประยุกต์การออกแบบฟิลเตอร์เพื่อใช้งานในด้านของเสียง เพื่อให้เกิดความหลากหลายของสัญญาณเสียงที่ได้และยังสามารถนำไปแก้ไขปัญหาในด้านของสัญญาณเสียงอีกทั้งสัญญาณเสียงที่ได้มีความไพเราะและมีความหลากหลายมากยิ่งขึ้นด้วย ดังนั้นจึงเกิดแนวคิดที่ต้องการศึกษาการออกแบบโปรแกรมประมวลผลสัญญาณเสียงดิจิตอล ที่สามารถนำไปประยุกต์ใช้งานในการปรับแต่งเสียงกีตาร์ไฟฟ้าให้มีลักษณะการทำงานเสมือน Multi-effect ซึ่งสามารถให้สัญญาณเสียงที่มีได้ความหลากหลายมากยิ่งขึ้น

1.2 วัตถุประสงค์

1) ศึกษาการออกแบบวงจรกรองความถี่เสียงดิจิตอล
2) ศึกษาการใช้งานโปรแกรม MATLAB ในการจำลองการทำงาน
3) ศึกษาการออกโปรแกรมโดยใช้ Visual C
4) ศึกษาการออกแบบแผงควบคุมในการสับเปลี่ยนการทำงานของเอฟเฟคกีตาร์

1.3 ขอบเขตของโครงงาน

ภาคเรียนที่ 1:
1. ออกแบบวงจรกรองความถี่เสียงดิจิตอลที่ใช้งานกับกีตาร์ไฟฟ้า
2. จำลองการทำงานของวงจรกรองความถี่เสียงดิจิตอลโดยใช้ MATLAB
3. ออกแบบส่วนการควบคุมเพื่ออำนวยความสะดวกในการใช้งาน

ภาคเรียนที่ 2:
1. ศึกษาการออกแบบซอฟต์แวร์โดยใช้ Visual C
2. จำลองการทำงานของวงจรกรองความถี่เสียงดิจิตอลโดยใช้ Visual C
3. ออกแบบระบบสำเร็จรูปที่ประกอบไปด้วยซอฟต์แวร์และแผงควบคุมด้วยเท้า

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 การประมวลผลสัญญาณดิจิตอล (Digital Signal Processing)

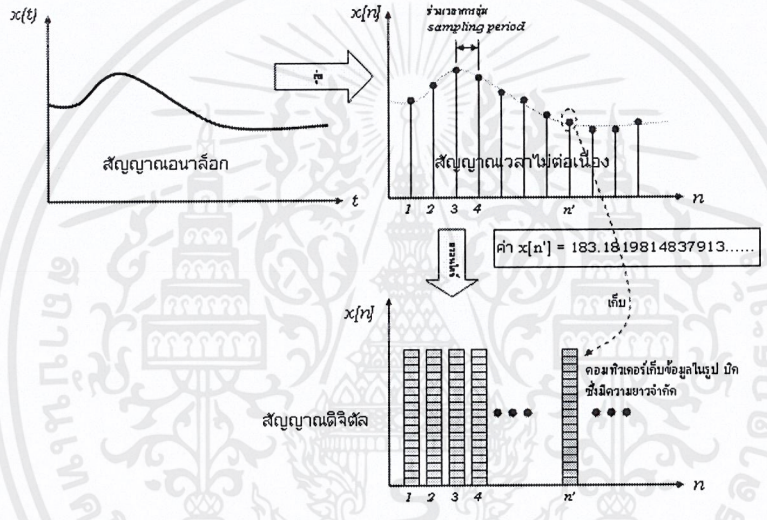
2.1.1 การประมวลผลสัญญาณดิจิตอล

การประมวลผลสัญญาณดิจิตอล หรือที่เรียกว่า ดีเอสพี (DSP) เป็นการศึกษาการประมวลผลสัญญาณที่อยู่ในรูปดิจิตอล (digital) โดยทั่วๆ ไป การประมวลผลสัญญาณ อาจแบ่งได้ตาม:

- รูปแบบของตัวแทนสัญญาณ : การประมวลผลสัญญาณดิจิตอล (digital signal processing) และ การประมวลผลสัญญาณอนาล็อก (analog signal processing)
- คุณสมบัติของสัญญาณ : การประมวลผลสัญญาณไม่สุ่ม (deterministic signal processing) และ การประมวลผลสัญญาณสุ่ม (stochastic/statistical signal processing)
- ลักษณะการประมวลสัญญาณ : เชิงเส้น (linear signal processing) และ ไม่เป็นเชิงเส้น (nonlinear signal processing)
- และ อื่นๆ ที่แบ่งตามคุณลักษณะเฉพาะของสัญญาณ หรือ ลักษณะเฉพาะของการประมวลผล เช่น adaptive signal processing, multiresolution signal processing, chaotic signal processing ฯลฯ

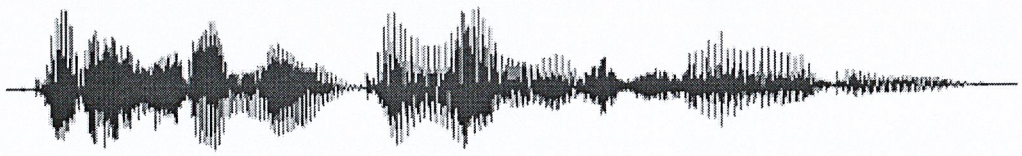
การประมวลผลสัญญาณดิจิตอลนี้อาจแบ่งออกได้เป็นในส่วนของ ซอฟต์แวร์ และ ฮาร์ดแวร์ หรือตามการประยุกต์เป็น การประมวลผลสัญญาณเสียง (audio signal processing) การประมวลผลภาพดิจิตอล (digital image processing) และ การประมวลผลคำพูด (speech processing)

ถึงแม้ว่าในการประมวลผลสัญญาณดิจิตอลนั้น สัญญาณที่ใช้ในพิจารณากันจะเป็นดิจิตอล แต่โดยทั่วไปสัญญาณเหล่านี้จากแหล่งกำเนิด จะอยู่ในรูปเดิมที่เป็นอนาล็อก การได้มาซึ่งสัญญาณดิจิตอลซึ่งเป็นตัวแทนสัญญาณอนาล็อกที่เราสนใจนี้ จะต้องผ่านกระบวนการแปลงสัญญาณอนาล็อกเป็นดิจิตอล (Analog-to-Digital Conversion, (ADC)) หรือการดิจิไทซ์ (digitization) ซึ่งประกอบด้วย การสุ่มตัวอย่าง (sampling) และการควอนไทซ์ (quantization) ให้อยู่ในรูปดิจิตอลก่อนที่จะทำการประมวลผลต่อไป ซึ่งสามารถแสดงการสุ่มสัญญาณ (signal sampling) และการควอนไทซ์ (quantization) ได้ดังรูปที่ 2.1



รูปที่ 2.1 รูปแสดงกระบวนการชักตัวอย่างสัญญาณและควอนไทซ์

โดเมนของเวลาและตำแหน่ง (Temporal and spatial domain)



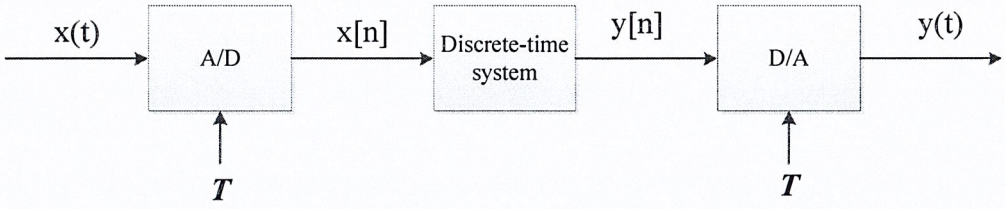
รูปที่ 2.2 รูปแสดงสัญญาณเสียงในรูปของตัวแปรเวลา

2.1.2 การประมวลผลสัญญาณเสียง

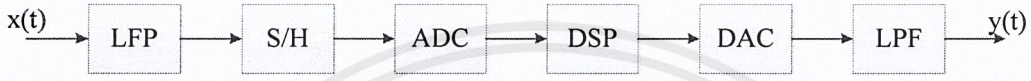
การประมวลผลสัญญาณเสียง หรือ การประมวลเสียง เป็นกระบวนการเกี่ยวกับ 'ตัวแทนสัญญาณเสียง' หรือ เสียงตัวแทนสัญญาณเสียงนี้อาจจะอยู่ในรูปดิจิตอล หรืออนาล็อก ซึ่งตัวแทนสัญญาณเสียงในแบบอนาล็อกมักจะอยู่ในรูปของสัญญาณไฟฟ้า โดยความต่างศักย์ไฟฟ้าจะแทนความดันอากาศของคลื่นเสียง ในทำนองเดียวกัน ตัวแทนสัญญาณเสียงแบบสัญญาณดิจิตอลจะแทนความดันนั้นด้วยชุดของสัญลักษณ์ ซึ่งโดยทั่วๆ ไปคือเลขฐานสอง

2.2 ตัวกรองเชิงเลข (Digital Filter)

โดยทั่วไปสัญญาณไฟฟ้าที่เกิดจากการตรวจจับ (Sensor) ส่วนใหญ่มักจะเป็นสัญญาณอนาล็อก ตัวอย่างเช่น สัญญาณเสียที่ได้จากไมโครโฟน สัญญาณรูปที่ได้จากกล้องวีดีโอ แรงดันไฟฟ้าจากตัวจับอุณหภูมิ สัญญาณทางการแพทย์ ซึ่งล้วนที่กล่าวมานี้เป็นสัญญาณอนาล็อกหรือสัญญาณที่มีความต่อเนื่องทางเวลา (Continuous Time Signal) ทั้งสิ้น ดังนั้นหากจะทำการวิเคราะห์สัญญาณเหล่านี้ด้วยระบบประมวลผลสัญญาณดิจิตอลต้องจำเป็นต้องแปลงสัญญาณอนาล็อกหรือสัญญาณต่อเนื่องทางเวลาให้เป็นสัญญาณดิจิตอลหรือสัญญาณที่ไม่ต่อเนื่องทางเวลา (Discrete Time Signal) โดยใช้วงจรแปลงอนาล็อกเป็นดิจิตอล (Analog to Digital Converter) จากนั้นส่งข้อมูลเข้าการประมวลผลสัญญาณแบบไม่ต่อเนื่องทางเวลา (Discrete Time System) เพื่อทำการคำนวณผลที่ได้ ซึ่งผลที่ได้จากการคำนวณก็จะถูกแปลงกลับเป็นสัญญาณอนาล็อก โดยวงจรแปลงดิจิตอลเป็นอนาล็อก โดยวงจรแปลงดิจิตอลเป็นอนาล็อกแสดงได้ตามรูปที่ 2.3 แต่ในการใช้งานจริงการป้อนสัญญาณที่ต่อเนื่องเข้าวงจรแปลงอนาล็อกเป็นดิจิตอลอาจเกิดข้อผิดผลาดเนื่องจากการสุ่มสัญญาณ โดยเฉพาะสัญญาณที่ความถี่สูงความผิดผลาดนี้เรียกว่า Aliasing ซึ่งการแก้ไขสามารถทำได้โดยการต่อ Anti-aliasing Filter (Low-pass Filter) เข้าทางด้านอินพุตและทางด้านเอาต์พุตเช่นเดียวกัน แสดงดังรูปที่ 2.4



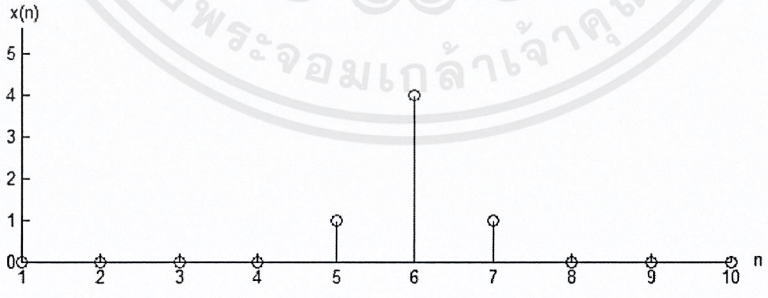
รูปที่ 2.3 รูปแสดงแผนผังโครงสร้างของระบบประมวลผลสัญญาณดิจิตอล



รูปที่ 2.4 รูปแสดงแผนผังโครงสร้างของระบบประมวณผลสัญญาณดิจิตอลที่สมบูรณ์

2.2.1 สัญญาณแบบไม่ต่อเนื่องทางเวลา (Discrete Time Signal)

ในระบบประมวลผลสัญญาณแบบไม่ต่อเนื่องทางเวลา สัญญาณที่ใช้ในระบบก็จะเป็นสัญญาณที่ไม่ต่อเนื่องเช่นเดียวกัน ดังที่ได้กล่าวมาแล้วซึ่งสัญญาณที่ไม่ต่อเนื่องนี้จะมีลักษณะดังรูปที่ 2.5 โดยจะเห็นได้ว่าสัญญาณที่ไม่ต่อเนื่องคือสัญญาณที่มีค่าใดค่าหนึ่ง ณ เวลาหนึ่งของสัญญาณที่ต่อเนื่อง แต่จะมีระยะเวลาห่างที่เท่ากันในแต่ละจุด เรียกระยะเวลาห่างนี้ว่า เวลาการสุ่ม (Sampling Time), อัตราการสุ่ม (Sapling Rate) หรือช่วงการสุ่ม (Sampling Periods) สัญญาณที่ไม่ต่อเนื่องสามารถเรียกเป็นลำดับสัญญาณ (Sequence Signal) และสามารถเขียนแทนได้ด้วยสมการคณิตศาสตร์ได้หลายๆ แบบดังต่อไปนี้



รูปที่ 2.5 รูปแสดงสัญญาณไม่ต่อเนื่องทางเวลา

เขียนในรูปของฟังก์ชั่น (Functional Representation)

$$x(n)=\begin{cases}1 & ,n=1,3\\ 4 & ,n=2\\ 0 & ,n>4\end{cases} \tag{2.1}$$

เขียนในรูปสมการแจกแจง (Tabular Representation)

n	...	-2	-1	0	1	2	3	...
$x(n)$	...	0	0	0	1	4	1	...

เขียนในรูปลำดับ (Sequence Representation)

$$x(n)=\{\ldots,0,0,1,4,1,0,0,\ldots\} \tag{2.2}$$

$$x(n)=\{0,1,4,1,0,0,\ldots\} \tag{2.3}$$

$$x(n)=\{3,-1,4,1,0,0\} \tag{2.4}$$

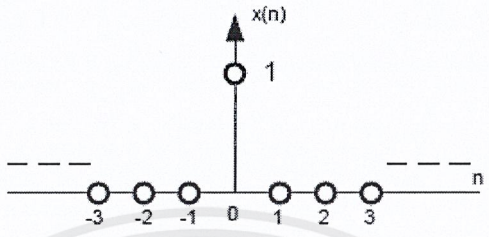
$$x(n)=\{0,1,4,1\} \tag{2.5}$$

สมาการที่ 2.2 และ 2.3 เป็นสมการลำดับแบบไม่มีจุดจบ (Infinite – duration Signal or Sequence) ส่วนสมการที่ 2.4 และ 2.5 เป็นลำดับแบบมีจุดจบ (Finite – duration Signal or Sequence)

2.2.1.1 ฟังก์ชันอิมพัลส์หนึ่งหน่วย

นิยามดังสมการที่ 2.6 และมีสัญญาณดังรูปที่ 2.6

$$\delta(n)=\begin{cases}0 & ,n\neq 0\\ 1 & ,n=0\end{cases} \tag{2.6}$$

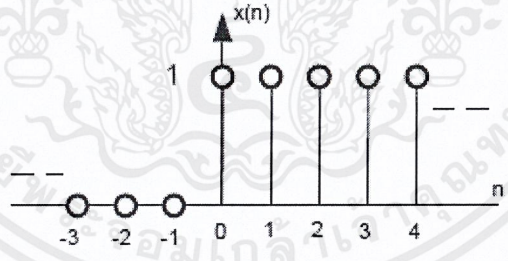


รูปที่ 2.6 รูปแสดงฟังก์ชั่นอิมพัลส์หนึ่งหน่วย

2.2.1.2 ฟังก์ชั่นขั้นหนึ่งหน่วย

นิยามดังสมการที่ 2.7 และมีรูปสัญญาณดังรูปที่ 2.7

$$u(n)=\begin{cases}0 & ,n<0\\ 1 & ,n\geq 0\end{cases} \tag{2.7}$$

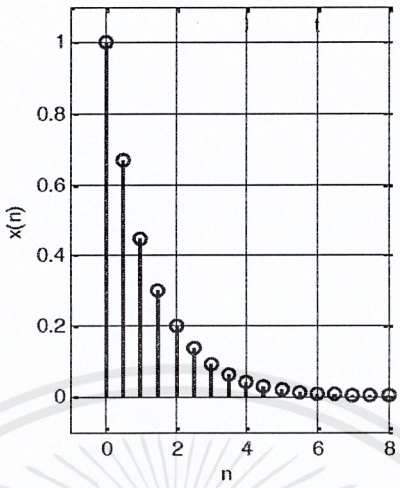


รูปที่ 2.7 รูปแสดงฟังก์ชั่นขั้นหนึ่งหน่วย

2.2.1.3 ฟังก์ชั่นเอกซ์โพเนนเชียล

นิยามดังสมการที่ 2.8 และมีรูปสัญญาณดังรูปที่ 2.8

$$x(n)=\alpha^{n} \tag{2.8}$$

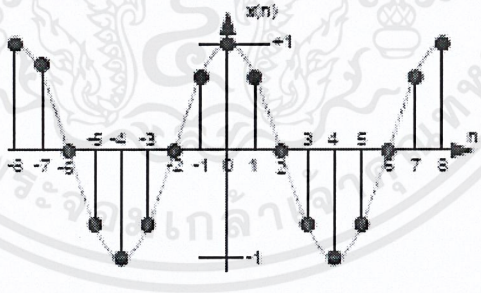


รูปที่ 2.8 รูปแสดงฟังก์ชั่นเอกโพเนนเชียล *x(n)*

2.2.1.4 ฟังก์ชันไซนด์

นิยามดังสมการที่ 2.9 และมีรูปสัญญาณดังรูปที่ 2.9

$$x(n) = A\cos(\omega_0 n + \phi) \tag{2.9}$$



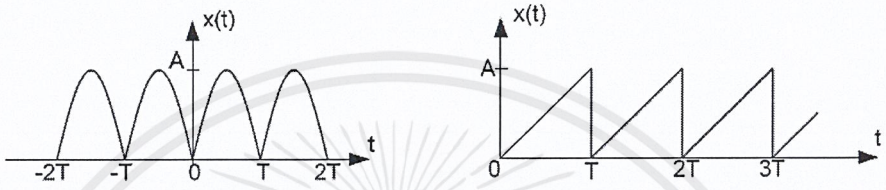
รูปที่ 2.9 รูปแสดงฟังก์ชั่นไซด์

2.2.1.5 สัญญาณมีคาบและสัญญาณไม่มีคาบ

สัญญาณมีคาบมีนิยามดังสมการที่ 2.10 มีรูปสัญญาณดังรูปที่ 2.10

$$x(n+N)=x(n) \tag{2.10}$$

เมื่อ *N* คือ Fundamental Period และลำดับสัญญาณไม่เป็นดังสมการที่ 2.10 จะเป็นลำดับสัญญาณแบบไม่มีคาบ

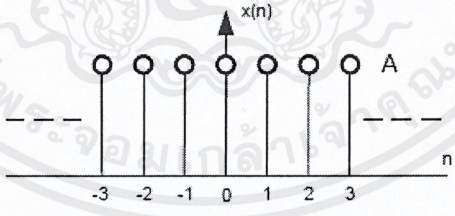


รูปที่ 2.10 รูปแสดงสัญญาณแบบมีคาบ

2.2.1.6 สัญญาณสมมาตร (even) และไม่สมมาตร (odd)

ลำดับสัญญาณที่สมมาตรกัน (Symmetric) มีนิยามเป็นดังสมการที่ 2.11 มีรูปสัญญาณดังรูปที่ 2.11

