การประยุกต์ใช้ซิกมา – เดลตา มอดดูเลชั่นกับสัญญาณเสียง

SIGMA – DELTA MODULATOR WITH 1 – BIT QUANTIZER FOR ANALOG TO DIGITAL CONVERSION IN AUDIO APPLICATION

โดย

นาย ธีระพงษ์ นวมอำพันธ์

นาย บัณฑิต คุณวุฒิ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การประยุกต์ใช้ซิกมา – เดลตา มอดดูเลชั่นกับสัญญาณเสียง

SIGMA – DELTA MODULATOR WITH 1 – BIT QUANTIZER FOR ANALOG TO DIGITAL CONVERSION IN AUDIO APPLICATION

โดย

นาย ธีระพงษ์ นวมอำพันธ์ เลขประจำตัว 44010229

นาย บัณฑิต คุณวุฒิ เลขประจำตัว 44010267

อาจารย์ที่ปรึกษา

รศ.ดร มนัส สังวรศิลป์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การประยุกต์ใช้ซิกมา – เดลตา มอดดูเลชั่นกับสัญญาณเสียง

SIGMA – DELTA MODULATOR WITH 1 – BIT QUANTIZER FOR ANALOG TO DIGITAL CONVERSION IN AUDIO APPLICATION

นาย ธีระพงษ์ นวมอำพันธ์ เลขประจำตัว 44010229

นาย บัณฑิต คุณวุฒิ เลขประจำตัว 44010267

ปริญญานิพนธ์ฉบับนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

(รศ.ดร. มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

ปริญญานิพนธ์ ปีการศึกษา 2547

ภาควิชา วิศวกรรมอิเล็คทรอนิคส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประยุกต์ใช้ซิกมา – เดลตามอดดูเลชั่นกับสัญญาณเสียง

ผู้จัดทำ

1. นาย ธีระพงษ์ นวมอำพันธ์ เลขประจำตัว 44010229
2. นาย บัณฑิต คุณวุฒิ เลขประจำตัว 44010267

(รศ.ดร. มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การประยุกต์ใช้ซิกมา – เดลตามอดดูเลชั่นกับสัญญาณเสียง

นาย ธีระพงษ์ นวมอำพันธ์
นาย บัณฑิต คุณวุฒิ
รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

ในโครงงานชิ้นนี้เป็นการวิเคราะห์และออกแบบ การแปลงสัญญาณอนาลอกเป็นดิจิตอลโดยใช้ซิกมา – เดลตามอดดูเลเตอร์เพื่อประยุกต์ใช้กับสัญญาณเสียง ส่วนประกอบโดยรวมจะมีหลักๆ 2 ส่วนคือ ส่วนอนาลอกที่เป็น ซิกมา- เดลตามอดดูเลชั่น โดยใช้มอดดูเลเตอร์อันดับที่ 3 เพื่อให้สามารถลดสัญญาณรบกวนที่เกิดจาก Quantization Noise และเพิ่มอัตราส่วนระหว่างสัญญาณอินพุทและสัญญาณรบกวนให้มีค่ามากกว่า 70 dB ส่วนที่สองคือส่วนที่เป็นวงจรกรองความถี่ผ่านแบบดิจิตอลแบบ FIR ทั้งการลดความถี่สุ่มลงและเพิ่มค่าความถูกต้องทางด้านเอาท์พุท (Resolution) ให้มากกว่า 8 บิท โดยที่อัตราส่วนในการสุ่มอยู่ที่ 320 เท่าซึ่งต้องใช้ความถี่สุ่มเท่ากับ 4.8 MHz ส่วนดิจิตอลทั้งหมดจะออกแบบโดยใช้ภาษา VHDL ออกแบบลงบน FPGA

SIGMA – DELTA MODULATOR WITH 1 – BIT QUANTIZER FOR ANALOG TO DIGITAL CONVERSION IN AUDIO APPLICATION

Mr. Teerapong Nuamampun
Mr. Bundit Koonnawuti
Assoc. Prof. Dr. Manas Sangworasil(Advisor)
1st Semester, Educational year 2004

Abstract

In this project describe analysis and design Sigma – Delta modulator with 1 – bit quantizer for analog to digital conversion in audio signal application. Sigma – Delta ADC consists two paths are analog and digital, analog path is Sigma – Delta modulation, in this project is implemented modulator as a Third – order for decrease quantization noise and increase SNR (signal to noise ratio) more than 70dB in baseband, second path is FIR digital filter refer as window function which achieve more than 8 –resolution with oversampling ratio is 320 and reduce sampling rate output, requiring a sampling frequency 4..8 MHz .In this path used VHDL language design digital low pass filter in FPGA

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี ด้วยคำแนะนำและคำปรึกษาจาก รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษาโครงงาน จึงขอกราบขอบพระคุณเป็นอย่างสูงในความอนุเคราะห์ทุกๆ ด้าน

ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และบุคคลผู้ที่เคารพรัก ที่ให้ความสนับสนุน ห่วงใย และคอยให้กำลังใจในการทำโครงงานตลอดมา

ขอขอบคุณ ผศ.ประภากร สุวรรณะ ที่ให้ความช่วยเหลือ และคำปรึกษาที่เป็นประโยชน์อย่างมากตลอดจนให้ความอนุเคราะห์ในการใช้ห้องทดลอง รวมพี่ๆนักศึกษาปริญญาโทในห้อง Bio Medical ทุกคนที่คอยให้คำแนะนำ

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ขอมอบแด่ผู้มีพระคุณทุกท่าน

ลงชื่อ

_____

(ธีระพงษ์ นวมอำพันธ์)

_____

(บัณฑิต คุณวุฒิ)

สารบัญ

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ประสิทธิภาพของการประมวลผลสัญญาณดิจิตอลและในทางโทรคมนาคมโดยทั่วไปจะถูกจำกัดความถูกต้องของสัญญาณดิจิตอลที่ถูกแลงมาจากสัญญาณอนาลอก ซิกมา – เดลตามอดดูเลชั่นก็เป็นทางเลือกหนึ่งสำหรับการแปลงอะนาลอกเป็นดิจิตอลแบบ ที่มีความถูกต้องสูง (High resolution) สามารถออกแบบลงบนวงจรรวมประมวลผลสัญญาณดิจิตอลได้ เนื่องจากว่าวงจรกว่า90%ของการแปลงอะนาลอกเป็นดิจิตอลแบบ แบบ ซิกมา – เดลตา ทำงานในลักษณะดิจิตอลคือวงจรกรองความถี่แบบดิจิตอล ดังนั้นในปริญญานิพนธ์จึงได้กล่าวถึงการออกแบบ ซิกมา –เดลตา มอดดูเลชั่นเพื่อใช้กับความถี่ เสียง (50 Hz – 20 kHz) ทั้งส่วนที่เป็นอนาลอกและดิจิตอล

ส่วนที่เป็นอนาลอกจะเรียกว่า ซิกมา – เดลตา มอดดูเลชั่น ในบทที่ 2 และบทที่ 3 จะกล่าวถึงเนื้อหาในส่วนนี้พร้อมทั้งพื้นฐานสำหรับการออกแบบ

สำหรับส่วนที่เป็นดิจิตอลจะประยุกต์ใช้ FPGA (Field Programmable Gate Array) โดยใช้ภาษา VHDL ในการสร้างวงจรกรองความถี่ต่ำผ่านแบบดิจิตอล

1.2 วัตถุประสงค์ของโครงงาน

1 . เพื่อศึกษาการทำงานและส่วนประกอบ ของ การแปลงอะนาลอกเป็นดิจิตอลแบบ ซิกมา – เดลตา มอดดูเลชั่น

2. สามารถประยุกต์ใช้งานกับสัญญาณอนาลอกในช่วงความถี่ เสียง

3 .สามารถออกแบบซิกมา – เดลตา มอดดูเลชั่นที่มี Order มากกว่า 1 เพื่อศึกษาเปรียบเทียบประสิทธิภาพในการกำจัดสัญญาณรบกวน

4 .สามารถใช้ FPGA (Field Programmable Gate Array) เป็นส่วนวงจรกรองความถี่ต่ำผ่านแบบดิจิตอล

5 .นำ ADC ซิกมา – เดลตา มอดดูเลชั่นไปใช้กับสัญญาณ Audio ได้

1.3 ขั้นตอนการทำงาน

1 . ศึกษาทฤษฎีที่เกี่ยวข้องกับ ซิกมา –เดลตา มอดดูเลชั่น

2 . ทำการออกแบบวงจรมอดูเลชั่นอันดับ 1 , 2 เพื่อศึกษาประสิทธิภาพในการกำจัดสัญญาณรบกวนในช่วงความถี่ เสียง

3 . ออกแบบส่วนของวงจรกรองความถี่ต่ำผ่าน ด้วยภาษา VHDL โดยใช้วงจรกรองความถี่ต่ำผ่านดิจิตอลแบบไม่ป้อนกลับเชิงเลข (Finite Impulse Respond หรือ Non - Reclusive)

4 . นำส่วนที่เป็นอนาลอก(ซิกมา – เดลตา มอดดูเลชั่น) และส่วนวงจรกรองความถี่ต่ำผ่านแบบดิจิตอล (FPGA) มาเชื่อมต่อกันและทดสอบการทำงานโดยรวม พร้อมทั้งทำการแก้ไข

1.4 ขอบเขตของชิ้นงาน

๑ . สามารถใช้งานได้กับสัญญาณที่มีความถี่ในช่วงความถี่ เสียง

๒. สัญญาณดิจิตอลเอาท์พุทมีค่ามากกว่า 12 บิท (Resolution >12)

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1 . เข้าใจทฤษฎีที่เกี่ยวข้องและหลักการทำงานของวงจรซิกมา – เดลตา มอดดูเลชั่นในทั้งมอดดูเลชั่นอันดับหนึ่งหรือมากกว่าหนึ่ง

2 . สามารถประยุกต์ใช้งานทฤษฎีทางด้านอิเล็กทรอนิกส์ในการออกแบบและแก้ปัญหาได้

3 . เกิดทักษะในการออกแบบและการเรียนรู้ในเพื่อนำไปใช้งานจริงพร้อมทั้งประสบการณ์ที่ได้จากการลงมือปฏิบัติเอง

4 . เข้าใจวิธีการออกแบบวงจรกรองความถี่แบบดิจิตอล

5 . มีความรู้ในการเขียน โปรแกรมภาษา VHDL

บทที่ 2

ทฤษฎีและหลักการของ ซิกมา-เดลตา มอดดูเลชั่น

2.1 ทฤษฎีพื้นฐานของการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล

สัญญาณโดยทั่วไปสามารถแบ่งออกได้เป็น 2 ประเภทด้วยกัน คือสัญญาณที่อนาลอก, X (t) ที่กำหนดให้มีความต่อเนื่องในโดเมนของเวลา (Time Domain) และอีกสัญญาณประเภทที่ 2 คือสัญญาณดิจิตอล, X (n) ที่ถูกแทนด้วยลำดับของจำนวนในโดเมนทางเวลาที่ไม่ต่อเนื่อง (Discrete Time Domain) ตัวแปร n ในสัญญาณที่ไม่ต่อเนื่องทางเวลา เป็นเลขจำนวนจริงซึ่งอธิบายได้ด้วยช่วงเวลาของการสุ่ม (Sampling Interval) สัญญาณที่ไม่ต่อเนื่องทางเวลา X*(n) สามารถแสดงได้ด้วยการสุ่มสัญญาณที่มีความต่อเนื่องทางเวลา X (t)

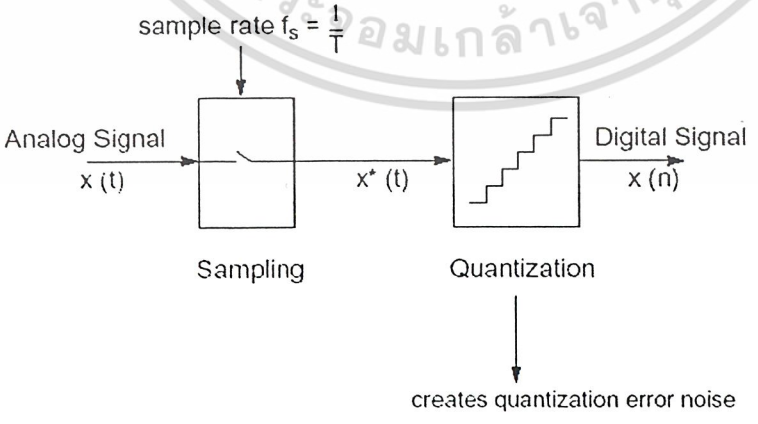
$$x^*(t) = \sum_{n=-\infty}^{\infty} x(t)\delta(t - nT) \quad \text{.....} \quad \text{....2.1}$$

โดยที่

$$\delta(t) = 1,\ t = 0$$

$$= 0,\ \text{อื่นๆ}$$

จากสมการที่ 2.1 จะเห็นว่าสัญญาณที่ไม่ต่อเนื่องทางเวลาหรือสัญญาณดิจิตอลที่ได้ เกิดจากคูณสัญญาณที่ต่อเนื่องทางเวลาด้วยสัญญาณกระตุ้น(Impulse Signal) โดยสัญญาณกระตุ้น แต่ละตัวห่างกันด้วยช่วงเวลาคงที่ค่าหนึ่งซึ่งก็คือคาบเวลาของการสุ่ม (Sampling Interval) นั่นเอง ในทางปฏิบัติแล้ว การแปลงอะนาลอกเป็นดิจิตอล (Analog - to - Digital Converters) คือกระบวนการที่แปลง x (t) เป็นสัญญาณที่ไม่ต่อเนื่องทางเวลา x*(t) โดยแต่ละช่วงเวลาของการสุ่มจะถูกกำหนดด้วยค่าความถูกต้องที่จำกัดซึ่งจะถูกประมาณด้วยรหัสทางดิจิตอล, x (t) จะแปลงเป็นลำดับของข้อมูลที่มีค่าจำกัดหรือ(Quantization Samples) x(n) กระบวนการดังกล่าวนี้จะเป็นตัวสร้างสัญญาณรบกวนขึ้นมาภายในระบบ



รูปที่ 2.1 หลักการพื้นฐานของ การแปลงอะนาลอกเป็นดิจิตอล

การแปลงอะนาลอกเป็นดิจิตอลสามารถแบ่งได้ออกเป็น 2 กลุ่มด้วยกัน โดยแบ่งจากอัตราการสุ่มหรือ (Sampling Rate) กลุ่มแรกคือการสุ่มแบบไนควิส์ท (Nyguist Rate) เช่น อะนาลอกเป็นดิจิตอลแบบ การประมาณค่า (Successive Approximation) อะนาลอกเป็นดิจิตอลแบบนี้ โดยพื้นฐานแล้วความถี่ของการสุ่มสัญญาณต้องมีค่าอย่างน้อยเป็น 2 เท่าของความถี่สูงสุดที่ทำการสุ่ม ถ้า Fin คือ ความของสัญญาณอินพุทสูงสุด และ Fs คือ ความถี่ที่ใช้ในการสุ่ม ดังนั้นจะได้ว่าความถี่ Fs ต้องมากกว่าหรือเท่ากับ 2 Fin ถ้าหากว่าความถี่ใช้สุ่มมีค่าน้อยกว่า 2Fin แล้วผลที่เกิดตามมาคือ สเปกตรัม ของสัญญาณเอาท์พุทที่ได้จะเกิดการซ้อนทับกันเป็นผลให้สัญญาณที่ได้เกิดความผิดเพี้ยนไปดังรูปด้านล่าง

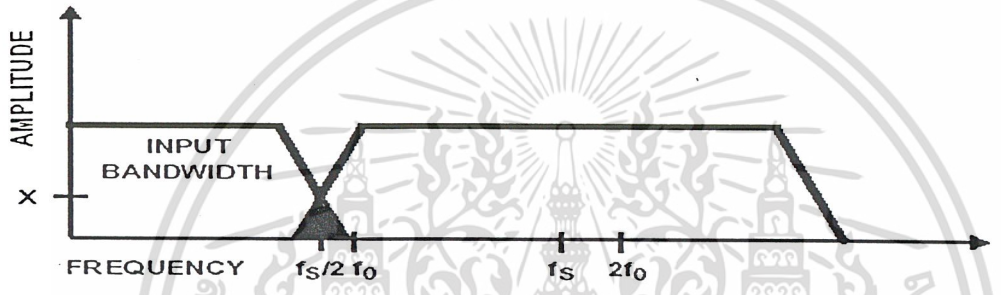


FIGURE 2A. UNDERSAMPLED SIGNAL SPECTRUM

รูปที่ 2.2 ผลที่เกิดจากการสุ่มด้วยความถี่น้อยกว่า $2F_O$ และมากกว่า $2F_O$

อะนาลอกเป็นดิจิตอล อีกกลุ่มเป็นแบบมีอัตราการสุ่มที่สูงมากกว่าสองเท่าของสัญญาณมาก ๆ(Oversampling Converters) ความถี่ที่ใช้ในการสุ่มสัญญาณอินพุทจะมีค่ามากกว่าความถี่สูงสุดของสัญญาณอินพุทมากๆ ซึ่งก็คือ $F_N << F_S$ เมื่อ F_N คือความถี่ของอินพุท และ F_S คือความถี่ของการสุ่ม

ดังรูปที่ 2.3 จะเป็นการแสดงกระบวนการพื้นฐานของการแปลงสัญญาณทางด้านอนาลอกอินพุท x (t) เป็นลำดับของรหัสตัวเลขดิจิตอล x(n) ที่มีช่วงเวลาการสุ่ม(Sampling Rate) คือ $F_s = 1/T$,โดย T คือช่วงเวลาในการสุ่ม จาก $\delta(t-nT)$ ในสมการที่ 1 คือ ฟังก์ชั่นคาบที่มีคาบเวลาเท่ากับ T สามารถแสดงได้ด้วย

อนุกรมฟูริเยร์

$$\sum_{n=-\infty}^{\infty} x(t)\delta(t-nT) = \frac{1}{T}\sum_{n=-\infty}^{\infty} x(t)e^{(j2n\pi t)/T} \quad \text{.....2.2}$$

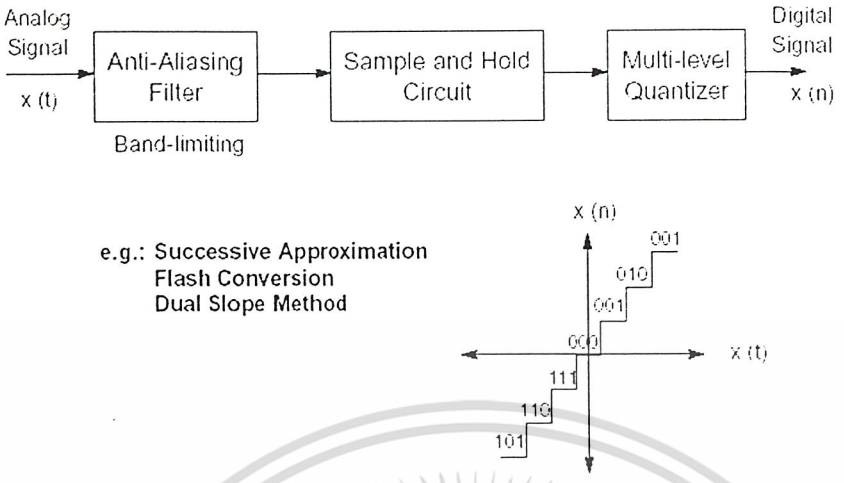
จากสมาการที่ 1 และ 2 จะได้ว่า

$$x^*(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty} x(t)e^{(j2n\pi t)/T} = \frac{1}{T}\sum x(t)e^{j2\pi f_s nt} \quad \text{..... .....2.3}$$

$$\sum_{n=-\infty}^{\infty} x(t)\,\delta(t - nT)$$

จากสมการข้างต้นจะสังเกตเห็นว่าจะเสมือนว่าเป็นการผสมสัญญาณ 2 ตัวเข้าด้วยกันคือ สัญญาณอินพุทและสัญญาณพาหที่มีความถี่เป็น 0, F_S, 2 F_S, ...นอกจากนี้แล้วสัญญาณที่ทำกาสุ่มสามารถแสดงในโดเมนของความถี่ดังแสดงในรูปที่ 4 โดยเสมือนว่าอนาลอกขึ่มากับขบวนพัลส์ ดังนั้นดังที่เคยได้กล่าวไปแล้วข้างต้นว่าสัญญาณอินพุทที่เกินความถี่ Nyquist จะสร้างสัญญาณตัวใหม่ขึ้นมาภายในช่วงความถี่ Base - Band ซึ่งไม่ปรากฏในสัญญาณอินพุท นี้คือผลของความไม่เป็นเชิงเส้นทำให้เกิดความผิดเพี้ยนของสัญญาณทางด้านเอาท์พุท

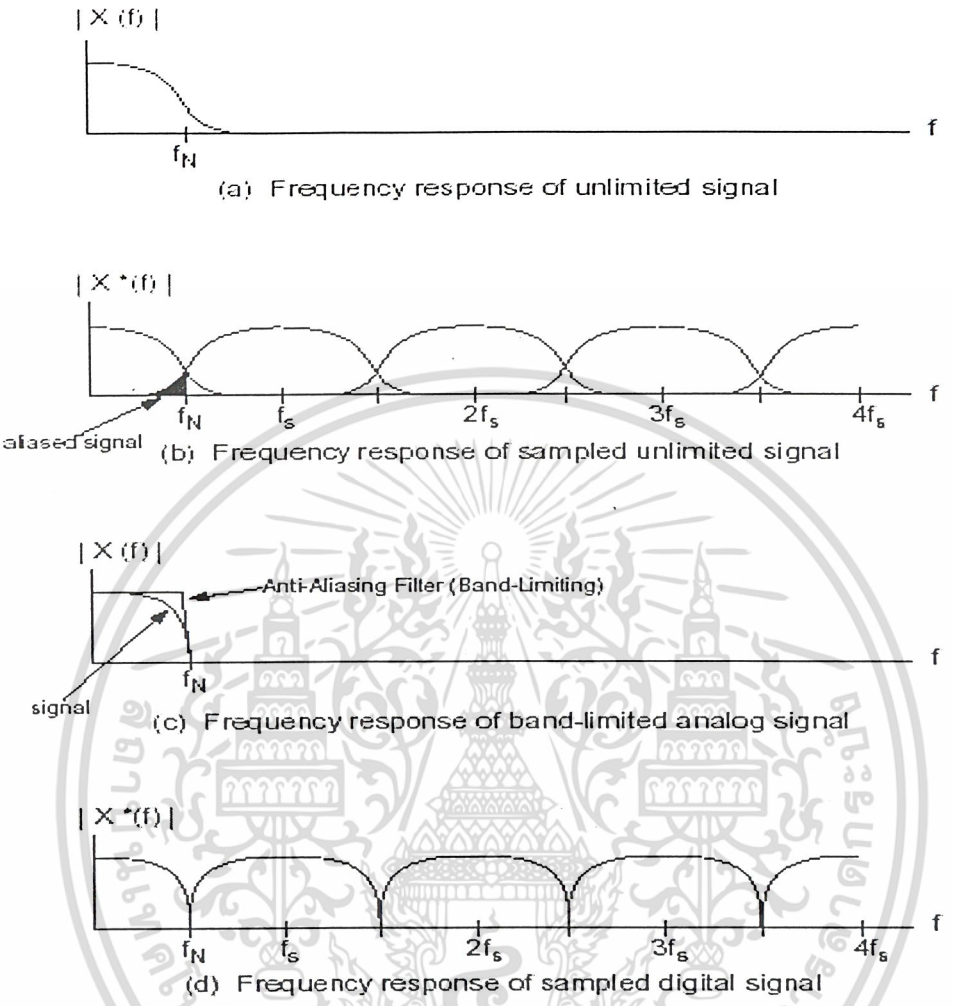
การป้องกันผลที่เกิดขึ้นมานี้ทำได้โดยการเพิ่มวงจรกรองความถี่ต่ำผ่าน เข้าไปเพื่อกรองอินพุทไม่ให้ความถี่เกินความถี่ไนควิสท์เข้ามาได้วงจรกรองความถี่ต่ำผ่าน ชุดนี้เรียกว่า Anti - Aliasing Filter จะให้ผลตอบสนองที่เรียบตลอดช่วงของความถี่ที่สนใจ (baseband) และลดทอนขนาดของสัญญาณที่มีความถี่มากกว่าความถี่ไนควิสท์ มากพอที่จะทำให้ขนาดของความถี่เหล่านั้นอยู่ในระดับของสัญญาณรบกวน อีกทั้งผลของความไม่เป็นเชิงเส้นของเฟสซึ่งเกิดจาก Anti - Aliasing Filter ความผิดเพี้ยนทางฮามอนิก (Harmonic Distortion) ดังนั้นแล้ว Analog anti - Aliasing Filter ซึ่งเป็นตัวควบคุมแบนด์วิดท์และความผิดเพี้ยนทางเฟสของสัญญาณอินพุทจำเป็นต้องใช้ anti - Aliasing ที่มีประสิทธิภาพสูงเพื่อให้เกิดความผิดเพี้ยนน้อยที่สุด



รูปที่ 2.3 กระบวนการแปลง อะนาลอกเป็นดิจิตอล

ในรูปที่ 2.3 นอกจากส่วนของ Anti - Aliasing Filter จำเป็นต้องมีส่วนของงวงจรสุ่มและคงค่า (Sampling –and- Hold) ทั้งนี้เนื่องจากว่าแม้สัญญาณอนาลอกจะมีการเปลี่ยนแปลงอย่างต่อเนื่อง, เอาท์พุทของวงจรสุ่มและคงค่า จะต้องคงที่ระหว่างการสุ่มทั้งนี้เพื่อให้ ตัวแปลงค่า (Converter) มีเวลาเพียงพอในการเปรียบเทียบระหว่างสัญญาณอินพุทและระดับอ้างอิง ถ้าเอาท์พุทวงจรสุ่มและคงค่า มีการเปลี่ยนแปลงในช่วงเวลาระหว่าง T จะทำให้เป็นตัวจำกัดประสิทธิภาพของการแปลงอะนาลอกเป็นดิจิตอลได้

ระดับอ้างอิงจะเป็นตัวกำหนดรหัสดิจิตอล ค่าอินพุทที่อ่านค่าเข้ามาจะถูกนำไปแปลงเป็นรหัสทางด้านดิจิตอลที่ใกล้เคียงกับอินพุทมากที่สุด โดยการเปรียบเทียบ ค่าความละเอียดของ การแปลงอะนาลอกเป็นดิจิตอล ของConverterถูกกำหนดโดย จำนวนของระดับอ้างที่อิงที่ถูกกำหนดขึ้น สำหรับ High - Resolution Nyquist Samples การสร้างระดับแรงดันอ้างอิงเป็นสิ่งที่มีความสำคัญมาก



รูปที่ 2.4 สเปกตรัมของ Analog และ สัญญาณที่สุ่ม

ตัวอย่างเช่น 16 - bit ของตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอล ซึ่งเป็นมาตรฐานของการแปลงสัญญาณอะนาลอกเป็นดิจิตอลความแม่นยำสูง (High accuracy A/D converter) ต้องการ 2^{16} -1 = 65535 ระดับอ้างอิงที่แตกต่างกัน ถ้าหากว่าอินพุทมีแรงดันเท่ากับ 1V หมายความว่าความละเอียดของ สัญญาณอะนาลอกเป็นดิจิตอลความแม่นยำสูง คือ 15uV ต่อการเปลี่ยนแปลง 1 บิท

2.2 ประเภทของวงจรแปลงสัญญาณทางอนาลอกเป็นดิจิตอล (Analog -to – Digital Converter)

การแปลงสัญญาณอะนาลอกเป็นดิจิตอลความแม่นยำสูง สามารถแบ่งออกได้ตามช่วงเวลาที่ใช้ในการแปลงสัญญาณได้เป็น 3 ชนิดใหญ่ๆ ดังนี้คือ

1) แบบ Dual-slope integrating ADC โดยวิธีการนี้จะใช้เวลาในการแปลงสัญญาณประมาณ 300 ms

2) แบบประมาณค่าด้วยวิธีซัคเซสซีฟ (Successive approximation ADC) วิธีนี้ใช้เวลาในการแปลงสัญญาณค่อนข้างรวดเร็วในหน่วยของ μs นิยมใช้ในการประมวลผลสัญญาณเสียงดิจิตอล

3) แบบเฟลช (Flash ADC) เป็นแบบที่ใช้เวลาน้อยที่สุด แต่ก็มีราคาแพงมากที่สุดเช่นกัน มักใช้ในการประมวลผลสัญญาณวีดีโอดิจิตอล

2.2.1 ADC แบบ Dual-Slope Integrating

ผังการทำงานของ ADC แบบ Dual-slope integrating แสดงดังรูปที่ 6 ประกอบด้วยวงจรลอจิกควบคุม เป็นส่วนที่ทำหน้าที่ที่ควบคุมลอจิกการทำงานของวงจร และควบคุมสวิตช์
อนาลอกเพื่อทำการแปลงแรงดันอนาลอกอินพุทให้เป็นสัญญาณดิจิตอลเอาท์พุท โดยมีวงจรสร้างสัญญาณนาฬิกาที่มีความถี่เท่ากับ 12 KHz ป้อนให้กับวงจรลอจิกควบคุม ซึ่งความถี่ของสัญญาณนาฬิกานี้สามารถแปรค่าได้โดยการกำหนดค่า R_T และ C_T จากภายนอก กระบวนการแปลงสัญญาณ
อนาลอกเป็นสัญญาณดิจิตอลแบบนี้สามารถแบ่งเฟสในการทำงานได้ 3 เฟส คือ

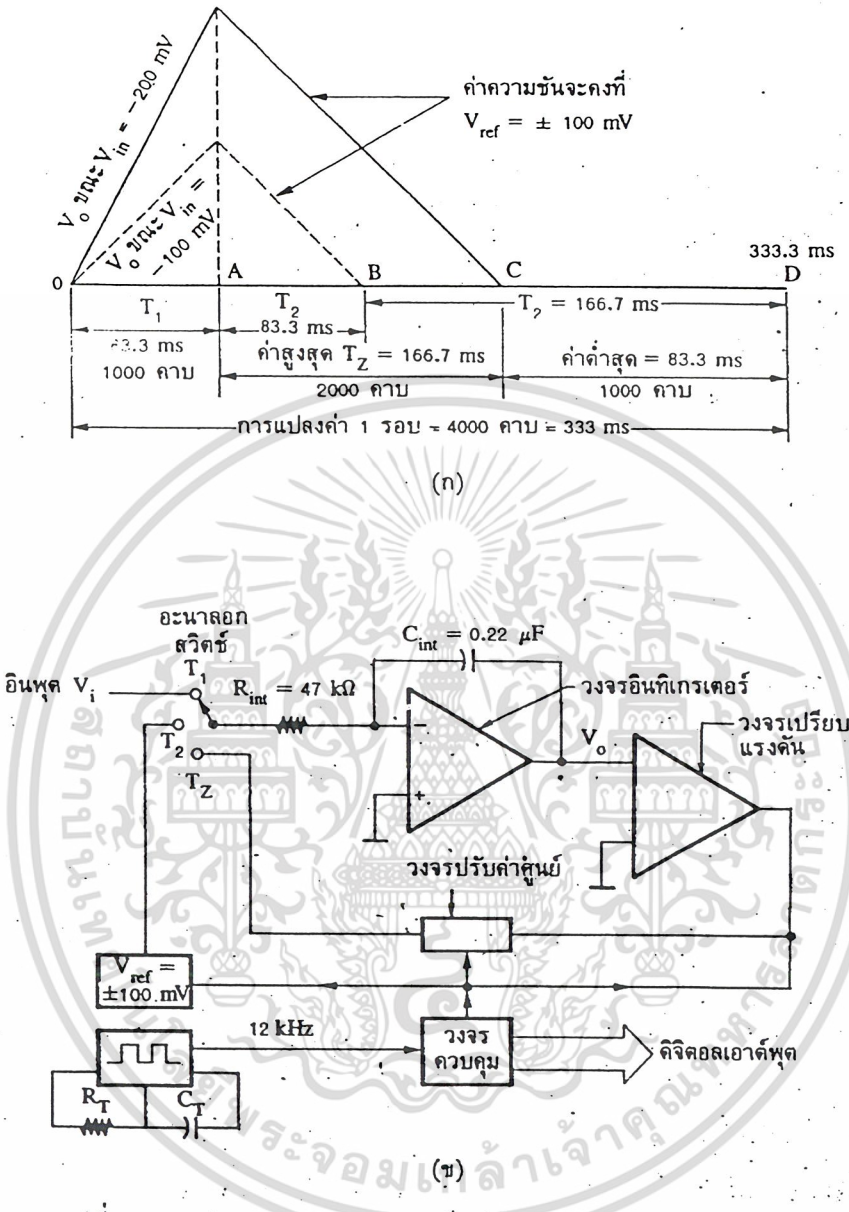
1) เฟสสัญญาณ (Signal Integrate Phase, T1)
2) เฟสอ้างอิง (Reference Integrate Phase, T2)
3) เฟสศูนย์อัตโนมัติ (Auto-Zero, Tz)

1 เฟสสัญญาณ (Signal Integrate Phase, T1)

การทำงานในเฟส T1 ของวงจรเริ่มต้นจากวงจรลอจิกควบคุมในรูปที่ 6(ข) จะทำการควบคุมให้สวิตช์อนาลอกต่อวงจรเพื่อป้อน Vin ให้กับวงจรอินทิเกรเตอร์ โดยแรงดันเอาท์พุทที่ได้จากวงจรอินทิเกรเตอร์นี้จะมีลักษณะเห็นสัญญาณลาด และมีทิศทางลาดขึ้นหรือลาดลงนั้นก็ขึ้นอยู่กับทิศทางของ Vin นั่นเอง ในขณะที่ความชันของสัญญาณลาดมีค่าขึ้นอยู่กับค่าของ Rint และ Cint หรือค่าคงที่เวลา (time constant)ของวงจร

2 เฟสอ้างอิง (Reference Integrate Phase, T2)

ขณะที่ทำงานในเฟส T_1 นั้นวงจรลอจิกควบคุมจะทำการประจุเข้าตัวเก็บประจุ C_{REF} (ไม่ได้แสดงไว้ในรูป) ซึ่งอยู่ภายในวงจรสร้างแรงดันอ้างอิง V_{REF} เอาไว้ด้วย เพื่อให้ V_{REF} มีค่าเป็น +100mV ดังนั้นเมื่อมีการเริ่มต้นทำงานในเฟส T_2 วงจรลอจิกควบคุมทำการควบคุมสวิตช์อนาลอกให้ต่อวงจรป้อน VREF (ซึ่งมีทิศทางตรงกันข้ามกับ Vin) ให้กับวงจรอินทิเกรเลอร์แรงดัน



รูปที่ 2.5 การทำงานของ อะนาลอกเป็นดิจิตอล (แบบ Dual – Slope)

เอาท์พุท จึงมีลักษณะเป็นสัญญาณลาดที่มีทิศทางลาดลงจรมีค่าเป็นศูนย์ และเนื่องจาก V_{REF} มีค่าคงที่ทำให้สัญญาณลาดที่ได้จากวงจรมีความชันคงที่ ดังแสดงดังรูปที่ 2.5 (ก) หลังจาก Vout มีค่าลดลงจนเป็นศูนย์ วงจรเปรียบเทียบสัญญาณ จะทำการส่งสัญญาณให้กับวงจรลอจิกควบคุมเพื่อบอกให้รู้ว่าสิ้นสุดการทำงานในช่วง T_2 แล้ว และเริ่มต้นเข้าสู่การทำงานในเฟสศูนย์อัตโนมัติ Tz ต่อไป จากการทำงานในเฟส T_2 และกราฟสัญญาณในรูปที่ 2.5 (ก) พบว่าช่วงเวลา T2 นั้นมีค่าเป็นสัดส่วนโดยตรงกับ Vout สามารถสรุปความสัมพันธ์ได้ดังนี้

$$T_2 = T_1\left(\frac{V_{in}}{V_{REF}}\right)$$

3 เฟสศูนย์อัตโนมัติ (Auto-Zero, Tz)

เมื่อเริ่มต้นการทำงานในเฟสที่สามหรือเฟสสุดท้ายของการแปลงสัญญาณ (Tz) วงจรลอจิกควบคุมจะควบคุมให้สวิตช์อนาลอกต่อเข้ากับส่วนของวงจร Auto-Zero เพื่อทำกาประจุให้

กับตัวเก็บประจุ D_{AZ} (Auto-Zero Capacitor) ซึ่งอยู่ภายในวงจร C_{AZ} จะถูกประจุให้แรงดันตกคร่อมตัวมันมีค่าประมาณแรงดันคลาดเคลื่อนเฉลี่ย (average error voltage) ซึ่งเกิดจากวงจรอินทิเกรเตอร์และผลของแรงดันออฟเซต ซึ่งเกิดจากวงจรเปรียบเทียบสัญญาณ ดังนั้นหลังจากช่วงเวลา T_1 และ T_2 ผ่านพ้นไปแล้ว ค่าแรงดันคลาดเคลื่อนซึ่งประจุไว้ใน C_{AZ} จึงถูกต่อเข้ากับวงจรเพื่อหักล้างผลของ

แรงดันคลาดเคลื่อนที่สะสมไว้ใน C_{REF} ทำให้สัญญาณจาก แปลงสัญญาณอะนาลอกเป็นดิจิตอล มีค่าเป็นศูนย์อย่างอัตโนมัติในทุกๆ ครั้งหลังจากการแปลงสัญญาณเสร็จสิ้นลง

2.2.2 <u>การแปลงอะนาลอกเป็นดิจิตอล แบบประมาณค่าด้วยวิธีซัคเซสซีฟ (Successive approximation ADC)</u>

ผังการทำงานของ การแปลงสัญญาณอะนาลอกเป็นดิจิตอล (แบบประมาณค่าด้วยวิธีซัคเซสซีฟ แสดงดังรูปที่ 2. ซึ่งประกอบด้วยวงจร การแปลงสัญญาณดิจิตอลเป็นอะนาลอก วงจรเปรียบเทียบสัญญาณ และรีจิสเตอร์แบบการประมาณค่าต่อเนื่อง

