การออกแบบและสร้างวงจรกรองสัญญาณเชิงเลขแบบปริภูมิสเตท ที่ใช้โครงสร้างเลขคณิตกระจายด้วย FPGA

DESIGN AND IMPLEMENTATION OF STATE-SPACE DISTRIBUTED ARITHMETIC DIGITAL FILTER USING FPGA

โดย

นางสาวพันธิวาห์ ศรสูงเนิน

นายอนุชิต ทองกู้เกียรติกูล

นายอภิชาติ เจียรสาธิต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

การออกแบบและสร้างวงจรกรองสัญญาณเชิงเลขแบบปริภูมิสเตท ที่ใช้โครงสร้างเลขคณิตกระจายด้วย FPGA

DESIGN AND IMPLEMENTATION OF STATE-SPACE DISTRIBUTED ARITHMETIC DIGITAL FILTER USING FPGA

โดย

นางสาวพันธิวาห์	ศรสูงเนิน	43010299
นายอนุชิต	ทองกู้เกียรติกูล	43010514
นายอภิชาติ	เจียรสาธิต	43010520

อาจารย์ที่ปรึกษา

รศ. ดร. กอบชัย เดชหาญ

อาจารย์ ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ปีการศึกษา 2546

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การออกแบบวงจรกรองสัญญาณเชิงเลขที่ใช้โครงสร้างเลขคณิตกระจายโดยการแทนด้วยปริภูมิสเตท**

Design and Implementation of State-Space Distributed Arithmetic Digital Filter using FPGA

ผู้จัดทำ

1. นางสาว พันธิวาห์ ศรสูงเนิน 43010299
2. นาย อนุชิต ทองกู้เกียรติกูล 43010514
3. นาย อภิชาติ เจียรสาธิต 43010520

.....

(รศ. ดร. กอบชัย เดชหาญ) อาจารย์ที่ปรึกษา

.....

(อาจารย์ ศรวัฒน์ ชิวปรีชา) อาจารย์ที่ปรึกษา

การออกแบบวงจรกรองสัญญาณเชิงเลขที่ใช้โครงสร้างเลขคณิตกระจายโดยการแทนด้วยปริภูมิสเตท

DESIGN AND IMPLEMENTATION OF STATE-SPACE DISTRIBUTED ARITHMETIC DIGITAL FILTER USING FPGA

โดย นางสาวพันธิวาห์ ศรสูงเนิน 43010299

นายอนุชิต ทองกู้เกียรติกูล 43010514

นายอภิชาติ เจียรสาธิต 43010520

อาจารย์ที่ปรึกษา รศ. ดร. กอบชัย เดชหาญ

อาจารย์ ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการออกแบบและสร้างวงจรกรองสัญญาณเชิงเลขที่ใช้โครงสร้างเลขคณิตกระจาย ซึ่งโดยทั่วไปจะสร้างขึ้นมาจากฟังก์ชั่นถ่ายโอนโดยตรง ส่วนในโครงงานนี้ได้ใช้วิธีการแทนให้อยู่ในรูปของปริภูมิสเตทก่อน ส่งผลให้จำนวนของสัญญาณอินพุทที่ใช้ในการอ้างอิงตำแหน่งของหน่วยความจำลดลง นอกจากนั้นยังได้นำเสนอวิธีการดัดแปลงสเปกตรัมก่อผิดพลาดซึ่งใช้ในการลดผลของสัญญาณรบกวนที่เกิดจากการปัดเศษ จากการคำนวณเนื่องจากผลของความยาวคำจำกัดซึ่งส่งผลโดยตรงต่ออัตราส่วนสัญญาณต่อสัญญาณรบกวนที่เกิดขึ้นที่เอาท์พุท ส่วนในแง่ของการสร้างได้ใช้ภาษา VHDL ในการบรรยายการทำงานของวงจรที่ได้ออกแบบ แล้วทำการสังเคราะห์เป็นวงจร โดยวงจรที่ได้จะถูกนำไปแม็ปลงไปยังอุปกรณ์ FPGA เพื่อทดสอบการทำงาน

ABSTRACT

This project presents a design and implementation of digital filter based on Distributed Arithmetic (DA). Generally, it can be obtained from transfer function directly, but this project is used state-space representation which results in order to reduce a number of input for memory addressing. Not only that, this project also proposes the Error Spectrum Shaping (ESS) which is the technique to reduce the roundoff noise from finite wordlength effect which affects directly with signal to noise ratio (SNR) at the output. An implementation is used VHDL to describe the hardware of this proposed design of digital filter. Finally, the synthesis and mapping onto FPGA for testing is carried out.

สารบัญ

สารบัญรูป

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ปัญหาและที่มาของปริญญานิพนธ์

โดยทั่วไปการสร้างวงจรกรองสัญญาณเชิงเลข (Digital Filter) สามารถที่จะแบ่งออกได้เป็น 2 แบบ คือ สร้างโดยการเลียนแบบ (Simulation) โครงสร้างของวงจรกรองด้วยคอมพิวเตอร์ โดยการเขียนโปรแกรมฟังก์ชั่นถ่ายโอน (Transfer Function) ของวงจรเก็บไว้แล้วนำสัญญาณที่จะกรองป้อนเข้าไปคำนวณกับโปรแกรมของฟังก์ชั่นถ่ายโอน ผลที่ได้ก็คือสัญญาณที่ผ่านวงจรกรอง การกรองแบบนี้อาจถือได้ว่าเป็นการกรองเชิงเลขในระบบเวลาไม่จริง (Non–Real Time System) อีกวิธีคือการสร้างวงจรฮาร์ดแวร์ โดยนำเอาอุปกรณ์ทางดิจิตอล เช่น ตัวเลื่อนข้อมูล (Shift Register), ตัวบวก/ลบ (Adder/Subtractor) และตัวคูณ (Multiplier) มาต่อเป็นวงจร หรือใช้ไมโครโปรเซสเซอร์, DSP Chip ก็ได้ ซึ่งการกรองแบบนี้ส่วนใหญ่เป็นการกรองในระบบเวลาจริง (Real Time System) ในปริญญานิพนธ์ฉบับนี้ได้ใช้วิธีการทางฮาร์ดแวร์ในการสร้างวงจรกรองสัญญาณ โดยนำโครงสร้างของตัวประมวลผลเลขคณิตกระจาย (Distributed Arithmetic) มาออกแบบสร้างวงจรกรองสัญญาณเชิงเลข ซึ่งส่งผลให้ความเร็วในการประมวลผลสูงทั้งยังใช้จำนวนของอุปกรณ์ต่าง ๆ น้อยทำให้วงจรกรองสัญญาณเชิงเลขที่ได้มีประสิทธิภาพสูงในการทำงาน

1.2 วัตถุประสงค์ของปริญญานิพนธ์

เนื่องจากในการใช้ไมโครโปรเซสเซอร์ หรือ DSP Chip เป็นตัวประมวลผลโดยตรง ต่างก็มีขีดจำกัดในเรื่องของความเร็วที่ใช้ในการประมวลผล เพราะในการกรองสัญญาณเชิงเลขจะต้องมีการคูณกันของสัญญาณอินพุทกับฟังก์ชั่นถ่ายโอน ซึ่งกระบวนการคูณนี้จะใช้เวลาในการทำงานมาก ส่วนในงานวิจัยนี้ได้ใช้โครงสร้างเลขคณิตกระจายในการสร้างเป็นวงจรกรองสัญญาณเชิงเลข ซึ่งจะใช้การคูณแบบเปิดตาราง (Look–up table) โดยผลบวกของผลคูณระหว่างสัญญาณอินพุทกับค่าสัมประสิทธิ์ของฟังก์ชั่นถ่ายโอนจะถูกเก็บไว้ในหน่วยความจำชนิดอ่านได้อย่างเดียว (ROM หรือ EPROM) และจะใช้สัญญาณอินพุทเป็นแอดเดรส (Address) ของหน่วยความจำโดยตรง ทำให้ลดเวลาที่ใช้ไปในกระบวนการคูณลงไปได้มาก

1.3 ทฤษฎีหรือแนวความคิดที่ใช้ในปริญญานิพนธ์

โครงสร้างเลขคณิตกระจาย เป็นการจัดรูปแบบทางคณิตศาสตร์ของสมการที่อยู่ในรูปของผลบวกของผลคูณ (Sum of Product) ระหว่างเลขฐานสิบให้กระจายออกเป็นบิท เพื่อให้การคำนวณทางคณิตศาสตร์สามารถแปลงเป็นวงจรดิจิตอลอิเล็กทรอนิกส์ได้ หลักการของโครงสร้างเลขคณิตกระจายคือการแปลงฟังก์ชั่นถ่ายโอน ซึ่งอยู่ในรูปสมการผลบวกของผลคูณระหว่างสัญญาณอินพุทกับค่าสัมประสิทธิ์ของวงจรกรอง โดยในการคูณกันระหว่างค่าสัมประสิทธิ์ของวงจรกรองกับสัญญาณอินพุทสำหรับโครงสร้างเลขคณิตกระจาย จะใช้แบบเปิดตาราง โดยค่าผลบวกของผลคูณระหว่างค่าสัมประสิทธิ์และอินพุทจะถูกเก็บ

ไว้ในหน่วยความจำ และจะใช้สัญญาณอินพุทเป็นแอดเดรสของหน่วยความจำโดยตรง ซึ่งในปริญญานิพนธ์ฉบับนี้ สัญญาณที่ใช้เป็นแอดเดรสของหน่วยความจำจะแตกต่างจากหลักการที่มีอยู่เดิม โดยได้ใช้เทคนิคการแทนสมการผลต่างสืบเนื่อง(Difference Equation) ด้วยปริภูมิสเตท (State–Space) ซึ่งจากหลักการที่มีอยู่เดิมจะใช้สมการผลต่างสืบเนื่องในการสร้างวงจรโดยตรง ส่วนหลักการที่นำเสนอนี้จะทำการนำสมการผลต่างสืบเนื่องมาแทนให้อยู่ในรูปของปริภูมิสเตทก่อน แล้วจึงนำสมการสเตท (State Equation) และสมการเอาท์พุท (Output Equation) ที่ได้จากการแทนด้วยปริภูมิสเตทมาสร้างเป็นวงจรกรองสัญญาณ ส่งผลให้จำนวนสัญญาณอินพุทที่ใช้ในการอ้างอิงตำแหน่งของหน่วยความจำมีค่าลดลง ทำให้สามารถใช้งานหน่วยความจำได้อย่างมีประสิทธิภาพขึ้น และมีขั้นตอนในการประมวลผลแตกต่างกันออกไปจากของเดิม

1.4 ขั้นตอนของการทำงาน

ในการทำงานนี้ได้แบ่งขั้นตอนการทำงานออกเป็นหัวข้อต่างๆ ดังนี้

1.4.1 ทำการศึกษาและออกแบบวงจรกรองสัญญาณป้อนกลับเชิงเลข โดยการออกแบบวงจรกรองสัญญาณเชิงอุปมานต้นแบบก่อน จากนั้นทำการแปลงให้เป็นวงจรกรองสัญญาณเชิงเลข โดยใช้การแปลงเชิงเส้นคู่

1.4.2 ทำการศึกษาถึงการนำฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงเลขที่ได้ มาทำการสร้างให้เป็นฮาร์ดแวร์โดยใช้โครงสร้างเลขคณิตกระจาย

1.4.3 ทำการศึกษาถึงรูปแบบการแทนสมการผลต่างสืบเนื่องให้อยู่ในรูปของปริภูมิสเตท รวมถึงทำการออกแบบโครงสร้างของวงจรกรองสัญญาณเชิงเลขที่ได้จากการแทนด้วยปริภูมิสเตทโดยใช้โครงสร้างเลขคณิตกระจาย

1.4.4 ทำการศึกษาถึงการนำภาษา VHDL มาใช้ในการออกแบบบรรยายการทำงานของวงจรกรองสัญญาณที่ได้ทำการพัฒนาไว้ รวมทั้งการใช้งานอุปกรณ์ FPGA มาเป็นอุปกรณ์ต้นแบบของวงจรกรองสัญญาณที่ได้

1.4.5 ทำการทดสอบการทำงานของวงจรกรองสัญญาณเชิงเลขที่ได้

1.4.6 สรุปผลการทดลอง รวมทั้งข้อเสนอแนะต่างๆ

บทที่ 2

ทฤษฎีและหลักการ

2.1 ทฤษฎีหลักการเบื้องต้นของโครงสร้างเลขคณิตกระจาย

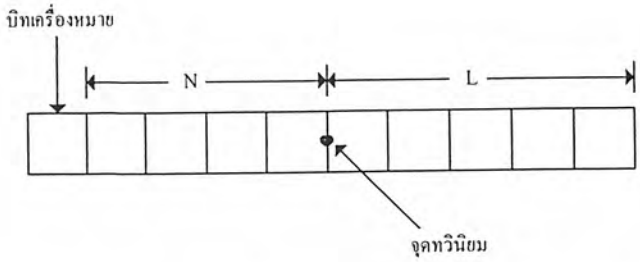
โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) หรือเรียกอย่างย่อๆว่า "DA" เป็นการจัดรูปแบบทางคณิตศาสตร์ของสมการที่อยู่ในรูปของผลบวกของผลคูณ (Sum of Product) ของเลขฐานสิบให้กระจายออกเป็นบิทหรือในรูปแบบของเลขฐานสอง เพื่อให้การคำนวณทางคณิตศาสตร์สามารถแปลงเป็นวงจรดิจิตอลอิเล็กทรอนิกส์ได้ โดยจะปรากฏอยู่ในงานทางด้านการประมวลผลสัญญาณเชิงเลข หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจายที่นำมาใช้งานด้านการประมวลผลสัญญาณเชิงเลข คือการแปลงฟังก์ชันถ่ายโอน (Transfer Function) ซึ่งเป็นสมการที่อยู่ในรูปผลบวกของผลคูณระหว่างสัญญาณอินพุทกับค่าสัมประสิทธิ์ของวงจรกรอง โดยในการคูณกันระหว่างค่าสัมประสิทธิ์ของวงจรกรองกับสัญญาณอินพุท จะใช้การคูณเลขฐานสองแบบส่วนเติมเต็มสอง (2's Complement) และการคูณจะใช้แบบเปิดตาราง (Look-Up Table) โดยค่าผลบวกของผลคูณระหว่างสัมประสิทธิ์และสัญญาณอินพุทจะถูกเก็บไว้ในหน่วย ความจำ EPROM และจะใช้สัญญาณอินพุทเป็นแอดเดรสของ EPROM โดยตรง ทั้งค่าสัมประสิทธิ์ของวงจรกรองและสัญญาณอินพุทจะถูกแทนด้วยเลขส่วนเติมเต็มสอง ดังนั้นโครงสร้างเลขคณิตกระจายจึงมีทฤษฎีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง (2's Complement Multiplication)

2.1.1 ระบบตัวเลข

สำหรับระบบเชิงเลข ตัวเลขต่างๆจะถูกแทนด้วยเลขฐานสอง ซึ่งโดยทั่วไปมีรูปแบบที่นิยมใช้กันอยู่ 2 รูปแบบ คือ รูปแบบจำนวนโดยตรง (Fixed Point Format) และรูปแบบจำนวนอิงดรรชนี (Floating Point Format) ซึ่งรูปแบบจำนวนโดยตรงจะมีวงจรฮาร์ดแวร์ที่ใช้ในการคำนวณที่ง่ายกว่า แต่ให้ค่าจากการคูณค่อนข้างจำกัด ส่วนรูปแบบจำนวนอิงดรรชนีจะสามารถแทนค่าของสัญญาณ คือให้ย่านพลวัตร (Dynamic Range) ได้มากกว่า แต่ต้องใช้วงจรฮาร์ดแวร์ที่สลับซับซ้อน แพงกว่า และให้ความเร็วในการประมวลผลที่ลดลง

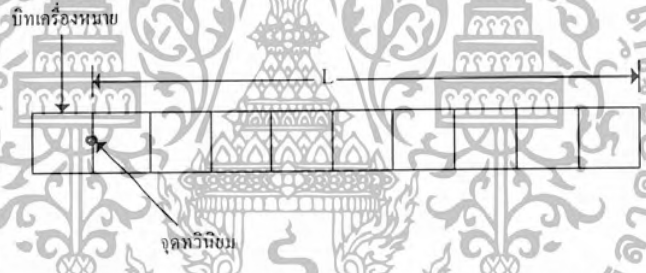
2.1.1.1 รูปแบบจำนวนโดยตรง

รูปแบบจำนวนโดยตรงปกติจะประกอบไปด้วย 3 ส่วน คือ บิทเครื่องหมาย (Sign bit) 1 บิท บิทจำนวนเต็ม (Integer bit) N บิท และบิทเศษส่วน (Fractional bit) L บิท โดยจะมีจุดทวินิยม (Binary point) อยู่ระหว่างบิทจำนวนเต็มและบิทเศษส่วน ดังแสดงในรูปที่ 2.1



รูปที่ 2.1 การจัดรูปแบบจำนวนโดยตรงที่ประกอบด้วยบิทจำนวนเต็มและบิทเศษส่วน

จำนวนบิท N เป็นตัวกำหนดย่านพลวัตรที่ต้องการ โดยถ้าเลือกให้มีจำนวนน้อยอาจทำให้เกิดการล้น (Overflow) จากการคำนวณได้ แต่ถ้าเลือกให้มีจำนวนมากความเที่ยงตรงก็จะน้อยลง ซึ่งในการสร้างวงจรกรองสัญญาณเชิงเลขโดยการแทนด้วยรูปแบบจำนวนโดยตรงนั้น นิยมที่จะทำมาตราส่วน (Scaling) เพื่อให้ขนาดของสัญญาณมีค่าอยู่ระหว่าง $-1 \le X < 1$ คือมีบิทเครื่องหมาย 1 บิท และบิทเศษส่วน L บิท ดังแสดงในรูปที่ 2.2



รูปที่ 2.2 การจัดรูปแบบจำนวนโดยตรงที่มีแต่บิทเศษส่วน

โดยทั่วไปเลขฐานสองแบบจำนวนโดยตรงแบ่งออกได้เป็น 3 รูปแบบด้วยกัน คือ (1) แบบขนาดและเครื่องหมาย (Sign magnitude) (2) แบบส่วนเติมเต็มหนึ่ง (1's complement) (3) แบบส่วนเติมเต็มสอง (2's complement) โดยคุณลักษณะที่สำคัญบางประการของการแทนตัวเลขด้วยเลขฐานสองแบบจำนวนโดยตรงทั้ง 3 รูปแบบสามารถสรุปได้ดังตารางที่ 2.1

ตารางที่ 2.1 คุณสมบัติที่สำคัญของรูปแบบจำนวนโดยตรง

Features	Sign and magnitude	2's complement	1's complement
Range	$-(1-2^{-L}) \le x \le (1-2^{-L})$	$-1 \le x \le (1-2^{-L})$	$-(1-2^{-L}) \le x \le (1-2^{-L})$
Representation of zero	0.000 and 1.000	0.000	0.000 and 1.111
Arithmetic rules	Simple must be kept track of, separately	Simple; negative numbers elegantly handled	Simple, but "end around carry" should be carefully handled
Suitability for serial arithmetic	Fair	Excellent	Good

ใน 3 รูปแบบนี้ตัวเลขแบบส่วนเติมเต็มสองเป็นที่นิยมใช้กันมากในระบบการประมวลผลสัญญาณเชิงเลข ทั้งนี้เนื่องมาจาก

1. มีการแทนค่าเลขศูนย์ได้เพียงค่าเดียว
2. การสร้างวงจรฮาร์ดแวร์สำหรับการบวก ลบและคูณ ของเลขส่วนเติมเต็มสองทำได้ง่าย โดยในการคูณสามารถใช้หลักการเลื่อนและบวก (Shift and add) หรือที่เรียกว่า บูทอัลกอริธึม (Booth's algorithm) ได้
3. ในระหว่างผลการบวกย่อย (Partial sum) ของการบวกเลขส่วนเติมเต็มสอง สามหรือสี่จำนวน ถึงแม้อาจจะเกิดการล้น (ตัวทดจากผลการบวกล้นข้ามไปทับบิทเครื่องหมาย) แต่ผลลัพธ์สุดท้ายมักให้ค่าถูกต้องเสมอ ถ้าผลบวกอยู่ในช่วง -1 ถึง $1-2^{-L}$ ดังตัวอย่าง

7/8	0.111	
+ 4/8	0.100	
11/8	1.011	ผลบวกย่อยที่ผิดเนื่องจากเกิดการล้น
- 6/8	1.010	
5/8	0.101	ผลบวกที่ถูกต้อง

2.1.1.2 รูปแบบจำนวนอิงดรรชนี

รูปแบบจำนวนโดยตรงมีข้อเสียที่สำคัญ 2 ประการ คือ (1) ย่านพลวัตรของตัวเลขมีค่าน้อย เช่น การแทนด้วยเลขส่วนเติมเต็มสอง ค่าที่น้อยที่สุดคือ –1 และค่าที่มากที่สุด คือ $1-2^{-L}$ (2) เปอร์เซ็นต์ความผิดพลาดที่เกิดจากการตัด (Truncation) หรือการปัด (Rounding) จะเพิ่มมากขึ้นเมื่อขนาดของตัวเลขมีค่าลดลง ตัวอย่างเช่น ถ้าจำนวน 0.11011010 และ 0.000110101 ถูกตัดให้จำนวนบิทเศษส่วนเหลือเพียง 4 บิท เปอร์เซ็นต์ความผิดพลาดจะเป็น 4.59 % และ 39.6 % ตามลำดับ โดยข้อเสียนี้สามารถแก้ไขได้โดยการใช้รูปแบบจำนวนอิงดรรชนี ซึ่งตัวเลข X แสดงได้โดย

$$X = M \times 2^{e} \tag{2.1}$$

โดย e เป็นจำนวนเต็ม และ $\frac{1}{2} \le |M| < 1$

M และ e เรียกว่า แมนทิสสา (Mantissa) และ เอ็กซ์โพเนนท์ (Exponent) ตามลำดับ ตัวอย่างเช่น จำนวน 0.00110101 และ 01001.11 สามารถแทนได้โดย 0.110101×2^{-2} และ 0.100111×2^{4} ตามลำดับ ส่วนจำนวนที่มีค่าเป็นลบก็ทำในลักษณะเดียวกัน รูปแบบจำนวนอิงดรรชนีสามารถแสดงได้ดังรูปที่ 2.3 โดยแบ่งเป็น 2 ส่วน คือส่วนหนึ่งสำหรับแมนทิสสา และอีกส่วนสำหรับเอ็กซ์โพเนนท์

รูปที่ 2.3 การจัดรูปแบบจำนวนอิงดรรชนี

ข้อดีของการใช้จำนวนอิงดรรชนี คือแทนค่าของสัญญาณได้ละเอียดกว่า และแม่นยำกว่าแบบจำนวนโดยตรง แต่การบวก ลบ หรือคูณจะยุ่งยากกว่ามาก วงจรจึงซับซ้อนและแพงกว่าแบบจำนวนโดยตรงมาก นอกจากนี้ความเร็วในการประมวลผลยังช้ากว่าด้วย ดังนั้นสำหรับการประมวลผลแบบเวลาจริง (Real time) จึงนิยมใช้ระบบตัวเลขแบบจำนวนโดยตรง ตารางที่ 2.2 เป็นการสรุปคุณสมบัติทางด้านคณิตศาสตร์ของตัวเลขรูปแบบจำนวนโดยตรงและรูปแบบจำนวนอิงดรรชนี

ตารางที่ 2.2 คุณสมบัติทางคณิตศาสตร์ของรูปแบบจำนวนทั้ง 2 แบบ

Features	Fixed-point fractions	Fixed-point integers	Floating-point
Overflow under multiplication	Impossible	Possible	Possible but unlikely
Overflow under addition	Possible but not harmful in most occasions	Possible	Possible but unlikely
Roundoff noise due to addition	No	No	Yes
Roundoff noise due to multiplication	Yes	No	Yes
Dynamic range available	Moderate	Moderate	Enormous
Implementation	Simple	Simple	Involved; more hardware and/or execution time required

2.1.2 ทฤษฎีเลขคณิตกระจาย

จากที่ได้กล่าวมาแล้วว่า โครงสร้างเลขคณิตกระจายมีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง ดังนั้นในหัวข้อนี้จะได้อธิบายถึงหลักการของการคูณเลขส่วนเติมเต็มสอง

ให้เลขส่วนเติมเต็มสองของ X ซึ่งแทนด้วย $\overline{X}$ และนิยามโดย

$$\overline{X} = \begin{cases} X & , X \geq 0 \\ 2 - |X| & , X < 0 \end{cases} \qquad (2.2)$$

โดย X เป็นเลขที่เป็นเศษส่วน (Fractional number)

ในระบบเลขส่วนเติมเต็มสองจะใช้บิทที่มีนัยสำคัญสูงสุด (MSB) เป็นบิทแสดงเครื่องหมาย โดยถ้าเป็นบวกแทนด้วย "0" และถ้าเป็นลบแทนด้วย "1" ถ้าให้ X แทนด้วยเลขฐานสองขนาด L+1 บิท ดังนั้นรูปแบบของเลขส่วนเติมเต็มสองจะเขียนได้ดังนี้

$$\overline{X} = X_0.X_1X_2...X_L \qquad (2.3)$$

ค่าของ $\overline{X}$ ในรูปของเลขฐานสิบสามารถหาได้ดังนี้

$$X = -X_0 + \sum_{i=1}^{L} X_i\, 2^{-i} \tag{2.4}$$

จากนั้นพิจารณาผลคูณต่อไปนี้

$$Y = X\, m \tag{2.5}$$

ให้ $\overline{Y}$, $\overline{X}$ และ $\overline{m}$ เป็นเลขส่วนเติมเต็มสองของ Y, X และ m ตามลำดับ จากนั้นพิจารณาจากสมการที่ (2.4) และ สมการที่ (2.5) จะได้

$$\begin{aligned} Y &= -Y_0 + \sum_{i=1}^{L} Y_i\, 2^{-i} \\ &= -X_0 m + \sum_{i=1}^{L} X_i\, m\, 2^{-i} \end{aligned} \tag{2.6}$$

ดังนั้น

$$\begin{aligned} \overline{Y} &= \text{ส่วนเติมเต็มสองของ} \left[-X_0 m + 2^{-1} X_1 m + 2^{-2} X_2 m + 2^{-3} X_3 m + \ldots + 2^{-L} X_L m \right] \\ &= \text{ส่วนเติมเต็มสองของ} \left[-X_0 m + 2^{-1} (X_1 m + \ldots + 2^{-1} (X_{L-1} m + 2^{-1} (X_L m))) \right] \end{aligned} \tag{2.7}$$

ต่อไปพิจารณาส่วนเติมเต็มสองของ $2^{-1}U$ โดย

$$\overline{U} = U_0 . U_1 U_2 \ldots U_M$$

สำหรับ $U \geq 0$ (หรือ $U_0 = 0$)

$$\text{ส่วนเติมเต็มสองของ } (2^{-1}U) = 2^{-1}\overline{U}$$

และสำหรับ $U < 0$ (หรือ $U_0 = 1$)

$$\text{ส่วนเติมเต็มสองของ } (2^{-1}U) = 2 - \left| 2^{-1}U \right| = 1 + 2^{-1}\left(2 - \left| U \right|\right) = 1 + 2^{-1}\overline{U}$$

ดังนั้นสรุปได้ว่า

$$\text{ส่วนเติมเต็มสองของ } (2^{-1}U) = \begin{cases} 2^{-1}\overline{U} & \text{ถ้า } U_0 = 0 \\ 1 + 2^{-1}\overline{U} & \text{ถ้า } U_0 = 1 \end{cases} \tag{2.8}$$

สมการที่ (2.8) นี้แสดงให้เห็นได้ว่า ส่วนเติมเต็มสองของ $(2^{-1}U)$ เป็นการเลื่อนข้อมูลของ $\overline{U}$ ไปทางขวา 1 บิท

$$\therefore \text{ ส่วนเติมเต็มสองของ } (2^{-1}U) = 2_2^{-1}\overline{U} \tag{2.9}$$

โดย $2_2^{-1}\overline{U}$ แสดงถึงการเลื่อนข้อมูลของ $\overline{U}$ ไปทางขวา 1 บิท แบบเลขส่วนเติมเต็มสอง ซึ่งสัญลักษณ์ 2_2^{-1} (ซึ่งโดยทั่วไปนิยมเขียนเป็น 2^{-1}) เป็นการแสดงว่าในกรณีที่ $\overline{U}$ เป็นเลขบวก ซึ่งบิทเครื่องหมายจะเป็นเลขศูนย์ โดยหลังจากเลื่อนข้อมูลไปทางขวา 1 บิทแล้ว บิทที่มาแทนบิทเครื่องหมายก็ยังคงเป็นเลขศูนย์ ส่วนในกรณีที่ $\overline{U}$ เป็นเลขลบ หลังจากเลื่อนข้อมูลไปทางขวา 1 บิทแล้ว บิทที่มาแทนบิทเครื่องหมายจะเป็นเลขหนึ่ง (จาก $1+2^{-1}\overline{U}$) ซึ่งในการสร้างเพื่อใช้งานจริงจำเป็นจะต้องมีวงจรที่ใช้ในการตรวจสอบบิทเครื่องหมาย (Sign digit) ทุกครั้งที่มีการเลื่อนข้อมูลโดยรายละเอียดจะกล่าวถึงในบทที่ 4

จากนั้นพิจารณาสมการที่ (2.7) และสมการที่ (2.8) จะได้ว่า

$$\begin{aligned}\overline{Y} &= -X_0\overline{m}+2^{-1}X_1\overline{m}+2^{-2}X_2\overline{m}+2^{-3}X_3\overline{m}+\ldots+2^{-L}X_L\overline{m} \\ &= -X_0\overline{m}+2^{-1}(X_1\overline{m}+\ldots+2^{-1}(X_{L-1}\overline{m}+2^{-1}(X_L\overline{m}))) \end{aligned} \quad (2.10)$$

ซึ่งจากสมการที่ (2.10) จะเห็นได้ว่าผลคูณจากสมการที่ (2.5) สามารถหาได้โดยการใช้หลักการเลื่อนและบวก (Shift and add) หรือ บูทอัลกอริธึมนั่นเอง โดยผลลัพธ์ที่ได้จากการคูณแบบเลขส่วนเติมเต็มสองสามารถหาได้ตามขั้นตอน ดังนี้

1. เคลียร์ค่าข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์
2. บวก $X_L\overline{m}$ กับค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์
3. เลื่อนค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ไปทางขวา 1 บิท
4. ทำซ้ำข้อ 2 และ 3 สำหรับค่า $X_{L-1},\ldots,X_1$
5. ลบค่า $X_0\overline{m}$ ออกจากค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ (ลบแบบเลขส่วนเติมเต็มสอง)

ตัวอย่างการทำงานตามอัลกอริธึมนี้

$Y = Xm = 0.8125(-0.390625)$ โดยสมมุติให้ใช้แอคคิวมูเลเตอร์รีจิสเตอร์ขนาด 12 บิท

$m = -0.390625$

$\overline{m} = 2-|m|$ $\because m$ เป็นเลขลบ

$= 2-0.390625$

$= 1.609375$

$\therefore \overline{m} = 1.100111$

$X = 0.8125 = \overline{X}$ $\because X$ เป็นเลขบวก

$\therefore \overline{X} = 0.1101 = X_0.X_1X_2X_3X_4$

โดยมีขั้นตอนการทำงาน ดังตารางต่อไปนี้

ตารางที่ 2.3 ขั้นตอนการคูณเลขส่วนเติมเต็มสอง

การดำเนินการ	ข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์
เคลียร์ ACC	0.000 0000 0000
ACC+ $X_4\overline{m}$	1.100 1110 0000
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.110 0111 0000
ACC+ $X_3\overline{m}$	1.110 0111 0000
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.111 0011 1000
ACC+ $X_2\overline{m}$	1.100 0001 1000
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.110 0000 1100
ACC+ $X_1\overline{m}$	1.010 1110 1100
เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิท	1.101 0111 0110
ACC- $X_0\overline{m}$	1.101 0111 0110

$$\therefore \quad \overline{Y} = 1.101 \quad 0111 \quad 0110 \quad = Y_0.Y_1Y_2\ldots Y_{11}$$

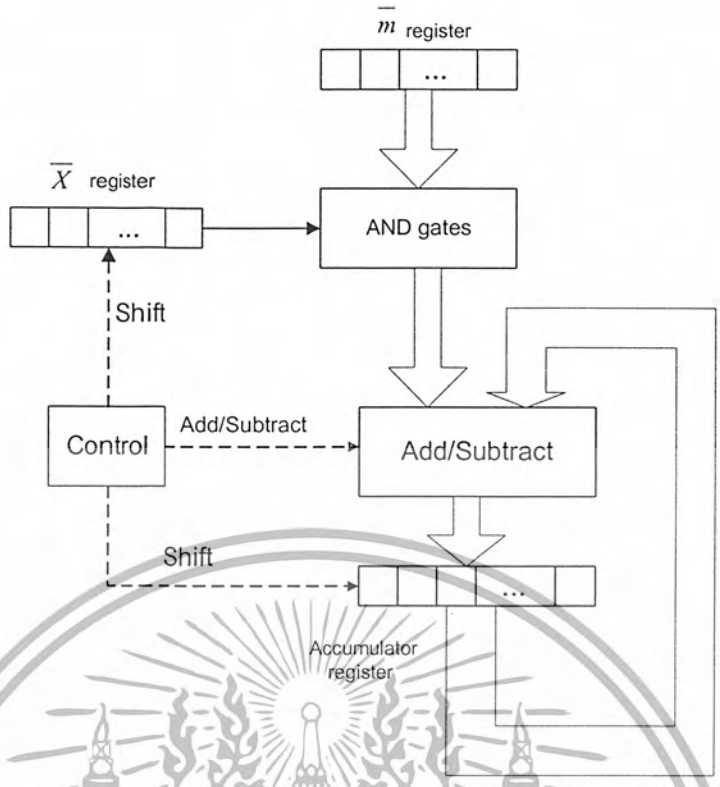
จะได้

$$Y = -Y_0 + \sum_{i=1}^{11} Y_i 2^{-i}$$

$$= -1 + \left[2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-9} + 2^{-10} \right]$$

$$= -0.3173828125$$

จากอัลกอริธึมดังกล่าว สามารถออกแบบการทำงานและสร้างวงจร แสดงได้ดังรูป



รูปที่ 2.4 การคูณแบบเลขส่วนเติมเต็มสองโดยใช้บูทอัลกอริธึม

ที่ผ่านมาเป็นหลักการคูณแบบเลขส่วนเติมเต็มสอง ส่วนทฤษฎีเลขคณิตกระจายก็อาศัยหลักการดังกล่าวมาใช้ โดยทำการกระจายสมการที่อยู่ในรูปผลบวกของผลคูณให้แตกออกมาอยู่ในระดับบิท (Bit level) ดังนี้

พิจารณาผลบวกของผลคูณต่อไปนี้

$$Y = \sum_{i=0}^{N} m_i X_i \tag{2.11}$$

โดย m_i เป็นค่าสัมประสิทธิ์ซึ่งมีค่าคงที่

X_i เป็นข้อมูลอินพุท

ถ้า X_i แต่ละค่าเป็นเลขส่วนเติมเต็มสอง โดย $|X_i| < 1$ สามารถแสดง X_i แต่ละค่าได้ดังนี้

$$X_i = -X_{i0} + \sum_{j=1}^{L} X_{ij} 2^{-j} \tag{2.12}$$

โดย X_{ij} = บิทต่างๆของข้อมูล X_i มีค่าเป็น 0 หรือ 1

X_{i0} = บิทแสดงเครื่องหมาย

X_{iL} = บิทที่มีนัยสำคัญต่ำสุด (LSB)

$L+1$ = จำนวนบิทที่แทนข้อมูลอินพุท

แทนค่า X_i ในสมการที่ (2.12) ลงในสมการที่ (2.11) จะได้

$$Y = \sum_{i=0}^{N} m_i \left[-X_{i0} + \sum_{j=1}^{L} X_{ij} 2^{-j} \right] \tag{2.13}$$

จัดเทอมของผลบวกใหม่จะได้

$$\begin{aligned} Y &= -X_{i0} \sum_{i=0}^{N} m_i + \sum_{j=1}^{L} X_{ij} 2^{-j} \sum_{i=0}^{N} m_i \\ &= -\sum_{i=0}^{N} X_{i0} m_i + \sum_{i=0}^{N} \sum_{j=1}^{L} X_{ij} m_i 2^{-j} \end{aligned} \tag{2.14}$$

จากนั้นทำการกระจายออกให้เป็นระดับบิท ได้ดังนี้

$$\begin{aligned} Y = &-(X_{00}m_0 + X_{10}m_1 + X_{20}m_2 + \ldots + X_{N0}m_N) \\ &+2^{-1}(X_{01}m_0 + X_{11}m_1 + X_{21}m_2 + \ldots + X_{N1}m_N) \\ &+2^{-2}(X_{02}m_0 + X_{12}m_1 + X_{22}m_2 + \ldots + X_{N2}m_N) \\ &+\ldots+2^{-L}(X_{0L}m_0 + X_{1L}m_1 + X_{2L}m_2 + \ldots + X_{NL}m_N) \end{aligned} \tag{2.15}$$

สมการที่ (2.15) นี้ ถูกกระจายออกให้อยู่ในรูปผลบวกของผลคูณระหว่างสัมประสิทธิ์กับข้อมูลอินพุทในระดับบิท ซึ่งเป็นนิยามของการคำนวณแบบเลขคณิตกระจาย และเมื่อเปรียบเทียบสมการที่ (2.15) กับสมการที่ (2.10) จะเห็นว่าการคำนวณหาค่า Y ก็ใช้บูทอัลกอริธึมนั่นเอง เพียงแต่นำค่าผลคูณย่อย (Partial product) ที่คำนวณไว้ล่วงหน้าแล้วสำหรับแต่ละค่าที่สอดคล้องกับแต่ละบิทของข้อมูลอินพุทไปเก็บไว้ในตารางเปิดดู ซึ่งเป็นหน่วยความจำ EPROM และใช้ข้อมูลอินพุทเป็นแอดเดรสของหน่วยความจำ เพื่อนำค่าในตารางเปิดดูมาผ่านขั้นตอนการคำนวณตามบูทอัลกอริธึม ซึ่งค่าในตารางเปิดดู สามารถแสดงได้ดังนี้

ตารางที่ 2.4 ค่าผลคูณย่อยที่เก็บไว้ในตารางเปิดดูที่กำหนดโดยข้อมูลอินพุท

Bit pattern ของข้อมูลอินพุท X_{Nj} ..... X_{2j} X_{1j} X_{0j}	ผลคูณย่อยที่เก็บไว้ในตารางเปิดดู
0 ..... 0 0 0	0
0 ..... 0 0 1	m_0
0 ..... 0 1 0	m_1
0 ..... 0 1 1	$m_1 + m_0$
0 ..... 1 0 0	m_2
0 ..... 1 0 1	$m_2 + m_0$
0 ..... 1 1 0	$m_2 + m_1$
0 ..... 1 1 1	$m_2 + m_1 + m_0$
⋮	⋮
1 ..... 1 1 1	$m_N + m_{N-1} + ... + m_2 + m_1 + m_0$