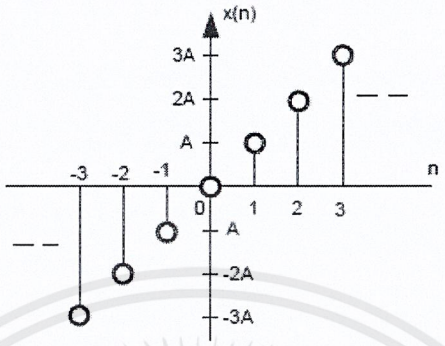
$$x(-n)=x(n) \tag{2.11}$$



รูปที่ 2.11 รูปแสดงสัญญาณที่สมมาตร

ลำดับสัญญาณที่ไม่สมมาตรกัน (Asymmetric) มีนิยามดังสมการที่ 2.12 มีรูปสมการดังรูปที่ 2.12

$$x(-n) = -x(n) \tag{2.12}$$



รูปที่ 2.12 รูปแสดงสัญญาณที่ไม่สมมาตร

2.2.1.7 การกระทำของสัญญาณแบบไม่ต่อเนื่องทางเวลา

1) การเลื่อนสัญญาณอาจเรียกว่าการหน่วงสัญญาณก็ได้เขียนเป็นสมการได้ดังสมการที่ 2.13

$$y(n) = x(n-k) \tag{2.13}$$

เมื่อ k คือ จำนวนเต็ม
$x(n)$ คือ อินพุตของระบบ
$y(n)$ คือ เอาต์พุตของระบบ

2) การบวกสัญญาณ 2 สัญญาณ หรือkสัญญาณเข้าด้วยกันได้ดังสมการที่ 2.14 และสมการที่ 2.15

$$y(n) = x_1(n) + x_2(n) \tag{2.14}$$

$$y(n) = x_1(n) + x_2(n) + \ldots + x_k(n) \tag{2.15}$$

3) การคูณสัญญาณ 2 สัญญาณหรือ k สัญญาณเข้าด้วยกันได้ดังสมการที่ 2.16 และสมการที่ 2.17

$$y(n) = x_1(n) + x_2(n) \tag{2.16}$$

$$y(n) = x_1(n) + x_2(n) + \ldots + x_k(n) \tag{2.17}$$

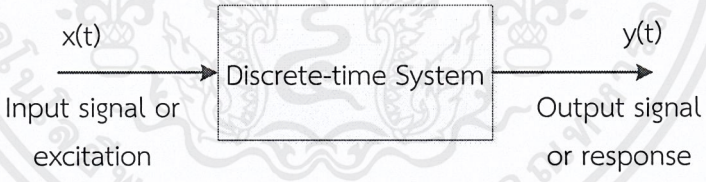
4) การสเกลสัญญาณเป็นการปรับขนาดสัญญาณ ได้โดยการคูณสัญญาณเข้ากับค่าคงที่ดังสมการที่ 2.18

$$y(n) = Ax(n) \tag{2.18}$$

เมื่อ A คือ ค่าคงที่

2.2.2 ระบบแบบไม่ต่อเนื่องทางเวลา

ระบบแบบไม่ต่อเนื่องมีบล๊อกไดอะแกรมดังรูปที่ 2.13



รูปที่ 2.13 รูปแสดงแผนผังโครงสร้างของระบบแบบไม่ต่อเนื่อง

เมื่อ $x(n)$ เป็นสัญญาณอินพุตหรือ Excitation ของระบบแบบไม่ต่อเนื่อง

$y(n)$ เป็นสัญญาณเอาต์พุต Response ของระบบแบบไม่ต่อเนื่อง

สามารถเขียนความสัมพันธ์ระหว่างอินพุตกับเอาต์พุตได้ดังสมการที่ 2.19

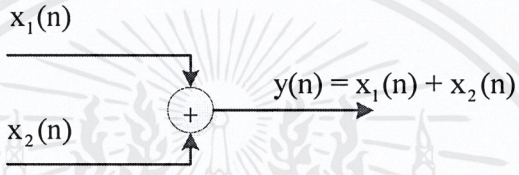
$$y(n) = \tau(x(n)) \tag{2.19}$$

เมื่อ τ เป็นตัวกระทำ (Operator)

2.2.2.1บล็อกไดอะแกรมที่ใช้แสดงในระบบไม่ต่อเนื่อง

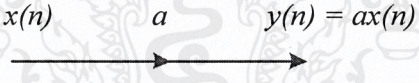
ในระบบแบบไม่ต่อเนื่องสามารถแทนได้ด้วยไดอะแกรมดังต่อไปนี้

1) ตัวบวก (Adder) ใช้สำหรับบวกสัญญาณเข้าด้วยกันแสดงดังรูปที่ 2.14



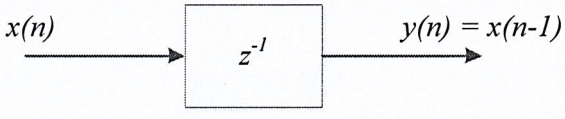
รูปที่ 2.14 รูปแสดงแผนผังโครงสร้างของการบวกในระบบแบบไม่ต่อเนื่อง

2) ตัวคูณคงที่ (Constant Multiplier) ใช้คูณค่าคงที่กับสัญญาณ แสดงดังรูปที่ 2.15



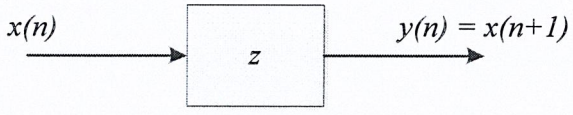
รูปที่ 2.15 รูปแสดงแผนผังโครงสร้างของตัวคูณสัญญาณในระบบแบบไม่ต่อเนื่อง

3) ตัวหน่วง (Delay Element) ใช้สำหรับหน่วงสัญญาณแสดงดังรูปที่ 2.16



รูปที่ 2.16 รูปแสดงแผนผังโครงสร้างของตัวหน่วงในระบบไม่ต่อเนื่อง

4) ตัวล้ำหน้า (Advance Element) ใช้สำหรับกำหนดค่าล่วงหน้าให้กับระบบดังรูปที่ 2.17



รูปที่ 2.17 รูปแสดงแผนผังโครงสร้างของตัวล้ำหน้าในระบบไม่ต่อเนื่อง

2.2.2.2 ชนิดของระบบไม่ต่อเนื่อง

1) ระบบแบบ Static เป็นระบบที่ไม่มีหน่วยความจำมีสมการของระบบดังสมการที่ 2.20 และสมการที่ 2.21

$$y(n) = ax(n) \tag{2.20}$$

$$y(n) = nx(n) + bx^3(n) \tag{2.21}$$

2) ระบบแบบ Dynamic เป็นระบบที่มีหน่วยความจำนั่นหมายถึงเป็นระบบที่สามารถบันทึกค่าอินพุตหรือเอาต์พุตที่เวลาผ่านไปแล้วได้มีสมการของระบบดังสมการที่ 2.22 ถึงสมการที่ 2.24

$$y(n) = x(n) + 3x(n-1) \tag{2.22}$$

$$y(n) = \sum_{k=0}^{n} x(n-k) \tag{2.23}$$

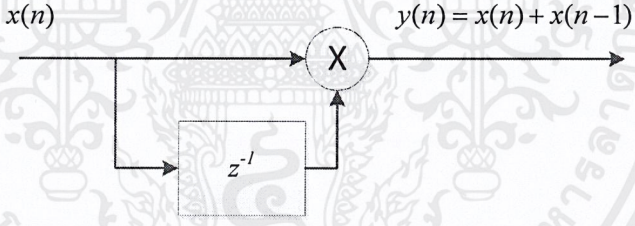
$$y(n) = \sum_{k=0}^{\infty} x(n-k) \tag{2.24}$$

3) ระบบแบบไม่แปรเปลี่ยนตามกาลเวลา (Time-invariant System) กำหนดให้ τ เป็นระบบไม่ต่อเนื่อง *x(n)* เป็นอินพุตของระบบ *y(n)* เป็นเอาต์พุตของระบบดังสมการที่ 2.25 ถ้ามีการหน่วงอินพุตเป็น *x(n-k)* ถ้าเอาต์พุตที่ได้ถูกหน่วงเป็น *y(n-k)* เช่นเดียวกับอินพุตดังสมการที่ 2.26 สามารถสรุปได้ว่าระบบไม่ต่อเนื่อง τ เป็นระบบไม่ต่อเนื่องที่แปรเปลี่ยนตามเวลา

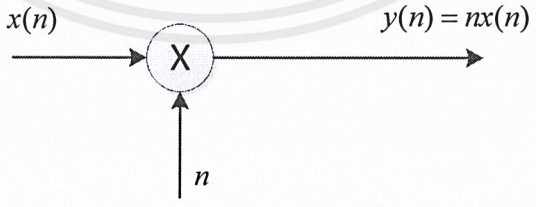
$$x(n) \xrightarrow{\tau} y(n) \tag{2.25}$$

$$x(n-k) \xrightarrow{\tau} y(n-k) \tag{2.26}$$

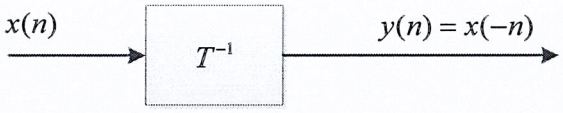
และระบบที่อินพุตถูกหน่วงไป *x(n) - x(n-k)* แต่ได้เอาต์พุตของระบบไม่เท่ากับ *y(n-k)* ระบบนั้นเป็นระบบที่แปรเปลี่ยนตามเวลา (Time-invariant System หรือ Time-varying System) รูปที่ 2.18 แสดงตัวอย่างระบบไม่ต่อเนื่องที่แปรเปลี่ยนตามเวลาและไม่แปรเปลี่ยนตามเวลา



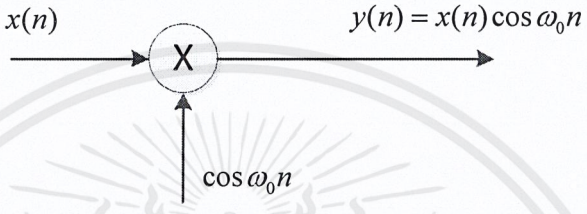
รูปที่ 2.18 (a) รูปแสดงระบบไม่ต่อเนื่องที่แปรเปลี่ยนตามเวลา



รูปที่ 2.18 (b) รูปแสดงระบบไม่ต่อเนื่องที่ไม่แปรเปลี่ยนตามเวลา



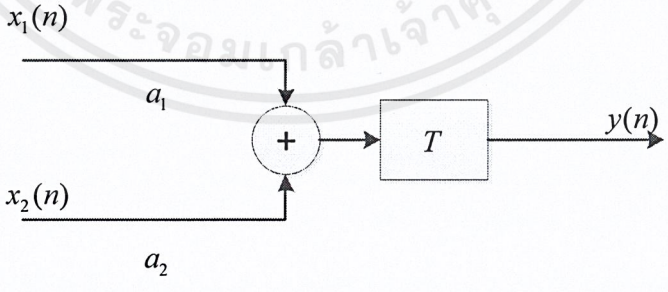
รูปที่ 2.18 (c) รูปแสดงระบบไม่ต่อเนื่องที่ไม่แปรเปลี่ยนตามเวลา



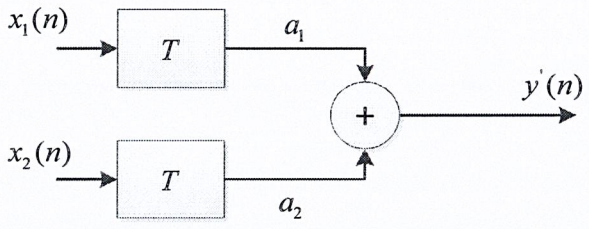
รูปที่ 2.18 (d) รูปแสดงระบบไม่ต่อเนื่องที่ไม่แปรเปลี่ยนตามเวลา

4) ระบบเชิงเส้นและไม่เชิงเส้น เป็นระบบที่ต้องมีคุณสมบัติที่สอดคล้องกับหลักการของ การทับซ้อน (Superposition) ดังสมการที่ 2.27 หรืออธิบายได้ดังรูปที่ 2.19 และระบบที่ไม่มีคุณสมบัติที่สอดคล้องกับหลักการของ Superposition จะเป็นระบบแบบไม่เชิงเส้น

$$\tau\left[a_1x_1(n)+a_2x_2(n)\right]=a_1\tau\left[x_1(n)\right]+a_2\tau\left[x_2(n)\right] \qquad (2.27)$$



รูปที่ 2.19 (a) รูปแสดงระบบไม่ต่อเนื่องแบบเชิงเส้น



รูปที่ 2.19 (b) รูปแสดงระบบไม่ต่อเนื่องแบบเชิงเส้น

5) ระบบแบบ Causal และแบบ Non-causalเป็นระบบที่สร้างได้จริงมีสมการของระบบดังสมการที่ 2.29 จะเห็นได้ว่าระบบแบบ Causal จะประกอบด้วยเทอมอินพุตปัจจุบันและอินพุตในอดีตเท่านั้น

$$y(n) = F\left[x(n), x(n-1), x(n-2), ..., x(n-k)...y(n-1), y(n-2), ..., y(n-k)\right] \quad (2.28)$$

$$y(n) = 2x(n) + 4.5x(n-2) - 3y(n-1) + 0.5y(n-2) \quad (2.29)$$

6) ระบบแบบเสถียร (Stable) และระบบแบบไม่เสถียร (Unstable) เป็นระบบที่มีเอาต์พุตขึ้นอยู่กับอินพุตจะเป็นระบบเสถียร ส่วนระบบที่มีเอาต์พุตไม่ขึ้นอยู่กับอินพุตจะเป็นระบบแบบไม่เสถียร

2.2.3 Z-transform

การแปลง z-transform ของสัญญาณที่ไม่ต่อเนื่องสามารถนิยามอยู่ในรูปของอนุกรมอนันต์ (Power series) ได้ดังสมการที่ 2.30 โดยจะเรียกว่าการแปลงแซดแบบตรง (Direct z-transform)

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) Z^{-n} \quad (2.30)$$

เมื่อ Z เป็นตัวแปรเชิงซ้อน (Complex Variable)

$x(n)$ เป็นบำดับสัญญาณ (Sequence Signal)

แต่สำหรับ Causal system การแปลงแซดของสัญญาณที่ไม่ต่อเนื่องสามารถเขียนได้ดังสมการที่ 2.31 เรียกสมการที่ 2.31 นี้ว่า One-side z-transform

$$X(z)=\sum_{n=0}^{\infty}x(n)z^{-n} \tag{2.31}$$

การแปลงแซดของสัญญาณแบบไม่ต่อเนื่องสามารถเขียนแทนได้อีกรูปแบบหนึ่งดังสมการที่ 2.32

$$X(z)=Z\{x(n)\} \tag{2.32}$$

2.2.4 วงจรกรองดิจิตอล (Digital Filter)

วงจรกรองดิจิตอลแบ่งออกได้เป็น 2 ชนิด คือ

- วงจรกรองความถี่ดิจิตอลแบบผลตอบสนองอิมพัลส์จำกัด (FIR: Finite Impulse Response Filter)
- วงจรกรองความถี่ดิจิตอลแบบผลตอบสนองอิมพัลส์ไม่จำกัด (IIR: Infinite Impulse Response Filter)

ซึ่งสามารถเปรียบเทียบกันได้ดังตารางที่ 2.1

ตารางที่ 2.1 ตารางเปรียบเทียบวงจรกรองแบบดิจิตอล

FIR	IIR
เสถียร	ไม่เสถียร
Transition band กว้าง	Transition band แคบ
Delay มาก	Delay น้อย
ผลตอบสนองเชิงเฟสเป็นเชิงเส้น	ผลตอบสนองเชิงเฟสผิดเพี้ยนสูง
สัญญาณรบกวนจากการปัดเศษน้อย	สัญญาณรบกวนจากการปัดเศษมาก
Ripple มาก	Ripple น้อย

2.3 เอฟเฟคกีตาร์

กีตาร์เอฟเฟค (Guitar Effect) หรือเครื่องช่วยกีตาร์ ในปัจจุบันนี้นอกจากกีตาร์และแอมพลิฟายแล้ว อีกอุปกรณ์ที่ได้รับความนิยมในการเล่นคือ กีตาร์เอฟเฟค ซึ่งจะช่วยเปลี่ยนเสียงจากกีต้าร์ให้เป็นเสียงต่างๆ ตามที่ตัวกีตาร์เอฟเฟคนั้นๆ ได้ถูกออกแบบมา ลักษณะการทำงานของแต่ละเอฟเฟคแสดงได้ดังนี้

2.3.1 Overdrive Effect

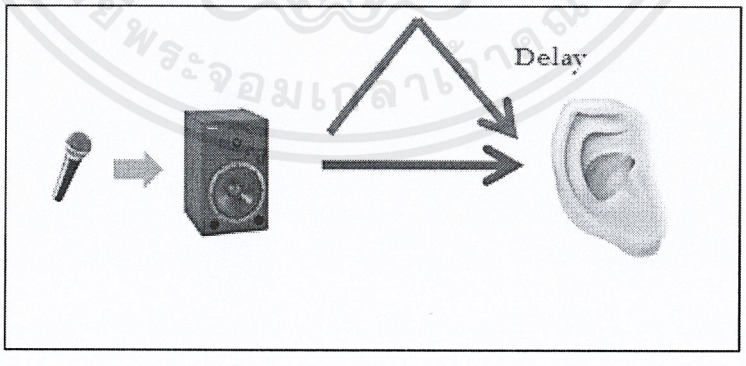
ลักษณะการทำงานของ Overdrive Effect จะเป็นการเพิ่มขนาดของอัตราการขยายของสัญญาณเสียงให้มีขนาดที่สูงขึ้นจนเกินขนาดของ Threshold ที่กำหนดไว้ โดยสัญญาณเสียงที่เข้าไปจะถูกขยายตามขนาดของอัตราการขยายของสัญญาณเสียง จากนั้นสัญญาณเอาต์พุตที่ได้จะมีลักษณะที่ถูกตัดยอดสัญญาณ (clipping) ทำให้เสียงที่ได้มีลักษณะแตกซ่า

2.3.2 Distortion Effect

Distortion Effect คือเสียงเสียงที่มีลักษณะแตกซ่าเหมือนกับ Overdrive Effect แต่เสียงจะมีความแตกซ่าที่ละเอียดกว่าและมีรูปแบบของสมการที่ต่างกัน

2.3.3 Delay Effect

Delay Effect เป็นเอฟเฟคที่ลักษณะเสียงของสัญญาณเสียงเอาต์พุตที่มีการหน่วงเวลาของสัญญาณอินพุตที่เข้าไปแสดงได้ดังรูปที่ 2.20



รูปที่ 2.20 รูปแสดงลักษณะการเดินของเสียงที่ถูกหน่วงเวลา

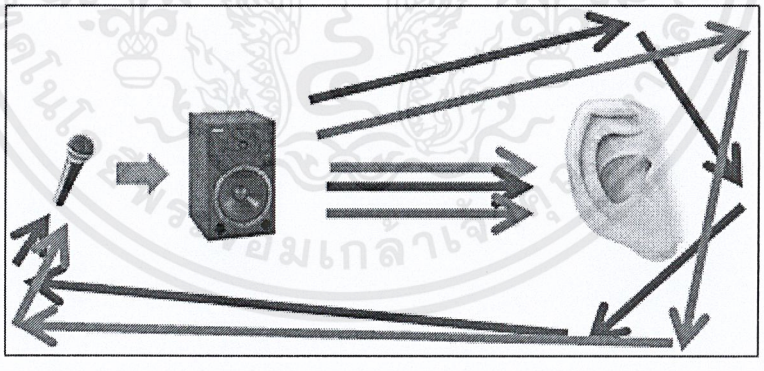
อีกทั้ง Delay Effect ยังเป็นพื้นฐานของการออกแบบเอฟเฟคชนิดอื่นๆ อาทิเช่น Reverberation, Chorus และ Flanger Effect ซึ่งล้วนแต่มีพื้นฐานมาจากการหน่วงเวลาทั้งสิ้น โดยการหน่วงเวลาของแต่ละเอฟเฟคนั้นจะอยู่ในหน่วยของมิลลิวินาที (millisecond) ซึ่งสามารถแสดงได้ดังตามตารางที่ 2.2

ตารางที่ 2.2 ตารางแสดงค่าเวลาในการหน่วงของเอฟเฟคชนิดต่างๆ

Delay Range (*ms*)	Modulation	Effect Name
0-15	Sinusoidal	Flanger
น้อยกว่า 12	-	Reverb
มากกว่า 50	-	Delay

2.3.4 Reverb Effect

Reverb Effect เป็นที่รู้จักกันดีในอีกด้านหนึ่งคือ ผลกระทบทางการสะท้อน ที่พบเจอได้ตามชีวิตประจำวัน เช่น การเกิดเสียงก้องในห้องประชุม เป็นต้น โดยเสียง Reverb Effect นี้เกิดจากการเดินทางของสัญญาณเสียงที่มีความแตกต่างกันไปของคลื่นเสียงที่เกิดการสะท้อนกลับมายังแหล่งกำเนิดสัญญาณเสียง สามารถแสดงได้ดังรูป 2.21



รูปที่ 2.21 รูปแสดงการเดินทางของสัญญาณเสียงที่เกิดการสะท้อนกลับไปยังแหล่งกำเนิดเสียง

2.3.5 Flanger Effect

Flanger Effect เป็นเอฟเฟคที่มีการรวมสัญญาณเสียงต้นฉบับกับสัญญาณเสียงที่ถูกรบกวนด้วยสัญญาณ sine wave ความถี่ต่ำ เสียงที่ได้จะให้ความรู้สึกวิงเวียน ซึ่ง Flanger Effect นี้จะนำมาใช้กับเอฟเฟคอื่นๆ เพื่อเพิ่มสีสันในการเล่นดนตรี

2.3.6 Wah-wah Effect

Wah-wah Effect เป็นเอฟเฟคที่เกิดจากเสียงสะท้อนของตัวโน๊ตที่ขยายออกจนเกิดเสียงที่ดังขึ้นคล้ายคำว่า "วา"เสียงเอฟเฟคที่ได้สามารถนำมาใช้กับเสียงเอฟเฟคอื่นๆ ซึ่งนำมาใช้ในการเพิ่มความบันเทิงในการเล่นดนตรี