(Successive approximation register) หรือ SAR โดยมีขั้วรับสัญญาณอินพุทที่เป็นแรงดันอนาลอกอินพุท หนึ่งขั้วสัญญาณ และมีสัญญาณดิจิตอลเอาท์พุทที่สามารถนำออกไปใช้งานได้ทั้งแบบขนาน และแบบอนุกรม สัญญาณควบคุมการทำงานของระบบประกอบด้วยสัญญาณสามชุด คือ

1. สัญญาณเริ่มต้นการแปลงสัญญาณ (Start of Conversion) เป็นสัญญาณที่ป้อนให้กับ ADC เพื่อสั่งเริ่มต้นกระบวนการแปลงสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอล

2. สัญญาณสิ้นสุดการแปลงสัญญาณ (End of Conversion) เป็นสัญญาณที่ป้อนให้กับ ADC เพื่อบอกให้รู้ว่ากระบวนการแปลงสัญญาณได้เสร็จสิ้นสมบูรณ์แล้ว

3. สัญญาณนาฬิกาภายนอก (Clock in) เป็นสัญญาณนาฬิกาจากภายนอกที่ป้อนให้กับ ADC เพื่อกำหนดฐานเวลาอ้างอิงให้กับกระบวนการแปลงสัญญาณ

1 หลักการทำงาน (Circuit Operation)

จากรูปที่ 2.6(ก) กระบวนการแปลงสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอลเริ่มต้นจากสัญญาณคำสั่งเริ่มต้นการแปลงสัญญาณ (Start of Conversion) จะเป็นสัญญาณกระตุ้นให้การแปลงสัญญาณอะนาลอกเป็นดิจิตอล เริ่มต้นกระบวนการแปลงสัญญาณในไซเคิลแรก รีจิสเตอร์ SAR ซึ่งต่อตรงกับ ดิจิตอลเป็นอะนาลอกโดยเรียงข้อมูลตามลำดับบิทแต่ละบิทของ ดิจิตอล

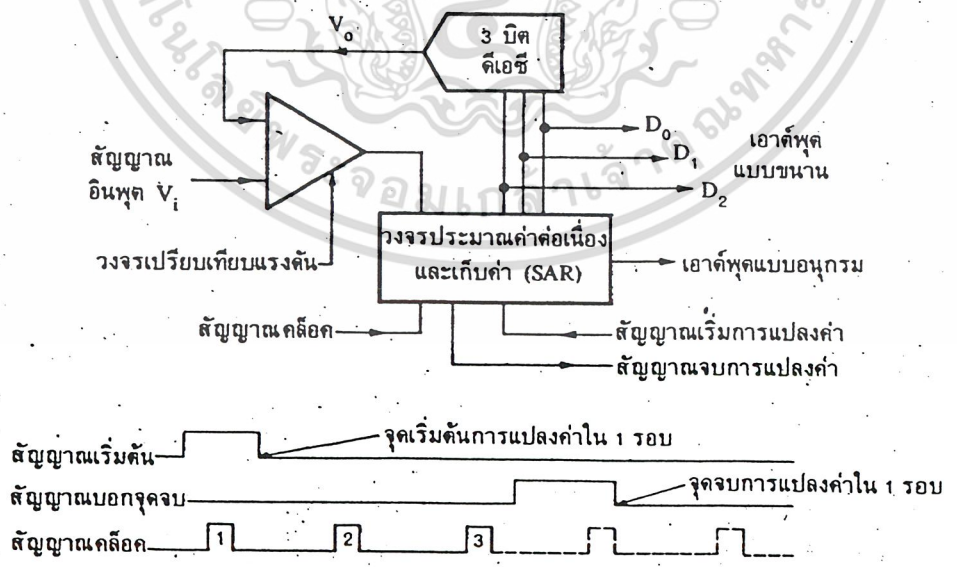
เป็นอะนาลอก จากนั้นดิจิตอลเป็นอะนาลอก ทำการเปลี่ยนข้อมูลดิจิตอลในรีจิสเตอร์ SAR ให้เป็นแรงดันอนาลอกเอาท์พุท V_{out}

หลังจากนั้นวงจรเปรียบเทียบสัญญาณจะนำแรงดันอนาลอกเอาท์พุท Vout ไปเปรียบเทียบกับแรงดันอนาลอกอินพุท Vin ของวงจร สัญญาณเอาท์พุทที่ได้จากวงจรเปรียบเทียบสัญญาณจะเป็นสัญญาณบอกให้รีจิสเตอร์ SAR ทราบว่า Vin มีค่ามากกว่าหรือน้อยกว่า Vout ที่ได้จาก DAC เพื่อนำค่านั้นไปเปรียบเทียบแล้วส่งออกไปเป็นสัญญาณดิจิตอลเอาท์พุทของวงจร กรณีดิจิตอลเอาท์พุทขนาด 3 บิทแล้วการเปรียบเทียบจะเกิดขึ้น 3 ครั้ง

โดยการเปรียบเทียบเริ่มต้นจากบิท □SB ไปจนสิ้นสุดที่บิท LSB เมื่อการเปรียบเทียบที่บิท LSB สิ้นสุดลง SAR จะส่งสัญญาณหยุดกระบวนการ(End of Conversion) ออกมาเพื่อบอกให้ทราบว่าขณะนี้ขั้นตอนการแปลงสัญญาณของ กระบวน การแปลงสัญญาณอะนาลอกเป็นดิจิตอล ได้เสร็จเรียบร้อยแล้ว และสัญญาณดิจิตอลเอาท์พุทของวงจรซึ่งเป็นค่าแปรผันตรงมาจาก Vin ก็จะปรากฏที่เอาท์พุทของ SAR

2 การประมาณค่าด้วยวิธีซัคเซสซีฟ (Successive Approximation Analogy)

จากแผนผังแสดงหลักการประมาณค่าด้วยวิธีซัคเซสซีฟขนาด 3 บิท ดังรูปที่ 2.6 เมื่อสัญญาณ Start เริ่มขึ้นกระบวนการแปลงสัญญาณก็เริ่มต้นเช่นกัน ค่าน้ำหนักของบิทแต่ละบิทของ SAR จะถูกวงจรเปรียบเทียบสัญญาณนำค่าไปเปรียบเทียบกับ Vin ดังรูปที่ 2.6 โดยเริ่มเปรียบเทียบจากบิท □SB (D_2) ไปจนถึงบิท LSB (D_0) หาก Vin มีค่ามากกว่าสัญญาณเอาท์พุทของ SAR จะถูกปรับให้เป็นลอจิก 0



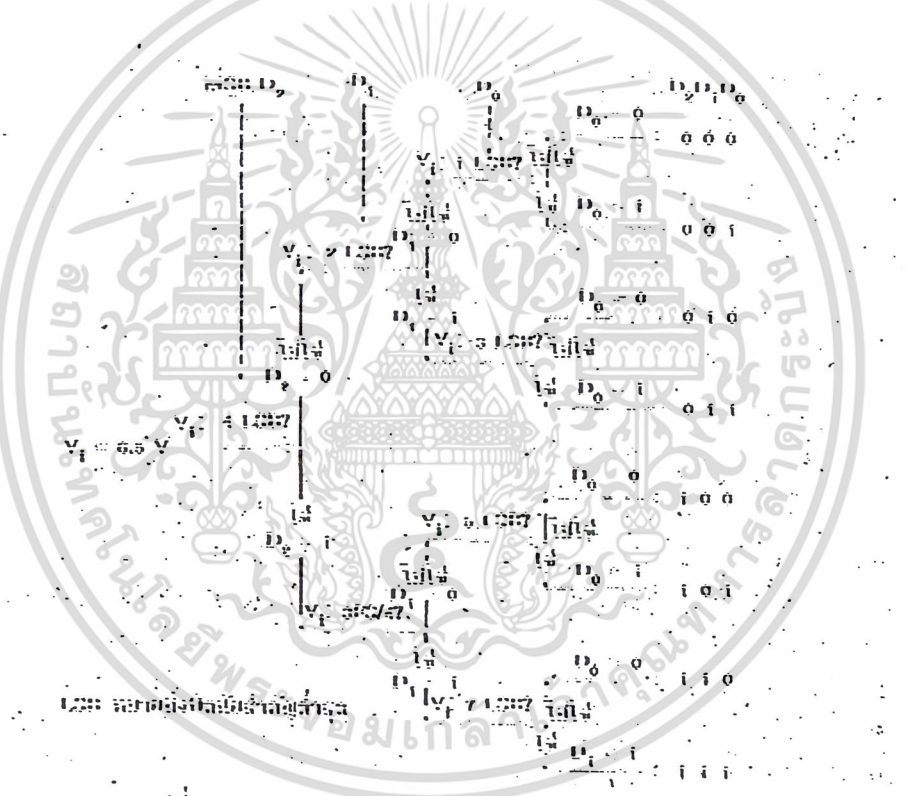
รูปที่ 2.6(ก) แผนผังแสดงหลักการประมาณค่าด้วยวิธีซัคเซสซีฟขนาด 3 บิท

3 ช่วงเวลาของการแปลงสัญญาณ (Conversion Time)

จากหลักการแปลงสัญญาณโดยใช้หลักการประมาณค่าด้วยวิธีซัคเซสซีฟ ดังรูปที่ 2.6 จะเห็นว่า SAR ต้องใช้สัญญาณนาฬิกาหนึ่งลูกเพื่อใช้ในการเปรียบเทียบในแต่ละบิทดังนั้นช่วงเวลาที่ใช้สำหรับการแปลงสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอลในหนึ่งรอบนั้นจึงมีค่าขึ้นอยู่กับคาบเวลาของสัญญาณนาฬิกาที่ใช้รวมทั้งจำนวนบิทของวงจรอีกด้วย ซึ่งสามารถเขียนความสัมพันธ์ได้เป็น

$$Tc = T \times (n+1)$$

โดยที่ Tc คือ ช่วงเวลาที่ใช้สำหรับการแปลงสัญญาณ T คือ คาบเวลาของสัญญาณนาฬิกาที่ใช้ในวงจร และ n คือ จำนวนบิทของวงจร แปลงสัญญาณอะนาลอกเป็นดิจิตอล



รูปที่ 2.6(ข) กระบวนการในการประมาณค่า

2.2.3 การแปลงสัญญาณอะนาลอกเป็นดิจิตอล แบบเฟลช (Flash ADC)

1 หลักการทำงาน (Circuit Operation)

การแปลงสัญญาณอะนาลอกเป็นดิจิตอล ที่มีช่วงเวลาที่ใช้ในการแปลงสัญญาณรวดเร็วที่สุดในบรรดา วิธีแปลงอะนาลอกเป็นดิจิตอลแบบต่างๆ ก็คือ อะนาลอกเป็นดิจิตอล แบบเฟลช หรือแบบขนาน ซึ่งมีผังการทำงานพื้นฐานของวงจรดังรูปที่ 2.8 แรงดันอ้างอิง V_{REF} และส่วนของตัวต้านทานแบ่งแรงดันทำให้ค่ารีโซลูชันของวงจรเท่ากับ 1 V/LSB ขาอินพุทบวกของวงจรเปรียบเทียบแรงดันทุกตัวทำหน้าที่เป็นขารับแรงดันอนาลอกอินพุท Vin ของวงจรเพื่อนำไป

เปรียบเทียบกับค่าของแรงดันที่ได้จากการแบ่งแรงดันในแต่ละโหนด จากนั้นแรงดันเอาท์พุทของวงจรเปรียบเทียบแรงดันจะป้อนเข้าวงจรเข้ารหัส (8-to-3 line encoder) เพื่อนำสัญญาณเอาท์พุทออกไปใช้งานเพียง 3 เส้น (เนื่องจากเป็นวงจร ADC ที่มีขนาด 3 บิท) โดยที่สัญญาณดิจิตอลเอาท์พุท 3 บิทที่ได้มีผลเป็นสัดส่วนโดยตรงกับค่าของแรงดันอนาลอกอินพุท Vin ที่ป้อนให้กับวงจรนั่นเอง

2 ช่วงเวลาของการแปลงสัญญาณ (Conversion Time)

ช่วงเวลาที่ใช้ในการแปลงสัญญาณของ ADC แบบเฟลชนั้นขึ้นอยู่กับสมรรถนะผลตอบสนองทางเวลาของวงจงเปรียบเทียบแรงดัน วงจรลอจิก และวงจรเกทต่างๆที่ใช้ในวงจร หรือ กล่าวได้ว่าหาก การแปลงสัญญาณอะนาลอกเป็นดิจิตอล แบบเฟลชที่มีความเร็วสูงมากขึ้นเท่าใด ราคาก็แพงและค่ารีโซลูชันของวงจรก็มากขึ้นตามไปด้วยเช่นกัน จากรายละเอียดของวงจรเปรียบเทียบแรงดันจำนวน 7 ตัว ดังนั้นจึงสามารถสรุปได้ว่าจำนวนวงจรเปรียบเทียบแรงดันที่ต้องใช้ในวงจร แปลงสัญญาณอะนาลอกเป็นดิจิตอล แบบเฟลชขนาด n บิท มีค่าเท่ากับ

$$\text{Number of comparators} = 2^n - 1$$

3 ผลตอบสนองทางความถี่ของวงจร แปลงสัญญาณอะนาลอกเป็นดิจิตอล

สำหรับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล แล้วความถี่ปฏิบัติงานสูงสุดของแรงดันอนาลอกอินพุท Vin รูปคลื่นซายน์ที่จะแปลงให้สัญญาณดิจิตอลเอาท์พุทที่มีความแม่นยำ ± ½ LSB มีค่าเท่ากับ

$$F_{MAX} = \frac{1}{(2\pi T_C 2^n)}$$

โดยที่ Tc คือ ช่วงเวลาที่ใช้ในการแปลงสัญญาณหนึ่งรอบของ ADC และ n คือ จำนวนบิทของวงจร แปลงสัญญาณอะนาลอกเป็นดิจิตอล

การควอนไตซ์เป็นกระบวนการที่แปลงสัญญาณอนาลอกเป็นสัญญาณที่ไม่ต่อเนื่องทางเวลา (Discrete Signal) หลังจากสุ่ม โดยผ่านกรับวนการเข้ารหัส (Coding) จัดให้สัญญาณที่ไม่ต่อเนื่องทางเวลานั้นอยู่ในรูปที่ของจำนวนเพื่อง่ายต่อการนำไปประมวลผลและเป็นสัดส่วนสัมพันธ์กับสัญญาณอนาลอกดัง รูปที่ 2.7โดยรูปกราฟแสดงให้เห็นถึงความสัมพันธ์กันระหว่างสัญญาณอนาลอกที่ขนาดอยู่ระหว่าง 0 ถึง 15 โวลต์ ถูกประมาณค่า (Quantized) และ เข้ารหัส (Encode) เป็นรหัสไบนารี (Binary) 3 บิทได้ 000 ถึง 111 เนื่องจากในระบบไบนารี่ รหัสดิจิตอลแต่ละค่าจะแทนขนาดของสัญญาณอนาลอกแต่ละค่าที่เป็นสัดส่วนกับค่าเต็มสเกล โดยค่าสูงสุดของรหัสดิจิตอลคือทุกบิทเป็น 1จะเท่ากับสัญญาณอนาลอกเต็มสเกลคูณด้วย ($1-2^n$) โดย n เป็นจำนวน

บิทของรหัสดิจิตอลและดิจิตอลแต่ละบิทที่เป็น 1จะเท่ากับขนาดเต็มสเกลของอนาลอกคูณด้วยค่าน้ำหนัก (Weighting) ของรหัสดิจิตอลนั้นหารด้วย 2^n

จำนวนสถานะเอาท์พุทกำหนดได้จากจำนวนบิทคือเท่ากับ 2^n ตัวอย่างกรณี การแปลงสัญญาณอะนาลอกเป็นดิจิตอล 8 บิท ตัวประมาณค่า จะให้เอาท์พุท 256 สถานะ และ 12 บิท ให้ 4096 สถานะต่อค่าเต็มสเกลของอนาลอกในไดอแกรมแสดงทรานสเฟอร์ฟังก์ชั่น จะเห็นจุดแบ่ง (Decision point หรือ Threshold level) สัญญาณอนาลอกจะมีจำนวน 2^n -1 จุด 0.625 , 1.875 , 3.125 , 4.375 , 5.625 , 6.875 และ 8.125 โวลต์ ระหว่างจุดดังกล่าวเป็นสัญญาณอนาลอกซึ่งแปลงเป็นรหัสดิจิตอล 1 สถานะ ดังนั้นค่าเหล่านี้ต้องปรับให้ถูกต้องมากที่สุดเพื่อแปลงขนาดของอนาลอกให้ตรงกับค่าที่ ประมาณ แรงดันที่ 1.25 , 2.50 , 3.75 , 5.0 , 6.25 , 7.2 และ 8.75 โวลต์เป็นจุดกึ่งกลางในช่วงของสัญญาณอนาลอกที่แสดงสถานะเอาท์พุทดิจิตอล ฟังก์ชั่นที่มีลักษณะเป็นขั้นบันไดสามารถประมาณเป็นเส้นตรงได้โดยการโยงเส้นตรงระหว่างจุดเริ่มและจุดปลาย ณ จุดกึ่งกลางของรหัสดิจิตอลสถานะสุดท้าย สังเกตว่าในทางทฤษฎีแล้วเส้นตรงนี้จะผ่านจุดกึ่งกลางของทุกระดับดิจิตอล

4.Quantizer Resolution and Error

ในแต่ละสถานะของสัญญาณดิจิตอลเอาท์พุทจะแทนขนาดของสัญญาณอนาลอกค่าใดค่าหนึ่งในช่วงเล็กๆ ระหว่างจุดแบ่งระดับ เรียกช่วงเล็ก ๆ นี้ว่าเป็นขนาดหนึ่ง Analog Quantization หรือ หนึ่งควันตัม (Quantum) หรือเรียกอีกอย่างว่า 1 LSB (Least Significant Bit) ของการแปลงสัญญาณ ดังแสดงในรูปที่ 7 ควอนตัม คือ 1.875 ค่านี้หาได้จากการคำนวณ

$$Q = \frac{FSR}{2^n}$$

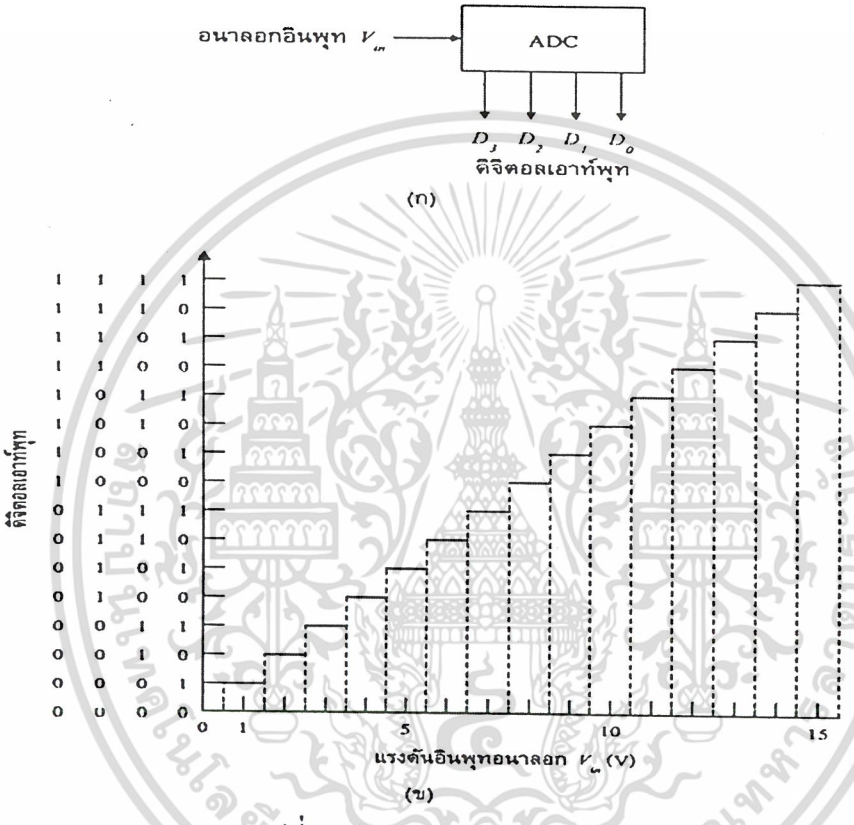
FSR = ช่วงเต็มสเกลของแรงดันอนาลอก (Full Scale Range)

n = จำนวนของรหัสดิจิตอล

จากสมการจะเห็นว่า หากการแปลงที่ให้จำนวนบิทมากขนาดของควอนตัมก็จะลดลง และถ้าให้สัญญาณอินพุทของ ตัวประมาณค่ากวาดไปตลอดช่วงของสัญญาณอนาลอกก็จะเห็นช่วงของผลต่างของอนาลอกอินพุทและดิจิตอลเอาท์พุทเป็นช่วงและพล็อตกราฟออกมาได้เป็นรูปฟันเลื่อย เรียกว่า ความผิดพลาดจากการประมาณค่า(Quantizing Error) ซึ่ง ความผิดพลาด นั้นก็คือ 1 ช่วงสัญญาณอนาลอกแปลงเป็นรหัสดิจิตอล 1 สถานะดังกล่าวมาแล้วนั่นเอง

เนื่องจากกระบวนการเปลี่ยนสัญญาณอนาลอก (กำหนดให้มีค่า Resolution เท่ากับ Infinity) เป็นช่วงของตัวเลขที่มีค่าจำกัด (Quantization) ดังนั้น ความผิดพลาด นี้ซึ่งเป็นธรรม

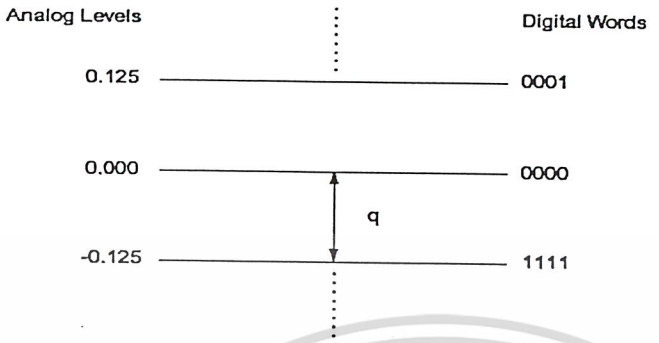
ชาติของ การประมาณค่า ซึ่งไม่สามารถแก้ไขได้นอกจากการแก้ไขให้น้อยลงโดยการเพิ่มจำนวนบิทของ ตัวประมาณค่าให้มากขึ้น เอาท์พุท ความผิดพลาด จะอยู่ระหว่าง Q –Q/2 ความผิดพลาดอาจเป็นศูนย์เมื่อสัญญาณอนาลอกค่าที่จุดกึ่งกลางของควอนตัมพอดี ลักษณะฟังก์ชั่นของ ความผิดพลาด จะสามารถพิจารณาเป็นสัญญาณรบกวนทางอินพุท ซึ่งมีค่าเป็น Q V_{P-P} และค่าเฉลี่ยเป็นศูนย์ ค่า rms เป็น $\frac{Q}{2\sqrt{3}}$ ซึ่งได้จากการวิเคราะห์ในรูปคลื่นฟันเลื่อย



รูปที่ 2.7 ลักษณะการ ประมาณค่า

เมื่อสัญญาณอินพุทถูกสุ่มเข้ามาให้อยู่ในรูปลำดับของตัวเลข แต่ละค่าที่อ่านเข้ามาจะถูกเข้ารหัสโดยใช้จำนวนบิทที่จำกัดรวมทั้งบิทเครื่องหมาย ดังเช่นว่าใช้การแปลงที่มีความยาวของรหัสเลขฐานสองเท่ากับ B - บิท จำนวนของระดับที่สามารถแบ่งได้ของสัญญาณ x(n) มีทั้งสิ้น 2^B ช่วงของความห่างแต่ละระดับให้เท่ากับ q เรียกว่า ระดับการประมาณค่า(Quantization Step Size) ข้อมูลที่ถูกสุ่มเข้ามา x*(t) จะถูกปัดเข้าหาจำนวนที่ใกล้เคียงมากที่สุด

$$q = \frac{1}{2^B - 1}$$



รูปที่ 2.8 Quantization Error

จากรูปที่ 2.8 จะเห็นว่าจะเกิดความไม่ต่อเนื่องของข้อมูลดิจิตอล ดังเช่นเมื่อขนาดของสัญญาณอนาลอกที่มีค่าอยู่ระหว่าง 0 .12 กับ 0.125 ค่ารหัสดิจิตอลที่ได้ออกมาจะมีค่าเท่ากันทำให้เอาท์พุทที่ได้มีความแตกต่างไปจากอินพุท หากใช้รหัสดิจิตอลขนาด 4 บิทดังรูป ความห่างกันของแต่ละระดับอ้างอิง (Quantization Step Size) จะเท่ากับ 0.125 Vและมีสถานะเอาท์พุท 16 สถานะ ถ้าเพิ่มจำนวนบิทของรหัสดิจิตอลเป็น 8 บิทความห่างกันของแต่ละระดับอ้างอิงจะเท่ากับ 0.00392Vและ 256 สถานะเอาท์พุท ดังนั้นจึงกล่าวได้ว่าจำนวนสถานะเอาท์พุทกำหนดจากจำนวนบิท คือเท่ากับ 2^B

เนื่องมาจากมีการปัดค่าเข้าหาค่าที่ใกล้ที่สุด ดังนั้นจึงทำให้เกิดสัญญาณที่ไม่ต้องการขึ้นมาดังสมการต่อไปนี้

$$x(n) = x^*(t) + e(n) \qquad (2.4)$$

ตามสมการที่ 2.4 เอาท์พุทของการแปลงอะนาลอกเป็นดิจิตอล จะประกอบด้วยสัญญาณอินพุทที่ถูกสุ่ม $x^*(t)$ และความผิดพลาด (Quantization Noise), e (n) สำหรับสัญญาณอินพุทที่มีขนาดใหญ่เมื่อเทียบกับ ระดับประมาณค่า

ขนาด ความผิดพลาดในเทอม e (n) เป็นปริมาณที่ไม่แน่นอนในช่วง (-q/2 , q/2) ซึ่งมีคาเท่ากับความน่าจะเป็น ดังนั้นค่ากำลังของสัญญาณรบกวนสามารถหาได้จาก

$$\sigma_e^2 = E(e^2) = \frac{1}{q}\int_{-q/2}^{q/2} e^2 de = \frac{q^2}{12} = \frac{2^{-2B}}{3} \qquad \text{.....2.5}$$

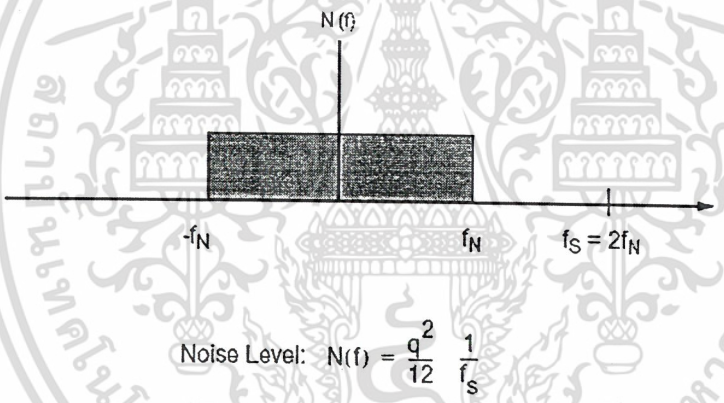
สัญญาณที่ถูก ประมาณค่า ด้วยความถี่ของการสุ่มเท่ากับ FS มีกำลังของสัญญาณรบกวนกระจายอยู่ตลอดช่วงของความถี่แบนด์ $0 < f < f_s / 2$ สมมติให้สัญญาณรบกวนเป็นปริมาณที่ ไม่มีค่าแน่นอน ความหนาแน่นสเปกตรัม ของสัญญาณรบกวนเท่ากับ

$$E(f) = e_{RMS}\left(\frac{2}{f_S}\right)^{\frac{1}{2}} \text{.....} \left(\frac{V}{\sqrt{Hz}}\right) \text{.....2.6}$$

$$n_0^2 = e_{RMS}^2\left(\frac{2f_0}{f_S}\right) \qquad (V^2) \text{.....2.7}$$

เปลี่ยนกำลังของสัญญาณรบกวนจากสมการข้างต้น โดยการยกกำลัง 2 และอินทิเกรต ตลอดช่วงความถี่ที่สนใจ (f_0) ได้ผลดังนี้

$$n_0 = e_{RMS}\left(\frac{2f_0}{f_S}\right)^{\frac{1}{2}} \qquad (V) \text{.....2.8}$$



รูปที่ 2.9 สเปกตรัม สัญญาณรบกวน ของ อัตราการสุ่มไนควิส์ท

รูปที่ 2.9 แสดงสเปกตรัมของสัญญาณรบกวนที่เกิดในการบวนการ ประมาณค่า จะพบว่ากำลังของสัญญาณรบกวนจะพบตลอดช่วงความถี่ทั้งหมด ระดับกำลังสัญญาณรบกวน

$$N(f) = \frac{q^2}{12f_S} = \frac{2^{-2B}}{3f_S}$$

แสดงได้ด้วย

2.3 ซิกมา – เดลตา มอดดูเลชั่น Quantization Noise

จากทฤษฎีของสัญญาณรบกวนและการสุ่มข้อมูล สามารถหา Differential Equation ของเอาท์พุทได้เป็น

$$y_i = x_{i-1} + (e_i - e_{i-1})$$

โดยที่ e คือ สัญญาณรบกวนที่เกิดจากการประมาณค่า

ความหนาแน่นของสเปกตราสัญญาณรบกวน ($n_i = e_i - e_{i-1}$) แสดงด้วย

$$E(f) = N(f)\left[1 - e^{-j\omega/F_S}\right] = 2e_{RMS}\left(\frac{1}{2f_S}\right)^{\frac{1}{2}} \sin(\frac{\omega}{2f_S}).....\left(\frac{V}{\sqrt{Hz}}\right)$$

กำลังสัญญาณรบกวนในช่วงความถี่ที่สนใจ

$$n_0^2 = e_{RMS}^2 \frac{\pi^2}{3}\left(\frac{2f_0}{f_S}\right)^3 .....(V^2)$$

หรือ

$$n_0 = e_{RMS} \frac{\pi}{\sqrt{3}}\left(\frac{2f_0}{f_S}\right)^{\frac{3}{2}} .....(V)$$

จากสมการข้างต้นพบว่าเมื่อเพิ่มความถี่ในการสุ่ม (การเพิ่ม OSR (Oversampling Ratio: $\frac{f_S}{2f_0}$) Over sampling อธิบายไว้ในหัวข้อถัดไป) ด้วยเฟ็คเตอร์ 2 จะลดสัญญาณรบกวนในช่วงความถี่ที่สนใจ ด้วยค่า 9dB พิจารณาอีกครั้งในมอดดูเลชั่นอันดับที่ 2

$$n_0 = e_{RMS} \frac{\pi}{\sqrt{5}}\left(\frac{2f_0}{f_S}\right)^{\frac{5}{2}} ..... (2.9)$$

เช่นเดียวกันว่าเมื่อเพิ่มค่า f_S ด้วยเฟ็คเตอร์สองกำลังสัญญาณรบกวนในช่วงความถี่แบนด์จะลดลงด้วยค่า 15 dB และสูตรโดยทั่วไปสำหรับกำลังสัญญาณรบกวนในแบนด์หาได้จาก

$$n_0 = e_{RMS}\left(\frac{\pi^M}{\sqrt{2M+1}}\right)\left(\frac{2f_0}{f_S}\right)^{M+\frac{1}{2}} .....(210)$$

และความถี่ในการสุ่ม (Sampling) เพิ่มขึ้น 2 เท่า จะลด Quantization Noise ลงด้วยค่า 3(2M+1) เมื่อ M คืออันดับของมอดดูเลชั่น ดังนั้นเราสามารถเขียนสมการของอัตราส่วนระหว่างสัญญาณรบกวนกับสัญญาณอินพุท (SNR: Signal To Noise Ratio) ทั้งแบบ Nyquist และ Oversampling

Nyquist Conversion Quantization Noise Spectrum

$$SNR = (6.02N + 1.76)dB \quad .....(2.11)$$

Over sampling Conversion Quantization Noise Spectrum

$$SNR = (6.02N + 1.76)dB + 10Log_{10}\left(\frac{f_S}{2f_C}\right)dB \quad \text{.....(2.12)}$$

เมื่อ

N = จำนวนบิทเอาท์พุท

f_S = ความถี่สุ่ม (Sampling)

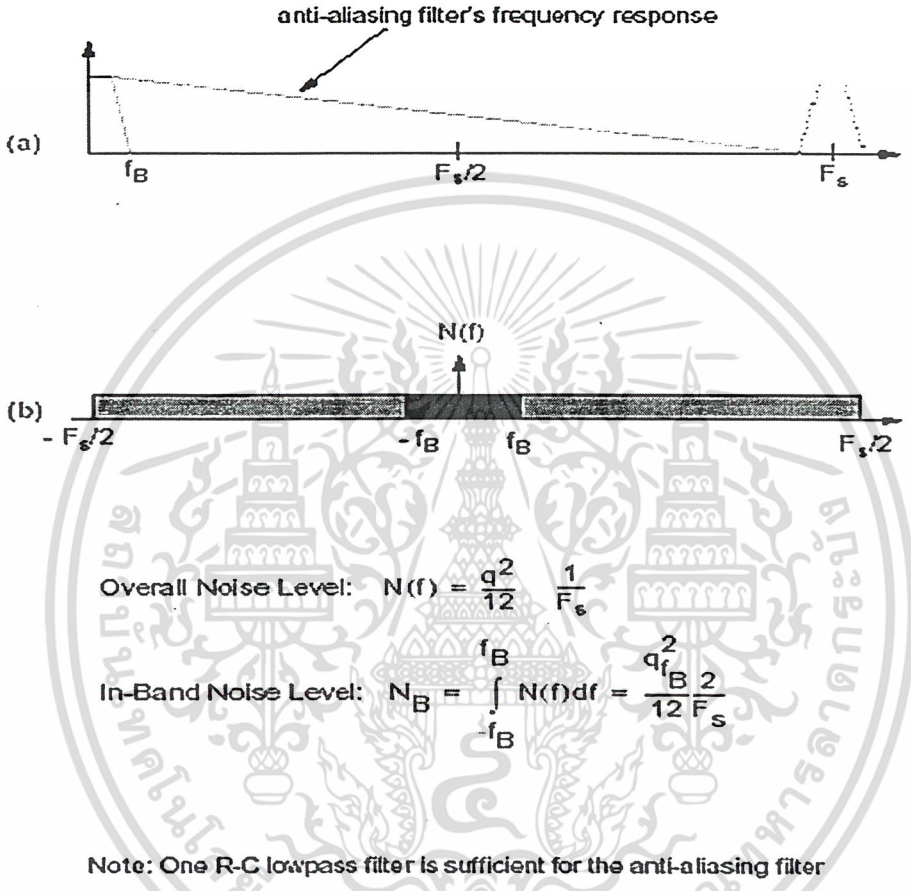
f_C =ความถี่อินพุทสูงสุด

2.4 อัตราการสุ่มสูง(Over sampling)

กระบวนการ ประมาณค่า ของ การแปลงอะนาลอกเป็นดิจิตอล แบบไนควิส์ท โดยทั่วไปแล้วจะมีความแตกต่างจาก การแปลงอะนาลอกเป็นดิจิตอล แบบ Oversampling เนื่องจากว่าการแปลงอะนาลอกเป็นดิจิตอล แบบนี้ความถี่ที่ใช้ในการสุ่ม (Sampling Rate) มีค่าสูงมากกว่าความถี่ของสัญญาณอินพุทมาก $F_s = Nf_o$, N คือ เลขจำนวนจริงอาจจะมีค่าสูงถึง 128 หรือมากกว่า จากนั้นตามด้วยกระบวนการที่พิจารณาใน Digital - Domain คือกระบวนการที่ลดความถี่ในการสุ่ม (Decimation) ให้ลดลงเหมือนกับการสุ่มแบบไนควิส์ท นอกจากนี้แล้วกระบวนการ ประมาณค่าของ การแปลงอะนาลอกเป็นดิจิตอล แบบ อัตราการสุ่มสูง ยังให้ผลดีในส่วนของการออกแบบ Anti - aliasing Filter ซึ่งจะได้กล่าวถึงในอันดับต่อไป

การสุ่มด้วยอัตราการสุ่มแบบไนควิส์ท ความถี่ที่ใช้สุ่มอย่างน้อยต้องมีค่าเป็น 2 เท่าของความถี่สูงสุดที่เราสนใจ ตัวอย่างเช่น ความถี่ที่ใช้ในการสุ่มเท่ากับ 48 kHz ความถี่สูงสุดที่สามารถผ่านได้โดยไม่มีความผิดพลาด (Aliasing) คือ 24 kHz แต่ในทางปฏิบัติผลจากขีดจำกัดของวงจรจะทำให้ความถี่สูงสุดที่ผ่านได้จะอยู่ที่ประมาณ 22 kHz ดังนั้นแล้ว Anti - aliasing Filter ในการแปลงอะนาลอกเป็นดิจิตอลแบบไนควิส์ท ต้องการผลตอบสนองที่เรียบและไม่มีความผิดเพี้ยนทางเฟสตลอดช่วงความถี่ที่สนใจ (20 kHz ที่มีการใช้งานใน Digital audio) เพื่อป้องกันความผิดเพี้ยนของสัญญาณที่ได้ทางเอาท์พุท ความถี่ที่มากกว่า 24 kHz สำหรับความถี่ในการสุ่มที่ 24 kHz จะถูกลดทอนอย่างน้อย 96 dB สำหรับ 16-bit dynamic resolution ความต้องการข้างต้นจะถูกควบคุมโดย วงจรกรองความถี่ต่ำผ่านแบบอนาลอก ดังแสดงในรูปที่ 2.10(a) ขณะที่รูปที่ 2.10(b) แสดงสเปกตรัมของสัญญาณดิจิตอลในโดเมนของความถี่ที่ถูกสุ่มด้วยความถี่ 48 kHz เราจะพิจารณาสัญญาณ Audio ที่เหมือนกันและถูกสุ่มด้วยความถี่ $2F_s$, 96 kHz Anti - aliasing filter ต้องการกำจัดความถี่ที่เกิน 74 kHz เท่านั้นขณะที่วงจรกรองความถี่มีผลตอบสนองที่เรียบตลอดช่วงจนถึง 22 kHz จากคุณสมบัติของ Anti - aliasing filter ข้างต้นสารถออกแบบได้อย่างง่ายเนื่องมาจากว่าวงจรกรองความถี่