2.1.3 การนำโครงสร้างเลขคณิตกระจายมาใช้กับวงจรกรองสัญญาณเชิงเลข

ในการนำโครงสร้างเลขคณิตกระจายมาใช้ในการออกแบบ สำหรับวงจรกรองสัญญาณเชิงเลขนั้น เพื่อความสะดวกจะใช้สมการผลต่างสืบเนื่องอันดับที่สอง (Second order difference equation) มาพิจารณา เพื่อเป็นพื้นฐานในการสร้างวงจรกรองที่มีอันดับที่สูงขึ้นต่อไป

พิจารณาสมการผลต่างสืบเนื่องอันดับสองดังนี้

$$Y(n) = a_0 X(n) + a_1 X(n-1) + a_2 X(n-2) - b_1 Y(n-1) - b_2 Y(n-2) \qquad (2.16)$$

แทนลำดับสัญญาณอินพุท $X(n)$ และลำดับสัญญาณเอาต์พุท $Y(n)$ ด้วยเลขส่วนเติมเต็มสองได้ดังนี้

$$\overline{X}(n) = X_0(n).X_1(n)\ X_2(n) \ldots X_L(n)$$
$$\overline{Y}(n) = Y_0(n).Y_1(n)\ Y_2(n) \ldots \quad Y_L(n)$$

และให้ $\overline{a}_i$ และ $\overline{b}_i$ เป็นเลขส่วนเติมเต็มสองของ a_i และ b_i ตามลำดับ

$X(n)$ และ $Y(n)$ สามารถแสดงได้โดย

$$X(n) = -X_0(n) + \sum_{i=1}^{L} X_i(n)\,2^{-i}$$

$$Y(n) = -Y_0(n) + \sum_{i=1}^{L} Y_i(n)\,2^{-i}$$

นำค่า $X(n)$ และ $Y(n)$ แทนลงในสมการที่ (2.16) ได้

$$Y(n) = \sum_{i=1}^{L} 2^{-i}\left[\, a_0 X_i(n) + a_1 X_i(n-1) + a_2 X_i(n-2) - b_1 Y_i(n-1) - b_2 Y_i(n-2) \,\right]$$
$$-\left[\, a_0 X_0(n) + a_1 X_0(n-1) + a_2 X_0(n-2) - b_1 Y_0(n-1) - b_2 Y_0(n-2) \,\right] \qquad (2.17)$$

คูณ 2^{-1} ทั้ง 2 ข้างได้

$$2^{-1}Y(n) = \sum_{i=1}^{L} 2^{-i}\left[\, 2^{-1}a_0 X_i(n) + 2^{-1}a_1 X_i(n-1) + 2^{-1}a_2 X_i(n-2) - 2^{-1}b_1 Y_i(n-1)\right.$$
$$\left. -2^{-1}b_2 Y_i(n-2) \,\right] - \left[\, 2^{-1}a_0 X_0(n) + 2^{-1}a_1 X_0(n-1) + 2^{-1}a_2 X_0(n-2)\right.$$
$$\left. -2^{-1}b_1 Y_0(n-1) - 2^{-1}b_2 Y_0(n-2) \,\right] \qquad (2.18)$$

พิจารณาสมการที่ (2.18) และสมการที่ (2.9) จะได้

$$2^{-1}\bar{Y}(n) = \sum_{i=1}^{L} 2_2^{-i}\left[\, 2^{-1}\bar{a}_0 X_i(n) + 2^{-1}\bar{a}_1 X_i(n-1) + 2^{-1}\bar{a}_2 X_i(n-2) + 2^{-1}\bar{b}_1 Y_i(n-1)\right.$$
$$\left. -2^{-1}\bar{b}_2 Y_i(n-2) \,\right] - \left[\, 2^{-1}\bar{a}_0 X_0(n) + 2^{-1}\bar{a}_1 X_0(n-1) + 2^{-1}\bar{a}_2 X_0(n-2)\right.$$
$$\left. -2^{-1}\bar{b}_1 Y_0(n-1) - 2^{-1}\bar{b}_2 Y_0(n-2) \,\right] \qquad (2.19)$$

เพราะฉะนั้น

$$\bar{Y}(n) = \sum_{i=1}^{L} 2_2^{-i} F_i - F_0 \qquad (2.20)$$

โดย

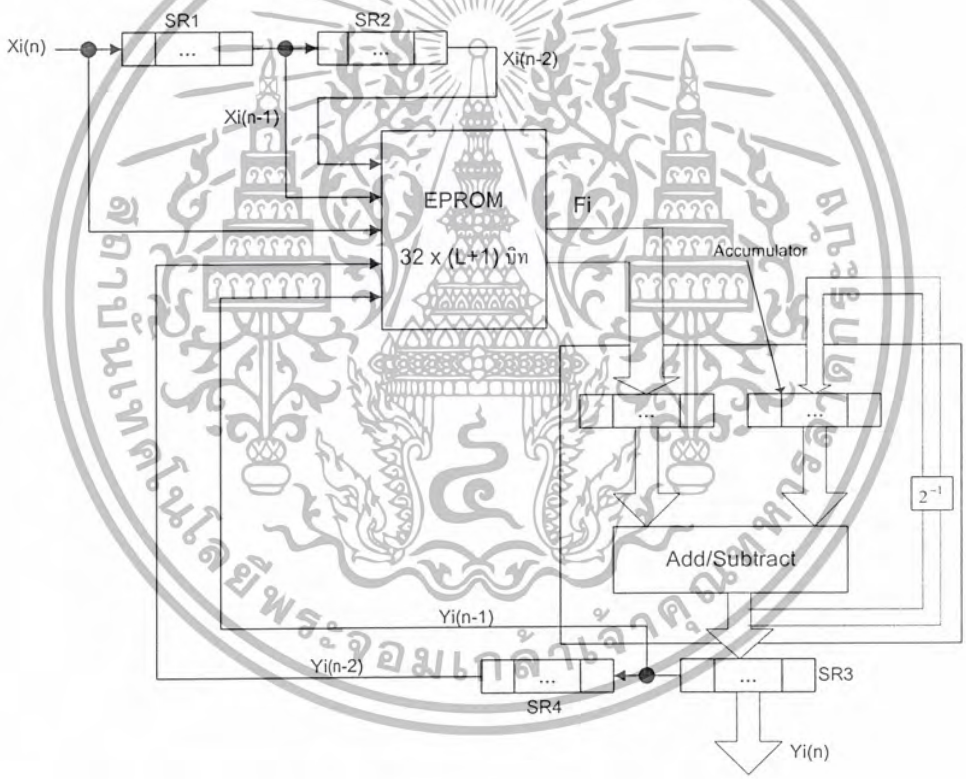
$$F_i = \bar{a}_0 X_i(n) + \bar{a}_1 X_i(n-1) + \bar{a}_2 X_i(n-2) - \bar{b}_1 Y_i(n-1) - \bar{b}_2 Y_i(n-2) \qquad (2.21)$$

$$F_0 = \bar{a}_0 X_0(n) + \bar{a}_1 X_0(n-1) + \bar{a}_2 X_0(n-2) - \bar{b}_1 Y_0(n-1) - \bar{b}_2 Y_0(n-2) \qquad (2.22)$$

ส่วนเติมเต็มสองของ $Y(n)$ สามารถหาได้โดยใช้อัลกอริธึมดังนี้

1. เคลียร์ค่าของข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์
2. คำนวณค่า F_i สำหรับ i = L
3. บวกค่า F_i กับค่าที่บรรจุอยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์
4. เลื่อนค่าที่บรรจุอยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ไปทางขวา1บิต (เลื่อนข้อมูลแบบเลขส่วนเติมเต็มสอง)
5. ทำซ้ำข้อ2 ถึง 4 สำหรับ i = L-1, L-2,...,1
6. คำนวณค่า F_0
7. ลบค่า F_0 ออกจากค่าที่บรรจุอยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ (ลบแบบเลขส่วนเติมเต็มสอง)

โดยอัลกอริธึมที่กล่าวมาสามารถออกแบบการทำงานและสร้างวงจรดังแสดงได้ดังรูป



รูปที่ 2.5 โครงสร้างวงจรกรองสัญญาณป้อนกลับเชิงเลขอันดับที่สอง

จากรูปที่ 2.5 ส่วนที่อยู่ในเส้นประนิยมเรียกกันว่าวงจรสเกลลิ่งแอคคิวมูเลเตอร์ (Scaling Accumulator) โดยค่าที่อยู่ในแอคคิวมูเลเตอร์ ก่อนที่จะส่งไปบวกกับค่าผลลัพธ์จาก EPROM (หรือ partial sum ตัวต่อไป) จะต้องทำการเลื่อนข้อมูลไปทางขวา 1 บิทก่อน ดังที่กล่าวมาแล้ว ซึ่งการเลื่อนข้อมูลไปทางขวา 1 บิทนี้ เขียนแทนด้วยการคูณด้วย 2^{-1} และผลจากสมการที่ (2.21) นำมาสร้างเป็นตารางเปิดดู

EPROM ค่าในตารางเปิดดูเป็นค่าของ F_i ซึ่งเกิดจากตัวแปรที่เป็นลำดับสัญญาณอินพุท 5 ตัว ดังนั้นค่าของ F_i จะมีค่า $2^5 = 32$ ค่า โดยขนาดของ EPROM จะมีขนาด $32 \times (L+1)$ บิท

2.2 ทฤษฎีการออกแบบวงจรกรองสัญญาณป้อนกลับเชิงเลข

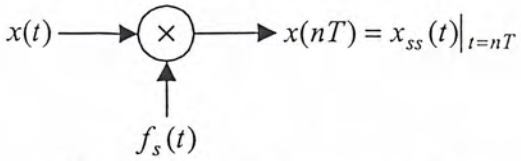
การออกแบบวงจรกรองสัญญาณป้อนกลับเชิงเลขนั้น โดยทั่วไปมีวิธีการออกแบบที่สำคัญอยู่ 2 วิธี คือ วิธีการออกแบบในโดเมน Z โดยตรง และวิธีการออกแบบโดยทำการแปลงจากฟังก์ชั่นถ่ายโอนของตัวกรองเชิงอุปมานที่มีเสถียรภาพดี โดยวิธีการนี้หากใช้การแปลงที่ดี การออกแบบก็ไม่ต้องคำนึงถึงเสถียรภาพของวงจรกรองอีก โดยที่ได้ผลตอบสนองแอมปลิจูดตามต้องการ ดังนั้นสิ่งสำคัญก็คือ การแปลงฟังก์ชั่นทางคณิตศาสตร์ของวงจรกรองสัญญาณเชิงอุปมานให้กลายเป็นฟังก์ชั่นทางคณิตศาสตร์ของวงจรกรองสัญญาณเชิงเลข โดยปกติวงจรกรองสัญญาณเชิงอุปมานจะเป็นฟังก์ชั่นของตัวแปร S (Laplace Variables) ขณะที่วงจรกรองสัญญาณเชิงเลขจะเป็นฟังก์ชั่นของตัวแปร Z ส่วนวิธีการออกแบบที่ใช้ในปริญญานิพนธ์นี้ได้เลือกใช้วิธีการหลัง ดังนั้นขั้นตอนในการออกแบบจึงแบ่งได้เป็นสองขั้นตอนคือ ออกแบบวงจรกรองสัญญาณเชิงอุปมานให้มีเสถียรภาพที่ดีซึ่งในที่นี้จะไม่กล่าวถึงรายละเอียด เนื่องจากมีการค้นคว้า ศึกษา และรวบรวมเป็นหลักการไว้อย่างดี จากนั้นจึงแปลงฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงอุปมานที่ได้ไปเป็นฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงเลข โดยในปริญญานิพนธ์นี้ได้เลือกใช้วิธีการแปลงเชิงเส้นคู่ (Bilinear Transform)

2.2.1 ทฤษฎีการสุ่มตัวอย่าง

ในการเปลี่ยนสัญญาณเชิงอุปมานไปเป็นสัญญาณเชิงเลขนั้น จำเป็นจะต้องมีการสุ่มตัวอย่าง ซึ่งความถี่ในการสุ่มตัวอย่าง โดยที่ไม่ทำให้สัญญาณสูญเสียข้อมูลที่สำคัญไปนั้นต้องเป็นไปตามทฤษฎีการสุ่มตัวอย่าง (Sampling theory) ของแชนนอน (Shannon) ซึ่งกล่าวไว้ว่า ถ้าสัญญาณเชิงอุปมาน $x(t)$ ซึ่งมีแบนด์วิดธ์เท่ากับ f_0 แล้ว จะสามารถทำการสุ่มตัวอย่างโดยสัญญาณที่ได้ไม่สูญเสียข้อมูลที่สำคัญ ก็ต่อเมื่อความถี่ในการสุ่มตัวอย่าง f_s มีค่ามากกว่าหรือเท่ากับสองเท่าของความถี่ f_0

$$f_s \geq 2f_0 \tag{2.23}$$

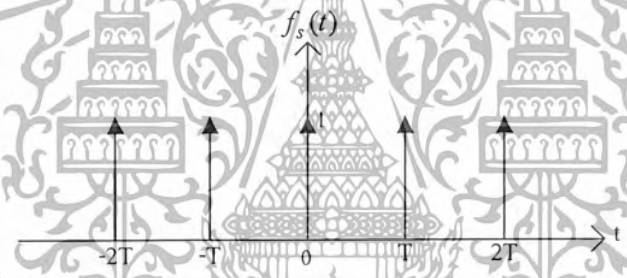
โดยทั่วไปอาจทำการสุ่มตัวอย่างด้วยความถี่ $f_{sn} = 2f_0$ พอดี ซึ่งค่าความถี่นี้เรียกว่าความถี่ไนควิสต์ (Nyquist frequency) และคาบเวลา $T_n = 1/2f_0$ นี้เรียกว่าช่วงเวลาสุ่มตัวอย่างไนควิสต์ (Nyquist interval) แต่ในทางปฏิบัติเพื่อหลีกเลี่ยงผลของปรากฏการณ์ไม่เป็นเชิงเส้น (Nonlinearity) ที่อาจเกิดจากการสุ่มตัวอย่าง จึงมักใช้ความถี่ในการสุ่มตัวอย่าง f_s มากกว่าค่าความถี่ไนควิสต์ f_{sn} ขึ้นไป ส่วนจะมีค่ามากกว่าเท่าใดนั้นขึ้นกับลักษณะงานไม่ได้มีการกำหนดค่าที่แน่นอน



รูปที่ 2.6 การสุ่มตัวอย่างสัญญาณเชิงอุปมาน

รูปที่ 2.6 แสดงการสุ่มตัวอย่างในโดเมนเวลาซึ่งก็คือการคูณสัญญาณเชิงอุปมาน $x(t)$ กับลำดับของอิมพัลส์หนึ่งหน่วย $f_s(t)$ โดยที่อิมพัลส์แต่ละตัวสมมติให้มีความห่างเท่ากับ T วินาที ซึ่งสามารถแทนได้ด้วยสมการ

$$f_s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{2.24}$$

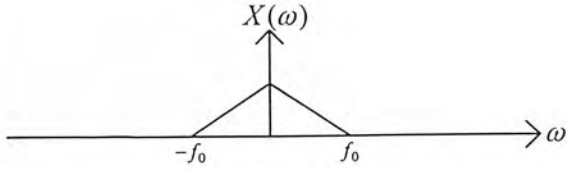


รูปที่ 2.7 สัญญาณสุ่มตัวอย่าง

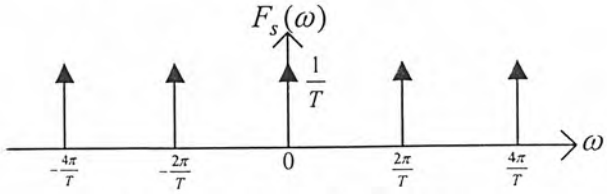
และเมื่อทำการแปลงฟูเรียร์ เพื่อทำการหาค่าสเปกตรัมความถี่ของ $f_s(t)$ จะได้

$$F_s(\omega) = \Im\{f_s(t)\} = \frac{2\pi}{T} \sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_s) \tag{2.25}$$

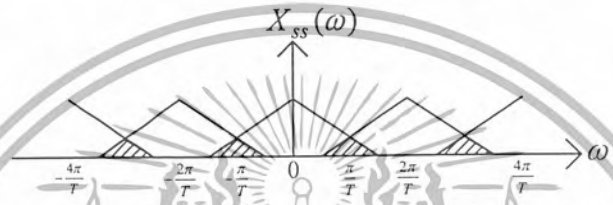
ซึ่งแสดงให้เห็นว่า เมื่อพิจารณาในโดเมนความถี่ สเปกตรัมความถี่ของสัญญาณ $f_s(t)$ เป็นอิมพัลส์ที่วางตัวห่างเท่าๆกันไปบนแกนความถี่เช่นกัน ดังแสดงในรูปที่ 2.8 (b)



(a) สเปกตรัมของสัญญาณเชิงอุปมาน $x(t)$



(b) สเปกตรัมของสัญญาณสุ่มตัวอย่าง $f_s(t)$



(c) สเปกตรัมของสัญญาณที่ได้จากการสุ่มตัวอย่าง $x_{ss}(t)$

รูปที่ 2.8 สเปกตรัมของสัญญาณจากการสุ่มตัวอย่าง

ถ้าให้ $x_{ss}(t)$ เป็นสัญญาณที่ได้จากการสุ่มตัวอย่าง ดังนั้น

$$x_{ss}(t) = f_s(t).x(t) \tag{2.26}$$

และถ้าให้ $X(\omega)$ เป็นสเปกตรัมความถี่ของ $x(t)$ เนื่องจากในโดเมนเวลาสัญญาณที่ได้จากการสุ่มตัวอย่างเป็นการคูณกันของสองสัญญาณ ดังนั้นในโดเมนความถี่จึงเป็นการทำคอนโวลูชัน (Convolution) ของสเปกตรัมของ $f_s(t)$ และ $x(t)$ หรือ

$$X_{ss}(\omega) = F_s(\omega) * X(\omega) \tag{2.27}$$

การคอนโวลูชันนี้แสดงเป็นแผนภาพดังในรูปที่ 2.8 ซึ่งผลลัพธ์ที่ได้จะเห็นว่าสเปกตรัมของสัญญาณที่ได้จากการสุ่มตัวอย่าง $X_{ss}(\omega)$ เป็นการนำสเปกตรัมของ $X(\omega)$ มาวางเรียงห่างเท่าๆกันไปตลอดบนแกนความถี่ ω ซึ่งจากรูปที่ 2.8 (c) จะเห็นว่าถ้าความถี่ของสัญญาณสุ่มตัวอย่าง $f_s(t)$ มีค่าน้อยกว่า

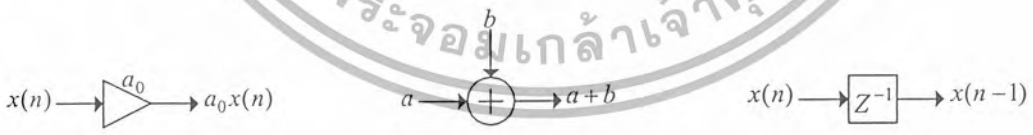
ความถี่ในควิสต์จะทำให้ช่วงห่างของแต่ละกลุ่มของสเปกตรัมเข้ามาเกยทับกัน ผลนี้ทำให้เกิดความผิดเพี้ยนไปของสเปกตรัมของสัญญาณเดิม ซึ่งผลนี้มีชื่อเรียกว่า ผลการเกิดเอเลสชิง (Aliasing effect) รูปที่ 2.8 ยังแสดงให้เห็นว่าผลตอบสนองความถี่ของวงจรกรองสัญญาณเชิงเลข จะมีผลตอบสนองความถี่ที่มีลักษณะเป็นคาบ คือเริ่มซ้ำค่าเดิมที่จุดที่มีค่าความถี่เป็น $2\pi / T$ หรือ $f_s / 2$ ซึ่งความถี่นี้เรียกว่าความถี่พับ ซึ่งการที่ผลตอบสนองความถี่มีลักษณะเป็นคาบก็เนื่องมาจากการสุ่มตัวอย่างสัญญาณนั่นเอง

2.2.2 สมการผลต่างสืบเนื่อง

ในการวิเคราะห์ระบบเชิงอุปมาน คุณสมบัติของระบบในโดเมนเวลา จะสามารถเขียนอธิบายได้โดยใช้สมการเชิงอนุพันธ์ (Differential equation) เช่นเดียวกันในระบบเชิงเลขก็จะมีสมการผลต่างสืบเนื่อง (Difference equation) ไว้ใช้ในการอธิบายคุณสมบัติของระบบในโดเมนเวลา ซึ่งสมการผลต่างสืบเนื่องอันดับที่ n สามารถเขียนได้เป็น

$$y(n) = \sum_{i=0}^{r} a_i\, x(n-i) - \sum_{i=1}^{m} b_i\, y(n-i) \tag{2.28}$$

โดยที่ $x(n)$ เป็นลำดับสัญญาณขาเข้า $y(n)$ เป็นลำดับสัญญาณขาออก และ a_i, b_i เป็นค่าสัมประสิทธิ์ จะเห็นได้ว่าในการสร้างตัวประมวลผลสัญญาณตามสมการที่ (2.28) นั้น ต้องมีอุปกรณ์ 3 ชนิด คือ ตัวคูณ (Multiplier) ตัวหน่วงลำดับสัญญาณ (Delay) หรือชิฟรีจิสเตอร์ และตัวรวมสัญญาณ (Summer) ซึ่งทั้ง 3 อุปกรณ์นี้เขียนเป็นสัญลักษณ์แทนได้ดังในรูปที่ 2.9



(a) ตัวคูณสัญญาณ (b) ตัวบวกสัญญาณ (c) ตัวหน่วงสัญญาณ

รูปที่ 2.9 อุปกรณ์ในการประมวลผลสัญญาณเชิงเลข

2.2.3 การแปลงแซด

การแปลงแซดเป็นการแปลงที่กระทำกับสัญญาณไม่ต่อเนื่อง แล้วให้ผลลัพธ์เป็นฟังก์ชั่นของตัวแปรเชิงซ้อน คือตัวแปร Z สำหรับสัญญาณ $x(n)$ ใดๆ การแปลงแซดของ $x(n)$ เขียนแทนด้วยสัญลักษณ์ $X(Z)$ มีนิยามว่า

$$Z\{x(n)\} = X(Z) = \sum_{n=-\infty}^{\infty} x(n)\, Z^{-n} \tag{2.29}$$

และการแปลงแซดผกผัน (Inverse Z – Transform) มีนิยามว่า

$$Z^{-1}\{X(Z)\} = x(n) = \frac{1}{2\pi j}\oint_C X(Z)\, Z^{n-1}\, dZ \tag{2.30}$$

โดย $Z = e^{ST}$ และโดยทั่วไปจะให้ $T = 1$ ซึ่งการแปลงแซดนี้จะนำไปใช้เป็นเครื่องมือในการวิเคราะห์ระบบแบบไม่ต่อเนื่อง สมมติว่าระบบแบบไม่ต่อเนื่องระบบหนึ่งมีผลตอบสนองอิมพัลส์ (Impulse response) เป็น $h(n)$ และการแปลงแซดของ $h(n)$ ได้ค่าเป็น $H(Z)$ ซึ่ง $H(Z)$ นี้ก็คือฟังก์ชั่นถ่ายโอน (Transfer function) ของระบบ โดยมีความสัมพันธ์กับการแปลงแซดของสัญญาณขาเข้าและสัญญาณขาออก ดังสมการ

$$H(Z) = \frac{Y(Z)}{X(Z)} \tag{2.31}$$

สมการที่ (2.31) นี้ ซึ่งเป็นสมการความสัมพันธ์ในโดเมน Z สามารถนำไปใช้ประโยชน์ในการคำนวณหาค่าต่างๆของระบบ เช่น สมการผลต่างสืบเนื่อง, $h(n)$, $H(Z)$, $y(n)$ เมื่อกำหนด $x(n)$ โดยถ้าหากทราบค่าใดค่าหนึ่ง ก็สามารถใช้การแปลงแซดในการหาค่าที่เหลืออยู่ทั้งหมดได้อย่างมีประสิทธิภาพ ในทางปฏิบัติสัญญาณ $x(n)$ ที่ใช้จะเป็นสัญญาณคอซอล (Causal signal) คือ $x(n) = 0$ เมื่อ $n < 0$ ทำให้ค่าตัวแปร n ในสมการที่ (2.29) จะมีค่าตั้งแต่ 0 จนถึง ∞ ซึ่งจะเรียกว่าเป็นการแปลงแซดด้านเดียว ซึ่งมีคุณสมบัติที่สำคัญบางประการของการแปลงแซด ดังนี้

1.) ความเป็นเชิงเส้น (Linearity)

$$Z\{ax(n) + by(n)\} = aX(Z) + bY(Z) \tag{2.32}$$

2.) การเลื่อน (Shift หรือ Translation)

$$Z\{x(n-k)\} = Z^{-k}X(Z) \tag{2.33}$$

3.) คอนโวลูชัน (Convolution)

$$Z\{f(n) * y(n)\} = Z\{f(n)\}Z\{g(n)\} = F(Z)\,G(Z) \tag{2.34}$$

2.2.4 การแปลงเชิงเส้นคู่

จากฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงอุปมานต้นแบบ เพื่อที่จะทำให้เป็นฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงเลข ในปริญญานิพนธ์นี้ได้ใช้การแปลงเชิงเส้นคู่เพื่อที่จะแปลงฟังก์ชั่นในเอสโดเมนไปเป็นฟังก์ชั่นในแซดโดเมน โดยการแปลงเชิงเส้นคู่มีหลักการอยู่บนการประมาณการอินทิเกรตด้วยกฎการอินทิเกรตแบบสี่เหลี่ยมคางหมู (Trapezoidal Integration rule) โดยพิจารณาวงจรกรองสัญญาณเชิงอุปมานที่มีฟังก์ชั่นถ่ายโอน $H(s)$ ดังสมการที่ (2.35)

$$H(s) = \frac{b}{s+a} \tag{2.35}$$

ฟังก์ชั่นถ่ายโอนตามสมการที่ (2.35) สามารถจัดให้อยู่ในรูปสมการเชิงอนุพันธ์ได้ ดังสมการที่ (2.36)

$$\frac{d\,y(t)}{dt} + a\,y(t) = b\,x(t) \tag{2.36}$$

แทนอนุพันธ์ในสมการที่ (2.36)และประมาณค่าด้วยการอินทิเกรตแบบสี่เหลี่ยมคางหมู ได้ดังสมการที่ (2.37)

$$y(t) = \int_{t_0}^{t} y'(\tau)\,d\tau + y(t_0) \tag{2.37}$$

เมื่อ $y(t)$ แทนอนุพันธ์ของ $y(t)$ การประมาณค่าของการอินทิเกรตในสมการที่ (2.36) ด้วยกฎการอินทิเกรตแบบสี่เหลี่ยมคางหมู ที่ $t = nT$ และ $t_0 = nT - T$ จะได้

$$y(nT) = \frac{T}{2} \times [y'(nT) + y'(nT - T)] + y(nT - T) \tag{2.38}$$

ดังนั้น ถ้าแทน $t = nT$ ในสมการเชิงอนุพันธ์ ที่ (2.36)จะได้

$$y'(nT) = -ay(nT) + bx(nT) \tag{2.39}$$

นำสมการที่ (2.39)แทนลงในสมการที่ (2.38)และแทน $y(n) = y(nT)$, $x(n) = x(nT)$ จะได้

$$(1+\frac{aT}{2})y(n)-(1-\frac{aT}{2})y(n-1)=\frac{bT}{2}[x(n)+x(n-1)] \tag{2.40}$$

ใช้การแปลงแซด เปลี่ยนสมการผลต่างสืบเนื่องในสมการที่ (2.40)จะได้

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

$$H(z)=\frac{Y(z)}{X(z)}=\frac{(bT/2)(1+z^{-1})}{1+(aT/2)-(1-aT/2)z^{-1}}$$

$$H(z)=\frac{b}{\frac{2}{T}\times\frac{(1-z^{-1})}{(1+z^{-1})}+a} \tag{2.41}$$

เทียบสัมประสิทธิ์จากสมการที่ (2.35) กับสมการที่ (2.41)จะได้

$$s=\frac{2}{T}\frac{(1-z^{-1})}{(1+z^{-1})} \tag{2.42}$$

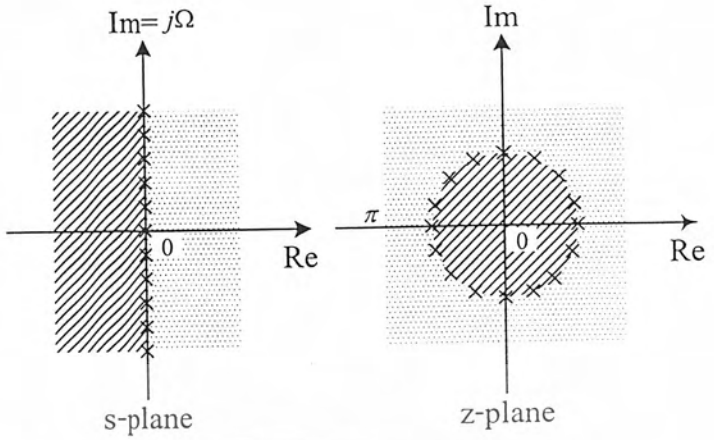
หรือในทางกลับกัน

$$z=\frac{2+sT}{2-sT} \tag{2.43}$$

โดย T คือคาบของการสุ่มตัวอย่างสัญญาณ

จากสมการที่ (2.42) และสมการที่ (2.43) จะเห็นว่าความสัมพันธ์ของตัวแปร s และ z มีลักษณะเป็นสมการเชิงเส้น (Linear) ทั้งในการแปลงจากตัวแปร s ไปเป็นตัวแปร z และในทางกลับกันจากตัวแปร z ไปเป็นตัวแปร s ด้วยเหตุนี้จึงเรียกการแปลงตัวแปรลักษณะนี้ว่าการแปลงเชิงเส้นคู่ (Bilinear Transform) โดยในการแม็ป (Mapping) จากระนาบเอสไปสู่ระนาบแซด จะเป็นไปในลักษณะหนึ่งต่อหนึ่ง (one to one mapping) และมีความสัมพันธ์ที่สำคัญ 3 ข้อ คือ

1. ค่าบนครึ่งด้านขวาของระนาบเอสจะถูกส่งไปบนบริเวณภายนอกวงกลมหนึ่งหน่วยของระนาบแซด
2. ค่าบนแกน $j\Omega$ ของระนาบเอส จะถูกส่งไปบนเส้นรอบวงของวงกลมหนึ่งหน่วยของระนาบแซด
3. ค่าบนครึ่งด้านซ้ายของระนาบเอสจะถูกส่งไปภายในวงกลมหนึ่งของระนาบแซด



รูปที่ 2.10 ความสัมพันธ์ของระนาบเอสและระนาบแซด

และจากสมการ

$$z = re^{j\omega T} \tag{2.44}$$

$$s = \sigma + j\Omega \tag{2.45}$$

$$s = \frac{2}{T} \times \frac{(1 - z^{-1})}{(1 + z^{-1})}$$

หรือ

$$s = \frac{2}{T} \times \frac{(z - 1)}{(z + 1)} \tag{2.46}$$

$$s = \frac{2}{T} \times \frac{(re^{j\omega T} - 1)}{(re^{j\omega T} + 1)} \tag{2.47}$$

$$s = \frac{2}{T} \times \left[\frac{r^2 - 1}{1 + r^2 + 2r\cos\omega T} + j \frac{2r\sin\omega T}{1 + r^2 + 2r\cos\omega T} \right] \tag{2.48}$$

เมื่อเทียบสัมประสิทธิ์จากสมการที่ (2.45) กับสมการที่ (2.48) จะได้

$$\sigma = \frac{2}{T} \times \frac{r^2 - 1}{1 + r^2 + 2r\cos\omega T} \tag{2.49}$$

$$\Omega = \frac{2}{T} \times \frac{2r\sin\omega T}{1 + r^2 + 2r\cos\omega T} \tag{2.50}$$

จากสมการที่ (2.49) และสมการที่ (2.50) จะพบว่า ถ้า $r<1$ แล้วจะได้ $\sigma<0$ และถ้า $r>1$ แล้วจะได้ $\sigma>0$ นั่นก็หมายความว่าค่าที่อยู่ทางด้านซ้ายของระนาบเอสจะไปอยู่ภายในวงกลมหนึ่งหน่วยในระนาบแซด และค่าที่อยู่ทางด้านขวาของระนาบเอสจะอยู่ภายนอกวงกลมหนึ่งหน่วยในระนาบแซด และถ้า $r=1$ แล้วจะได้ $\sigma=0$ ซึ่งหมายความว่าค่าที่อยู่บนแกนจินตภาพบนระนาบเอสจะไปอยู่บนเส้นรอบวงของวงกลมหนึ่งหน่วยบนระนาบแซด และเมื่อ $r=1$ จากสมการที่ (2.50) จะได้ค่า Ω มีค่าดังนี้

$$\Omega=\frac{2}{T}\times\frac{\sin\omega T}{1+\cos\omega T} \tag{2.51}$$

$$\Omega=\frac{2}{T}\tan\frac{\omega T}{2} \tag{2.52}$$

จากสมการที่ (2.52) นี้ เป็นความสัมพันธ์ระหว่างค่าความถี่เชิงมุมเชิงอุปมาน (Ω) กับความถี่เชิงมุมเชิงเลข (ω) ที่ผ่านการแปลงเชิงเส้นคู่ โดยความสัมพันธ์ที่ได้เป็นฟังก์ชั่นแบบแทนเจนต์ ซึ่งเป็นความสัมพันธ์ที่ไม่เป็นเชิงเส้น โดยช่วงความถี่สูงที่ได้จากการแปลงจะหดแคบลงไป ยิ่งความถี่สูงมากก็ยิ่งหดแคบมาก ซึ่งผลอันนี้เรียกว่าปรากฏการณ์หดแคบ (Wrapping effect) ดังแสดงในรูปที่ 2.11

รูปที่ 2.11 ปรากฏการณ์หดแคบที่มีผลต่อผลตอบสนองความถี่ของวงจรกรองสัญญาณเชิงเลข

ดังนั้น ในการออกแบบจึงต้องมีการชดเชยผลของปรากฏการณ์หดแคบนี้ (Prewrapping) โดยสมการที่ (2.52) ก่อนที่จะทำการแปลงตัวแปร โดยสามารถที่จะสรุปขั้นตอนการออกแบบวงจรกรองสัญญาณป้อนกลับเชิงเลขโดยการแปลงเชิงเส้นคู่ ได้ดังนี้

1. ออกแบบวงจรกรองสัญญาณเชิงอุปมานต้นแบบ โดยการหาฟังก์ชั่นถ่ายโอน $H(s)$
2. กำหนดค่าความถี่ตัดหรือความถี่ขอบแถบผ่าน รวมทั้งความถี่ที่ใช้ในการสุ่มตัวอย่างสัญญาณของวงจรกรองสัญญาณเชิงเลข
3. ทำการชดเชยผลของปรากฏการณ์หดแคบ โดยการหาค่า $\Omega = \frac{2}{T}\tan\frac{\omega T}{2}$
4. ทำการสเกลความถี่ (Frequency scaling) ของ $H(s)$ โดยแทนค่า $s = \frac{s}{\Omega}$
5. หาค่า $H(z)$ โดยแทนค่า $s = \frac{2}{T}\frac{(z-1)}{(z+1)}$

เช่นการออกแบบวงจรกรองสัญญาณความถี่ต่ำผ่านอันดับที่ 2 แบบบัตเตอร์เวิร์ท โดยกำหนดให้ความถี่ตัด (f_c) มีค่า 50 Hz และความถี่การสุ่มตัวอย่างสัญญาณ (f_s) มีค่า 250 Hz โดยมีฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงอุปมานความถี่ต่ำผ่านแบบบัตเตอร์เวิร์ทอันดับที่ 2 คือ

$$H(s) = \frac{1}{s^2 + \sqrt{2}\,s + 1}$$

$$\omega_c = 2\pi f_c = 100\pi \;;\; T = \frac{1}{250} = 4 \text{ ms}$$

ทำการชดเชยผลของปรากฏการณ์หดแคบ โดย

$$\Omega = \Omega_c = \frac{2}{T}\tan(\frac{\omega_c T}{2}) = \frac{2}{4\times10^{-3}}\tan(50\pi\times4\times10^{-3}) = 363.27$$

ทำการสเกลความถี่ โดยแทนค่า $s = \frac{s}{\Omega}$ ลงในสมการ จะได้ $H(s)$ เท่ากับ

$$H(s) = \frac{1}{\frac{s^2}{131965} + \frac{\sqrt{2}\,s}{363.27} + 1}$$

แทนค่า $s = \frac{2}{T}\frac{(z-1)}{(z+1)}$ จะได้

$$H(z) = \frac{0.2066 + 0.4131z^{-1} + 0.2066z^{-2}}{1 - 0.3695z^{-1} + 0.1958z^{-2}}$$

2.3 ทฤษฎีการวิเคราะห์โครงสร้างเลขคณิตกระจายโดยการแทนด้วยปริภูมิสเตท

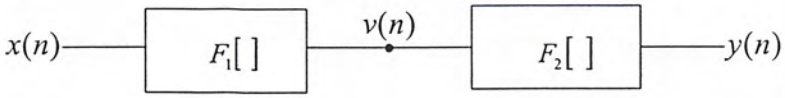
การนำโครงสร้างเลขคณิตกระจายมาใช้ในการสร้างวงจรกรองสัญญาณเชิงเลขดังที่นำเสนอไว้ใน ถูกสร้างขึ้นมาจากฟังก์ชั่นถ่ายโอนหรือสมการผลต่างสืบเนื่องโดยตรง ซึ่งสัญญาณที่ใช้เป็นแอดเดรสของหน่วยความจำจะเป็นฟังก์ชั่นของสัญญาณอินพุทและเอาท์พุท ส่วนในปริญญานิพนธ์นี้ จะทำการแทนสมการผลต่างสืบเนื่องด้วยปริภูมิสเตท ซึ่งจะทำให้สัญญาณที่เป็นแอดเดรสของหน่วยความจำเป็นฟังก์ชั่นของสัญญาณอินพุทและตัวแปรสเตท โดยจำนวนสัญญาณที่ใช้เป็นแอดเดรสของหน่วยความจำนี้จะมีจำนวนน้อยกว่าการสร้างจากฟังก์ชั่นถ่ายโอนโดยตรง จึงทำให้ขนาดของหน่วยความจำที่ใช้มีขนาดลดลงที่อันดับของวงจรเดียวกัน ทำให้มีความประหยัดในแง่ของหน่วยความจำ ส่วนการแทนระบบด้วยปริภูมิสเตทนั้น โดยปกติจะมีลักษณะที่ไม่เป็นหนึ่งเดียว (non-unique) เนื่องจากปริภูมิสเตทเป็นการแทนระบบให้อยู่ในรูปของเมตริกซ์ ซึ่งสามารถที่จะทำการแปลงคล้าย (Similarity Transformation) ได้เป็นจำนวนอนันต์แบบ แต่รูปแบบที่นำเสนอในปริญญานิพนธ์นี้เป็นรูปแบบที่มีการคำนวณทางคณิตศาสตร์น้อยที่สุด (มีการคูณกันของสัญญาณน้อยที่สุด) ซึ่งได้มาจากการแทนสมการผลต่างสืบเนื่องให้อยู่ในรูปของปริภูมิสเตทโดยตรง เรียกว่ารูปแบบบัญญัติควบคุมได้ (Controllable canonical form) ซึ่งอาจถือว่าเป็นรูปแบบโดยตรง (direct form) สำหรับปริภูมิสเตท ส่งผลให้เมื่อนำโครงสร้างเลขคณิตกระจายมาใช้ในการสร้างแล้ว การคำนวณค่าตัวแปรสเตท (State variable) ที่อันดับต่ำลงมาสามารถทำได้โดยการผ่านตัวเลื่อนข้อมูล (shift register) เท่านั้น