2.4 MATLAB

MATLAB เป็นคำที่ย่อมาจาก Matrix Laboratory เป็นภาษาคอมพิวเตอร์ชั้นสูง มีความสามารถในการดำเนินงานทางเทคนิคที่ประกอบด้วยการคำนวณเชิงตัวเลข กราฟิกที่ซับซ้อนและการจำลองแบบเพื่อให้มองเห็นภาพได้ง่ายและชัดเจน โปรแกรม MATLAB ได้เขียนขึ้นเพื่อคำนวณทาง matrix หรือเป็น matrix software ที่พัฒนามาจาก LINKPACK และ EISPACK ประเภทของการใช้งานจะเป็นดังนี้

- ทางด้านคณิตศาสตร์ลารคำนวณ (Math and computation)
- การพัฒนาอัลกอริทึม (Algorithm development)
- การเก็บข้อมูล (Data acquisition)
- การสร้างแบบจำลอง (Modeling, simulation, and visualization)
- การวิเคราะห์ข้อมูล (Data analysis, exploration, and visualization)
- ทางด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ (Scientific and engineering graphics)
- การพัฒนาโปรแกรมประยุกต์ที่มีการสร้างส่วนติดต่อกับผู้ใช้เป็นทางด้านกราฟิก (Application development)

ลักษณะของโปรแกรม MATLAB มีการเพิ่มในส่วนของการทำงานแบบพิเศษหรือกล่องเครื่องมือที่ใช้ในการหาคำตอบ เรียกว่าทูลบอกซ์ (Toolboxes) ที่เหมาะกับงานในแต่ละสาขา เช่น การประมวลผลสัญญาณ (Signal processing toolbox) การประมวลผลสัญญาณภาพ (Image

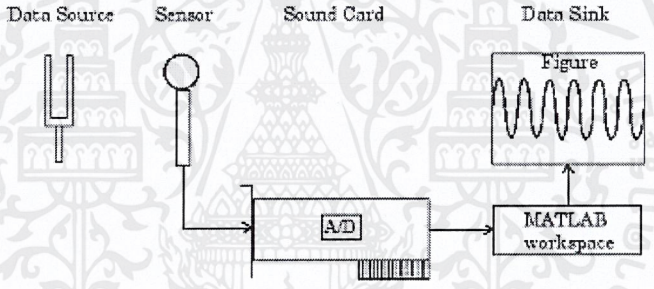
processing toolbox) ระบบควบคุม (Control system toolbox) การติดต่อสื่อสาร (Communication toolbox) สถิติ (Statistics toolbox) เป็นต้น

2.4.1 Data Acquisition Toolbox

เป็นทูลบอกซ์ (Toolboxes) ที่ใช้ในการสร้างกอินเตอร์เฟสติดต่อกับส่วนของฮาร์ดแวร์ เพื่อใช้ในการเก็บข้อมูลของสัญญาณอนาลอก โดยการสร้างอินเตอร์เฟสติดต่อกับซาวด์การ์ดของเครื่องคอมพิวเตอร์มีการทำงานงานประกอบไปด้วย

2.4.1.1 Analog Input

เป็นการสร้างอินเตอร์เฟสติดต่อกับส่วนของฮาร์ดแวร์ เพื่อจำลองการทำงานของซาวด์การ์ดให้มีการทำงานเป็นวงจรแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล ซึ่งมีลักษณะทำงานแสดงตามรูปที่ 2.22



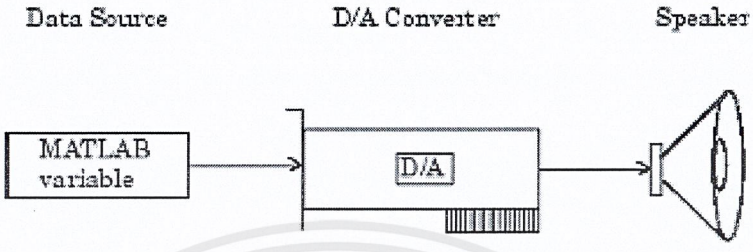
รูปที่ 2.22 รูปแสดงลักษณะการทำงานของ analog input

จากรูปที่ 2.22 จะเห็นได้ว่ามีการการรับเอาสัญญาณเสียงที่เป็นสัญญาณอนาลอกเข้ามายังโปรแกรม MATLAB ได้โดยการสร้างอินเตอร์เฟสติดต่อกับซาวด์การ์ดของคอมพิวเตอร์ ซึ่งข้อมูลที่รับเข้ามาได้จะอยู่ในรูปของเวคเตอร์ที่ค่าของข้อมูลขึ้นอยู่กับตำแหน่งของ sample rate นั้นๆ ซึ่งสามารถนำข้อมูลที่เก็บได้มาใช้ในการประมวลผลสัญญาณดิจิตอลได้

2.4.1.2 Analog Output

เป็นการสร้างอินเตอร์เฟสติดต่อกับส่วนของฮาร์ดแวร์ เช่นเดียวกับ analog input แต่ในส่วนของ analog output จะเป็นการกำหนดการทำงานให้กับซาวด์การ์ดให้มี

การทำงานเป็นวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก โดยมีลักษณะการทำงานแสดงตามรูปที่ 2.23

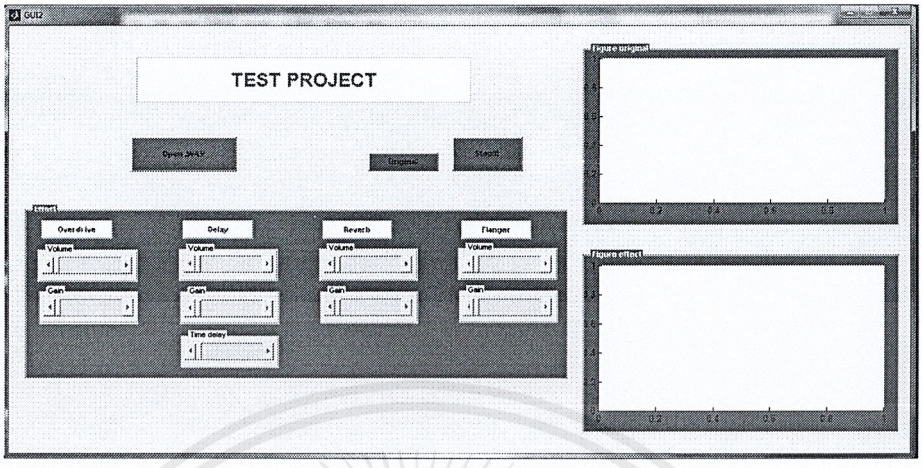


รูปที่ 2.23 รูปแสดงลักษณะการทำงานของ analog output

จากรูปที่ 2.23 จะเห็นได้ว่ามีการสร้างการอินเตอร์เฟสติดต่อกับส่วนซาวด์การ์ดเพื่อจำลองให้มีลักษณะการทำงานเป็นวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก โดยการนำเอาข้อมูลที่ได้จากการประมวลผลสัญญาณดิจิตอลออกในรูปของสัญญาณอนาลอก ทำให้สัญญาณเสียงขึ้นตามค่าข้อมูลที่ได้จากการประมวลผลสัญญาณดิจิตอล

2.4.2 GUI (Graphic User Interface)

เป็นส่วนการทำงานของโปรแกรม MATLAB ในส่วนของการพัฒนาโปรแกรมประยุกต์ที่มีการสร้างส่วนติดต่อกับผู้ใช้เป็นทางด้านกราฟิก (Application development) ซึ่งเป็นการออกแบบส่วนติดต่อกับผู้ใช้งาน เพื่อให้ผู้งานเกิดความสะดวกสบายในการใช้งาน โดยสามารถกำหนดรายละเอียดของการใช้งานได้ตามต้องการ ซึ่งสามารถแสดงได้ตามรูปที่ 2.24



รูปที่ 2.24 รูปแสดงส่วนติดต่อกับผู้ใช้งาน (GUI)

2.5 การวิเคราะห์สเปคตรัม (Spectrum Analysis)

2.5.1 คลื่นเสียง

คลื่นเสียงเป็นคลื่นกลชนิดคลื่นตามยาว (longitudinal waves) ซึ่งสามารถเคลื่อนผ่านตัวกลางที่เป็นทั้งของแข็ง ของเหลว และแก๊ส

เสียงเกิดจากการสั่นของวัตถุ การทำให้วัตถุสั่นด้วยวิธีการ ดีด สี ตี และเป่า เมื่อแหล่งกำเนิดเสียงเกิดการสั่นจะทำให้โมเลกุลอากาศสั่นตามไปด้วยความถี่เท่ากับการสั่นของแหล่งกำเนิดเสียงเกิดเป็นช่วงอัดช่วงยายของโมเลกุลของอากาศ ซึ่งพลังงานของการสั่นจะแผ่ออกไปรอบๆ แหล่งกำเนิดเสียงตรงกลางส่วนอัดและตรงกลางส่วนขยายโมเลกุลอากาศจะไม่มีการเคลื่อนที่ (การกระจัดเป็นศูนย์) แต่ตรงกลางส่วนอัดความดันอากาศจะมากและตรงกลางส่วนขยายความดันอากาศจะน้อยมาก ดังนั้นคลื่นเสียงจึงเป็นคลื่นตามยาวเพราะโมเลกุลของอากาศจะสั่นในทิศเดียวกับทิศที่เสียงเคลื่อนที่ไปความดังของเสียงจะขึ้นอยู่กับช่วงกว้างของการสั่น (แอมพลิจูด) ถ้าแอมพลิจูดมากเสียงจะดังมาก การเปลี่ยนความดันอากาศนี้สามารถเคลื่อนที่ไปข้างหน้าจนถึงหูของผู้ฟังทำให้ได้ยินเสียง

เมื่อคลื่นเสียงเดินทางเข้ากระทบเยื่อแก้วหู การเปลี่ยนความดันอากาศนี้สามารถเคลื่อนที่ไปข้างหน้าถึงหูของผู้ฟังทำให้เยื่อแก้วหูของผู้ฟังสั่น การสั่นแปรเป็นกระแสประสาทส่งไปยังสมอง

2.5.2 การวิเคราะห์สเปกตรัมของเสียง

เริ่มด้วยการรับสัญญาณเสียงโดยใช้ไมโครโฟนที่เชื่อมต่อกับคอมพิวเตอร์สัญญาณเสียงที่ได้จะเป็นสัญญาณอนาล็อก (Analog Signal) จากนั้นจะต้องทำการแปลงสัญญาณที่ได้ให้เป็นสัญญาณดิจิตอลโดยใช้เครื่องเปลี่ยนสัญญาณอนาล็อกเป็oสัญญาณดิจิตอล (Analog to Digital Converter)

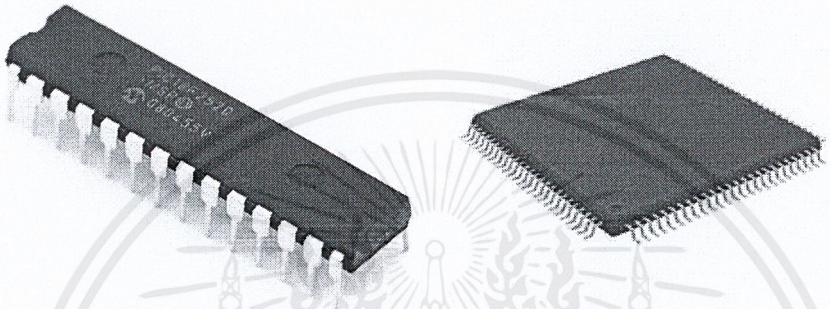
แอมพลิจูดของคลื่นเสียงจะมีขนาดต่างกันในแต่ละช่วงเวลา เรานำความสัมพันธ์ของแอมพลิจูดของคลื่นเสียงกับเวลามาวาดกราฟบนเครื่องคอมพิวเตอร์ จากนั้นเราทำการหาคาบของคลื่นเสียงออกมา

การหาสมการของคลื่นเสียงในคลื่นรูปไซน์ ที่เกิดจากการรวมตัวกันของคลื่นเสียงที่มีแอมพลิจูดเหมือนกันแต่ความถี่ต่างกัน สมการนี้สามารถวิเคราะห์โดยใช้เทคนิคที่เรียกว่าการแปลงฟูเรียร์แบบเร็ว (Fast Fourier Transform, FFT)) ซึ่งการแปลงฟูเรียร์แบบเร็วจะทำการเปลี่ยนข้อมูลจากโดเมนเชิงเวลา (Time domain) เป็นโดเมนเชิงความถี่ (Frequency domain) จากนั้นใช้การแปลงฟูเรียร์แบบเร็วกับสมการของคลื่นเสียงจะสามารถแยกความถี่ของคลื่นเสียงออกมาได้

2.6 ไมโครคอนโทรลเลอร์ (Microcontroller)

ไมโครคอนโทรลเลอร์ คือ อุปกรณ์ที่มีลักษณะเป็นไอซีทั่วๆ ไปหรือชิพ ซึ่งมีหน้าที่ในการควบคุมอุปกรณ์อิเล็กทรอนิกส์ต่างๆ ให้ทำงานตามคำสั่งที่เราได้โปรแกรมเข้าไป สิ่งที่เราต้องทราบก่อนในการใช้งานไมโครคอนโทรลเลอร์นั้นจะประกอบไปด้วย 3 ส่วน คือ ส่วนอินพุต ส่วนประมวลผล และส่วนเอาต์พุต กล่าวคือ เมื่อต้องการป้อนอินพุตเข้าไมโครคอนโทรลเลอร์เป็นค่าๆ หนึ่ง แล้วจะทำการโปรแกรมไมโครคอนโทรลเลอร์ให้มีการประมวลผลการทำงานอย่างไรเพื่อให้ได้ค่าของเอาต์พุตออกมาที่มีค่าตามที่เราต้องการ ในปัจจุบันมีไมโครคอนโทรลเลอร์อยู่มากจากหลายบริษัทที่ผลิตขึ้นมา แต่ละบริษัทได้ทำการแบ่งรุ่น แบ่งเบอร์ เพื่อตอบสนองตามความต้องการของ

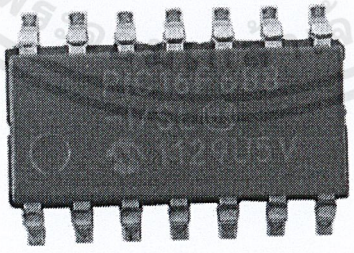
ผู้ออกแบบระบบวงจรที่นำไปใช้ในงานที่แตกต่างกันออกไป สิ่งที่เป็นตัวชี้ว่าไมโครคอนโทรลเลอร์ตัวใดเหมาะสมกับงานแบบไหนคือ ความเร็วของสัญญาณนาฬิกา และจำนวนของขาในการรับอินพุตและส่งเอาต์พุต นอกจากนี้สิ่งที่อาจจะเป็นปัจจัยในการเลือกคือ ราคา และภาษาที่ใช้ในการเขียนโปรแกรม



รูปที่ 2.25 รูปแสดงไมโครคอนโทรลเลอร์

2.6.1 ไมโครคอนโทรลเลอร์ PIC

ไมโครคอนโทรลเลอร์ตระกูลนี้ มีจุดเด่นคือได้ทาการรวมเอาทุกอย่างไว้ในชิพเพียงตัวเดียวโดยไม่ต้องต่ออุปกรณ์ใดๆ เพิ่มเติม ผลที่ตามมาก็คือแผ่นวงจรจะมีขนาดเล็ก และอุปกรณ์ที่ใช้นั้นไม่มาก บางงานอาจจะใช้แค่ PIC เพียงตัวเดียวโดยไม่ต้องใช้ชิพอื่นมาเพิ่ม ซึ่งนี่คือคุณสมบัติพิเศษของไมโครคอนโทรลเลอร์ตระกูล PIC



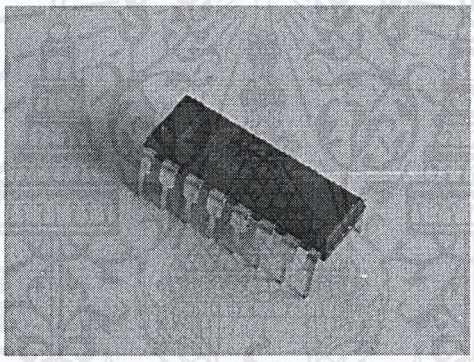
รูปที่ 2.26 รูปแสดงไมโครคอนโทรลเลอร์ PIC16F688

2.6.2 MPLAB

MPLAB คือ โปรแกรมซอฟแวร์ที่ใช้ในการสร้างโปรแกรมคำสั่งและเชื่อมโยงคำสั่งจากคนไปยังไมโครคอนโทรลเลอร์ตระกูล PIC เพื่อให้ไมโครคอนโทรลเลอร์ทำงานตามคาสั่งที่ถูกโปรแกรมไว้ โดย MPLAB จะประกอบด้วยส่วนของ MPLAB IDE ซอฟท์แวร์ที่ใช้ในการเขียนโปรแกรมคาสั่ง และ MPLAB ICD คือ อุปกรณ์ที่ใช้ในการเชื่อมต่อระหว่างคอมพิวเตอร์กับบอร์ดไมโครคอนโทรลเลอร์

2.6.3 MAX232

MAX232 เป็นไอซีที่แปลงระดับสัญญาณของ RS-232 มาเป็นระดับ TTL และในทำนองเดียวกันก็แปลงระดับสัญญาณ TTL ไปเป็นระดับสัญญาณ RS-232



รูปที่ 2.27 รูปแสดง IC MAX232

2.6.4 RS232 (Recommended Standard 232)

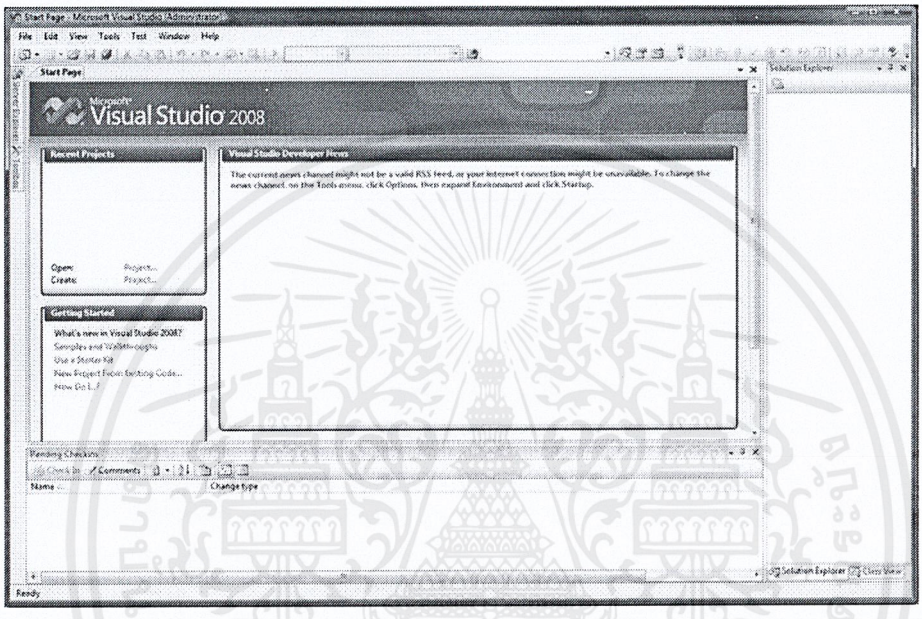
RS232 คือ มาตรฐานการเชื่อมต่อข้อมูลแบบ Serial ใช้เพื่อเพิ่มระยะทางในการส่งข้อมูล แบบ Serial ให้สามารถส่งได้ระยะทางที่มากขึ้น โดยมีการเปลี่ยนระดับแรงดัน ของ Logic จากเดิมที่จะอยู่ในช่วง 0-5 V หรือ 0-3.3 V เป็นช่วง -15 ถึง 15 V

2.7 ไมโครซอฟท์ วิชวลสตูดิโอ (Microsoft Visual Studio)

ไมโครซอฟท์ วิชวลสตูดิโอ (Microsoft Visual Studio) คือ Integrated Development Environment พัฒนาขึ้นโดยไมโครซอฟท์ ซึ่งเป็นเครื่องมือที่ช่วยนักพัฒนา

ซอฟต์แวร์พัฒนาโปรแกรมคอมพิวเตอร์ เว็บไซต์ เว็บแอปพลิเคชัน และ เว็บเซอร์วิส ระบบที่รองรับการทำงานนั้นมีไมโครซอฟท์ วินโดวส์ พ็อคเกตพีซี Smartphone และ เว็บเบราว์เซอร์

ในปัจจุบัน วิชวลสตูดิโอนั้นสามารถใช้ภาษาโปรแกรมที่เป็นภาษาดอตเน็ต (.net) ในโปรแกรมเดียวกัน เช่นVB.NET C++ C# J# เป็นต้น