ต่ำผ่านไม่จำเป็นต้องมีช่วง Transition ที่แคบแต่ในทางกลับกันช่วง Transition สามารถทำให้กว้างได้ถึง 52 kHz (22 kHz ถึง 74 kHz)



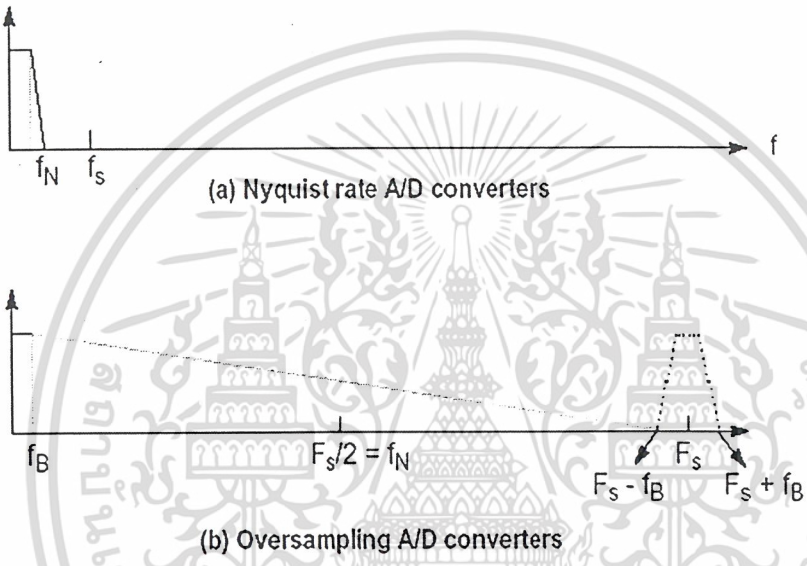
รูปที่ 2.10 ผลตอบสนองของ Anti - aliasing filter และสเปกตรัมของสัญญาณรบกวน ใน การแปลงอะนาลอกเป็นดิจิตอลด้วยอัตราการสุ่มสูง

รูปที่ 2.10(a) แสดงผลตอบสนองทางความถี่โดยทั่วไปของ Anti - aliasing filter ที่ใช้ใน N เท่า การสุ่มสูงขณะที่ระดับสเปกตรัมของสัญญาณรบกวนที่เกิดจากกระบวนการ ประมาณค่าและระดับของสัญญาณรบกวนในช่วงความถี่ที่สนใจ หลังจากผ่านกระบวนการลดความถี่ของการสุ่ม (Decimation) และผ่านวงจรกรองความถี่แบบดิจิตอลแสดงในรูปที่ ๑๐(b) ส่วนผลรวมค่ากำลังของสัญญาณรบกวนทั้งแบบ การสุ่มสูง และ ไนควิส์ท มีค่าเหมือนกัน ดังสมการข้างล่าง

$$N_B = \int_{-f_B}^{f_B}(f)df = \frac{2f_B q^2}{F_S 12} \quad \text{.....2.13}$$

เมื่อ N_B = กำลังของสัญญาณรบกวนใน ช่วงความถี่ที่สนใจ

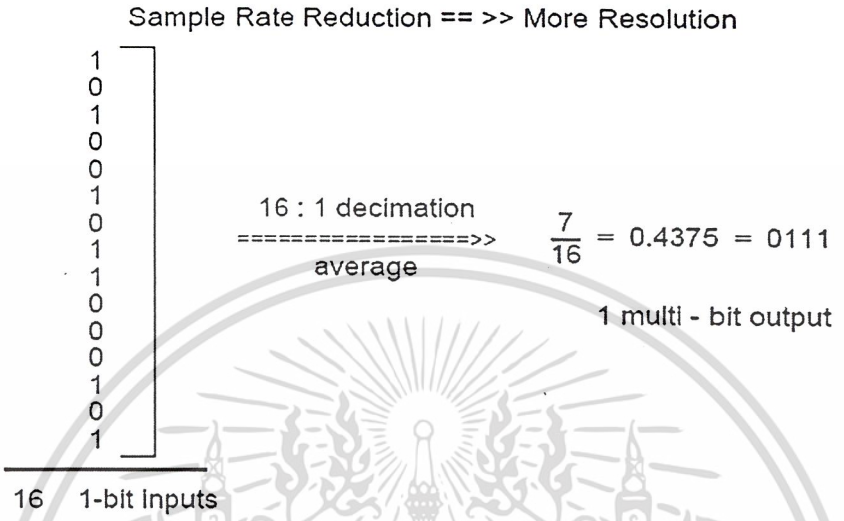
จะพบว่ากำลังของสัญญาณรบกวนจะมีค่าน้อยกว่าการแปลงอะนาลอกเป็นดิจิตอลแบบไนควิส์ทโดยเฉพาะอย่างยิ่งเมื่อความถี่ใช้สุ่มมีค่าสูงมากกว่าความถี่สูงสุดของช่วงความถี่ที่สนใจ FS >> F_B



รูปที่ 2.11 ผลตอบสนองทางความถี่ของ Anti - aliasing filter

รูปที่ 2.11 เป็นการเปรียบเทียบความต้องการวงจรกรองความถี่ต่ำผ่าน (Anti - aliasing filter) ระหว่าง 1 เท่าและ N เท่า พบว่าไนควิส์ท ต้องการวงจร กรองความถี่ต่ำผ่านที่มีช่วงทรานซิชั่น แคบเพื่อผลในการจำกัดแบนด์วิทธ์ สัญญาณอินพุท แต่ในส่วนของอะนาลอกเป็นดิจิตอลแบบอัตราการสุ่มสูง วงจรกรองความถี่ต่ำที่ต้องการ สามารถมีช่วงทรานซิชั่นกว้างมากกว่าช่วงความถี่ผ่านได้ เพราะวงจรต้องการป้องกันเฉพาะช่วงความถี่ระหว่าง $NF_S - F_B$ และ $NF_S + F_B$ ทั้งนี้ความซับซ้อนของการออกแบบวงจรกรองความถี่ผ่านขึ้นอยู่กับความกว้างของช่วงทรานซิชั่นและช่วงความถี่ผ่าน ดังนั้นแล้ววงจร อะนาลอกเป็นดิจิตอลแบบอัตราการสุ่มสูงจึงต้องการการออกแบบวงจรกรองความถี่ที่ง่ายกว่า อะนาลอกเป็นดิจิตอลแบบอัตราการสุ่มไนควิส์ท ทำให้ อะนาลอกเป็นดิจิตอลแบบอัตราการสุ่มสูง แบบนี้สามารถประหยัด Anti - aliasing filter ได้ ในส่วนของภาคดิจิตอลกระบวนการลดความถี่สุ่ม (Decimation) จะทำหน้าที่เพิ่ม ความละเอียดให้กับ อะนาลอกเป็นดิจิตอลในรูปที่ 2.12 แสดงตัวอย่าง 16 : 1 Decimation กับ 1 – บิทอินพุท การ

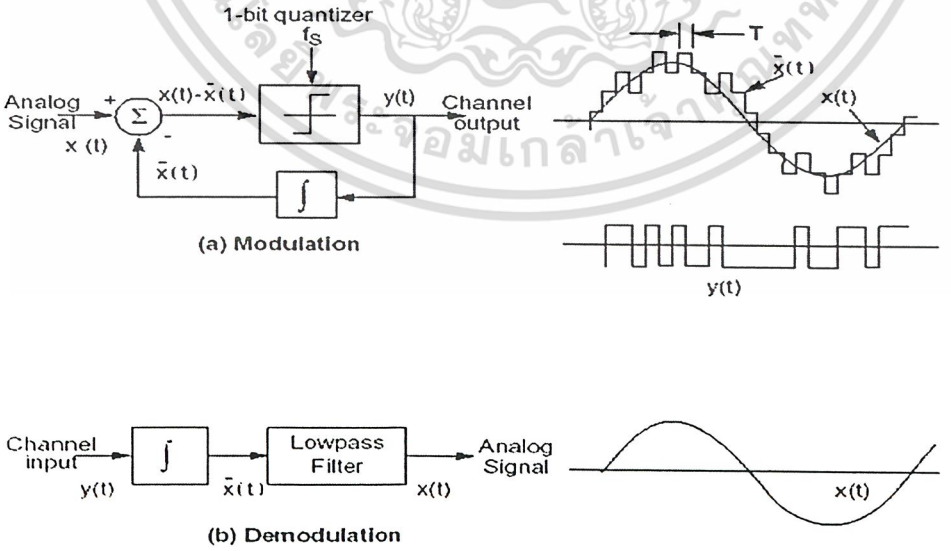
เฉลี่ย (Decimation) เป็นผลให้เสมือนว่าความถี่ในการสุ่มมีค่าลดลงด้วยอัตราส่วน 16 ต่อ 1 แน่นอน ว่าความถี่ที่ใช้สุ่มที่สูงจะถูกเปลี่ยนไปเป็น ความละเอียด ที่สูงขึ้น



รูปที่ 2.12 ตัวอย่างแบบง่ายของกระบวนการ Decimation

2.5 เดลตา มอดดูเลชั่น (Delta Modulation)

ก่อนที่จะไปพิจารณาการทำงานของ อะนาลอกเป็นดิจิตอล แบบ เดลตา – ซิกมา เราจะมาทำความเข้าในใน ส่วนของโครงสร้าง เดลตามอดดูเลชั่นและดีมอดดูเลชั่น อะนาลอกเป็น ดิจิตอล ในรูปที่ 2.13 แสดงบล็อกไดอแกรมของเดลตามอดดูเลเตอร์ และ ดีมอดดูเลเตอร์



รูปที่ 2.13 บล็อกไดอแกรมของ เดลตามอดดูเลชั่น และ ดีมอดดูเลชั่น

พบว่าเอาท์พุทของเดลตามอดดูเลชั่นจะมีการป้อนกลับมาโดยผ่านวงจรอินทีเกรเตอร์และนำไปรวมกับ อินพุทต่อไปจะได้ในส่วนของ x (t) – x (t) ทั้งนี้แล้วประสิทธิภาพการทำงานของวงจรจะขึ้นอยู่กับความถี่ที่ใช้ในการสุ่ม

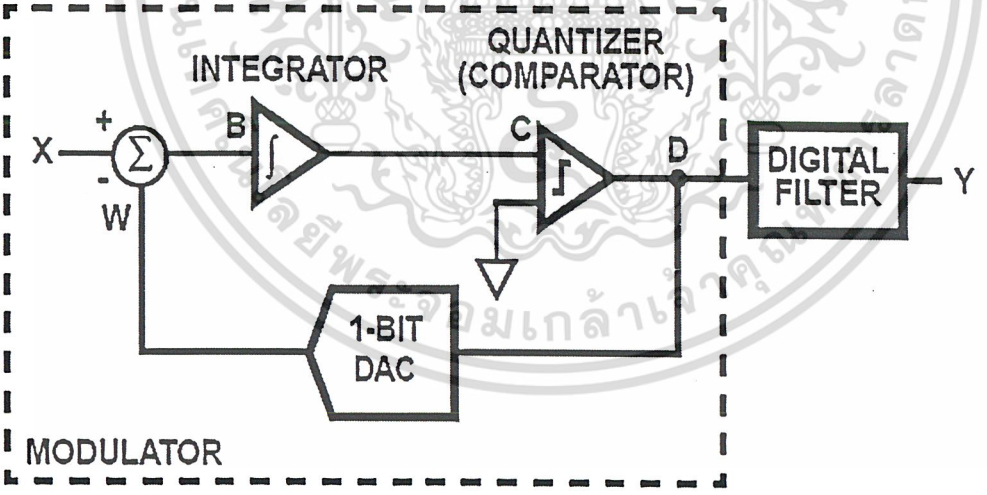
2.6 องค์ประกอบพื้นฐานของการแปลงสัญญาณอนาลอกเป็นดิจิตอลแบบเดลตา – ซิกมา (Delta – Sigma ADC)

การแปลงสัญญาณแบบเดลตา – ซิกมาจะประกอบไปด้วยพื้นฐาน 2 ส่วน คือ

1 อนาลอกมอดดูเลชั่น

2 ดิจิตอลฟิลเตอร์

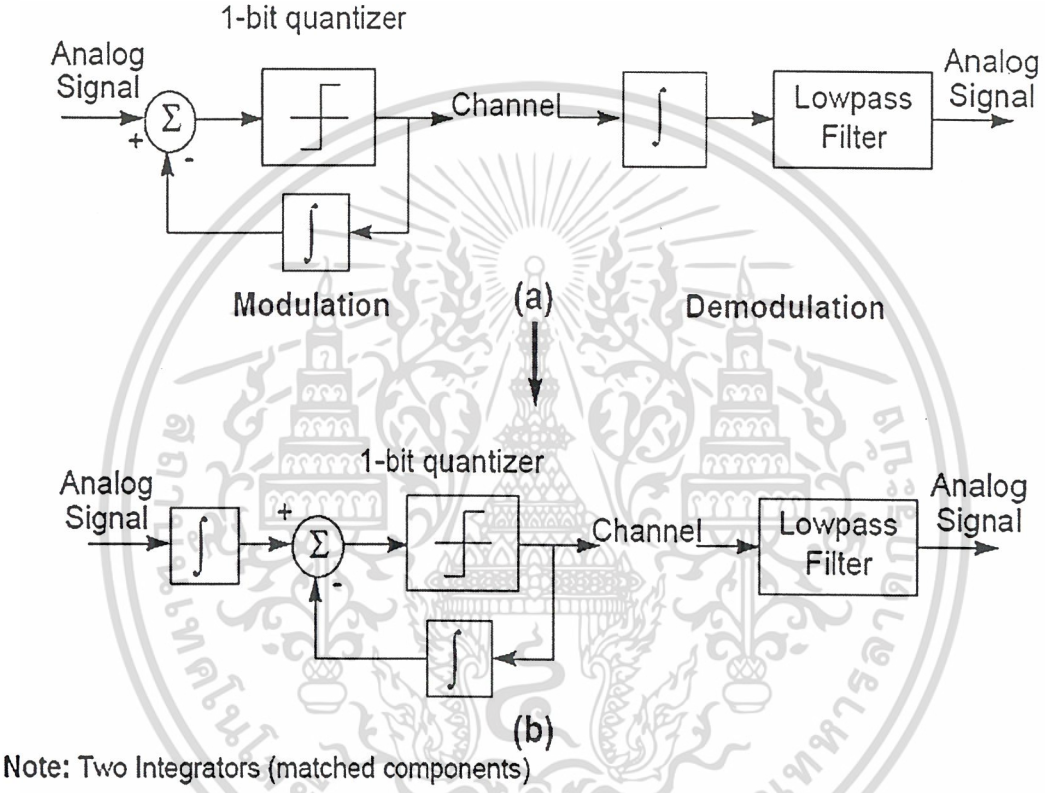
พิจารณาเดลตา – ซิกมาดังรูปที่ 14 สัญญาณอนาลอกอินพุท ที่เข้ามาในส่วนของอนาลอกมอดดูเลเตอร์ด้วยความถี่ที่มีค่าสูงกว่าความถี่สูงสุดที่เข้ามามากๆ หลังจากนั้นสัญญาณที่ได้จากการ การสุ่มสูง จะถูกนำไปเปรียบเทียบกับสัญญาณที่ถูกป้อนกลับผ่าน ดิจิตอลเป็นอะนาลอก สัญญาณที่ได้จะป้อนเข้าสู่วงจรอิรทิเกรต และคอมพาราเตอร์ตามลำดับ สัญญาณเอาท์พุทที่ได้จากวงจรคอมพาราเตอร์จะเป็นสัญญาณดิจิตอล 1 บิท ที่มีอัตราการส่งข้อมูลสูง ต่อข้อมูลที่ได้จากถูกส่งต่อเข้าส่วนของดิจิตอลฟิลเตอร์ ซึ่งจะทำการแปลงสัญญาณดิจิตอล 1 บิทเป็นข้อมูลดิจิตอล n บิท ที่มีคุณภาพดีขึ้นพร้อมกับการลดความถี่ในการส่งข้อมูลลงเสมือนว่าอัตราการสุ่มลดลง



รูปที่ 2.14 การเข้ารหัสอันดับที่ 1 ของเดลตา – ซิกมา

สิ่งหนึ่งที่มีความสำคัญที่ต้องคำนึงถึงและทำความเข้าใจคือ รูปร่างของสัญญาณรบกวนที่จะเกิดขึ้นในช่วงความถี่ที่ทำการพิจาณาโดยจะได้มีการอธิบายในอันดับต่อไปในขั้นแรกจะขอพูดถึงลักษณะโครงสร้างโดยทั่วไปและเป็นพื้นฐานสำหรับทำความเข้าใจ

วงจรเดลตามอดดูเลชั่นต้องการวงจรอินทิเกรเตอร์ 2 ส่วนด้วยกันสำหรับกระบวนการมอดดูเลชั่นและดีมอดดูเลชั่นดังแสดงในรูปที่ 2.15(a) เนื่องจากว่าการอินทิเกรตทำงานในลักษณะเชิงเส้นดังนั้นแล้วจึงสามารถย้ายส่วนของอินทิเกรเตอร์มาไว้ก่อนตัวมอดดูเลเตอร์ โดยที่ไม่ทำให้ลักษณะสัญญาณอินพุทและเอาท์พุทเปลี่ยนแปลง ดังที่กล่าวมาแล้วนั้น วงจรอินทิเกรเตอร์ทั้ง 2 ตัวในรูปที่ 2.15 จึงสามารถยุบให้เหลือเพียงตัวเดียวด้วยคุณสมบัติของความเป็นเชิงเส้น



รูปที่ 2.15 การเปลี่ยน เดลตามอดดูเลชั่นเป็น ซิกมา – เดลตา มอดดูเลชั่น

จากรูปที่ 2.16(b) ถ้าให้สัญญาณเอาท์พุทของมอดดูเลชั่น คือ y(n) และอินพุทคือ x(t) จุดที่สัญญาณทั้งสองตัวมารวมกันและได้สัญญาณก่อนที่จะเข้าสู่ ตัวประมาณค่าแบบ 1 – bit ให้เท่ากับ x*(t) ลักษณะของสัญญาณคือ

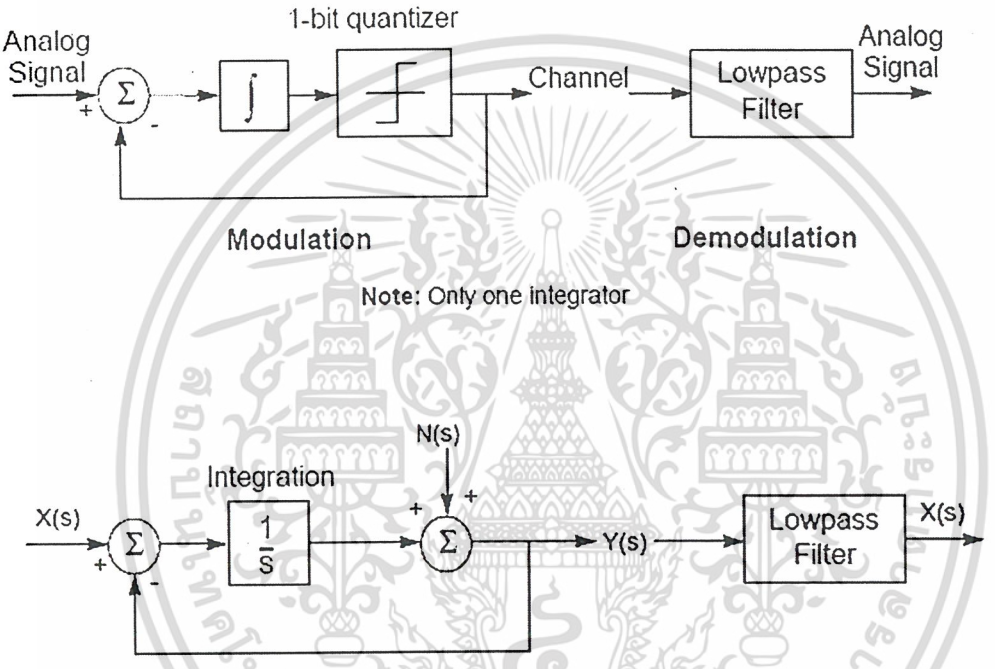
$$x^*(t) = \frac{1}{s}x(t) - \frac{1}{s}y(n)$$

$$x^*(t) = \frac{1}{s}\left(x(t) - y(n)\right)$$

$$When, \frac{1}{s} = Integration$$

พบว่าส่วนของวงจรอินทิเกรเตอร์จะลดลงเหลือเพียง 1 ตัวเท่านั้นโดยที่นำสัญญาณเอาท์พุทที่ได้และอินพุทมากระทำกันก่อนแล้วค่อยอินทิเกรตพร้อมกันดังแสดงในรูปที่ 16 เรียกโครงสร้างของวงจรแบบนี้ว่า

ซิกมา – เดลตา มอดดูเลเตอร์ (Sigma – Delta Modulator ; $\Sigma - \Delta$)



รูปที่ 2.16 บล็อกไดอแกรมของการมอดดูเลชั่นแบบ ซิกมา – เดลตา

จากรูปข้างต้นเราจะทำการวิเคราะห์โดยใช้การพิจารณาใน S – Domain ทั้งลักษณะสัญญาณเอาท์พุทและสัญญาณรบกวนที่เกิดขึ้นจากกระบวนการ ประมาณค่า (Quantization Noise) ของเดลตา – ซิกมาจะใช้การ ประมาณค่าที่หยาบๆ โดยใช้เพียงคอมพาราเตอร์ ดังนั้นที่ซัมมิ่งโหนดทางขวาของอินทิเกรตเตอร์จึงเสมือนแหล่งกำเนิดสัญญาณรบกวนให้กับระบบ ต่อไปลองพิจารณา ทรานเฟอร์ฟังก์ชั่นของสัญญาณอินพุทและเอาท์พุท จะได้ว่า

Signal Transfer Function:

$$Y(s) = [X(s) - Y(s)]\frac{1}{s}$$

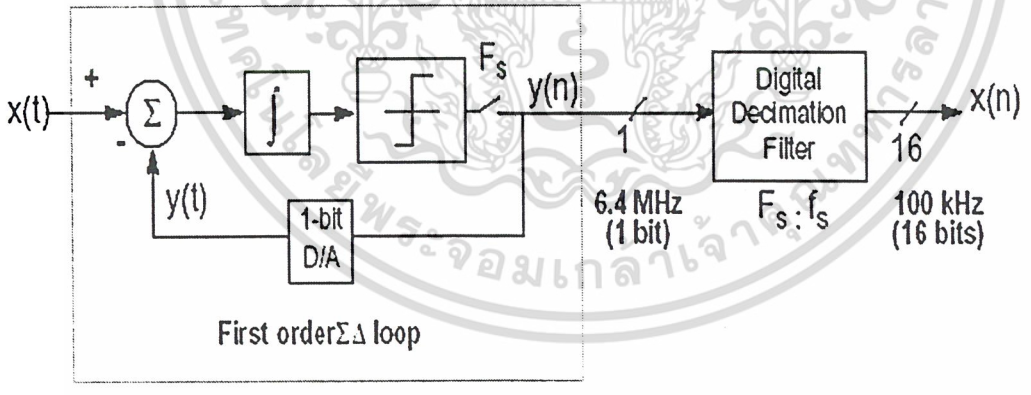
$$\frac{Y(s)}{X(s)} = \frac{\frac{1}{s}}{1+\frac{1}{s}} = \frac{1}{s+1}$$

สมการข้างต้นแสดงให้เห็นว่า Transfer Function ของเอาท์พุทและอินพุทจะเป็นลักษณะของวงจรกรองความถี่ต่ำผ่าน ซึ่งหมายความว่าสัญญาณที่ได้ทางด้านเอาท์พุทจะไม่เปลี่ยนแปลงตลอดช่วงความถี่ที่ไม่เกินความถี่ตัดของวงจรกรองความถี่ ต่อไปดู Transfer Function ของสัญญาณรบกวน

$$\text{Noise Transfer Function} : Y(s) = -Y(s) + N(s)$$

$$\frac{Y(s)}{N(s)} = \frac{1}{1+\frac{1}{s}} = \frac{s}{s+1}$$

พบว่า Transfer Function ของสัญญาณรบกวนจะเป็นลักษณะของวงจรกรองความถี่สูงผ่าน เป็นที่ชัดเจนว่า อัตราสุ่มสูง อินพุทจะกระจายสัญญาณรบกวนที่มาจากกระบวนการควอนไตซ์ตลอดช่วงแบนด์วิทด์ที่กว้างเป็นผลให้ความหนาแน่นของสัญญาณรบกวนในช่วงความถี่ที่สนใจมีค่าลดลง

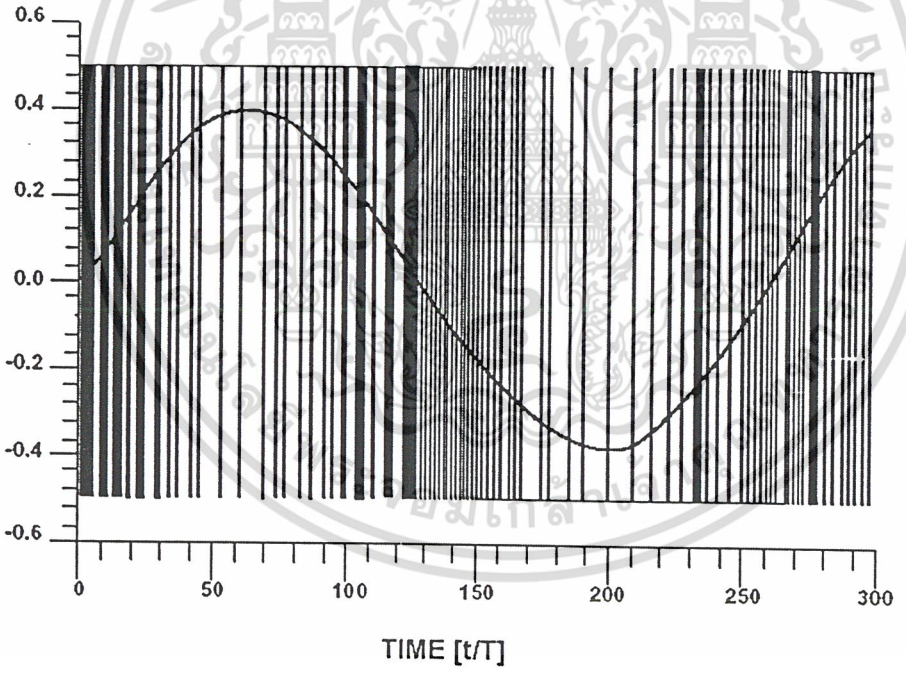


รูปที่ 2.17 มอดดูเลชั่นอันดับหนึ่งของ อะนาลอกเป็นดิจิตอลแบบ ซิกมา-เดลตา

รูปที่ 2.17 แสดงบล็อกไดอแกรมของ มอดดูเลชั่นอันดับหนึ่งของ อะนาลอกเป็นดิจิตอลแบบ ซิกมา-เดลตา1 บิทดิจิตอลเอาท์พุทจากมอดดูเลเตอร์ถูกส่งเข้าสู่ดิจิตอล Decimation filter เพื่อเพิ่มความถูกต้องและลดอัตราการสุ่มของอินพุทที่เข้ามา มอดดูเลชั่นอันดับหนึ่ง ซิกมา – เดลตา ประกอบด้วย Analog differential node อินทิเกรเตอร์ 1บิทควอนไตเซอร์(A/D converter)

และ 1 บิท ดิจิตอลเป็นอะนาลอก ซึ่งเป็นตัวป้อนกลับ ส่วนเอาท์พุทจะมีเพียง 1 บิทข้อมูล และมี 2 ระดับ เอาท์พุทของมอดดูเลเตอร์ y(n) เปลี่ยนเป็น y(t) ด้วย 1- บิท ดิจิตอลเป็นอะนาลอก

อินพุทของมอดดูเลเตอร์เกิดจากความแตกต่างระหว่างสัญญาณอินพุทและเอาท์พุทของมอดดูเลเตอร์ ถูกป้อนกลับและเปลี่ยนเป็นสัญญาณอนาลอก y(t) ทำหน้าที่โดย ดิจิตอลเป็นอะนาลอก และละทิ้งผลของการหน่วงเวลา ความแตกต่างระหว่างอินพุท x(t) ตัวป้อนกลับ y(t) อินพุทอินทิเกรเตอร์เท่ากับ ความผิดพลาดจากการประมาณค่า สัญญาณรบกวนตัวนี้ถูกรวมเข้าโดยอินทิเกรเตอร์และถูก ประมาณค่า โดย 1 –บิท อะนาลอกเป็นดิจิตอล ถึงแม้ว่าสัญญาณรบกวนที่เกิดในกระบวนการ ประมาณค่าจะเกิดขึ้นทุกๆช่วงของการสุ่มและมีขนาดใหญ่ตามธรรมชาติของการประมาณค่า 2 ระดับ แต่ $\Sigma - \Delta$ จะสร้างเอาท์พุทมีค่า ± 1 ที่เกิดจากการเฉลี่ยหลายอินพุทที่มาจากการสุ่ม การเฉลี่ย จะทำหน้าที่โดยดิจิตอลเดซิเมชั่นคูตามรูปที่ ๑๖ รูปคลื่น x(t) และ y(n) สำหรับมอดดูเลชั่นอันดับหนึ่งของ อะนาลอกเป็นดิจิตอลแบบ ซิกมา-เดลตา แสดงในรูปที่ 2.18 เมื่ออินพุทเป็นคลื่นซายน์



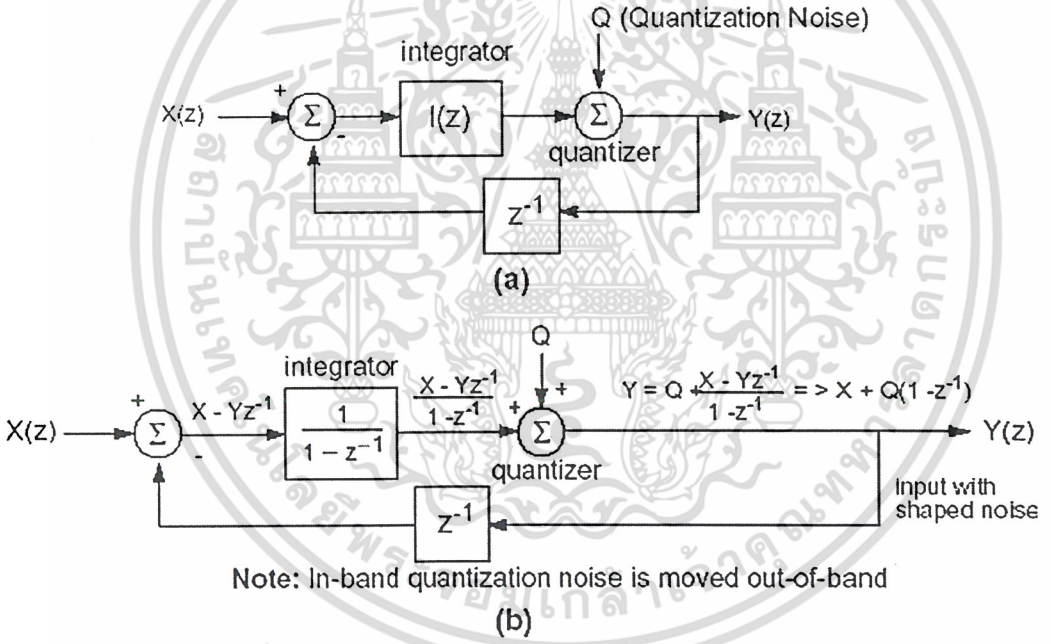
รูปที่ 2.18 อินพุทและเอาท์พุทของ มอดดูเลชั่นอันดับหนึ่งของ
อะนาลอกเป็นดิจิตอลแบบ ซิกมา-เดลตา

การทำงานของมอดดูเลเตอร์ทั้งการสุ่มและการประมาณค่า คล็อกแต่ละลูกที่เข้ามาเอาท์พุทของตัวมอดดูเลเตอร์จะบวกเต็มสเกลหรือลบเต็มสเกลตามผลของ 1 – บิท อะนาลอกเป็นดิจิตอล เมื่อ

สัญญาณอินพุทที่เป็นซายน์เข้ามาที่มอดดูเลเตอร์ ค่าเข้าใกล้บวกเต็มสเกล เอาท์พุทจะเป็นบวกตลอด Clock cycle เช่นเดียวกันเมื่ออินพุทมีค่าลบเต็มสเกลเอาท์พุทก็จะเป็นค่าลบตลอด Clock cycle เช่นเดียวกัน โดยปกติแล้วค่าเฉลี่ยเอาท์พุทจะ เทร็คตามสัญญาณอินพุทเมื่ออินพุทมีค่าใกล้ศูนย์การเปลี่ยนแปลงของเอาท์พุทจะเปลี่ยนเป็นบวกและลบอย่างรวดเร็วค่าเฉลี่ยจะประมาณเท่ากับศูนย์

2.6.1 การวิเคราะห์ ซิมา – เดลตา มอดดูเลชั่นใน แซดโดเมน (Analysis of Sigma – Delta Modulation in the Z – Transform Domain)

พิจาณาการมอดดูเลชั่นอันดับหนึ่งโดยดูได้จากรูปที่ ๑๘ จะพิจารณา Transfer Function ของระบบด้วย Z – Domain Transform แทนอินทิเกรเตอร์ด้วย I(z) และ 1 - บิท Quantizer แทนด้วยแหล่งกำเนิดสัญญาณรบกวน



รูปที่ 2.19 การวิเคราะห์วงจรอันดับหนึ่งใน Z - Domain

มาตรฐานการวิเคราะห์สัญญาณที่ไม่ต่อเนื่องทางเวลาจะได้สัญญาณทางเอาท์พุทของวงจรมอดดูเลเตอร์

$$Y(z) = Q(z) + I(z)[X(z) - Z^{-1}Y(z)] \quad \text{.....2.14}$$

สามารถจัดรูปสมการใหม่

$$Y(z) = X(z)\frac{I(z)}{1+I(z)z^{-1}} + Q(z)\frac{1}{1+I(z)z^{-1}} \quad \text{.....2.15}$$

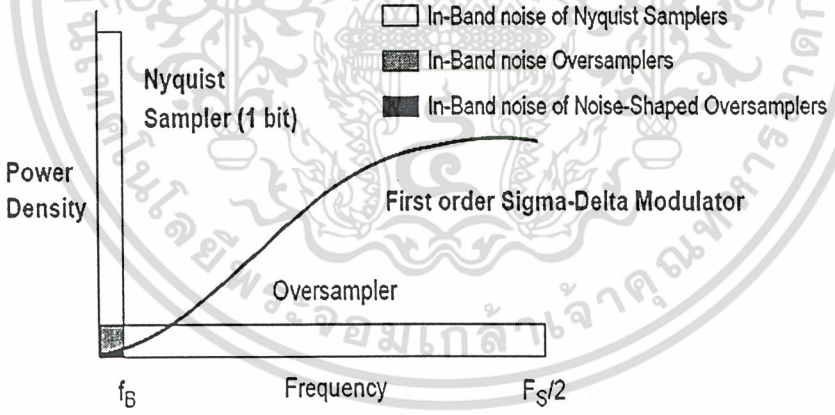
อินทิเกรเตอร์ทางอุดมคติแทนด้วย

$$I(z) = \frac{1}{1-z^{-1}}$$

แทนลงในสมการที่ 2.15

$$Y(z) = X(z) + (1-z^{-1})Q(z) \quad \text{.....2.16}$$

เพราะว่า สัญญาณรบกวนจากการประมาณค่า เป็นปริมาณที่ถูกสมมติให้มีค่าไม่แน่นอน (Random) ในสมการที่ 4 Differentiator $(1-z^{-1})$ ค่า z คือเอ็กโพเนนเชียลที่ยกกำลังเป็นจำนวนเชิงซ้อน $z = e^{\omega T}$ แทนลงในสมการที่ 4 พบว่าค่ากำลังของสัญญาณรบกวนจะเพิ่มมากขึ้นเมื่อความถี่สูงขึ้น ดังนั้นแล้วสัญญาณอนาลอกอินพุทที่ถูกสุ่มแบบ อัตราการสุ่มสูง ความถี่สูงของสัญญาณรบกวนจากการประมาณค่า จะถูกกำจัดโดยวงจรกรองความถี่แบบดิจิตอล โดยไม่มีผลต่อลักษณะของสัญญาณอินพุทในช่วงความถี่ความถี่ที่สนใจ ส่วนกรองความถี่ต่ำนี้จะอยู่ในกระบวนการDecimation



รูปที่2.20 สเปกตรัมของสัญญาณรบกวน มอดดูเลชั่นอันดับหนึ่ง

รูปที่ 2.20 แสดงสเปกตรัมของสัญญาณรบกวนสำหรับ มอดดูเลชั่นอันดับหนึ่ง สิ่งหนึ่งที่เรามองเห็นได้ชัดเจนคือ กำลังของสัญญาณรบกวนที่ปรากฏในช่วงความถี่ที่สนใจ (Baseband) จะมีค่าน้อยกว่าที่เกิดขึ้นใน อัตราการสุ่มไนควิส์ท ทั้งนี้แล้วกำลังของสัญญาณรบกวนก็มีค่าเพิ่มขึ้นตามความถี่ที่เพิ่มขึ้นด้วยเช่นกัน อย่างไรก็ตามในการออกแบบวงจร แปลงอะนาลอกเป็นดิจิตอล ที่มีค่า ความละเอียด เท่ากับ 16 บิทมอดดูเลชั่นอันดับหนึ่งก็ยังไม่เพียงพอที่จะทำให้

อัตราส่วนของสัญญาณรบกวนกับสัญญาณอินพุทในช่วงความถี่ที่สนใจ (Baseband) มีค่าน้อยกว่า -96dB ดังนั้นจึงต้องมีการสร้างการมอดดูเลชั่นที่มีอันดับมากขึ้น หลักการสร้างก็คือการนำมอดดูเลชั่นอันดับหนึ่งมาต่อกันในลักษณะ Cascade แล้วป้อนสัญญาณรบกวนกลับเข้ามาอีกครั้งทั้งมอดดูเลชั่นอันดับสอง และมากกว่าสองการพิจารณาจะกระทำในลักษณะเดียวกัน