2.3.1 การแทนสมการผลต่างสืบเนื่องในรูปปริภูมิสเตท

ในการวิเคราะห์ระบบซึ่งในที่นี้ คือวงจรกรองสัญญาณเชิงเลขที่เป็นระบบเชิงเส้นไม่แปรตามเวลา (Linear time-invariant : LTI System) ด้วยปริภูมิสเตทนั้น เป็นการลดสมการผลต่างสืบเนื่องอันดับที่ N (N^{th} order differenc equation) ให้อยู่ในรูปของระบบที่มีสมการผลต่างสืบเนื่องเป็นอันดับ ที่ 1 (First order difference equation) จำนวน N สมการ สำหรับวงจรกรองสัญญาณเชิงเลขอันดับที่ N สามารถที่จะอธิบายด้วยสมการผลต่างสืบเนื่องดังสมการที่ (2.53)

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) - \sum_{i=1}^{N} b_i y(n-i) \qquad (2.53)$$

ทำการแบ่งสมการผลต่างสืบเนื่องให้อยู่ในลักษณะต่อเรียงกัน (cascade) ดังแสดงในรูปที่ 2.12



รูปที่ 2.12 การต่อเรียงกันของสมการผลต่างสืบเนื่อง

ซึ่งจากรูปจะได้

$$v(n) = F_1[x(n)] = \sum_{i=0}^{N} a_i x(n-i) \tag{2.54}$$

และ

$$y(n) = F_2[v(n)] = v(n) - \sum_{i=1}^{N} b_i y(n-i) \tag{2.55}$$

ต่อไปสมมุติว่า $x(n)$ ถูกจ่ายเข้าโดยตรงที่อินพุทของ F_2 และให้

$$y'(n) = F_2[x(n)]$$

ซึ่งจะทำให้ได้

$$y'(n) = x(n) - \sum_{i=1}^{N} b_i y'(n-i)$$

ขั้นตอนต่อไปจะทำการนิยามตัวแปรสเตท (state variables) ดังนี้

$$\begin{aligned} q_1(n) &= y'(n-N) \\ q_2(n) &= y'(n-N+1) \\ &\vdots \qquad\quad \vdots \\ q_N(n) &= y'(n-1) \end{aligned} \tag{2.56}$$

เพราะฉะนั้นสามารถทำการเขียนสมการได้ใหม่ ดังนี้

$$\begin{aligned} q_1(n+1) &= y'(n-N+1) = q_2(n) \\ q_2(n+1) &= y'(n-N+2) = q_3(n) \\ &\vdots \qquad\quad \vdots \qquad\quad \vdots \\ q_{N-1}(n+1) &= y'(n-1) = q_N(n) \end{aligned}$$

$$q_N(n+1) = y'(n) = x(n) - b_1 y'(n-1) - b_2 y'(n-2) - \ldots - b_N y'(n-N)$$
$$= x(n) - b_1 q_N(n) - b_2 q_{N-1}(n) - \ldots - b_N q_1(n) \tag{2.57}$$

สมการเหล่านี้สามารถเขียนให้อยู่ในรูปของเมตริกซ์ได้คือ

$$\begin{bmatrix} q_1(n+1) \\ q_2(n+1) \\ \vdots \\ q_{N-1}(n+1) \\ q_N(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -b_N & -b_{N-1} & \cdots & \cdots & -b_1 \end{bmatrix} \begin{bmatrix} q_1(n) \\ q_2(n) \\ \vdots \\ q_{N-1}(n) \\ q_N(n) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} x(n) \tag{2.58}$$

ขั้นตอนต่อไปเป็นการหาผลตอบสนองของวงจรที่มีอินพุทคือ $x(n)$ จากสมการที่ (2.54) และ (2.55)

$$y(n) = F_2[v(n)] = F_2[\sum_{i=0}^{N} a_i x(n-i)] = \sum_{i=0}^{N} a_i F_2[x(n-i)] = \sum_{i=0}^{N} a_i y'(n-i)$$

กำจัดเทอม $y'(n)$ โดยใช้สมการที่ (2.57) และเทอม $y'(n-i)$ โดยใช้สมการที่ (2.56) จะได้

$$y(n) = a_0 x(n) + c_1 q_1(n) + \ldots + c_N q_N(n)$$

โดย

$$c_1 = a_N - a_0 b_N$$
$$c_2 = a_{N-1} - a_0 b_{N-1}$$
$$\vdots \qquad \vdots$$
$$c_N = a_1 - a_0 b_1$$

เพราะฉะนั้นผลตอบสนองของวงจรสามารถเขียนให้อยู่ในรูปของเมตริกซ์ได้คือ

$$y(n) = \begin{bmatrix} c_1 & c_2 & \cdots & c_N \end{bmatrix} \begin{bmatrix} q_1(n) \\ q_2(n) \\ \vdots \\ q_N(n) \end{bmatrix} + \begin{bmatrix} a_0 \end{bmatrix} x(n) \tag{2.59}$$

สรุปได้ว่าวงจรกรองสัญญาณเชิงเลขอันดับที่ N สามารถที่จะแทนด้วยปริภูมิสเตท ดังนี้

$$q(n+1) = Aq(n) + Bx(n) \tag{2.60}$$

$$y(n) = Cq(n) + Dx(n) \tag{2.61}$$

สมการที่ (2.60) เรียกว่า สมการสเตท (state equation) และ สมการที่ (2.61) เรียกว่า สมการเอาท์พุท (output equation) ตามลำดับ A, B, C และ D เป็นเมตริกซ์ แสดงได้ดังนี้

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -b_N & -b_{N-1} & \cdots & \cdots & -b_1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_N \end{bmatrix} = \begin{bmatrix} a_N - a_0 b_N & a_{N-1} - a_0 b_{N-1} & \cdots & a_1 - a_0 b_1 \end{bmatrix}$$

$$D = \begin{bmatrix} a_0 \end{bmatrix}$$

2.3.2 การวิเคราะห์ปริภูมิสเตท

สำหรับวงจรกรองสัญญาณเชิงเลขอันดับที่ N สามารถที่จะอธิบายด้วยปริภูมิสเตทปริภูมิสเตทได้ดังสมการที่ (2.60) และ สมการที่ (2.61) คือ

$$q(n+1) = Aq(n) + Bx(n)$$

$$y(n) = Cq(n) + Dx(n)$$

โดย $x(n)$ เป็นสัญญาณอินพุท, $y(n)$ เป็นสัญญาณเอาท์พุท ซึ่งเป็นปริมาณสเกลาร์ และ q(n) เป็นสเตทเวกเตอร์ เมตริกซ์ A, B, C และ D เป็นเมตริกซ์สัมประสิทธิ์ค่าคงที่จำนวนจริง ขนาด N× N, N× 1, 1× N, 1× 1 ตามลำดับ ซึ่งฟังก์ชั่นถ่ายโอนที่อธิบายในเทอมของเมตริกซ์สัมประสิทธิ์เป็นดังสมการ

$$H(z) = C(zI-A)^{-1} B+D \tag{2.62}$$

ค่าไอเก็น (Eigen value) ของเมตริกซ์ A คือโพลของฟังก์ชั่นถ่ายโอน โดยขนาดของค่าไอเก็นทั้งหมดจะต้องน้อยกว่า 1 เพื่อความมีเสถียรภาพของระบบ ส่วนผลตอบสนองอิมพัลส์ (Impulse response) จาก $x(n)$ ถึง $y(n)$ ที่กำหนดในเทอมของ A, B, C, D กำหนดได้โดย

$$h(n) = \mathrm{CA}^{(n-1)}\mathrm{B}\,u(n-1) + \mathrm{D}\,\delta(n) \tag{2.63}$$

จากนั้นสมมุติเงื่อนไขเริ่มต้น (Initial condition) โดยให้ q(0)=0 และถ้าจ่ายสัญญาณอิมพัลส์เป็นสัญญาณอินพุท $x(n)$ ดังนั้นจะได้สเตทเวกเตอร์ คือ

$$\mathrm{q(n)} = \mathrm{A}^{n-1}\mathrm{B}\,u(n-1) \tag{2.64}$$

ดังนั้น สมการนี้เป็นผลตอบสนองอิมพัลส์จาก $x(n)$ ถึง q(n) ในลักษณะที่คล้ายกัน ภายใต้เงื่อนไขที่ให้อินพุทเป็นศูนย์ ถ้า q(0) เป็นสเตทเวกเตอร์ ที่ $n = 0$ คือ $\mathrm{q(0)}=\mathrm{q}_0$ ดังนั้น

$$y(n) = \mathrm{CA}^{n}\mathrm{q}_0\,u(n) \tag{2.65}$$

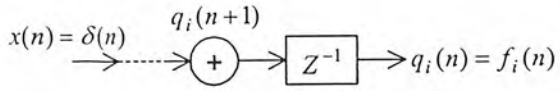
ดังนั้น ผลตอบสนองอิมพัลส์จาก $x(n)$ ถึง $q_i(n)$ คือ

$$f_i(n) = \begin{cases} [\mathrm{A}^{n-1}\mathrm{B}]_i & n \geq 0 \\ 0 & n \leq 0 \end{cases} \tag{2.66}$$

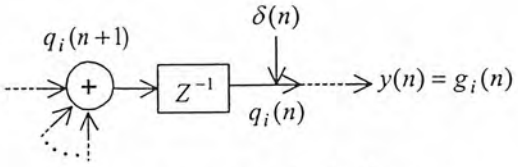
และผลตอบสนองอิมพัลส์จากสเตทเทอมินัลอันดับที่ i (i^{th} state terminal) ถึงเอาท์พุทเทอมินัล คือ

$$g_i(n) = \begin{cases} [\mathrm{CA}^{n}]_i & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{2.67}$$

โดย $f_i(n)$ และ $g_i(n)$ จะเป็นตัวแปรสำคัญที่ใช้ในกระบวนการลดระดับสัญญาณรบกวนที่เกิดจากการปัดเศษ (Roundoff Noise) ให้มีค่าน้อยที่สุด ซึ่งสามารถแสดงได้ดังรูปที่ 2.13



(a) ผลตอบสนองอิมพัลส์ $f_i(n)$



(b) ผลตอบสนองอิมพัลส์ $g_i(n)$

รูปที่ 2.13 ผลตอบสนองอิมพัลส์

2.3.2.1 การแปลงคล้าย

ในความเป็นจริงค่าของเมตริกซ์ A, B, C และ D ในสมการที่ (2.60) และ สมการที่ (2.61) มีจำนวนมากซึ่งให้ฟังก์ชั่นถ่ายโอนเหมือนกับสมการที่ (2.62) โดยผ่านการแปลงคล้าย (Similarity Transformation) ของเมตริกซ์ ซึ่งใช้เมตริกซ์ T ซึ่งเป็นเมตริกซ์ไม่เป็นเอกฐาน (Nonsingular matrix) ขนาด N × N ในการแปลงดังนี้

$$A' = T^{-1}AT,\ B' = T^{-1}B,\ C' = CT,\ D' = D \tag{2.68}$$

ระบบในสมการที่ (2.68) นี้ จะมีฟังก์ชั่นถ่ายโอนเป็น

$$H(z)' = C'(zI-A')^{-1}B'+D' \tag{2.69}$$

แทนค่าสมการที่ (2.68) ลงในสมการที่ (2.69) จะได้

$$\begin{aligned} H(z)' &= CT(zI-T^{-1}AT)^{-1}T^{-1}B+D \\ &= CTT^{-1}(zI-A)^{-1}TT^{-1}B+D \\ \therefore\quad H(z)' &= C(zI-A)^{-1}B+D = H(z) \end{aligned} \tag{2.70}$$

ถ้าวงจรกรองสัญญาณเชิงเลขเป็นระบบเชิงเส้นจะมีระบบจำนวนมากเป็นอนันต์ (infinity) ที่อธิบายได้โดยสมการที่ (2.70) สำหรับเมตริกซ์ T ที่แตกต่างกันก็จะมีพฤติกรรมที่เหมือนกัน แต่อย่างไรก็ตาม เนื่องจากวงจรกรองสัญญาณเชิงเลขในทางปฏิบัติเมื่อนำไปสร้างใช้งานจริงจะไม่เป็นระบบเชิงเส้น ทั้งนี้เนื่องจากผลของความยาวคำจำกัด (Finite wordlength effect) ดังนั้นสำหรับเมตริกซ์ T ที่ต่างกันเมื่อผ่านการแปลงคล้ายแล้ว ระบบจะมีคุณสมบัติที่ต่างกัน การเปลี่ยนไปของคุณสมบัติของระบบถูกกำหนดโดยสภาวะภายใน ซึ่งก็คือ สเตทเวกเตอร์ ดังนั้นเมื่อผ่านการแปลงคล้ายแล้วสเตทเวกเตอร์ก็จะเปลี่ยนไปเป็น

$q(n)' = T^{-1}q(n)$ ดังนั้นจะเห็นได้ว่าการออกแบบวงจรกรองสัญญาณเชิงเลขโดยการแทนด้วยปริภูมิสเตทนั้นจะมีข้อดีที่เหนือกว่าการใช้ฟังก์ชั่นถ่ายโอนโดยตรง เนื่องจากฟังก์ชั่นถ่ายโอนถูกกำหนดจากความสัมพันธ์ของอินพุทและเอาท์พุทเท่านั้นแต่ไม่ทราบถึงสภาวะภายในระบบ ส่วนปริภูมิสเตทนั้นเราสามารถที่จะทราบสภาวะภายในระบบและสามารถควบคุมสภาวะภายในนั้นได้โดยการแปลงคล้าย ซึ่งเมตริกซ์ T ที่ใช้เพื่อลดผลของความไม่เป็นเชิงเส้นที่เกิดจากผลของความยาวคำจำกัดมีวิธีในการหาหลายวิธี ขึ้นอยู่กับความเหมาะสมของการนำไปใช้งาน

2.3.2.2 ความซับซ้อนของโครงสร้างปริภูมิสเตท

ในการออกแบบวงจรกรองสัญญาณเชิงเลขแบบปริภูมิสเตทนั้น โดยทั่วไปมีข้อดีคือ สามารถลดระดับของสัญญาณรบกวนที่เกิดจากการปัดเศษของสัญญาณ (Roundoff noise) อันเป็นผลกระทบของปรากฏการณ์ไม่เป็นเชิงเส้นเนื่องจากการใช้ความยาวคำจำกัดให้มีค่าน้อยที่สุดได้ แต่โครงสร้างลักษณะนี้จะต้องใช้จำนวนตัวคูณสัญญาณมากที่สุด คือ $(N+1)^2$ ตัว ในขณะที่การออกแบบโดยใช้โครงสร้างแบบโดยตรง I ที่ได้จากสมการผลต่างสืบเนื่องโดยตรงนั้นจะใช้ตัวคูณเพียง $2N+1$ตัว ส่วนโครงสร้างปริภูมิสเตทที่นำเสนอซึ่งได้จากการแทนสมการผลต่างสืบเนื่องให้อยู่ในรูปปริภูมิสเตทโดยตรงนั้น จะใช้ตัวคูณจำนวน $2(N+1)$ ตัว ซึ่งเป็นรูปแบบของโครงสร้างปริภูมิสเตทที่มีการคำนวณน้อยที่สุด เพื่อที่จะให้เห็นถึงความแตกต่างของโครงสร้างแบบปริภูมิสเตทโดยทั่วไป ที่ใช้จำนวนตัวคูณมากที่สุดกับโครงสร้างแบบปริภูมิสเตทที่นำเสนอซึ่งใช้ตัวคูณน้อยที่สุด โดยทำการพิจารณาจากสมการที่ (2.60) และสมการที่ (2.61) ซึ่งสามารถแสดงสมการของวงจรกรองสัญญาณเชิงเลขอันดับที่ 2 ที่แทนในรูปของปริภูมิสเตทโดยทั่วไปได้ดังนี้

$$\begin{bmatrix} q_1(n+1) \\ q_2(n+1) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} q_1(n) \\ q_2(n) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} x(n) \tag{2.71}$$

$$y(n) = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} q_1(n \\ q_2(n) \end{bmatrix} + D\,x(n) \tag{2.72}$$

จากฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงเลขอันดับที่ 2 ซึ่งแสดงได้ดังนี้

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{2.73}$$

สามารถแสดงเป็นรูปแบบโดยตรงของวงจรกรองสัญญาณเชิงเลขแบบปริภูมิสเตท ที่ได้มาจากการแปลงฟังก์ชั่นถ่ายโอนโดยตรง ซึ่งแสดงได้ดังนี้

$$\begin{bmatrix} q_1(n+1) \\ q_2(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} q_1(n) \\ q_2(n) \end{bmatrix} + \begin{bmatrix} 0 \\ b_2 \end{bmatrix} x(n) \tag{2.74}$$

$$y(n) = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} q_1(n \\ q_2(n) \end{bmatrix} + D\,x(n) \tag{2.75}$$

โดย $a_{21} = -b_2$, $a_{22} = -b_1$, $b_2 = 1$, $c_1 = a_2 - a_0 b_2$, $c_2 = a_1 - a_0 b_1$, $D = a_0$

โดยสามารถแสดงเป็นกราฟการไหลของสัญญาณ (Signal Flow Graph) ของทั้งสองรูปแบบได้ ดังนี้

(a) รูปแบบโดยทั่วไป (b) รูปแบบที่นำเสนอ

รูปที่ 2.14 กราฟการไหลของสัญญาณของวงจรกรองสัญญาณเชิงเลขแบบปริภูมิสเตทอันดับที่ 2

การเลือกใช้โครงสร้างแบบใดนั้นขึ้นอยู่กับการพิจารณาสำหรับการนำไปใช้งาน โดยรูปแบบที่นำเสนอเป็นรูปแบบที่ใช้ตัวคูณสัญญาณน้อยที่สุดคือ 6 ตัว แต่ในแง่ของการลดระดับสัญญาณรบกวนที่เกิดจากการปัดเศษจะด้อยกว่ารูปแบบสัญญาณรบกวนต่ำ (Low Roundoff noise) ที่ใช้โครงสร้างแบบโดยทั่วไป ซึ่งใช้ตัวคูณมากที่สุดคือ 9 ตัว แต่ถ้ามีการกำหนดคุณลักษณะ (Specification) ของวงจรให้ดี ระดับของสัญญาณรบกวนที่เกิดจากการปัดเศษของโครงสร้างแบบที่นำเสนอ ก็จะให้ค่าต่ำใกล้เคียงกับแบบสัญญาณรบกวนต่ำ

2.4 สัญญาณรบกวนจากการปัดเศษและย่านพลวัตรในวงจรกรองสัญญาณเชิงเลข

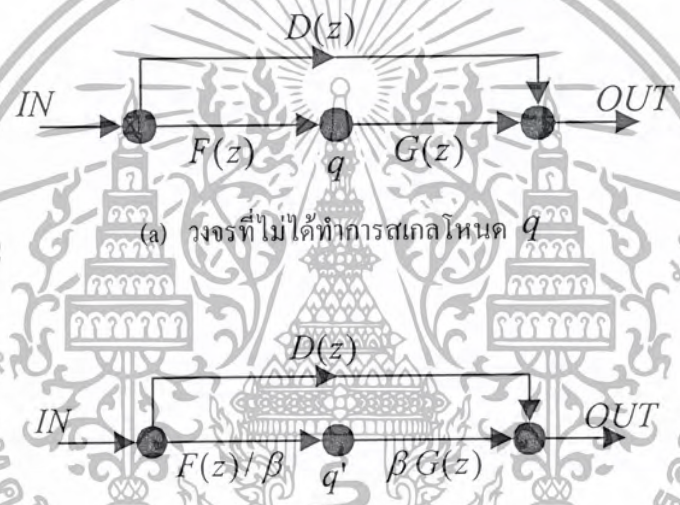
ในการสร้างวงจรกรองสัญญาณเชิงเลขที่ใช้รูปแบบจำนวนโดยตรง พฤติกรรมของสัญญาณอินพุทและเอาท์พุทจะมีลักษณะที่ไม่เป็นอุดมคติเนื่องจากการจัดระดับ (Quantization) ของสัญญาณและค่าสัมประสิทธิ์โดยใช้ความยาวคำจำกัด รวมทั้งผลของการปัดเศษผลลัพธ์ที่ได้จากการคำนวณด้วย ซึ่งถือว่าเป็นแหล่งกำเนิดของสัญญาณรบกวนที่เกิดขึ้น นอกจากนี้ยังอาจมีพฤติกรรมที่ไม่พึงปารถนา คือ การแกว่งของวัฏภาคจำกัด (Limit Cycle Oscillations) ซึ่งจะทำให้เกิดส่วนประกอบของสัญญาณที่เป็นคาบขึ้นที่เอาท์พุทถึงแม้ว่าจะไม่มีสัญญาณอินพุทก็ตาม โดยอาจมีสาเหตุเกิดมาจากการปัดภายใน (Internal

rounding) หรือเกิดการล้น (Overflow) ดังนั้นจึงต้องมีการทำการสเกลลิ่ง เพื่อบังคับให้ย่านพลวัตรของตัวแปร (Dynamic range) อยู่ภายในขนาดของความยาวคำ

โดยการอธิบายระบบด้วยตัวแปรสเตทนี้มีความเหมาะสมทางคณิตศาสตร์และเป็นประโยชน์อย่างมาก ในการคำนวณปริมาณที่ขึ้นอยู่กับโครงสร้างภายในของวงจรกรองสัญญาณเชิงเลข เช่น ค่ากำลังที่แต่ละโหนดภายใน และค่าสัญญาณรบกวนที่เกิดจากการปัดเศษที่จะเกิดขึ้นที่เอาท์พุท ซึ่งสามารถคำนวณได้โดยง่ายถ้าวงจรกรองสัญญาณเชิงเลขอธิบายด้วยรูปแบบของตัวแปรสเตท

2.4.1 การสเกลลิ่งและสัญญาณรบกวนจากการปัดเศษ

การสเกลลิ่งเป็นกระบวนการปรับปรุงค่าพารามิเตอร์ที่เป็นอัตราขยายภายใน เพื่อที่จะทำให้สัญญาณภายในอยู่ในช่วงที่เหมาะสมกับฮาร์ดแวร์ โดยที่ไม่ทำให้ฟังก์ชั่นถ่ายโอนจากอินพุทถึงเอาท์พุทเกิดการเปลี่ยนแปลง



(a) วงจรที่ไม่ได้ทำการสเกลโหนด q

(b) วงจรที่ทำการสเกลโหนด q'

รูปที่ 2.15 กราฟการไหลของวงจรกรองสัญญาณเชิงเลข

จากรูปที่ 2.15 (a) ซึ่งเป็นวงจรที่ไม่ได้ทำการสเกลโหนด q มีฟังก์ชั่นถ่ายโอนดังนี้

$$H(z) = D(z) + F(z)\,G(z) \tag{2.76}$$

ถ้าทำการสเกลโหนด q จะทำการหาร $F(z)$ ด้วยค่าคงที่บางค่า (β) และคูณ $G(z)$ กลับด้วยค่าคงที่นั้น ดังในรูปที่ 2.15 (b) ซึ่งจะเห็นได้ว่าฟังก์ชั่นถ่ายโอนจะไม่เปลี่ยนแปลง แต่ระดับสัญญาณที่โหนด q เท่านั้นที่เปลี่ยนแปลง ค่าคงที่ที่ใช้ในการสเกลลิ่งซึ่งเรียกว่า "สเกลลิ่งพารามิเตอร์" (β) สามารถที่จะเลือกตามกฎการสเกลลิ่ง (Scaling rule) ดังนี้

$$L_1\ norm \quad : \quad \beta = \sum_{n=0}^{\infty} |f(n)|$$

$$L_2\ norm \quad : \quad \beta = \sqrt{\sum_{n=0}^{\infty} |f^2(n)|} \tag{2.77}$$

โดย $f(n)$ เป็นผลตอบสนองอิมพัลส์จากอินพุทถึงโหนด q

ถ้าสัญญาณอินพุทถูกกำหนดขอบเขต (Bound) โดย $|x(n)| \le 1$ ดังนั้น

$$|q(n)| = \left| \sum_{n=0}^{\infty} f(n)\, x(k-n) \right| \le \sum_{n=0}^{\infty} |f(n)| \tag{2.78}$$

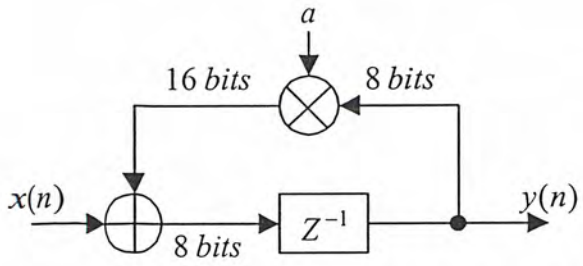
สมการที่ (2.78) แทนขอบเขตที่ถูกต้องบนช่วงของตัวแปร q และสามารถที่จะหลีกเลี่ยงการเกิดการล้นได้อย่างสมบูรณ์โดยใช้การสเกลลิ่งด้วย $L_1\ norm$ ดังสมการที่ (2.77)

สัญญาณอินพุทโดยทั่วไปจะถูกสมมุติเป็นสัญญาณรบกวนขาว (White noise) สำหรับอินพุทที่เป็นสัญญาณรบกวนขาวความแปรปรวนเป็นหนึ่ง (Unit variance white noise) ค่าความแปรปรวนที่โหนด q กำหนดโดย

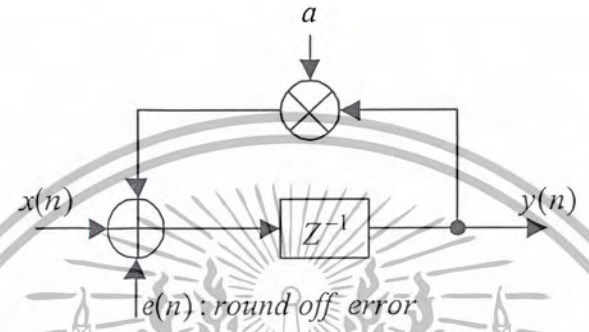
$$E[q^2(n)] = \sum_{n=0}^{\infty} f^2(n) \tag{2.79}$$

การสเกลลิ่งด้วย $L_2\ norm$ จะถูกนำมาใช้งานโดยทั่วๆ ไป เพราะสัญญาณอินพุทส่วนใหญ่สามารถที่จะสมมุติเป็นสัญญาณรบกวนขาว

ส่วนในการคูณกันของเลขจำนวนโดยตรงขนาด L บิท 2 จำนวน ผลลัพธ์ที่ได้จะมีขนาด $2L$ บิท โดยที่ผลคูณนี้จะต้องถูกจัดให้มีขนาดเป็น L บิท โดยวิธีการปัดหรือตัด ซึ่งผลที่เกิดจากการปัดหรือตัดนี้ก่อให้เกิดสัญญาณรบกวนที่เรียกว่า "สัญญาณรบกวนที่เกิดจากการปัดเศษของสัญญาณ" ขึ้น โดยพิจารณาจากวงจรกรองสัญญาณป้อนกลับเชิงเลขอันดับที่ 1 ดังแสดงในรูปที่ 2.16 สมมุติว่าความยาวคำของสัญญาณอินพุทมีขนาด 8 บิท และความยาวคำของค่าสัมประสิทธิ์ที่จะนำไปคูณก็มีขนาด 8 บิท ในการที่จะทำให้เอาท์พุทมีค่าความละเอียดเต็มช่วงจำเป็นต้องมีการเพิ่มความยาวคำของเอาท์พุทเข้าไปอีก 8 บิท ต่อรอบการคำนวณ ซึ่งเป็นไปไม่ได้ในทางปฏิบัติ ดังนั้นผลลัพธ์ที่ได้จะต้องทำการปัดหรือตัดให้แทนได้ด้วยขนาด 8 บิท ซึ่งนำไปสู่สัญญาณรบกวนที่เกิดจากการปัดเศษ $e(n)$ ดังแสดงในรูปที่ 2.17



รูปที่ 2.16 วงจรกรองสัญญาณป้อนกลับเชิงเลขอันดับที่ 1



รูปที่ 2.17 แบบจำลองของสัญญาณรบกวนที่เกิดจากการปัดเศษ

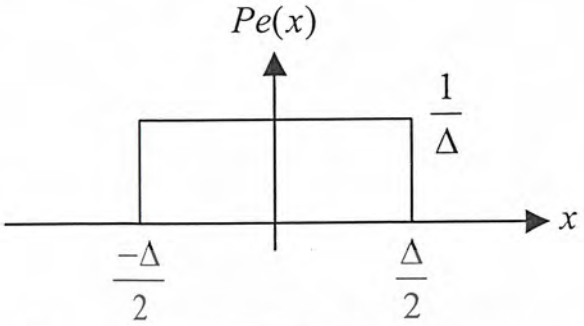
แบบจำลองคณิตศาสตร์ของสัญญาณรบกวนที่เกิดจากการปัดเศษโดยทั่วไปจะจำลองเป็นระบบที่มีความละเอียดเป็นอนันต์รวมกับสัญญาณความผิดพลาดจากภายนอกดังในรูปที่ 2.17 โดยการปัดเป็นการดำเนินการที่ไม่เป็นเชิงเส้น แต่ผลของมันที่เอาท์พุทสามารถที่จะทำการวิเคราะห์โดยใช้ทฤษฎีระบบเชิงเส้นด้วยข้อสมมุติฐานของ $e(n)$ ดังนี้

1. $e(n)$ เป็นสัญญาณรบกวนขาวที่มีการกระจายแบบเอกรูป (uniformly distributed white noise)
2. $e(n)$เป็นสัญญาณรบกวนแบบ wide-sense stationary random process (ค่าเฉลี่ยและค่าโคแวเรียนซ์ของ $e(n)$ เป็นอิสระจากดัชนีเวลา (n)
3. $e(n)$เป็นสัญญาณรบกวนที่ไม่สหสัมพันธ์ (uncorrelated) กับสัญญาณอื่นๆทั้งหมด เช่น สัญญาณอินพุท และสัญญาณรบกวนอื่นๆ

ถ้าให้ความยาวคำของเอาท์พุทเป็น L บิท ดังนั้นค่าความผิดพลาดที่เกิดจากการปัดเศษ $e(n)$ สามารถกำหนดได้โดย

$$-\frac{2^{-(L-1)}}{2} \le e(n) \le \frac{2^{-(L-1)}}{2} \tag{2.80}$$

ค่าความผิดพลาดนี้ถูกสมมุติเป็นการกระจายแบบเอกรูปเหนือช่วงในสมการที่ (2.80) ซึ่งสอดคล้องกับการกระจายของความน่าจะเป็น (Probability distribution) ดังแสดงในรูปที่ 2.18 โดย Δ คือความยาวของช่วง และ $\Delta = 2^{-(L-1)}$



รูปที่ 2.18 การกระจายความน่าจะเป็นของความผิดพลาด

ค่าเฉลี่ย (mean) $E[e(n)]$ และค่าความแปรปรวน (variance) $E[e^2(n)]$ ของฟังก์ชั่นความผิดพลาดนี้คือ

$$E[e(n)] = \int_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} x\,P_e(x)\,dx = \frac{1}{\Delta}\frac{x^2}{2}\Bigg|_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} = 0 \tag{2.81}$$

$$E[e^2(n)] = \int_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} x^2\,P_e(x)\,dx = \frac{1}{\Delta}\frac{x^3}{3}\Bigg|_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} = \frac{\Delta^2}{12} = \frac{2^{-2L}}{3} \tag{2.82}$$

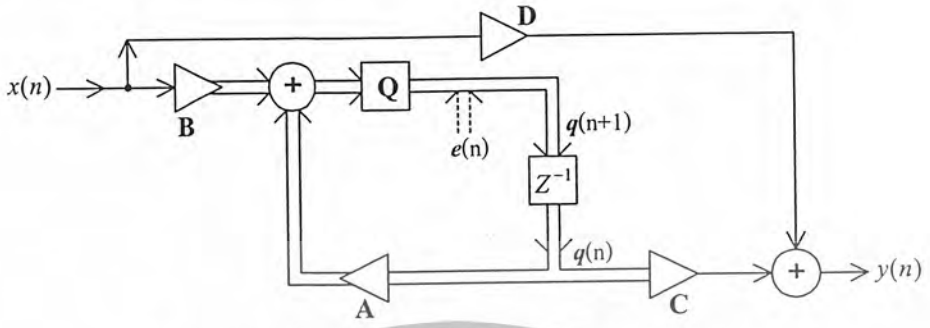
สมการที่ (2.82) สามารถเขียนใหม่ได้เป็นสมการที่ (2.83) โดย σ_e^2 เป็นค่าความแปรปรวนของความผิดพลาดที่เกิดจากการปัดในระบบที่มีความละเอียดของความยาวคำจำกัดเท่ากับ L บิท

$$\sigma_e^2 = \frac{2^{-2L}}{3} \tag{2.83}$$

จะสังเกตได้ว่าค่าความแปรปรวนเป็นสัดส่วนกับ 2^{-2L} ดังนั้นการเพิ่มความยาวคำเข้าไป 1 บิท จะทำให้ค่าความผิดพลาดลดลงเป็นสัดส่วน 4 เท่า และในการวิเคราะห์เกี่ยวกับสัญญาณรบกวนส่วนมากจะใช้ตัวแอคคิวมูเลเตอร์ที่มีความยาวเป็นสองเท่า โดยการปัดจะเกิดขึ้นหลังจากผลคูณขนาด $2L$ บิท 2 จำนวนบวกเข้าด้วยกันก่อน ซึ่งตัวคูณนี้เองที่ถือว่าเป็นแหล่งกำเนิดของสัญญาณรบกวนที่เกิดจากการปัดเศษ

2.4.2 การอธิบายตัวแปรสเตทของวงจรกรองสัญญาณเชิงเลข

พิจารณาแบบจำลองของวงจรกรองสัญญาณเชิงเลขที่แทนด้วยปริภูมิสเตทดังในรูปที่ 2.19 ซึ่งอธิบายได้ด้วยสมการที่ (2.60) และสมการที่ (2.61)



รูปที่ 2.19 แบบจำลองวงจรกรองสัญญาณเชิงเลขที่แทนด้วยปริภูมิสเตท

จากรูปแสดงแบบจำลองของวงจรกรองสัญญาณเชิงเลขที่แทนด้วยปริภูมิสเตท โดยประกอบไปด้วยตัวปัด (Quantizer) ซึ่งเปรียบเสมือนเป็นแหล่งกำเนิดสัญญาณรบกวน $e(n)$ และให้ $f_i(n)$ เป็นผลตอบสนองอิมพัลส์จากอินพุท $x(n)$ ถึงสเตท $q_i(n)$ และให้ $g_i(n)$ เป็นผลตอบสนองอิมพัลส์จากสเตท $q_i(n)$ ถึงเอาท์พุท $y(n)$ ในการพิจารณามีความจำเป็นที่จะต้องทำการสเกลสัญญาณอินพุทที่จะเข้าสู่ตัวคูณเพื่อเป็นการหลีกเลี่ยงการล้นภายใน โดยสัญญาณ $q(n)$ เป็นอินพุทที่เข้าสู่ตัวคูณดังในรูปที่ 2.19 ดังนั้นจึงต้องทำการคำนวณค่า $f(n)$ สำหรับทำการสเกลในทางกลับกัน เพื่อที่จะหาค่าความแปรปรวนของสัญญาณรบกวนที่เอาท์พุท ก็ต้องทำการหาผลตอบสนองอิมพัลส์จากตำแหน่งของแหล่งกำเนิดสัญญาณรบกวน $e(n)$ ถึง $y(n)$ ดังนั้น $g(n)$ จะแทนผลตอบสนองอิมพัลส์ของฟังก์ชั่นถ่ายโอนของสัญญาณรบกวน (Noise transfer function) จากรูปที่ 2.19 สามารถแสดงสมการได้ดังนี้

$$\frac{Q(z)}{X(z)} = \frac{B.z^{-1}}{I - z^{-1}.A} \tag{2.83}$$

ดังนั้น สามารถเขียนการแปลงแซดของ $f(n)$ คือ $F(z)$ ได้ดังนี้

$$F(z) = \frac{Q(z)}{X(z)} = (I + A\,z^{-1} + A^2\,z^{-2} + \ldots)\,B\,z^{-1} \tag{2.84}$$