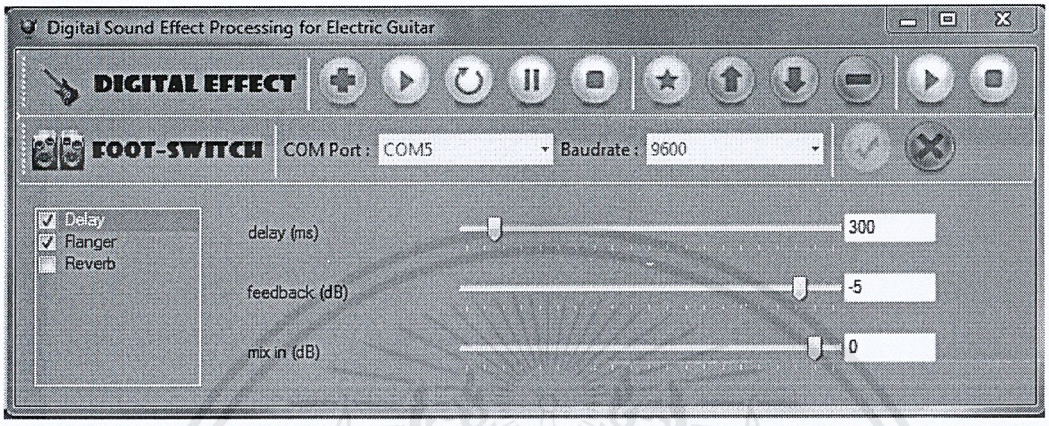


รูปที่ 2.28 รูปแสดงโปรแกรม Microsoft Visual Studio

2.7.1 Windows Form Application

การเขียนโปรแกรมบน Windows Form Application ด้วย .NET Framework ใน Application บน Visual Studio นั้น Windows Form ถือเป็น Project พื้นฐานที่สามารถพัฒนาโปรแกรมที่ทำงานบน Windows ได้ง่ายและรวดเร็วที่สุดก็ว่าได้ เพราะเป็นการออกแบบรูปแบบ GUI การใส่ Control หรือกำหนด Event ต่าง ๆ ก็สามารถสร้างเหตุการณ์ต่าง ๆ ได้จาก Properties ของ Control และเค้าโครงการเขียนนั้นก็มีพื้นฐานมาจากภาษา Visual Basic 6.0 ซึ่งจุดนี้เอง นักโปรแกรมเมอร์ที่พัฒนาโปรแกรมด้วย VB6 มาก่อนหน้านี้ก็สามารถต่อยอดการเขียนได้อย่างง่ายดาย รวมทั้งรูปแบบคำสั่งที่เป็นภาษา (VB.NET) ก็ไม่ได้ยากอะไรมากมาย ซึ่งใน .NET Framework นี้เราสามารถพัฒนาโปรแกรมให้มีความสามารถและการทำงานได้หลากหลาย และยัง

สามารถเขียนเพื่อใช้งานร่วมกับ Application อื่น ๆ ที่พัฒนาด้วย .NET Framework ได้เช่นเดียวกัน



รูปที่ 2.29 รูปแสดง Windows Form Application

จากรูปที่ 2.29 จะเห็นได้ว่าในส่วนของ Windows Form Application จะเป็นการออกแบบในส่วนของการใช้งานหรือ GUI เพื่อเป็นการอำนวยความสะดวกในการใช้งานของผู้ใช้งาน ซึ่งในโปรแกรมและซอฟแวร์คอมพิวเตอร์ทั่วไปจะมีการออกแบบในส่วนของ GUI ที่มีรูปแบบแตกต่างกันออกไปตามรูปแบบการทำงานของโปรแกรมหรือซอฟแวร์คอมพิวเตอร์นั่นๆ

บทที่ 3

การออกแบบและการจัดทำโครงงาน

3.1 การออกแบบเอฟเฟคสำหรับกีตาร์ไฟฟ้า

สำหรับการออกแบบเอฟเฟคสำหรับกีตาร์ไฟฟ้านั้น จะทำการออกแบบเอฟเฟคดังนี้คือ Overdrive Effect, Distortion Effect, Delay Effect, Flanger Effect และ Reverb Effect ซึ่งมีหลักการออกแบบและผังโครงสร้างการทำงานของแต่ระบบเป็นดังนี้

3.1.1 Overdrive Effect

Overdrive Effect คือเสียงที่มีลักษณะแตกซ่าอันเนื่องมาจากการปรับแอมพลิจูดที่มากเกินไปทำให้สัญญาณเสียงที่ได้มีลักษณะเป็นสัญญาณที่ถูกตัดยอดสัญญาณแบบ hard-clipping ต้นกำเนิดเสียงของ Overdrive Effect เกิดขึ้นเมื่อ **จอห์นนี อัลเลน เฮนดริกซ์ [11]** นักกีตาร์ นักร้อง นักแต่งเพลงชาวอเมริกัน ได้คิดค้นเทคนิดเสียงที่แปลกใหม่โดยการปรับอัตราขยายของเสียงที่ออกมาจากลำโพงมีเสียงแตก

รูปที่ 3.1 รูปแสดงแผนผังโครงสร้างของการออกแบบ Overdrive Effect

แสดงสมการของOverdrive Effect ซึ่งแบ่งช่วงของการตัดยอดสัญญาณ 3 ช่วงได้ดังนี้

$$f(x)=\begin{cases} 2x & for \quad 0 \le x < 1/3 \\ 3-(2-3x)^2/3 & for \quad 1/3 \le x < 2/3 \\ 1 & for \quad 2/3 \le x < 1 \end{cases} \tag{3.1}$$

จากสมการที่ 3.1 เป็นการกำหนดช่วงของการตัดยอดสัญญาณ

3.1.2 Distortion Effect

Distortion Effect คือเสียงเสียงที่มีลักษณะแตกซ่าเช่นเดียวกับOverdrive Effect แต่เสียงเอาต์พุตที่ได้จะมีความแตกซ่าที่มีความละเอียดของสัญญาณเอาต์พุตมากกว่าและมีสมการการออกแบบที่ต่างกันโดย Distortion Effect จะมีการตัดยอดสัญญาณแบบเอกซ์โพเนนเชียล

รูปที่ 3.2 รูปแสดงแผนผังโครงสร้างของการออกแบบ Distortion Effect

แสดงสมการของDistortion Effect ได้ดังนี้

$$f(x)=\frac{x}{|x|}(1-e^{\alpha x^2/|x|}) \tag{3.2}$$

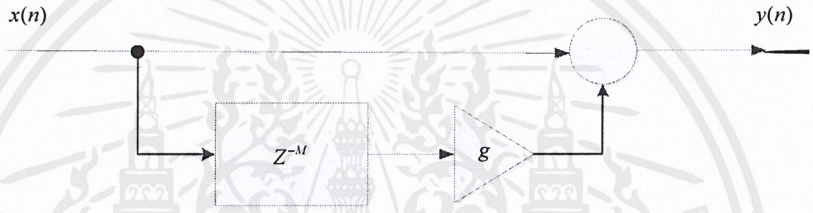
จากสมการที่ 3.2 เป็นการกำหนดช่วงของการตัดยอดสัญญาณ

3.1.3 Delay Effect

Delay Effect มีพื้นฐานในการออกแบบจากการสังเกตระยะเวลาที่ใช้ในการเดินทางของเสียงที่มีระยะเวลาในการเดินทางไปยังผู้ฟังที่แตกต่างกันออกไป ซึ่งเกิดจากการสะท้อนของเสียง

ก่อนจะเดินทางไปยังผู้ฟังทำให้เสียงที่ผู้ฟังได้ยินนั้นเกิดความล่าช้ากว่าเสียงที่แหล่งกำเนิดเสียงได้ส่งออกมา

ซึ่งในการดนตรีนั้นเสียงที่เกิดความล่าช้าหรือเกิดดีเลย์นั้นสามารถนำเอามาปรับใช้ในการเพิ่มอรรถะรสในการรับฟังของผู้ใช้งาน เป็นการเพิ่มลูกเล่นให้กับการเล่นดนตรีได้อีกแบบหนึ่ง โดยการนำเอาเสียงที่เกิดความล่าช้าหรือเกิด Delay มาผสมให้เกิดเสียงที่แปลกใหม่ในการเล่นดนตรีหรือเครื่องดนตรีทำให้เกิดความไพเราะของเสียงมากยิ่งขึ้น ซึ่งการออกแบบเสียง Delay Effect นั้นมีแผนผังโครงสร้างของการทำงานแสดงได้ดังรูป 3.3



รูปที่ 3.3 รูปแสดงแผนผังโครงสร้างของการออกแบบ Delay Effect

จากรูปที่ 3.3 รูปแสดงแผนผังโครงสร้างของการออกแบบ Delay Effect สามารถเขียน Difference equationของ Delay Effect ได้เป็นดังนี้

$$y(n) = x(n) + gx(n-M) \tag{3.3}$$

จากสมการที่ 3.3 สามารถเขียนเป็น Transfer Function ได้เป็น

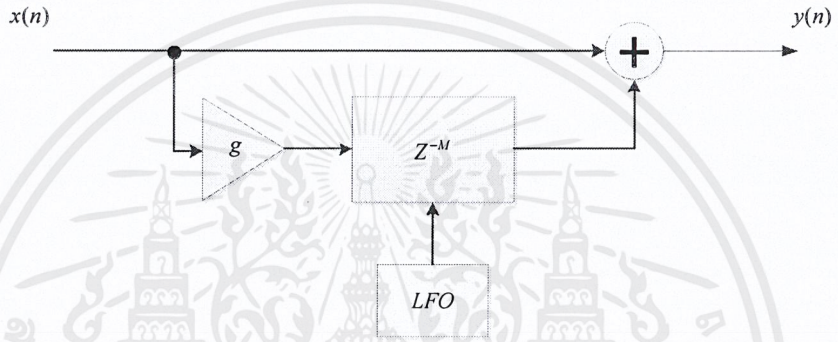
$$H(z) = 1 + gz^{-M} \tag{3.4}$$

โดยที่ g คือ ค่าของ gain ซึ่ง $|g| < 1$

M คือ ค่าของการหน่วงเวลา

3.1.4 Flanger Effect

Flanger Effectเป็นเอฟเฟคที่มีลักษณะของสัญญาณเสียงที่จะรู้สึก วิงเวียนชวนปวดหัว โดยมีหลักการทำงานคล้ายกับ Delay Effect แต่ในที่นี้ในส่วนของ Flanger Effect จะมีการใช้สัญญาณ sinewave ความถี่ต่ำในการกวนสัญญาณอินพุตที่มีหน่วงเวลาออกไปโดยค่าเวลาหน่วงเวลามาตรฐานของ effect จะอยู่ที่ มากกว่า 3 ms ซึ่งสามารถเขียนแผนผังโครงสร้างของระบบได้ดังรูป 3.5



รูปที่ 3.4 รูปแสดงแผนผังโครงสร้างของการออกแบบ Flanger Effect

จากรูปที่ 3.4 รูปแสดงแผนผังโครงสร้างของการออกแบบ Flanger Effect สามารถเขียนสมการ Difference equationของ Flanger Effect ได้ดังนี้

$$y(n) = x(n) + gx(n - \sin\_delay) \tag{3.5}$$

เมื่อ $\sin\_delay = sample\_delay \times \sin\_LFO$

$sample\_delay = time\_delay \times Fs$

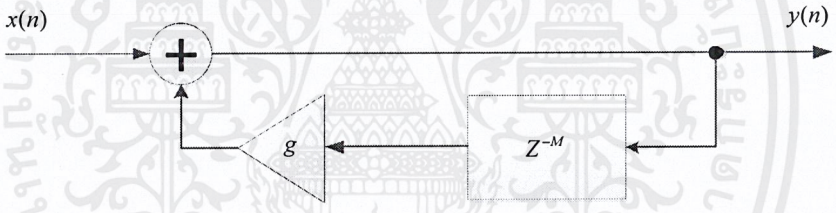
$\sin\_LFO = \sin(2\pi \times input \times (rate / Fs))$

ค่า $rate$ ค่าความถี่ของสัญญาณไซน์

3.1.5 Reverb Effect

Reverb Effect เป็นที่รู้จักกันดีในอีกด้านหนึ่งคือ ผลกระทบทางการสะท้อน ที่พบได้ตามชีวิตประจำวัน ซึ่งเป็นผลจากความล่าช้าที่เกิดจากเสียงสะท้อนในพื้นที่ทำให้เสียงต้นฉบับได้ถูกหักล้างไป โดยเกิดจากเส้นทางเดินของสัญญาณเสียงที่มีความแตกต่างกันไปของคลื่นเสียงที่เกิดการสะท้อนหลายเส้นทาง ทำให้บางส่วนเกิดการดูดซับไปทำให้เกิดการหักล้างของสัญญาณเสียงต้นฉบับ ซึ่งเสียงที่ได้จาก Reverb Effect จะเป็นเสียงที่มีลักษณะการสะท้อนตามช่วงเวลา โดยค่าเวลาการดีเลยมาตรฐานอยู่ที่น้อยกว่า 12ms

การออกแบบ Reverb Effect นั้นมีพื้นฐานการทำงานจาก Delay Effect โดยมีข้อแตกต่างกันที่ Revreb Effect จะมีการออกแบบโดยมีพื้นฐานการออกแบบมาจากระบบแบบ IIR แต่ Delay Effect มีพื้นฐานการออกแบบมาจากระบบแบบ FIR จากหลักการข้างต้นสามารถเขียนแผนผังโครงสร้างของReverb Effect ได้เป็นดังรูป3.5



รูปที่ 3.5 รูปแสดงแผนผังโครงสร้างของการออกแบบ Reverb Effect

จากรูปที่ 3.5 รูปแสดงแผนผังโครงสร้างของการออกแบบ Reverb Effect สามารถเขียนสมการ Difference equationของ Reverb Effect ได้ดังนี้

$$y(n) = x(n) + gy(n-M) \tag{3.6}$$

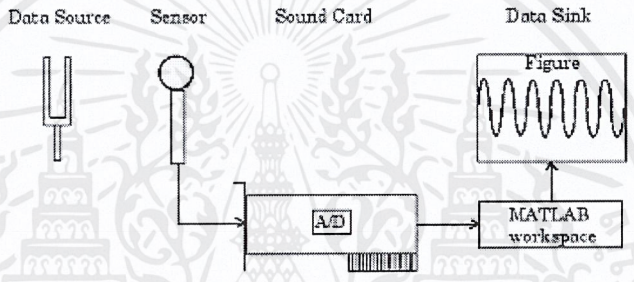
จากสมการที่ 3.6สามารถเขียนเป็น Transfer Function ของระบบได้เป็น

$$H(z) = \frac{1}{1-gz^{-M}} \tag{3.7}$$

โดย g คือ ค่าของ gain
M คือ ค่าของการหน่วงเวลา

3.2 การออกแบบการรับอินพุตแบบ Real-time

การออกแบบการรับอินพุตแบบ Real-time นั้นเป็นการสร้างอินเตอร์เฟสติดต่อกับส่วนของฮาร์ดแวร์ เพื่อกำหนดการทำงานของซาวด์การ์ดให้มีการทำงานเป็นวงจรแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล ซึ่งมีลักษณะทำงานแสดงตามรูปที่ 3.6



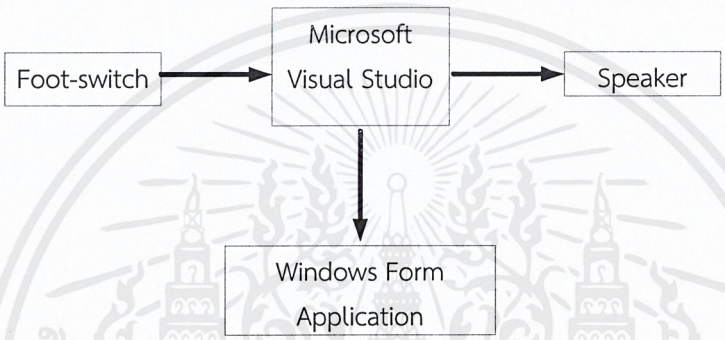
รูปที่ 3.6 รูปแสดงลักษณะการทำงานของอนาล็อกอินพุต

จากรูปที่ 3.6 รูปแสดงลักษณะการทำงานของอนาล็อกอินพุต ซึ่งจะเป็นการรับอินพุตจากแหล่งกำเนิดเสียงเข้ายังช่องไมโครโฟน โดยไมโครโฟนนั้นจะต่ออยู่กับซาวด์การ์ดโดยซาวด์การ์ดจะทำหน้าที่ในการแปลงสัญญาณอนาล็อกไปเป็นสัญญาณดิจิตอล เพื่อให้โปรแกรม MATLAB นำสัญญาณที่ได้นั้นไปประมวลผลสัญญาณต่อไป

3.3 การนำไปประยุกต์ใช้งาน

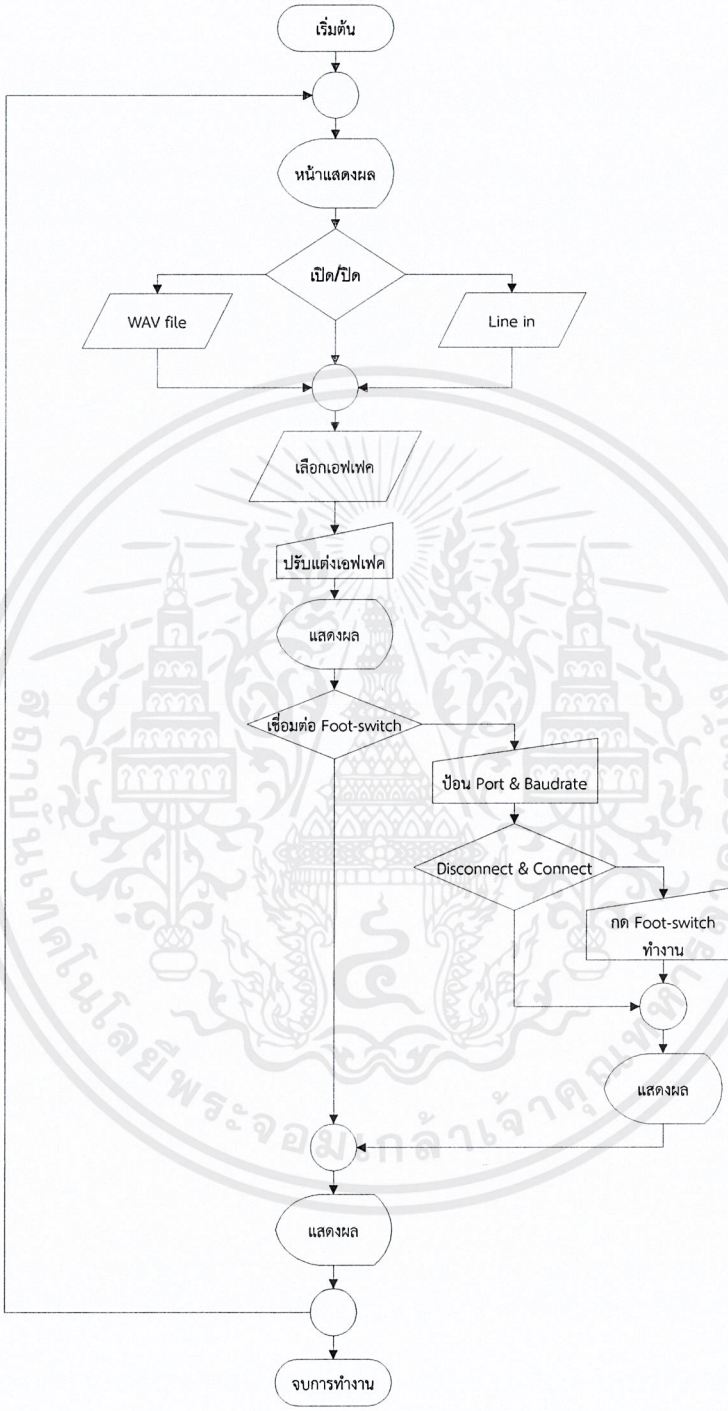
จากการที่ผู้จัดทำได้ทำการออกแบบอัลกอลิทึมของเอฟเฟคเสียงต่างๆและทำการออกแบบการรับอินพุตแบบ Real-time แล้ว เพื่อการใช้งานที่สะดวกขึ้น จึงได้ทำการออกแบบอัลกอลิทึมของเสียงเอฟเฟคต่างๆ ในข้างต้นมาประยุกต์และเขียนลงบนโปรแกรม Microsoft Visual Studio เพื่อให้ได้ซอฟแวร์คอมพิวเตอร์และไฟล์ .exe โดยใช้ภาษาที่ใช้ออกแบบโปรแกรมคือภาษา

C# ซึ่งตัวโปรแกรม Microsoft Visual Studio นั้นมีการอินเตอร์เฟสติดต่อกับอุปกรณ์จากภายนอกได้หลากหลายชนิดเช่นเดียวกับโปรแกรม MATLAB อีกทั้งในส่วนของWindows Form Applicationที่มีลักษณะการทำงานเหมือนกับระบบ Graphic User Interface ของโปรแกรม MATLAB และในส่วนของระบบ Windows Form Applicationที่ทำการออกแบบเพื่อให้ได้ไฟล์ .exe นั้น ทางผู้จัดทำได้เพิ่มส่วนของfoot-switch เพื่อการควบคุมการทำงานของเอฟเฟคต่างๆ ได้ง่ายขึ้น ซึ่งมีโครงสร้างการทำงานของระบบดังนี้