การพิจารณาจะอ้างอิงรูปที่ 2.19 ให้เอาท์พุทคือ y(n) อินทิเกรเตอร์ แทนด้วย $I(z) = \frac{1}{1-z^{-1}}$ ส่วน 1- บิท DAC จะแทนด้วย z^{-1} และคอมพาราเตอร์ที่ใช้สำหรับการquantized จะเป็นเสมือนจุดที่สัญญาณรบกวนถูกป้อนเข้ามาสู่วงจร และ Q(z) คือสัญญาณรบกวนดังที่เคยกล่าวไว้แล้วในการพิจารณากามอดดูเลชั่นอันดับหนึ่ง

$$Y(z) = \left[\left[\left(X(z) - Y(z)z^{-1}\right)\frac{1}{1-z^{-1}}\right] - Y(z)z^{-1}\right]\frac{1}{1-z^{-1}} + Q(z)$$

$\frac{1}{1-z^{-1}}$ แทนด้วย $\frac{z}{z-1}$

จะได้

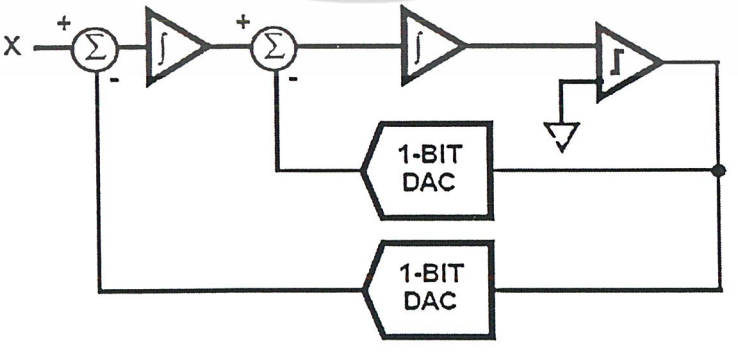
$$Y(z) = \left(X(z)\frac{z}{z-1} - Y(z)\frac{1}{z-1}\right)\frac{z}{z-1} - Y(z)\frac{1}{z-1} + Q(z)$$

$$Y(z) = X(z)\frac{z^2}{(z-1)^2} - Y(z)\frac{z}{(z-1)^2} - Y(z)\frac{1}{z-1} + Q(z)$$

$$Y(z)\frac{z^2}{(z-1)^2} = X(z)\frac{z^2}{(z-1)^2} + Q(z)$$

ดังนั้นแล้วจะได้ว่าเอาท์พุทของการมอดดูเลชั่นในอันดับที่ 2 คือ

$$Y(z) = X(z) + (1-z^{-1})^2 Q(z)$$



รูปที่ 2.21 การมอดดูเลชั่นอันดับที่ 2

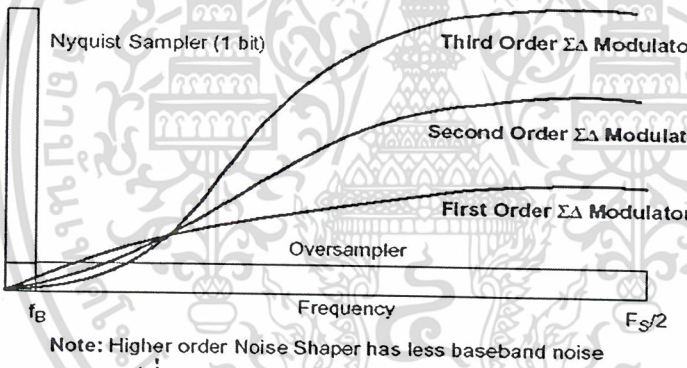
สำหรับในอันดับที่สูงกว่าสอง เช่นมอดดูเลชั่นอันดับ 3 สมการเอาท์พุทก็จะได้ดังนี้

$$Y(z) = X(z) + (1 - z^{-1})^3 Q(z)$$

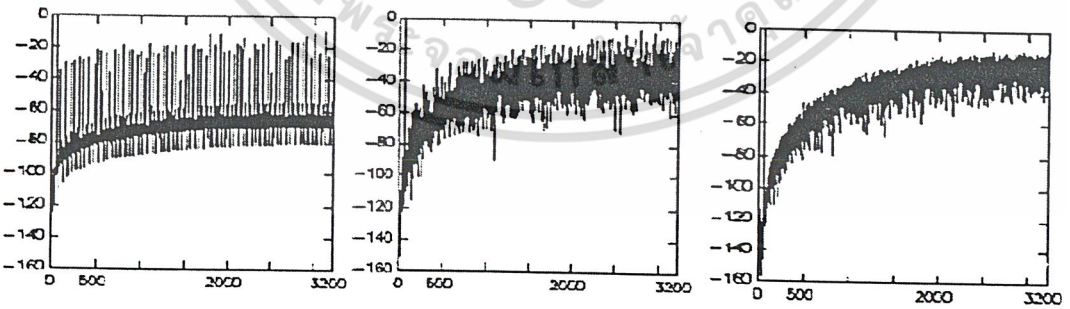
ในอันดับ n ใดๆ

$$Y(z) = X(z) + (1 - z^{-1})^n Q(z) \qquad (***)$$

ดังในรูปที่ 2.22 เป็นการแสดงค่าความหนาแน่นกำลังของสัญญาณรบกวนที่ปรากฏในช่วงความถี่ที่กำลังพิจารณา โดยหลักแล้ว Noise shaping จะเป็นส่วนกลับTransfer function ของวงจรกรองความถี่ $(1 - z^{-1})^{-1}$ ในส่วนป้อนไปด้านหน้าของมอดดูเลเตอร์ วงจรกรองความถี่มีค่าอัตราขยายสูงในช่วงความถี่ต่ำและลดทอนสัญญาณรบกวนได้ดีในช่วงความถี่ Base band ดังนั้นการทำงานของมอดดูเลเตอร์ที่มีอันดับสูงจะมีความสามารถในการลดทอนสัญญาณรบกวนที่เกิดขึ้น ในช่วงความถี่ Baseband ที่แตกต่างกัน เมื่อเปรียบเทียบกำลังสัญญาณรบกวนที่เกิดขึ้น ในความถี่ Baseband ของมอดดูเลชั่นอันดับหนึ่งและมอดดูเลชั่นอันดับ 3 พบว่ามอดดูเลชั่นอันดับ 3 จะมีสัญญาณรบกวนน้อยกว่ามอดดูเลชั่นอันดับหนึ่ง



รูปที่ 2.22 Multi – Order Sigma – Delta Noise Shapers



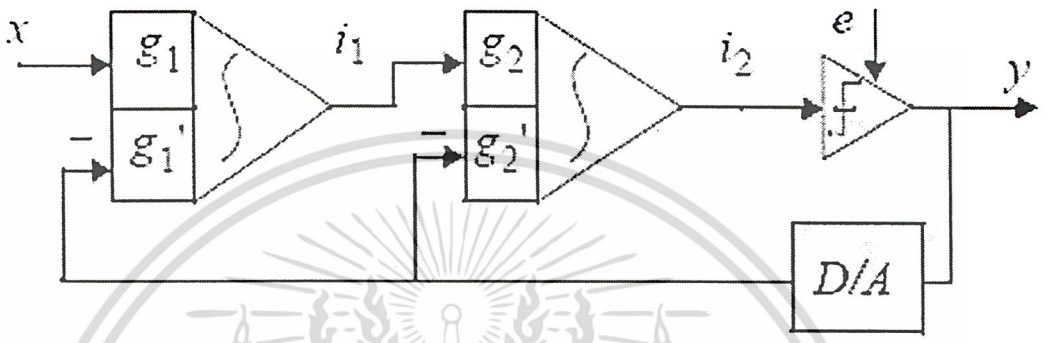
(a) First order sigma-delta (b) Second order sigma-delta (c) Third order sigma-delta

NOTE: Frequency band of interest (in-band): 0 - 5 kHz
Rest of frequency band will be removed by digital decimation filters

รูปที่ 2.23 (a) สเปกตรัมของสัญญาณรบกวนใน ซิกมา – เดลตาอันดับที่ 1

(b) อันดับที่ 2 **(c)** อับดับที่ 3

จากที่เรากล่าวมาในขั้นต้นอินพุทที่ป้อนเข้ามอดดูเลเตอร์และเอาท์พุทที่ป้อนกลับมา เป็นการป้อนเข้าโดยตรง โดยไม่ได้มีค่าถ่วงน้ำหนักเข้ามาเกี่ยงข้องด้วย ดังนั้นในตอนนี้เราจะกล่าวถึงมอดดูเลเตอร์ที่มีค่าถ่วงน้ำหนัก ดังรูปที่ 2.23 ที่มีการถ่วงน้ำหนักทั้งสองกรณีและค่าสัมประสิทธิ์การถ่วงน้ำหนักแสดงไว้ในตารางที่ 1 ซึ่งเป็นค่าที่ได้ทำการทดสอบและเป็นทางเลือกที่ดี สำหรับการออกแบบ



รูปที่ 2.24 Single Loop Second – Order Modulation

ตารางที่ 2.1 ค่าถ่วงน้ำหนักของอินทิเกรเตอร์

Weight	[3]	[4]	[5]	[6]
$g_1 \cdot g_1'$	0.5	0.25	1/3	0.25
g_2	0.5	0.5	0.6	1
g_2'	0.5	0.25	0.4	0.5

บทที่3

Digital Filter

ในการออกแบบดิจิตอลฟิลเตอร์สำหรับกรองสัญญาณที่ไม่ต้องการออกจากช่วงสัญญาณที่สนใจซึ่งในโครงงานนี้คือสัญญาณที่มีความถี่น้อยกว่า 20KHz โดยทั่วไปเรามักจะให้ความสำคัญกับผลตอบสนองแอมปิจูดมากกว่านั่นคือ ออกแบบให้ผลตอบสนองแอมปลิจูดตามต้องการโดยยอมให้ผลตอบสนองเฟสเป็นอะไรก็ได้ แต่สำหรับโครงงานนี้เราต้องคำนึงถึงผลของการตอบสนองทางเฟสด้วยเนื่องจากการใช้งานในบางงานต้องการคุณสมบัตินี้ เช่น การประมวลผลสัญญาณ อีซีจ (ECG: Electrocardiogram) แต่ทั้งนี้ยังสามารถใช้งานกับสัญญาณที่ความถี่ไม่เกิน 20 KHz ไม่ว่าจะเป็นสะญญาณออดิโอ สัญญาณเสียงพูด

ดังนั้นแล้วเราจึงเลือกดิจตอลฟิลเตอร์ที่มีความเป็นเชิงเส้นทางด้านเฟสเรียกว่า "Non – recursive หรือ FIR: Finite Impulse Respond"

ลักษณะของดิจดตอลแบบ FIR

1. FIR ดิจิตอลฟิลเตอร์มีความเป็นเชิงเส้นทางเฟส ความเป็นเชิงเส้นทางเฟสมีความสำคัญสำหรับการใช้งานที่ความผิดเพี้ยนและความไม่เป็นเชิงเส้นทางเฟสสามารถลดการทำงานของระบบ ตัวอย่างเช่น การประมวลผลสัญญาณเสียงพูด การส่งผ่านข้อมูล และความสำพันธ์ของข้อมูล
2. FIR ฟิลเตอร์ไม่มีการป้อนกลับจึงมีเสถียรภาพเสมอ ผลตอบสนองอิมพัลล์มีความยาวที่จำกัดดังนั้นจึงมีขอบเขต
3. FIR ฟิลเตอร์สามารถใช้งานแบบมัลิเรท DSP ได้

3.1 คุณสมบัติของ FIR ฟิลเตอร์

3.1.1 ความเป็นเชิงเส้นทางเฟส

จากสมการฟังก์ชั่นถ่ายโอนของตัวกรองไม่ป้อนกลับเชิงเลขแบบ Causal ได้ว่า

$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n} \quad \text{.....(3.1)}$$

โดยที่ h (n) คือผลตอบสนองอิมพัลล์ของตัวกรอง differential equation ของสมการที่ 3.1สามารถหาได้จากการทำอินเวอร์ส แซดทรานสฟอร์ม ในสมการที่ 3.1

$$y(iT) = \sum_{n=0}^{N-1} h(n)x(iT - nT) \quad \text{.....(3.2)}$$

จากสมการที่ 3.2 จะพบว่าเป็นผลรวมการผสาน(Convolution summation) สังเกตุพบว่าผลตอบสนองอิมพัลส์ที่จำกัดเหมือนกันกับค่าสัมประสิทธ์ของฟิลเตอร์

จากสมการที่ 2.15 Fourier Transform ของลำดับที่จำกัด h (n) หาได้โดย

$$H(e^{j\omega T}) = \sum_{n=0}^{N-1} h(n)e^{-i\omega nT} = \left|H(e^{i\omega T})\right|e^{i\theta(\omega)} \quad \text{....(3.3)}$$

ผลตอบสนองทางขนาดและเฟสแสดงได้ด้วย

$$M(\omega) = \left|H(e^{j\omega T})\right|$$

$$\theta(\omega) = \tan^{-1}\frac{-\operatorname{Im} H(e^{j\omega T})}{\operatorname{Re} H(e^{j\omega T})} \quad \text{.....(3.4)}$$

ต่อไปเป็นการอธิบายการทำงานของ การหน่วงทางเฟสและการหน่วงทางกลุ่ม ซึ่งแสดงได้

$$\tau_p = -\frac{\theta(\omega)}{\omega} \quad and \quad \tau_g = -\frac{d\theta(\omega)}{d\omega} \quad \text{.....(3.5)}$$

ในเบื้องต้นดูกรณี ตัวกรองที่ให้ผลตอบสนองเฟสเป็นแบบ ความหน่างเฟสคงตัว และความหน่วงกลุ่มคงตัว พร้อมกันซึ่งตามนิยามแล้วผลตอบสนองเฟสต้องมีคุณสมบัติคือ

$$\theta(\omega) = -\tau\omega \quad -\pi < \omega < \pi \quad \text{.....(3.6)}$$

เมื่อประยุกต์ใช้นิยามของผลตอบสนองเฟสจากสมการที่ 3.3 , 3.4 และ 3.6 จะได้ผลลัพเป็น

$$\theta(\omega) = \tan^{-1}\frac{-\sum_{n=0}^{N-1} h(n)\sin\omega nT}{\sum_{n=0}^{N-1} h(n)\cos\omega nT} \quad \text{.....(3.7).}$$

หรือ

$$\tan = \frac{\sum_{n=0}^{N-1} h(n)\sin \omega nT}{\sum_{n=0}^{N-1} h(n)\cos \omega nT} \quad \text{.....(3.8)}$$

ทำการคูณไขว้ได้เงื่อนไขที่กำหนดถึง คุณสมบัติอิมพัลส์ของตัวกรองไม่ป้อนกลับเชิงเลข นี้เป็น

$$\sum_{n=0}^{N-1} h(nT)\cdot\{\cos n\omega T \sin \omega\tau - \sin n\omega T \cos \omega\tau\} = 0 \text{ .....(3.9)}$$

หรือ $$\sum_{n=0}^{N-1} h(nT)\sin(\tau - nT)\omega = 0 \text{.....(3.10)}$$

เห็นได้ว่าการที่ตัวกรองไม่ป้อนกลับเชิงเลขจะมีคุณสมบัติผลตอบสนองเฟสเชิงเส้น ได้นั้นผลตอบสนองอิมพัลส์ต้องถูกหน่วงน้ำหนักด้วยพจน์ $\sin(\tau - nT)\omega$ ซึ่งเป็นฟังก์ชั่นซายน์และนำผลบวกรวมกันได้ผลรวมเป็น 0 การที่มีคุณสมบัติอย่างนี้ได้นั้นเมื่อนับจากจุดกึ่งกลางผลตอบสนองอิมพัลส์ทั้งสองด้าน ต้องสมมาตรกัน สมการที่ 3.9 จะเป็นจริงก็ต่อเมื่อเงื่อนไขเหล่านี้เป็นจริง

$$\tau = \frac{(N-1)T}{2} \text{..... (3.11a)}$$

$$h(nT) = h(N-1-nT) \text{..... (3.11b)}$$

โดยที่ $0 < n < (N-1)$

เงื่อนไข 3.10a หมายถึงว่า ตัวกรองไม่ป้อนกลับเชิงเลขสามารถออกแบบให้มีผลตอบสนองเฟสที่มีค่า τ_p และ τ_g คงตัวตลอดย่านความถี่ที่สนใจได้ ก็ต่อเมื่อผลตอบสนองอิมพัลส์ต้องถูกหน่วงเวลาออกไป $(N-1)T/2$ ลำดับ ส่วนเงื่อนไข 3.10b นั้นแสดงว่าผลตอบสนองอิมพัลส์ของตัวกรองแบบนี้ต้องสมมาตรกัน โดยที่ถ้าหาก N เป็นเลขคี่จุดกึ่งกลางของการสมมาตรจะอยู่ที่ผลตอบสนองอิมพัลส์ลำดับ หรือจุดที่ (N-1)/2

3.1.2 ผลตอบสนองความถี่

พิจารณาผลตอบสนองอิมพัลส์แบบสมมาตรคู่ และ สำหรับค่า N เป็นเลขคี่เมื่อเราแทนค่าผลตอบสนองอิมพัลส์ ลงในสมการที่ 2.17 ได้ว่า

$$H(e^{j\omega T}) = \sum_{n=0}^{(N-3)/2} h(nT)e^{-jn\omega T} + h\left\{\frac{(N-1)T}{2}\right\}e^{-j\omega(N-1)T/2} + \sum_{n=(N+1)/2}^{N-1} h(nT)e^{-jn\omega T} \text{.....(3.12)}$$

ซึ่งตามสมการเราเขียนแยกผลตอบสนองอิมพัลส์ได้ เป็นสามส่วนด้วยกันโดยที่ในส่วนพจน์ที่ 3 อาจเขียนใหม่ให้สามารถรวมกับพจน์ที่ 1 ได้ โดยการเขียนแทนให้ $h(nT) = h\{(N-n-1)T\}$ ซึ่งใช้ผลจาก 3.10a ดังนั้นเราแทนดรรชนี m = N-1-n

$$\sum_{n=(N+1)/2}^{N-1} h(nT)e^{jn\omega T} = \sum_{m=(N-3)/2}^{0} h(mT)e^{j\omega(N-1-m)T} \quad \text{.....(3.13)}$$

และเมื่อแทน m = n ผลที่ได้คือ

$$\sum_{n=(N+1)/2}^{N-1} h(nT)e^{jn\omega T} = \sum_{n=0}^{(N-3)/2} h(nT)e^{-j\omega(N-1-n)T} \quad \text{.....(3.14)}$$

แทนผลลัพธ์นี้ในพจน์ที่ 3 ของสมการที่ 2.28 ได้ผลเป็น

$$H(e^{j\omega T}) = e^{-j\omega(N-1)\omega T/2}\left\{h\left(\frac{N-1}{2}\right) + \sum_{n=1}^{(N-3)/2} 2h(nT)\cos(\omega\frac{(N-1-n)T}{2})\right\}$$

ซึ่งพจน์ $e^{-j(N-1)\omega T/2}$ เกิดจากการที่ผลตอบสนองอิมพัลส์ถูกหน่วงไป (N-1)/2 ทำให้เกิดการเลื่อนเฟสขึ้น และเมื่อเราเปลี่ยนตัวแปรให้ ((N-1) / 2) – n = k จะเขียนได้ใหม่เป็น

$$H(e^{j\omega T}) = e^{-j\omega(N-1)T/2}\left\{\sum a_k \cos\omega kT + a_0\right\} \quad \text{.....(3.15)}$$

โดยที่

$$a_0 = h\left\{\frac{(N-1)T}{2}\right\}$$

$$a_k = 2h\left\{\left(\frac{(N-1)}{2} - k\right)T\right\} \quad \text{.....(3.16)}$$

3.1.3 <u>ตำแหน่งของซีโร</u>

ตัวกรองไม่ป้อนกลับเชิงเลขนั้น เนื่องจากว่าไม่มีการป้อนกลับ ดังนั้นฟังก์ชั่นถ่ายโอนจริงมีซีโรเพียงอย่างเดียวและด้วยคุณสมบัติของความสมมาตรทั้งสองด้านใน z- Domain คุณสมบัตินี้แสดงได้โดย

$$H(z) = z^{-(N-1)/2} \sum_{k=0}^{(N-1)/2} \frac{a(k)}{2}(z^k + z^{-k}) \quad \text{.....(3.17)}$$

ค่า a_0 and a_k ดูได้จากสมการที่ 3.15 จากนั้นแทน z ด้วย z^{-1} ในสมการที่ 2.29 เราจะได้ว่า

$$H(z^{-1}) = z^{(N-1)} H(z) \quad \text{.....(3.18)}$$

นั่นแสดงว่า $H(z)$ และ $H(z^{-1})$ มีฟังก์ชั่นเหมือนกันแตกต่างกันที่มีการหน่วงเวลาออกไป (N-1) ลำดับ สมการที่ 3.17 แสดงว่าซีโรของตัวกรอง $H(z^{-1})$ เป็นซีโรของตัวกรอง $H(z)$ ด้วยดังนั้นถ้าให้ Z1 เป็นซีโรของตัวกรองผลตอบสนองเฟสเชิงเส้น ซีโรนี้มีคุณสมบัติสรุปได้ดังนี้

1. ถ้า $z_1 = a$ หรือเป็นซีดรค่าจริง ของตัวกรอง $H(z)$ ซีโร่ที่อยู่ที่ $z^{-1} = a^{-1}$ ก็เป็นตัวกรองของตัวกรอง $H(z)$ ด้วย
2. ถ้า $z_1 = e^{j\theta_1}$ เป็นซีโร $H(z)$ ซึ่งอยู่รอบวงของวงกลมหนึ่งหน่วย โดยที่ $\theta_1 \neq 0$ และ $\theta_1 \neq \pi$ แล้ว $z^{-1} = z_1 = e^{-j\theta_1}$ ก็เป็นซีโรของตัวกรอง $H(z)$ ด้วย
3. ถ้า $z_1 = e^{j\theta_1}$ เป็นซีโรเชิงซ้อนของ $H(z)$ โดยที่ $r_1 \neq 1, \theta_1 \neq 0, \theta_1 \neq \pi$ แล้วซีโรที่อยู่ที่ $z_1 = r_1 e^{-j\theta_1}, z_1^{-1} = \left(\frac{1}{r_1}\right) e^{-j\theta}, z^{-1} = \left(\frac{1}{r_1}\right) e^{j\theta}$ ก็เป็นซีโร่ของตัวกรอง $H(z)$ ด้วย

<u>3.2 การออกแบบโดยใช้วินโดว์</u>

ในส่วนนี้เราจะกล่าวถึงการออกแบบดิจิตอลฟิลเตอร์แบบไม่มีการป้อนกลับ ซึ่งการออกแบบฟิลเตอร์มีวิธีการหลายแบบด้วยกัน เช่น การใช้อนุกรมฟูริเยอร์ และการออกแบบโดยความถี่สุ่ม แต่ในโครงงานชิ้นนี้ใช้การออกแบบโดยวินโดว์ ดังนั้นจึงอธิบายเฉพาะแบบนี้เท่านั้น

ขอเริ่มจากถ้าให้ $H_d(\omega)$ แทนผลตอบสนองความถี่ทางอุดมคติ และ ผลตอบสนองนี้สามารถเขียนแทนด้วยอนุกรมฟูริเยอร์ความยาวอนันต์พจน์ ได้คือ

$$H_d(\omega) = \sum_{n=-\infty}^{\infty} h_d(n) e^{-jn\omega T} \quad \text{.....(3.19)}$$

โดยที่ $h_d(\omega)$ หาได้จาก

$$h_d(n) = \left(\frac{1}{2\pi}\right) \int_{-\pi}^{\pi} H_d(\omega) \bullet e^{jn\omega T} d\omega \quad \text{.....(3.20a)}$$

หรือ

$$h_d(n) = \left(\frac{1}{\omega_m}\right)\int_{-\omega_m/2}^{\omega_m/2} H_d(\omega) \bullet e^{jn\omega T}\, d\omega \text{.....(3.20}b\text{)}$$

โดย ω_m เป็นความถี่ในการสุ่มตัวอย่าง เป็นที่ทราบกันว่าการนำสมการข้างต้นมาใช้ในการออกแบบตัวกรองไม่ป้อนกลับเชิงเลขจะเกิดปัญหา 2 ประการคือ ประการแรกต้องมีการตัดปลายจากความยาวอนันต์เป็นความยาวที่มีค่าจำกัดค่าหนึ่ง หรือ

$$H_d(\omega) = \sum_{n=-(N-1)/2}^{(N-1)/2} h_d(n) e^{-jn\omega T} \text{ ....(3.21)}$$

ซึ่งผลลัพธ์ก็คือ ทำให้เกิดคลื่นบนผลตอบสนองแอมปิจูด ประการที่สองคือ เมื่อตัดปลายอนุกรมแล้วอนุกรมข้างต้นยังไม่สามารถนำไปใช้งานได้ เพราะว่าผลตอบสนองอิมพัลส์เริ่มจากลำดับที่ n = (N-1) / 2

ซึ่งในระบบเวลาจริงไม่สามารถสร้างลำดับที่มีเวลาลบได้ ข้อนี้แก้ไขได้โดยการเลื่อนหรือหน่วงผลตอบสนองอิมพัลส์ออกไปโดยให้ค่าเริ่มจาก n = 0

$$H(\omega) = \sum_{n=0}^{N-1} h(n) e^{-jn\omega T}$$

จากสมการข้างต้น จะเห็นว่าจากการที่เราตัดปลายอนุกรมฟูริเยอร์ ก็คือการที่ทำการเลือกให้

$$\begin{aligned} h(n) &= h_d(n) \quad ; 0 < n < N-1 \\ &= 0 \quad\quad\; ; n \end{aligned}$$

ซึ่งสมการนี้หากเขียนอยู่ในรูปสมการทั่วไปก็คือ การนำเอาลำดับ $h_d(\omega)$ มาคูณกับค่าลำดับจำกัด $w(n)$ หรือ

$$h(n) = h_d(n) \bullet w(n) \text{.....(3.22)}$$

โดยที่ $w(n)$ แทนลำดับของวินโดว์แบบต่างๆ ลำดับต่อไปเราจะมาทำความเข้าใจถึงคุณสมบัติผลตอบสนองความถี่ของวินโดว์สี่เหลี่ยมว่าเป็นอย่างไร

3.2.1 คุณสมบัติของวินโดว์สี่เหลี่ยม

สำหรับวินโดว์สี่เหลี่ยมอาจนิยามโดยทั่วไปได้โดย

$$\begin{aligned} w_R(n) &= 1 \quad when \ |n| \le (N-1)/2 \\ &= 0 \quad othwise, \end{aligned} \qquad \text{.....(3.23)}$$

เราจะได้

$$h(n) = h_d(n)a_R(n) \qquad \text{.....(3.24)}$$

เพราะว่าการคูณในโดเมนทางเวลาตรงกันกับการผสานกันใน แกนทางความถี่ ประเด็นการออกแบบตัวกรองแบบไม่ป้อนกลับเชิงเลข คือ การหาวินโดว์ทีมีค่าจำกัดที่ใช้การแปลงฟูริเยอร์ ที่มี Side lopes มีขนาดที่ต่ำและให้ความสำคัญกับการทำให้ main lope มีขนาดใหญ่

ผลการตอบสนองของวินโดว์ หาได้โดยการประยุกต์ผลการแปลงแซดเข้ากับลำดับ ของวินโดว์แล้วแทนค่าให้ $Z = e^{-j\omega T}$ เพื่อจัดรูปสมการใหม่จะได้

$$\begin{aligned} W_R(\omega) &= \sum_{n=-(N-1)/2}^{(N-1)/2} e^{-jn\omega T} \\ &= \left\{ \frac{e^{j\omega(N-1)T/2} - e^{-j\omega(N+1)T/2}}{1 - e^{-j\omega T}} \right\} \\ &= \left\{ \frac{e^{j\omega NT/2} - e^{-j\omega NT/2}}{e^{j\omega T/2} - e^{-j\omega T/2}} \right\} \\ &= \frac{\sin(\omega NT/2)}{\sin(\omega T/2)} \end{aligned} \qquad \text{.....(3.25)}$$

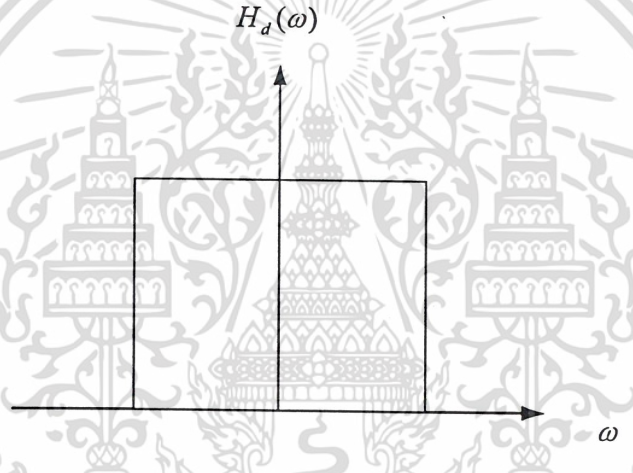
โดยที่ผลตอบสนองทางแอมปิจูดของวินโดว์ มีคุณสมบัติ คือ

1. ความกว้างของ โหลบหลัก(Main lope) มีค่า $4\pi / N$ และมีค่าขนาดเป็น N
2. เมื่อ N มีค่ามากขึ้น ความกว้างของโหลบหลักลดลง แต่ค่าขนาด N มากขึ้น
3. โหลบข้าง (Side lopes) มีการแกว่งโดยผ่านจุดศูนย์ทุกๆ ค่าความถี่ $2\pi / N$ และมีค่าขนาดลดลงไปเรื่อย และ $W_R(\omega) = 0$ ณ ค่าความถี่ $W = m\omega_m / N$ เมื่อ $m = \pm 1, \pm 2, ...$ผลนี้เห็นได้ว่าถ้า N มีค่ามากโหลบข้างจะเข้ามารวมกันใกล้ความถี่ $\omega = 0$ มากขึ้น
4. ถ้านิยามให้**พจน์ อัตราลูกคลื่น** (ripple ratio : RR)

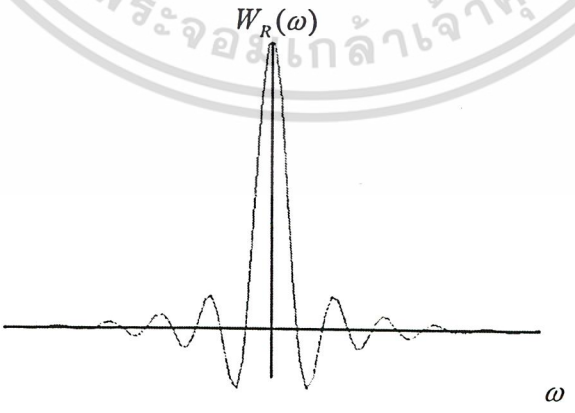
$$RR = \frac{Peak\ sidelope}{Peak\ mainlope} x100$$

การที่เรานำเอาลำดับผลตอบสนองอิมพัลส์ มาคูณกับค่าลำดับของวินโดว์ในโดเมนของเวลา เมื่อทำการพิจารณาในโดเมนของความถี่เท่ากับเป็นการนำเอาผลตอบสนองความถี่อุดมคติ มาทำการคูณการประสาน กับผลตอบสนองความถี่ของวินโดว์ หรือ

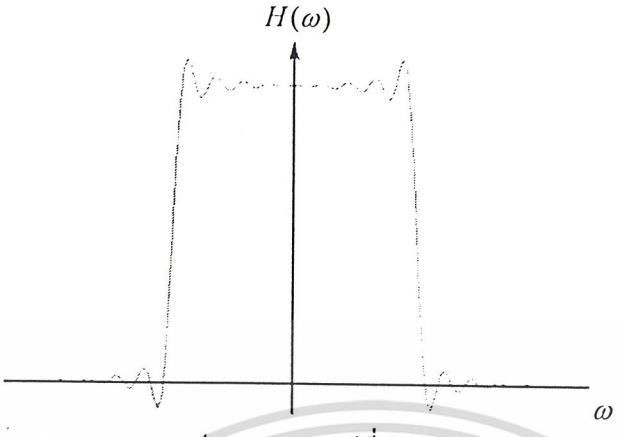
$$H_A(e^{j\omega T}) = \frac{1}{2\pi F}\int_0^{2\pi F} H(e^{j\omega T})A(e^{j(\omega-\Omega)T})d\Omega \text{.....(3.26)}$$



3.1(ก) ผลตอบสนองทางความของวงจรกรองความถี่ทางอุดมคติ



3.1(ข) ผลตอบสนองสัญญาณกระตุ้น(Impulse Respond)



3.1(ค) ผลการประสานของรูปที่ 3.1(ก) และ 3.1(ข)

รูปที่ 3.1 แสดงการทำวินโดว์ในโดเมนความถี่

ผลการประสานนี้อธิบายโดยใช้แผนภาพ ข้างต้น ซึ่งเห็นได้ชัดว่า

1. ค่าความชันของโหลบหลักของสเปกตรัมของวินโดว์ ทำให้เกิดแถบเปลี่ยนสถานะของผลตอบสนองความถี่ เพราะฉะนั้นถ้า N มาก ก็ทำให้แถบเปลี่ยนสถานะลดลง
2. ลูกคลื่นบนผลตอบสนองความถี่ เป็นผลมาจากโหลบข้าง ของสเปกตรัมของวินโดว์ ดังนั้นถ้าเลือก N ค่ามากโหลบข้างเลื่อนเข้าใกล้ความถี่ $\omega = 0$ มากขึ้นเป็นผลทำให้เกิดลูกคลื่นบนยอดผลตอบสนองความถี่มากขึ้น

ดังนั้นผลทั้งสองประการนี้เราจึงต้องหาวินโดว์หรือดัดแปลงวินโดว์สี่เหลี่ยมเพื่อให้มีคุณสมบัติคือ

1. ให้โหลบหลักของผลตอบสนองความถี่ แต่ภายในโหลบหลักสามารถครอบคลุมพลังงานของสัญญาณทั้งหมดไว้ได้มากที่สุด
2. ให้ผลตอบสนองความถี่ของวินโดว์ มีค่าขนาดของโหลบข้างน้อย และมีค่าขนาดลดลงอย่างรวดเร็ว เมื่อความถี่ลดลงอย่างรวดเร็ว

3.2.2 วินโดว์แบบฮาร์มมิ่ง ฮาน และ แบล็กแมน

ฮานวินโดว์(Han window) และฮาร์มมิ่งวินโดว์ (Hamming window) สามารถเขียนสมการของลำดับได้ โดยให้ T = 1 วินาที คือ

$$W_H = \alpha + (1-\alpha)\cos\left(\frac{2\pi n}{N-1}\right) \qquad \text{when } |n| \le \frac{(N-1)}{2}$$

$$= 0 \qquad \text{otherwise}$$

โดยที่ $\alpha = 0.5$ สำหรับกรณีเป็น ฮานมิ่งและ $\alpha = 0.54$ สำหรับกรณีฮาร์มมิ่งวินโดว์ ผลตอบสนองความถี่ของวินโดว์ทั้ง 2 แบบสามารถหาได้โดยเขียนสมการลำดับของวินโดว์ ได้คือ

$$\begin{aligned} W_H(n) &= W_R(n)\left\{\alpha - (1-\alpha)\cos\frac{2\pi n}{N-1}\right\} \\ &= \alpha W_R(n) + \left(\frac{1-\alpha}{2}\right)W_R(n)\left\{e^{\frac{j2\pi n}{N-1}} + e^{\frac{-j2\pi n}{N-1}}\right\} \quad ...(3.27) \end{aligned}$$

เมื่อประยุกต์ใช้ผลการแปลงแซดกับลำดับในสมการข้างต้น ได้ผลตอบสนองความถี่เป็น

$$\begin{aligned} W_H(e^{j\omega T}) &= \alpha W_R(e^{j\omega T}) + \frac{(1-\alpha)}{2}W_R(e^{j(\omega T - \frac{2\pi}{N-1})}) \\ &\quad + \frac{(1-\alpha)}{2}W_R(e^{j(\omega T + \frac{2\pi}{N-1})}) \quad .....(3.28) \end{aligned}$$

แทนค่า $W_R(e^{j\omega T})$ ลงในสมการ 2.41

$$\begin{aligned} W_H(e^{j\omega T}) &= \alpha\frac{\sin(\omega N/2)}{\sin(\omega/2)} + \frac{1-\alpha}{2}\bullet\frac{\sin\{(\omega N/2) - N\pi/(N-1)\}}{\sin\{(\omega/2) - \pi/(N-1)\}} \\ &\quad + \frac{1-\alpha}{2}\bullet\frac{\sin\{(\omega N)/2 + N\pi/(N-1)\}}{\sin\{(\omega/2) + \pi/(N-1)\}} \quad .....(3.29) \end{aligned}$$

สเปกตรัมของวินโดว์นี้ซึ่งเห็นได้ว่าเทอมที่2 และ 3 ของสมการ 3.28 เป็นการถ่วงน้ำหนัก $W_R(e^{j\omega T})$ แล้วเลื่อนไปทางขวาและซ้าย $2\pi/(N-1)$ ลำดับ การทำอย่างนี้จุดประสงค์นี้เพื่อไปหักล้าง กับโหลบข้างทั้งสองด้านแล้วไปเพิ่มค่าขนาดของโหลบหลัก ผลการรวมจะได้ดังรูป โดยเห็นได้ชัดว่าโหลบหลักเพิ่มขึ้นและลูกคลื่นลบลงไปอย่างมาก ความกว้างของโหลบหลักของวินโดว์นี้มีค่าโดยประมาณเท่ากับ $4W_m/N$

จากการเปรียบเทียบกันระหว่างสเปกตรัมความถี่ของวินโดว์สี่เหลี่ยม ฮานวินโดว์และฮาร์มมิ่งวินโดว์ ซึ่งเห็นได้ชัดว่าแบบสี่เหลี่ยมพลังงานในส่วนความถี่นอกโหลบมีอยู่มากส่วนแบบ ฮานและฮาร์มมิ่งพลังงานได้ถ่ายโอนในโหลบหลักเกือบ 99%