เพราะฉะนั้น

$$f(n) = A^{n-1}\,B \qquad ; \quad n \geq 1 \tag{2.85}$$

ซึ่งสามารถคำนวณค่า $f(n)$ โดยแทนค่า $x(n)$ ด้วย $\delta(n)$ และใช้การเวียนกลับของสมการที่ (2.86) โดยกำหนดเงื่อนไขเริ่มต้น $f(0)=0$

$$f(n+1) = A\, f(n) + B\, \delta(n) \tag{2.86}$$

ส่วนผลตอบสนองอิมพัลส์ $g(n)$ จากสเตท $q(n)$ ถึงเอาท์พุท $y(n)$ สามารถทำการคำนวณได้ในลักษณะที่คล้ายกัน โดยให้ $x(n)=0$ ซึ่งสอดคล้องกับกราฟการไหลของสัญญาณดังแสดงในรูปที่ 2.20 โดยแทนด้วยฟังก์ชั่นถ่ายโอน $G(z)$ ดังนี้

$$G(z) = \frac{C}{I - A z^{-1}} \tag{2.87}$$

เพราะฉะนั้น

$$g(n) = C\, A^n \quad ; \quad n \geq 0 \tag{2.88}$$

รูปที่ 2.20 กราฟการไหลของสัญญาณ $g(n)$

สเตทโคแวเรียนซ์เมตริกซ์ (State covariance matrix) K มีนิยาม ดังนี้

$$K = E\left\{ q(n)\, q^T(n) \right\} \tag{2.89}$$

โดย q เป็นเวกเตอร์ขนาด $N \times 1$ และ K เป็นเมตริกซ์ขนาด $N \times N$

เมตริกซ์ K เป็นการวัดความผิดพลาดของกำลังงาน (error power) ที่สเตทต่างๆ (ส่วนประกอบในแนวทแยง (diagonal element) K_{ii} เป็นค่าพลังงานของสัญญาณความผิดพลาดที่สเตท q_i ที่เกิดจากอินพุทที่เป็นสัญญาณรบกวนขาว) สเตทเวกเตอร์ $q(n)$ สามารถคำนวณได้โดยการคอนโวลูชั่นระหว่าง $x(n)$ และ $f(n)$ โดยใช้สมการที่ (2.85) สำหรับค่า $f(n)$ ซึ่งจะได้

$$\boldsymbol{q}(n) = [\, q_1(n), q_2(n), \ldots, q_N(n) \,]^T \tag{2.90}$$

$$= \boldsymbol{f}(n) * x(n) = \sum_{l=0}^{\infty} \boldsymbol{A}^l \boldsymbol{B}\; x(n-l-1) \tag{2.91}$$

ดังนั้น

$$\boldsymbol{K} = E\left\{ \sum_{l=0}^{\infty} (\boldsymbol{A}^l \boldsymbol{B})\, x(n-l-1) \sum_{m=0}^{\infty} x(n-m-1)(\boldsymbol{A}^m \boldsymbol{B})^T \right\}$$

$$= E\left\{ \sum_{l=0}^{\infty}\sum_{m=0}^{\infty} \boldsymbol{A}^l \boldsymbol{B}\, x(n-l-1)\, x(n-m-1)(\boldsymbol{A}^m \boldsymbol{B})^T \right\}$$

$$= \sum_{l=0}^{\infty}\sum_{m=0}^{\infty} \boldsymbol{A}^l \boldsymbol{B}\; E\left[x(n-l-1)\, x(n-m-1) \right](\boldsymbol{A}^m \boldsymbol{B})^T \tag{2.92}$$

สมมุติ $x(n)$ เป็นสัญญาณรบกวนขาวความแปรปรวนเป็นหนึ่งค่าเฉลี่ยเป็นศูนย์ (zero-mean unit-variance white noise) ดังนั้นจะได้

$$E\left[x^2(n) \right] = 1 \tag{2.93}$$

$$E\left[x(n)\, x(n-k) \right] = 0 \tag{2.94}$$

แทนค่าสมการที่ (2.93) และ สมการที่ (2.94) ลงในสมการที่ (2.92) จะได้

$$\boldsymbol{K} = \sum_{l=0}^{\infty}\sum_{m=0}^{\infty} \boldsymbol{A}^l \boldsymbol{B}\; \delta_{lm}\, (\boldsymbol{A}^m \boldsymbol{B})^T = \sum_{l=0}^{\infty} \boldsymbol{f}(l)\, \boldsymbol{f}^T(l) = \sum_{l=0}^{\infty} \boldsymbol{A}^l \boldsymbol{B}\, (\boldsymbol{A}^l \boldsymbol{B})^T$$

$$= \boldsymbol{B}\boldsymbol{B}^T + \sum_{l=1}^{\infty} \boldsymbol{A}^l \boldsymbol{B}\, (\boldsymbol{A}^l \boldsymbol{B})^T = \boldsymbol{B}\boldsymbol{B}^T + \sum_{k=0}^{\infty} \boldsymbol{A}^{k+1} \boldsymbol{B}\, (\boldsymbol{A}^{k+l} \boldsymbol{B})^T$$

$$= \boldsymbol{B}\boldsymbol{B}^T + \sum_{k=0}^{\infty} \boldsymbol{A} \left[\boldsymbol{A}^k \boldsymbol{B}\, (\boldsymbol{A}^k \boldsymbol{B}) \right] \boldsymbol{A}^T = \boldsymbol{B}\boldsymbol{B}^T + \boldsymbol{A} \left[\sum_{k=0}^{\infty} \boldsymbol{A}^k \boldsymbol{B}\, (\boldsymbol{A}^k \boldsymbol{B})^T \right] \boldsymbol{A}^T \tag{2.95}$$

สุดท้ายจะได้ผลเฉลยเป็นสมการเลียปูโนฟ (Lyapunov equation) ดังนี้

$$\boldsymbol{K} = \boldsymbol{B}\boldsymbol{B}^T + \boldsymbol{A}\boldsymbol{K}\boldsymbol{A}^T \tag{2.96}$$

ถ้าบางสเตท q_i มีค่า $E\left[q_i^2 \right]$ มากกว่าของสเตทอื่นๆ ดังนั้น สเตท q_i จะต้องการจำนวนบิทของความยาวคำเพิ่มขึ้นซึ่งจะนำไปสู่การใช้ฮาร์ดแวร์ในลักษณะพิเศษและไม่เป็นที่นิยมในการออกแบบ แต่ถ้าใช้การสเกลลิ่งจะทำให้ทุกโหนดมีกำลังงานเท่ากันแน่นอนและทุกโหนดสามารถที่จะใช้ความยาวคำเดียวกันได้

โครงสร้างของวงจรกรองแบบตั้งฉาก (Orthogonal filter structure) มีนิยามคือ ตัวแปรภายในทั้งหมดจะไม่สหสัมพันธ์กันเลย และมีค่าความแปรปรวนเป็นหนึ่งโดยที่สมมุติให้อินพุทเป็นสัญญาณรบกวนขาว ซึ่งจะทำให้ได้เงื่อนไขการตั้งฉากกันดังนี้

$$K = I = AA^T + BB^T \tag{2.97}$$

ข้อดีของโครงสร้างวงจรกรองแบบตั้งฉากคือ

- จะเข้าเงื่อนไขของกฎการสเกลลิ่งโดยอัตโนมัติ
- อัตราขยายของสัญญาณรบกวนที่เกิดจากการปัดเศษมีค่าต่ำและไม่เปลี่ยนแปลงภายใต้การแปลงความถี่ (Frequency transformation)
- การแกว่งเนื่องจากการล้น (Overflow oscillations) จะไม่เกิดขึ้น

ในลักษณะที่คล้ายกัน นิยามของเอาท์พุทโคแวเรียนซ์เมตริกซ์ (Output covariance matrix) W สามารถแสดงได้ดังนี้

$$W = \sum_{n=0}^{\infty} g^T(n)\, g(n) = \sum_{n=0}^{\infty} (C A^n)^T C A^n \tag{2.98}$$

สุดท้ายจะได้ผลเฉลยเป็นสมการเลียปูโนฟ ดังนี้

$$W = A^T W A + C^T C \tag{2.99}$$

2.4.3 การคำนวณการสเกลลิ่งและสัญญาณรบกวนจากการปัดเศษ

ที่ขนาดความยาวคำเดียวกันสามารถที่จะใช้กับตัวแปรทั้งหมดของระบบได้ ถ้าสเตทที่งหมดมีกำลังเท่ากัน ซึ่งจะได้มาจากการสเกลลิ่งสเตทเวกเตอร์ด้วยการคูณกับค่าอินเวิร์สของสเกลลิ่งเมตริกซ์ T และถ้าแทนสเตทที่ทำการสเกลแล้วด้วย q_s ดังนั้นสามารถแสดงได้ดังนี้

$$q_s(n) = T^{-1} q(n) \quad \Rightarrow \quad q(n) = T\, q_s(n) \tag{2.100}$$

แทนค่า q จากสมการที่ (2.100) ลงในสมการที่ (2.60) เพื่อหาผลเฉลยคือ q_s ได้ดังนี้

$$T\, q_s(n+1) = A\, T\, q_s(n) + B\, x(n) \tag{2.101}$$

$$q_s(n+1) = T^{-1} A\, T\, q_s(n) + T^{-1} B\, x(n) \tag{2.102}$$

$$q_s(n+1) = A_s\, q_s(n) + B_s\, x(n) \tag{2.103}$$

โดย $A_s = T^{-1} A\, T, \quad B_s = T^{-1} B$

ในลักษณะที่คล้ายกัน จากสมการเอาท์พุทคือสมการที่ (2.61) สามารถแสดงได้ดังนี้

$$\begin{aligned} y(n) &= C\, T\, q_s(n) + D\, x(n) \\ &= C_s\, q_s(n) + D_s\, x(n) \end{aligned} \tag{2.104}$$

โดย $C_s = C\, T, \quad D_s = D$

ส่วนเมตริกซ์ K ที่ทำการสเกลแล้วกำหนดโดย

$$K_s = E\left[q_s\, q_s^T\right] = E\left[T^{-1} q\, q^T (T^{-1})^T\right] = T^{-1} E\left[q\, q^T\right](T^{-1})^T$$

$$K_s = T^{-1} K (T^{-1})^T \tag{2.105}$$

เนื่องจากสิ่งที่ต้องการคือสเตทุกสเตทมีค่ากำลังที่เท่ากัน ดังนั้นเมตริกซ์ที่ใช้ในการแปลง T จะต้องทำให้เมตริกซ์ K_s ของระบบที่ทำการสเกลแล้วมีส่วนประกอบในแนวทแยงเป็น 1 นอกจากนี้สมมุติให้ T เป็นเมตริกซ์ในแนวทแยง คือ

$$T = diag\left[t_{11}, t_{22}, \ldots, t_{NN}\right] \tag{2.106}$$

$$T^{-1} = diag\left[\frac{1}{t_{11}}, \frac{1}{t_{22}}, \ldots, \frac{1}{t_{NN}}\right] = (T^{-1})^T \tag{2.107}$$

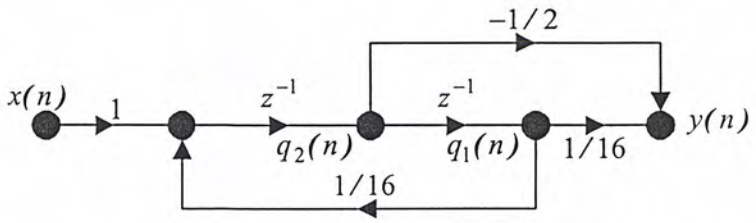
จากสมการที่ (2.105) ถึง สมการที่ (2.107) โดยให้ $(K_s)_{ii} = 1$ จะได้

$$(K_s)_{ii} = \frac{K_{ii}}{t_{ii}^2} = 1 \tag{2.108}$$

$$t_{ii} = \sqrt{K_{ii}} \tag{2.109}$$

สรุปได้ว่าโดยการเลือกค่าในแนวทแยงของเมตริกซ์ T ให้มีค่าเท่ากับรากที่สองของค่าในแนวทแยงของเมตริกซ์ K จะทำให้ค่าสเตททั้งหมดในระบบจะมีกำลังที่เท่ากัน

เช่นการพิจารณาวงจรกรองสัญญาณอันดับที่ 2 ซึ่งยังไม่ได้ทำการสเกลดังรูปที่ 2.21 โดยมีค่าเมตริกซ์ของตัวแปรสเตทดังนี้



รูปที่ 2.21 กราฟการไหลของสัญญาณของวงจรกรองอันดับที่ 2 ซึ่งไม่ได้ทำการสเกล

$$A=\begin{bmatrix}0 & 1\\ \frac{1}{16} & 0\end{bmatrix},\quad B=\begin{bmatrix}0\\ 1\end{bmatrix},\quad C=\begin{bmatrix}\frac{1}{16} & \frac{-1}{2}\end{bmatrix},\quad D=0$$

สเตทโคแวเรียนซ์เมตริกซ์ K สามารถคำนวณได้โดยใช้สมการที่ (2.96) คือ

$$\begin{bmatrix}K_{11} & K_{12}\\ K_{21} & K_{22}\end{bmatrix}=\begin{bmatrix}0 & 1\\ \frac{1}{16} & 0\end{bmatrix}\begin{bmatrix}K_{11} & K_{12}\\ K_{21} & K_{22}\end{bmatrix}\begin{bmatrix}0 & \frac{1}{16}\\ 1 & 0\end{bmatrix}+\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}$$

$$=\begin{bmatrix}K_{22} & \frac{1}{16}K_{21}\\ \frac{1}{16}K_{12} & \frac{1}{256}K_{11}+1\end{bmatrix}$$

ดังนั้น จะได้ $K_{11}=K_{22}=\frac{256}{255},\quad K_{12}=K_{21}=0$

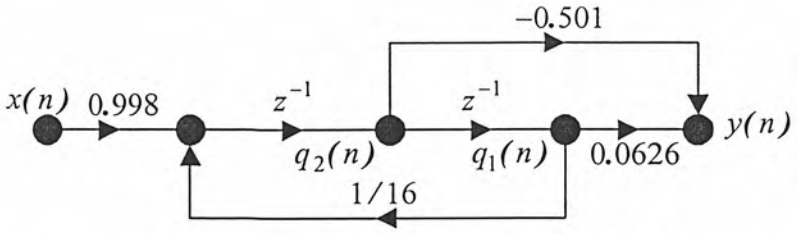
สำหรับการสเกลด้วย L_2 *norm* จะได้เมตริกซ์ที่ใช้ในการแปลง คือ

$$T=\begin{bmatrix}\frac{16}{\sqrt{255}} & 0\\ 0 & \frac{16}{\sqrt{255}}\end{bmatrix}$$

ดังนั้น วงจรกรองที่ผ่านการสเกลจะอธิบายด้วยเมตริกซ์ดังต่อไปนี้ และแสดงดังในรูปที่ 2.22

$$A_s=T^{-1}AT=\begin{bmatrix}0 & 1\\ \frac{1}{16} & 0\end{bmatrix},\qquad B_s=T^{-1}B=\begin{bmatrix}0\\ \frac{\sqrt{255}}{16}\end{bmatrix}$$

$$C_s=CT=\begin{bmatrix}\frac{1}{\sqrt{255}} & \frac{-8}{\sqrt{255}}\end{bmatrix},\qquad D_s=0$$



รูปที่ 2.22 กราฟการไหลของสัญญาณของวงจรกรองอันดับที่ 2 ซึ่งผ่านการสเกล

สังเกตว่า สเตทโคแวเรียนซ์เมตริกซ์ K_s ของวงจรกรองที่ทำการสเกลแล้ว คือ

$$K_s = \begin{bmatrix} \frac{\sqrt{255}}{16} & 0 \\ 0 & \frac{\sqrt{255}}{16} \end{bmatrix} \begin{bmatrix} \frac{256}{255} & 0 \\ 0 & \frac{256}{255} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{255}}{16} & 0 \\ 0 & \frac{\sqrt{255}}{16} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

การคำนวณสัญญาณรบกวนที่เกิดจากการปัดเศษทำการพิจารณาโดยให้ $e_i(n)$ เป็นความผิดพลาดที่เกิดจากการปัดเศษที่สเตท q_i ดังนั้นสัญญาณรบกวนจากการปัดเศษที่เอาท์พุท $y_i(n)$ ที่เกิดจากความผิดพลาดนี้สามารถเขียนเป็นการคอนโวลูชั่นของสัญญาณความผิดพลาด $e_i(n)$ กับผลตอบสนองอิมพัลส์จากสเตทถึงเอาท์พุท $g_i(n)$ ดังนี้

$$y_i(n) = e_i(n) * g_i(n) = \sum_{l=0}^{\infty} e_i(l)\, g_i(n-l) \tag{2.110}$$

พิจารณาค่าเฉลี่ยและความแปรปรวนของ $y_i(n)$ เนื่องจาก $e_i(n)$ เป็นสัญญาณรบกวนขาวที่มีค่าเฉลี่ยเป็นศูนย์ ดังนั้นจะได้

$$E\left[\, y_i(n) \,\right] = 0 \tag{2.111}$$

$$E\left[\, y_i^2(n) \,\right] = E\left[\sum_l e_i(l)\, g_i(n-l) \sum_m e_i(m)\, g_i(n-m) \right]$$

$$= \sum_l \sum_m g_i(n-l)\, E\left[\, e_i(l)\, e_i(m) \,\right] g_i(n-m)$$

และกำหนดให้ $\sigma_e^2 = E\left[\, e_i^2(n) \,\right] = vriance\left[\, e_i(n) \,\right]$

$$E\left[\, y_i^2(n) \,\right] = \sum_l \sum_m g_i(n-l)\, \sigma_e^2\, \delta_{lm}\, g_i(n-m)$$

$$= \sigma_e^2 \sum_l g_i^2(n-l) = \sigma_e^2 \sum_n g_i^2(n) \tag{2.112}$$

จากนั้นทำการขยายเมตริกซ์ W จากสมการที่ (2.98) โดยจะสังเกตได้ว่าส่วนประกอบในแนวทแยงทั้งหมดจะอยู่ในรูป $\sum_n g_i^2(n)$

$$W = \sum_n g^T(n)\ g(n) = \sum_n \begin{bmatrix} g_1(n) \\ \cdots \\ g_N(n) \end{bmatrix} \begin{bmatrix} g_1(n) & \cdots & g_N(n) \end{bmatrix} \tag{2.113}$$

$$= \begin{bmatrix} \sum_n g_1^2(n) & \sum_n g_1(n)g_2(n) & \cdots & \sum_n g_1(n)g_N(n) \\ \sum_n g_2(n)g_1(n) & \sum_n g_2^2(n) & \cdots & \sum_n g_2(n)g_N(n) \\ \cdots & \cdots & \cdots & \cdots \\ \sum_n g_N(n)g_1(n) & \sum_n g_N(n)g_2(n) & \cdots & \sum_n g_N^2(n) \end{bmatrix} \tag{2.114}$$

โดยการใช้สมการที่ (2.112) สามารถที่จะเขียนสมการแสดงสัญญาณรบกวนที่เกิดจากการปัดเศษรวมที่เอาท์พุทในเทอมของค่าเทรซ (Trace) ของ เมตริกซ์ W ได้คือ

$$total\quad roundoff\quad noise = \sigma_e^2 \sum_{i=1}^{N} \sum_n g_i^2(n) = \sigma_e^2 \sum_{i=1}^{N} W_{ii} = \sigma_e^2\ Trace(W) \tag{2.115}$$

สมการที่ (2.115) สามารถที่จะใช้ในการคำนวณสัญญาณรบกวนที่เกิดจากการปัดเศษรวมสำหรับระบบที่ผ่านการสเกลแล้ว ในเทอมของค่าเทรซ ของ เมตริกซ์ W_s ที่ทำการสเกลแล้ว ได้คือ

$$total\quad roundoff\quad noise(scaled\ system) = \sigma_e^2\ Trace(W_s) \tag{2.116}$$

แทนค่าพารามิเตอร์ต่างๆที่ผ่านการสเกลแล้วลงในสมการที่ (2.98) จะได้

$$W_s = T^T W T \tag{2.117}$$

ดังนั้นสามารถแสดงในเทอมของค่าในแนวทแยงของเมตริกซ์ T ได้คือ

$$Trace(W_s) = \sum_{i=1}^{N} (W_s)_{ii} = \sum_{i=1}^{N} (t_{ii}^2\ W_{ii}) \tag{2.118}$$

จาก $t_{ii} = \sqrt{K_{ii}}$ ดังนั้นสมการที่ (2.118) สามารถแสดงได้ดังสมการที่ (2.119)

$$Trace\,(\,\boldsymbol{W}_s\,) = \sum_{i=1}^{N} (\,K_{ii}\,W_{ii}\,) \tag{2.119}$$

ดังนั้น สัญญาณรบกวนที่เกิดจากการปัดเศษรวมสำหรับระบบที่ทำการสเกลแล้ว คือ

$$total \quad roundoff \quad noise(scaled\ system) = \sigma_e^2 \sum_{i=1}^{N} (\,K_{ii} W_{ii}\,) \tag{2.120}$$

เช่นการคำนวณหาค่าสัญญาณรบกวนจากการปัดเศษที่เอาท์พุทของวงจรกรองที่ผ่านการสเกลด้วย $L_2\ norm$ ดังในรูปที่ 2.22 โดยเมตริกซ์ W สามารถคำนวณโดยใช้สมการที่ (2.99) ดังนี้

$$\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{16} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ \frac{1}{16} & 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{255} & \frac{-8}{255} \\ \frac{-8}{255} & \frac{64}{255} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{256} W_{22} + \frac{1}{255} & \frac{1}{16} W_{21} - \frac{8}{255} \\ \frac{1}{16} W_{12} - \frac{8}{255} & W_{11} + \frac{64}{255} \end{bmatrix}$$

ดังนั้น

$$\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} 0.0049 & -0.033 \\ -0.0332 & 0.2559 \end{bmatrix}$$

สัญญาณรบกวนจากการปัดเศษที่เอาท์พุทรวมสำหรับวงจรกรองที่ผ่านการสเกลแล้ว คือ

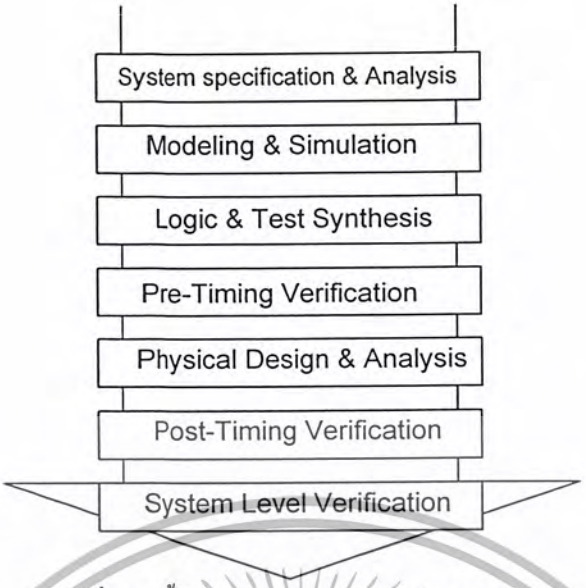
$$(\,W_{11} + W_{22}\,)\,\sigma_e^2 = 0.2608\,\sigma_e^2$$

บทที่ 3
การออกแบบสร้างวงจรกรองสัญญาณเชิงเลขด้วยภาษา VHDL

การออกแบบวงจรเชิงเลข (Digital Circuit) นั้น ในปัจจุบันก้าวหน้าไปอย่างมาก โดยการใช้ภาษาบรรยายการทำงานของวงจร (Hardware Description Language : HDL) ซึ่งเป็นภาษาที่ใช้สำหรับออกแบบฮาร์ดแวร์ โดยภาษาที่เป็นมาตรฐานสากลเช่น Verilog หรือ VHDL (VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit)) หรือภาษาที่ไม่เป็นมาตรฐานเช่น AHDL (Altera Hardware Description Language) หรือ PHDL (Philips Hardware Description Language) เป็นต้น มาบรรยายการทำงานของวงจรที่ได้ออกแบบไว้ ซึ่งในปริญญานิพนธ์นี้ ได้ใช้ภาษา VHDL มาทำการออกแบบวงจรกรองสัญญาณเชิงเลข (Digital Filter) ทำให้ลดความยุ่งยากในการนำเอาอุปกรณ์มาเชื่อมต่อให้เป็นวงจร รวมทั้งลดเวลาที่ใช้ในการออกแบบและทดสอบการทำงาน ซึ่งมีความแตกต่างเป็นอย่างมากเมื่อเปรียบเทียบกับการออกแบบในอดีตที่ผ่านมา คือผู้ออกแบบจะต้องนำเอาอุปกรณ์แต่ละตัวที่ทำการออกแบบไว้ มาทำการต่อทดลองในแผงวงจรจริง และทำการทดสอบวงจรเพื่อหาข้อผิดพลาด ซึ่งต้องใช้เวลานานกับการแก้ปัญหาแต่ละอย่างที่เกิดขึ้น แต่ในการออกแบบด้วยภาษา VHDL ผู้ออกแบบเพียงแต่เขียนซอสโค๊ด (Source Code) บรรยายการทำงานของวงจร หลังจากนั้นก็ทำการคอมไพล์ (Compile) แล้วจำลองการทำงาน (Simulate) ดูว่าได้ฟังก์ชั่นการทำงานและไทม์มิ่ง (Timing) ตามที่ต้องการหรือไม่ จากนั้นก็นำซอสโค๊ดที่ได้ไปทำการสังเคราะห์ด้วยโปรแกรมสังเคราะห์ (Synthesis Tool) สุดท้ายนำวงจรที่ได้จากการสังเคราะห์ไปทำการแม็ป (Map) ลงไปยัง FPGA (Field Programmable Gate Array) เพื่อเป็นชิป (Chip) ต้นแบบสำหรับการนำไปทดสอบการทำงาน

3.1 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมเชิงเลขขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของบล็อกไดอะแกรมก่อน จากนั้นจึงวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายการทำงานของแต่ละบล็อก และวิเคราะห์การทำงาน แก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์ เพื่อให้ได้การทำงานตามที่ต้องการ โดยการออกแบบในลักษณะนี้เรียกว่า หลักการออกแบบจากบนลงล่าง (Top-Down Design) ซึ่งถ้าเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาในการออกแบบมากกว่า เพราะเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ จำลองการทำงาน ตรวจสอบความถูกต้อง ซึ่งใช้เวลามาก และถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากในการออกแบบลักษณะนี้ ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นวิธีการที่เหมาะสมสำหรับการออกแบบและพัฒนาวงจรที่มีความซับซ้อนมากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ



รูปที่ 3.1 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 3.1 แสดงให้เห็นถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย โดยขั้นตอนของการออกแบบจากบนลงล่างมีรายละเอียด ดังนี้

1. ขั้นตอนการสร้างข้อกำหนดของความต้องการและวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา
2. ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด
3. ขั้นตอนการสังเคราะห์ ซึ่งจะต้องทำการกำหนดเทคโนโลยีที่จะมารองรับวงจรที่ออกแบบและระบบช่วยออกแบบ จะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท (Gate Level) และ การเชื่อมต่อกันของอุปกรณ์เหล่านั้น หรือไม่ก็อยู่ในรูปของเน็ทลิสต์ (Net List) ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้
4. หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือเน็ทลิสต์แล้ว ข้อมูลที่ได้นอกจากจะ เป็นข้อมูลสำหรับจำลองการทำงานในเรื่องของความถูกต้องของฟังก์ชั่นแล้ว ยังมีข้อมูลที่ เกี่ยวกับเวลาด้วย ซึ่งจากความจริงที่ว่า อุปกรณ์อิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของ การเคลื่อนผ่าน (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับนาโนวินาที แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชั่นต่างๆจำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจจะทำให้การทำงานของวงจรทั้งหมดผิดไปหรือไม่สามารถทำงานในย่านความถี่ สัญญาณนาฬิกาสูงๆได้
5. ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and Device Mapping) โดยนำข้อมูลที่ได้ จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของอุปกรณ์ FPGA หรือวงจรรวม ASIC

6. หลังจากที่ได้วงจรจริงมาแล้วยังต้องมีความจำเป็นที่จะต้องตรวจสอบการทำงานที่คำนึงถึง เวลาด้วยเพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆให้ เป็นระบบ เพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วยอินพุทและเอาท์พุทแพด (Pad) ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก
7. หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบแล้วนั้น จะต้องทดสอบ การทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆอีกครั้ง ซึ่งเป็นการทดสอบการทำงานจริงขั้น สุดท้าย

3.2 ภาษา VHDL และ ส่วนประกอบต่างๆของภาษา

วิวัฒนาการของภาษา VHDL นั้นเริ่มต้นประมาณปี ค.ศ. 1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกา หรือ DOD (Department of Defense) ได้ทำการพัฒนาโครงการที่มีชื่อว่า VHSIC ซึ่งเป็นการพัฒนาโปรแกรมซึ่งจัดเป็นภาษาระดับสูงเช่นเดียวกับภาษา C หรือ Pascal แต่สามารถบรรยายพฤติกรรมการทำงานของวงจรเชิงเลข หรือโครงสร้างของวงจรได้ ทั้งนี้เพื่อให้สามารถออกแบบและสร้างวงจรรวมได้รวดเร็วขึ้น

ในการเขียนรูปแบบบรรยายระบบเชิงเลขในลักษณะของการออกแบบจากบนลงล่าง จะต้องทำความเข้าใจในเรื่องของโครงสร้างและส่วนประกอบต่างๆของรูปแบบภาษา VHDL เสียก่อน ซึ่งส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียนมี 4 หน่วย คือ

- หน่วยการออกแบบเอนทิตี (Entity Design Unit)
- หน่วยการออกแบบสถาปัตยกรรม (Architecture Design Unit)
- หน่วยการออกแบบแพ็กเก็ต (Package Design Unit)
- หน่วยการออกแบบโครงแบบ (Configuration Design Unit)

3.2.1 หน่วยการออกแบบเอนทิตี

หน่วยการออกแบบนี้เป็นส่วนที่ใช้สำหรับติดต่อระหว่างภายนอกกับรูปแบบที่เขียนขึ้น โดยเป็นการกำหนดจุดเชื่อมต่อของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น รูปที่ 3.2 แสดงให้เห็นถึงโครงสร้างของหน่วยการออกแบบเอนทิตี

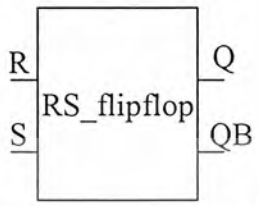
```
Entity component_name is
    Input and Output ports
    Physical and other parameters
End [component_name];
```

รูปที่ 3.2 โครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี

ส่วนนี้จะขึ้นต้นด้วยคำว่า Entity และ is ระหว่างคำทั้งสองคำเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component_name) หลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและออกของข้อมูล (input-output) รวมทั้งพารามิเตอร์อื่นๆ และที่สำคัญคือ หน่วยการออกแบบเอนทิตีจะต้องปิดท้ายด้วยคำว่า End และเครื่องหมายอัฒภาคเสมอ (;)

```
Entity  RS_ff is
        port(S,R : in bit;
             Q,QB : out bit);
End RS_ff;
```

(a) หน่วยการออกแบบเอนทิตีในรูปของภาษา VHDL



(b) มุมมองของตัวเชื่อมประสาน (Interfacing)

รูปที่ 3.3 รูปแบบของ RS_flipflop

ในรูปที่ 3.3 เป็นหน่วยการออกแบบเอนทิตีที่บรรยายอุปกรณ์ชื่อ RS_flipflop ในส่วนหัวของเอนทิตีมีการกำหนดจุดต่อ 4 จุด ภายใต้ชุดคำสั่ง port โดยที่ 2 จุดแรกเป็นจุดให้ข้อมูลไหลผ่านเข้า ได้แก่ R, S ซึ่งกำหนดด้วยทิศทางการติดต่อกับโลกภายนอกเป็นการไหลเข้าของข้อมูล (in) ส่วยจุดเอาท์พุทเป็นจุดให้ข้อมูลไหลออก ได้แก่ Q,QB ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลออก (out) ส่วนประเภทของข้อมูลที่จะไหลเข้าและออกนั้นเป็นประเภท bit ที่สามารถมีค่าได้เพียงสองค่าเท่านั้น คือ “0” และ “1” เท่านั้น

3.2.2 หน่วยการออกแบบสถาปัตยกรรม

หน่วยการออกแบบสถาปัตยกรรม คือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบในมุมมองของการจำลองการทำงาน พฤติกรรมต่างๆที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออกตรงช่องทางตลอดจนพารามิเตอร์ต่างๆที่กำหนดในหน่วยการออกแบบเอนทิตี รูปที่ 3.4 แสดงให้เห็นถึงโครงสร้างของหน่วยการออกแบบสถาปัตยกรรม

```
Architecture identifier of component_name is
        [declaration]
    Begin
        Specification of the functionality
        of the component in terms of its
        input lines and as influenced by
        physical and other parameters
End [identifier];
```

รูปที่ 3.4 โครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม

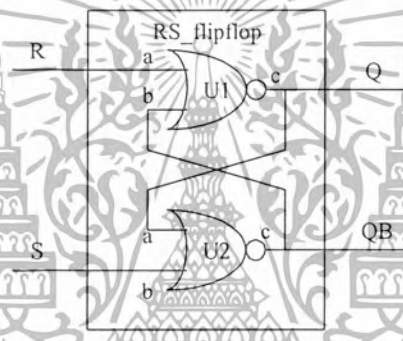
ส่วนของหน่วยการออกแบบสถาปัตยกรรมเริ่มต้นด้วยคำว่า Architecture และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า Architecture นั้นใช้บรรยายหน่วยการออกแบบเอนทิตีใดๆ (of <entity design unit> is) ส่วนที่อยู่ระหว่าง Architecture และ Begin เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (Architecture declaration area) ที่เป็นส่วนเผื่อเลือก (Option) ในบริเวณนี้สามารถใช้เขียนประกาสกำหนดค่าต่างๆที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ อาทิเช่น ประเภท (Type) ต่างๆ (ตัวอย่างเช่น bit, bit_vector), สัญญาณ (signal), ค่าคงที่ (constant), โปรแกรมย่อย (ได้แก่ function และ procedure) และอุปกรณ์ (component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้าและไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง port) นั้น จะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า Begin กับ End ของหน่วยการออกแบบสถาปัตยกรรม และนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (Concurrent statement) เท่านั้น คือทุกๆ statement จะทำงานพร้อมกัน ลำดับก่อนหลังจะไม่มีผลต่อการทำงานของรูปแบบ หน่วยการออกแบบสถาปัตยกรรมจะต้องปิดท้ายด้วยคำสั่ง End และชื่อของสถาปัตยกรรมนั้นๆ โดยทั่วไปการเขียนรูปแบบระบบเชิงเลขด้วยภาษา VHDL สามารถเขียนได้ในลักษณะต่างๆ ดังนี้

- ลักษณะการไหลของข้อมูล (Dataflow style)
- ลักษณะพฤติกรรม (Behavioral style)
- ลักษณะโครงสร้าง (Structural style)
- ลักษณะผสม (Mixed Model style)

```
Architecture dataflow of RS_ff is
        Begin
                Q  <=  not(QB or R);
                QB <= not(Q or S);
End dataflow;
```

รูปที่ 3.5 หน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop
ตามฟังก์ชั่นบูลีน $Q = \overline{QB + R}$ และ $QB = \overline{Q + S}$

รูปที่ 3.5 ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (R, S) กับข้อมูลที่ไหลออก (Q, QB) ประกอบด้วยชุดคำสั่งแบบแข่งขนาน 2 ชุด ซึ่งเขียนเป็นประเภทการไหลของข้อมูล หรือเรียกว่า ระดับการถ่ายโอนข้อมูลระหว่างรีจิสเตอร์ (RTL : Register Transfer Level)



รูปที่ 3.6 โครงสร้างภายในสถาปัตยกรรมของ RS_flipflop

รูปที่ 3.7 เป็นหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะโครงสร้าง ซึ่งเปรียบเสมือนการนำอุปกรณ์ที่มีอยู่ในไลบรารี่ (Library) มาต่อเป็นวงจรตามต้องการ โดยใช้นอร์เกท 2 อินพุท (nor2) จำนวนสองตัวมาสร้างตามฟังก์ชั่นบูลีนของรูปที่ 3.6

```
Architecture struc of RS_ff is
        component nor2
            port(a,b : in bit;
                 c :out bit);
        end component;

    Begin
        U1 : nor2 port map(R,QB,Q);
        U2 : nor2 port map(S,Q,QB);
End struc;
```

รูปที่ 3.7 หน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะโครงสร้าง

```
Architecture behave of RS_ff is
    Begin
        process(R,S)
            begin
                if R='0' and S='1' then
                    Q  <= '1';
                    QB <= '0';
                elsif R='1' and S='0' then
                    Q  <= '1';
                    QB <= '0';
                end if;
        end process;
End behave;
```

รูปที่ 3.8 หน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะพฤติกรรม

รูปที่ 3.8 เป็นการเขียนบรรยายการทำงานของรูปแบบในลักษณะพฤติกรรม ซึ่งจะเห็นได้ว่ามีลักษณะที่เหมือนกับการเขียนโปรแกรมทั่วไป โดยจะต้องมีการใช้งานส่วนที่เรียกว่า process และการทำงานของรูปแบบจะขึ้นอยู่กับการเปลี่ยนแปลงของสิ่งที่อยู่ภายใน process (อินพุท R, S) ซึ่งเรียกว่า

Sensitivity list การเขียนในลักษณะนี้ลำดับก่อนหลังของชุดคำสั่งจะมีผลต่อการทำงานของรูปแบบที่เขียนขึ้น

```
Architecture mixed of RS_ff is
        component nor2
                port(a,b : in bit;
                        c : out bit);
        end component;

Begin
        U1 : nor2 port map(R,QB,Q);
        QB <= not(Q or S);
End mixed;
```

รูปที่ 3.9 หน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะผสม

ไม่ว่าจะเขียนบรรยายส่วนของสถาปัตยกรรมของ RS_flipflop ในลักษณะของพฤติกรรม การไหลของข้อมูล โครงสร้าง หรือผสมที่นำเอาแต่ละลักษณะมาเขียนไว้ในส่วนของสถาปัตยกรรมก็ตาม ต่างก็มีพฤติกรรมเดียวกัน และจะให้ผลลัพธ์จากการจำลองการทำงานที่เหมือนกัน ซึ่งถือว่าเป็นข้อดีของภาษา VHDL

3.2.3 หน่วยการออกแบบแพ็กเก็ต

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อยที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบเชิงเลขสามารถเก็บไว้ในส่วนของแพ็กเก็ตได้ และข้อมูลเหล่านี้สามารถเรียกไปใช้ได้โดยหน่วยการออกแบบเอนทิตี หน่วยการออกแบบสถาปัตยกรรม หรือจากหน่วยการออกแบบแพ็กเก็ตอื่นๆ โดยปกติแล้ว แพ็กเก็ตจะแบ่งออกเป็น 2 ส่วน คือ การประกาศแพ็กเก็ต (Package Declaration) และส่วนของบอดี้แพ็กเก็ต (Package Body) เนื่องจากแพ็กเก็ตถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้นการที่จะนำแพ็กเก็ตไปใช้นั้นจะต้องมีการเชื่องโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถกระทำได้ด้วยชุดคำสั่ง USE

Package Declaration

ส่วนที่มีความสำคัญที่สุดของแพ็กเก็ต (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ส่วนการประกาศแพ็กเก็ต เพราะจะเป็นส่วนที่กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็กเก็ตสำหรับนำไปใช้ภายนอกตัวของแพ็กเก็ตเอง สิ่งใดๆที่ถูกประกาศไว้ในส่วนของบอดี้แพ็กเก็ตแต่ไม่ได้ถูกประกาศไว้ใน