รูปที่ 3.7 รูปแสดงบล็อกไดอะแกรมของระบบ

จากรูปที่ 3.7 รูปแสดงบล็อกไดอะแกรมของระบบสามารถเขียนผังการทำงานของโปรแกรมได้ดังนี้

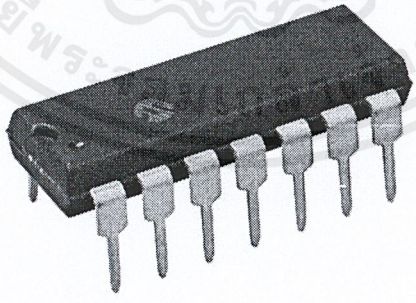


รูปที่ 3.8 รูปแสดงผังการทำงานของโปรแกรม

จากรูปที่ 3.8 รูปแสดง Flowchart การทำงานของระบบซึ่งสามารถอธิบายได้ว่า เมื่อผู้ใช้งานได้เปิดโปรแกรมจากไฟล์ .exe ขึ้นมาแล้วจะมีการแสดงในส่วนของโปรแกรมที่รอรับคำสั่งออกมา ซึ่งทางผู้ใช้งานสามารถเลือกได้ว่าจะใช้งานในส่วนของการเล่นไฟล์เสียง .wav หรือการทำงานในส่วนของ Microphone Input เมื่อเลือกส่วนของเลือกในส่วนของอินพุตแล้ว ผู้ใช้งานสามารถเลือกเสียงเอฟเฟคที่ต้องการได้โดยการกดเพิ่มเอฟเฟคต่างที่ต้องการโดยการทำงานของเอฟเฟคที่เลือกจะทำงานเรียงลำดับการทำงานตามลำดับที่ได้ทำงานเพิ่มลงไปยังโปรแกรมหลัก อีกทั้งผู้ใช้งานสามารถกดเลือกเปิด/ปิดเสียงอฟเฟคชนิดต่างๆได้จากส่วนของการควบคุมเสียงเอฟเฟคและผู้ใช้งานสามารถซื่อมต่อในส่วนของแผงควบคุมด้วยเท้าเข้ากับโปรแกรม โดยเมื่อกดสวิตช์ที่แผงควบคุมด้วยเท้าจะทำให้กระแสไฟไหลเข้าไปที่ขาอินพุตของไมโครคอนโทรลเลอร์ จากนั้นจะส่งค่าที่เรากำหนดไว้ในไมโครคอนโทรลเลอร์ ไปยังตัวแกรมที่ได้ออกแบบไว้ผ่านพอร์ท RS-232 เมื่อตัวโปรแกรมรับค่าจากไมโครคอนโทรลเลอร์แล้ว จะแสดงผลผ่านทางหน้าโปรแกรมเช่นเดียวกับการใช้เมาส์คลิกเลือกการทำงานของเอฟเฟคว่าเอฟเฟคชนิดไหนมีการทำงานและเอฟเฟคชนิดไหนที่ยังไม่มีการทำงาน จากนั้นเสียงของเอฟเฟคจะส่งออกมาทางลำโพงของเครื่องคอมพิวเตอร์

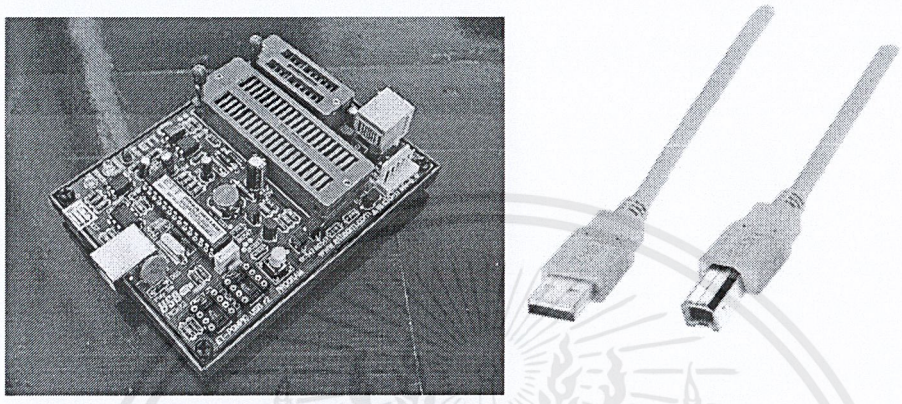
3.4 เครื่องมือที่ใช้ในการทดลอง

3.4.1 ไมโครคอนโทลเลอร์ PIC16F688



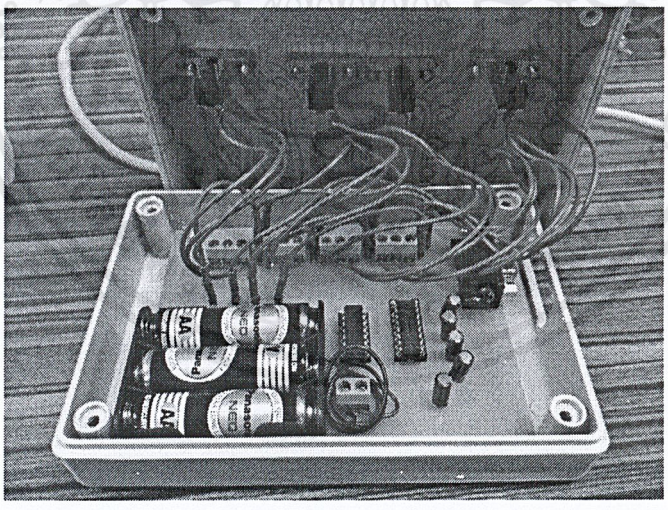
รูปที่ 3.9 รูปแสดงไมโครคอลโทลเลอร์ PIC16F688

3.4.2 เครื่องมือโปรแกรมไมโครคอนโทรลเลอร์ ET-PGM PIC USB V1 และสายต่อ USB TYPE A/B



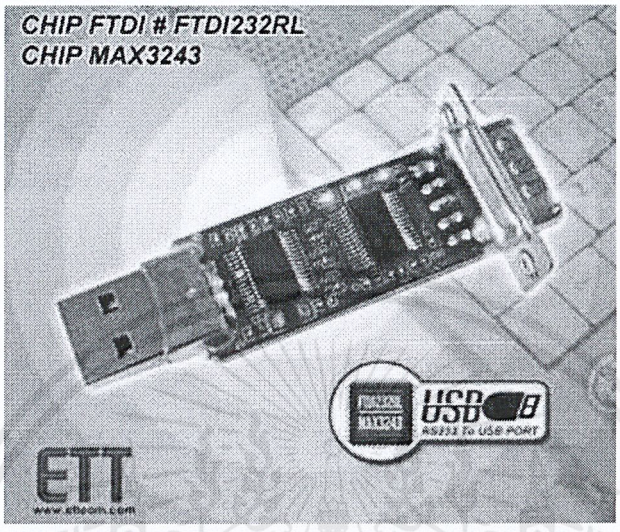
รูปที่ 3.10 รูปแสดงเครื่องมือโปรแกรม ET-PGM PIC USB V1

3.4.3 วงจรแผงควบคุมด้วยเท้า



รูปที่ 3.11 รูปแสดงวงจรแผงควบคุมด้วยท้า

3.4.4 พอร์ท RS232 ของ ET-USB/RS232 MINI



รูปที่ 3.12 รูปแสดงฟอร์ท RS232 ของET-USB/RS232 MINI

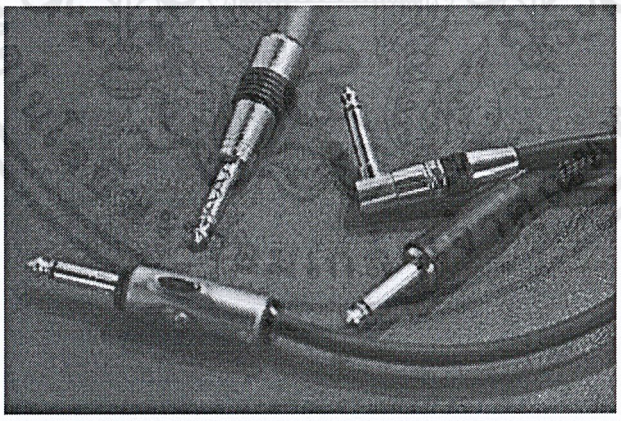
3.4.5 คอมพิวเตอร์

รูปที่ 3.13 รูปแสดงคอมพิวเตอร์

3.4.6 กีตาร์ไฟฟ้า

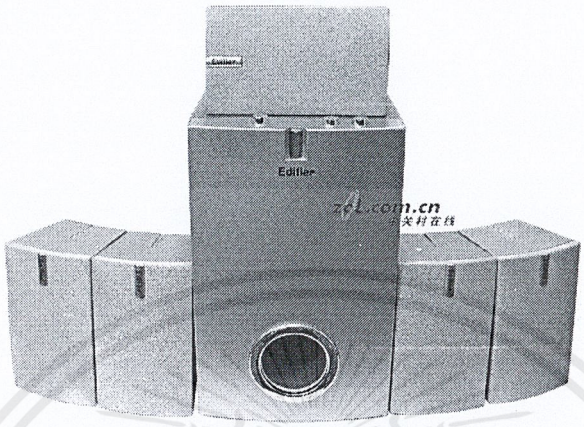
รูปที่ 3.14 รูปแสดงกีตาร์ไฟฟ้า

3.4.7 สายนำสัญญาณ (Instrument cable)



รูปที่ 3.15 รูปแสดงสายนำสัญญาณ

3.4.8 ลำโพง (Speaker)



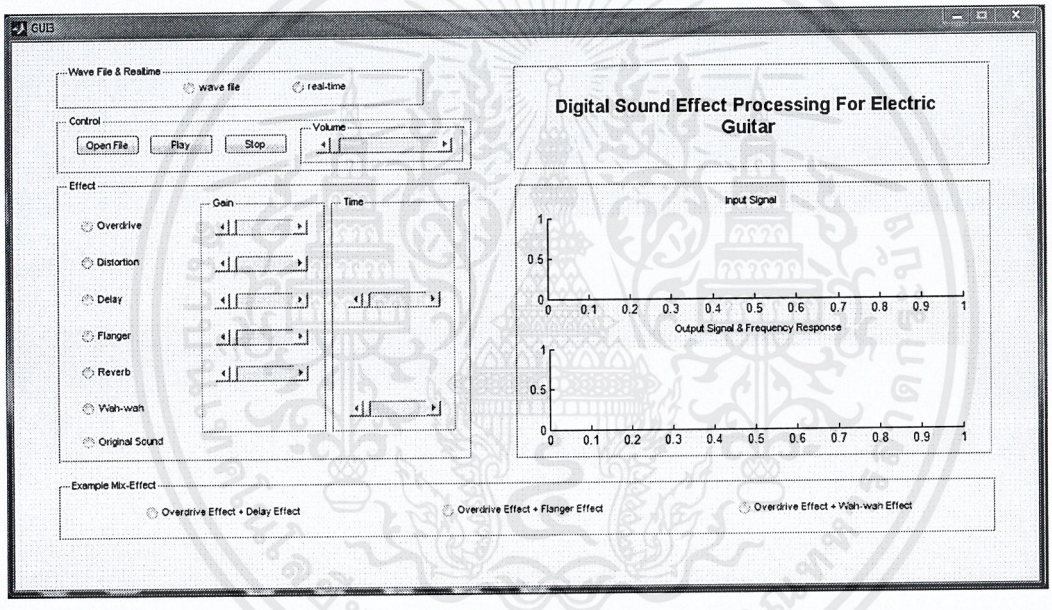
รูปที่ 3.17 รูปแสดงลำโพง (Speaker)

บทที่ 4

ผลการทดลอง

4.1 ผลการทดลองเสียงเอฟเฟคสำหรับกีตาร์ไฟฟ้า

จากการออกแบบอัลกอลิทึมของเสียงเอฟเฟคแต่ละชนิดโดยการจำลองการทำงานจากโปรแกรม MATLAB และทำการออกแบบในส่วนของการติดต่อกับผู้ใช้งานโดยการสร้าง GUI ในการอำนวยความสะดวกในการใช้งาน แสดงได้ดังรูปที่ 4.1

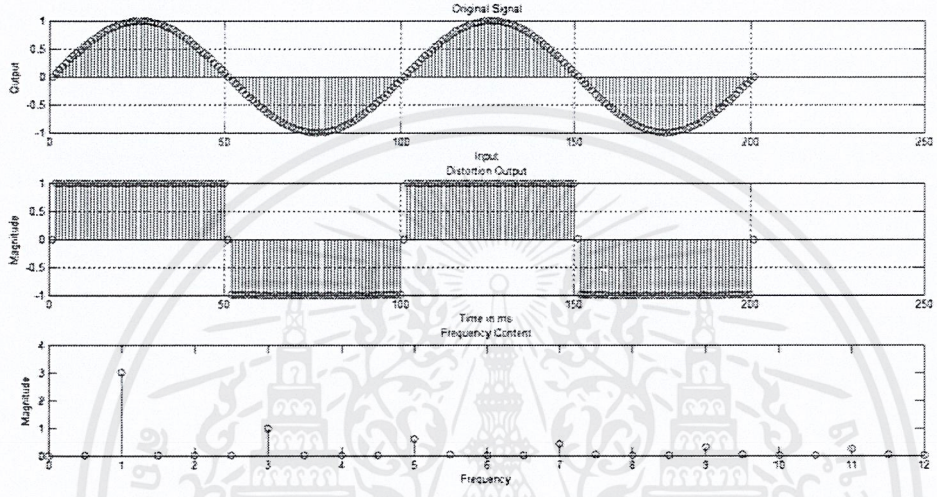


รูปที่ 4.1 รูปแสดงส่วนการทำงานของอัลกอลิทึมของเสียงเอฟเฟค

จากรูปที่ 4.1 เป็นการรวมเอาอัลกอลิทึมของแต่ละเอฟเฟคเข้าไว้ด้วยกัน โดยผู้ใช้งานสามารถกำหนดค่าพารามิเตอร์ต่างๆ ที่อัลกอลิทึมของแต่ละเสียงเอฟเฟคต้องการได้ผ่านส่วนติดต่อผู้ใช้งานหรือ GUI โดยเมื่อทำการป้อนสัญญาณอินพุตไปยังแต่ล่ะอัลกอลิทึมที่ได้ทำการออกแบบไว้จะได้ผลการทดลองเป็นดังนี้

4.1.1 ผลการทดลองของ Distortion Effect

ในการออกแบบของ Distortion Effect มีลักษณะการทำงานของระบบเป็นลักษณะของการขยายขนาดของสัญญาณอินพุตที่เข้าไปให้มีขนาดของสัญญาณที่สูงขึ้นจนเกินขนาดของ Threshold ที่กำหนดไว้ ทำให้สัญญาณเอาต์พุตที่ได้มีขนาดสัญญาณที่ถูกตัดยอดสัญญาณจากการขยายขนาดของสัญญาณอินพุต จึงทำให้ได้สัญญาณเสียงที่มีลักษณะเสียงที่แตกซ่า เมื่อทำการทดลองป้อนสัญญาณอินพุตเป็นสัญญาณ sine waveแสดงดังรูปที่ 4.2

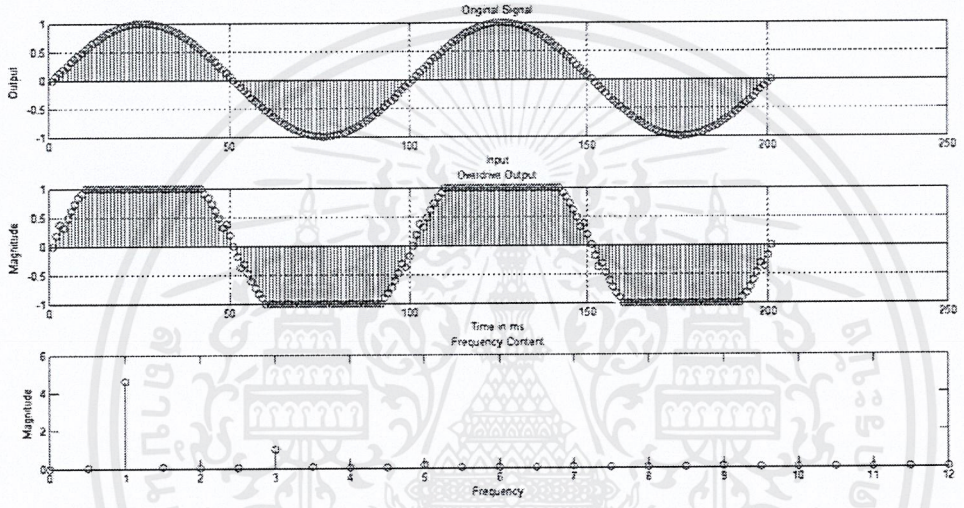


รูปที่ 4.2รูปแสดงสัญญาณอินพุตและสัญญาณเอาต์พุตของ Distortion Effect

จากรูปที่ 4.2จะเห็นได้ว่าเมื่อป้อนสัญญาณ sine wave ขนาด 200 samples เข้าไปในระบบของ Distortion Effect เมื่อสัญญาณอินพุตถูกขยายแล้วทำให้สัญญาณเอาต์พุตที่ได้เกิดการตัดยอดของสัญญาณ เมื่อทำการพล็อตกราฟในเทอมของ Frequency Response จะเห็นได้ว่าเมื่อ สัญญาณอินพุตถูกตัดยอดสัญญาณจะได้สัญญาณเอาต์พุตที่มีลักษณะของความถี่ฮาร์มอนิกที่เกิดขึ้นในตำแหน่งที่ 3, 5, 7, 9 ซึ่งทำให้สัญญาณเอาต์พุตที่ได้มีเสียงที่แตกซ่านั่นเอง

4.1.2 ผลการทดลองของ Overdrive Effect

การออกแบบเสียงเอฟเฟคแบบ Distortion Effect นั้นจะมีบักษณะการทำงานของระบบเช่นเดียวกับการออกแบบเสียงเอฟเฟคแบบ Overdrive Effect แต่จะมีลักษณะของการออกแบบสมการที่แตกต่างกันโดยจะเป็นการขยายขนาดของสัญญาณอินพุตที่แบ่งออกเป็นช่วงๆ ซึ่งลักษณะของสัญญาณเอาต์พุตที่ได้จะมีการขยายของสัญญาณเสียงที่ละเอียดกว่าการขยายสัญญาณอินพุตแบบ Overdrive Effect เมื่อทดลองป้อนสัญญาณอินพุตเป็นสัญญาณ sine waveแสดงได้ดังรูปที่ 4.3

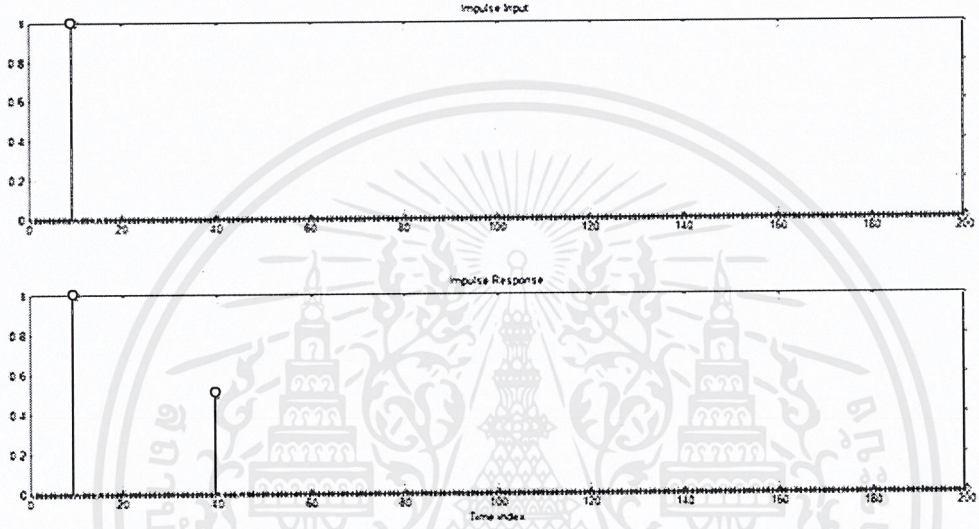


รูปที่ 4.3รูปแสดงสัญญาณอินพุตและสัญญาณเอาต์พุตของ Overdrive Effect

จะเห็นได้ว่า เมื่อทำการทดลองป้อนสัญญาณอินพุตเป็นสัญญาณ sine wave ขนาด 200 samples เข้าไปในระบบของ Overdrive Effect เมื่อมีสัญญาณอินพุตเข้ามาในระบบแล้วจะทำให้เกิดการขยายของแอมพลิจูดซึ่งมีลักษณะการทำงานคล้ายคลึงกับ Distortion Effect แต่มีข้อแตกต่างในการตัดยอดของสัญญาณคือ Overdrive Effect จะมีการตัดยอดของสัญญาณอินพุตที่ถูกขยายแล้วในระดับที่แตกต่างกัน แล้วส่งออกเป็นสัญญาณเอาต์พุตที่มีลักษณะเป็นเสียงแตกที่มีความแตกซ่าของเสียงน้อยกว่าแบบ Distortion Effect