สำหรับวินโดว์แบบ แบล็กแมนวินโดว์(Blackman window) สามารถเขียนได้คือ

$$\begin{aligned} W_B(n) &= 0.42 + 0.5\cos\left(\frac{2n\pi}{N-1}\right) + 0.08\cos\left(\frac{4n\pi}{N-1}\right) \quad ;|n| \le (N-1)/2 \\ &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ;\text{otherwise} \end{aligned}$$

ฮาร์มมิ่งวินโดว์

$$a_R(n) = 0.54 + 0.46\cos\left(\frac{2\pi n}{N-1}\right) \qquad ;|n| \le (N-1)/2$$

$$= 0 \qquad ;otherwise$$

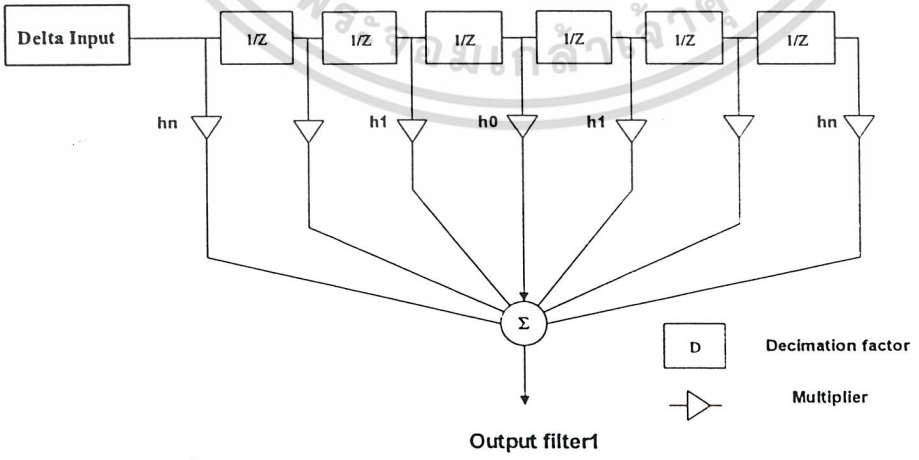
วินโดว์แบบนี้ให้คุณสมบัติที่ดีกว่าแบบฮาน และ แบบ แฮมมิง คือ สามารถถ่ายทอดพลังงานเข้าไปในโหลบหลักได้มากกว่า ดังได้แสดงคุณสมบัติเพื่อเป็นการเปรียบเทียบไว้ในตารางที่ 3.1

ชนิดของวินโดว์	Main lobe width	Ripple Ration % N=31	Peak-Amp of Side lobe	Min-stop band Att(dB)
Rectangular	$2\omega_s / N$	21.80	-13	-21
Hann	$4\omega_s / N$	2.67	-31	-44
Hamming	$4\omega_s / N$	0.82	-41	-53
Blackman	$6\omega_s / N$	0.12	-57	-74

ตารางที่ 3.1 แสดงคุณสมบัติของวินโดว์แบบต่างๆ

3.3 โครงสร้างของวงจรกรองแบบไม่ป้อนกลับเชิงเลข

จากที่เราได้ศึกษาทฤษฎีการทำงาน และการประมาณค่า ของวงจรกรองแบบไม่ป้อนกลับเชิงเลข ทั้งคุณสมบัติคสามเป็นเชิงเส้นทางเฟส และ ผลตอบสนองทางขนาด มาแล้วในส่วนต่อไปนี้เป็นการนำ Differential equation ของวงจรไม่ป้อนกลับเชิงเลขมาสร้างเป็นโครงสร้างทางดิจิตอล



รูปที่ 3.2 โครงสร้างแบบตรงของตัวกรองไม่ป้อนกลับเชิงเลข

ต่อไปเรามาพิจารณาสมการของตัวกรองไม่ป้อนกลับเชิงเลขกันอีกครั้ง เพื่อหาโครงสร้างทางดิจิตอลในรูปแบบอื่นๆ

$$H(z)=\sum_{n=0}^{N-1}h(n)z^{-n} \quad \text{.....(3.30)}$$

หาสมการ Differential Equation โดยการทำส่วนกลับของการแปลงแซดจากสมการที่ 2.44 ให้ผลได้ว่า

$$y(iT)=\sum_{n=0}^{N-1}h(n)x[(i-n)T] \quad \text{.....(3.31)}$$

ซึ่งยังคงจำได้ว่าสมการที่ 2.45 เป็นผลบวกของการประสาน สำหรับโครงสร้างแบบตรง จะให้ผลเป็นดังรูปที่ 2.25 จากที่เราทำการศึกษาคุณสมบัติกันมาแล้ว จากคุณสมบัติความเป็นเชิงเส้นทางเฟส

$$h(n)=h(N-1-n) \quad \text{.....(3.32)}$$

$$H(e^{j\omega T})=e^{-j\omega(N-1)\omega T/2}\left\{h\left(\frac{N-1}{2}\right)+\sum_{n=1}^{(N-3)/2}2h(nT)\cos(\omega\frac{(N-1-n)T}{2})\right\}$$

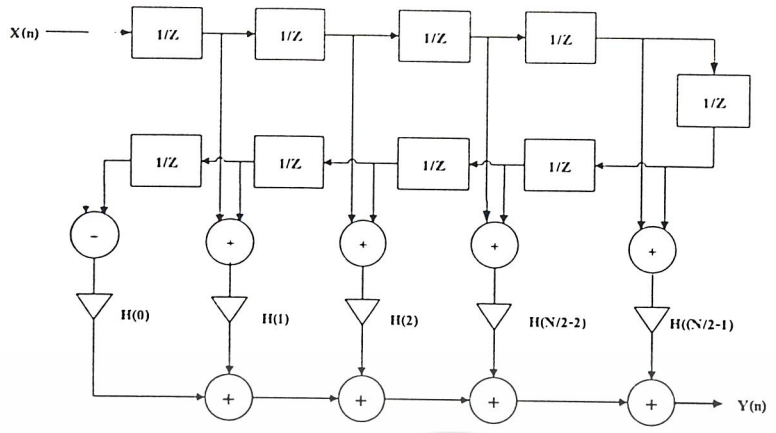
จากที่เราได้การพิจารณาสมการข้างต้นมาบ้างแล้ว สำหรับกรณีที่ N เป็นเลขจำนวนคี่ และแทน $z=e^{j\omega T}$ สามารถจัดสมการได้ ดังนี้

$$H(z)=\sum_{n=0}^{(N-3)/2}h(n)\left[z^{-n}+z^{-(N-1-n)}\right]+h\left(\frac{N-1}{2}\right)z^{-[(N-1)/2]} \quad \text{.....(3.33)}$$

เช่นเดียวกันเมื่อ N เป็นเลขคู่

$$H(z)=\sum_{n=0}^{N-1}h(n)\left[z^{-n}+z^{-(N-1-n)}\right] \quad \text{.....(3.34)}$$

ต่อไปเป็นการพิจารณาสมการ Differential equation ของสมการที่ 3.32 และ 3.33 โดยทำส่วนกลับของการแปลงแซด กรณี N เป็นเลขคี่

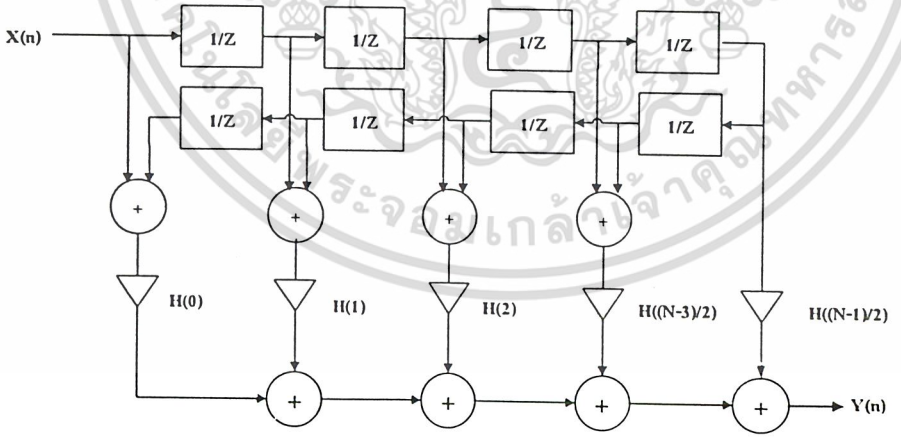


รูปที่3.3 โครงสร้างแบบตรงของตัวกรองแบบไม่ป้อนกลับ แบบสมมาตรคู่

$$y(iT) = \sum_{n=0}^{(N-3)/2} h(n)\{x[(i-n)T] + x[(i+n+1-N)T]\} + h\left(\frac{N-1}{2}\right)x\left[\left(i-\frac{N-1}{2}\right)T\right] \quad \text{.....(3.35)}$$

เช่นเดียวกันกรณี N เป็นเลขคู่

$$y(iT) = \sum_{n=0}^{N/2-1} h(n)\{x[(i-n)T] + x[(i+n+1-N)T]\} \quad \text{.....(3.36)}$$



รูปที่ 3.4 โครงสร้างแบบตรง ของตัวกรองแบบไม่ป้อนกลับ แบบสมมาตรคี่

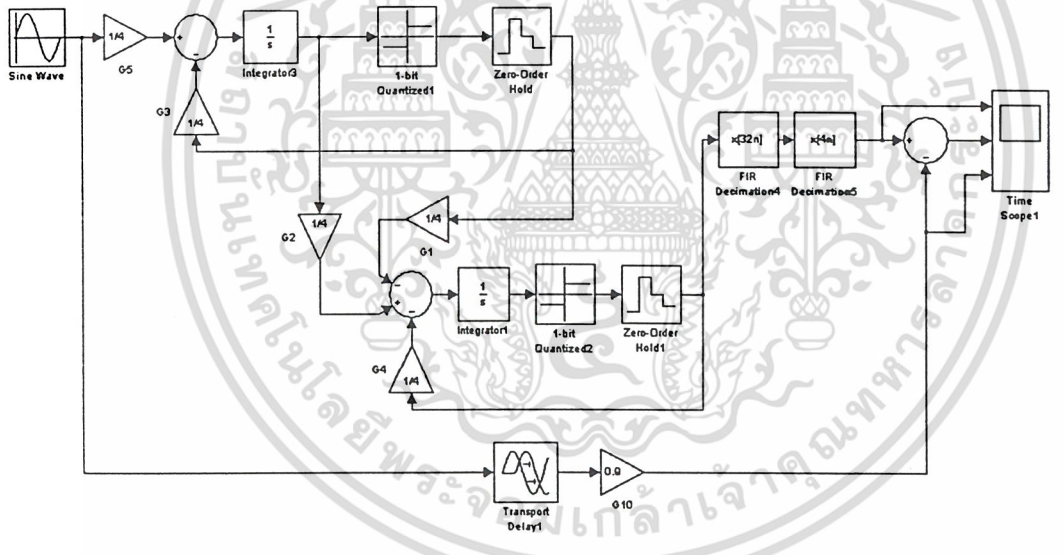
บทที่4

การวิเคราะห์และการออกแบบ

4.1 การออกแบบ ซิกมา-เดลตา มอดดูเลชั่นอันดับสอง

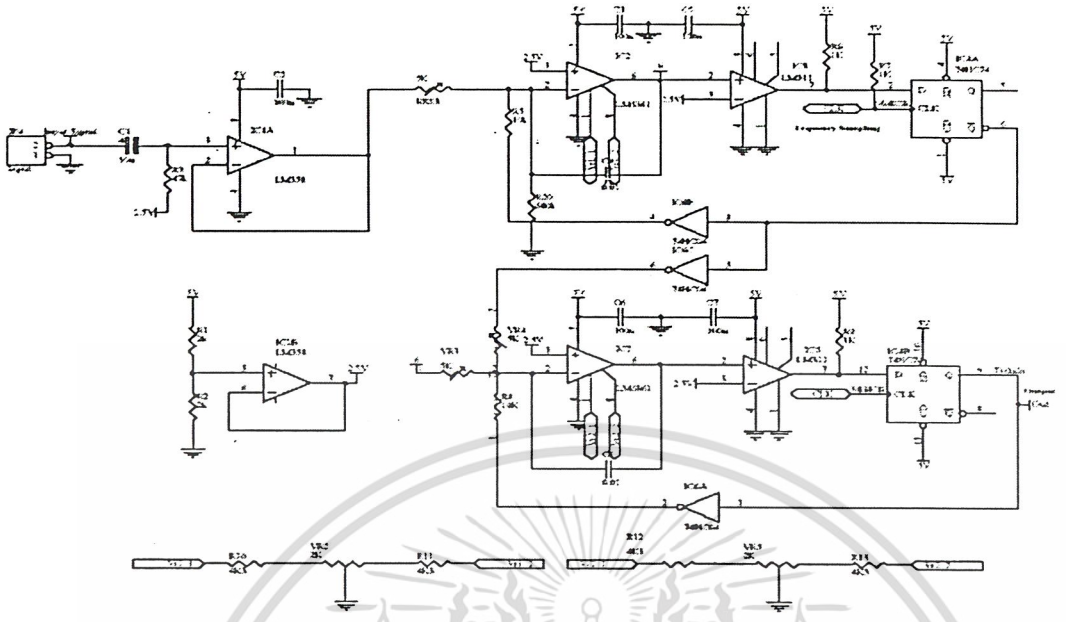
ในกรออกแบบ ซิกมา-เดลตามอดดูเลชั่นนี้จะอาศัยผลของการทดลอง Simulation ด้วยSimulink ของ Mathlab และอาศัยบล็อกแต่ละบล็อกของ Simulink มาสังเคราะห์เป็นวงจรทางไฟฟ้า โดยวงจรมอดดูเลชั่นแบบ 2-order จะมีความคล้ายคลึงกับแบบ 1-order ซึ่งก็เป็นการนำเอา 1 ออเดอร์ มาต่อกันเป็น 2 Loop ทั้งนี้เพื่อผลในการกำจัดสัญญาณรบกวน จากรูปที่ 4.1 สามารถแยกแต่ละส่วนออกได้เป็นส่วนหลัก ๆ ได้ คือ วงจรอินทิเกรเตอร์ ส่วนที่เป็นการประมาณค่าแบบ 1 – บิท และส่วนที่กำหนดอัตราการสุ่มและคงค่าไว้

สำหรับรูปการทดลองด้วย Mathlab แสดงได้ดังนี้



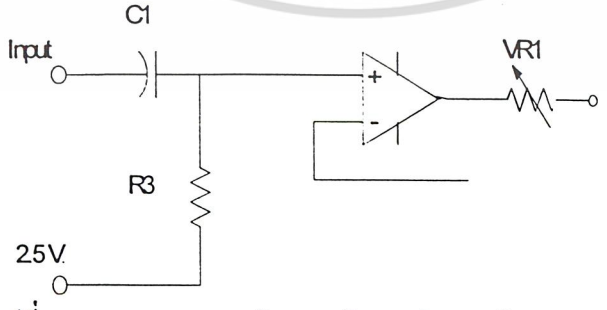
รูปที่ 4.1 Simulink ใน Mathlab ของซิกมา-เดลตา มอดดูเลชั่น

หลังจากที่ดูบล็อกการทำงานแล้วขั้นต่อไปเป็นการนำบล็อกแต่ละบล็อกมาสร้างเป็นวงจรอิเล็กทรอนิกส์ที่สามารถใช้งานได้จริงในทางฮาร์ดแวร์ โดยจะแยกพิจารณาออกเป็นส่วนๆตามรูปที่4.2 จากที่ทำการสร้างวงจรการมอดดูเลชั่นอันดับ 2 แล้วจะได้วงจรที่สมบูรณ์ดัง รูปที่ 4.2 พร้อมทั้งในส่วนถัดไป จะอธิบายการทำงานและแนวการออกแบบ



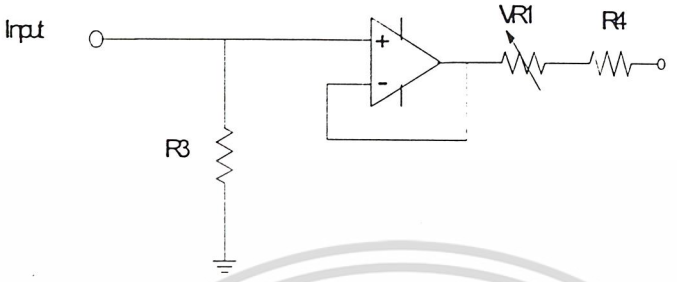
รูปที่ 4.2 วงจร ซิกมา-เดลตา มอดดูเลชั่น อันดับ 2

จากรูปที่ 4.2 IC1 ซึ่งเป็น Opamp เบอร์ LM358 ซึ่งเป็น Opamp แบบ กินพลังงานต่ำ มี GB (Gain bandwidth) ที่ 1 MH_z ดังนั้นจึงสามารถใช้งานกับโครงงานชิ้นนี้ได้เนื่องจากเราต้องการให้ใช้งานที่ความถี่น้อยกว่า 20 KH_z IC1B เป็น Opamp ที่ต่อเป็น Buffer อัตราขยายเท่ากับ 1 ที่ขา 5 (Non-Inverting) มี R1 และ R2 ค่า 2 $K\Omega$ ต่ออยู่เพื่อสร้างแรงดันที่เอาท์พุทของ opamp ให้คงที่อยู่ 2.5 V. , IC1A การทำงานของ opamp ตัวนี้มีหน้าที่ในการรวมสัญญาณอินพุทที่ต้องการวัดกับแรงดัน offset 2.5 V. จาก IC1B ทั้งนี้ก็เพราะว่าวงจรทั้งหมดของเราใช้ Single Supply จาก 0-5 V. แต่สัญญาณอินพุทเป็นผลรวมของคลื่นซายน์มีการแกว่งจากบวก-ลบ จึงต้องมีการเพิ่มสัญญาณ DC เข้าไปเพื่อไม่ให้สัญญาณถูกขลิบในซีกลบการออกแบบสามารถทำได้ดังนี้ คือ จากวงจร



รูปที่ 4.3 แสดงวงจรรวมสัญญาณอินพุทกับแรงดันออฟเซต

มองดูจากวงจรลักษณะคล้ายกลับ Summing Amp ในกรณีการวิเคราะห์แบบ AC สามารถเขียนวงจรใหม่ได้เป็น

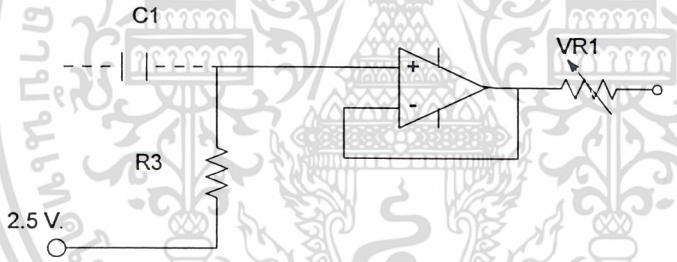


รูปที่ 4.4 แสดงวงจรในกรณีวิเคราะห์แบบ AC

จะพบการต่อในลักษณะนี้เป็นการต่อแบบ บัฟเฟอร์อัตราขยายจะเท่ากับ 1 ดังนั้นจึงได้ว่า

$\therefore$ Vout = Input

วิเคราะแบบ DC ค่า C ทุกตัวจะมองเป็น Open-Circuit จึงยุบวงจรได้เป็น

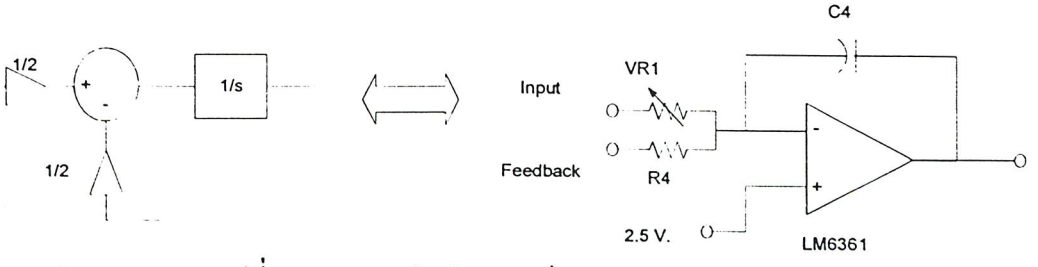


รูปที่ 4.5 แสดงวงจรในกรณีวิเคราะห์แบบ DC

เช่นเดียวกัน Vout = 2.5 V.

$\therefore$ Vout (T) = Vout (AC) + Vout (DC) = Vinput + 2.5 V.

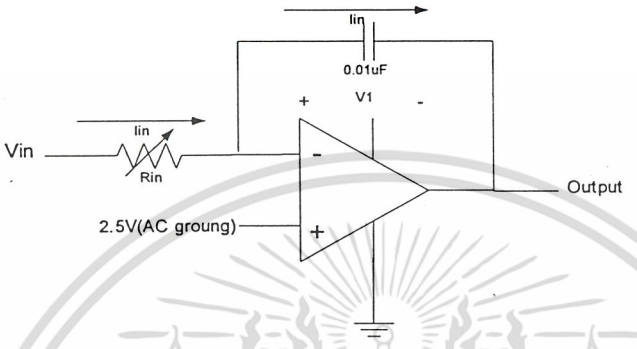
ต่อไปเป็นส่วนของการพิจารณาส่วนของวงจรอินทิเกรเตอร์ และ Summing Node



รูปที่ 6 แสดงวงจรอินทิเกรเตอร์และ Summing Node

IC2 LM6361 เป็นวงจรทำหน้าที่เป็นอินทิเกรเตอร์ LM6361 เป็น opamp ที่มีความกว้างของ Babwidth สูงถึง 50 MH_z ที่เลือกใช้เนื่องจากว่าในโครงงานชิ้นนี้ใช้ความถี่ในการ Sampling 4.8 MH_z จำเป็นต้องใช้ opamp ที่มีความถี่สูง การวิเคราะห์วงจรอินทิเกรเตอร์ทำได้ดังนี้

แยกพิจารณาจากทีละแหล่งจ่ายอันดับแรกพิจารณาจากสัญญาณอินพุท



รูปที่ 4.7 แสดงวงจรอินทิเกรเตอร์ที่พิจารณาเฉพาะสัญญาณอินพุท

จากรูปที่ 4.7 เริ่มวิเคราะห์วงจรอินทิเกรเตอร์ โดยพิจารณาที่ขาอินเวอร์ติ้งเนื่องจาก หลักพื้นฐานการวิเคราะห์ออปแอมป์ ที่ขานี้จะมีแรงดันเท่ากับที่ขา นอนอินเวอร์ติ้ง (กราวด์เสมือน) ดังนั้นจึงกำหนดว่า กระแสที่ไหลเข้าที่จุดนี้มีค่าเป็นบวก ตรงกันข้ามกระแสที่ไหลออกกำหนดเป็นเครื่องหมายลบ โดยที่กระแสไหลเข้าเท่ากับไหลออกไม่มีกระแสไหลเข้าออปแอมป์ จะได้ว่า

$$\frac{V_{in}}{R_{in}} = -C\frac{dV}{dt}$$

$$\frac{dV}{dt} = -\frac{V_{in}}{CR_{in}}$$

$$dV = -\frac{V_{in}}{CR_{in}}dt$$

ดังนั้นสัญญาณทางเอาท์พุทจะได้

$$V_{01} = -\frac{1}{CR_{in}}\int V_{in}dt \qquad .(1)$$

ต่อไปจะเป็นการพิจารณาสัญญาณที่เกิดจากการป้อนกลับของวงจรมอดดูเลชั้น วิธีการคิดจะเหมือนกับกรณีแรก แต่จะแตกต่างกันตรงค่า ความต้านทานจะมีค่าคงที่ตลอด

เพราะฉะนั้นสัญญาณเอาท์พุทที่เกิดจากสัญญาณป้อนกลับจะได้ว่า

$$V_{0.2} = -\frac{1}{CR_f}\int Vfdt \qquad ..(2)$$

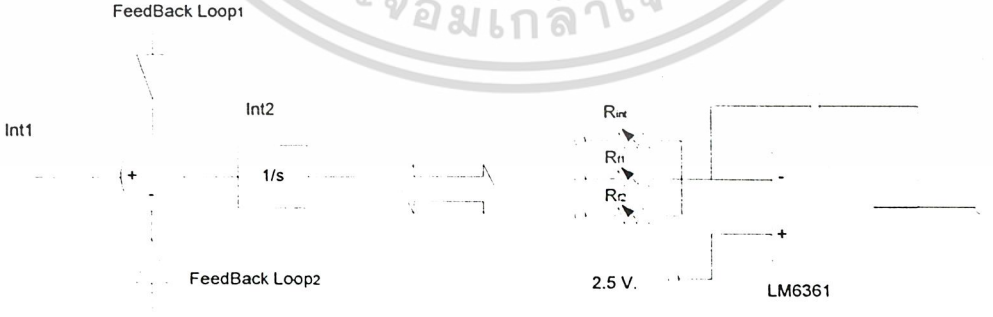
จากนั้นนำสัญญาณเอาท์พุทที่ได้จากแต่ละแหล่งจ่ายมาบวกกัน จะได้เป็นสัญญาณเอาท์พุทจริงขอ งวงจรอินทิเกรเตอร์ และบวกกับแรงดันออฟเซตที่ขานอนอินเวอร์ติ้ง 2.5 โวลต์ ได้ว่า

$$V_{out} = V_{01} + V_{02}$$
$$= -\frac{1}{CR_{in}}\int V_{in}dt - -\frac{1}{CR_f}\int V_f\,dt + 2.5$$
$$V_{out} = 2.5 - \left\{\frac{V_{in}}{sCR_{in}} + \frac{V_f}{sCR_f}\right\}$$

ต่อจากกระบวนการอินทิเกรต ระหว่างสัญญาณ อินพุท กับค่าความผิดพลาดของวงจรก็จะเป็นกระบวนการที่เรียกว่า " การควอนไตซ์ หรือการประมาณค่า " ในการมอดดูเลชั่น แบบซิกมานี้จะใช้การควอนไตซ์เพียง 1 ระดับซึ่งถือเป็นเอกลักษณ์ของการแปลงอะนาลอกเป็นดิจิตอล แบบนี้ในโครงงานนี้เราเลือกใช้ LM311 คอมพาราเตอร์เป็นตัวปรับระดับ ซึ่งมีเพียง 2 ค่าคือ 1 และ 0 เท่านั้น ความต้านทานที่ต่ออยู่ที่ขาเอาท์พุทของ คอมพาราเตอร์ ถูกต่อขึ้นไฟเลี้ยงทั้งนี้เนื่องจากว่าเอาท์พุทของ LM311 เป็นแบบ Open Drain มี 2 สถานะคือ Z และ 0 จึงต้องต่อ R ค่า 1 KΩ Pull Up ไว้ด้วย ทั้งนี้ที่ขา นอน-อินเวอร์ติ้ง จะป้อนแรงดันคงที่ไว้ที่ 2.5 โวลต์ เพราะอินพุทของ คอทพาราเตอร์ก็มีแรงดันออฟเซต 2.5 โวลต์เช่นกัน

สุดท้ายของมอดดูเลชั่น Loop ที่ 1 คือ ส่วนที่เป็นตัวที่สุ่มสัญญาณและคงสถานะ โดยใช้ D Flip Flop เป็นตัวกำหนดอัตราการสุ่ม ของวงจรและคงค่าสถานะก่อนไว้

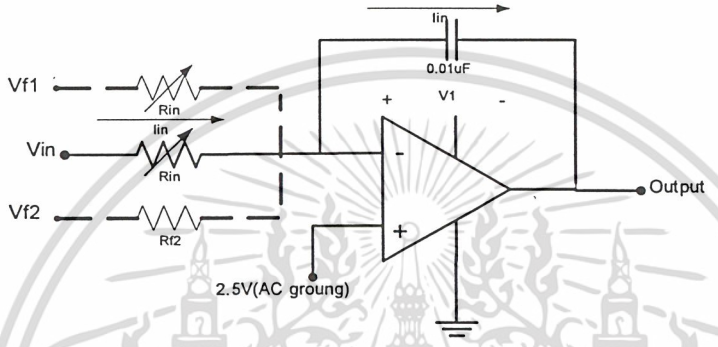
ต่อไปเป็นการพิจารณามอดดูเลชั่น Loop ที่ 2 ซึ่งมีความคล้ายคลึงกับมอดดูเลชั่นอันดับ 1 มากจะแตกต่างกันในเฉพาะในวงจรส่วน อินทิเกรต เท่านั้น คือ



รูปที่ 4.8 แสดงวงจรส่วนอินทิเกรเตอร์ในส่วนของมอดดูเลชั่น Loop ที่ 2

อินพุท ของอินทิเกรเตอร์ เป็น Summing จาก 3 แหล่งจ่าย โดยอินพุทจาก การป้อนกลับจากวงจรมอดดูเลชั้นอันดับหนึ่ง และ จาก เอาท์พุทของอินทิเกรเตอร์ตัวที่ 1 ความแตกต่างระหว่าง 2 แหล่งจ่ายหรือแรงดันตกคร่อม ตัวประมาณค่า 1 - บิท ก็คือค่าความผิดพลาด หรืออาจจะเรียกว่าสัญญาณรบกวนที่เกิดจากกระบวนการของการประมาณค่า ซึ่งเป็นปริมาณที่ไม่สามารถหลีกเลี่ยงได้ในกระบวนการแปลงอะนาลอกเป็นดิจิตอล

การวิเคราะห์เหมือนกันกับในมอดดูเลชั่นอันดับหนึ่ง ที่ได้วิเคราะห์มาแล้ว



รูปที่ 4.9 แสดงวงจรอินทิเกรเตอร์ที่พิจารณาเฉพาะ

ดังนั้นสัญญาณเอาท์พุทที่ได้ คือ

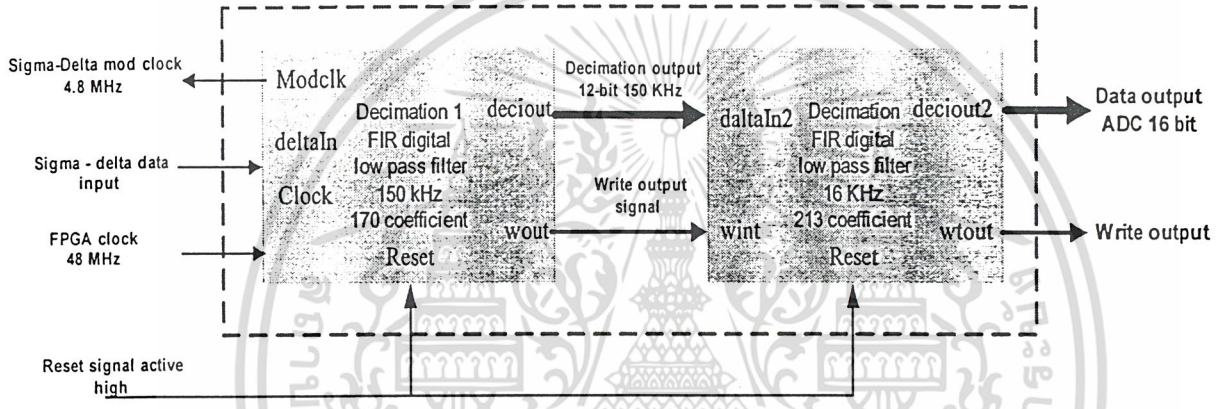
$$V_{out} = -\left\{\frac{1}{CR_{f1}}\int V_{f1}dt + \frac{1}{CR_{f2}}\int V_{f2}dt + \frac{1}{CR_{in}}\int V_{in}dt\right\} + 2.5$$

$$V_{out} = 2.5 - \left\{\frac{V_{f1}}{sCR_{f1}} + \frac{V_{f2}}{sCR_{f2}} + \frac{V_{in}}{sCR_{in}}\right\}$$

จากการวงจรในรูปที่ 4.2 ค่าความต้านทานที่ต่อเข้าอินพุทของอินทิเกรเตอร์จะเป็น ตัวต้านทานที่สามารถเปลี่ยนค่าได้ทั้งนี้เพื่อว่า เราต้องการปรับค่าคงที่ทางเวลา (Time constant) ของอินทิเกรเตอร์ และมีความยุ่งยากในการที่จะระบุค่าความต้านทานที่เหมาะที่สุดลงไปได้ ทั้งนี้การปรับค่าคงที่ทางเวลามีผลต่อประสิทธิภาพของการแปลงอะนาลอกเป็นดิจิตอล ดังนั้นเราเพิ่มความยืดหยุ่นโดยการปรับค่าคงที่ทางเวลาเพื่อหาจุดที่ได้สัญญาณเอาท์พุทที่ดีที่สุด

4.2 กรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลข(FIR digital lowpass filter)

จากเนื้อหาในส่วนที่เป็นทฤษฎีเราได้อธิบายและกล่าวถึงไปแล้วว่า เพราะเหตุใดเราจึงเลือกใช้วงจรกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลข ดังนั้นในส่วนนี้เราจะมาทำความเข้าใจเกี่ยวกับการออกแบบโดยละเอียด และอธิบายความสำคัญในแต่ละส่วนว่ามีกระบวนการคิดอย่างไร และมีการออกแบบเช่นไร ในขั้นต้นจะกล่าวถึงโครงสร้างโดยรวมทั้งหมดของวงจรกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลข ว่าแบ่งออกเป็นกี่ส่วนจากนั้นจึงจะเข้าไปพูดถึงในแต่ละส่วนโดยละเอียด



รูปที่4.10 บล็อกของวงจรกรองความถี่ไม่ป้อนกลับเชิงเลข

จากรูปที่ 4.10 เป็นส่วนประกอบของวงจรกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลข ส่วนที่อยู่ในกรอบเส้นประ คือ ตัวกรองความถี่โดยออกแบบบน FPGA โดยแบ่งออกเป็นสองภาค ภาคแรกจะรับรหัสดิจิตอล 1 – บิทที่ผ่านกระบวนการมอดดูเลชั่นแบบซิกมา – เดลตา และควบคุมอัตราการสุ่ม ซึ่งในโครงงานชิ้นนี้ใช้ความถี่สุ่มที่ 4.8 MHz เอาท์พุทของวงจรกรองชุดแรกจะแปลงจาก อินพุท 1-บิทเป็นดิจิตอลขนาด 12-บิท ด้วยความถี่สุ่ม 150 KHz กระบวนการลดความถี่สุ่มนั้นจะได้มีการกล่าวถึงในส่วนต่อไป

อินพุทของวงจรกรองชุดที่สองเป็นดิจิตอล 12-บิท สำหรับความถี่ตัดของตัวกรองชุดนี้เท่ากับ 20KHz กระบวนการหลักๆ แล้วทั้งสองชุดมีวิธีการคิดคล้ายๆกัน คือ การประสาน (Convolution): ซึ่งเป็นกระบวนการหลักของวงจรกรองความถี่ไม่ป้อนกลับเชิงเลขในหัวข้อต่อจะขอกล่าวถึงรายละเอียดของวงจรกรองแต่ละส่วนว่ามีกระบวนการทำงานอย่างไรพร้อมทั้งอธิบายแนวคิด

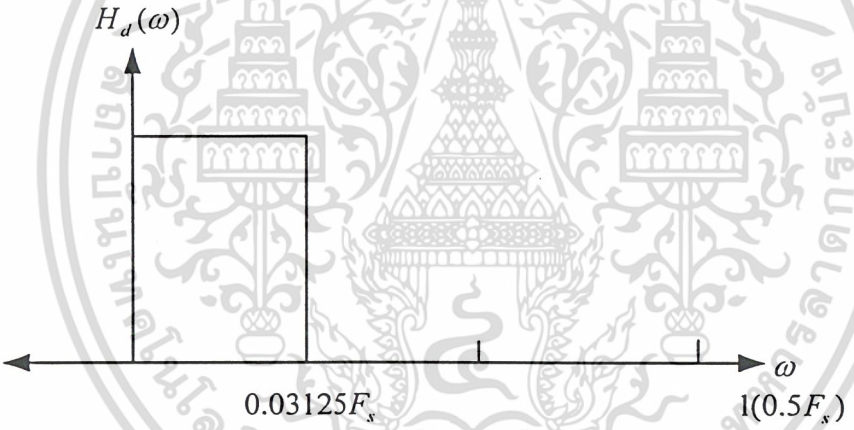
4.2.1 วงจรกรองความถี่ต่ำภาคแรก

วงจรกรองความถี่ต่ำผ่านภาคแรกดังที่ได้กล่าวนำไปแล้วว่า รับ 1– บิท อินพุทจากซิกมา – เดลตามอดดูเลชั่นอันดับสอง ด้วยความถี่ในการสุ่มที่ 4.8 MHz ดังนั้นอันดับต่อไปเป็นการออกแบบกำหนดความถี่ต่ำผ่านพร้อมกับการลดค่า สุ่มสัญญาณ

วงจรกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลข ออกแบบให้ลดค่าอัตราการสุ่ม ลดลง 32 เท่า ดังนั้นความถี่สุ่มที่ออกมา

$$F_s = \frac{4.8MHz}{32} = 150KHz$$

ด้วยทฤษฎีการแปลงสัญญาณอะนาลอกเป็นดิจิตอล ความถี่สูงสุดที่สามารถใช้งานได้ ต้องมีไม่เกิน ครึ่งหนึ่งของความถี่สุ่ม จากข้างต้นความถี่สุ่มเท่ากับ 150 KHz ดังนั้นความถี่ตัดของชุดกรองความถี่เท่ากับ 75 KHz อันดับต่อไปเป็นการสร้าง สัมประสิทธิ์การประสานหรือค่าผลตอบสนองสัญญาณกระตุ้น



รูปที่ 4.11 ผลตอบสนองทางความถี่ชุดกรองความถี่ต่ำผ่านที่ใช้ออกแบบ

รูปที่ 4.11 แสดงผลตอบสนองในทางอุดมคติซึ่งใช้อ้างอิงในการออกแบบ ซึ่งจากรูปความถี่ตัดมีค่า 0.0156 เท่าของความถี่สุ่ม สำหรับผลตอบสนองทางความถี่ของวงจรกรองความถี่ทางดิจิตอลจะมีค่าเป็นคาบ ซึ่งเท่ากับความถี่สุ่ม ความถี่ตัดมีค่าเท่ากับ $0 < \omega_c < 1$, 1 ตรงกับ 0.5 เท่าความถี่สุ่ม ความถี่ตัดหาได้จาก

$$\omega_c = \frac{2 * F_C}{F_S} = \frac{2 * 75KHz}{4.8MHz}$$

$$= 0.03125$$

เมื่อหาความถี่ตัดได้อันดับต่อไปจะนำค่าความถี่ตัดที่ได้ ไปใช้ในการสร้างสัมประสิทธิ์การประสานหรือผลตอบสนองสัญญาณกระตุ้น ในโครงงานชิ้นนี้เราเลือกใช้ วิธีการสร้างผลตอบสนองสัญญาณกระตุ้นแบบ แฮมมิ่ง จากหลายๆแบบที่ได้กล่าวไว้ในส่วนทฤษฎี