ส่วนการประกาศแพ็กเก็ตจะไม่สามารถถูกนำค่าและพฤติกรรมไปใช้ส่วนนอกได้ ซึ่งสามารถเปรียบเทียบได้ดับสิ่งที่ประกาศไว้ในส่วนของการประกาศเอนทิตี คือจุดเชื่อมต่อ หรือพอร์ท ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเก็ตสามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้ และยังสามารถถูกนำไปใช้จากรูปแบบภายนอกได้ เช่น ใช้สำหรับประกาศชนิด (Type) หรือสัญญาณ เช่นเดียวกันกับส่วนบอดี้แพ็กเก็ตที่ไม่จำเป็นต้องมีส่วนของการประกาศแพ็กเก็ต แต่แพ็กเก็ตนั้นจะไม่สามารถถูกนำไปใช้จากรูปแบบอื่นได้

```
Package package_name is
  Package_declaration_part
End package_name;
```

รูปที่ 3.10 โครงสร้างโดยทั่วไปของส่วนการประกาศแพ็กเก็ต

Package body

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆในรูปของคำสั่งลำดับ (Sequence) ที่ใช้บรรยายฟังก์ชั่นการทำงานของโปรแกรมย่อย (Subprogram) ทั้งหลาย ที่ชื่อของโปรแกรมย่อยนั้นๆที่ถูกประกาศไปในส่วนของการประกาศแพ็กเก็ตแล้วจะถูกเก็บไว้ในส่วนบอดี้แพ็กเก็ต ทั้งนี้รวมทั้งการกำหนดค่าคงที่ต่างๆ อันได้แก่ค่าคงที่ที่ถูกประกาศชื่อก่อนในส่วนของการประกาศแพ็กเก็ต แต่ถูกกำหนดค่าในส่วนของบอดี้แพ็กเก็ต ฉะนั้นส่วนบอดี้แพ็กเก็ตจึงไม่จำเป็นต้องมีถ้าในส่วนของการประกาศแพ็กเก็ตไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อยหรือค่าคงที่ การเขียนบอดี้แพ็กเก็ตนั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 3.11

```
Package body package_name is
  declarative part
End package_name;
```

รูปที่ 3.11 โครงสร้างโดยทั่วไปของบอดี้แพ็กเก็ต

3.2.4 หน่วยการออกแบบโครงแบบ

ดังที่ทราบกันแล้วว่ารูปแบบหนึ่งของระบบเชิงเลขไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนทิตีได้เพียงหนึ่งเดียวเท่านั้น แต่หน่วยการออกแบบเอนทิตีหนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมีหน่วยการออกแบบโครงแบบมาเพื่อกำหนดการใช้โครงแบบ (Configuration) ประกอบเอนทิตีกับหน่วยการออกแบบสถาปัตยกรรมหน่วยไหนเข้าด้วยกัน

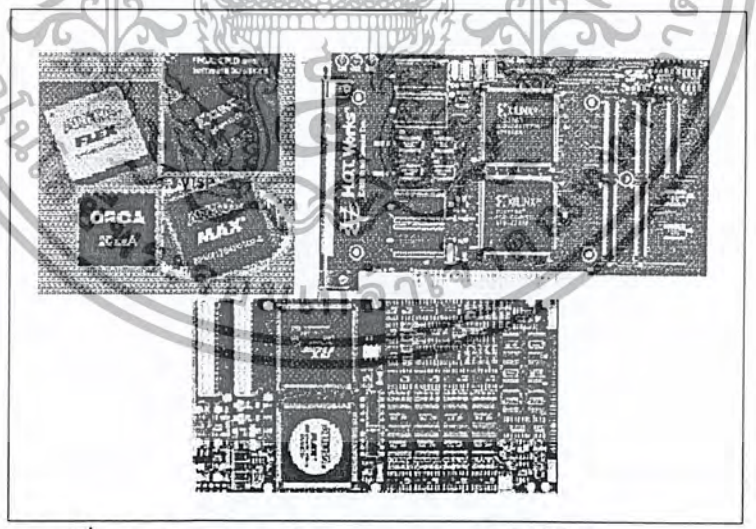
```
Configuration identifier of entity_name is
        Configuration_declarative_part
End;
```

รูปที่ 3.12 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

3.3 การออกแบบวงจรเชิงเลขด้วยอุปกรณ์ FPGA

อุปกรณ์ FPGA (Field Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการทำงานตามที่ออกแบบไว้ ในการทำ FPGA ซึ่งเป็นวิธีการออกแบบ IC (Integrated Circuit) แบบ Semicustom อีกวิธีหนึ่ง เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuits) แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือ การทำ FPGA จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในอุปกรณ์ FPGA จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัด และการทำ FPGA ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำ ASIC มากและการตรวจสอบหรือแก้ไขการออกแบบก็ทำได้สะดวก

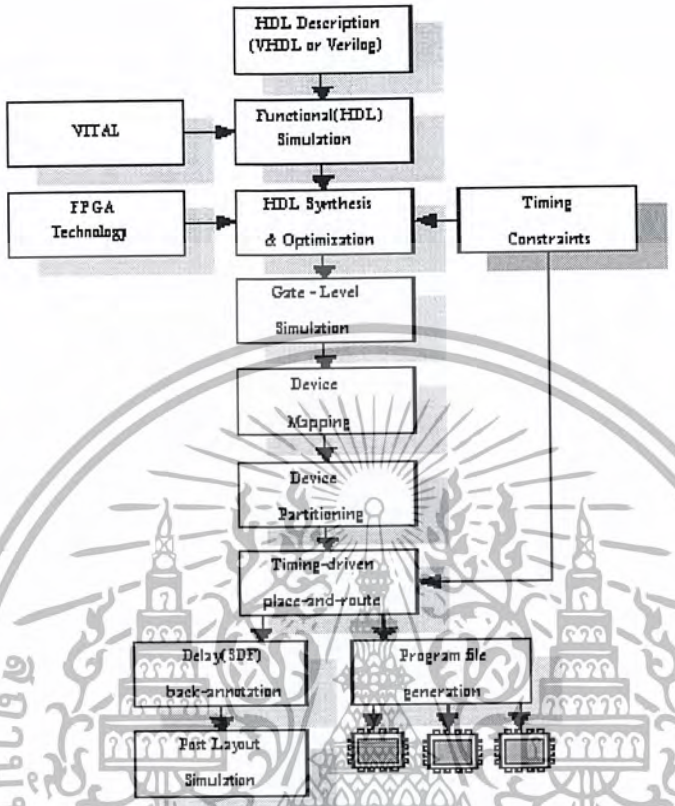
การทำ FPGA ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตอุปกรณ์ FPGA ได้เพิ่มความสามารถของอุปกรณ์ FPGA โดยเพิ่มจำนวนองค์ประกอบภายใน หรือปรับปรุงโครงสร้างสถาปัตยกรรมภายในและยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ PPR (Partitioning, Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย ลักษณะของตัวFPGAและการนำไปใช้งานแสดงดังในรูปที่ 3.13



รูปที่ 3.13 ลักษณะของตัว FPGA และการนำไปใช้งาน

สำหรับตัวอุปกรณ์ FPGA นั้นก็มีโครงสร้างพื้นฐาน เทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นอุปกรณ์ FPGA ของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นอุปกรณ์ FPGA สามารถนำไปประยุกต์ใช้งาน

ได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP : Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์ เป็นต้น โดยมีขั้นตอนในการออกแบบ ดังแสดงในรูปที่ 3.14



รูปที่ 3.14 ขั้นตอนการออกแบบโดยใช้อุปกรณ์ FPGA

3.3.1 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์

ในการออกแบบวงจรเชิงเลขนั้นทำได้โดยการวาดวงจร หรือใช้ภาษาอธิบายฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASIC ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะแตกต่างกันโดยที่การทำวิธีนี้จะต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นได้ว่าการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ทำได้สะดวกกว่า เพราะการทำด้วยวิธีนี้ไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ (Technology Independence) และที่สำคัญการออกแบบด้วยวิธีนี้สามารถที่จะแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่าเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

ในการเขียนโค้ด สิ่งที่ต้องคำนึงถึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และให้คุณสมบัติของวงจรตามที่กำหนด เพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับวงจรที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้นซอร์ฟแวร์สังเคราะห์วงจร (Synthesis Tools) จะทำการสังเคราะห์ตามโค้ดที่เขียน ถ้าอธิบายการทำงานของวงจรเดียวกัน แต่เขียนโค้ดในลักษณะที่ต่างกันเมื่อสังเคราะห์แล้วจะได้วงจรที่

ต่างกัน และจากวงจรที่ต่างกัน เมื่อนำไปทำต้นแบบด้วย FPGA หรือการทำ ASIC แล้วจะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านของขนาดหรือความเร็ว (area and time) ส่วนการเขียนโค้ดลักษณะใดเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ในการออกแบบ

3.3.2 การจำลองการทำงานของวงจร (Simulation)

ขั้นตอนนี้เป็นขั้นตอนที่สำคัญเพราะเป็นขั้นตอนที่ใช้ตรวจสอบฟังก์ชั่นการทำงานของวงจรว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหน เพื่อที่จะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สำหรับทำการจำลองการทำงานของวงจร เช่น V-System และ ModelSim ของบริษัท Model Technology

3.3.3 การสังเคราะห์วงจร

ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สังเคราะห์วงจร (Synthesis tools) ทำการสังเคราะห์โค้ดเพื่อให้ได้เป็นวงจรขึ้นมา แต่ต้องตรวจสอบด้วยว่าซอร์ฟแวร์นั้นๆสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการใช้หรือไม่ โดย FPGA ที่นิยมใช้งานเช่นของบริษัท Xilinx ตระกูล XC4000 และบริษัท Altera ตระกูล FLEX 10 K ซอร์ฟแวร์สังเคราะห์วงจรที่นิยมใช้เช่นโปรแกรม Leonardo Spectrum ของบริษัท Exemplar Logic ซึ่งในขั้นตอนนี้ซอร์ฟแวร์สังเคราะห์วงจรจะแปลงโค้ดและทำการออปติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ นอกจากนี้ยังสามารถกำหนดข้อบังคับสำหรับวงจรได้เช่น ข้อบังคับในเรื่องของเวลา (time constraints) หรือข้อบังคับในเรื่องของพื้นที่ ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอนออปติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการออปติไมซ์คือการเทียบ (mapping) วงจรให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์ FPGA ในกรณีของ Xilinx ตระกูล XC4000 และ Altera ตระกูล FLEX 10 K จะเทียบโดยใช้วิธี LUT (Look Up Table) เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอร์ฟแวร์สังเคราะห์วงจรก็จะมีการรายงานผลว่าวงจรที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วง (delay) เท่าไร ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เป็นต้น

3.3.4 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ให้เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอร์ฟแวร์ทำ โดยซอร์ฟแวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิปฟลอป (flipflop) ลงในทรัพยากรต่างๆ ที่มีอยู่ภายในอุปกรณ์ FPGA (CLBs, IOBs, BUFT และ edge decoder) หลังจากทำขั้นตอนนี้เสร็จแล้วสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนซอร์ฟแวร์ที่ใช้ในขั้นตอนนี้ขึ้นอยู่กับตัว FPGA ที่ใช้งาน เช่น FPGA ของบริษัท Xilinx จะใช้ Xilinx Foundation Series 2.1i ซึ่งซอร์ฟแวร์ตัวนี้จะรวมเอาซอร์ฟแวร์ย่อยอื่นๆอีก เพื่อให้การทำ PPR (Partitioning, Placement and Routing) เป็นไปอย่างต่อเนื่อง ส่วน FPGA ของบริษัท Altera จะใช้ Altera MAX+II

3.3.5 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (partitioning) มาแล้วว่าควรจะอยู่ในตำแหน่งใดในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (route) ได้ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือตัว Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

3.3.6 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA เช่นระหว่าง CLBs หรือระหว่าง CLBs กับ IOBs ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอร์ฟแวร์ ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมดหรือเกิดความหน่วงเกินค่าที่กำหนดในข้อบังคับ โดยสามารถทำขั้นตอนนี้ได้โดยใช้ซอร์ฟแวร์ เช่นกัน หรือจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (manual layout) ก็ได้ แต่ทางที่ดีควรใช้ซอร์ฟแวร์ทำดีกว่าโดยให้ทำการค้นหาเส้นทางหลายๆครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา (time constraints) จะช่วยให้ผลที่ได้จากการทำการเชื่อมต่อสัญญาณดีขึ้นได้

3.3.7 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่วงจรผ่านขั้นตอนต่างๆจนกระทั่งผ่านการทำ PPR (Partitioning, Placement and Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้ให้เป็นข้อมูลวงจร (Configuration data) ซึ่งอยู่ในรูปของบิทสตรีม (Bit-stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการทำงานตามวงจรที่ออกแบบไว้

จากที่อธิบายมาทั้งหมดจะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้น ทำได้สะดวกกว่าการทำ ASIC มากเพราะใช้เวลาน้อยกว่ามาก ส่วนสำคัญที่ใช้ในการทำ FPGA คือซอร์ฟแวร์ที่ใช้ตั้งแต่การเขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงในอุปกรณ์ FPGA ซึ่งซอร์ฟแวร์ที่ใช้ต้องเป็นซอร์ฟแวร์ที่ใช้ทำงานต่อเนื่องกัน

3.4 สถาปัตยกรรมภายในของ FPGA

สถาปัตยกรรมภายในของ Xilinx FPGA จะมีลักษณะเป็นตารางของลอจิกบล็อก (Logic Block) และล้อมรอบไปด้วยบล็อกการเชื่อมต่อของไอโอ (I/O Interface Block) การเชื่อมต่อระหว่างซีแอลบี (CLB : Configuration Logic Block) และไอโอบี (IOB : Input Output Block) ทำได้โดยผ่านช่องว่างที่พาดผ่านระหว่างแถว (Row) และคอลัมน์ (Column) หน้าที่ของซีแอลบีและไอโอบีแต่ละตัว การเชื่อมต่อภายใน (Interconnection) จะถูกกำหนดไว้ในโปรแกรมคอนฟิกกูเรชั่น (Configuration program) โดยแสดงรายละเอียดของทั้ง 3 ส่วนประกอบใหญ่ๆ ได้ดังนี้

CLB (Configuration Logic Block)

ภายในFPGAจะเป็นเมทริกซ์ของซีแอลบี ซึ่งซีแอลบีแต่ละตัวจะประกอบด้วยหน่วยของคอมบิเนชั่นลอจิกที่สามารถโปรแกรมได้ (Programmable Combination logic) และส่วนของรีจิสเตอร์เก็บข้อมูล (Storage register) ส่วนของวงจรคอมบิเนชั่นลอจิกสามารถใช้สร้างวงจรทางด้านฟังก์ชั่น บูลีนของอินพุท ส่วนรีจิสเตอร์รับค่าจากส่วนคอมบิเนชั่นหรือโดยตรงจากเอาท์พุทของซีแอลบี สามารถขับวงจรคอมบิเนชั่นลอจิกโดยตรงผ่านเส้นทางเดินย้อนกลับ (Feedback path)

IOB (Input Output Block)

เป็นส่วนติดต่อกับวงจรภายนอกของ FPGA สร้างมาจากส่วนของอุปกรณ์อินพุท/เอาท์พุทที่สามารถโปรแกรมได้ (Programmable Input/Output device) แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้เป็นอินพุท/เอาท์พุทแบบ 3 สถานะ หรือไอโอแบบสองทิศทางก็ได้ โดยอินพุทสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอลและซีมอส เทรดโฮลของไอโอบีแต่ละตัวมีฟลิปฟลอปสามารถใช้เป็นบัฟเฟอร์สำหรับอินพุทและเอาท์พุท

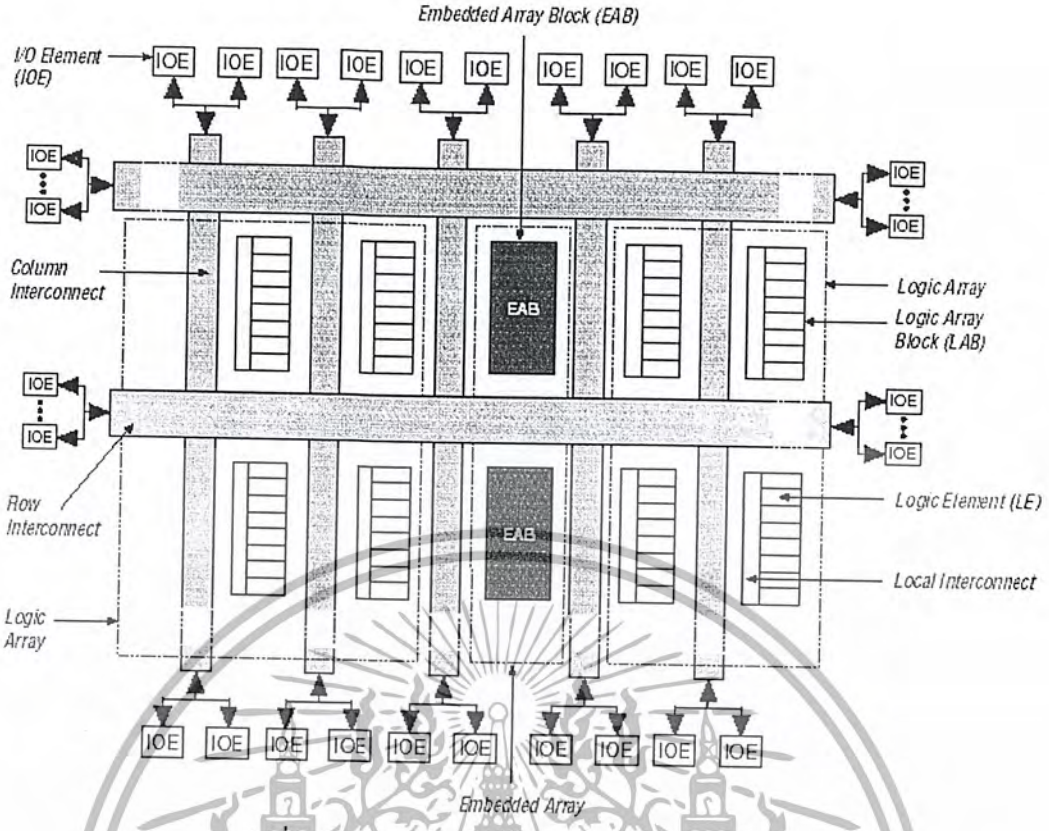
Interconnection

ความยืดหยุ่นของการใช้ FPGA มาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรมทรัพยากรต่างๆที่อยู่ภายในเข้าด้วยกัน การที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปจะประกอบไปด้วยเน็ตเวิร์ก 2 ทิศทาง คือทางแถวและคอลัมน์ ซึ่งวางอยู่ระหว่าง CLB Programmable switch จะทำการเชื่อมต่ออินพุทและเอาท์พุทของไอโอบีและซีแอลบีที่จุดต่อร่วม ระหว่างแถวกับคอลัมน์และสามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่างๆได้

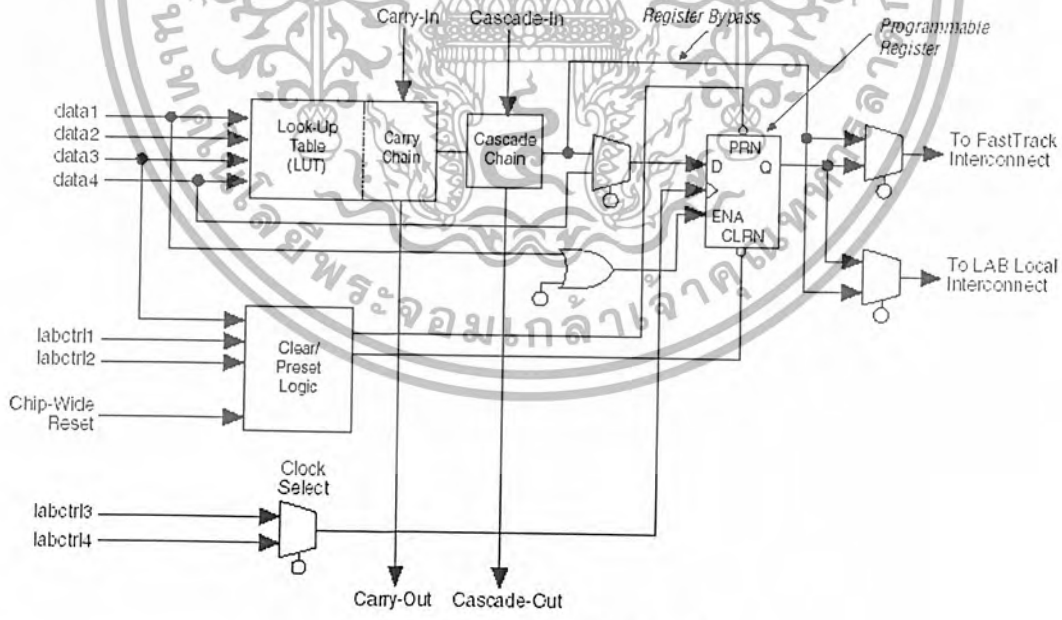
FPGA ของบริษัท Altera ตระกูล FLEX 10 K เป็นอุปกรณ์ที่มีความหนาแน่นเกทประมาณตั้งแต่ 10,000 – 250,000 เกท โดยการจัดโครงสร้าง (Configuration) จะใช้วิธีโหลดโครงสร้างเข้าไปใน SRAM ภายใน ซึ่งหมายความว่าถ้าไม่ได้มีการจ่ายไฟเลี้ยงให้ โครงสร้างที่จัดเอาไว้ก็จะหายไป FPGA ประเภทนี้จะสามารถโปรแกรมซ้ำได้ไม่จำกัดจำนวนครั้ง และการทำงานตามลอจิกฟังก์ชั่นจะใช้วิธีการเปิดตารางดู (Look Up table : LUT) โดยโครงสร้างของ FPGA ตระกูล FLEX 10 K แสดงดังรูปที่ 3.15 โดยในโครงสร้างของ FPGA ตระกูล FLEX 10 K สามารถที่จะแบ่งเป็นส่วนต่างๆ ได้ดังนี้

Logic Element (LE)

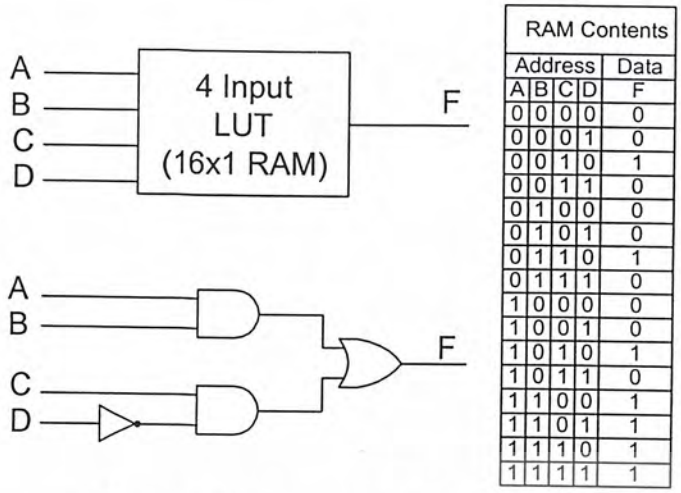
ในรูปที่ 3.16 แสดงโครงสร้างภายในของ LE โดยการกระทำทางบูลีนของลอจิกเกทจะสร้างด้วยวิธีการ LUT โดย LUT คือ 1x16 SRAM ซึ่ง LUT เพียงตัวเดียวสามารถนำมาทำโครงข่ายของลอจิกเกทที่มี 4 อินพุท และ 1 เอาท์พุท โดยโครงข่ายของลอจิกเกทจะถูกแปลงไปเป็นตารางค่าความจริง (Truth Table) ดังแสดงในรูปที่ 3.17



รูปที่ 3.15 โครงสร้างของ FPGA ตระกูล FLEX 10K



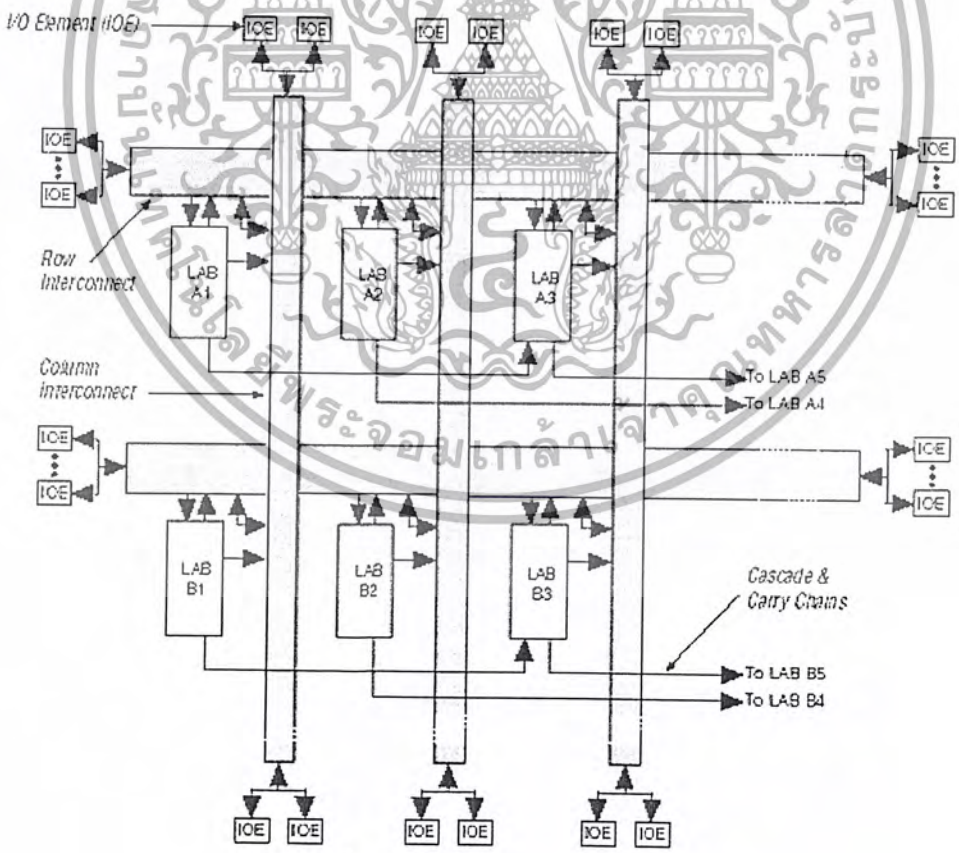
รูปที่ 3.16 โครงสร้างภายในของ LE



RAM Contents				
Address				Data
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

รูปที่ 3.17 การใช้งาน LUT เป็นโครงข่ายของลอจิก

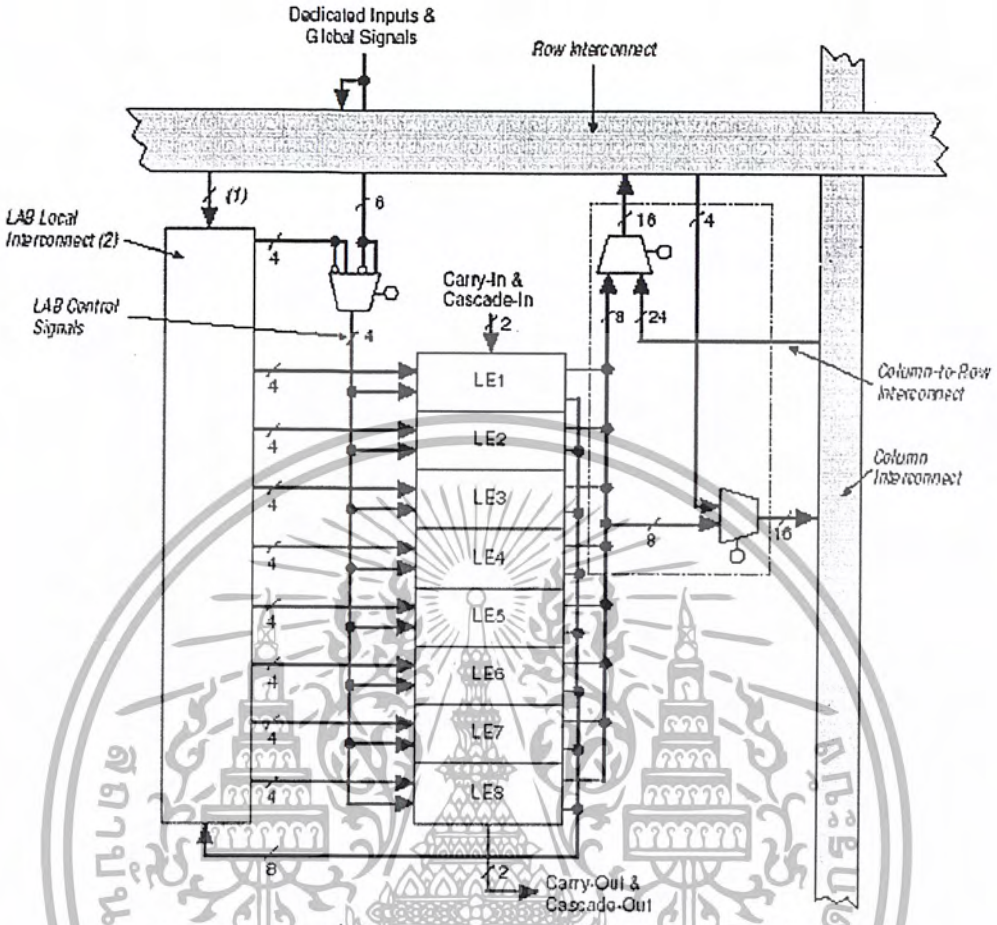
ถ้าโครงข่ายของลอจิกเกทมีความซับซ้อนขึ้นจะต้องใช้ LUT ของแต่ละ LE เป็นจำนวนหลายตัว โดยเอาท์พุทของ LUT จะส่งต่อไปยังฟลิปฟลอปและต่อไปยังโครงข่ายการเชื่อมต่อ (Interconnection Network) ดังแสดงในรูปที่ 3.18



รูปที่ 3.18 โครงข่ายของการเชื่อมต่อ

Logic Array Block (LAB)

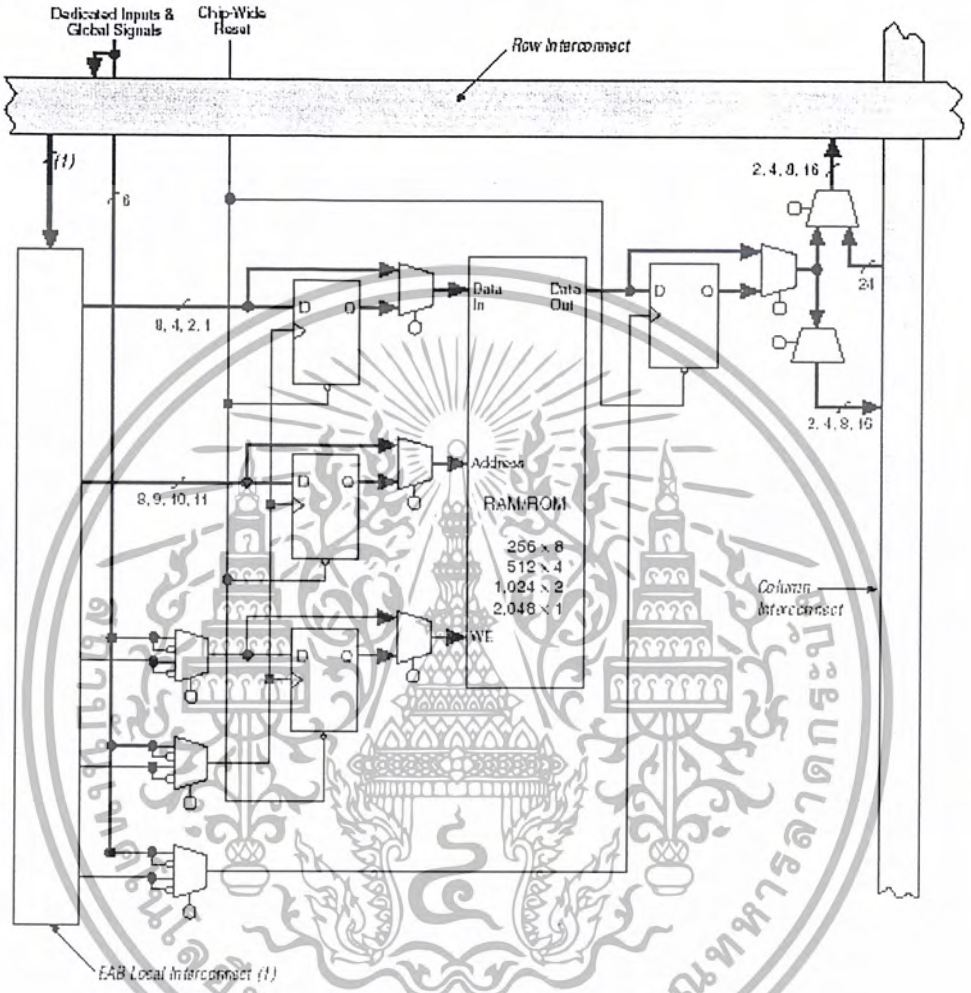
LAB 1 ตัว จะประกอบไปด้วย 8 LE ดังแสดงในรูปที่ 3.19



รูปที่ 3.19 โครงสร้างภายในของ LAB

Embedded Array Block (EAB)

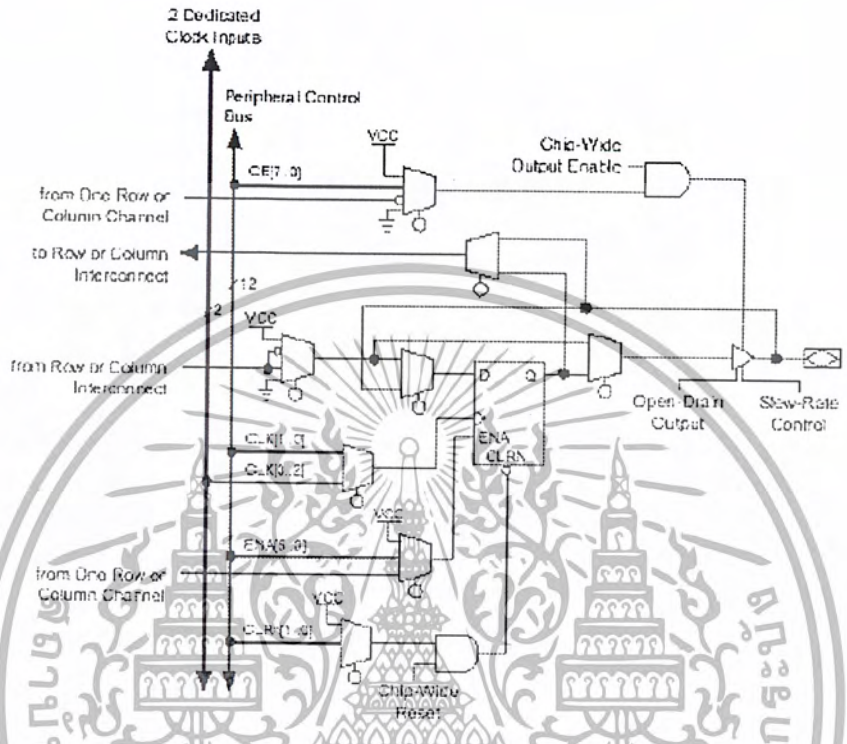
สถาปัตยกรรมโดยทั่วไปของ FLEX 10 K จะมีลักษณะของ LAB ที่มีการจัดเรียงแบบเมตริกซ์ และ EAB ซึ่งมีการเชื่อมต่อผ่านทางแถวและคอลัมน์ โดยในแต่ละแถวจะมี 1 EAB ซึ่ง 1 EAB จะมีขนาด 2048 บิท และสามารถกำหนดความกว้าง (Width) ความลึก (Depth) ของ EAB ได้โดยไม่ส่งผลต่อความเร็ว



รูปที่ 3.20 โครงสร้างภายใน EAB

Input Out Element (IOE)

IOE จะถูกต่ออยู่กับขา I/O โดยจะประกอบด้วยส่วนของวงจรที่เป็น Tri State และส่วนที่เป็นฟลิปฟลอป ซึ่งเป็น option ดังแสดงในรูปที่ 3.21



รูปที่ 3.21 โครงสร้างภายในของ IOE

บทที่ 4
การออกแบบและผลการออกแบบวงจรกรองสัญญาณเชิงเลข

4.1 การออกแบบสถาปัตยกรรมของวงจรกรองสัญญาณด้วยโครงสร้างเลขคณิตกระจาย

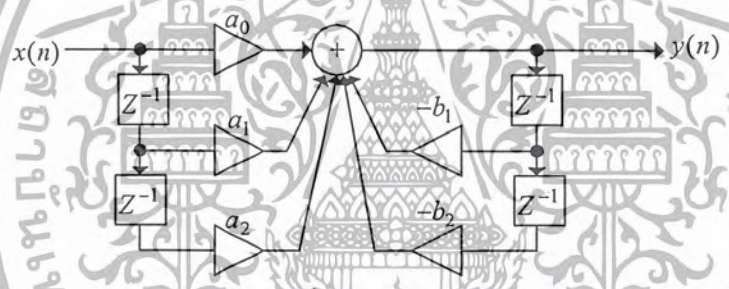
เพื่อที่จะแสดงให้เห็นถึงความแตกต่างทางสถาปัตยกรรม ของวงจรกรองสัญญาณเชิงเลขที่สร้างขึ้นโดยใช้โครงสร้างเลขคณิตกระจายแต่ใช้รูปแบบของสมการตั้งต้นที่ต่างกัน จะทำให้โครงสร้างสุดท้ายที่ได้ต่างกัน เพื่อความสะดวกจะทำการพิจารณาจากวงจรกรองสัญญาณเชิงเลขในอันดับที่ 2 โดยแยกเป็นโครงสร้าง ดังนี้

4.1.1 โครงสร้างโดยตรง 1

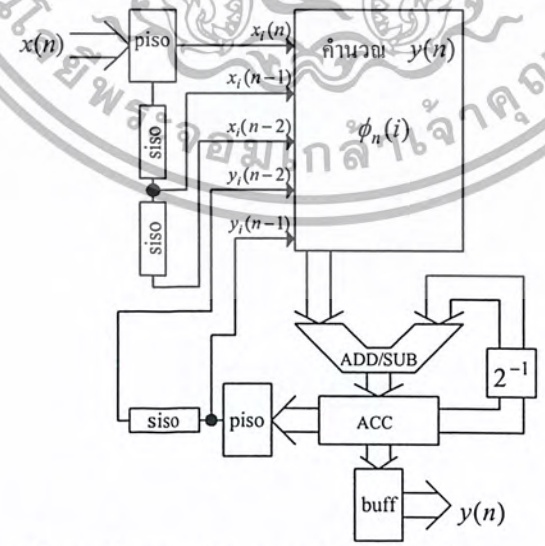
รูปแบบพื้นฐานสำหรับการใช้โครงสร้างเลขคณิตกระจาย ในการสร้างวงจรกรองสัญญาณเชิงเลขโดยทั่วไปอาศัยโครงสร้างแบบโดยตรง 1 ซึ่งจากฟังก์ชั่นถ่ายโอนในสมการที่ (2.73) สามารถแทนเป็นสมการผลต่างสืบเนื่องได้ดังนี้

$$y(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) - b_1 y(n-1) - b_2 y(n-2) \qquad (4.1)$$