4.1.3 ผลการทดลองของ Delay Effect

ในการออกแบบเสียงเอฟเฟคแบบ Delay Effect จะมีลักษณะการทำงานเป็นการหน่วงเวลาของสัญญาณอินพุตที่เข้าไป ทำให้สัญญาณเอาต์พุตที่ได้มีลักษณะถูกหน่วงเวลาออกไป เมื่อทดลองป้อนสัญญาณ อินพุตจำนวน 1sample ที่มีขนาดเท่ากับ 1เข้าไปในระบบของ Delay Effect ที่มีการหน่วงสัญญาณออกไป 20 samples จะได้สัญญาณเอาต์พุต แสดงดังรูปที่ 4.4



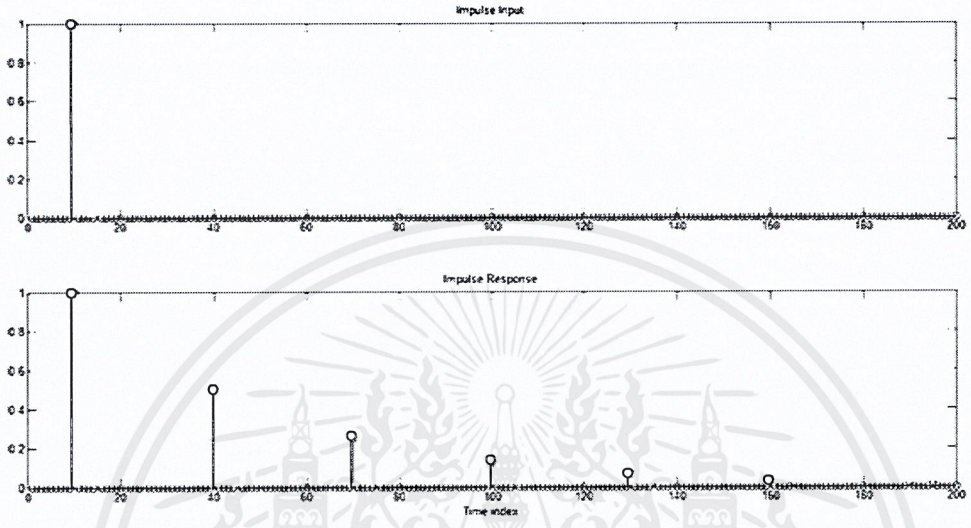
รูปที่ 4.4รูปแสดงสัญญาณอินพุตและสัญญาณเอาต์พุตของ Delay Effect

จากรูปที่ 4.4เมื่อป้อนสัญญาณอินพุตเข้าไปในระบบของ Delay Effect จะได้สัญญาณเอาต์พุตที่มีการรวมสัญญาณอินพุตกับสัญญาณเอาต์พุตที่ถูกเลื่อนเวลาออกไปตามเวลาของ Time delay

4.1.4 ผลการทดลองของ Reverb Effect

ในการออกแบบเอฟเฟคแบบ Reverb Effect จะมีลักษะการทำงานของระบบเป็นแบบการหน่วงเวลาของสัญญาณอินพุตที่เข้าไปเช่นเดียวกับเสียงเอฟเฟคแบบ Delay Effect แต่ในส่วนของ Reverb Effect จะมีการนำเอาสัญญาณเอาต์พุตที่ได้มาทำการหน่วงเวลาของสัญญาณออกไปอีก ซึ่งทำให้สัญญาณเอาต์พุตที่ได้จะมีลักษณะของสัญญาณที่มีการหน่วงเวลาออกไปเรื่อย

แบบไม่สิ้นสุดเมื่อ ทดลองป้อนสัญญาณ อินพุตจำนวน 1sample ที่มีขนาดเท่ากับ 1เข้าไปในระบบของ Reverb Effect จะได้ลักษณะของสัญญาณเอาต์พุต แสดงดังรูปที่ 4.5



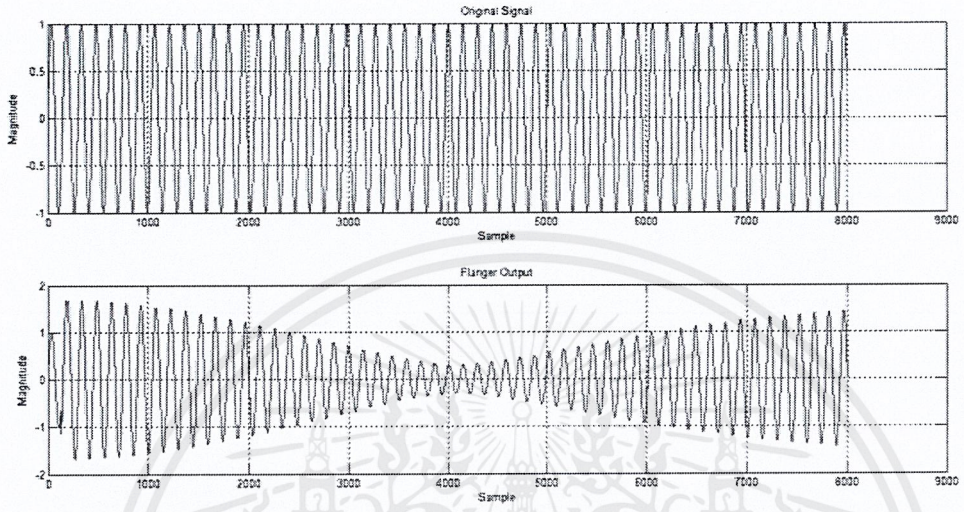
รูปที่ 4.5รูปแสดงสัญญาณอินพุตและสัญญาณเอาต์พุตของ Reverb Effect

จากรูปที่ 4.5เมื่อป้อนสัญญาณอินพุตเข้าไปในระบบของ Reverb Effect ทำให้สัญญาณอินพุตที่ป้อนเข้ามาเกิดการเลื่อนเวลาออกไปตามตาม Time delay แล้วนำสัญญาณที่ได้กลับมารวมกับสัญญาณอินพุตก่อนจะทำการเลื่อนเวลาของสัญญาณอินพุตออกไปอีกครั้ง ทำให้ได้สัญญาณเอาต์พุตที่มีลักษณะเป็นการเลื่อนเวลาของสัญญาณอินพุตออกไปเรื่อยๆ โดยมีขนาดของแอมพลิจูดลดลงตามลำดับ

4.1.5 ผลการทดลองของ Flanger Effect

ในการออกแบบเสียงเอฟเฟคแบบ Flanger Effect จะมีลักษณะการทำงานของระบบเช่นเดียวกับเสียงเอฟเฟคแบบ Delay Effect โดยการทำงานของ Flanger Effect สัญญาณอินพุตที่ป้อนเข้าไปจะถูกหน่วงเวลาออกไปและนำสัญญาณ sine wave ที่มีความถี่ต่ำเข้าไปกวนกับสัญญาณที่ถูกหน่วงเวลา ทำให้ได้ลักษณะของสัญญาณเอาต์พุตที่มีลักษณะเป็นลูกคลื่นเนื่องการ

รวมกันของแต่ละตำแหน่งของสัญญาณที่ถูกหน่วงเวลาออก เมื่อ ทดลอง ป้อนสัญญาณ อินพุตเป็นสัญญาณ sine wave จะได้ผลการทดลองแสดงดังรูปที่ 4.6



รูปที่ 4.6รูปแสดงสัญญาณอินพุตและสัญญาณเอาต์พุตของ Flanger Effect

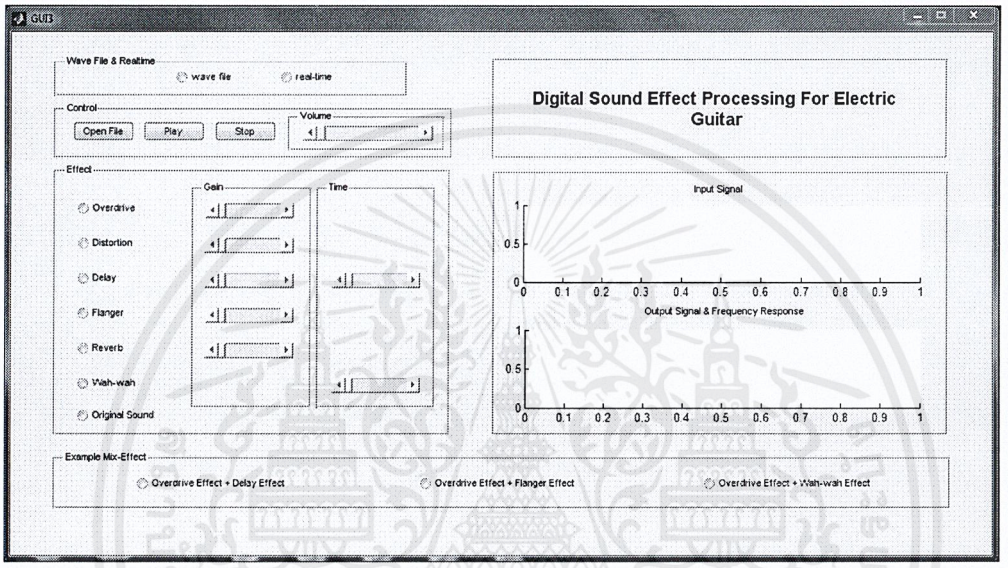
จากรูปที่ 4.6เมื่อป้อนอินพุตเข้าระบบของ Flanger Effect จะทำให้ได้สัญญาณเอาต์พุตที่ถูกหน่วงเวลาออกไป และนำสัญญาณ sine wave ที่มีความถี่ต่ำเข้าไปกวนกับสัญญาณที่ถูกหน่วงเวลา ทำให้ได้สัญญาณเสียงที่มีลักษณะการขึ้นลงของระดับแอมพลิจูดของสัญญาณเอาต์พุต

4.2 ผลการทดลองการรับอินพุตแบบReal-Time

จากการออกแบบการรับอินพุตแบบ Real-time นั้น สามารถออกแบบให้มีการรับอินพุตแบบ Real-time ได้ แต่มีเกิดการดีเลย์ของสัญญาณเสียงเมื่อมีการเล่นเสียงออกจากโปรแกรม MATLAB ประมาณ 1-2 วินาที เนื่องจากการประมวลผลของคอมพิวเตอร์ส่งผลต่อการทำงานของโปรแกรม MATLAB ที่ใช้ในการรับอินพุตแบบ Real-time ซึ่งจะทำให้เกิดการดีเลย์และการกระตุกของสัญญาณเสียงเมื่อมีการเล่นสัญญาณเสียงที่รับเข้าไปและสัญญาณเสียงที่ออกมาจากซาวด์การ์ดผ่านลำโพง อีกทั้งเนื่องจากในโปรแกรม MATLAB ยังมีข้อจำกัดในระบบที่ทำงานในแบบ parallel ที่มีลักษณะของการทำงานที่ทำงานแบบควบคู่กัน

4.3 ผลการทดลองการออกแบบGraphic User Interface (GUI)

เป็นส่วนการทำงานของโปรแกรม MATLAB ในส่วนของการพัฒนาโปรแกรมประยุกต์ที่มีการสร้างส่วนติดต่อกับผู้ใช้เป็นทางด้านกราฟิก (Application development) ซึ่งเป็นการออกแบบส่วนติดต่อกับผู้ใช้งาน เพื่อให้ผู้งานเกิดความสะดวกสบายในการใช้งาน ซึ่งการออกแบบการเขียนสร้าง GUI สามารถออกแบบแสดงได้ดังรูปที่ 4.7

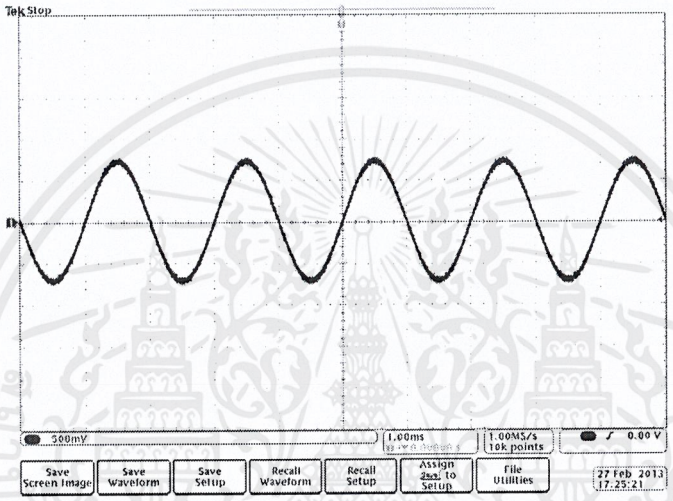


รูปที่ 4.7รูปแสดงส่วนการติดต่อกับผู้ใช้งาน Graphic User Interface

จากรูปที่ 4.7 เป็นการออกแบบส่วนติดต่อกับผู้ใช้งาน หรือ GUI (Graphic User Interface) โดยมีลักษณะการทำงานของส่วนติดต่อคือ เมื่อผู้ใช้งานเลือกการทำงานในส่วนของอินพุตแล้ว จะสามารถเลือกการทำงานในส่วนของอัลกอลิทึมของแต่ละเอฟเฟคได้ โดยผู้ใช้งานสามารถเลือกป้อนค่าพารามิเตอร์ที่สำคัญของแต่ล่ะเอฟเฟคได้ ซึ่งจะทำให้สัญญาณเสียงเอาต์พุตที่ออกมานั้นมีลักษณะที่แต่ต่างกันออกไปตามการป้อนค่าพารามิเตอร์ อีกทั้งยังมีการแสดงผลในส่วนของการประมวลผลที่สามารถแสดงลักษณะของการประมวลสัญญาณของแต่ล่ะเอฟเฟคได้

4.4 ผลการทดลองการออกแบบเสียงเอฟเฟคกีตาร์ในโปรแกรม Visual Studio

จากการออกแบบบอัลกอลิทึมเสียงเอฟเฟคจากโปรแกรม MATLAB ทางผู้จัดทำได้ทำการประยุกต์อัลกอลิทึมของเสียงแต่ละเอฟเฟคไปยังโปรแกรม Visual Studio เพื่อเป็นการอำนวยความสะดวกแก่ผู้ใช้งาน โดยใช้ภาษา C# ในการออกแบบอัลกอลิทึมของเสียงแต่ละเอฟเฟค โดยทำการทดลองป้อนสัญญาณ sine wave ความถี่ 500 Hz ขนาด 1 V_pแสดงได้ดังรูปที่ 4.8

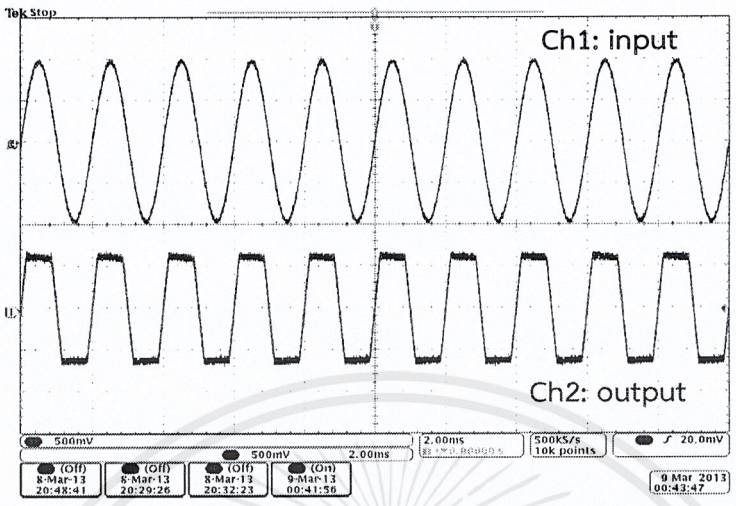


รูปที่ 4.8 รูปแสดงสัญญาณ sine waveความถี่ 500 Hz ขนาด 1 V_p

จากรูปที่ 4.8 เมื่อทำการป้อนสัญญาณ sine wave ไปยังอัลกอลิทึมของเสียงแต่ละเอฟเฟค ได้ผลการทดลองแสดงได้ดังนี้

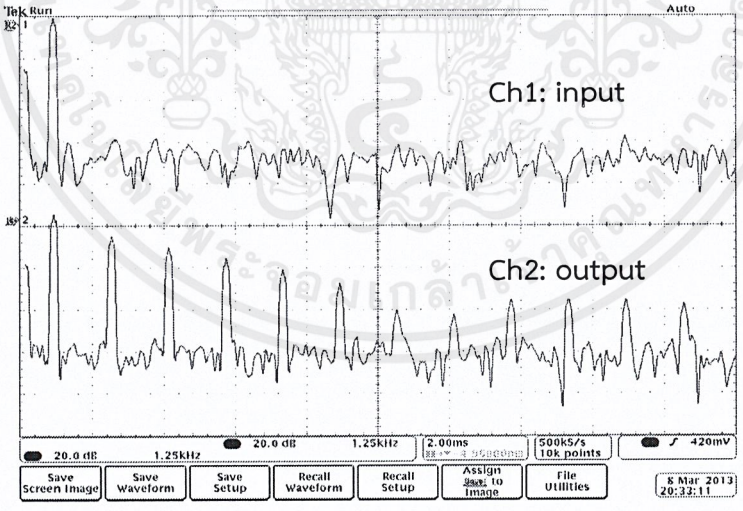
4.4.1 ผลการทดลองของ Overdrive Effect และ Distortion Effect

ในการออกแบบเสียงเอฟเฟคแบบ Overdrive Effect และ Distortion Effect ทางผู้จัดทำได้ทำการรวมการออกแบบเสียงอัลกอลิทึมของเอฟเฟคทั้งสองชนิดเข้าด้วยกัน เนื่องจากการทำงานของ Overdrive Effect และ Distortion Effect มีการทำงานที่คล้ายคลึงกัน ซึ่งจากการทดลองป้อนสัญญาณเสียง sine wave ขนาดความถี่ 500 Hz ได้ผลการทดลองแสดงได้ดังรูปที่ 4.9



รูปที่ 4.9รูปแสดงสัญญาณเอาต์พุตของ Overdrive Effectและ Distortion Effect

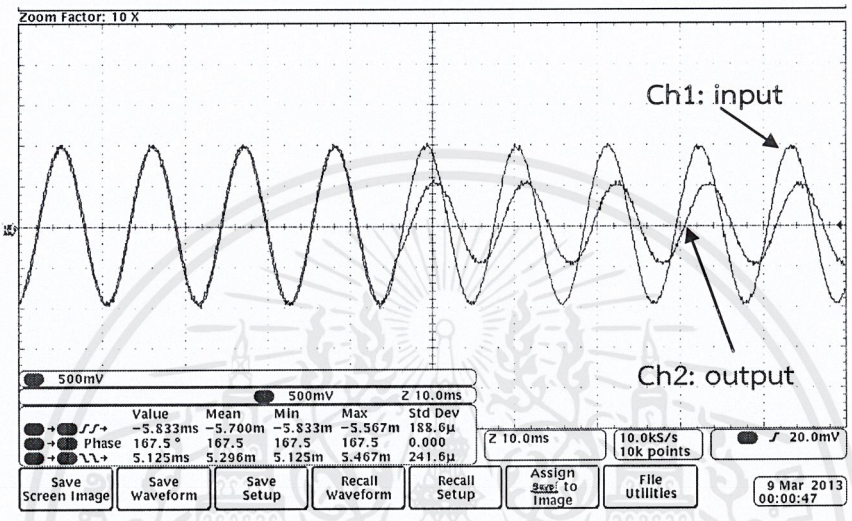
จากรูปที่ 4.9 เมื่อทำการป้อนสัญญาณอินพุตเข้ายังอัลกอลิทึมของเอฟเฟค จะได้ลักษณะของสัญญาณเอาต์พุตที่มีการตัดยอดของสัญญาณทำให้ได้เสียงที่แตกซ่า เนื่องจากสัญญาณที่ถูกตัดยอดจะมีการเพิ่มของฮาร์มอนิกที่ตำแหน่งอื่นๆ ทำให้เสียงมีลักษณะแตกซ่า ซึ่งสามารถแสดงดังรูปที่ 4.10



รูปที่ 4.10รูปแสดงสัญญาณ spectrum ของ Overdrive Effect และ Distortion Effect

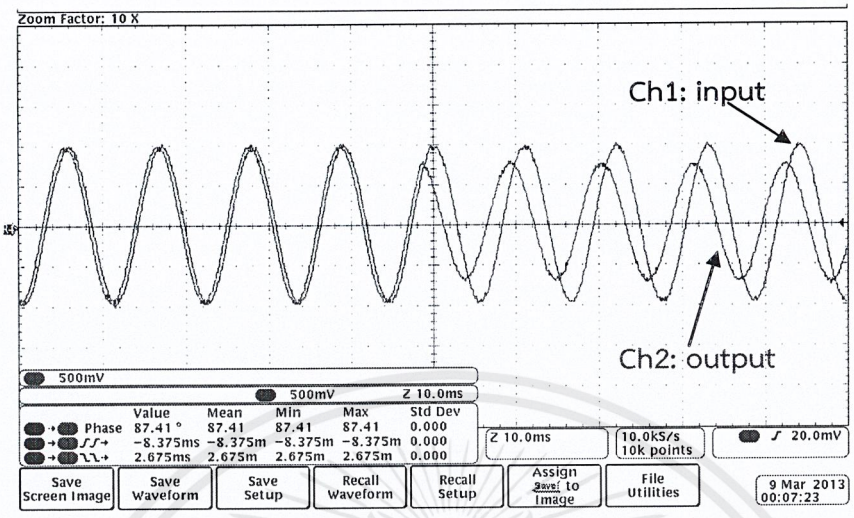
4.4.2 ผลการทดลองของ Delay Effect

จากการป้อนสัญญาณ sine wave ขนาดวามถี่ 500 Hz โดยเมื่อทำการปรับค่าพารามิเตอร์ในส่วนของ Time Delay ให้มีค่าเท่ากับ 1 วินาทีและขนาดของแอมพลิจูดเท่ากับ $0.5V_p$ได้ผลการทดลองแสดงได้ดังรูปที่ 4.11