จากผลตอบสนองความถี่นี้ เราทำการหาผลตอบสนองอิมพัลส์ h(n) โดยใช้สมการดังนี้

$$w(n) = \int_0^{0.03125} \cos(n\pi\omega)d\omega = \left|\frac{\sin(n\pi\omega)}{n\pi}\right|_0^{0.03125}$$

$$w(n) = \frac{\sin(0.03125n\pi)}{n\pi} \qquad n = 1, \ldots M$$

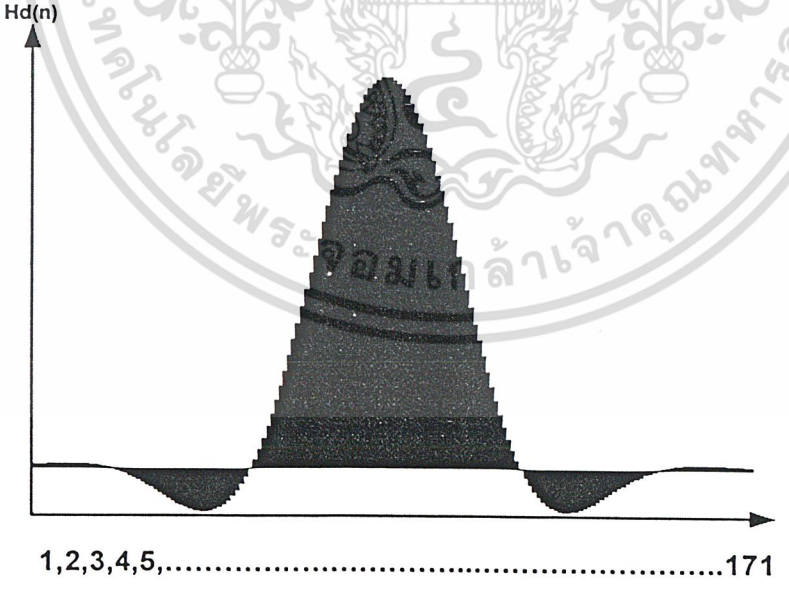
เมื่อ M = 171 หรือ

$$H(z) = \sum_0^{171} w(nT)z^{-n}$$

ในโครงงานชิ้นนี้เลือกใช้ การออกแบบวินโดว์ของ ฮามมิ่ง

$$h_m(n) = 0.54 + 0.46 * \cos\left(\frac{2n\pi}{N-1}\right)$$

ดังนั้นสามารถแสดงผลตอบสนองอิมพัลส์ได้ ดังรูปที่ 4.12 ด้านล่าง



รูปที่ 4.12 ผลตอบสนองอิมพัลส์ที่ความถี่ตัด 75 KHz

การหาค่าผลตอบสนองสามารถใช้ โปรแกรม Mathlab โดยการใช้คำสั่ง คือ $h(n) = fir1\,(170,\ 0.03125)$ เมื่อใช้คำสั่งนี้โปรแกรมจะสร้างผลตอบสนองอิมพัลส์ของ แฮมมิ่ง ความถี่ตัดที่ 0.03125 เท่าของความถี่สุ่ม และสร้างจำนวนสัมประสิทธิ์ทั้งสิ้น 171 ตัว ตัวอย่างเช่น

H(85) = 0.0311

H(86) = 0.0312

•

•

H(170) = 2.807e-004

h(171) = 2.637e-004

จากค่าที่ได้พบว่าบางค่าจะมีค่าน้อยมากและมีค่าใกล้เคียงกันและมีค่าน้อยมากโดยเฉพาะค่าขอบทางด้านซ้ายและขวาของผลตอบสนองอิมพัลส์ (ดูจากรูปที่ 4.12) ดังนั้นเมื่อนำไปแปลงเป็นค่าทางดิจิตอล อาจให้ค่าออกมาที่ไม่แตกต่างกันเลย เพื่อแก้ปัญหาดังกล่าวเราสามารถหาค่าคงที่ค่า ใดๆ มาคูณกับผลตอบสนองอิมพัลส์ โดยที่เมื่อคูณค่าสูงสุดแล้วค่าที่ได้ไม่เกิน 0.5 จะได้ $h(n) = fir1\,(170,\ 0.03125)$ *16 ผลคูณที่ได้จะไม่มีผลต่อการตอบสนองทางความถี่ของชุดกรองความถี่ จะมีผลในแง่ของการขยายสัญญาณเอาท์พุทให้มีขนาดใหญ่ขึ้น

ค่าสัมประสิทธิ์ที่ได้จะถูกแปลงเป็นเลขดิจิตอลขนาด 12 บิทๆ ที่ 12 จะเป็นบิทแสดงเครื่องหมายเพราะผลตอบสนองอิมพัลส์มีทั้งค่าบวก และ ลบ 11 บิทที่เหลือจะเป็นขนาด ข้อมูลนี้จะถูกเก็บไว้ในรอม(ROM: Read only memory) เพื่อเป็นค่าสัมประสิทธิ์การประสาน ดังเช่นว่าจากสัมประสิทธิ์ด้านบนแปลงเป็นดิจิตอล 12 – บิทได้ดังนี้

$$H(85) = 0.0311*16 = 0.497$$

$$X = \frac{0.497 * 2^{11}}{0.5} = 2034$$

$$Digital = 011111101100$$

1. การเก็บค่าสัมประสิทธิ์ภายในรอม

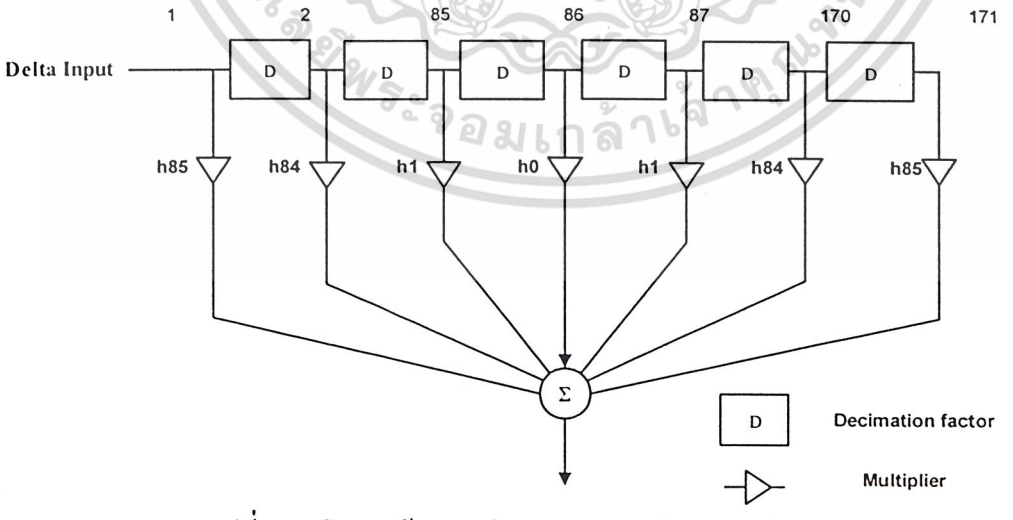
ในการสร้างรอมตัวโปรแกรม (MAX –Plus 2) จะมีเครื่องมือที่ใช้ในการสร้างโดยที่เราไม่ต้องเขียนโปรแกรมสร้างขึ้นมาเอง เพียงแต่กำหนดจำนวนตำแหน่งข้อมูลทั้งหมดที่ต้องการ และจำนวนบิทของข้อมูลที่ต้องการ และ เลือกว่าให้มีการสร้างด้วยโครงสร้างภาษาอะไร หลังจากนั้นโปรแกรมจะสร้างไฟล์ขึ้นมาให้และสามารถเรียกใช้ได้เลย แต่ทั้งนี้ต้องมีการกำหนดข้อมูลที่จะถูกบันทึกเข้าไปไว้ภายในรอม ซึ่งก็คือสัมประสิทธิ์ของเรานั่นเอง ต่อไปจะกล่าวถึงการสร้างไฟล์ที่เป็นข้อมูลของรอม

```
Width = 12;
Depth = 256;
address_radix = HEX;
data_radix = BIN;
Content begin
00      :       000000001101;
01      :       000000000110;
02      :       000000000111;
03      :       000000001001;
End;
```

เขียนไฟล์ดังตัวอย่างด้านบน จากนั้นเก็บ (Save) เป็นนามสกุล .mif ภายใต้ Content begin จนถึง end เป็นข้อมูลที่เราต้องใส่ลงไปโดยเรียงจากตำแหน่ง 00 ถึง 255 (จะใส่ทั้งหมดหรือไม่ก็ได้) ไฟล์นี้ควรสร้างก่อนที่จะสร้างรอมเนื่องจากในขั้นตอนการสร้างรอมจะมีการอ้างถึง ไฟล์ๆนี้ แต่ถ้าสร้างตอนหลังก็สามารถเพิ่มลงไปได้

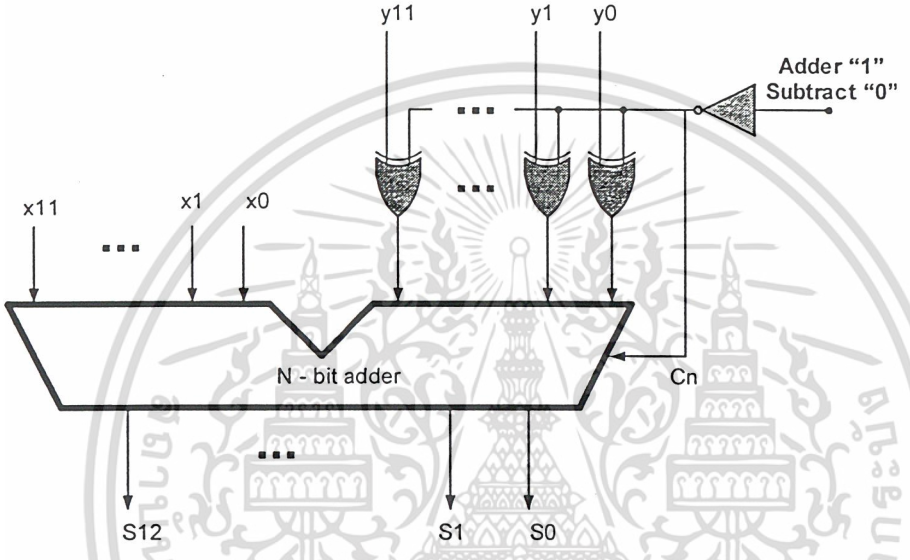
2. โครงสร้างและการออกแบบการประสาน

การประสานถือเป็นหัวใจของการออกแบบวงจรกรองความถี่แบบไม่ป้อนกลับเชิงเลข โดยในขั้นตอนนี้จะทำการกรองความถี่ที่มากกว่าความถี่ตัดออกไป และลดอัตราการสุ่มสัญญาณอินพุทลง 32 เท่า ดังนั้นเราจึงต้องเข้าโครงสร้างของการกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลขก่อน



รูปที่ 4.13 โครงสร้างการประสานและลดอัตราการสุ่ม

รูปที่ 4.13 แสดงโครงสร้างแบบตรงของการประสาน และ ลดอัตราการสุ่มลง จากรูปจะพบว่ามีส่วนที่เป็นการคูณ แต่ว่าอินพุทของชุดกรองความถี่ เป็นดิจิตอลเพียง 1 บิท เท่านั้นในการออกแบบจริงชุดวงจรคูณ สามารถใช้การบวกแทนได้ ดังรูปที่ 4.14 วงจรบวก/ลบ เลขที่ใช้แทนชุดวงจรคูณ สาเหตุที่ต้องบวกและลบเนื่องจากว่า โดยทฤษฎีแล้วเอาท์พุทของ ซิกมา – เดลตา มอดดูเลชั่นมีค่า 1 และ -1 เท่านั้น แต่ในกรณีนี้เราใช้งานแบบดิจิตอล ซึ่งมีสถานะ 1 และ 0 เท่านั้นดังนั้นในกรณีที่อินพุทเป็น 0 จะต้องเสมือนว่าคูณ (ชุดวงจรบวก) ด้วยเลขลบนั่นเอง



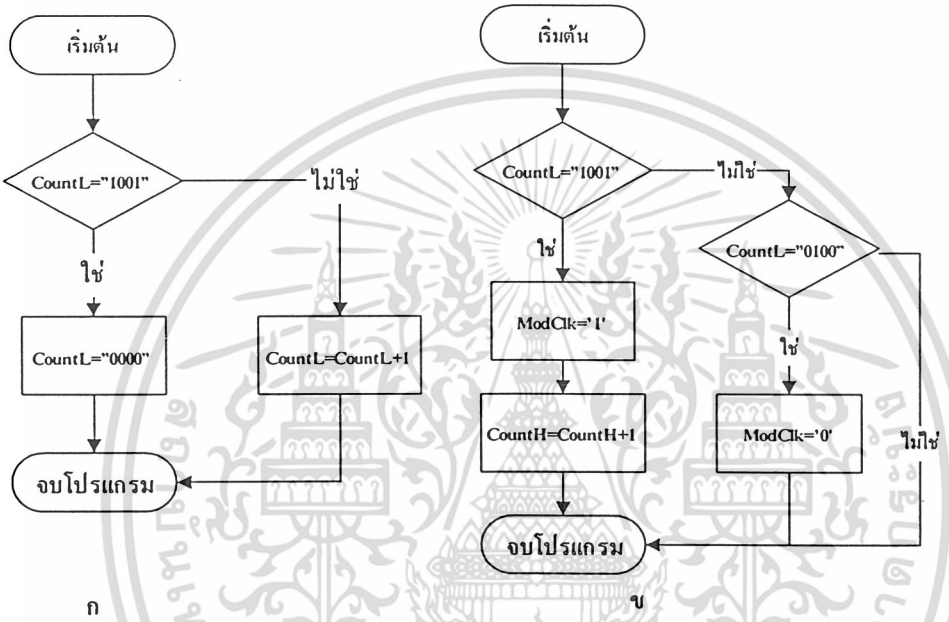
รูปที่ 4.14 ชุดวงจรบวก/ลบ

ชุดวงจรบวกลบก็ใช้หลักการง่ายในการออกแบบ จากรูปสามารถอธิบายได้ดังนี้ เมื่อข้อมูลที่เข้ามาเป็น 1ชุดวงจรบวกก็จะทำหน้าที่เป็นวงจรบวกธรรมดา อินพุทของเอ็กคลูซีฟออร์เป็นลอจิก 0 ดังนั้นเอาท์พุทไม่มีการเปลี่ยนแปลง และตัวทดมีค่าเป็น 0 แต่เมื่ออินพุทเป็น 0 จากคุณสมบัติของเอ็กคลูซีฟออร์เอาท์พุทของมันจะกลับสถานะกับอินพุท กระบวนการนี้เรียกว่า "1'Complement" จากนั้นตัวทดเท่ากับ 1 จะถูกบวกเพิ่มเข้ามาค่าที่ได้จะเป็น "2'Complement" จากนิยามที่ว่าค่า 2'Complement ของตัวเลขใดๆก็คือค่าลบของตัวเลขนั้นๆ

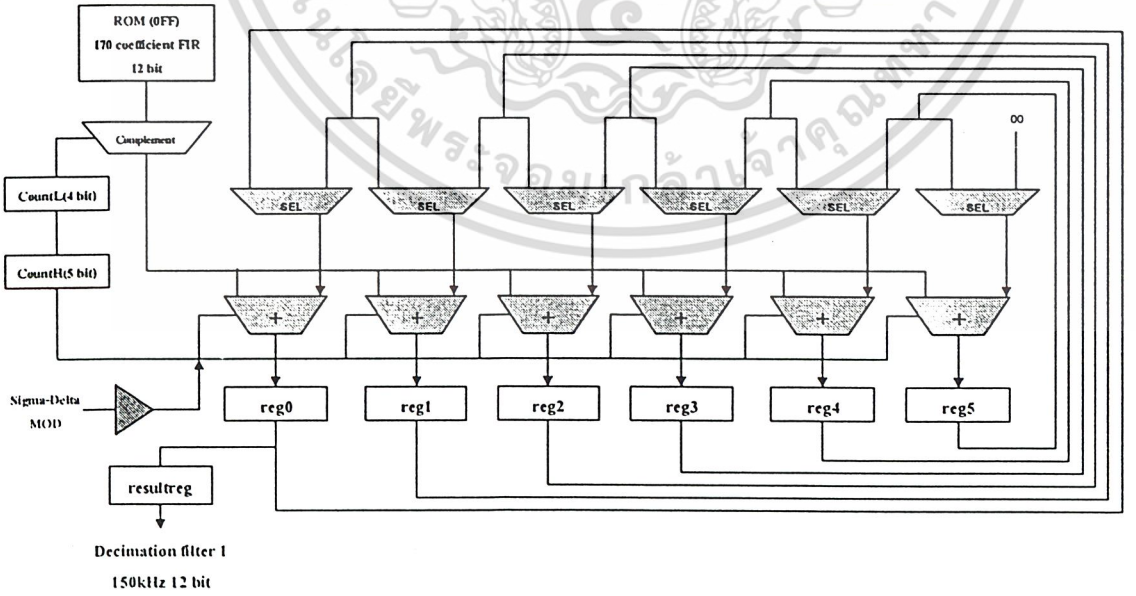
3. โครงสร้างในการเขียนโปรแกรม

ในการออกแบบการเขียนโปรแกรมอาศัย บล็อกไดอะแกรมแสดงทิศทางการไหลของข้อมูลจากรูปที่ 4.15 พบว่าส่วนประกอบหลักๆมีอยู่ด้วยกันดังนี้ คือ ส่วนของรอมที่เก็บสัมประสิทธิ์การประสาน ชุดของการคำนวณซึ่งก็คือ ชุดวงจรบวก และส่วนสุดท้าย ส่วนที่ทำหน้าที่กำหนดอัตราการไหลหรืออัตราการถ่ายโอนของข้อมูลและควบคุมกระบวนการทั้งหมดในส่วนนี้ประกอบรีจิสเตอร์สองตัว คือ

CountL และ CountH, CountL เป็นตัวนับ 10 หรือ วงจรหาร 10 เพื่อสร้างสัญญาณสุ่ม 4.8 MHz ให้กับซิกมา – เดลตามอดดูเลชั่น ดังนั้นรีจิสเตอร์ตัวนี้มีขนาด 4 – บิท ส่วน CountH เป็นรีจิสเตอร์ขนาด 5 บิท จะเพิ่มค่าขึ้นที่ละหนึ่ง ทุกครั้งที่ CountL นับได้ 9 CountH จะเป็นตัวกำหนดอัตราการไหลออกของข้อมูลทางเอาท์พุท เมื่อ CountH นับครบ 31 เอาท์พุทจะถูกเขียนออกมานั่นหมายความว่า CountH ทำหน้าที่หาร 32 ของความสุ่ม นอกจากนี้แล้ว CountL และ CountH ยังใช้เป็นตัวชี้ตำแหน่งของหน่วยความจำรอมเพื่อนำเข้าสู่

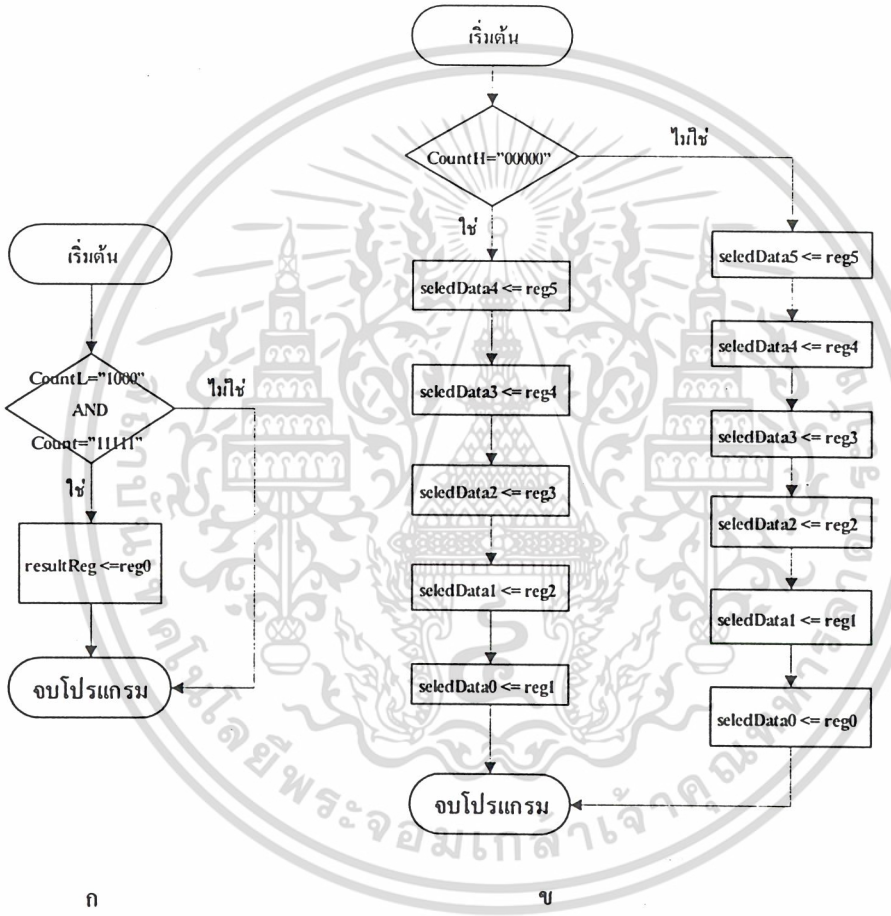


รูปที่ 4.14 โฟลวชาร์ตของ รีจิสเตอร์ CountLและ CountH



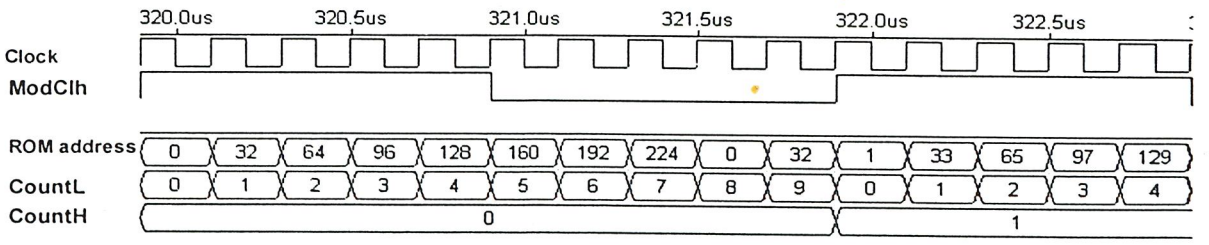
รูปที่ 4.15 บล็อกไดอแกรมแสดงทิศทางการไหลของข้อมูล

กระบวนการประสาน CountL และ CountH มีการเปลี่ยนแปลงดังรูปที่ 4.14โดยนำ CountL และ CountH มาต่อกัน ดังนี้ CountL (2 DOWNTO 0) & CountH (4 DOWNTO 0) จะได้ความสัมพันธ์กันของตำแหน่งหน่วยความจำรอมดังรูปที่ 4.16 สำหรับรีจิสเตอร์ reg0-reg5 จะเก็บผลการประสานทุกครั้งที่ CountH นับ 31 จะโอนย้ายข้อมูลไปสู่รีจิสเตอร์ทางด้านซ้าย และผลการประสานจะถูกอ่านออกที่ reg0 กระบวนการประสานที่เกิดขึ้นอธิบายได้ดังรูปที่ 4.16ข และ4.16ก แสดงส่วนที่ควบคุมผลทางเอาท์พุท ด้วยความถี่ 150 KHz ขนาด 12 – บิท เป็นอินพุทของชุดกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลขที่สอง



ก ข

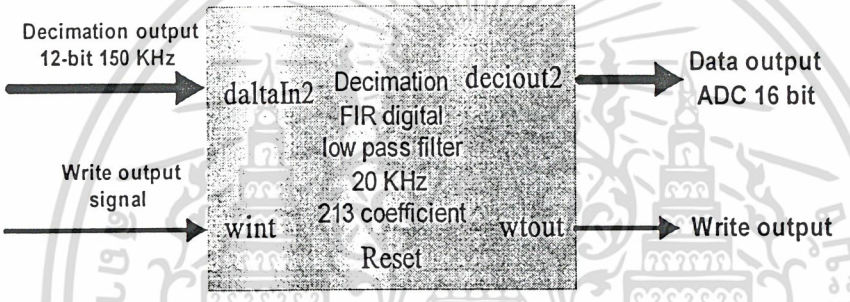
รูปที่ 4.16 ก.การส่งผลออกเอาท์พุท 4.16ข ขั้นตอนการประสาน



รูปที่ 4.17 ผลการจำลองการทำงานในส่วนควบคุม

4.2.2 วงจรกรองความถี่ต่ำภาคที่สอง

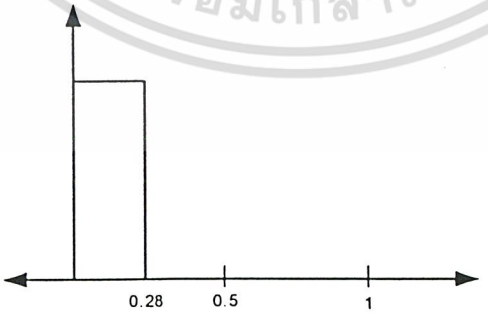
รูปที่ 4.17 แสดงการเชื่อมต่ออินพุทและเอาท์พุทของชุดกรองความถี่แบบไม่ป้อนกลับเชิงเลข อินพุทจะประกอบด้วยส่วนที่เป็นข้อมูลจากวงจรกรองชุดแรก ซึ่งมีขนาด 12 บิท อัตราการเปลี่ยนแปลงข้อมูลเท่ากับ 150 KHz และสัญญาณอีกเส้น เป็นตัวบอกว่ามีการเขียนข้อมูลออกมา เพื่อให้ชุดกรองที่สองเก็บค่าเข้ามาที่แรม (RAM: Random access memory) ส่วนเอาท์พุทมีสายสัญญาณสองเส้นเช่นเดียวกันคือ สายข้อมูลอะนาลอกที่เปลี่ยนเป็นดิจิตอลขนาด 16 บิท ส่วนออกมาเป็นแบบ I2S มีสาย 3 เส้นสำหรับติดต่อประกอบด้วย สายข้อมูลออกมาแบบอนุกรม สายเลือกช่องสัญญาณ (ซ้าย หรือ ขวา) และสายสัญญาณนาฬิกา เอาท์พุทมีอัตราการสุ่มเหลือ 75 KHz



รูปที่ 4.18 ชุดกรองความถี่ที่ 2

1. การหาผลตอบสนองอิมพัลส์

อินพุทเป็นดิจิตอล 12 บิทความถี่สุ่ม 150 KHz ในชุดกรองความถี่ชุดนี้ต้องการความถี่สุ่มทางเอาท์พุทลดลง 2 เท่า และช่วงความถี่ที่สนใจน้อยกว่า 20 KHz ดังนั้นแล้วเมื่อได้ข้อกำหนดที่ต้องการต่อไปขั้นต่อไปคือการผลตอบสนองอิมพัลส์ ซึ่งลักษณะการคำนวณจะเหมือนกับกับในภาคแรก ดังนี้

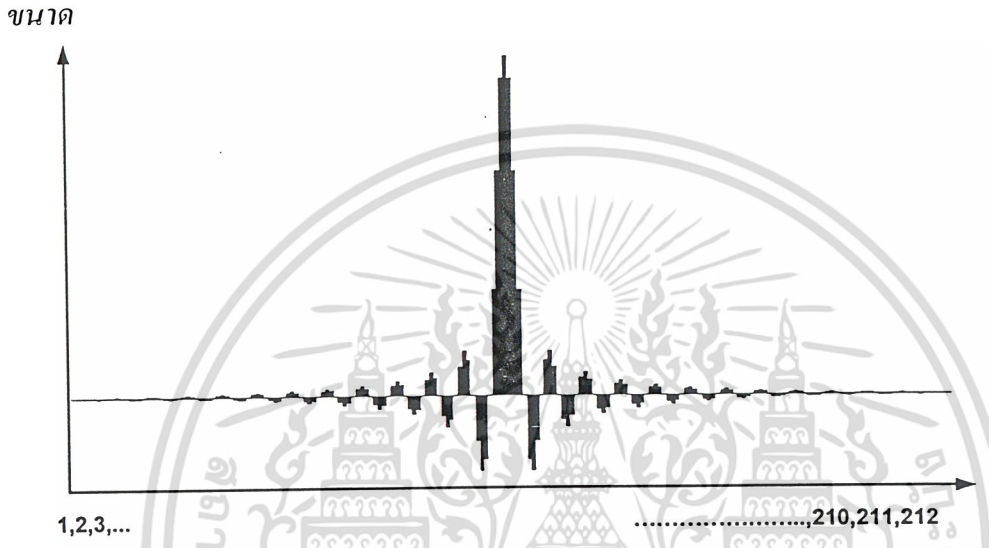


รูปที่ 4.19 ผลตอบสนองทางความถี่ในอุดมคติ

ต้องการความถี่ตัดที่ 21 KHz

$$\omega_{c} = \frac{2 * 21KHz}{150KHz} = 0.28$$

ใช้โปรแกรมMatlab หาค่าผลตอบสนองอิมพัลส์ 212 จำนวน ผลตอบสนองที่ได้จะใช้วินโดว์แบบแฮมมิ่ง ดังรูปที่ 4.18 แสดงกราฟผลตอบสนองอิมพัลส์แบบแฮมมิ่ง



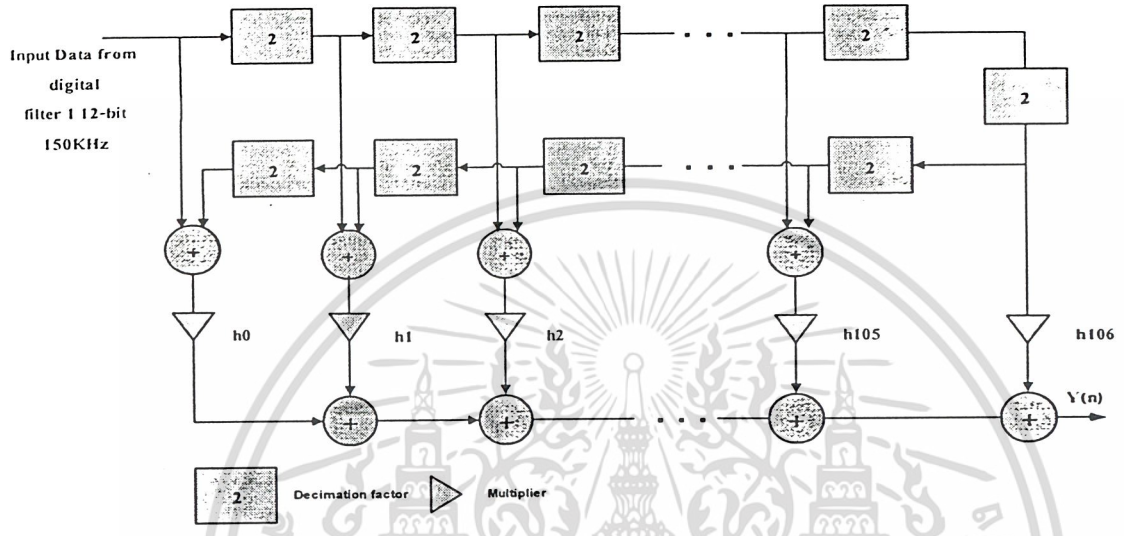
รูปที่ 4.20 ผลตอบสนองอิมพัลส์

ในขั้นตอนของการเปลี่ยนผลตอบสนองอิมพัลส์เป็นค่าทางดิจิตอลจะเหมือนกับวงจรชุดแรกแตกต่างกันตรงที่ ในวงจรชุดนี้เราเลือกใช้ค่าดิจิตอลขนาด 16 บิต เพื่อเพิ่มความเม่นยำให้กับข้อมูลและลดความผิดพลาดอีกด้วย เช่นเดียวกันการเก็บค่าลงในรอมและการสร้างก็จะกระทำในรูปแบบเดียวกัน เพียงแต่การกำหนดขนาดข้อมูลจะแตกต่างกันออกไป และที่สำคัญคือโครงสร้างที่เราใช้ในการออกแบบเป็นโครงสร้างแบบตรงที่อาศัยคุณสมบัติความสมมาตรคู่ในการออกแบบ ดังนั้นแล้วการเก็บผลตอบสนองอิมพัลส์ลงในรอม จึงสามารถเก็บเพียงครึ่งเดียวได้เนื่องจากค่าเท่ากัน ดังสมการด้านล่าง

$$H(z) = \sum_{n=0}^{105} h(n)\left[z^{-n} + z^{-(211-n)}\right] \quad \text{.....(4.1)}$$

การเก็บลงในรอมจึงเริ่มเก็บข้อมูลตั้งแต่ลำดับที่ 106 ไปจนถึง 212 ดังนั้นเราสามารถใช้ตำแหน่งของข้อมูลภายในรอมเพียง 128 ตำแหน่งในขั้นตอนการออกแบบรอมจึงกำหนดสายสัญญาณที่ทำหน้าที่ชี้ตำแหน่งข้อมูล 7 เส้นได้และข้อมูลเอาท์พุท 16 เส้น เมื่อสร้างรอมเสร็จสิ้นให้กำหนดไฟล์ที่เก็บข้อมูลรอมให้ด้วย

จากรูปที่ 4.20 แสดงโครงสร้างแบบตรงของชุดกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลขที่อาศัยคุณสมบัติสมมาตรคู่ องค์ประกอบโดยหลักๆ แล้วประกอบด้วย ส่วนที่ทำหน้าที่บวกเลข ส่วนวงจรคูณ และ ส่วนที่ทำการลดอัตราการสุ่มลง ดูจากโครงสร้างแล้วข้อมูลอินพุทที่รับเข้ามา จะต้องนำมา



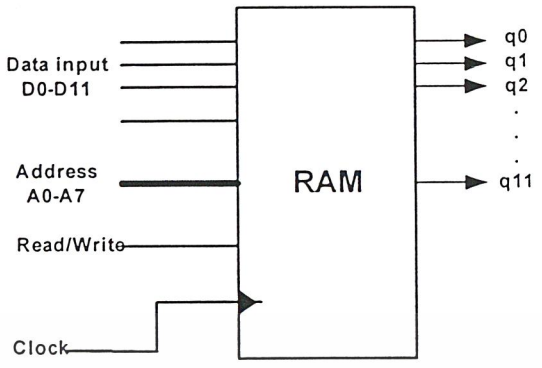
รูปที่ 4.21 โครงสร้างแบบตรงสมมาตรคู่

รวมกันก่อนที่จะคูณด้วยค่าสัมประสิทธิ์ ดังสมการที่ 4.1 ข้อมูลชุดปัจจุบันรวมกับข้อมูลก่อนหน้านี้สามารถอธิบายได้คือ

$$y(iT) = \sum_{n=0}^{105} h(n)\{x[(i-n)T] + x[(i+n-211)T]\} \quad \text{.....(4.2)}$$

จากสมการที่ 4.2 จะมีข้อมูลลำดับติลบนั่นหมายความว่าเราต้องมีการเก็บข้อมูลก่อน แล้วค่อยเรียกออกมารวมกัน ดังนั้นในชุดกรองความถี่ที่สองจึงมีองค์ประกอบที่แตกต่างไปจากชุดแรก คือมีส่วนของแรมเพิ่มขึ้นมา

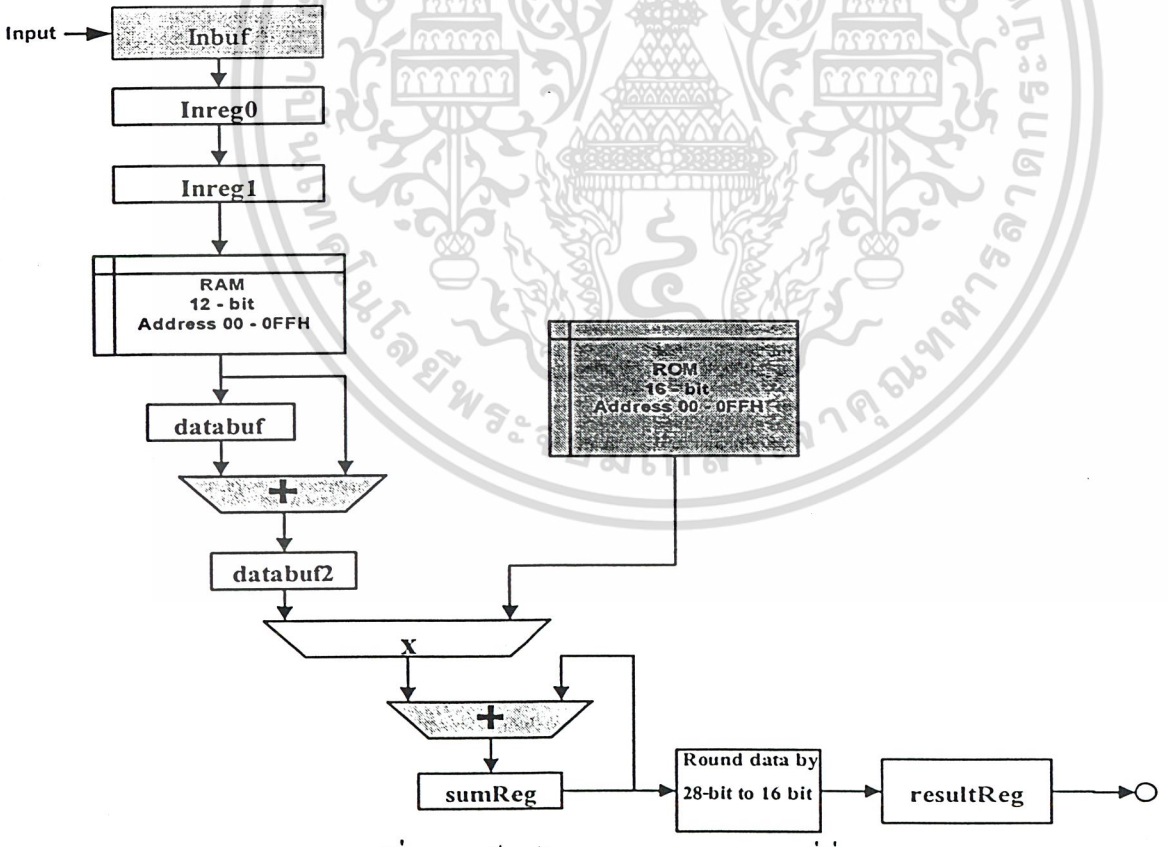
การสร้างแรมสามารถทำได้หลายแบบไม่ว่า จะเป็นการเขียนโปรแกรมเพื่อใช้เกตภายในสร้างแรม หรือการใช้โครงสร้างของแรมที่ตัว FPGA จัดเตรียมไว้ (มีเฉพาะบางเบอร์) ในโครงงานนี้เลือกใช้วิธีการหลังทั้งนี้ภายในตัวโปรมแกรมที่ใช้ออกแบบ มีเครื่องมือเฉพาะสำหรับการสร้างแรมที่ถูกออกแบบไว้เป็นมาตรฐานซึ่งเราสามารถเรียกใช้ได้เลย โดยโปรแกรมจะสร้างแรมขึ้นมาให้เพียงแต่เรากำหนดคุณสมบัติของแรม เช่นว่า จำนวนตำแหน่งข้อมูล ขนาดของอินพุท เพียงเท่านี้ก็สมารถนำไปใช้ในการออกแบบได้โดยอ้างอิงเข้าไปในรูปแบบของ คอมโพแนนส์ แรมที่ออกแบบมีโครงสร้างดังรูปที่ 4.21