โดยแสดงโครงสร้างได้ดังรูป



(a) โครงสร้างแบบโดยตรง 1



(b) โครงสร้างเลขคณิตกระจายสำหรับแบบโดยตรง 1

รูปที่ 4.1 โครงสร้างแบบโดยตรง 1 ที่แทนด้วยโครงสร้างเลขคณิตกระจาย

4.1.2 โครงสร้างปริภูมิสเตทโดยทั่วไป

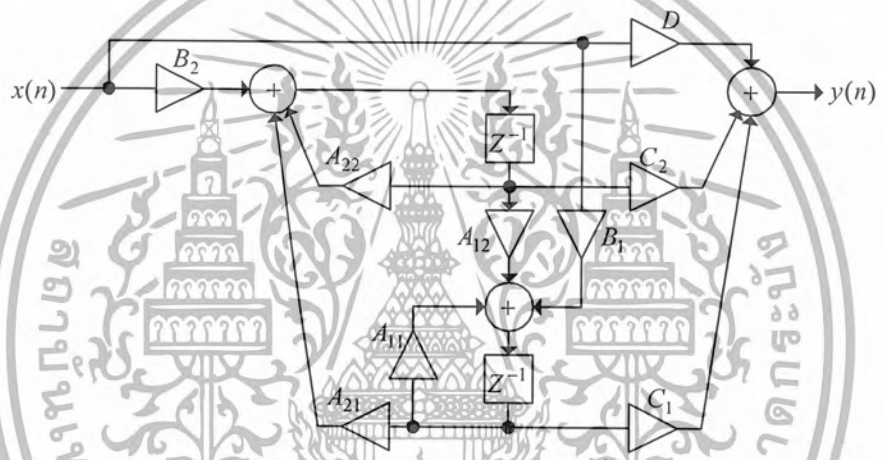
สำหรับโครงสร้างแบบปริภูมิสเตทโดยทั่วไป ซึ่งอธิบายในสมการที่ (2.71) และสมการที่ (2.72) สามารถแสดงเป็นสมการสเตทและสมการเอาท์พุทได้ดังนี้

$$q_1(n+1) = a_{11}q_1(n) + a_{12}q_2(n) + b_1x(n) \tag{4.2}$$

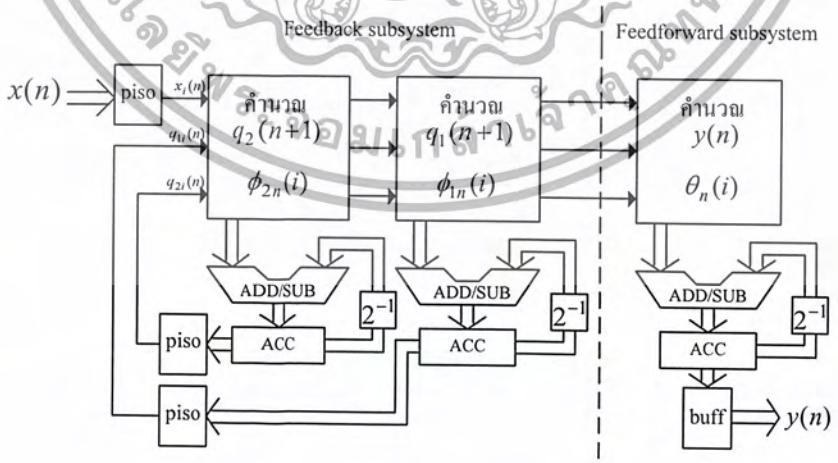
$$q_2(n+1) = a_{21}q_1(n) + a_{22}q_2(n) + b_2x(n) \tag{4.3}$$

$$y(n) = c_1q_1(n) + c_2q_2(n) + D\,x(n) \tag{4.4}$$

โดยแสดงโครงสร้างได้ดังรูป



(a) โครงสร้างแบบปริภูมิสเตทโดยทั่วไป



(b) โครงสร้างเลขคณิตกระจายสำหรับแบบปริภูมิสเตทโดยทั่วไป

รูปที่ 4.2 โครงสร้างแบบปริภูมิสเตทโดยทั่วไป ที่แทนด้วยโครงสร้างเลขคณิตกระจาย

4.1.3 โครงสร้างปริภูมิสเตทแบบที่นำเสนอ

เนื่องจากในทางปฏิบัติ โครงสร้างแบบปริภูมิสเตทสามารถที่จะทำการแปลงคล้ายได้เป็นจำนวนอนันต์แบบขึ้นอยู่กับเมตริกซ์ T ที่ใช้ในการแปลงคล้าย ในที่นี้จะนำเสนอโครงสร้างแบบปริภูมิสเตทจำนวน 2 โครงสร้าง ซึ่งเรียกว่า โครงสร้างแบบ Controllable Canonical Form และโครงสร้างแบบ Minimum Noise

โครงสร้างแบบ Controllable Canonical Form เป็นโครงสร้างแบบปริภูมิสเตทที่ใช้จำนวนตัวคูณน้อยที่สุดคือ $2(N+1)$ ตัว เมื่อ N คืออันดับของวงจรกรองสัญญาณที่ทำการออกแบบ ในขณะที่โครงสร้างแบบ Minimum Noise ต้องใช้ตัวคูณถึง $(N+1)^2$ แต่ให้ผลดีในแง่ระดับของสัญญาณรบกวนหรือ Roundoff Noise ที่เกิดขึ้นที่เอาท์พุทจะมีค่าต่ำ

ในการศึกษาจะทำการศึกษาวงจรกรองสัญญาณอันดับที่ 2 เพราะว่าวงจรกรองความถี่อันดับสูงสามารถสร้างได้จากการนำวงจรกรองสัญญาณอันดับที่ 2 มาต่อ Cascade หรือ ต่อขนานกัน พิจารณาฟังก์ชันถ่ายโอนของวงจรกรองสัญญาณอันดับที่ 2 ดังนี้

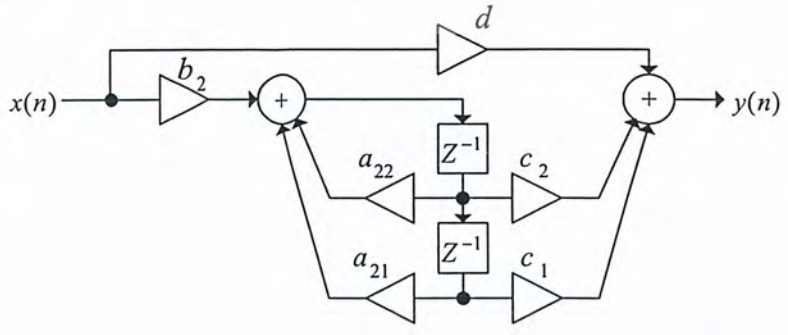
$$H(Z)=d+\frac{\alpha_1 Z^{-1}+\alpha_2 Z^{-2}}{1+\beta_1 Z^{-1}+\beta_2 Z^{-2}} \tag{4.5}$$

สามารถที่จะแทนให้อยู่ในรูปแบบปริภูมิสเตทได้เป็น

$$\begin{bmatrix} q1(n+1) \\ q2(n+1) \end{bmatrix}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} q1(n) \\ q2(n) \end{bmatrix}+\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}x(n) \tag{4.6}$$

$$y(n)=\begin{bmatrix} c_1 & c_2 \end{bmatrix}\begin{bmatrix} q1(n) \\ q2(n) \end{bmatrix}+[d]\,x(n) \tag{4.7}$$

โครงสร้างปริภูมิสเตทแบบ Controllable Canonical Form เป็นโครงสร้างที่ได้มาจากการแปลงจากฟังก์ชันถ่ายโอนโดยตรงซึ่งถือว่าเป็นโครงสร้างแบบโดยตรง (Direct Form) สำหรับโครงสร้างแบบปริภูมิสเตทและใช้จำนวนตัวคูณน้อยที่สุดคือ 6 ตัว ดังแสดงในรูปที่ 4.3



รูปที่ 4.3 โครงสร้างปริภูมิสเตทแบบ Controllable Canonical Form

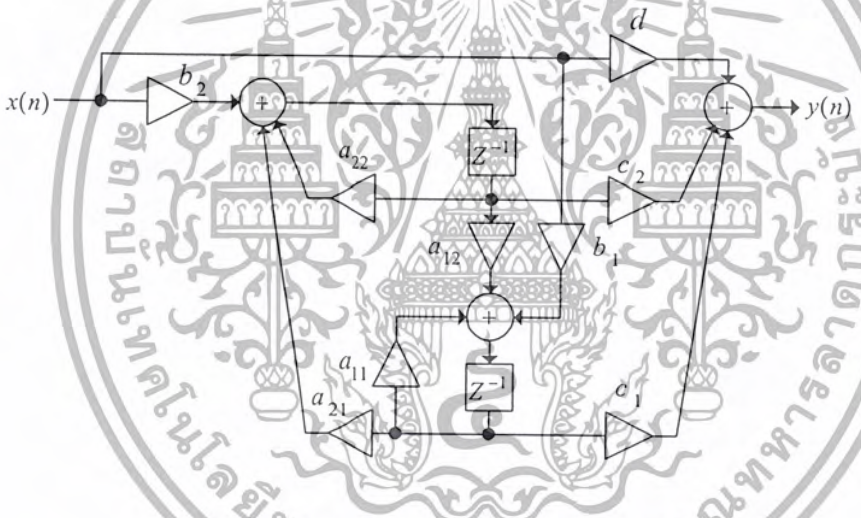
ค่าสัมประสิทธิ์ของเมตริกซ์จะใช้วิธี L_2 norm ในการสเกลลิ่ง จะได้

$$\begin{aligned} a_{11} &= 0 \qquad a_{21} = -\beta_2 \qquad b_1 = 0 \qquad c_1 = \alpha_2 / b_2 \\ a_{12} &= 1 \qquad a_{22} = -\beta_1 \qquad b_2 = \sqrt{\frac{1-\beta_2}{1+\beta_2}[(\beta_2+1)^2 - \beta_1^2]} \qquad c_2 = \alpha_1 / b_2 \end{aligned} \tag{4.8}$$

สำหรับโครงสร้างแบบ Minimum noise จะให้ค่า noise gain G_N มีค่าต่ำสุด โดยที่เงื่อนไขสำหรับ Minimum noise ในสมการอันดับ 2 จะมีค่าดังนี้

$$a_{11} = a_{12} \qquad \text{และ} \qquad b_1 c_1 = b_2 c_2 \tag{4.9}$$

โครงสร้าง Minimum noise แบบปริภูมิสเตทได้มาจากการแปลงคล้ายจากการแปลงเมตริกซ์ T ในรูปที่ 4.4 แสดงโครงสร้าง Minimum noiseโดยใช้ตัวคูณสูงสุด 9 ตัว



รูปที่ 4.4 โครงสร้างแบบ Minimum noise Structure

ค่าสัมประสิทธิ์ของเมตริกซ์จะใช้วิธี L_2 norm ในการสเกลลิ่ง จะได้

$$a_{11}=a_{22}=-\frac{\beta_1}{2} \qquad \mu=\sqrt{\left(\frac{\alpha_2}{\alpha_1}\right)^2-\frac{\alpha_2}{\alpha_1}\beta_1+\beta_2} \qquad \gamma=\frac{\alpha_2}{\alpha_1}-\mu$$

$$\xi=\frac{\alpha_2}{\alpha_1}+\mu \qquad \lambda=(\beta_2-1)\,[(\beta_2+1)^2-\beta_1^2] \qquad \epsilon=\left(\frac{\beta_1}{2}\right)^2-\beta_2 \tag{4.10}$$

$$b_1=\sqrt{\frac{\lambda}{2\beta_1\gamma-(\beta_2+1)(1+\gamma^2)}} \qquad b_2=\sqrt{\frac{\lambda}{2\beta_1\xi-(\beta_2+1)(1+\xi^2)}}$$

$$a_{21}=\sqrt{\frac{b_2^2+\beta_2-1}{b_1^2+\beta_2-1}}\epsilon \qquad a_{12}=\frac{\epsilon}{a_{21}} \qquad c_1=\frac{\alpha_1}{2b_1} \qquad c_2=\frac{\alpha_1}{2b_2}$$

4.1.4 โครงสร้างเลขคณิตกระจายของวงจรกรองสัญญาณเชิงเลขแบบปริภูมิสเตท

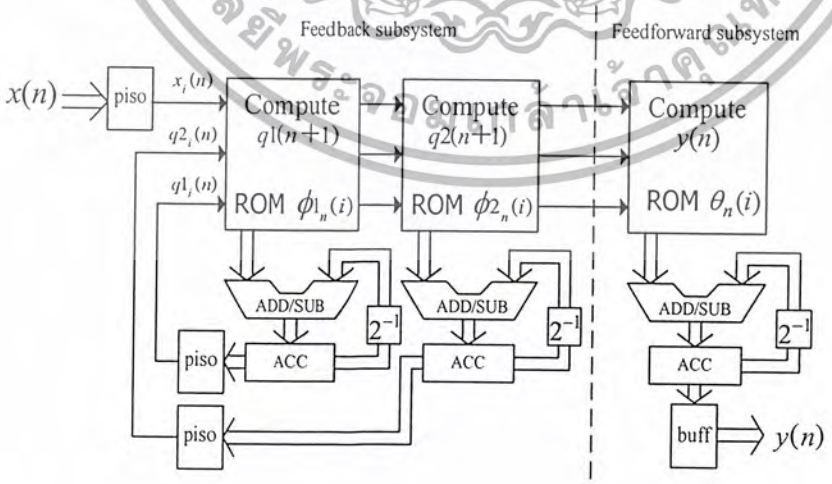
วิธีแก้ปัญหาของจำนวนตัวคูณแบบปริภูมิสเตทสามารถทำได้โดยโครงสร้างเลขคณิตกระจาย สำหรับโครงสร้าง minimum noise ซึ่งอธิบายในสมการที่ (4.6) และ (4.7) สามารถเขียนใหม่ให้อยู่ในรูปสมการสเตท 2 สมการ และสมการเอาท์พุต 1 สมการดังนี้

$$q1(n+1)=a_{11}q1(n)+a_{12}q2(n)+b_1x(n) \tag{4.11}$$

$$q2(n+1)=a_{21}q1(n)+a_{22}q2(n)+b_2x(n) \tag{4.12}$$

$$y(n)=c_1q1(n)+c_2q2(n)+d\,x(n) \tag{4.13}$$

การคำนวณแบบโครงสร้างเลขคณิตกระจายใช้หน่วยความจำ 2 ตัวคือ ROM $\phi1_n(i)$ และ ROM $\phi2_n(i)$ สำหรับทำการคำนวณค่าstate vector 2 สมการตามสมการที่(4.11) และ (4.12) และใช้ ROM $\theta_n(i)$ ทำการคำนวณสมการเอาท์พุต วงจรกรองสัญญาณแบบ minimum noiseโครงสร้างเลขคณิตกระจายแสดงดังในรูปที่ 4.5



รูปที่ 4.5 โครงสร้าง Minimum noise ที่แทนด้วยโครงสร้างเลขคณิตกระจาย

ค่าที่เก็บอยู่ใน ROM $\phi 1_n(i), \phi 2_n(i), \theta_n(i)$ สามารถหาค่าโดยใช้สมการ (4.11) - (4.13) โดยแสดงค่าต่าง ๆ ที่เก็บเป็น ระดับบิตได้ดังที่แสดง

$$\phi 1_n(i) = a_{11} q1_i(n) + a_{12} q2_i(n) + b_1 x_i(n) \tag{4.14}$$

$$\phi 2_n(i) = a_{21} q1_i(n) + a_{22} q2_i(n) + b_2 x_i(n) \tag{4.15}$$

$$\theta_n(i) = c_1 q1_i(n) + c_2 q2_i(n) + d\, x_i(n) \tag{4.16}$$

ค่า สเตทเวกเตอร์ และ สมการเอาท์พุตสามารถหาได้ดังสมการ

$$q1(n+1) = -\phi 1_n(0) + \sum_{i=1}^{L-1} \phi 1_n(i)\, 2^{-i} \tag{4.17}$$

$$q2(n+1) = -\phi 2_n(0) + \sum_{i=1}^{L-1} \phi 2_n(i)\, 2^{-i} \tag{4.18}$$

$$y(n) = -\theta_n(0) + \sum_{i=1}^{L-1} \theta_n(i)\, 2^{-i} \tag{4.19}$$

ค่า L บิทจะมีค่าที่แม่นยำ โดยที่กำนดให้ทุกสัญญาณอยู่ในช่วง ± 1 และอยู่ในรูปเลขส่วนเติมเต็มสอง เพราะว่าใช้ ROM 3 ตัว แต่ละ ROM สามารถเก็บค่าได้ 8 ค่า

เขียน รูปแบบ Controllable canonical form ให้อยู่ในรูป

$$q1(n+1) = q2(n) \tag{4.20}$$

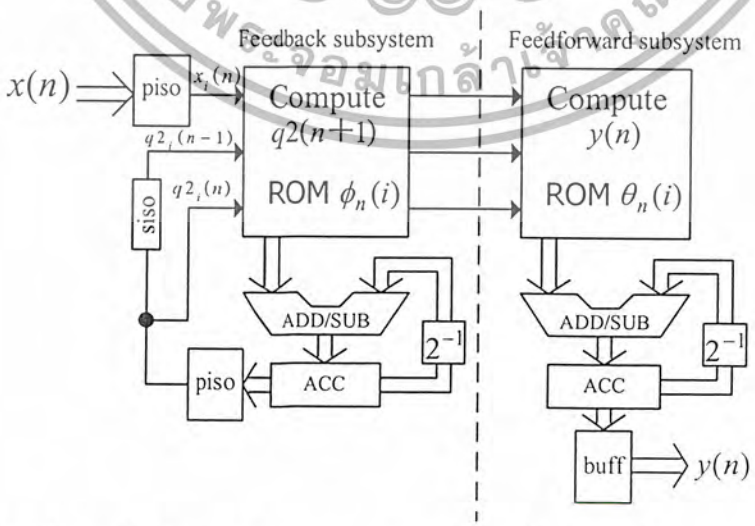
$$q2(n+1) = a_{21} q1(n) + a_{22} q2(n) + b_2 x(n) \tag{4.21}$$

$$y(n) = c_1 q1(n) + c_2 q2(n) + d\, x(n) \tag{4.22}$$

จากสมการที่ (4.20) จะเห็นว่า $q1(n)$ มีค่าเท่ากับการเลื่อนบิทของ $q2(n)$ ไป 1 ตัวอย่าง โดยใช้ วงจรเลื่อนบิต เราจึงเขียนสมการได้ใหม่

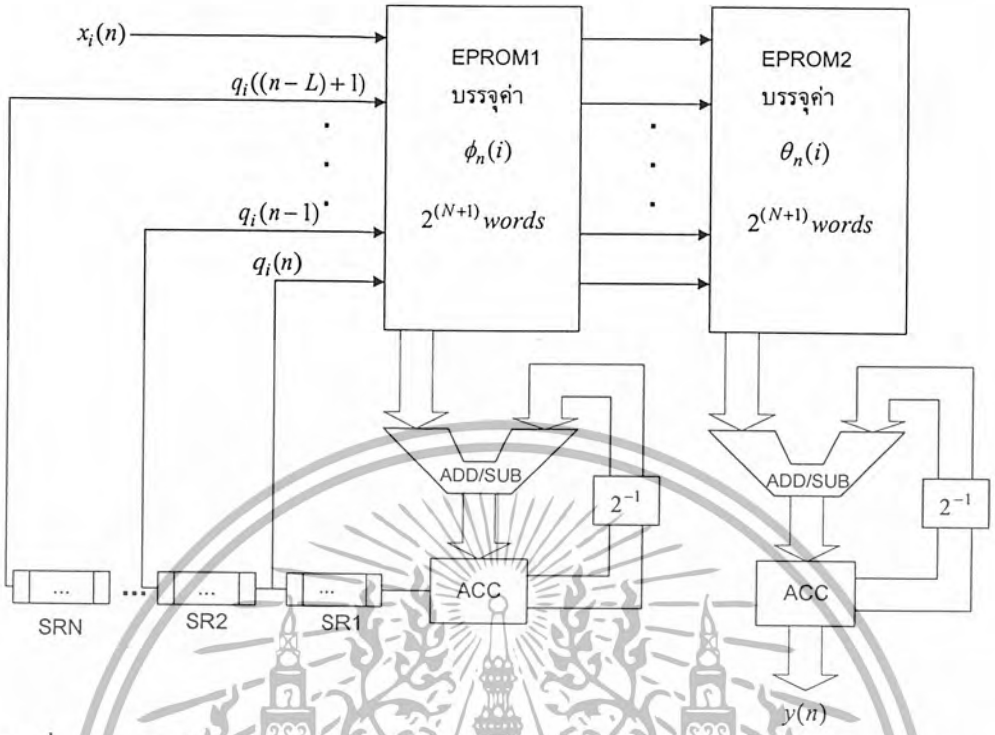
$$q2(n+1) = a_{22} q2(n) + a_{21} q2(n-1) + b_2 x(n) \tag{4.23}$$

$$y(n) = c_2 q2(n) + c_1 q2(n-1) + d\, x(n) \tag{4.24}$$



รูปที่ 4.6 โครงสร้าง Controllable canonical form ที่แทนด้วยโครงสร้างเลขคณิตกระจาย

โดยจากที่กล่าวมาทั้งหมดสามารถออกแบบเป็นวงจรกรองสัญญาณอันดับที่ N ได้ ดังแสดงในรูปที่ 4.7



รูปที่ 4.7 โครงสร้างวงจรกรองสัญญาณเชิงเลขอันดับที่ N แบบ Controllable canonical form

วิธีการแทนสมการผลต่างสืบเนื่องด้วยปริภูมิสเตทแบบ Controllable canonical form นี้ จะทำให้ตัวแปรที่เป็นลำดับสัญญาณอินพุทเหลือจำนวนเพียง N + 1 ตัว ดังนั้น ค่าในตารางเปิดดูจะมีค่าเท่ากับ $2\times(2^{N+1})$ ค่า เพราะฉะนั้นขนาดของหน่วยความจำที่ใช้จะมีขนาด $2L\times(2^{N+1})$ บิท ซึ่งจะเห็นได้ว่า วงจรกรองสัญญาณเชิงเลขที่ออกแบบโดยการแทนด้วยปริภูมิสเตทที่นำเสนอจะใช้หน่วยความจำน้อยกว่าการออกแบบจากสมการผลต่างสืบเนื่องโดยตรง และถ้าอันดับของวงจรกรองยิ่งมีค่ามากขึ้นซึ่งโดยปกติขนาดของหน่วยความจำที่ใช้จะมีขนาดเพิ่มขึ้นในลักษณะเอ็กซ์โปเนนเชียล อัตราส่วนการประหยัดหน่วยความจำ (saving ratio) ก็จะมีค่ามากขึ้นตามไปด้วยซึ่งเป็นการใช้หน่วยความจำอย่างมีประสิทธิภาพ โดยสามารถสรุปได้เป็นสมการ

$$r = \frac{L\times(2^{2N+1})}{2L\times(2^{N+1})}$$

$$= \frac{2^{2N}}{2^{N+1}}$$

$$\therefore \quad r = 2^{N-1} \tag{4.25}$$

โดย r = saving ratio

N = อันดับของวงจรกรองสัญญาณ

4.2 การออกแบบวงจรกรองสัญญาณเชิงเลขที่ใช้โครงสร้างเลขคณิตกระจายโดยการแทนด้วยปริภูมิสเตทในอันดับที่ 2

ฟังก์ชั่นถ่ายโอนของวงจรกรองสัญญาณเชิงเลขอันดับที่ 2 คือ

$$H(z) = \frac{0.0675 + 0.1349z^{-1} + 0.0675z^{-2}}{1 + 1.1430z^{-1} - 0.4128z^{-2}}$$

สามารถเขียนเป็นสมการผลต่างสืบเนื่องแทนความสัมพันธ์ของวงจรกรองได้ดังนี้

$$y(n) = 0.0675\,x(n) + 0.1349(n-1) + 0.0675\,x(n-2) - 1.1430y(n-1) + 0.4128\,y(n-2)$$

ทำการหาค่าสเตทโคแวเรียนซ์เมตริกซ์ K เพื่อใช้สำหรับการสเกลลิ่ง ตามกฎของ L_2 *norm* จะได้

$$K = \begin{bmatrix} 1 & 0.57735026 \\ 0.57735026 & 1 \end{bmatrix}$$

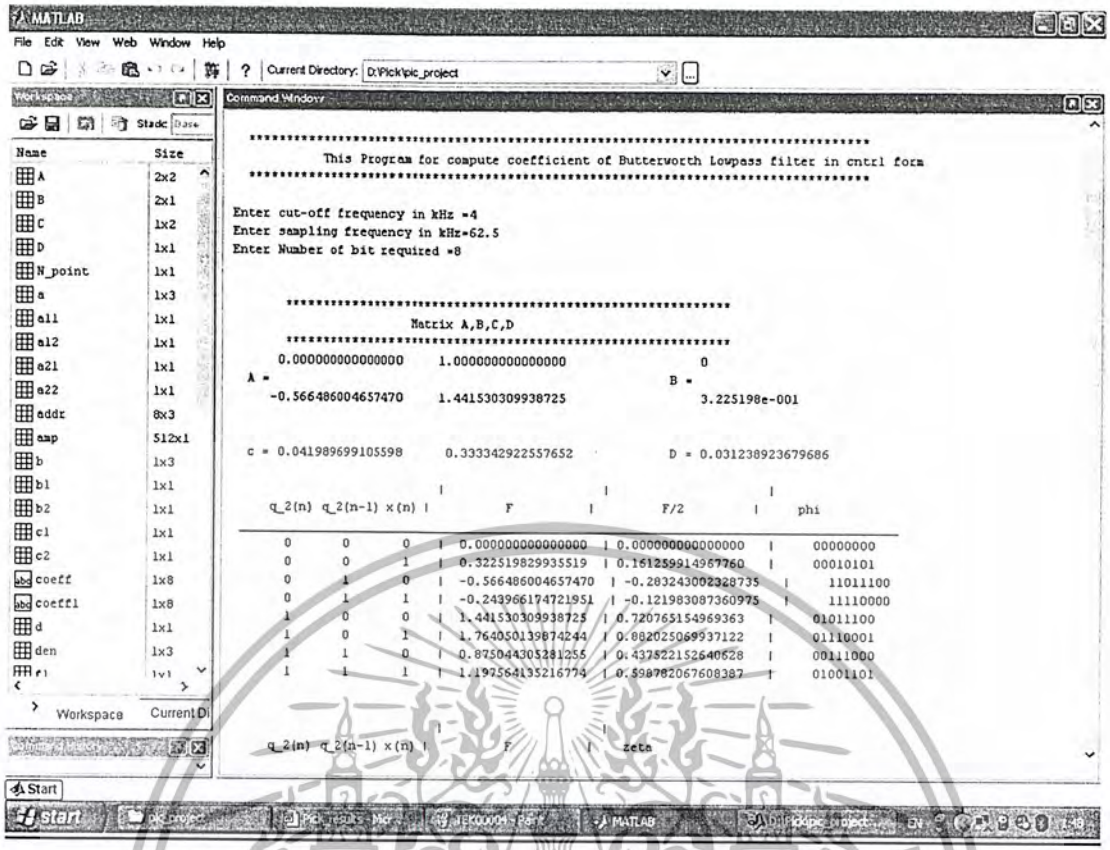
ทำการแทนสมการผลต่างสืบเนื่องให้อยู่ในรูปของปริภูมิสเตท โดยทำการหาค่าเมตริกซ์ A, B, C และ D ที่ผ่านการสเกล จะได้

$$A_s = \begin{bmatrix} 0.72076515 & -0.12514471 \\ 0.37543413 & 0.7207651 \end{bmatrix} \qquad B_s = \begin{bmatrix} 0.75431447 \\ 0.16457527 \end{bmatrix}$$

$$C_s = \begin{bmatrix} 0.07126318 & 0.32662774 \end{bmatrix} \qquad D_s = \begin{bmatrix} 0.03123892 \end{bmatrix}$$

4.3 ผลการใช้โปรแกรม Matlab ในการออกแบบ

หน้าที่หลักของโปรแกรม Matlab จะใช้ในการคำนวณหาค่าเก็บไว้ในตารางเปิดดู โดยการนำค่าเมตริกซ์ที่ผ่านการสเกลแล้วคือ A_s, B_s, C_s, D_s ในข้างต้นมาทำการหาค่าเก็บไว้ในตารางเปิดดู โดยอาศัยสมการที่ (4.14) และสมการที่ (4.16) โดยมีค่าแอดเดรสเริ่มจาก 000 – 111 จากนั้นแทนค่าแอดเดรสทั้งหมดลงในสมการ โดยโครงสร้างแบบ Controllable Canonical form จะใช้ 2 สมการก็จะได้ค่า $\phi_n(i)$ และ $\theta_n(i)$ ตามลำดับ ส่วน โครงสร้างแบบ Minimum Noise จะใช้ 3 สมการคือสมการที่ (4.14),(4.15) และ (4. 16) ก็จะได้ค่า $\phi_{1n}(i)$, $\phi_{2n}(i)$ และ $\theta_n(i)$ ตามลำดับ โดยหน้าจอของโปรแกรมที่ใช้ในการออกแบบสามารถแสดงได้ดังนี้



รูปที่ 4.8 ตัวอย่างผลการออกแบบและคำนวณค่าที่เก็บไว้ในหน่วยความจำ EPROM ทั้ง 2 ตัว ของโครงสร้างแบบ Controllable Canonical form

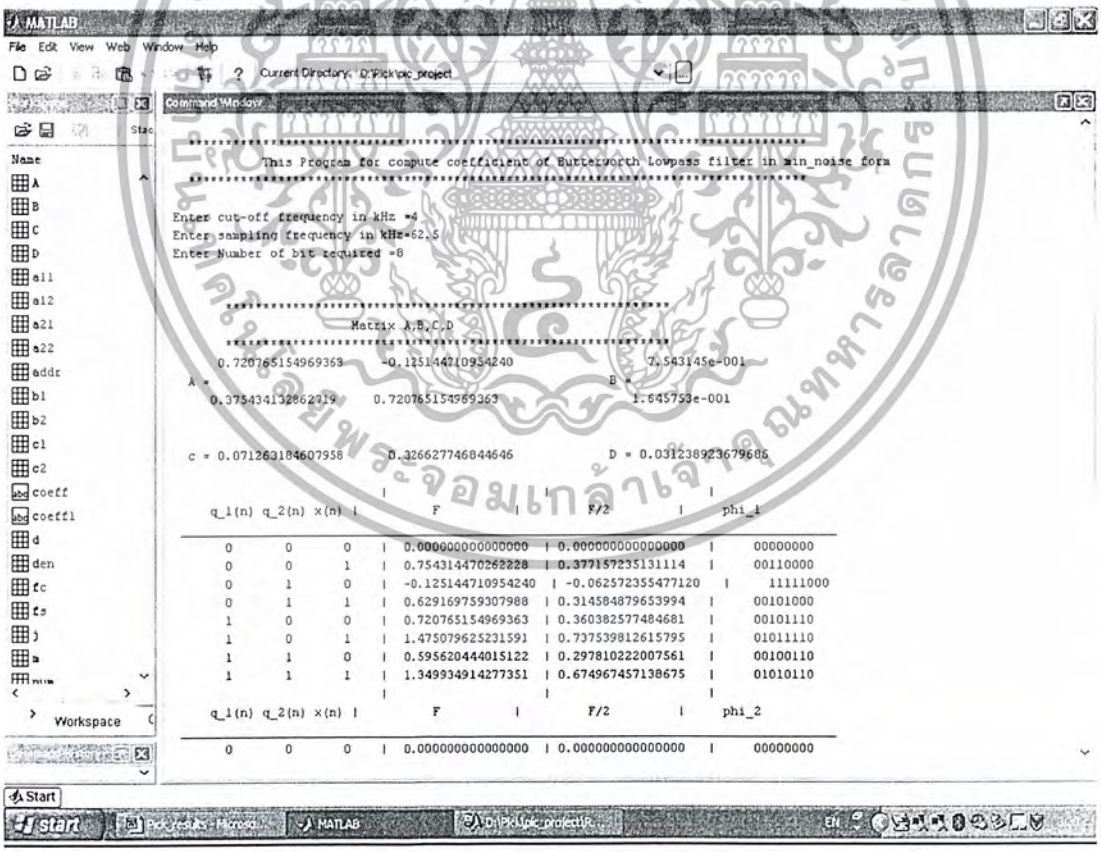
ค่าที่เก็บไว้ในตารางเปิดดู ของโครงสร้างแบบ Controllable Canonical form สามารถแสดงได้ดังนี้

ตารางที่ 4.1 ค่า $\phi_n(i)$ ที่บรรจุไว้ในตารางเปิดดูของโครงสร้าง Controllable Canonical form

$q_i(n)$	$q_i(n-1)$	$x_i(n)$	F	$F/2$	$\phi_n(i)$
0	0	0	0.00000000	0.0000000	0000 0000
0	0	1	0.32251982	0.16125991	00010101
0	1	0	-0.56648600	-0.28324300	11011100
0	1	1	-0.24396617	-0.12198308	11110000
1	0	0	1.44153030	0.72076515	01011100
1	0	1	1.76405013	0.88202506	0110 1011
1	1	0	0.87504430	0.43752215	00111000
1	1	1	1.197564135	0.59878206	01001101

ตารางที่ 4.2 ค่า $\theta_n(i)$ ที่บรรจุไว้ในตารางเปิดดูของโครงสร้าง Controllable Canonical form

$q_i(n)$	$q_i(n-1)$	$x_i(n)$	F	$\theta_n(i)$
0	0	0	0.00000000	0000 0000
0	0	1	0.03123892	00000100
0	1	0	0.04198969	00000101
0	1	1	0.07322862	00001001
1	0	0	0.33334292	00101011
1	0	1	0.36458184	00101111
1	1	0	0.37533262	00110000
1	1	1	0.40657154	00110100



รูปที่ 4.9 ตัวอย่างผลการออกแบบและคำนวณค่าที่เก็บไว้ในหน่วยความจำ EPROM ทั้ง 2 ตัว ของโครงสร้างแบบ Minimum Noise Structure

ค่าที่เก็บไว้ในตารางเปิดดู ของโครงสร้างแบบ Minimum Noise Structure สามารถแสดงได้ดังนี้

ตารางที่ 4.3 ค่า $\phi_1(\mathrm{i})$ ที่บรรจุไว้ในตารางเปิดดูของโครงสร้างแบบ Minimum Noise Structure

$q_1(n)$	$q_1(n-1)$	$x_1(n)$	F	$F/2$	$\phi_1(\mathrm{i})$
0	0	0	0.00000000	0.0000000	0000 0000
0	0	1	0.75431447	0.37715723	00010101
0	1	0	-0.12514471	-0.0625723	11111000
0	1	1	0.62916975	0.3145848	00101000
1	0	0	0.72076515	0.36038257	00101110
1	0	1	1.47507962	0.73753981	01011110
1	1	0	0.595620444	0.29781022	00100110
1	1	1	1.349934914	0.67496745	01010110

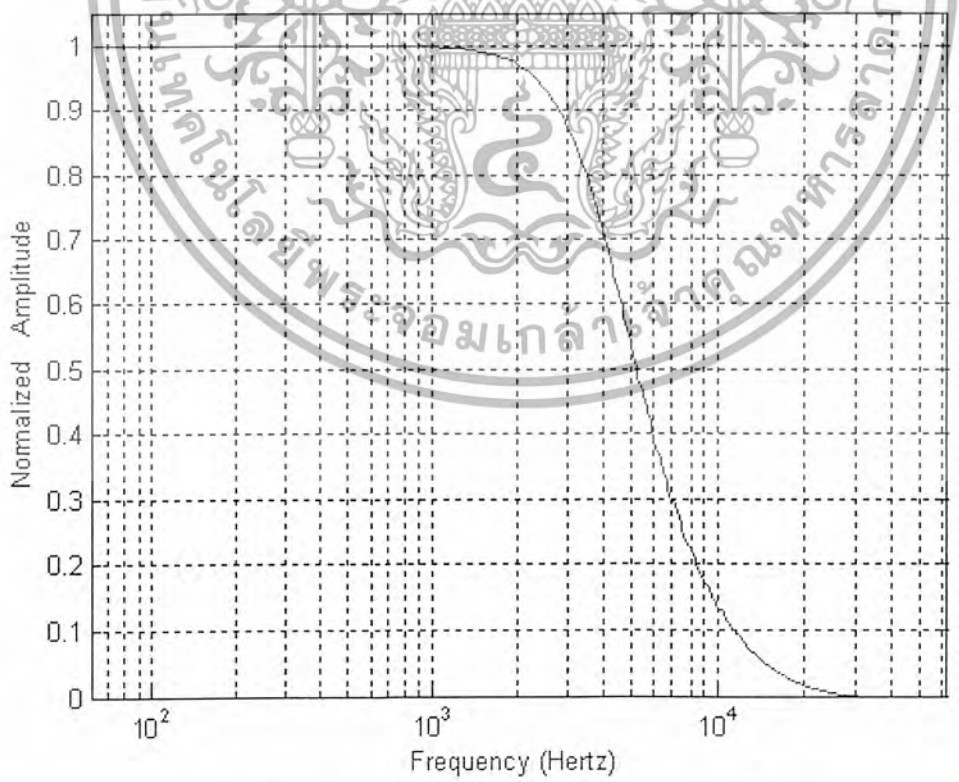
ตารางที่ 4.4 ค่า $\phi_2(\mathrm{i})$ ที่บรรจุไว้ในตารางเปิดดูของโครงสร้างแบบ Minimum Noise Structure

$q_2(n)$	$q_2(n-1)$	$x_2(n)$	F	$F/2$	$\phi_2(\mathrm{i})$
0	0	0	0.00000000	0.0000000	0000 0000
0	0	1	0.16457527	0.08228763	00001011
0	1	0	0.72076515	0.36038257	00101110
0	1	1	0.885340430	0.44267021	00111001
1	0	0	0.375434132	0.187717066	00011000
1	0	1	0.540009408	0.270004704	00100011
1	1	0	1.096199287	0.548099643	01000110
1	1	1	1.2607745630	0.630387281	01010001

ตารางที่ 4.5 ค่า $\theta_n(i)$ ที่บรรจุไว้ในตารางเปิดดูของโครงสร้างแบบ Minimum Noise Structure