รูปที่ 4.11รูปแสดงรูปแสดงสัญญาณเอาต์พุตของ Delay Effect ที่ Time Delay เท่ากับ 1 วินาที

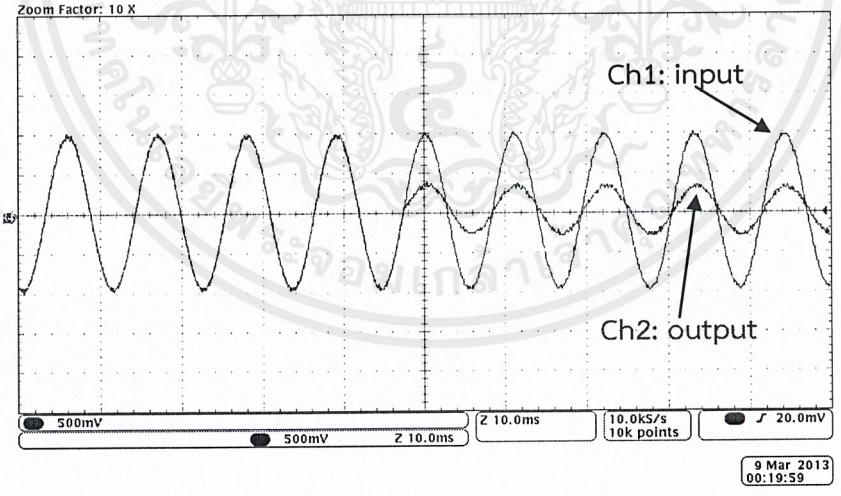
จากรูปที่ 4.12 เมื่อทำการเทียบสัญญาณอินพุตและสัญญาณเอาต์พุตเข้าด้วยกัน ทำให้ทราบได้ว่าสัญญาณเอาต์พุตที่ได้มีการเริ่มต้นเฟสของสัญญาณเปลี่ยนไปเนื่องจากการหน่วงเวลาของสัญญาณอินพุต และมีขนาดของแอมพลิจูดลดลงตามการปรับของค่าพารามิเตอร์ทำการปรับค่าของพารามิเตอร์ในส่วนของ Time Delay ให้มีค่าเท่ากับ 2 วินาที และขนาดของแอมพลิจูดเท่ากับ $0.7\ V_p$ จะได้ผลการทดลองแสดงได้ดังรูปที่ 4.12



รูปที่ 4.12 รูปแสดงรูปแสดงสัญญาณเอาต์พุตของ Delay Effect ที่ Time Delay เท่ากับ 2 วินาที

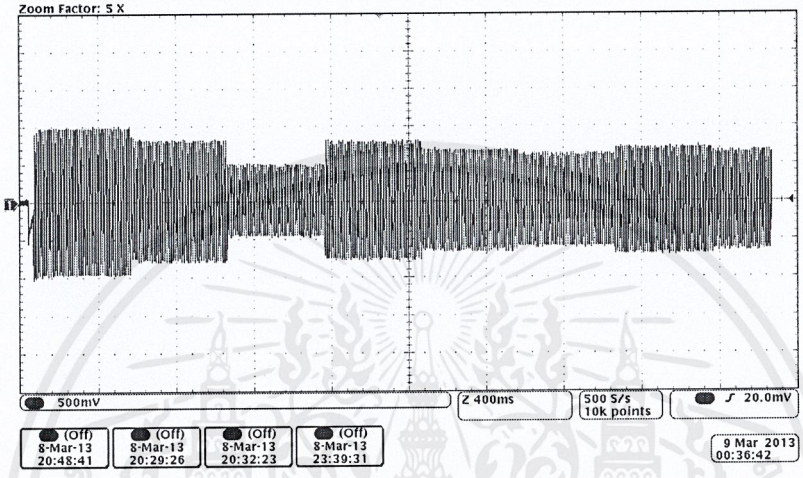
4.4.3 ผลการทดลองของ Reverb Effect

จากการป้อนสัญญาณ sine wave ขนาดวามถี่ 500 Hz โดยการปรับค่าพารามิเตอร์ในส่วนของ Time Delay มีค่าเท่ากับ 0.15 วินาทีได้ผลการทดลองแสดงได้ดังรูปที่ 4.13



รูปที่ 4.13รูปแสดงสัญญาณเอาต์พุตของ Reverb Effectที่ Time Delay เท่ากับ 0.15 วินาที

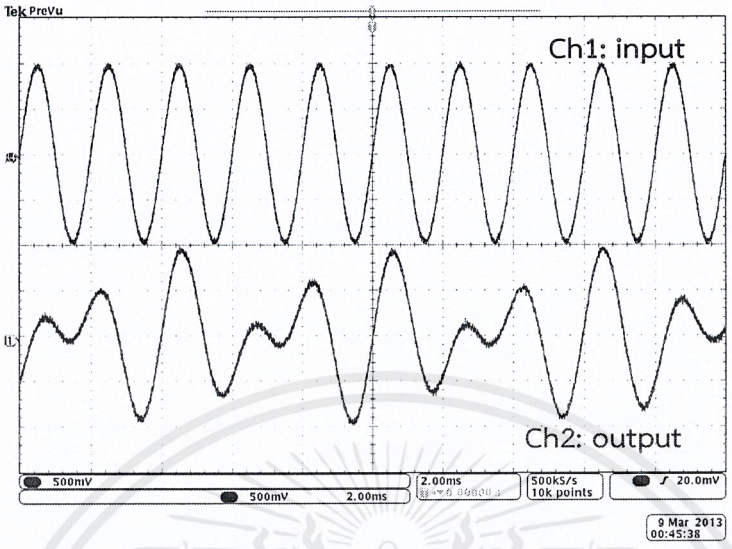
จากรูปที่ 4.13 สัญญาณที่ได้เกิดจากการหน่วงเวลาของสัญญาณอินพุตออกไปตามค่าของTime delay แล้วนำสัญญาณที่ได้กลับมารวมกับสัญญาณเดิม ก่อนจะทำการหน่วงเวลาของสัญญาณออกไปอีกครั้ง ทำให้ได้สัญญาณเอาต์พุตที่มีลักษณะเป็นการหน่วงเวลาของสัญญาณอินพุตออกไปเรื่อยๆ ซึ่งแสดงลักษณะของสัญญาณเอาต์พุตที่ได้ดังรูปที่ 4.14



รูปที่ 4.14 รูปแสดงสัญญาณเอาต์พุตของ Reverb Effect

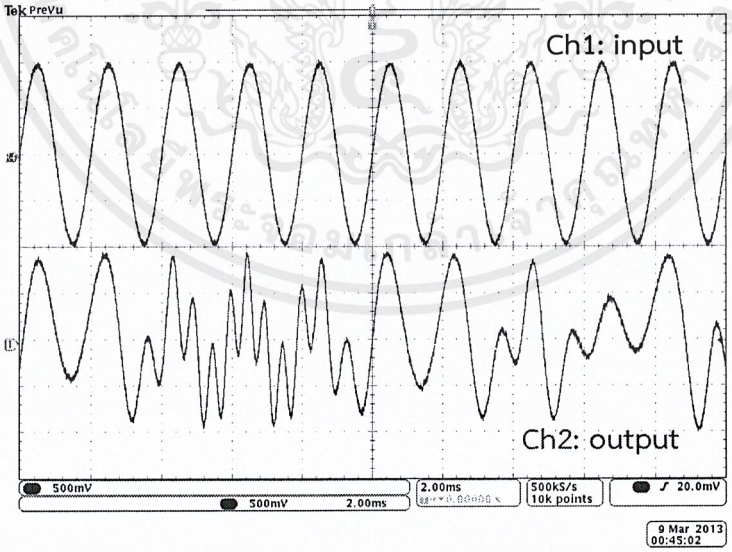
4.4.4 ผลการทดลองของ Flanger Effect

จากการป้อนสัญญาณ sine wave ขนาดความถี่ 500 Hz โดยมีการปรับค่าพารามิเตอร์ในส่วนของสัญญาณความถี่ต่ำ (LFO) ที่ขนาดความถี่ 20 Hzได้ผลการทดลองแสดงได้ดังรูปที่ 4.15



รูปที่ 4.15รูปแสดงสัญญาณเอาต์พุตของFlanger Effect ที่ LFO มีค่าเท่ากับ 20 Hz

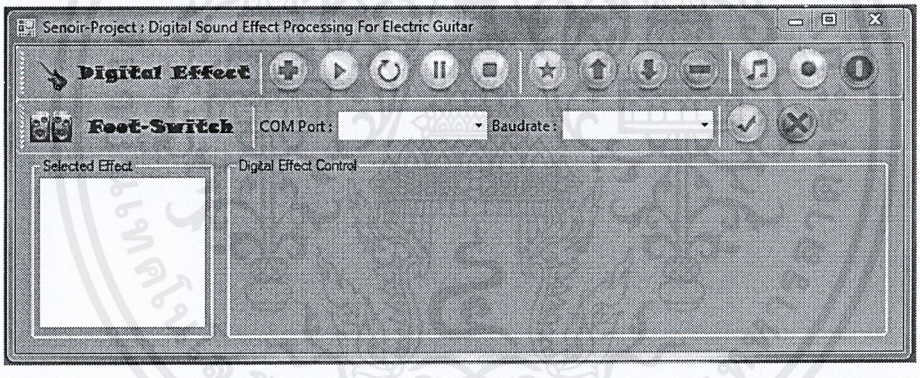
จากรูปที่ 4.15 สัญญาณเอาต์พุตที่ได้จะมีลักษณะถูกกวนด้วยสัญญาณความถี่ต่ำ ทำให้ได้ลักษณะของสัญญาณเอาต์พุตที่มีขนาดของแอมพลิจูดที่ขึ้นลง เนื่องจากรบกวนของสัญญาณความถี่ต่ำเมื่อทำการเพิ่มค่าพารามิเตอร์ในส่วนของสัญญาณความถี่ต่ำ (LFO) ให้มีขนาดความถี่ 100 Hz ได้ผลการทดลองแสดงได้ดังรูปที่ 4.16



รูปที่ 4.16 รูปแสดงสัญญาณเอาต์พุตของFlanger Effect ที่ LFO มีค่าเท่ากับ 100 Hz

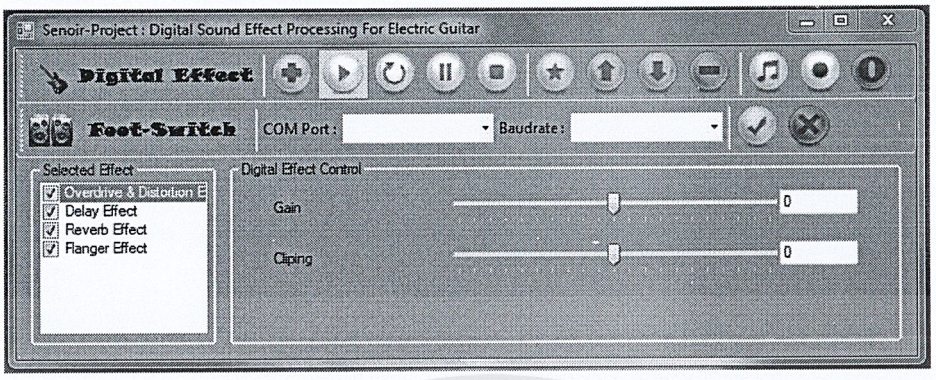
4.5 ผลการทดลองการนำไปประยุกต์ใช้งาน

จากการประยุกต์การใช้งานโดยการนำอัลกอลิทึมที่ได้ทำการออกไว้จากโปรแกรม MATLAB ไปประยุกต์โดยการใช้ภาษา C#ในการสร้างไฟล์ .exe เพื่ออำนวยความสะดวกในการใช้งานและแก้ไขปัญหาการประมวลแบบควบคู่ ซึ่งสามารถแก้ไขปัญหาดังกล่าวได้ โดยการทำงานของโปรแกรมจะสามารถทำงานได้ใน 2 ลักษณะ ได้แก่ การทำงานในส่วนของoff-line คือ ผู้ใช้งานสามารถเลือกไฟล์เสียง.wav หรือ .mp3 เพื่อกำหนดอินพุตของโปแกรม และการทำงานในส่วนของ on-line คือ ผู้ใช้งานสามารถเชื่อมต่อสัญญาณอินพุตผ่านช่อง Microphone หรือ Line-in ของคอมพิวเตอร์ เพื่อเป็นการใช้งานในส่วนของการเล่นแบบ real-time ซึ่งในการทำงานของโปรแกรม MATLAB จะสามารถใช้งานในส่วนของ on-line ได้ แต่จะมีปัญหาในส่วนของการดีเลย์ของสัญญาณเอาต์พุตและสัญญาณเอาต์พุตมีลักษณะกระตุกเมื่อเล่นเสียงผ่านลำโพง แต่การทำงานในส่วนของ on-line ในโปรแกรม Visual Studio จะลดการดีเลย์และการกระตุกของสัญญาณเสียงได้ อีกทั้งขณะที่โปรแกรมกำลังทำงานหรือขณะรับอินพุตจะสามารถเพิ่มเปลี่ยนเสียงเอฟเฟคที่ต้องการได้ อีกทั้งยังสามารถเลือกการทำงานของแต่ละเอฟเฟคได้ขณะที่โปรแกรมกำลังทำงานอยู่ ซึ่งลักษณะของโปรแกรมที่ได้ทำการออกแบบแสดงได้ดังรูปที่ 4.17



รูปที่ 4.17รูปแสดงลักษณะของโปรแกรมที่ทำการออกแบบ

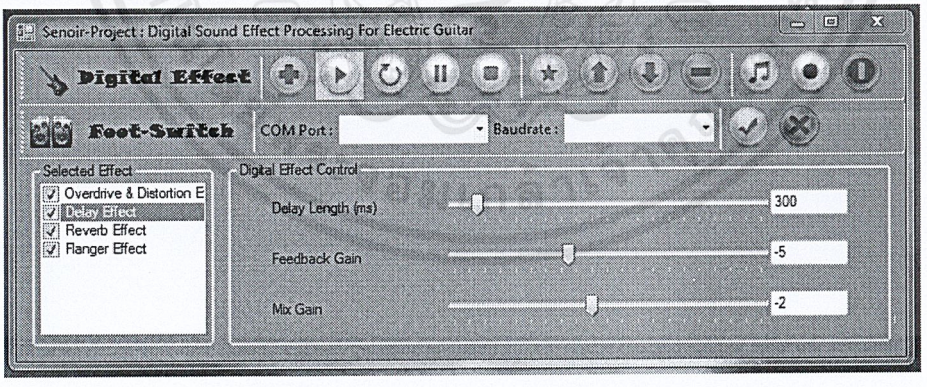
จากรูปที่ 4.17เมื่อทำการเลือกเอฟเฟคที่ต้องการจะมีการแสดงค่าพารามิเตอร์ที่สามารถตั้งค่าได้ระหว่างการทำงานของโปรแกรม สามารถแสดงได้ดังรูปที่ 4.18



รูปที่ 4.18รูปแสดงการทำงานของโปรแกรมเมื่อเลือกเอฟเฟคที่ต้องการ

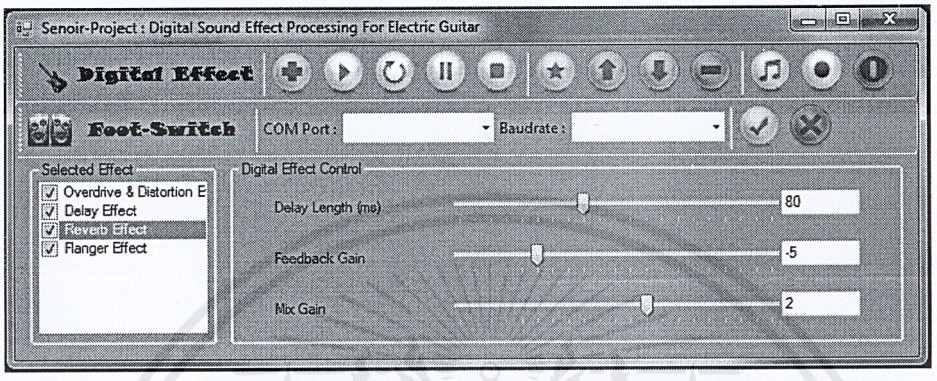
จากรูปที่ 4.18เมื่อทำการเพิ่มเสียงเอฟเฟคที่ต้องการแล้วสามารถกำหนดค่าพารามิเตอร์ที่ต้องการได้ โดยเสียงเอฟเฟคแบบ Overdrive Effect และเสียงเอฟเฟคแบบ Distortion Effect จะมีค่าพารามิเตอร์ที่สำคัญ 2 ส่วนคือ ส่วนของอัตราการขยายและส่วนของขนาด Threshold ที่สามารถปรับขนาดของแต่ละส่วนตามต้องการได้ ซึ่งลักษณะของสัญญาณเสียงที่ได้จะมีลักษณะเสียงที่แตกต่างกันออกไปตามค่าพารามิเตอร์ได้กำหนดไว้

ในขณะที่ผู้ใช้งานทำการเลือกใช้เสียงแต่ละเอฟเฟคผู้ใช้งานสามารถเลือกปรับค่าพารามิเตอร์ของแต่ละเอฟเฟคได้เพื่อเป็นการกำหนดลักษณะของสัญญาณเอาต์พุตที่ได้ โดยค่าพารามิเตอร์ของแต่ละเอฟเฟคสามารถปรับได้ดังนี้



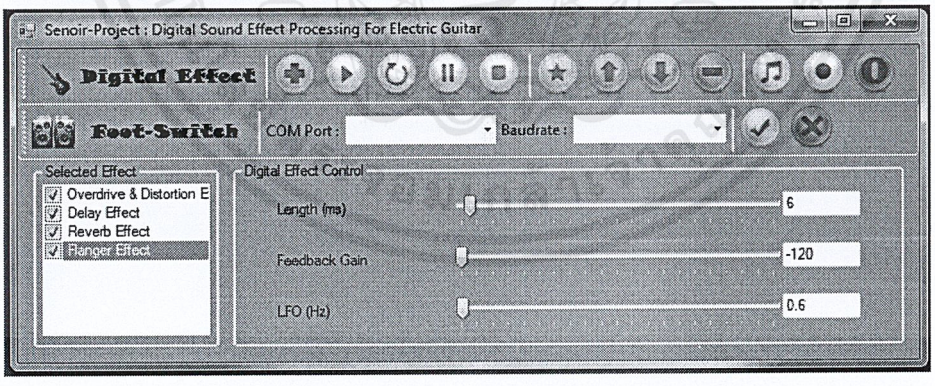
รูปที่ 4.19รูปแสดงการปรับค่าพารามิเตอร์ของ Delay Effect

จากรูปที่ 4.19เป็นการแสดงการปรับค่าพารามิเตอร์ของ Delay Effect โดยผู้ใช้งานสามารถปรับค่าในส่วนของการหน่วงเวลาของสัญญาณอินพุต และในส่วนของอัตราการขยายของสัญญาณอินพุตที่ทำการหน่วงเวลาออกไป



รูปที่ 4.20รูปแสดงการปรับค่าพารามิเตอร์ของ Reverb Effect

จากรูปที่ 4.20 เป็นการปรับค่าพารามิเตอร์ของ Reverb Effect โดยค่าพารามิเตอร์ของ Reverb Effect จะมีค่าของพารามิเตอร์ที่ใกล้เคียงกับค่าพารามิเตอร์ของ Delay Effect แต่ค่าพารามิเตอร์ของการหน่วงเวลาของ Reverb Effect จะมีค่าน้อยกว่าค่าพารามิเตอร์ของ Delay Effect และค่าพารามิเตอร์ของการหน่วงเวลาของ Flanger Effect ซึ่งจะทำให้ได้ลักษณะของเสียงสัญญาณเอาต์พุตที่แตกต่างกันออกไป