รูปที่ 4.22 แรมที่ได้จากการออกแบบ

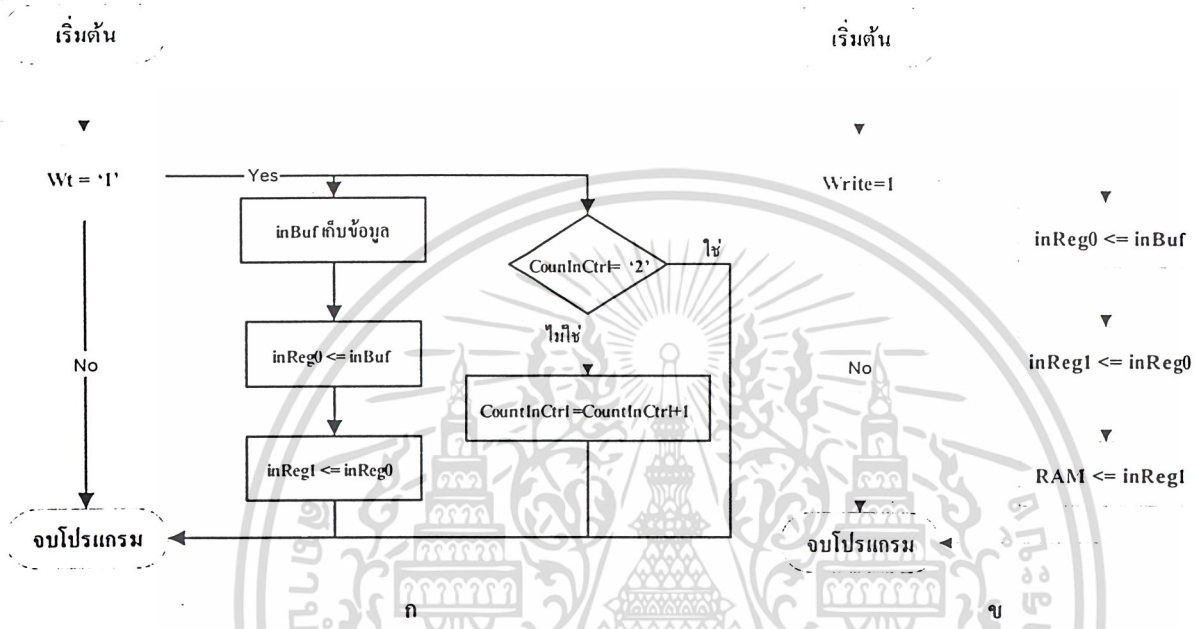
2. การเขียนโปรแกรม

วิธีการสำหรับออกแบบนั้นดูได้จากรูปที่ 4.23 ซึ่งแสดงโครงสร้างทั้งหมดและส่วนประกอบต่างๆในการออกแบบ เพื่อความเข้าใจยิ่งขึ้นจะขอกล่าวแยกเป็นส่วนเพื่อป้องกันความสับสน เนื่องมาจากลักษณะของภาษาที่ใช้ในการออกแบบเป็น ภาษาประเภท Concurrent ซึ่งมีการทำงานพร้อมกันทุกบรรทัด



รูปที่ 4.23 บล็อกไดอแกรมชุดกรองความถี่ที่ 2

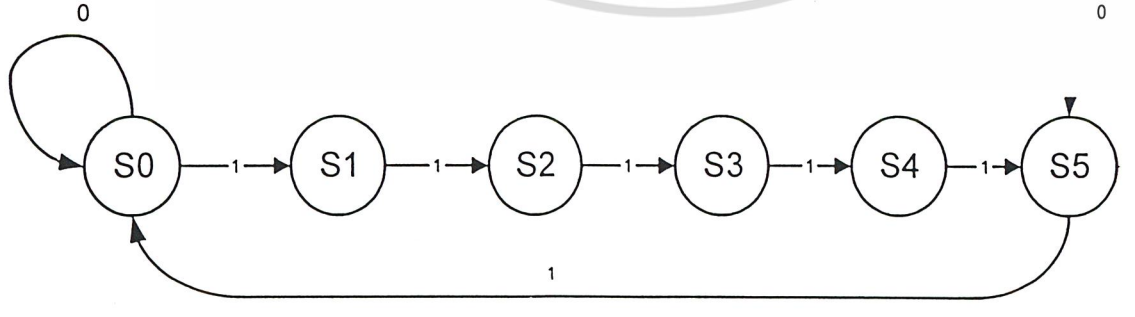
ส่วนแรกคือ ชุดที่เก็บข้อมูลเข้ามาในแรม เมื่อชุดกรองความถี่แรกส่งข้อมูลออกมาจะมีสัญญาณที่คอยบอกเสมอ ดังนั้นโปนแกรมในส่วนนี้ จะคอยตรวจสอบตลอดเวลา เมื่อได้รับสัญญาณ รีจิสเตอร์ InBuf จะเก็บข้อมูลเข้าไว้และจะถ่ายโอนข้อมูลไปสู่รีจิสเตอร์ inReg0 เมื่อได้รับสัญญาณอีกครั้ง



รูปที่ 4.23ก Flowchart การเก็บอินพุทเข้าบัฟเฟอร์ 4.23ข การเก็บลงแรม

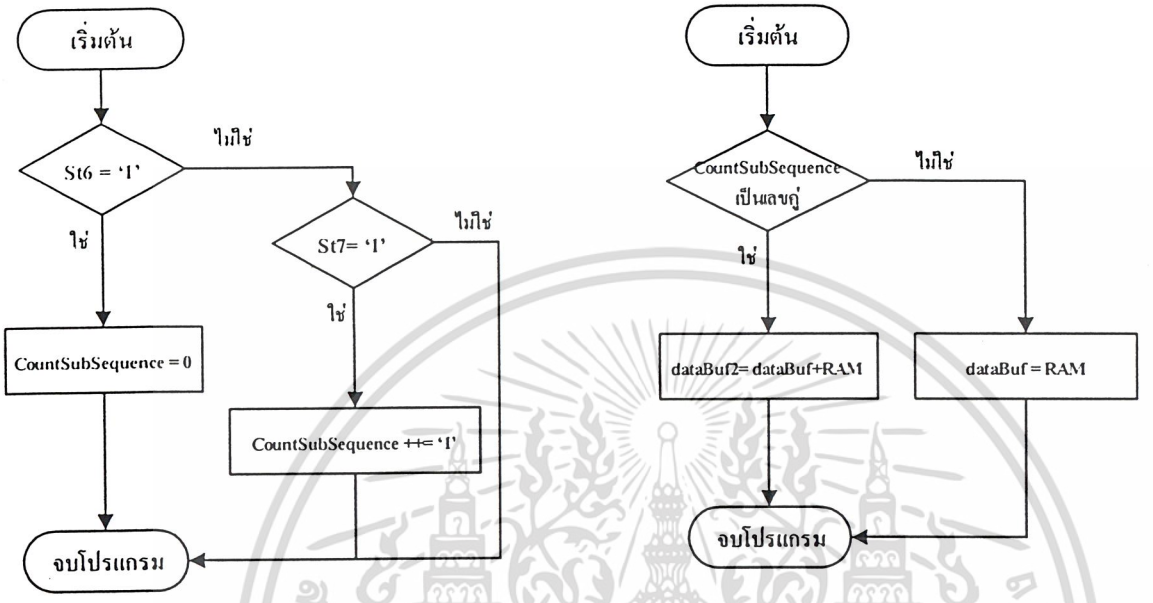
พร้อมกับรับข้อมูลตัวใหม่เข้ามา และ InReg0 ก็จะส่งข้อมูลไปที่ inReg1ดังรูปที่ 4.23ก เช่นเดียวกัน การเขียนลงแรมก็ใช้หลักการเช่นเดียวกัน ข้อมูลจะถูกเลื่อนเป็นทอดๆ ดูรูปที่ 4.23ข

ส่วนต่อไปเป็นสเตดไดอแกรม ที่ควบคุมกระบวนการทั้งหมดดังรูปที่ 4.24 การเปลี่ยนสถานะของแต่ละสเตดจะเกิดขึ้นจำนวนสัญญาณนาฬิกา 1ลูกเท่านั้น เช่น ถ้า สเตด 0 เป็นลอกจิก 1 ในสัญญาณนาฬิกาลูกปัจจุบันในสัญญาณนาฬิกาลูกถัดไปจะเป็น 0 การเปลี่ยนสถานะของแต่ละสเตด



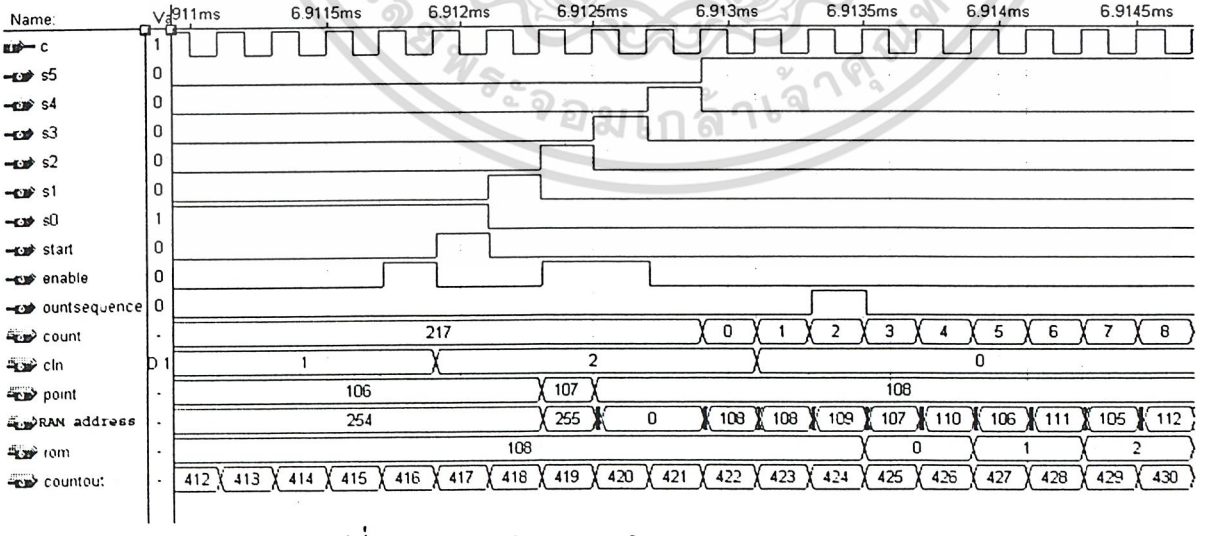
รูปที่ 4.24 State diagram ที่ควบคุมกระบวนการ

จะเป็นตัวกวบคุมรีจิสเตอร์ที่ทำหน้าที่นับทั้งหมดภายกระบวนการ และกำหนดจังหวะการเขียนข้อมูลลงที่แรม รีจิสเตอร์ที่ทำหน้านับที่สำคัญคือ CountSubsequence มีขนาด 8 บิตจะเป็นตัวที่ใช้อ้างอิงหรือชี้



รูปที่ 4.25ก รีจิสเตอร์ควบคุมตำแหน่งรอมและแรม **4.25ข** การรวมข้อมูลก่อนประสาน

ตำแหน่งข้อมูล ของรอม และดึงข้อมูลจากหน่วยความจำแรมออกมาเพื่อบวกกัน ดังรูปที่ 4.25 เมื่อ CountSubsequence นับเป็นเลขคี่ข้อมูลของแรมจะถูกดึงมาเก็บไว้ในรีจิสเตอร์ dataBuf ขณะที่เป็นเลขคู่ข้อมูลที่แอดเดรสชี้อยู่ขณะนั้นจะถูกึงออกมาและรวมกับค่าในรีจิสเตอร์ dataBuf ผลลัพธ์เก็บไว้ที่ dataBuf2 ขณะเดียวกันข้อมูลที่อยู่ใน dataBuf2 จะนำไปคูณกับค่าสัมประสิทธิ์ของผลตอบสนองอิมพัลส์ที่อยู่ในรอม



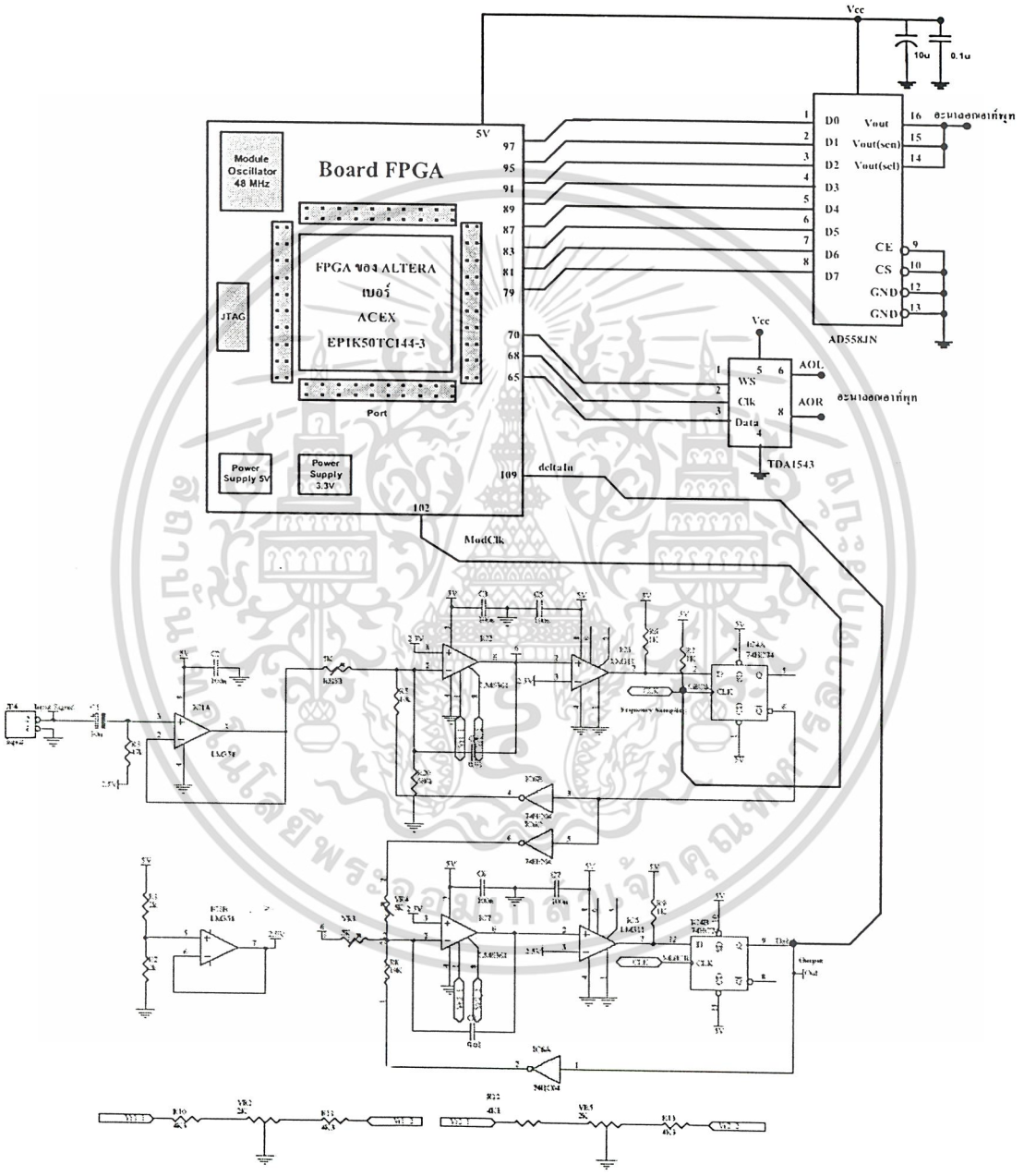
รูปที่ 4.26 ผลการจำลองการทำงาน

ณ ตำแหน่งที่แอดเดรสชี้ ผลคูณมีขนาด 29 บิต เพราะว่าข้อมูลของเรามีขนาด 12 บิต และผลตอบสนองอิมพัลส์ 16 บิต ดังนั้นผลคูณที่ได้จึงมีขนาด 28 บิต แทนขนาดและ 1 บิตแทนเครื่องหมาย ผลจากการคูณเป็นเลขดิจิตอล 29 บิตแต่ในการออกแบบเราต้องการอะนาลอกเป็นดิจิตอลขนาด 16 บิต ดังนั้นจึงต้องมีการปัดค่าเพื่อให้ได้ 16 บิต โดยรับค่าจาก บิตที่ 13 ถึง 29 กระบวนการต่อไปก็เป็นขั้นตอนของการส่งข้อมูลอนุกรม ประกอบด้วยสายสัญญาณ 3 เส้น คือ สายข้อมูลบิตที่มีนัยสำคัญสูงสุดจะถูกส่งออกมาก่อนโดยมีความสัมพันธ์กับสายสัญญาณนาฬิกา สัญญาณนาฬิกาจะมีความถี่เป็น32เท่าของความถี่สุ่มเอาท์พุทเพราะข้อมูลของเรามีขนาด 16 บิตทั้งสองแชนแนลแม้ไม่ใช้ช่องใดช่องหนึ่งก็ตาม ดังนั้นความถี่ของสัญญาณนาฬิกาคือ 2.4 MHz และสายเลือกช่องสัญญาณทำหน้าที่เลือกช่องสัญญาณเอาท์พุทว่าเป็นช่องซ้ายหรือขวา ในโครงงานนี้เลือกเพียงช่องใดช่องหนึ่งเท่านั้น ความถี่ที่ได้ของสายสัญญาณเส้นนี้เท่ากับ 75 KHz หรือเท่ากับความถี่สุ่ม

บทที่ 5

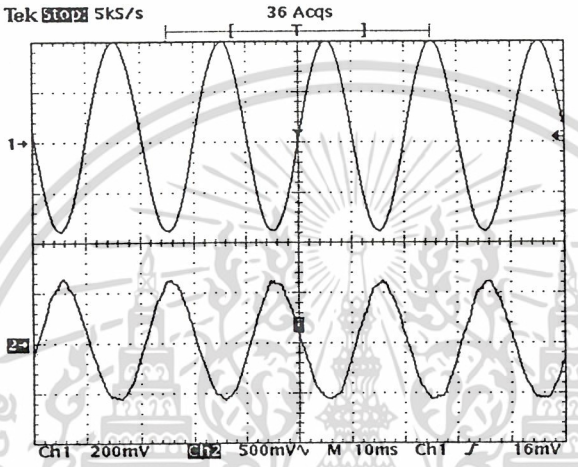
ผลการทดลอง และ สรุปผลการทดลอง

ในการทดลองให้ต่อวงจรดังรูปที่ 5.1 ซึ่งวงจรทั้งหมดที่ออกแบบ ทั้งส่วนที่เป็น

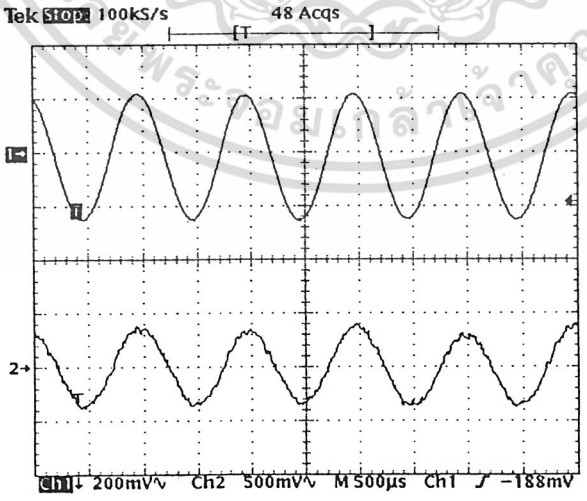


รูปที่ 5.1 วงจรที่ใช้ทดลอง

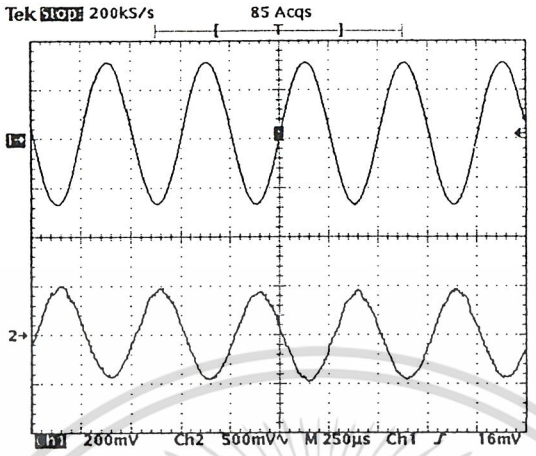
ซิมา — เดลตา มอดดูเลชั่น (อะนาลอก) และส่วนที่เป็นดิจิตอลฟิลเตอร์ที่ออกแบบบน FPGA การทดลองจะพิจารณาตลอดช่วงของความถี่ที่มนุษย์ได้ยิน คือจาก 50Hz – 20 KHz ทั้งนี้เพื่อพิจารณาว่าสามารถใช้กับความถี่ Audio ได้หรือไม่ ผลการทดลองเป็นดังนี้



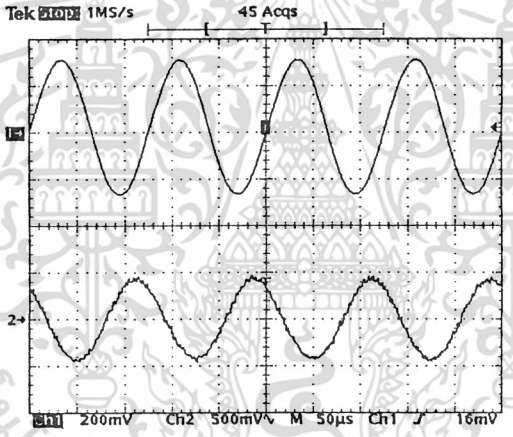
รูปที่ 5.2 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 50 Hz



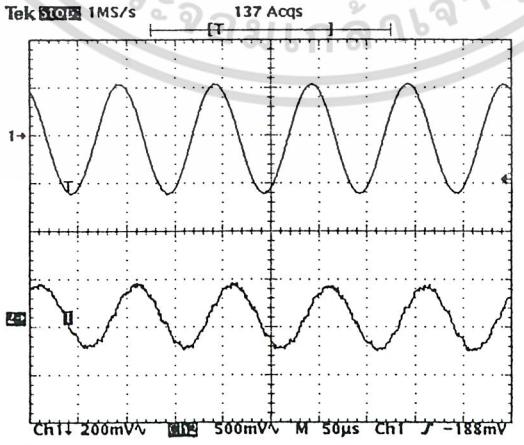
รูปที่ 5.3 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 1 KHz



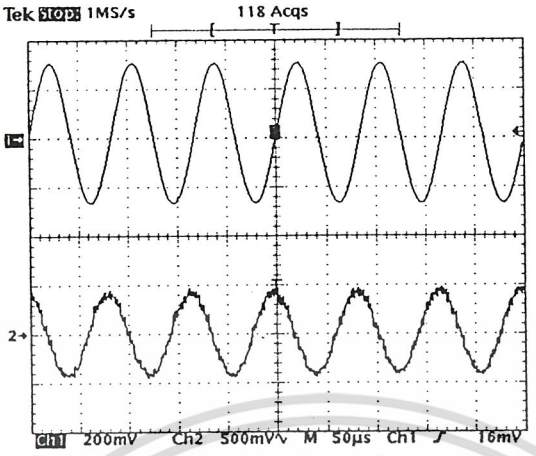
รูปที่ 5.4 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 2 KHz



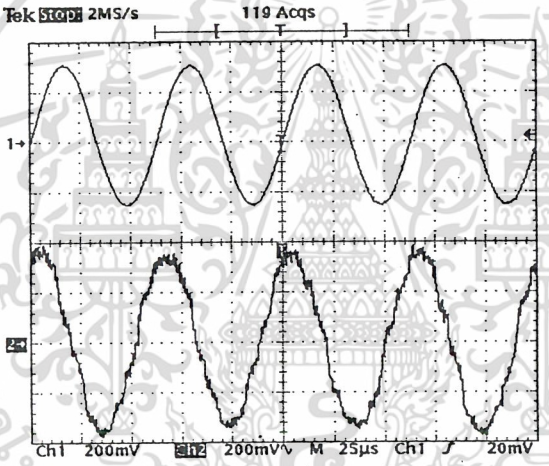
รูปที่ 5.5 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 8 KHz



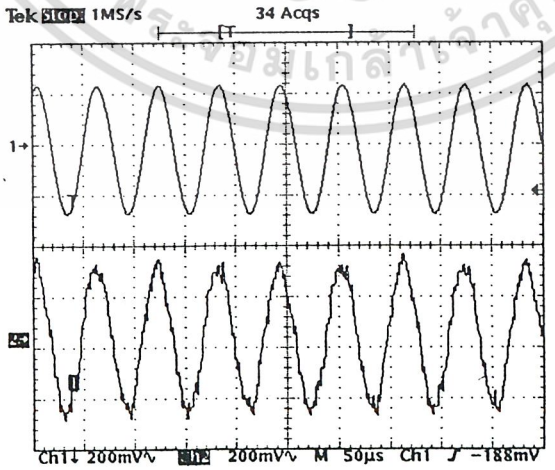
รูปที่ 5.6 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 10 KHz



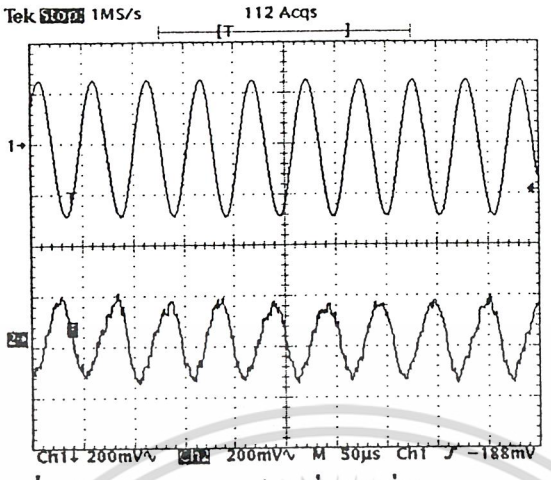
รูปที่ 5.7 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 12 KHz



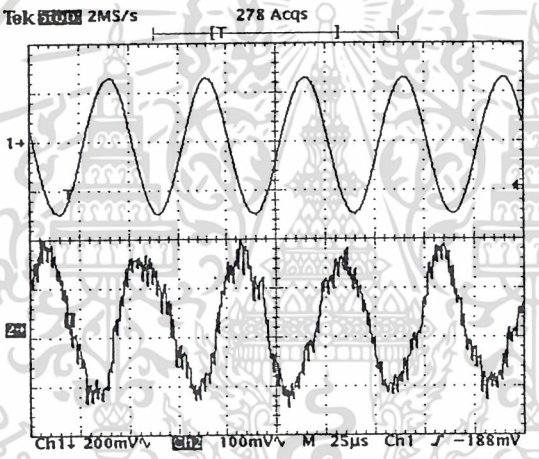
รูปที่ 5.8 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 16 KHz



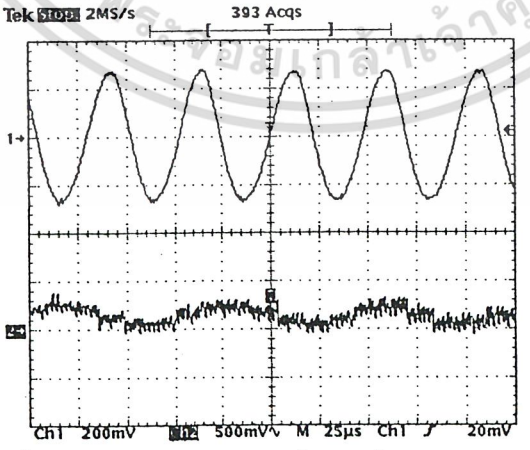
รูปที่ 5.9 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 17 KHz



รูปที่ 5.10 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 19 KHz



รูปที่ 5.11 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 20 KHz



รูปที่ 5.12 อินพุทต่อเอาท์พุทที่ความถี่อินพุท 21 KHz

สรุป

สรุปผลการทดลอง

คุณสมบัติที่สำคัญของอะนาลอกเป็นดิจิตอลแบบ ซิกมา – เดลตามอดดูเลชั่น คือการบวนการประมาณค่าแบบ 1 บิตและใช้อัตราการสุ่มที่สูงกว่าความถี่สนใจมาก 1 บิตที่ได้จะถูกป้อนเข้าสู่ชุดกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลขผ่านการลดอัตราการสุ่มและเพิ่มบิตทางดิจิตอลเอาท์พุท โดยทั่วไปอะนาลอกเป็นดิจิตอลแบบนี้จะมีค่า Resolution มากกว่า 12 บิต นิยมใช้กับย่านความถี่เสียงมีแบนด์วิทธ์ไม่สูงมาก

ในเบื้องต้นขอแยกพิจารณาสรุปผลในแต่ละส่วน ของอะนาลอกและดิจิตอลเนื่องจากว่าแนวทางวิธีการคิดค่อนข้างแตกต่างกัน ในส่วนของวงจรที่เป็นอะนาลอกนั้นในขั้นต้นได้พิจารณาการออกแบบ ซิกมา – มอดดูเลชั่นอันดับหนึ่งก่อน จากนั้นได้นำทฤษฎีของการออกแบบในอันดับที่สูงขึ้นมาพิจารณา และจำลองการทำงานบนโปรแกรม Matlab และเปรียบเทียบกับผลของมอดดูเลชั่นอันดับหนึ่งเมื่อเห็นว่าผลทางเอาท์พุทมีคุณภาพดีกว่า จึงเริ่มออกแบบในรูปของวงจรทางด้านอิเล็กทรอนิกส์ โดยอ้างอิงจากบล็อกแต่ละบล็อกของ Matlab เช่นว่ามีบล็อกของอินทิเกรเตอร์ ก็นำมาออกแบบวงจรอินทิเกรเตอร์ในเชิงของอิเล็กทรอนิกส์ ดังนั้นจึงเป็นไปได้ว่า

- ความบกพร่องของตัวอุปกรณ์ในทางโครงสร้างอาจจะทำให้ประสิทธิภาพที่ได้น้อยกว่าที่ได้จากการทดลองบนโปรแกรม
- ความคลาดเคลื่อนในการออกแบบวงจรในแต่ละบล็อก เนื่องจากว่าการทำงานในโปรแกรมอาจจะใกล้ความเป็นอุดมคติ แต่ว่าในวงจรจริง มิได้เป็นเช่นนั้น ดังเช่นว่า ค่าเวลาคงที่ของวงจรอินทิเกรเตอร์เป็นการยากที่จะระบุค่าที่แน่ชัดดังนั้นในการออกแบบจริงจึงใช้ความต้านทานปรับค่าได้เพื่อหาผลที่ดีที่สุด
- สัญญาณรบกวนจากภายนอก เนื่องจากว่าสัญญาณที่ใช้สุ่มมีความถี่สูงจึงมีการเหนี่ยวนำเข้ามาภายในวงจร

การพิจารณาชุดกรองความถี่ต่ำผ่านแบบไม่ป้อนกลับเชิงเลข ซึ่งถูกเลือกใช้ในโครงงานนี้เนื่องจากเราต้องการความเป็นเชิงเส้นทางเฟส โดยที่กระบวนการหลักของชุดกรองคือ การประสาน(Convolution) ชุดกรองแบ่งออกเป็นสองชุด ชุดแรกความถี่ตัดที่ 75KHz ค่าสัมประสิทธิ์การประสานเท่ากับ 171 ตัวความถี่สุ่มเอาท์พุต 150 KHz จำนวนของสัมประสิทธิ์ยิ่งมีค่ามากความชันของช่วง Transition จะสูงขึ้น ส่วนชุดกรองที่ 2 รับอินพุทแบบ 12 บิต ความถี่ตัดที่ 21 KHz จำนวนสัมประสิทธิ์การประสานเท่ากับ 212 ตัว ความถี่สุ่มเป็น 75 KHz

แนวทางการพัฒนาต่อ

จากการออกแบบที่ได้ในโครงงานนี้ผลที่ได้อาจจะยังไม่ดีที่สุดดังการพัฒนาให้ดีขึ้นควรจะมองในแง่ของ

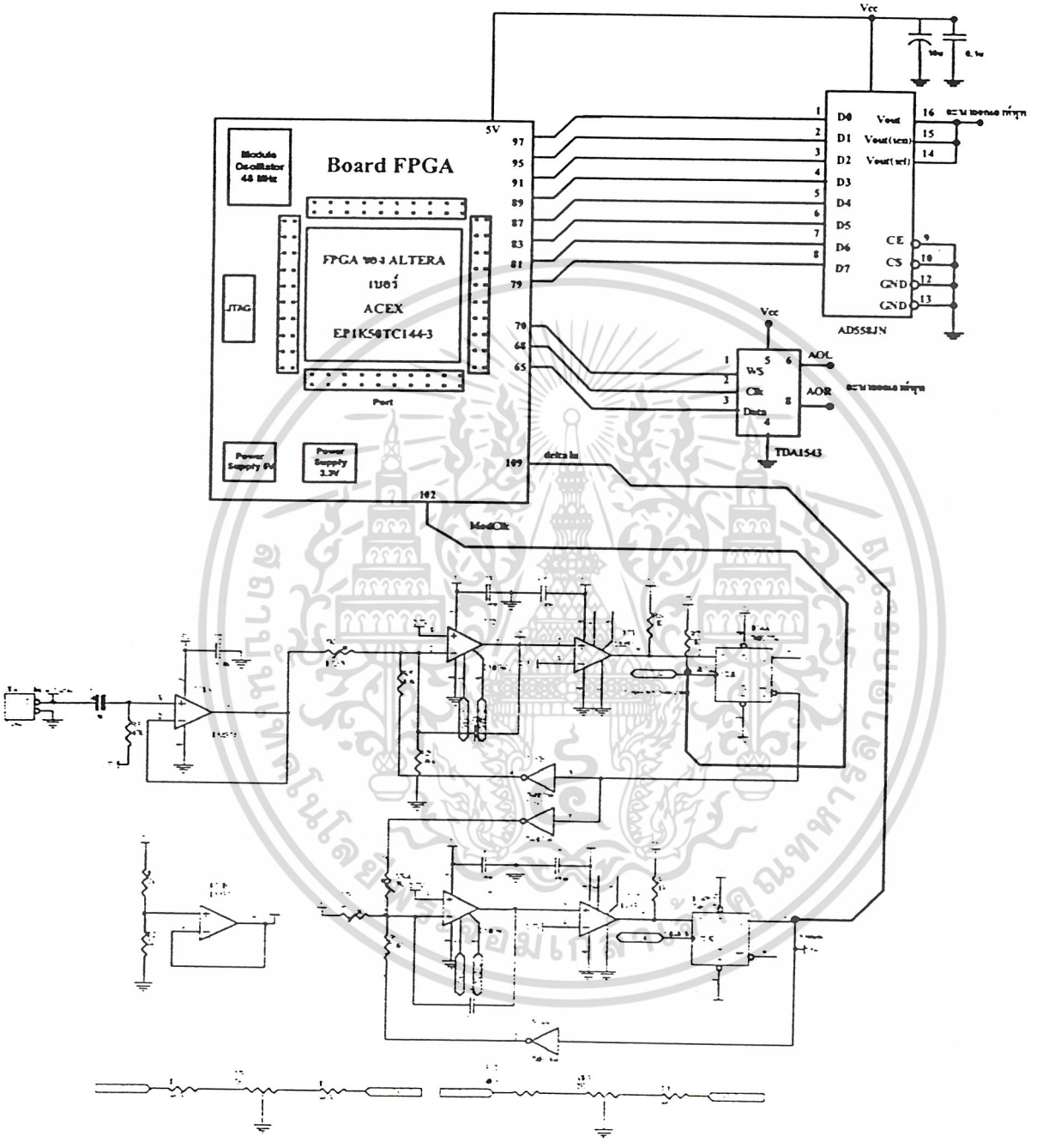
- การขยาย Bandwidths ของช่วงสัญญาณที่สนใจ
- การลดสัญญาณรบกวนที่ยังคงมีอยู่ให้น้อยลงอาจจะด้วยการ เพิ่มจำนวนของสัมประสิทธิ์การประสานให้สูงขึ้น หรือ การออกแบบในส่วนมอดดูเลชั่นให้มีอันดับที่สูงขึ้นจากเดิม
- เพิ่มค่าความถูกต้องของผลตอบสนองสัญญาณกระตุ้นในขั้นตอนการแปลงเป็นเลขดิจิตอลก่อนเก็บไว้ในรอม เช่นจากเดิมมีขนาด 12 บิต ก็เพิ่มเป็น 16 บิต ผลที่ได้จากการประสานจะมีความแม่นยำมากขึ้นตามไปด้วย
- เพิ่มอัตราการสุ่มให้มากขึ้นแต่ทั้งนี้ ต้องมีการออกแบบวงจรให้ดีขึ้นเพราะว่าความถี่สูงจะเหนียวนำเข้าสู่วงได้

จากหลายข้อของการพัฒนานาต่อให้ประสิทธิภาพดีขึ้นแล้ว ชุดกรองความถี่แบบไม่ป้อนกลับเชิงเลขยังสามารถใช้เป็นพื้นฐานการออกแบบดิจิตอลฟิลเตอร์ในแบบอื่นๆ ได้และกำหนดความถี่ตัดให้กับชุดกรองใหม่

บรรณานุกรม

1. วัลลภ สุระกำพลธร, "การประมวลผลสัญญาณเชิงเลข ", คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 348 หน้า, 2533
2. ชำนาญ ปัญญาใส, วัชรกร หนูทอง, "ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล", ซีเอ็ดยูเคชั่น, 432 หน้า, 2547
3. David J. De Falta, Joseph G. Lucas, William S. Hodgkiss, "DIGITAL SIGNAL PROCESSING", JONH WILEY & SONS, 661 p., 1988.
4. Adel S. Sedra, Kenneth C. Smith, "Microelectronic circuits", Oxford University Press, 1054 p., 1991
5. Sangil Park, "Motorola Digital Signal Processors : Principle of Sigma-Delta Modulation for Analog-to-Digital Converters", Available_Source : http://www.numerix-dsp.com/appsnotes/APR8-sigma-delta.pdf.