$q_1(n)$	$q_1(n-1)$	$x_1(n)$	F	$\theta_n(i)$
0	0	0	0.00000000	0000 0000
0	0	1	0.03123892	00000100
0	1	0	0.32662774	00101010
0	1	1	0.35786667	00101110
1	0	0	0.07126318	00001001
1	0	1	0.10250210	00001101
1	1	0	0.39789093	00110011
1	1	1	0.42912985	00110111

นอกจากนี้เรายังโปรแกรม Matlab ในการแสดงค่าผลตอบสนองทางความถี่โดยกำหนดให้ เป็นวงจรกรองสัญญาณความถี่ต่ำผ่านแบบบัตเตอร์เวิร์ท อันดับที่ 2 ความถี่คัทออฟ ที่ 4 kHz และ ความถี่ในการสุ่มตัวอย่างที่ 62.5 kHz รวมทั้งทำการจำลองการทำงานและสเปกตรัมของสัญญาณอินพุตและเอาท์พุตที่ใช้ในการจำลองการทำงาน



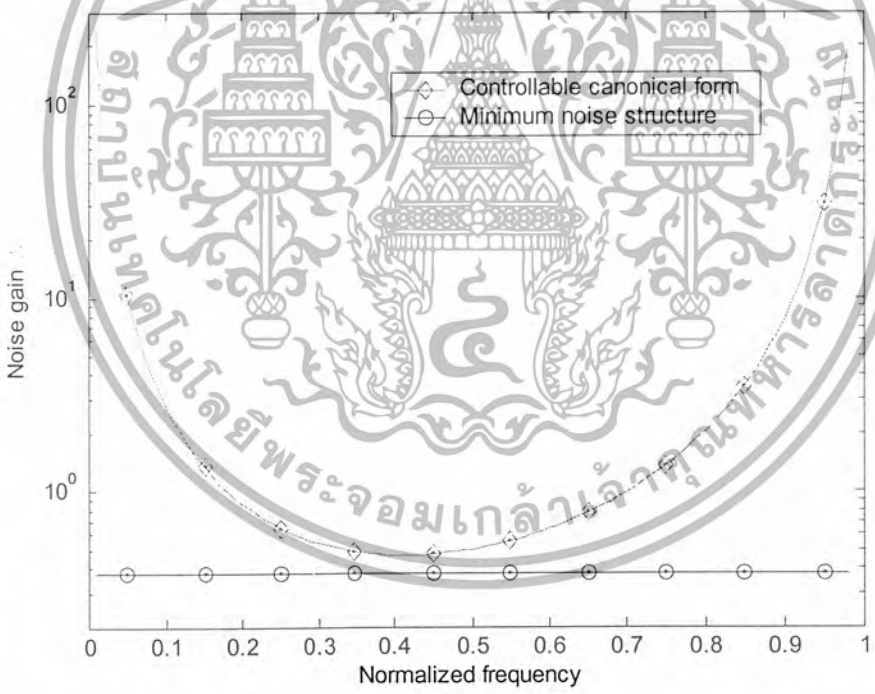
รูปที่ 4.10 แสดงการจำลองผลตอบสนองทางความถี่

4.3 การเปรียบเทียบคุณสมบัติระหว่างโครงสร้าง Controllable canonical form กับโครงสร้าง Minimum Noise structure

ถึงแม้ว่าโครงสร้าง minimum noise แบบโครงสร้างเลขคณิตกระจาย ต้องใช้ ROM และ สเกลลิ่งแอดดิวมูเลเตอร์เพื่อใช้ในการคำนวณ state vector มากกว่าแบบ Controllable canonical form เพราะว่าแบบ Controllable canonical form มีสมการสเตทเพียง 1 สมการ แต่ทั้งสองแบบก็ใช้วิธีการเดียวกันนั่นก็คือใช้วงจร control unit เดียวกัน เพราะว่าสัญญาณทุกสัญญาณในระดับบิตจะต่อแบบขนานกับ ROM แต่ละตัว ซึ่งการประมวลผลจะทำงานแบบขนานกัน ดังนั้นโครงสร้าง minimum noise และ Controllable canonical form สามารถทำงานที่ความถี่เดียวกัน

ผลการ Simulation และ Synthesis โครงสร้างแบบ Controllable canonical form และแบบ minimum noise structure

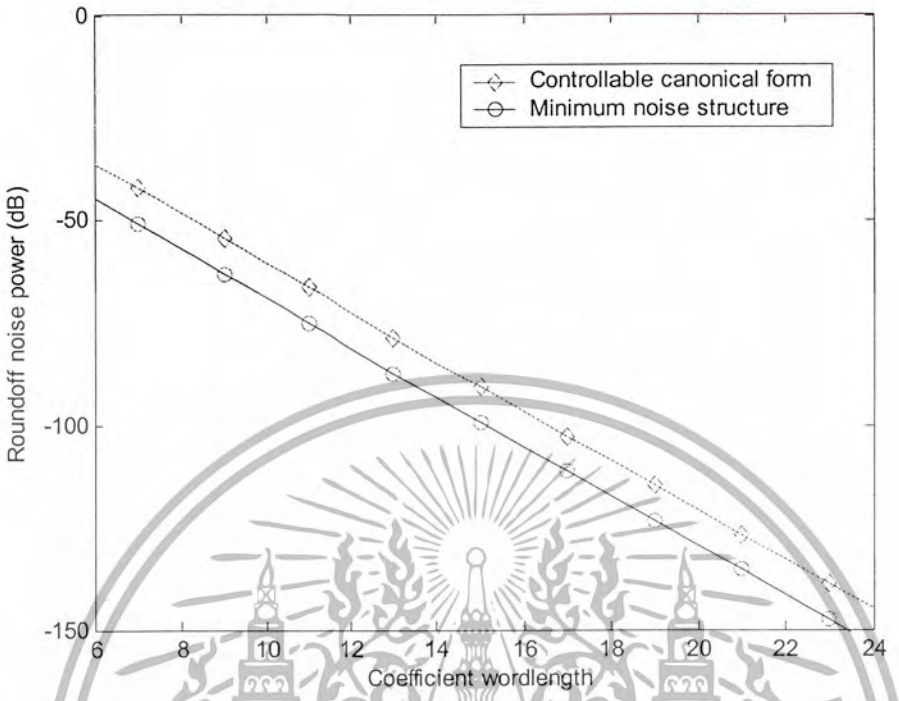
ค่าคงที่ในการปัดเศษสัญญาณรบกวนระหว่าง โครงสร้างแบบ Controllable canonical form และแบบ minimum noise structure แสดงในรูปที่ 4.11 โดยตัวอย่างที่ใช้สังเคราะห์เป็นแบบวงจรกรองสัญญาณความถี่ต่ำผ่านแบบบัตเตอร์เวิร์ทอันดับที่ 2



รูปที่ 4.11 การเปรียบเทียบของค่าคงที่การปัดเศษของสัญญาณรบกวน

จากรูปที่ 4.11 จะเห็นว่าแบบ Controllable canonical form จะมีค่าสัญญาณรบกวนต่ำที่สุดที่ตรงย่านความถี่กลาง normalize หรือสามารถกล่าวได้ว่าย่านความถี่ผ่านอยู่ตรงกลาง ซึ่งก็คือ pole ของระบบอยู่ใกล้ $\pm\pi/2$ สำหรับโครงสร้างแบบ minimum noise จะมีค่าสัญญาณรบกวนต่ำที่สุดทุกย่านความถี่ normalize จากข้างต้นถ้าต้องการออกแบบวงจรกรองสัญญาณแบนด์แคบเช่น $\omega_c = 0.1\pi$ โครงสร้างแบบ minimum noise

สามารถทำได้เพราะจากกราฟค่า noise gain มีค่า 0.375 แต่ว่าถ้าเป็นแบบ Controllable canonical form มีค่า noise gain คือ 2.7461

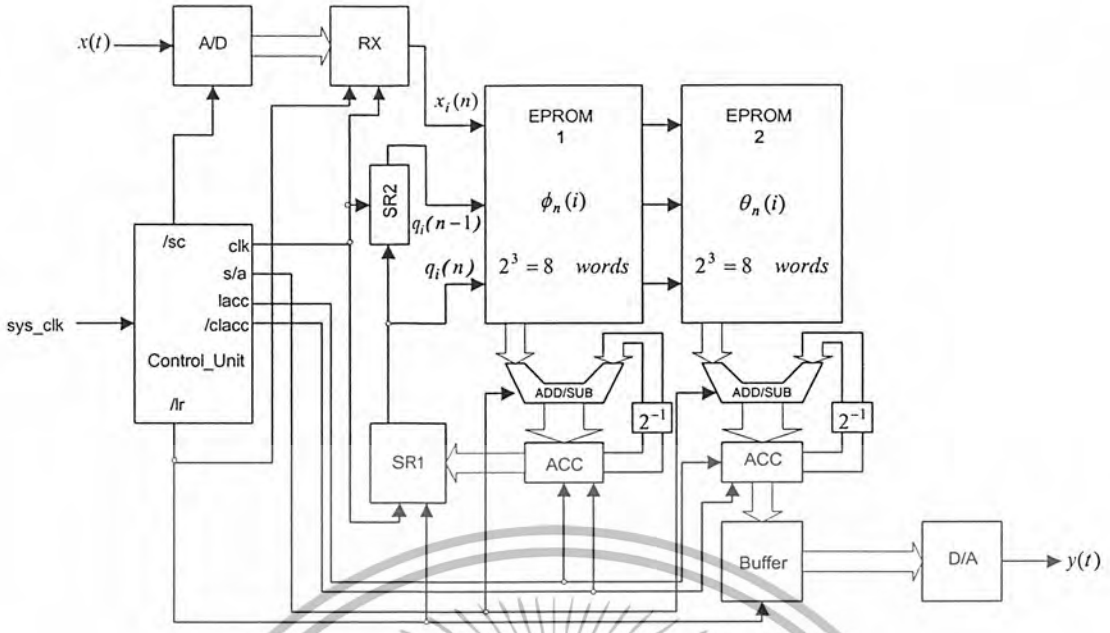


รูปที่ 4.12 กำลังงานของสัญญาณรบกวนเนื่องจากการปัดเศษของทั้งสองโครงสร้าง

รูปที่ 4.12 แสดงค่ากำลังงานของสัญญาณที่ต้องการจากการปัดเศษที่ค่าความยาวบิต 6-24 บิต พิจารณาที่ค่าความยาวบิต 8 บิต โครงสร้างแบบ minimum noise และ Controllable canonical form มีค่า -57.1957dB และ -48.5487 dB ตามลำดับ จากกราฟหากเพิ่มค่าความยาวบิตประมาณ 2 บิต เพื่อที่จะทำการลดค่า สัญญาณรบกวน ทั้งสองแบบจะให้ผลเท่ากัน

4.5 ขั้นตอนในการออกแบบและทดสอบการทำงานของวงจรภายใน

ในการออกแบบโครงสร้างของวงจรกรองสัญญาณเชิงเลข ที่ใช้โครงสร้างเลขคณิตกระจายโดยการแทนด้วยปริภูมิสเตทที่นำเสนอนั้น จะทำการพิจารณาจากวงจรกรองในอันดับที่ 2 เนื่องจากในวงจรที่มีอันดับสูงขึ้นก็จะมีส่วนประกอบที่ใช้งานที่เหมือนกัน ซึ่งโครงสร้างของวงจรกรองสัญญาณเชิงเลขอันดับที่ 2 จากการแทนด้วยปริภูมิสเตท โดยใช้โครงสร้างแบบ Controllable canonical form แสดงดังในรูปที่ 4.13

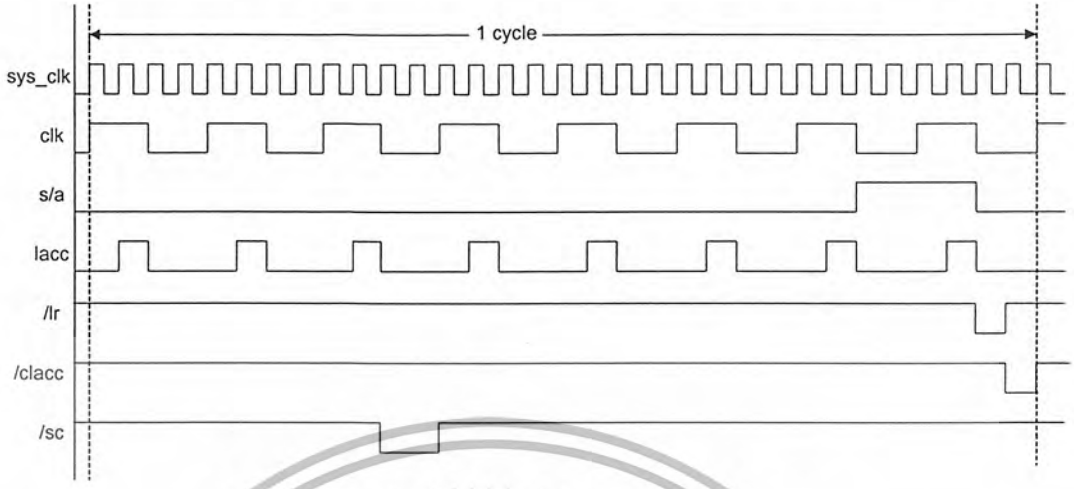


รูปที่ 4.13 โครงสร้างของวงจรกรองสัญญาณเชิงเลขอันดับที่ 2 จากการแทนด้วยปริภูมิสเตทแบบ Controllable Canonical form

จากรูปที่ 4.13 จะแยกอธิบายขั้นตอนของการทำงานเป็น 4 ขั้นตอน ดังนี้

1. วงจร A/D (Analog to Digital Converter) ซึ่งถูกควบคุมด้วยสัญญาณ /sc จะทำการแปลงสัญญาณเชิงอุปมาน(analog) $x(t)$ ให้เป็นลำดับสัญญาณเชิงเลข(digital) $x(n)$ ขนาด 8 บิท
2. สัญญาณควบคุม/lr ทำหน้าที่โหลดลำดับสัญญาณหรือข้อมูล $x(n)$ เข้าไปเก็บไว้ในรีจิสเตอร์ RX
3. สัญญาณนาฬิกา clk จะทำการเลื่อนข้อมูลภายในรีจิสเตอร์ RX, SR1 และ SR2 ไปครั้งละ 1 บิท เอาท์พุทของรีจิสเตอร์ RX, SR1 และ SR2 ที่ถูกเลื่อนแต่ละครั้งจะเป็นแอดเดรสของ EPROM ทั้ง 2 ตัว เอาท์พุทของ EPROM ทั้ง 2 ตัว จะถูกส่งไปบวกกับค่าที่อยู่ใน ACC ด้วยวงจร ADD/SUB (ซึ่งถูกควบคุมด้วยสัญญาณ s/a) ผลลัพธ์ที่ได้จะถูกโหลดเข้าเก็บไว้ใน ACC ด้วยสัญญาณ lacc (การคูณค่าที่อยู่ใน ACC ด้วย 2^{-1} หรือเป็นการเลื่อนข้อมูลไปทางขวา 1 บิท ก่อนที่จะนำไปบวกกับค่าจาก EPROM ถูกออกแบบในลักษณะฮาร์ดแวร์สเกลลิ่ง (Hardware scaling) จึงไม่จำเป็นที่จะต้องมีวงจรเลื่อนข้อมูลไปทางขวา 1 บิท)
4. clk จะเลื่อนข้อมูลในแต่ละรีจิสเตอร์ไปอีก 1 บิท แล้วกระทำซ้ำข้อ 3 จนกระทั่ง clk เลื่อนข้อมูลไปถึงบิทที่ 8 จึงนำค่าที่ได้จากเอาท์พุทของ EPROM ทั้ง 2 ตัว ไปลบออกจากค่าที่อยู่ใน ACC ผลลัพธ์ที่ได้จากการคำนวณของ EPROM1 จะถูกโหลดเข้าไปเก็บไว้ในรีจิสเตอร์ SR1 ส่วนผลลัพธ์ที่ได้จากการคำนวณของ EPROM2 จะถูกโหลดเข้าไปเก็บไว้ใน buffer (ด้วยสัญญาณ /lr) เพื่อทำการแปลงสัญญาณเชิงเลขให้เป็นสัญญาณเชิงอุปมาน ด้วยวงจร D/A จากนั้นทำการลบข้อมูลภายใน ACC ด้วยสัญญาณ /clacc และจะวนกลับไปทำงานซ้ำในขั้นตอนที่ 1, 2, 3, 4 ตามลำดับ

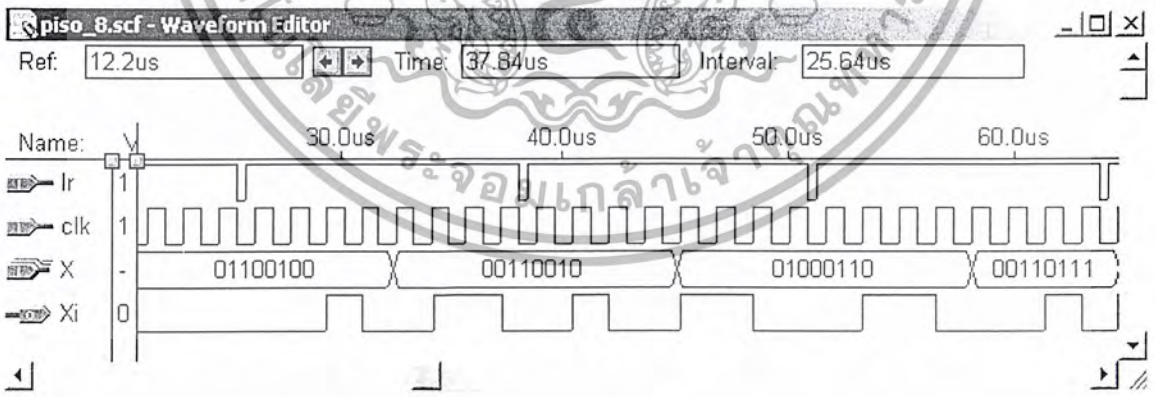
ส่วนสัญญาณที่ใช้ในการควบคุมการทำงานสามารถแสดงไทม์มิ่งไดอะแกรม(Timing diagram) ได้ดังนี้



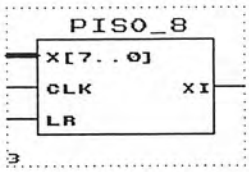
รูปที่ 4.14 ไทม์มิ่งไดอะแกรมของสัญญาณควบคุม

โดยมีขั้นตอนในการออกแบบส่วนต่างๆ ดังนี้

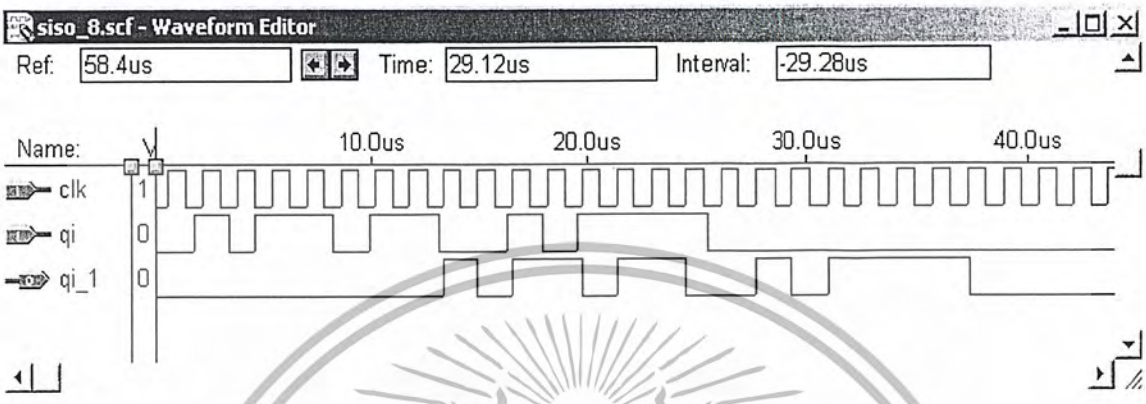
1. รีจิสเตอร์ RX, SR1 ซึ่งเป็น PISO (Parallel in Serial out shift register) และ รีจิสเตอร์ SR2 ซึ่งเป็น SISO (Serial in Serial out shift register) โดยในส่วนของรีจิสเตอร์นี้ถูกออกแบบไว้สำหรับ เลื่อนข้อมูลหรือสัญญาณในรีจิสเตอร์ขนาด 8 บิท ซึ่งผลที่ได้จากการเลื่อนข้อมูลแต่ละบิทจะเป็นตัวกำหนดแอดเดรสของ EPROM และรีจิสเตอร์ Buffer ซึ่งใช้ในการเก็บผลลัพธ์ที่ได้จากการ คำนวณไว้เพื่อรอการแปลงเป็นสัญญาณเชิงอุปมาน โดยมีฟังก์ชั่นการทำงานตามผลการจำลอง การทำงาน ดังนี้



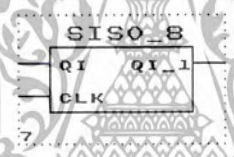
รูปที่ 4.15 การจำลองการทำงานของ PISO



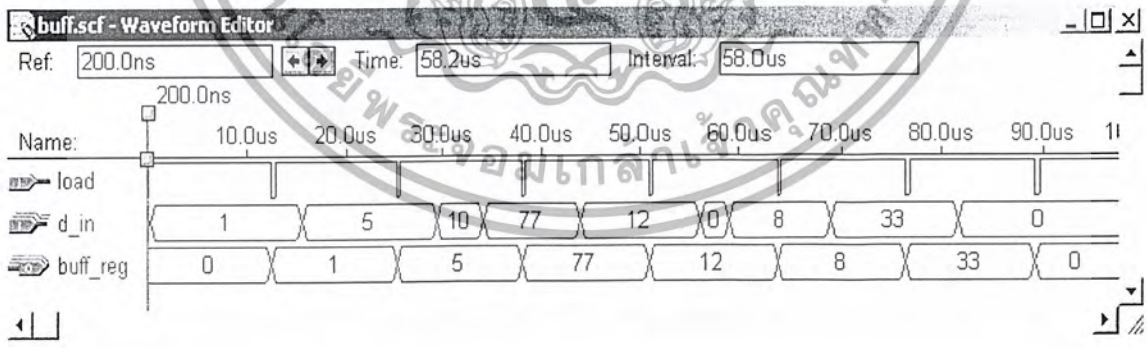
รูปที่ 4.16 สัญลักษณ์วงจร Parallel-in Serial-out ที่เกิดจากการ Schematic



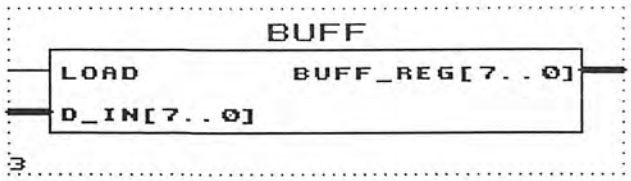
รูปที่ 4.17 การจำลองการทำงานของ SISO



รูปที่ 4.18 สัญลักษณ์วงจร Serial-in Serial-out ที่เกิดจากการ Schematic



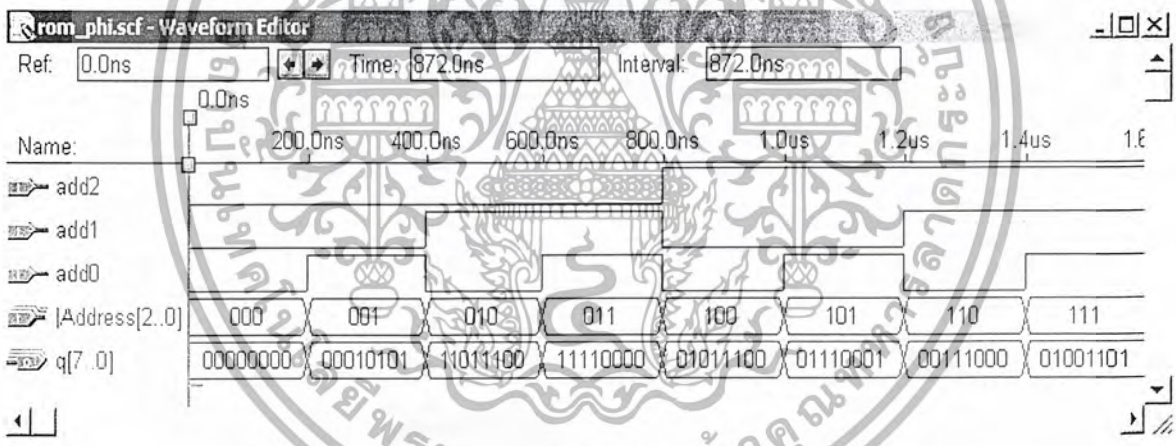
รูปที่ 4.19 การจำลองการทำงานของ Buffer



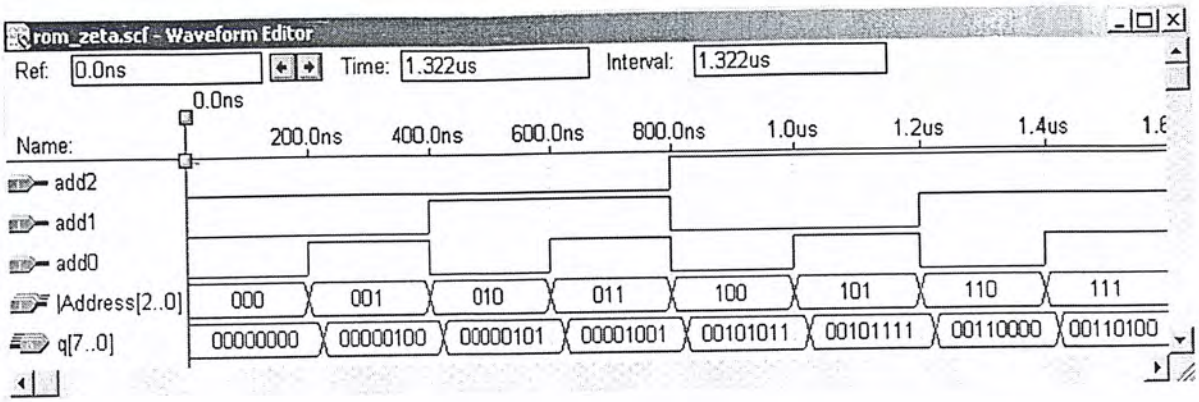
รูปที่ 4.20 สัญลักษณ์วงจร Buffer ที่เกิดจากการ Schematic

2. หน่วยความจำ EPROM1 และ EPROM2 สำหรับเก็บค่าผลคูณย่อยไว้ในตารางเปิดดู ซึ่งเมื่อเข้าใจหลักการทำงานของโครงสร้างเลขคณิตกระจายก็จะทราบว่าจะต้องใช้ขนาดของหน่วยความจำเท่าใด กรณีของวงจรกรองในอันดับที่ 2 แต่ละหน่วยความจำจะมีขนาดเท่ากับ $8 \times 8 = 64$ บิท (กรณีใช้ความยาวคำของค่าในหน่วยความจำขนาด 8 บิท) โดย EPROM1 จะเก็บค่าที่ใช้คำนวณสมการสเตท และ EPROM2 เก็บค่าที่ใช้ในการคำนวณสมการเอาท์พุท ซึ่งทำการเขียนเป็นหน่วยการออกแบบแพ็กเก็ต (Package Design unit) เพื่อเก็บค่าในตารางเปิดดู $\phi_n(i)$ และ $\theta_n(i)$ ไว้ในหน่วยการออกแบบนี้ ค่าในตารางเปิดดูนี้สามารถเรียกไปใช้ได้โดยหน่วยการออกแบบเอนทิตี้ (Entity Design unit) และ หน่วยการออกแบบสถาปัตยกรรม (Architecture Design unit)

ผลการจำลองการทำงานสามารถแสดงได้ดังนี้

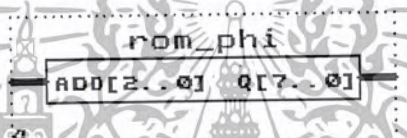


ก. ผลลัพธ์จาก EPROM 1

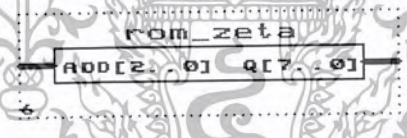


ข. ผลลัพธ์จาก EPROM 2

รูปที่ 4.21 การจำลองการทำงานของหน่วยความจำ EPROM 1 และ EPROM 2

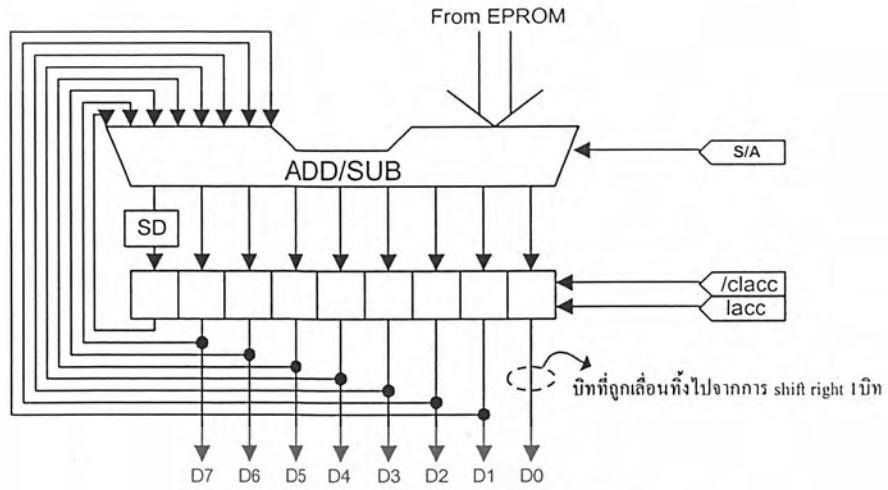


รูปที่ 4.22 สัญลักษณ์ตัวเก็บข้อมูล EPROM1 ที่เกิดจากการ Schematic



รูปที่ 4.23 สัญลักษณ์ตัวเก็บข้อมูล EPROM2 ที่เกิดจากการ Schematic

3. วงจรสเกลลิ่งแอคคิวมูเลเตอร์ (Scaling Accumulator) โดยอาศัยคุณสมบัติการบวกลบแบบเลข ส่วนเติมเต็มสองแทนการคูณโดยตรง ซึ่งประกอบด้วยวงจร บวก/ลบ (ADD/SUB) และ รีจิสเตอร์แอคคิวมูเลเตอร์ เอาท์พุทของรีจิสเตอร์แอคคิวมูเลเตอร์ถูกออกแบบให้เป็นในลักษณะ ฮาร์ดแวร์สเกลลิ่ง (Hardware Scaling) ด้วย 2^{-1} ก่อนที่จะป้อนกลับไปบวกกับค่าที่ออกมาจากหน่วยความจำตัวต่อไป และเพื่อเป็นการหลีกเลี่ยงการเลื่อนบิทผิดพลาดไปยังตำแหน่งของบิท เครื่องหมายในกรณีที่ตัวบวกเกิดการล้น (Overflow) ดังนั้นจึงต้องมีวงจรเพิ่มเติมสำหรับการ ตรวจสอบบิทเครื่องหมาย (Sign Digit) โดยใช้เอ็กคลูซีฟออร์เกท นั่นคือจะเอาบิทเครื่องหมายของข้อมูลที่เข้าวงจรบวก (a° และ b°) และบิทตัวทด (c°) มาทำเอ็กคลูซีฟออร์กัน โดยจากวงจรบวก ถ้า $a+b=d$ ดังนั้นบิทเครื่องหมายของผลลัพธ์จากการเลื่อนข้อมูล (d) ไปทางขวา 1 บิท ($2^{-1}d$) หาได้จากสมการ $d^{\circ}=a^{\circ}\oplus b^{\circ}\oplus c^{\circ}$

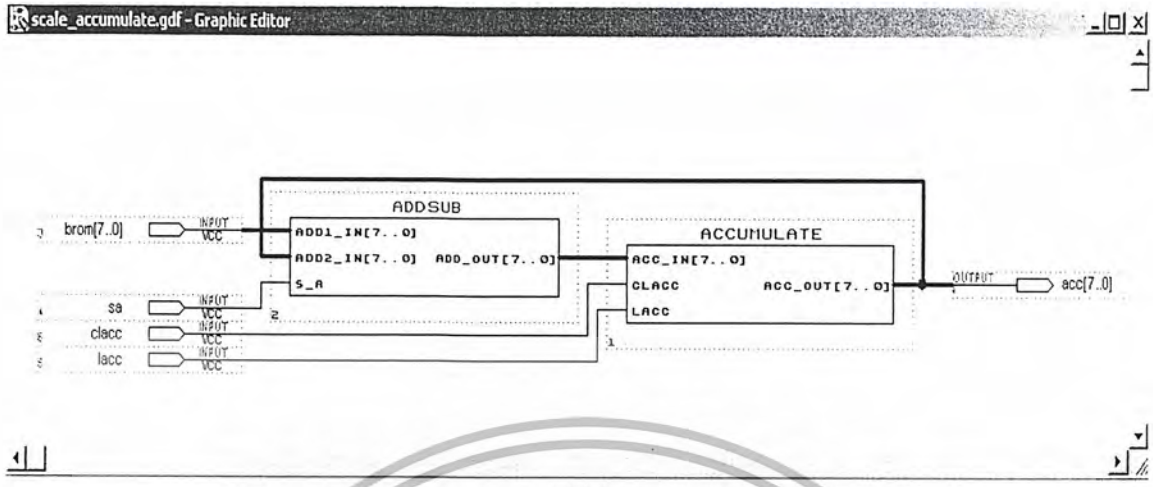


รูปที่ 4.24 วงจรสเกลลิ่งแอคคิวมูเลเตอร์

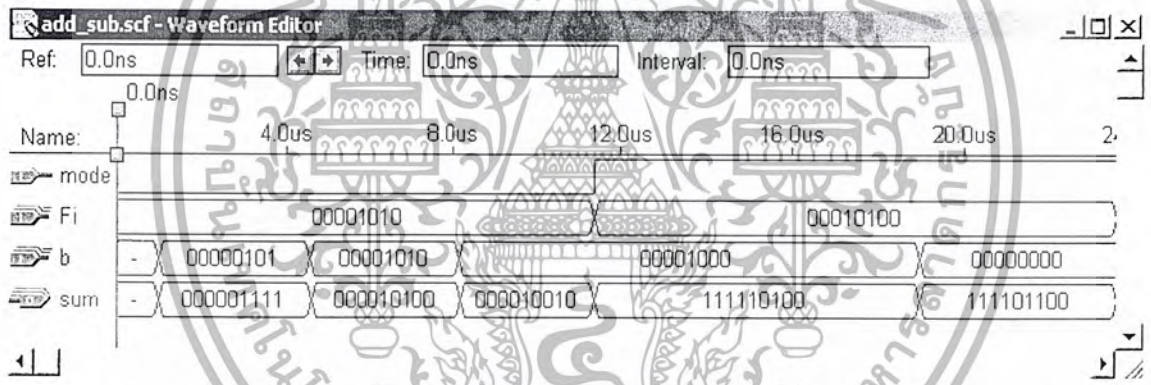
รูปที่ 4.24 แสดงวงจรสเกลลิ่งแอคคิวมูเลเตอร์ โดยมีวงจร SD ใช้เพื่อป้องกันกรณีที่เกิดการล้นของสัญญาณไปทับบิทเครื่องหมาย และแสดงถึงลักษณะการเชื่อมโยงของสัญญาณในลักษณะของฮาร์ดแวร์สเกลลิ่งด้วย 2^{-1} หรือเลื่อนข้อมูลไปทางขวา 1 บิท ซึ่งวงจร SD เสมือนเป็นวงจรที่ใช้เติมบิทเครื่องหมายหลังจากการเลื่อนข้อมูลไปทางขวา โดยแสดงเป็นตัวอย่างการทำงานของวงจรเมื่อเปรียบเทียบกับการคำนวณโดยเลขฐานสิบ ดังนี้

ค่าจาก EPROM	0.046875	0.000 0000
ค่าอินพุทที่จะนำมาบวก	+ 0.015625	+ 0.000 0010
ค่าที่เก็บไว้ใน ACC	0.0625	0 0.000 1000 ; วงจร SD จะให้ผลลัพธ์เป็น 0
	÷2	÷ 2
ค่าที่ได้หลังจากการคูณด้วย 2^{-1}	0.03125	0.000 0100 ; เลื่อนข้อมูลไปทางขวา 1 บิท
ค่าจาก EPROM	- 0.25	+ 1.110 0000 ; ลบเลขแบบส่วนเติมเต็มสอง
∴ เอาท์พุท	- 0.21875	1.110 0100 ; มีค่าเท่ากับ –0.21875

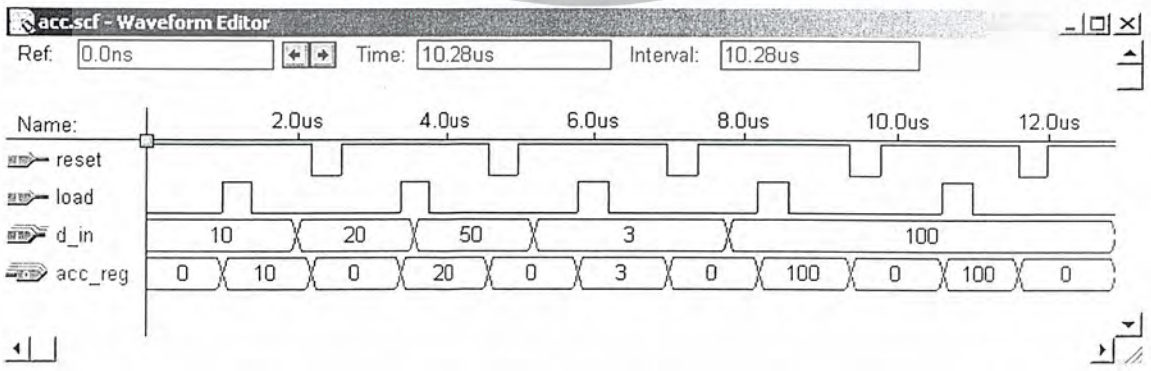
ภายในวงจรสเกลลิ่งแอคคิวมูเลเตอร์ Scaling Accumulator แสดงได้ดังนี้