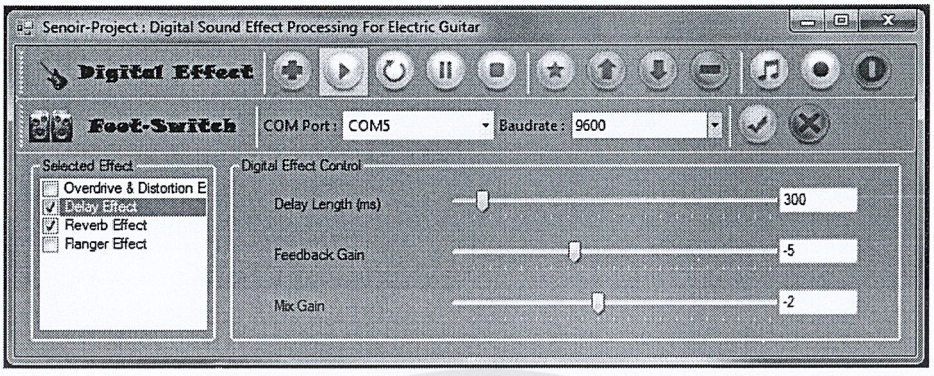
รูปที่ 4.21รูปแสดงการปรับค่าพารามิเตอร์ของ Flanger Effect

จากรูปที่ 4.21เป็นการปรับค่าพารามิเตอร์ของ Flanger Effect ที่มีการกำหนดค่าของการหน่วงเวลาและค่าความถี่ต่ำที่ใช้ในการมอตดูเลตกับสัญญาณอินพุต ทำให้ได้ลักษณะเสียงของสัญญาณเอาต์พุตที่แตกต่างกันออกไป

ซึ่งจากขั้นต้นเมื่อทำการเพิ่มเสียงเอฟเฟคที่ต้องการแล้ว สามารถควบคุมการทำงานของเสียงเอฟเฟคได้โดยการเชื่อมต่อแผงควบคุมด้วยเท้า (Foot-switch) เข้ากับโปรแกรมโดยการเชื่อมต่อพอร์ท RS-232 เมื่อทำการเชื่อมต่อกับโปรแกรมแล้วสามารถเลือกการทำงานของเอฟเฟคได้จากแผงควบคุมด้วยเท้า สามารถแสดงได้ดังรูปที่ 4.22

รูปที่ 4.22รูปแสดงการเชื่อมต่อแผงควบคุมด้วยเท้า

จากรูปที่ 4.22เมื่อทำการกดสวิสต์ที่ 1 และ 4 เพื่อปิดการทำงานของเอฟเฟคที่ 1 และเอฟเฟคที่ 4 แผงควบคุมด้วยเท้าก็จะสามารถส่งข้อมูลผ่านพอร์ท RS-232 เพื่อปิดการทำงานของเอฟเฟคดังกล่าว และแสดงผลผ่านโปรแกรม แสดงได้ดังรูปที่ 4.23



รูปที่ 4.23รูปแสดงการควบคุมผ่านแผงควบคุมด้วยเท้า

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

เนื่องด้วยโครงงานนี้เป็นโครงงานที่เน้นเกี่ยวกับการปรับแต่งเสียง เพื่อให้ได้เสียงที่มีคุณลักษณะที่พึงประสงค์ ตอบสนองตามความต้องการทางด้านดนตรี โดยการเน้นในส่วนของการสร้างเอฟเฟคต่างๆ ได้แก่ Distortion Effect, Overdrive Effect, Delay Effect, Reverb Effect, และ Flanger Effect เริ่มจากการออกแบบโดยใช้หลักการของการประมวลผลสัญญาณดิจิตอลและได้ศึกษาการใช้งานแบบ Real-time โดยการสร้างและออกแบบด้วยโปรแกรม MATLAB แล้วจึงสร้างระบบ GUI (Graphic User Interface) โดยเป็นการออกแบบซอฟแวร์ให้อยู่ในรูปของไฟล์ .exe เพื่อความสะดวกในการใช้งาน

อย่างไรก็ตามโครงงานนี้เป็นเพียงการจำลอง และทดสอบการประมวลผลสัญญาณเสียงโดยใช้ภาษา C# ในการจำลองและออกแบบอัลกอลิทึมในการสร้างเสียงแต่เอฟเฟคอีกทั้งในการประยุกต์ใช้งานยังมีปัญหาในการเล่นแบบ Real-time เนื่องจากสัญญาณที่ได้จากการอินเตอร์เฟสจากซาวด์การ์ดเป็นการเรียกใช้ API จากระบบปฏิบัติการของ Windows โดยสัญญาณอินพุตที่ได้นั้นจะอยู่ในรูปของข้อมูลชนิดไบท์ที่มีขนาดของข้อมูลที่ไม่ทราบตำแหน่งและมีขนาดของข้อมูลไม่จำกัด อีกทั้งสัญญาณที่ได้เกิดการ Delay ของสัญญาณ ซึ่งไม่สามารถนำไปประมวลผลสัญญาณเสียงได้

5.2 ข้อเสนอแนะ

การออกแบบการประมวลผลแบบ Real-time ในระบบปฏิบัติการของ Windows นั้น ยังมีข้อจำกัดในส่วนของการรับสัญญาณอินพุตที่มีการ Delay ของสัญญาณ เนื่องในการอินเตอร์เฟสซาวด์การ์ดเพื่อจำลองการใช้งานให้เปรียบเสมือนวงจรแปลงอนาลอกเป็นดิจิตอลและวงจรแปลงดิจิตอลเป็นอนาลอกนั้นไม่สามารถทำงานพร้อมกันได้ ซึ่งทำให้สัญญาณที่ได้เกิดการ Delay ของสัญญาณ

บรรณานุกรม

[1] Litan. *Digital Signal Processing Fundamentals and Application.*Academic Press, UK, 2008.

[2] Zolzer, Udo. *Digital Audio Signal Processing.* John Wiley & Sons, Ltd, Publication, 2008.

[3] Smith, B.J., *Acoustics*, Longman, 1971.

[4] audiocity2u.com. "Effects units"
http://www.audiocity2u.com/Knowledge-Effects-units.html

[5] mathworks.com. "Data Acquisition Toolbox"
http://www.mathworks.com/products/daq/

[6] cs.sfu.ca. "Matlab Real-Time"
http://www.cs.sfu.ca/personal/ Matlab Real-Time.htm

[7] microchip.com "PIC16F688 Data Sheet"
http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010215

[8] alldatasheet.com "MAX232n Data Sheet"
http://www.alldatasheet.com/data-sheetpdf/pdf/27230/TI/MAX232N.html

[9] aplayboysreturn.blogspot.com "Instrument cable"
http://aplayboysreturn.blogspot.com/2012/07/instrument-cables.html

[10] microsoft.com "Microsoft Visual Studio"
http://www.microsoft.com/visualstudio/eng/team-foundation-service

[11] th.wikipedia.org "เจมส์ มาร์แชลล์ เฮนดริกซ์"
http://th.wikipedia.org/wiki/%E0%B8%88%E0%B8%B4%E0%B8%A1%E0%B8%B4_%E0%B9%80%E0%B8%AE%E0%B8%99%E0%B8%94%E0%B8%A3%E0%B8%B4%E0%B8%81%E0%B8%8B%E0%B9%8C

ภาคผนวก ก

อัลกอลิทึมเสียงเอฟเฟค

การออกแบบอัลกอลิทึมของเสียงเอฟเฟคในโปรแกรม MATLAB

- อัลกอลิทึมเสียง Delay Effect

```
%%-Gain, Threshold other-%%
      t = 0.4;
      m = t*ai.SampleRate;
      k = 1;
%%-End-%%

%%-Effect Function-%%
      x = data;
      num = [1 zeros(1,m-1) -k];
      den = 1;
      y = filter(num,den,x);
%%-End Effect Function-%%
```

- อัลกอลิทึมเสียง Flanger Effect

```
%%-Gain, Time delay-%%
      max_time_delay=0.003;
      amp = 2;
%%-End-%%

%%-Effect Function-%%
      x = data;
      rate=1;
      index=1:length(x);
      sin_ref = (sin(2*pi*index*(rate/ai.SampleRate)))';
      max_samp_delay=round(max_time_delay*ai.SampleRate);
      y = zeros(length(x),1);

      for i = (max_samp_delay):length(x),
            cur_sin=abs(sin_ref(i));
            cur_delay=ceil(cur_sin*max_samp_delay);
            y(i) = (amp*x(i)) + amp*(x(i-cur_delay));
      end
```

- อัลกอลิทึมเสียง Reverb Effect

```
%%-Gain, Time delay, Feedback gain-%%
      t = 0.12;
      m = t*ai.SampleRate;
      j = 0.5;
%%-End-%%
%%-Effect Function-%%
      x = data;
      den = [1 zeros(1,m-1) -j];
```

```
      num = 1;
      y = filter(num,den,x);
%%-End Effect Function-%%
```

- อัลกอลิทึมเสียง Overdrive Effect

```
th = 1/3; % threshold for symmetrical soft clipping

%Overdrive Function
      x = data;
      N=length(x);
      y=zeros(1,N); % Preallocate y
      % by Formula
            for i=1:N
                if abs(x(i))< th, y(i)=8*x(i);end;
                if abs(x(i))>=th,
                    if x(i)> 0, y(i)=(3-(2-x(i)*3).^2)/3; end;
                    if x(i)< 0, y(i)=-(3-(2-abs(x(i))*3).^2)/3;
end;
                end;
                if abs(x(i))>5*th,
                    if x(i)> 0, y(i)=1;end;
                    if x(i)< 0, y(i)=-1;end;
                end;
            end;
```

- อัลกอลิทึมเสียง Distortion Effect

```
%%-Gain, Threshold other-%%
    gain = 5;
    mix = 1;
%%-End-%%

%%-Effect Function-%%
      m = data;
      x = m';
      q = 10*gain*x/max(abs(x));
      z = sign(-q).*(1-exp(sign(-q).*q)); % non-linear 1>0,
0=0, -1<0
      y = mix*z*max(abs(x))/max(abs(z))+(1-mix)*x; %Mix
original input with distortion
      y = y.*max(abs(x))/max(abs(y)); %Get output signal
%%-End Effect Function-%%
```

การออกแบบอัลกอลิทึมของเสียงเอฟเฟคในโปรแกรม Visual Studio

.Net Framework Effect Algorithm in Visual Studio

เป็นส่วนที่ช่วยในการแปลงข้อมูลที่ได้รับเข้ามาจาก 16 บิตเป็น 32 บิต และอยู่ในรูปของ float ที่มีขนาดของข้อมูลตั้งแต่ -1 ถึง 1 เพื่อนำบิตข้อมูลที่ได้ไปประมวลผล โดยจะเก็บข้อมูลของสัญญาณอินพุตให้มีลักษณะเป็นตำแหน่ง เมื่อมีการเรียกใช้งานเอฟเฟคอัลกอลิทึม โปรแกรมจะนำค่าของข้อมูลสัญญาณอินพุตที่ถูกเก็บเป็นตำแหน่งนี้ไปประมวลผล ซึ่งในการประมวลผลจะเป็นการประมวลผลแบบทีละตำแหน่งไปเรื่อยๆ จนสิ้นสุดของสัญญาณข้อมูล โดยมีลักษณะของโปรแกรมดังนี้

```
namespace DSP
{
    public class EffectStream : WaveStream
    {
        private EffectChain effects;
        public WaveStream source;
        private object effectLock = new object();
        private object sourceLock = new object();

        public EffectStream(EffectChain effects, WaveStream
sourceStream)
        {
            this.effects = effects;
            this.source = sourceStream;
            foreach (Effect effect in effects)
            {
                InitialiseEffect(effect);
            }
        }

        public EffectStream(WaveStream sourceStream)
            : this(new EffectChain(), sourceStream)
        {
        }

        public EffectStream(Effect effect, WaveStream sourceStream)
            : this(sourceStream)
        {
            AddEffect(effect);
        }

        public override WaveFormat WaveFormat
        {
            get { return source.WaveFormat; }
        }

        public override long Length
        {
            get { return source.Length; }
        }

        public override long Position
```

```
        {
            get { return source.Position; }
            set { lock (sourceLock) { source.Position = value; } }
        }

        public override int Read(byte[] buffer, int offset, int
count)
        {
            int read;
            lock (sourceLock)
            {
                read = source.Read(buffer, offset, count);
            }
            if (WaveFormat.BitsPerSample == 16)
            {
                lock (effectLock)
                {
                    Process16Bit(buffer, offset, read);
                }
            }
            return read;
        }

        private void Process16Bit(byte[] buffer, int offset, int
count)
        {
            int samples = count / 2;
            foreach (Effect effect in effects)
            {
                if (effect.Enabled)
                {
                    effect.Block(samples);
                }
            }

            for (int sample = 0; sample < samples; sample++)
            {
                // get the sample
                int x = offset + sample * 2;
                short sample16Left = BitConverter.ToInt16(buffer, x);
                short sample16Right = sample16Left;
                if (WaveFormat.Channels == 2)
                {
                    sample16Right = BitConverter.ToInt16(buffer, x +
2);
                    sample++;
                }

                // run these samples through the effect
                float sample64Left = sample16Left / 32768.0f;
                float sample64Right = sample16Right / 32768.0f;
                foreach (Effect effect in effects)
                {
                    if (effect.Enabled)
                    {
                        effect.Sample(ref sample64Left, ref
sample64Right);
                    }
```

```
                }

                sample16Left = (short)(sample64Left * 32768.0f);
                sample16Right = (short)(sample64Right * 32768.0f);

                // put them back
                buffer[x] = (byte)(sample16Left & 0xFF);
                buffer[x + 1] = (byte)((sample16Left >> 8) & 0xFF);

                if (WaveFormat.Channels == 2)
                {
                    buffer[x + 2] = (byte)(sample16Right & 0xFF);
                    buffer[x + 3] = (byte)((sample16Right >> 8) &
0xFF);
                }
            }
        }


        public bool MoveUp(Effect effect)
        {
            lock (effectLock)
            {
                return effects.MoveUp(effect);
            }
        }

        public bool MoveDown(Effect effect)
        {
            lock (effectLock)
            {
                return effects.MoveDown(effect);
            }
        }

        public void AddEffect(Effect effect)
        {
            InitialiseEffect(effect);
            lock (effectLock)
            {
                this.effects.Add(effect);
            }
        }

        private void InitialiseEffect(Effect effect)
        {
            effect.SampleRate = WaveFormat.SampleRate;
            effect.Init();
            effect.Slider();
        }

        public bool RemoveEffect(Effect effect)
        {
            lock (effectLock)
            {
                return this.effects.Remove(effect);
            }
        }
```

```
    }
}
```

Effect Framework in Visual Studio

- Delay Effect

```
namespace DSP
{
    [Export(typeof(Effect))]
    public class Delay : Effect
    {
        float delaypos;
        float odelay;
        float delaylen;
        float wetmix;
        float drymix;
        float wetmix2;
        float drymix2;
        float rspos;
        int rspos2;
        float drspos;
        int tpos;
        float[] buffer = new float[500000];

        public Delay()
        {
            AddSlider(300, 0, 4000, 20, "Delay Length (ms)");
            AddSlider(-5, -20, 20, 1, "Feedback Gain");
            AddSlider(-2, -20, 20, 1, "Mix Gain");        }

        public override string Name
        {
            get { return "Delay Effect"; }
        }

        public override void Init()
        {
            delaypos = 0;
        }

        public override void Slider()
        {
            odelay = delaylen;
            delaylen = Math.Min(slider1 * SampleRate / 1000, 500000);
            if (odelay != delaylen)
            {
                if (slider1 > 0 && odelay > delaylen)
                {
                    // effect function
                    rspos = 0; rspos2 = 0;
                    drspos = odelay / delaylen;
                    for (int n = 0; n < delaylen; n++)
                    {
                        tpos = ((int)rspos) * 2;
                        buffer[rspos2 + 0] = buffer[tpos + 0];
```

```
                    buffer[rspos2 + 1] = buffer[tpos + 1];
                    rspos2 += 2;
                    rspos += drspos;
                }
                delaypos /= drspos;
                delaypos = (int)delaypos;
                if (delaypos < 0) delaypos = 0;
            }
        }
    }
```

- Flanger Effect

```
namespace DSP
{
    [Export(typeof(Effect))]
    public class Flanger : Effect
    {
        public Flanger()
        {
            AddSlider(6, 0, 200, 1, "Length (ms)");
            AddSlider(-120, -120, 20, 1, "Feedback Gain");
            AddSlider(0.6f, 0.001f, 100, 0.1f, "LFO (Hz)");
        }

        float odelay;
        float delaylen;
        float dpposssc;
        float dpbacksc;
        float dppos;
        float dpback;
        float dpint;
        float delaypos;
        float[] buffer = new float[500000];

        public override string Name
        {
            get { return "Flanger Effect"; }
        }

        public override void Init()
        {
            delaypos = 0;
        }

        public override void Slider()
        {
            odelay = delaylen;
            delaylen = Math.Min(slider1 * SampleRate / 1000, 500000);

            dpposssc = 2 * PI * slider3 / SampleRate;
            dpbacksc = delaylen * 0.5f - 1;
        }

        public override void Sample(ref float spl0, ref float spl1)
        {
```

```
            dppos = dppos + dpposssc;
            dpback = (sin(dppos) + 1) * dpbacksc;
            dpint = delaypos - dpback - 1;
            if (dpint < 0) dpint += delaylen;

            dpint *= 2;

            float os1 = buffer[(int)dpint + 0];
            float os2 = buffer[(int)dpint + 1];

            dpint = delaypos * 2;

            buffer[(int)dpint + 0] = spl0 + os1 * wetmix;
            buffer[(int)dpint + 1] = spl1 + os2 * wetmix;
            delaypos += 1;
            if (delaypos >= delaylen) delaypos = 0;
        }
    }
}
```

- Reverb Effect

```
namespace DSP
{
    [Export(typeof(Effect))]
    public class Reverb : Effect
    {
        float delaypos;
        float odelay;
        float delaylen;
        float rspos;
        int rspos2;
        float drspos;
        int tpos;
        float[] buffer = new float[500000];

        public Reverb()
        {
            AddSlider(80, 0, 200, 20, "Delay Length (ms)");
            AddSlider(-5, -10, 10, 1, "Feedback Gain");
            AddSlider(2, -10, 10, 1, "Mix Gain");
        }

        public override string Name
        {
            get { return "Reverb Effect"; }
        }

        public override void Init()
        {
            delaypos = 0;
        }

        public override void Slider()
        {
            odelay = delaylen;
```

```
            delaylen = Math.Min(slider1 * SampleRate / 1000, 500000);
            if (odelay != delaylen)
            {
                if (slider1 > 0 && odelay > delaylen)
                {
                    // resample down delay buffer
                    rspos = 0; rspos2 = 0;
                    drspos = odelay / delaylen;
                    for (int n = 0; n < delaylen; n++)
                    {
                        tpos = ((int)rspos) * 2;
                        buffer[rspos2 + 0] = buffer[tpos + 0];
                        buffer[rspos2 + 1] = buffer[tpos + 1];
                        rspos2 += 2;
                        rspos += drspos;
                    }
                    delaypos /= drspos;
                    delaypos = (int)delaypos;
                    if (delaypos < 0) delaypos = 0;
                }
                else
                {
                    if (slider1 > 0 && odelay < delaylen)
                    {
                        drspos = odelay / delaylen;
                        rspos = odelay;
                        rspos2 = (int)delaylen * 2;
                        for (int n = 0; n < (int)delaylen; n++)
                        {
                            rspos -= drspos;
                            rspos2 -= 2;

                            tpos = ((int)(rspos)) * 2;
                            buffer[rspos2 + 0] = buffer[tpos + 0];
                            buffer[rspos2 + 1] = buffer[tpos + 1];
                        }
                        delaypos /= drspos;
                        delaypos = (int)delaypos;
                        if (delaypos < 0) delaypos = 0;
                    }
                    else
                    {
                        if (slider6 != 0 && delaypos >= delaylen)
delaypos = 0;
                    }
                }
            }
        }

        public override void Sample(ref float spl0, ref float spl1)
        {
            int dpint = (int)delaypos * 2;
            float os1 = buffer[dpint + 0];
            float os2 = buffer[dpint + 1];

            buffer[dpint + 0] = Math.Min(Math.Max(spl0 * drymix + os1
* wetmix, -4), 4);
```

```
            buffer[dpint + 1] = Math.Min(Math.Max(spl1 * drymix + os2
* wetmix, -4), 4);

            if ((delaypos += 1) >= delaylen)
            {
                delaypos = 0;
            }
        }
    }
}
```

- Overdrive and Distortion Effect

```
namespace DSP
{
    [Export(typeof(Effect))]
    public class Volume : Effect
    {
        public Volume()
        {
            AddSlider(0, -100, 100, 1, "Gain");
            AddSlider(0, -100, 100, 1, "Cliping");
        }

        public override string Name
        {
            get { return "Overdrive & Distortion Effect"; }
        }

        float adj1;
        float adj2;
        float adj1_s;
        float dadj;
        float doseek;

        public override void Slider()
        {
            adj1 = pow(2, (slider1 / 6));
            adj2 = pow(2, (slider2 / 6));
            doseek = 1;
        }

        public override void Block(int samplesblock)
        {
            if (doseek != 0)
            {
                dadj = (adj1 - adj1_s) / samplesblock;
                doseek = 0;
            }
            else
            {
                dadj = 0;
                adj1_s = adj1;
            }
        }
```

```
        public override void Sample(ref float spl0, ref float spl1)
        {
            spl0 = min(max(spl0 * adj1_s, -adj2), adj2);
            spl1 = min(max(spl1 * adj1_s, -adj2), adj2);
            adj1_s += dadj;
        }
    }
}
```