ภาคผนวก



รูปวงจรสมบูรณ์ของ ซิกมา-เดลตา มอดดูเลชั่นอันดับสอง

Source Code Program

```
--**************************************************************************
-- deciFil1
-- decimation filter 1 for delta-sigma AD converter
--**************************************************************************
-- add (with carry input) library
--**************************************************************************
library ieee;
use ieee.std_logic_1164.all;
package libAdc_deciFil1 is
function adc18Nco(aa,bb : std_logic_vector(17 downto 0);cc : std_logic) return
std_logic_vector;
end libAdc_deciFil1;
package body libAdc_deciFil1 is
function adc18Nco(aa,bb : std_logic_vector(17 downto 0);cc : std_logic) return
std_logic_vector is
        variable interCarry : std_logic;
        variable temp : std_logic_vector(aa'high downto aa'low);
begin
interCarry := cc;
        for j in aa'low to aa'high loop
        temp(j) := aa(j) xor bb(j) xor interCarry;
        interCarry := (aa(j) and bb(j)) or ( (aa(j) or bb(j)) and interCarry);
                                end loop;
return(temp);
end adc18Nco;
end libAdc_deciFil1;
----------------------------------------------------------
-- decimation filter 1
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_unsigned.all;
use work.libAdc_deciFil1.all;
entity deciFil1 is
port( deltaIn : in std_logic; -- delta-sigma modulate signal input
          c : in std_logic; -- system clock
         nr : in std_logic; -- system reset
     modClk : out std_logic; -- clock output for delta-sigma modulator
     deciOut : out std_logic_vector(11 downto 0); -- decimation data output
       wtOut : out std_logic); -- output data write signal
end deciFil1;
architecture rtl of deciFil1 is
component rom12x256f1
port( address : in std_logic_vector(7 downto 0);
          q : out std_logic_vector(11 downto 0) );
end component;
signal countH : std_logic_vector(4 downto 0); -- control counter
signal countL : std_logic_vector(3 downto 0);
signal countHZero : std_logic; -- decode signal of control counter
signal countH31 : std_logic;
signal countL8 : std_logic;
signal loadResult : std_logic; -- result Reg's data load signal
signal sampledData : std_logic; -- sampled delta-sigma modulate signal
signal invSampledData : std_logic; -- inverted "sampledData"
signal romAddress : std_logic_vector(7 downto 0); -- address for constant ROM
signal romData : std_logic_vector(11 downto 0); -- constant ROM data
signal expandedRomData : std_logic_vector(17 downto 0); -- expanded(12 -> 18bit) ROM data
signal compedRomData : std_logic_vector(17 downto 0); -- complemented/non-complemented
ROM data
signal zero18 : std_logic_vector(17 downto 0); -- 18bit zero data
signal seledData5 : std_logic_vector(17 downto 0); -- sum's source data
signal seledData4 : std_logic_vector(17 downto 0);
signal seledData3 : std_logic_vector(17 downto 0);
```

```
signal seledData2 : std_logic_vector(17 downto 0);
signal seledData1 : std_logic_vector(17 downto 0);
signal seledData0 : std_logic_vector(17 downto 0);
signal reg5In : std_logic_vector(17 downto 0); -- sum Reg's source data
signal reg4In : std_logic_vector(17 downto 0);
signal reg3In : std_logic_vector(17 downto 0);
signal reg2In : std_logic_vector(17 downto 0);
signal reg1In : std_logic_vector(17 downto 0);
signal reg0In : std_logic_vector(17 downto 0);
signal reg5 : std_logic_vector(17 downto 0); -- sum Reg.
signal reg4 : std_logic_vector(17 downto 0);
signal reg3 : std_logic_vector(17 downto 0);
signal reg2 : std_logic_vector(17 downto 0);
signal reg1 : std_logic_vector(17 downto 0);
signal reg0 : std_logic_vector(17 downto 0);
signal resultReg : std_logic_vector(17 downto 0); -- filter calculation's result Reg.
signal wtFlag : std_logic_vector(1 downto 0); -- delay for generate "wtOut" signal
signal roundedData : std_logic_vector(12 downto 0); -- rounded data of result Reg.
signal plusOver : std_logic; -- roundedData's puls overflow
signal minusOver : std_logic; -- roundedData's minus overflow
begin
-- ### register discription ###
-- ##########################################################
process(nr,c)
begin
if (nr = '0') then
        countL <= (others => '0');
        countH <= (others => '0');
    sampledData <= '0';
        reg5 <= (others => '0');
        reg4 <= (others => '0');
        reg3 <= (others => '0');
```

```
signal seledData2 : std_logic_vector(17 downto 0);
signal seledData1 : std_logic_vector(17 downto 0);
signal seledData0 : std_logic_vector(17 downto 0);
signal reg5In : std_logic_vector(17 downto 0); -- sum Reg's source data
signal reg4In : std_logic_vector(17 downto 0);
signal reg3In : std_logic_vector(17 downto 0);
signal reg2In : std_logic_vector(17 downto 0);
signal reg1In : std_logic_vector(17 downto 0);
signal reg0In : std_logic_vector(17 downto 0);
signal reg5 : std_logic_vector(17 downto 0); -- sum Reg.
signal reg4 : std_logic_vector(17 downto 0);
signal reg3 : std_logic_vector(17 downto 0);
signal reg2 : std_logic_vector(17 downto 0);
signal reg1 : std_logic_vector(17 downto 0);
signal reg0 : std_logic_vector(17 downto 0);
signal resultReg : std_logic_vector(17 downto 0); -- filter calculation's result Reg.
signal wtFlag : std_logic_vector(1 downto 0); -- delay for generate "wtOut" signal
signal roundedData : std_logic_vector(12 downto 0); -- rounded data of result Reg.
signal plusOver : std_logic; -- roundedData's puls overflow
signal minusOver : std_logic; -- roundedData's minus overflow
begin
-- ### register discription ###
-- ##########################################################
process(nr,c)
begin
if (nr = '0') then
        countL <= (others => '0');
        countH <= (others => '0');
    sampledData <= '0';
            reg5 <= (others => '0');
            reg4 <= (others => '0');
            reg3 <= (others => '0');
```

```
            reg2 <= (others => '0');
            reg1 <= (others => '0');
            reg0 <= (others => '0');
      resultReg <= (others => '0');
          wtFlag <= (others => '0');
         modClk <= '0';
         elsif (c'event and c = '1') then
-------------------------------------------------------
-- control counter
         if (countL = "1001") then
         countL <= "0000";
                              else
         countL <= countL + '1';
         end if;
if (countL = "1001") then
countH <= countH + '1';
end if;
-- clock for delta-sigma modulator
         if (countL = "1001") then
         modClk <= '1';
         elsif (countL = "0100") then
         modClk <= '0';
         end if;
-- input data sampling
if (countL = "1000") then
sampledData <= deltaIn;
end if;
-- calc. register
         if (countL = "0101") then
         reg5 <= reg5In;
         end if;
if (countL = "0100") then
```

```
reg4 <= reg4In;
end if;
if (countL = "0011") then
reg3 <= reg3In;
end if;
        if (countL = "0010") then
        reg2 <= reg2In;
        end if;
if (countL = "0001") then
reg1 <= reg1In;
end if;
        if (countL = "0000") then
                reg0 <= reg0In;
        end if;
-- result register
if (loadResult = '1') then
resultReg <= reg0;
end if;
-- generate output data write signal
wtFlag <= wtFlag(0) & loadResult;
-------------------------------------------------------
end if;
end process;
-- ### logic discription ###
-- ##################################################################
-- counter decord
countH31 <= '1' when (countH = "11111") else '0';
countL8 <= '1' when (countL = "1000") else '0';
loadResult <= countH31 and countL8;
countHZero <= '1' when (countH = "00000") else '0';
-- constant rom
romAddress <= countL(2 downto 0) & countH;
```

```
rom : rom12x256f1 port map(
address => romAddress,
q => romData);
-- calculation
zero18 <= "000000000000000000";
seledData5 <= zero18 when (countHZero = '1') else reg5;
seledData4 <= reg5 when (countHZero = '1') else reg4;
seledData3 <= reg4 when (countHZero = '1') else reg3;
seledData2 <= reg3 when (countHZero = '1') else reg2;
seledData1 <= reg2 when (countHZero = '1') else reg1;
seledData0 <= reg1 when (countHZero = '1') else reg0;
expandedRomData <= romData(11) & romData(11) & romData(11) & romData(11) &
romData(11) &
romData(11) & romData;
compedRomData <= expandedRomData when (sampledData = '1') else (not expandedRomData);
invSampledData <= not sampledData;
reg5In <= adc18Nco(seledData5,compedRomData,invSampledData);
reg4In <= adc18Nco(seledData4,compedRomData,invSampledData);
reg3In <= adc18Nco(seledData3,compedRomData,invSampledData);
reg2In <= adc18Nco(seledData2,compedRomData,invSampledData);
reg1In <= adc18Nco(seledData1,compedRomData,invSampledData);
reg0In <= adc18Nco(seledData0,compedRomData,invSampledData);
-- round output data
roundedData <= (resultReg(17 downto 5) + '1') when (resultReg(4) = '1') else resultReg(17
downto
5);
plusOver <= '1' when (roundedData(12 downto 11) = "01") else '0';
minusOver <= '1' when (roundedData(12 downto 11) = "10") else '0';
deciOut <= "011111111111" when (plusOver = '1') else
"100000000000" when (minusOver = '1') else
roundedData(11 downto 0);
-- output assign
```

```
wtOut <= wtFlag(1);
end rtl;
--**************************************************************************
-- deciFil2
-- decimation filter 2 for delta-sigma AD converter
--**************************************************************************
----------------------------------------------------------
-- add (with carry input) library
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
package libAdc_deciFil2 is
function adc8Nco(aa,bb : std_logic_vector(7 downto 0);cc : std_logic) return std_logic_vector;
end libAdc_deciFil2;
package body libAdc_deciFil2 is
function adc8Nco(aa,bb : std_logic_vector(7 downto 0);cc : std_logic) return std_logic_vector
is
        variable interCarry : std_logic;
        variable temp : std_logic_vector(aa'high downto aa'low);
begin
interCarry := cc;
        for j in aa'low to aa'high loop
        temp(j) := aa(j) xor bb(j) xor interCarry;
        interCarry := (aa(j) and bb(j)) or ( (aa(j) or bb(j)) and interCarry);
end loop;
return(temp);
end adc8Nco;
end libAdc_deciFil2;
----------------------------------------------------------
-- 15-bit x 13-bit (signed data) multiplier
----------------------------------------------------------
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity mulS15x13 is
port( x : in std_logic_vector(14 downto 0);
      y : in std_logic_vector(12 downto 0);
      z : out std_logic_vector(27 downto 0) );
end mulS15x13;
architecture rtl of mulS15x13 is
signal m1 : std_logic_vector(26 downto 14);
signal m2 : std_logic_vector(26 downto 12);
signal sumPlus : std_logic_vector(27 downto 0);
signal sumMinus : std_logic_vector(26 downto 12);
begin
sumPlus(27) <= '0';
sumPlus(26) <= x(14) and y(12);
sumPlus(25 downto 0) <= x(13 downto 0) * y(11 downto 0);
m1 <= ('0' & y(11 downto 0) ) when (x(14) = '1') else "0000000000000";
m2 <= ('0' & x(13 downto 0) ) when (y(12) = '1') else "000000000000000";
sumMinus(26 downto 14) <= m1 + m2(26 downto 14);
sumMinus(13 downto 12) <= m2(13 downto 12);
z(27 downto 12) <= sumPlus(27 downto 12) - sumMinus;
z(11 downto 0) <= sumPlus(11 downto 0);
end rtl;
------------------------------------------------------
-- decimation filter 2
------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.libAdc_deciFil2.all;
entity deciFil2 is
port( deciIn : in std_logic_vector(11 downto 0); -- data input
```

```
        wt : in std_logic; -- data write signal
         c : in std_logic; -- system clock
        nr : in std_logic; -- system reset
   deciOut : out std_logic_vector(15 downto 0); -- decimation data output
     wtOut : out std_logic); -- data output write signal
end deciFil2;
architecture rtl of deciFil2 is
component ram12x256f
port( address : in std_logic_vector(7 downto 0);
      inclock : in std_logic;
           we : in std_logic := '1';
data : in std_logic_vector(11 downto 0);
q : out std_logic_vector(11 downto 0) );
end component;
component rom15x128f2
port( address : in std_logic_vector (6 downto 0);
            q : out std_logic_vector (14 downto 0) );
end component;
component mulS15x13
port( x : in std_logic_vector(14 downto 0);
      y : in std_logic_vector(12 downto 0);
      z : out std_logic_vector(27 downto 0) );
end component;
signal inBuf : std_logic_vector(11 downto 0); -- data input buffer
signal flagInBuf : std_logic; -- data input flag
signal enableShift : std_logic; -- data shift-in signal
signal inReg0 : std_logic_vector(11 downto 0); -- data input Reg.(4-stage)
signal inReg1 : std_logic_vector(11 downto 0);
signal countInCtrl : std_logic_vector(1 downto 0); -- input control counter
std_logic_vector(2 downto 0)
signal st0 : std_logic; -- one-hot sequencer's state
signal st1 : std_logic;
```

```
signal st2 : std_logic;
signal st3 : std_logic;
signal st4 : std_logic;
signal st5 : std_logic;
signal nonState : std_logic; -- all state zero signal
signal startSq : std_logic; -- sequencer start signal
signal incPointer : std_logic; -- pointer increment signal
signal writeRam : std_logic; -- synchronouse write signal for RAM
signal countSubSequence : std_logic_vector(7 downto 0); -- sub-sequence counter
signal countSubSequence02H : std_logic; -- decode signal of sub-sequence counter
signal endSubSequence : std_logic; -- decode signal of sub-sequence counter
signal delayedEndSubSequence : std_logic; -- decode signal of sub-sequence counter
(1-cycle delayed)
signal pointer : std_logic_vector(7 downto 0); -- pointer for center data of RAM
signal offsetNum : std_logic_vector(6 downto 0); -- offset number from center data
signal doComp : std_logic; -- calculate minus value of "offsetNum" signal
signal compedOffset : std_logic_vector(7 downto 0); -- complemented/non-complemented data of
offsetNum
signal ramAddress : std_logic_vector(7 downto 0); -- RAM address
signal ramData : std_logic_vector(11 downto 0); -- RAM's read data
signal dataBuf : std_logic_vector(11 downto 0); -- RAM's read data buffer
signal dataBuf2 : std_logic_vector(12 downto 0); -- buffer of symmetrical data's sum
signal romAddress : std_logic_vector(6 downto 0); -- address for constant ROM
signal romAddress2 : std_logic_vector(6 downto 0); -- address for constant ROM (1-cycle
delayed)
signal romAddress3 : std_logic_vector(6 downto 0); -- address for constant ROM (2-cycle
delayed)
signal romData : std_logic_vector(14 downto 0); -- constant ROM data
signal product : std_logic_vector(27 downto 0); -- multiply result
signal sumReg : std_logic_vector(28 downto 0); -- sum Reg.
signal roundedSum : std_logic_vector(16 downto 0); -- rounded data of sum Reg.
signal overSum : std_logic; -- rounded sum Reg's overflow
```

```
signal underSum : std_logic; -- rounded sum Reg's underflow
signal resultReg : std_logic_vector(15 downto 0); -- result Reg.
signal resultIn : std_logic_vector(15 downto 0); -- result Reg's source data
begin
-- ### register discription ###
-- ######################################################################
process(nr,c)
begin
        if (nr = '0') then
                inBuf <= (others => '0');
                flagInBuf <= '0';
                countInCtrl <= (others => '0');
                inReg0 <= (others => '0');
                inReg1 <= (others => '0');
                st0 <= '1';
                st1 <= '0';
                st2 <= '0';
                st3 <= '0';
                st4 <= '0';
                st5 <= '0';
                countSubSequence <= (others => '0');
                pointer <= (others => '0');
                dataBuf <= (others => '0');
                dataBuf2 <= (others => '0');
                romAddress <= (others => '0');
                romAddress2 <= (others => '0');
                romAddress3 <= (others => '0');
                sumReg <= (others => '0');
                delayedEndSubSequence <= '0';
                wtOut <= '0';
                resultReg <= (others => '0');
        elsif (c'event and c = '1') then
```

```
-------------------------------------------------------------

-- input data buffer & flag
        if (wt = '1') then
                inBuf <= deciIn;
        end if;
                flagInBuf <= wt;
-- input data control counter
if (st3 = '1') then --st5
countInCtrl <= (others => '0');
elsif (flagInBuf = '1') then
        if (countInCtrl(1) = '0') then
                countInCtrl <= countInCtrl + '1';
end if;
        end if;
-- input data reg.
if (enableShift = '1') then
        inReg0 <= inBuf;
        inReg1 <= inReg0;
--inReg2 <= inReg1;
--inReg3 <= inReg2;
end if;
-- control sequencer
        if (nonState = '1') then
                st0 <= '1';
        elsif ( (st0 and countInCtrl(1) ) = '1') then
                st0 <= '0';
        end if;
  st1 <= st0 and countInCtrl(1);
  st2 <= st1;
  st3 <= st2;
  st4 <= st3;
                if (st4 = '1') then --st6
```

```
                    st5 <= '1'; --st7
                elsif ( (st5 and endSubSequence) = '1') then --st7
                    st5 <= '0'; --st7
                end if;
-- sub sequence counter
        if (st4 = '1') then --st6
                countSubSequence <= (others => '0');
        elsif (st5 = '1') then --st7
                countSubSequence <= countSubSequence + '1';
        end if;
-- ram pointer
if (incPointer = '1') then
pointer <= pointer + '1';
end if;
-- sum of symmetrical data
        if (countSubSequence(0) = '1') then
        dataBuf <= ramData;
        end if;
if (countSubSequence(0) = '0') then
dataBuf2 <= (dataBuf(11) & dataBuf) + (ramData(11) & ramData);
end if;
-- delay rom address
romAddress <= countSubSequence(7 downto 1);
romAddress2 <= romAddress;
romAddress3 <= romAddress2;
-- sum
                if (countSubSequence02H = '1') then
                    sumReg <= (others => '0');
                elsif (countSubSequence(0) = '0') then
                    sumReg <= sumReg + (product(27) & product);
                end if;
-- data output timing
```

```
delayedEndSubSequence <= endSubSequence;
wtOut <= delayedEndSubSequence;
                    if (delayedEndSubSequence = '1') then
                    resultReg <= resultIn;
                    end if;
-------------------------------------------------------

        end if;
end process;
-- ### logic discription ###
-- ###################################################################
-- sequencer control
nonState <= not ( st5 or st4 or st3 or st2 or st1 or st0); --st7 or st6 or
startSq <= st0 and countInCtrl(1);
-- decode sub-sequence counter
countSubSequence02H <= '1' when (countSubSequence = "00000010") else '0';
-- D8h = 216d
endSubSequence <= '1' when (countSubSequence = "11011000") else '0';
-- input data buffer shift enable
enableShift <= flagInBuf or writeRam;
-- write RAM control signal
incPointer <= st1 or st2;--or st3 or st4;
writeRam <= st2 or st3;--or st4 or st5;
-- data ram & address control
offsetNum <= countSubSequence(7 downto 1) when (st5 = '1') else "1101100"; --st7
doComp <= countSubSequence(0);
compedOffset <= ('1' & (not offsetNum) ) when (doComp = '1') else ('0' & offsetNum);
ramAddress <= adc8Nco(pointer,compedOffset,doComp);
ram : ram12x256f port map(
address => ramAddress,
inclock => c,
we => writeRam,
data => inReg1,
```

```
q => ramData);
-- constant rom
rom : rom15x128f2 port map(
address => romAddress3,
q => romData);
-- multiplier
mul : muls15x13 port map(
x => romData,
y => dataBuf2,
z => product);
-- round data
roundedSum <= (sumReg(28 downto 13) + '1') when (sumReg(12) = '1') else sumReg(28 downto
13);
overSum <= '1' when (roundedSum(16 downto 15) = "01") else '0';
underSum <= '1' when (roundedSum(16 downto 15) = "10") else '0';
resultIn <= "0111111111111111" when (overSum = '1') else
"1000000000000000" when (underSum = '1') else
roundedSum(15 downto 0);
-- output assign
deciOut <= resultReg;
end rtl;
--**************************************************************************
--Decifil3
--decimation filter 3 cutoff 0.25 sampling rate
--**************************************************************************
library ieee;
use ieee.std_logic_1164.all;
package libAdc_deciFil3 is
function adc8Nco(aa,bb : std_logic_vector(7 downto 0);cc : std_logic) return std_logic_vector;
end libAdc_deciFil3;
package body libAdc_deciFil3 is
function adc8Nco(aa,bb : std_logic_vector(7 downto 0);cc : std_logic) return std_logic_vector
```

```
is
variable interCarry : std_logic;
variable temp : std_logic_vector(aa'high downto aa'low);
begin
interCarry := cc;
for j in aa'low to aa'high loop
temp(j) := aa(j) xor bb(j) xor interCarry;
interCarry := (aa(j) and bb(j)) or ( (aa(j) or bb(j)) and interCarry);
end loop;
return(temp);
end adc8Nco;
end libAdc_deciFil3;
----------------------------------------------------------
-- 15-bit x 13-bit (signed data) multiplier
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity mulS15x13f1 is
port( x : in std_logic_vector(14 downto 0);
      y : in std_logic_vector(12 downto 0);
      z : out std_logic_vector(27 downto 0) );
end mulS15x13f1;
architecture rtl of mulS15x13f1 is
signal m1 : std_logic_vector(26 downto 14);
signal m2 : std_logic_vector(26 downto 12);
signal sumPlus : std_logic_vector(27 downto 0);
signal sumMinus : std_logic_vector(26 downto 12);
begin
sumPlus(27) <= '0';
sumPlus(26) <= x(14) and y(12);
sumPlus(25 downto 0) <= x(13 downto 0) * y(11 downto 0);
```

```
m1 <= ('0' & y(11 downto 0) ) when (x(14) = '1') else "0000000000000";
m2 <= ('0' & x(13 downto 0) ) when (y(12) = '1') else "000000000000000";
sumMinus(26 downto 14) <= m1 + m2(26 downto 14);
sumMinus(13 downto 12) <= m2(13 downto 12);
z(27 downto 12) <= sumPlus(27 downto 12) - sumMinus;
z(11 downto 0) <= sumPlus(11 downto 0);
end rtl;
------------------------------------------------------------
-- decimation filter 3
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.libAdc_deciFil3.all;
entity deciFil3 is
port( deciIn : in std_logic_vector(11 downto 0); -- data input
          wt : in std_logic; -- data write signal
           c : in std_logic; -- system clock
          nr : in std_logic; -- system reset
     deciOut : out std_logic_vector(11 downto 0); -- decimation data output
       wtOut : out std_logic); -- data output write signal
end deciFil3;
architecture rtl of deciFil3 is
component ram12x256f1
port( address : in std_logic_vector(7 downto 0);
      inclock : in std_logic;
           we : in std_logic := '1';
         data : in std_logic_vector(11 downto 0);
            q : out std_logic_vector(11 downto 0) );
end component;
component rom15x128f2
port( address : in std_logic_vector (6 downto 0);
```

```
        q : out std_logic_vector (14 downto 0) );
end component;
component mulS15x13f1
port( x : in std_logic_vector(14 downto 0);
      y : in std_logic_vector(12 downto 0);
      z : out std_logic_vector(27 downto 0) );
end component;
signal inBuf : std_logic_vector(11 downto 0); -- data input buffer
signal flagInBuf : std_logic; -- data input flag
signal enableShift : std_logic; -- data shift-in signal
signal inReg0 : std_logic_vector(11 downto 0); -- data input Reg.(4-stage)
signal inReg1 : std_logic_vector(11 downto 0);
signal inReg2 : std_logic_vector(11 downto 0);
signal inReg3 : std_logic_vector(11 downto 0);
signal countInCtrl : std_logic_vector(2 downto 0); -- input control counter
signal st0 : std_logic; -- one-hot sequencer's state
signal st1 : std_logic;
signal st2 : std_logic;
signal st3 : std_logic;
signal st4 : std_logic;
signal st5 : std_logic;
signal st6 : std_logic;
signal st7 : std_logic;
signal nonState : std_logic; -- all state zero signal
signal startSq : std_logic; -- sequencer start signal
signal incPointer : std_logic; -- pointer increment signal
signal writeRam : std_logic; -- synchronouse write signal for RAM
signal countSubSequence : std_logic_vector(7 downto 0); -- sub-sequence counter
signal countSubSequence02H : std_logic; -- decode signal of sub-sequence counter
signal endSubSequence : std_logic; -- decode signal of sub-sequence counter
signal delayedEndSubSequence : std_logic; -- decode signal of sub-sequence counter
(1-cycle delayed)
```

```
signal pointer : std_logic_vector(7 downto 0); -- pointer for center data of RAM
signal offsetNum : std_logic_vector(6 downto 0); -- offset number from center data
signal doComp : std_logic; -- calculate minus value of "offsetNum" signal
signal compedOffset : std_logic_vector(7 downto 0); -- complemented/non-complemented data of
offsetNum
signal ramAddress : std_logic_vector(7 downto 0); -- RAM address
signal ramData : std_logic_vector(11 downto 0); -- RAM's read data
signal dataBuf : std_logic_vector(11 downto 0); -- RAM's read data buffer
signal dataBuf2 : std_logic_vector(12 downto 0); -- buffer of symmetrical data's sum
signal romAddress : std_logic_vector(6 downto 0); -- address for constant ROM
signal romAddress2 : std_logic_vector(6 downto 0); -- address for constant ROM (1-cycle
delayed)
signal romAddress3 : std_logic_vector(6 downto 0); -- address for constant ROM (2-cycle
delayed)
signal romData : std_logic_vector(14 downto 0); -- constant ROM data
signal product : std_logic_vector(27 downto 0); -- multiply result
signal sumReg : std_logic_vector(28 downto 0); -- sum Reg.
signal roundedSum : std_logic_vector(12 downto 0); -- rounded data of sum Reg.
signal overSum : std_logic; -- rounded sum Reg's overflow
signal underSum : std_logic; -- rounded sum Reg's underflow
signal resultReg : std_logic_vector(11 downto 0); -- result Reg.
signal resultIn : std_logic_vector(11 downto 0); -- result Reg's source data
begin
-- ### register discription ###
-- ##################################################################
process(nr,c)
begin
        if (nr = '0') then
                inBuf <= (others => '0');
                flagInBuf <= '0';
                countInCtrl <= (others => '0');
                inReg0 <= (others => '0');
```

```
                inReg1 <= (others => '0');
                inReg2 <= (others => '0');
                inReg3 <= (others => '0');
                st0 <= '1';
                st1 <= '0';
                st2 <= '0';
                st3 <= '0';
                st4 <= '0';
                st5 <= '0';
                st6 <= '0';
                st7 <= '0';
                countSubSequence <= (others => '0');
                pointer <= (others => '0');
                dataBuf <= (others => '0');
                dataBuf2 <= (others => '0');
                romAddress <= (others => '0');
                romAddress2 <= (others => '0');
                romAddress3 <= (others => '0');
                sumReg <= (others => '0');
                delayedEndSubSequence <= '0';
                wtOut <= '0';
                resultReg <= (others => '0');
        elsif (c'event and c = '1') then
-------------------------------------------------------------
-- input data buffer & flag
                        if (wt = '1') then
                                inBuf <= deciIn;
                        end if;
flagInBuf <= wt;
-- input data control counter
        if (st5 = '1') then
                countInCtrl <= (others => '0');
```

```
        elsif (flagInBuf = '1') then
                if (countInCtrl(2) = '0') then
                        countInCtrl <= countInCtrl + '1';
                end if;
        end if;
-- input data reg.
        if (enableShift = '1') then
                inReg0 <= inBuf;
                inReg1 <= inReg0;
                inReg2 <= inReg1;
                inReg3 <= inReg2;
        end if;
-- control sequencer
                if (nonState = '1') then
                        st0 <= '1';
                elsif ( (st0 and countInCtrl(2) ) = '1') then
                        st0 <= '0';
                end if;
 st1 <= st0 and countInCtrl(2);
 st2 <= st1;
 st3 <= st2;
 st4 <= st3;
 st5 <= st4;
 st6 <= st5;
        if (st6 = '1') then
                st7 <= '1';
        elsif ( (st7 and endSubSequence) = '1') then
                st7 <= '0';
        end if;
 -- sub sequence counter
 if (st6 = '1') then
 countSubSequence <= (others => '0');
```

```
elsif (st7 = '1') then
countSubSequence <= countSubSequence + '1';
end if;
-- ram pointer
        if (incPointer = '1') then
                pointer <= pointer + '1';
        end if;
-- sum of symmetrical data
                if (countSubSequence(0) = '1') then
                        dataBuf <= ramData;
                end if;
        if (countSubSequence(0) = '0') then
                dataBuf2 <= (dataBuf(11) & dataBuf) + (ramData(11) & ramData);
        end if;
-- delay rom address
romAddress <= countSubSequence(7 downto 1);
romAddress2 <= romAddress;
romAddress3 <= romAddress2;
-- sum
        if (countSubSequence02H = '1') then
                sumReg <= (others => '0');
        elsif (countSubSequence(0) = '0') then
                sumReg <= sumReg + (product(27) & product);
        end if;
-- data output timing
delayedEndSubSequence <= endSubSequence;
wtOut <= delayedEndSubSequence;
                if (delayedEndSubSequence = '1') then
                        resultReg <= resultIn;
                end if;
-------------------------------------------------------------
        end if;
```

```
end process;
-- ### logic discription ###
-- ########################################################################
-- sequencer control
nonState <= not (st7 or st6 or st5 or st4 or st3 or st2 or st1 or st0);
startSq <= st0 and countInCtrl(2);
-- decode sub-sequence counter
countSubSequence02H <= '1' when (countSubSequence = "00000010") else '0';
-- D8h = 216d
endSubSequence <= '1' when (countSubSequence = "11011000") else '0';
-- input data buffer shift enable
enableShift <= flagInBuf or writeRam;
-- write RAM control signal
incPointer <= st1 or st2 or st3 or st4;
writeRam <= st2 or st3 or st4 or st5;
-- data ram & address control
offsetNum <= countSubSequence(7 downto 1) when (st7 = '1') else "1101100";
doComp <= countSubSequence(0);
compedOffset <= ('1' & (not offsetNum) ) when (doComp = '1') else ('0' & offsetNum);
ramAddress <= adc8Nco(pointer,compedOffset,doComp);
ram : ram12x256f1 port map(
address => ramAddress,
inclock => c,
we => writeRam,
data => inReg3,
q => ramData);
-- constant rom
rom : rom15x128f2 port map(
address => romAddress3,
q => romData);
-- multiplier
mul : muls15x13f1 port map(
```

```
x => romData,
y => dataBuf2,
z => product);
-- round data
roundedSum <= (sumReg(28 downto 18) + '1') when (sumReg(17) = '1') else sumReg(28 downto
18);
overSum <= '1' when (roundedSum(10 downto 9) = "01") else '0';
underSum <= '1' when (roundedSum(10 downto 9) = "10") else '0';
resultIn <= "0111111111" when (overSum = '1') else
"1000000000" when (underSum = '1') else
roundedSum(9 downto 0);
-- output assign
deciOut <= resultReg;
end rtl;
--**********************************************************************
-- deciFil
--
-- decimation filter for delta-sigma AD converter
--
--**********************************************************************
--
------------------------------------------------------
-- test decimation filter A
------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity testDeciFilA is
port( test : in std_logic; -- test input
nDeltaIn : in std_logic; -- delta-sigma modulate signal input (invert input)
        c : in std_logic; -- system clock
       nr : in std_logic; -- system reset
```

```
modClk : out std_logic; -- clock output for delta-sigma modulator
  wt_4k : out std_logic; -- data write signal 75k sampling rate
    wtOut : out std_logic; -- data write signal 37.5k sampling rate
      sck : out std_logic; -- bit data output for i2s bus
   i2sOut : out std_logic; -- i2s output
  nDOut7 : out std_logic; -- data output (inverted MSB for monitor DAC)
     dOut : out std_logic_vector(7 downto 0) ); -- data output
end testDeciFilA;
architecture rtl of testDeciFilA is
component deciFil1
port( deltaIn : in std_logic;
            c : in std_logic;
           nr : in std_logic;
    modClk : out std_logic;
     deciOut : out std_logic_vector(11 downto 0);
      wtOut : out std_logic);
end component;
component deciFil2
port( deciIn : in std_logic_vector(11 downto 0);
          wt : in std_logic;
           c : in std_logic;
           nr : in std_logic;
    deciOut : out std_logic_vector(15 downto 0);
      wtOut : out std_logic);
end component;
component deciFil3
port( deciIn : in std_logic_vector(11 downto 0);
          wt : in std_logic;
           c : in std_logic;
           nr : in std_logic;
    deciOut : out std_logic_vector(9 downto 0);
      wtOut : out std_logic);
```

```
end component;
signal deltaIn : std_logic; -- delta-sigma modulate signal
signal modClkBuf : std_logic; -- clock for delta-sigma modulater (buffer foropen drain output)
signal deciBuf1 : std_logic_vector(11 downto 0); -- decimation output of deciFil1
signal wtOut1 : std_logic; -- data output write signal of deciFil1
signal deciBuf2 : std_logic_vector(15 downto 0); -- decimation output of deciFil2
signal wtOut2 : std_logic; -- data output write signal of deciFil2
signal deciBuf3 : std_logic_vector(9 downto 0); --decimation output of deciFil3
signal wtout3 : std_logic; --data output write signal of deciFil3
signal extendedDeciBuf3 : std_logic_vector(8 downto 0); -- (1-bit) extended "deciBuf2(bit9 -
bit2)"
signal roundedDeci : std_logic_vector(8 downto 0); -- rounded "extendedDeciBuf3"
signal limitedDeci : std_logic_vector(7 downto 0); -- range limited "roundedDeci" (1-bit
compress)
signal limiteDeci2 : std_logic_vector(15 downto 0);
signal seledData : std_logic_vector(7 downto 0); -- multiplexed data for output
signal counter : std_logic_vector(9 downto 0); -- generate write data signal
signal Buff_i2s : std_logic_vector(15 downto 0); -- i2s buffer
signal sckCount : std_logic_vector(4 downto 0); -- generate sck i2s bus
signal Load : std_logic; -- load data R-ch and L-ch
begin
-- decimation filter 1
deltaIn <= not nDeltaIn;
fil1 : deciFil1 port map(
deltaIn => deltaIn,
c => c,
nr => nr,
modClk => modClkBuf,
deciOut => deciBuf1,
wtOut => wtOut1);
-- open drain output
modClk <= 'Z' when (modClkBuf = '1') else '0';
```

```
-- decimation filter 2
fil2 : deciFil2 port map(
deciIn => deciBuf1,
wt => wtOut1,
c => c,
nr => nr,
deciOut => deciBuf2,
wtOut => wtOut2);
--decimation filter3
fil3 : deciFil3 port map(
deciIn => deciBuf1,
wt => wtOut1,
c => c,
nr => nr,
deciOut => deciBuf3,
wtOut => wtOut3);
-- round da
extendedDeciBuf3 <= deciBuf3(9) & deciBuf3(9 downto 2);
roundedDeci <= (extendedDeciBuf3 + '1') when (deciBuf3(1) = '1') else extendedDeciBuf3;
limitedDeci <= "01111111" when (roundedDeci(8 downto 7) = "01") else roundedDeci(7 downto
0);
limiteDeci2 <= "0111111111111111" when (deciBuf2(15 downto 14) = "01") else deciBuf2(15
downto 0);
Sampling : process(c)
begin
        if (c'event and c = '1')then
                if (wtOut2 = '0') then
                        counter <= counter + '1';
                elsif (wtOut2 = '1') then
                        counter <= (others => '0');
                end if;
--Shift data to serial output
```

```
                    if (Load = '1') then
                        Buff_i2s <= limiteDeci2;
                    elsif (sckcount = "10100")then
                        Buff_i2s <= Buff_i2s(14 downto 0)& Buff_i2s(15);
                    end if;
--Select channel for i2s
                if (counter <= "1001111111") then
                    wtOut <= '1';
                end if;
        if (counter <= "0100111111")then
                wtOut <= '0';
        end if;
-- i2s clock
if (sckcount = "10100")then
sckcount <= (others => '0');
else
sckcount <= sckcount + '1';
end if;
                if (sckcount = "01010")then
                    sck <= '1';
                elsif (sckcount = "10100")then
                    sckcount <= "00000";
                    sck <= '0';
                end if;
        end if;
end process Sampling;
Load <= '1' when (counter = "1001111111")or(counter = "0101111111") else '0';
-- selector for test
seledData <= limitedDeci;
-- output assign
nDOut7 <= not seledData(7);
dOut <= seledData;
```

```
i2sOut <= Buff_i2s(15);
-- write data signal 4k bandwidth
wt_4k <= wtOut3;
end rtl;
```