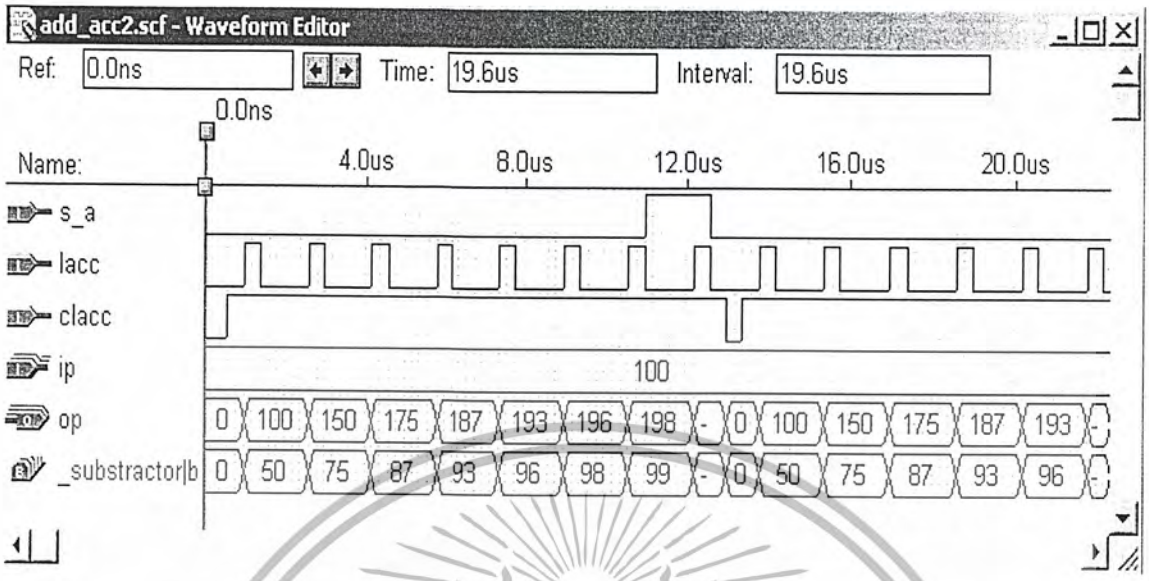
รูปที่ 4.25 วงจรภายในของ Scaling Accumulator ซึ่งประกอบไปด้วย
วงจรบวกลบ วงจรบวกสะสม และวงจรหารสอง



รูปที่ 4.26 การจำลองการทำงานของวงจรบวกลบสัญญาณ Add/Sub



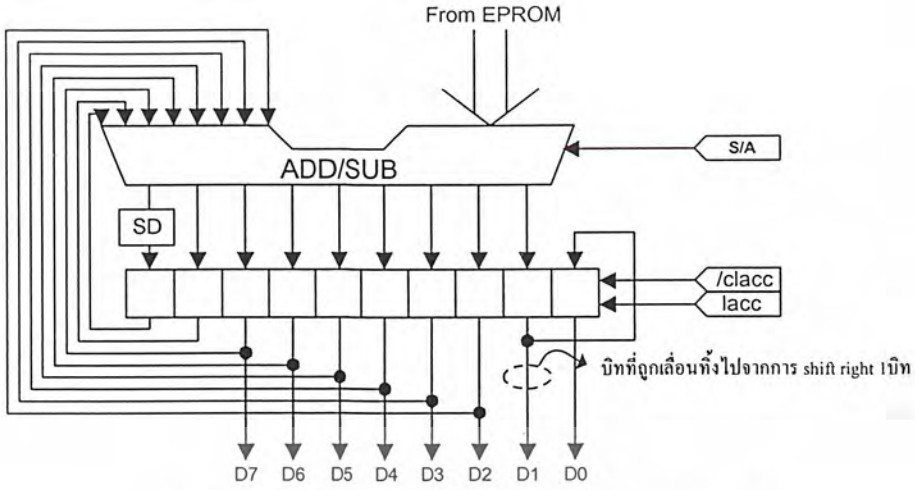
รูปที่ 4.27 การจำลองการทำงานของวงจรบวกสะสม Accumulator



รูปที่ 4.28 การจำลองการทำงานของสเกลลิ่งแอคคิวมูเลเตอร์

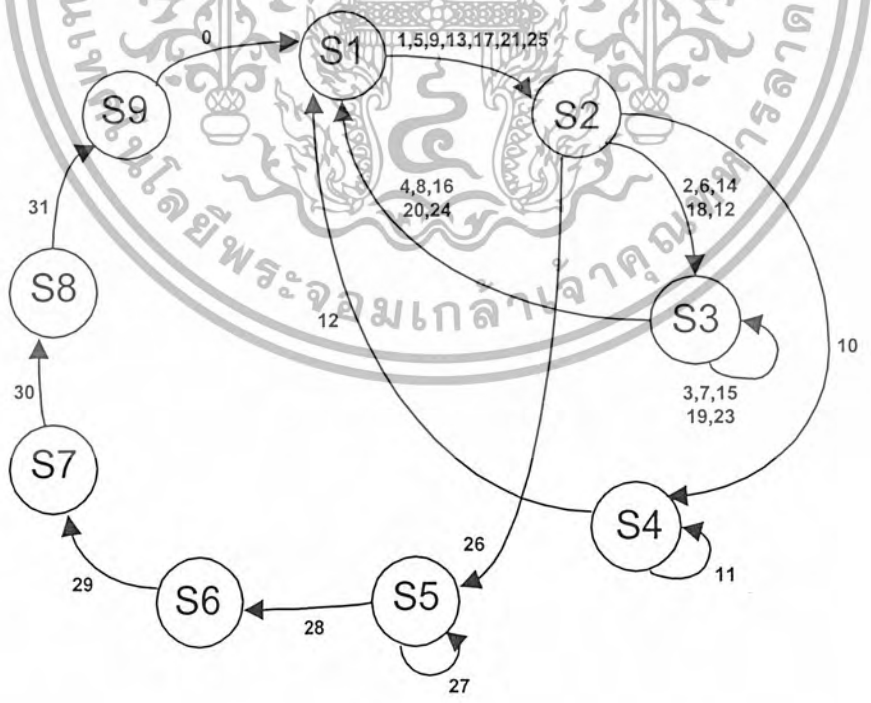
รูปที่ 4.29 สัญลักษณ์วงจรสเกลลิ่งแอคคิวมูเลเตอร์ ที่เกิดจากการ Schematic

ในกรณีค่าที่จะเก็บไว้ในตารางเปิดดูมีค่าเกิน 1 จะต้องทำการหารให้มีค่าไม่เกินหนึ่ง แต่ก่อนที่จะโหลดเอาท์พุทออกจะต้องมีการคูณกลับก่อน เช่นกรณีที่ค่าในตารางเปิดดูถูกหารด้วย 2 ดังนั้นเวลาโหลดเอาท์พุทออกจะต้องคูณ 2 กลับคืน ซึ่งสามารถแสดงวงจรสเกลลิ่งแอคคิวมูเลเตอร์ ได้ดังนี้



รูปที่ 4.30 วงจรสเกลลิ่งแอคคิวมูเลเตอร์ที่มีการคูณ2 กลับคืน

5. วงจรควบคุม Control Unit มีไว้เพื่อควบคุมส่วนต่างๆภายในวงจรกรองสัญญาณเชิงเลขให้ทำงานสอดคล้องกัน เพื่อที่จะสร้างเอาท์พุทที่ถูกต้องตามต้องการ วงจรควบคุมโดยทั่วไปจะผลิตสัญญาณนาฬิกาให้สอดคล้องกับการทำงานและคงสภาพของระดับสัญญาณนั้นๆ ไว้ตาม ช่วงเวลาที่เหมาะสมกับการทำงานนั้นๆ โดยในการออกแบบจะใช้วิธีเขียนเป็นสเตทแมชชีน (State Machine) จากไทม์ทิ่งไดอะแกรมที่กำหนดไว้เพื่อผลิตสัญญาณควบคุม clk, /lr, /sc, lacc, /clacc, s/a โดยมีสเตทไดอะแกรม (State diagram) ดังนี้



รูปที่ 4.31 สเตทไดอะแกรมของ Control Unit

โดยมีการกำหนดค่าสเตท (State Assignments) ดังนี้

S1 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 100111

S2 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 101111

S3 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 000111

S4 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 000110

S5 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 010111

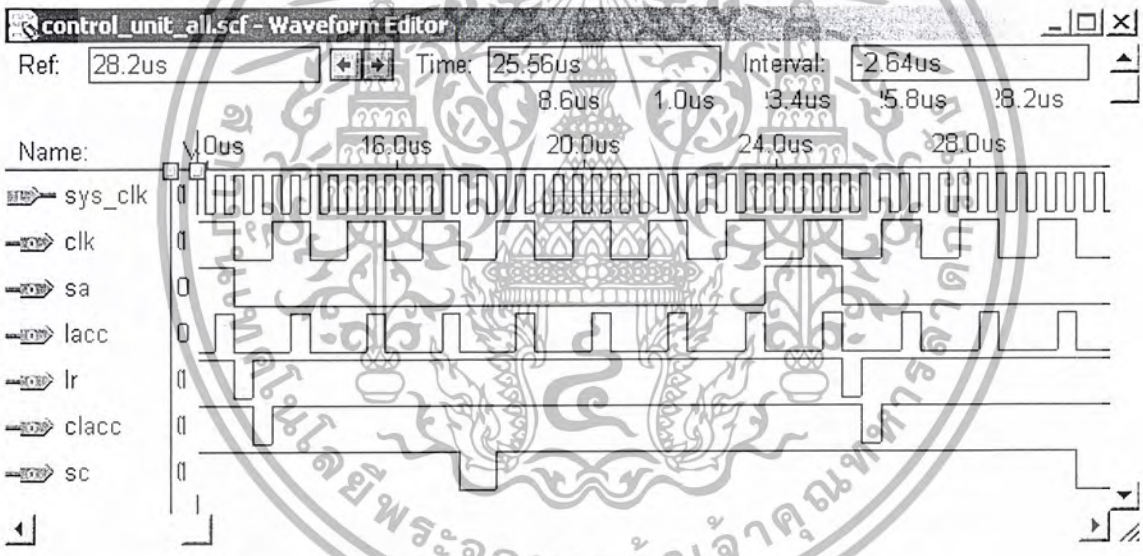
S6 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 110111

S7 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 111111

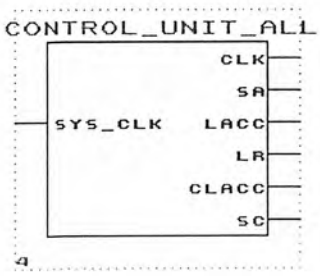
S8 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 000011

S9 เป็นสเตทที่ clk, s/a, lacc, /lr, /clacc, /sc มีค่า 000101

ส่วนการจำลองการทำงานสามารถแสดงได้ดังรูป

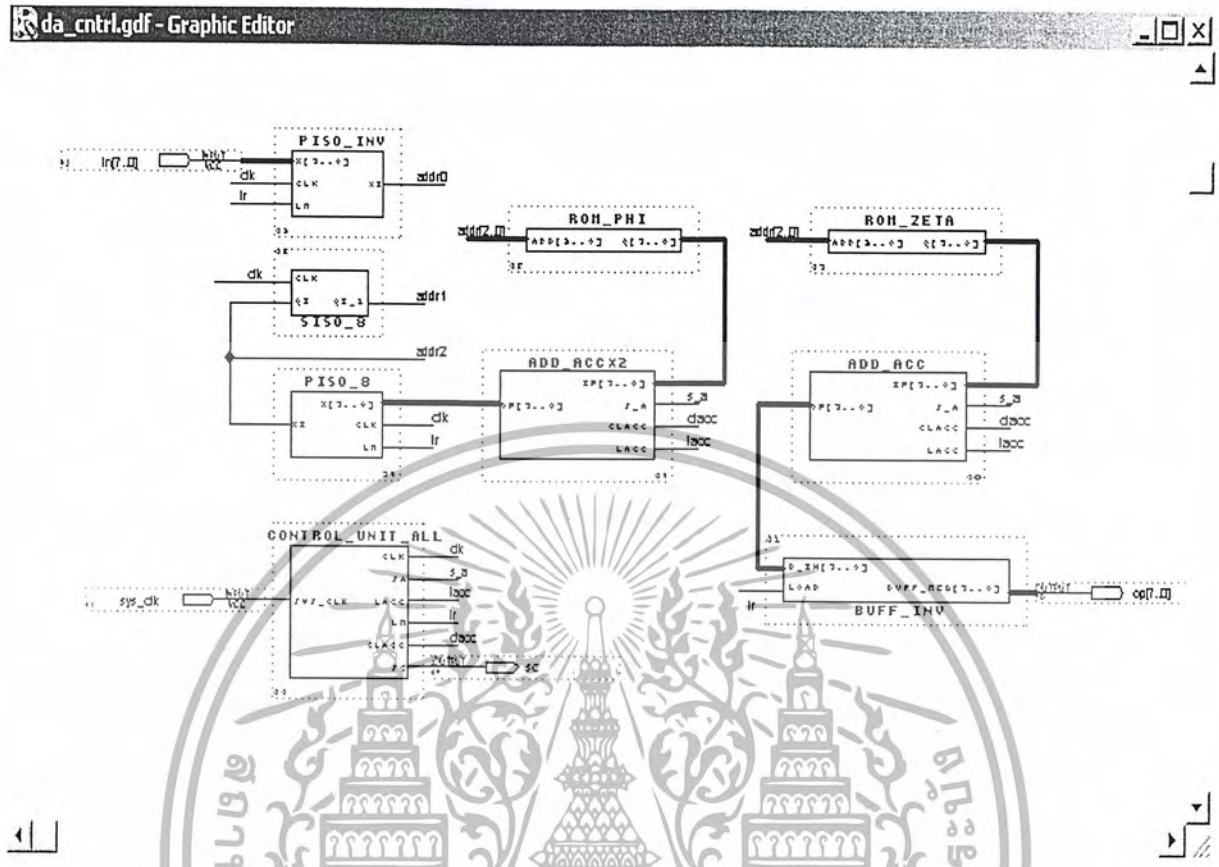


รูปที่ 4.32 การจำลองการทำงานของ Control Unit



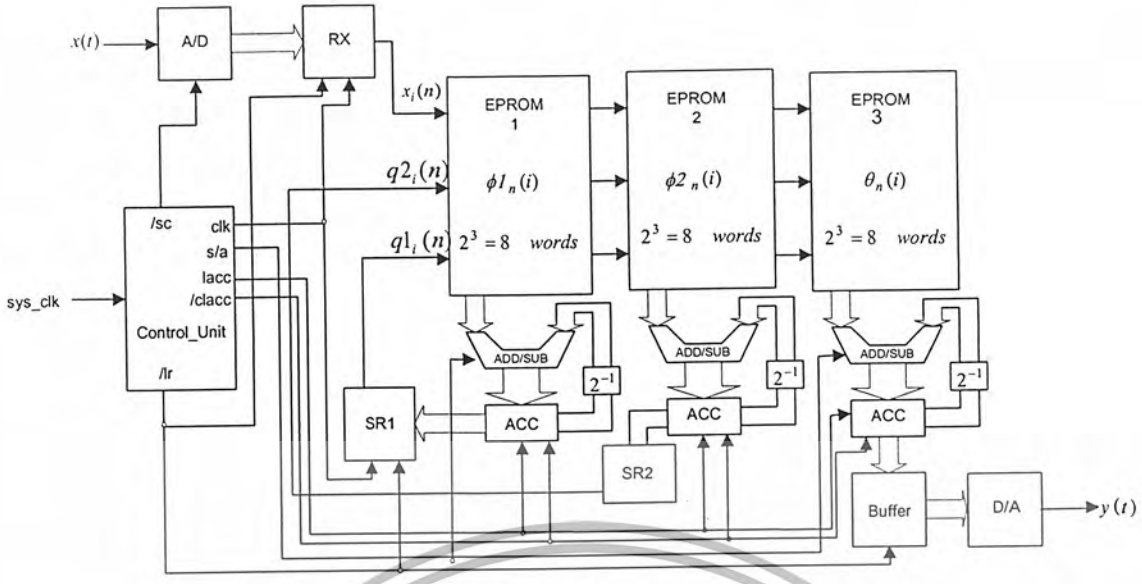
รูปที่ 4.33 สัญลักษณ์วงจร Control Unit ที่เกิดจากการ Schematic

เมื่อได้วงจรครบทุกวงจรแล้วนำแต่ละวงจรมาประกอบกันเป็นวงจร Digital filter

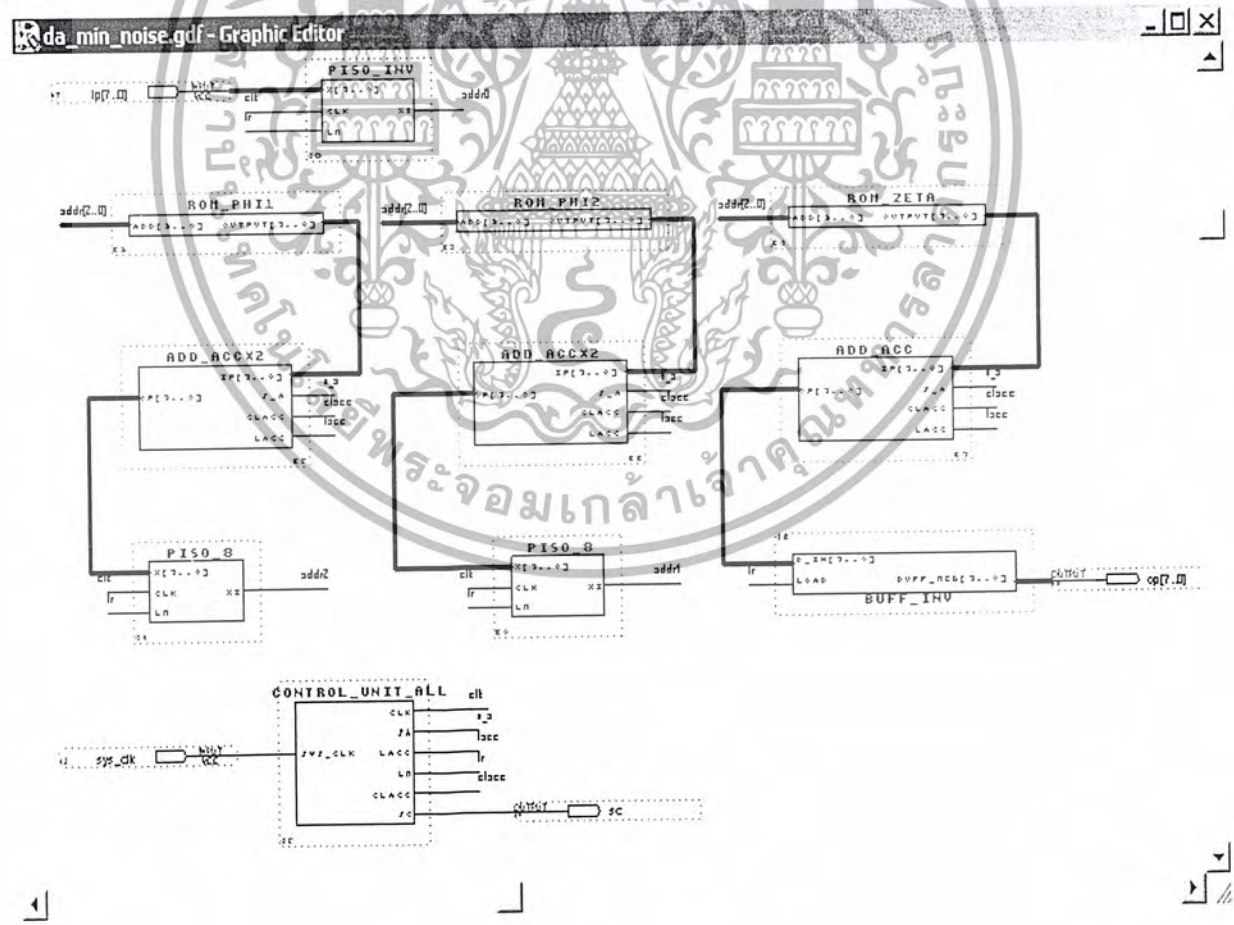


รูปที่ 4.34 วงจรกรองสัญญาณเชิงเลข โครงสร้าง Controllable canonical form ที่จะทำการบรรจุลงใน FPGA

ในการออกแบบโครงสร้างของวงจรกรองสัญญาณเชิงเลข อันดับที่ 2 ที่ใช้โครงสร้างเลขคณิตกระจาย โดยการแทนด้วยปริภูมิสเตทที่นำเสนอนั้น จะทำการศึกษาโครงสร้างแบบ Minimum Noise ซึ่งแสดงโครงสร้างดังในรูปที่ 4.35



รูปที่ 4.35 โครงสร้างของวงจรกรองสัญญาณเชิงเลขอันดับที่ 2 โดยใช้โครงสร้างเลขคณิตกระจายจากการแทนด้วยปริภูมิสเตทแบบ Minimum Noise

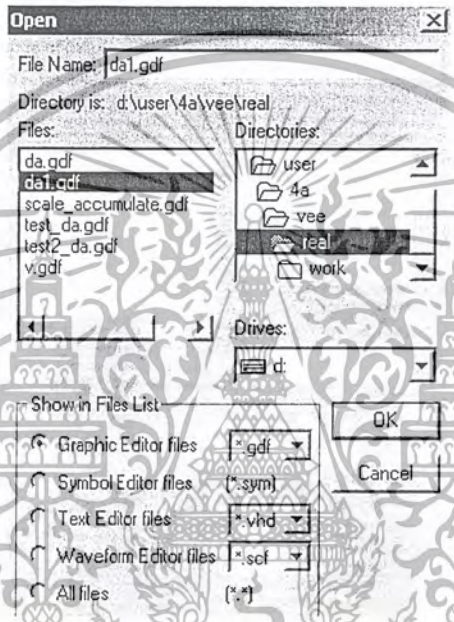


รูปที่ 4.36 โครงสร้างของวงจรกรองสัญญาณเชิงเลข แบบ Minimum Noise ที่จะบรรจุลงใน FPGA

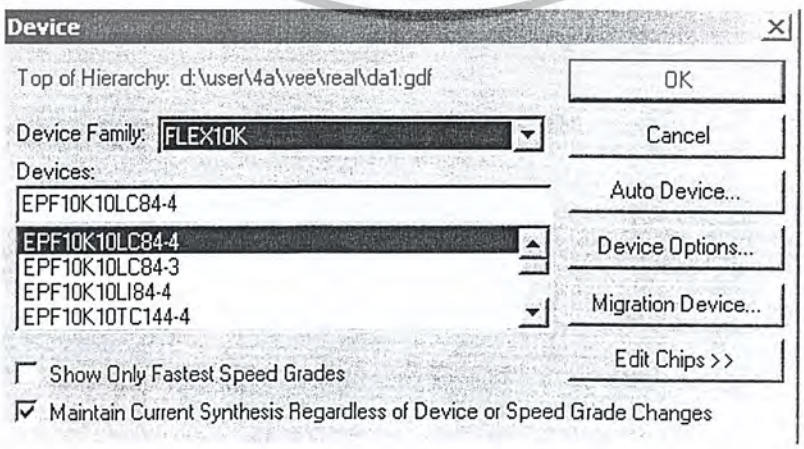
4.5 การใช้งานโปรแกรม MAX+plus II สำหรับการออกแบบ digital filter ด้วย FPGA

โปรแกรม MAX+plusII เป็นโปรแกรมที่ถูกพัฒนาขึ้นมาเพื่อใช้ในการออกแบบระบบเชิงเลข(Digital System) กับชิป FPGA ของบริษัท Altera โดยเฉพาะเป็นโปรแกรมที่มีความสะดวกในการออกแบบสำหรับอุปกรณ์ FPGA ของบริษัท Altera อย่างมาก เนื่องจากสามารถทำงานได้ตั้งแต่เขียน Software code, Compile เพื่อตรวจสอบไวยากรณ์ของภาษา VHDL (Syntax check) จำลองการทำงานสังเคราะห์วงจรทำ PPR: Partitioning place and route จนถึงสร้าง file สำหรับ download ลงบน FPGA ภายในโปรแกรมเพียงตัวเดียว โดยมีขั้นตอนในการใช้งานดังนี้

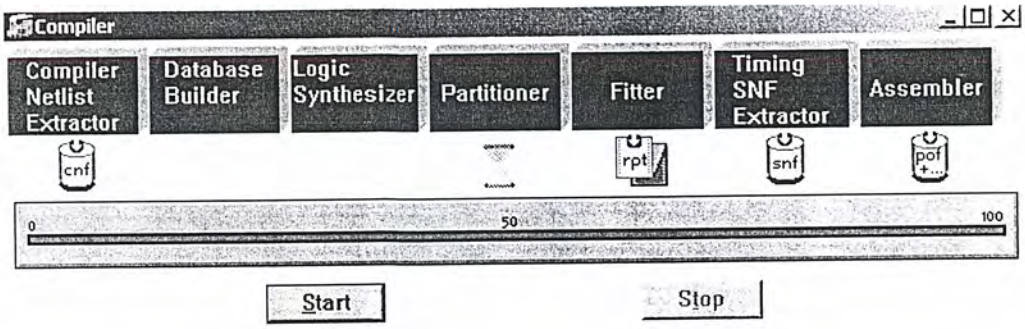
1. เรียกโปรแกรม MAX+plusII และเปิดไฟล์ da1.gdf



2. ทำการ Compile โดยก่อนทำการ Compile จะต้องทำการ Set Project to Current File โดยใช้เมนูคำสั่ง File | Project | Set Project to Current File และควรตรวจสอบเบอร์ชิปที่จะใช้งานก่อน ซึ่งในที่นี้คือชิปตระกูล Flex 10k เบอร์ EPF10K10LC84-4 โดยเลือกเมนูคำสั่ง Assign | Device

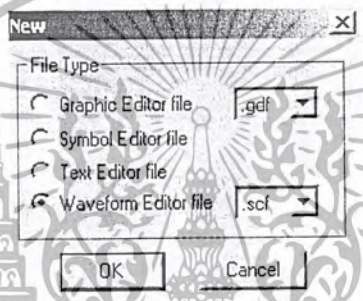


จากนั้นให้เลือกเมนูคำสั่ง MAX+plusII | Compiler จะปรากฏหน้าต่าง Compiler ออกมา จากนั้นกดปุ่ม start



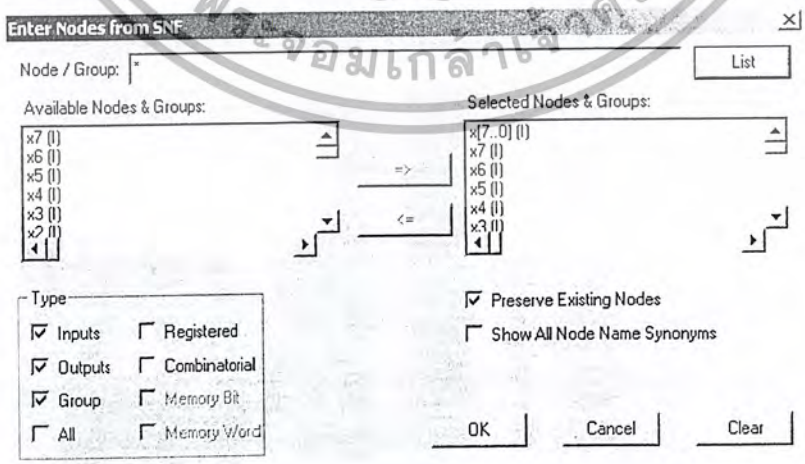
ทำการสร้าง Symbol เพื่อใช้ในการวาด Schematic โดยใช้เมนูคำสั่ง File | Create Default Symbol

3. ทำการจำลองการทำงาน โดยให้เลือกเมนูคำสั่ง File | New แล้วเลือก Waveform Editor file (.scf)



ทำการ save เป็นไฟล์ .scf

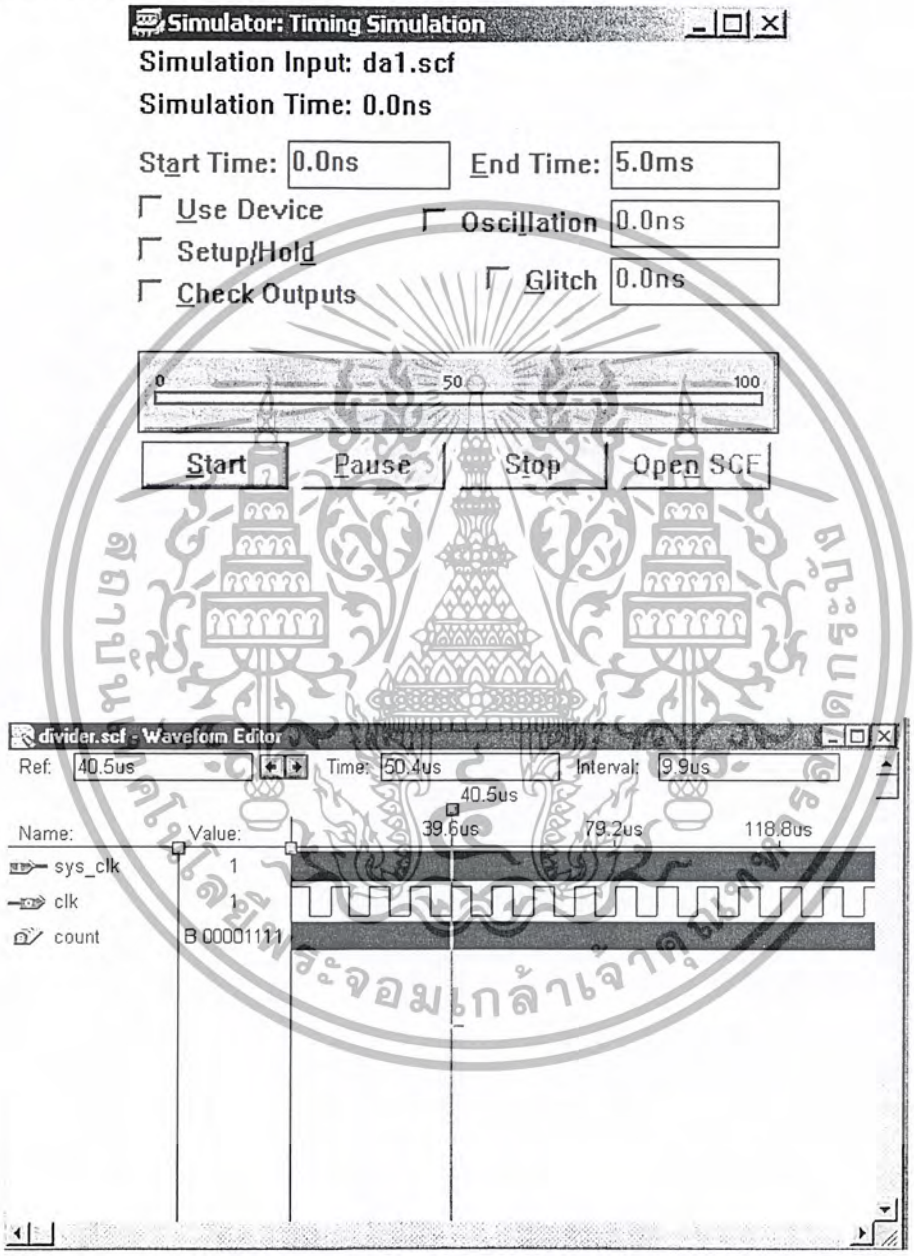
4. ทำการดึงสัญญาณที่ต้องการจำลองการทำงานเข้ามา โดยเลือกเมนูคำสั่ง Node | Enter Nodes From SNF จากนั้นคลิกที่ปุ่ม List เพื่อดูว่ามีสัญญาณอะไรบ้างโดยจะเห็นรายชื่อสัญญาณต่างๆอยู่ในหน้าต่างทางซ้ายมือ จากนั้นให้กดปุ่ม => เพื่อเลือกสัญญาณทั้งหมด แล้วคลิก OK



5.กำหนดค่า End Time ของ Waveform Editor เป็น 5 ms โดยใช้เมนูคำสั่ง File | End Time จากนั้นทำการกำหนด Grid Size ให้มีขนาด 156.2 ns โดยเลือกเมนูคำสั่ง Options | Grid Size

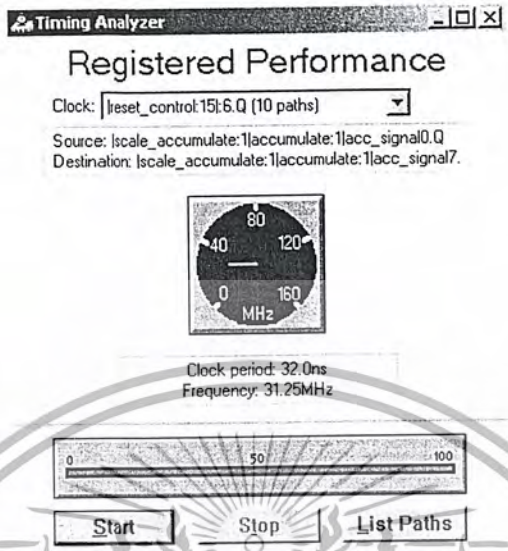
6. ทำการสร้างสัญญาณอินพุท คือ sys_clk โดยคลิกที่สัญญาณ sys_clk ให้ขึ้นแถบสีดำ แล้วเลือกเมนูคำสั่ง Edit | Overwrite | Clock โดยให้ค่า Multiplied By เท่ากับ 1

7.เรียกหน้าต่าง Simulator ขึ้นมา โดยเลือกเมนูคำสั่ง MAX+plus II | Simulator จากนั้นคลิกปุ่ม Start โปรแกรมจะทำการจำลองการทำงานตั้งแต่เวลาเท่ากับ 0 ถึง 5 ms



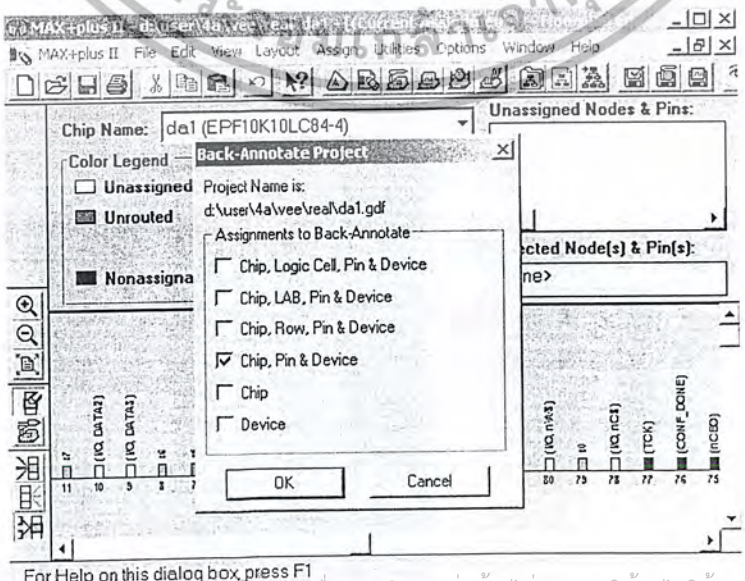
8. การตรวจสอบความเร็วสูงสุดที่ทำงานได้และปริมาณลอจิกเซลที่ใช้ไป โดยการตรวจสอบDelay ที่โหนดต่างๆ จะใช้หน้าต่าง Timing Analyzer โดยเลือกเมนูคำสั่ง MAX+plusII | Timing Analyzer จากนั้นคลิก

start และค่าความถี่สูงสุดที่วงจรยังคงทำงานได้อย่างถูกต้อง สามารถดูได้โดยเลือกเมนูคำสั่ง Analysis | Register Performance แล้วคลิกปุ่ม start

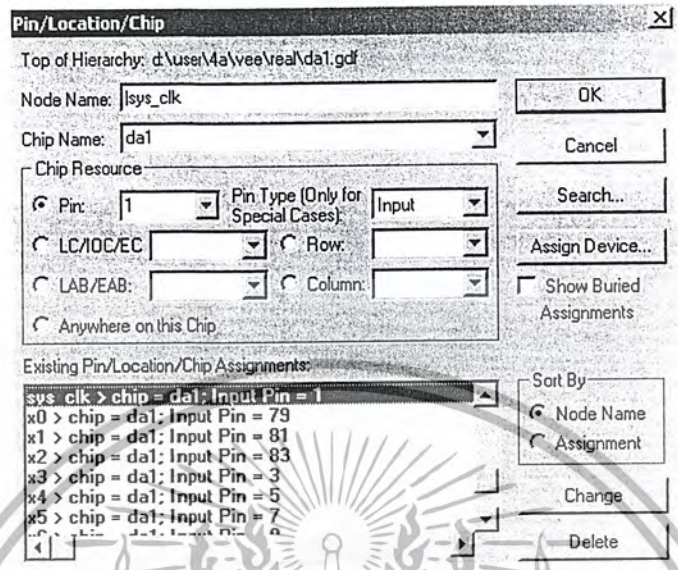


ส่วนการบอกถึงจำนวนขาที่ใช้, Pin diagram และปริมาณรวมถึงการใช้งานลอจิกเซล ดูได้ใน report file ซึ่งอยู่ในหน้าต่าง Compiler โดยดับเบิ้ลคลิกที่ไอคอน  ซึ่งโปรแกรมจะเปิดไฟล์ firlp16tap.rpt ขึ้นมาให้ตรวจสอบดูว่าได้ใช้ขา I/O และใช้ลอจิกเซลไปกี่เปอร์เซ็นต์ของทั้งหมด

9.ทำการเปลี่ยนตำแหน่งขา I/O โดยเลือกเมนูคำสั่ง MAX+plusII | Floorplan Editor เพื่อเปิดหน้าต่าง Floorplan Editor ขึ้นมา จากนั้นเลือก Layout | Device View และ Layout | Last Complilation Floorplan ซึ่งจะเห็น pin diagram ของชิปที่ใช้ โดยโปรแกรมจะทำการเลือกให้เราโดยอัตโนมัติ ถ้าต้องการเปลี่ยนตำแหน่งขาที่ใช้งานเอง สามารถทำได้โดยเลือก Layout | Current Assignment Floorplan จากนั้นทำการเลือก Assign | Back-Annotate Project แล้วเลือกหัวข้อ Chips, Pin & Device จากนั้นกด OK



จากนั้นสามารถเปลี่ยนตำแหน่งขา I/O ได้ตามต้องการ โดยใช้เมาส์ลากจากขาเก่าไปวางไว้ที่ขาใหม่ หรือเลือกที่เมนู Assign | Pin/Location/Chip

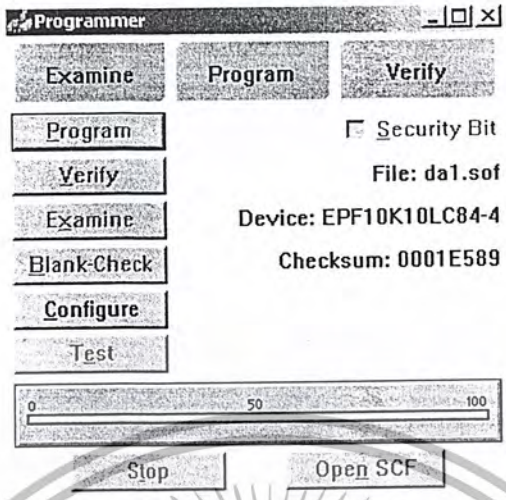


แล้วทำการเปลี่ยนขาใหม่ดังตาราง

Signal	Pin	Signal	Pin
sys_clk	1	s_c	10
in7	30	out7	54
in6	35	out6	53
in5	36	out5	52
in4	37	out4	51
in3	38	out3	50
in2	39	out2	49
in1	42	out1	48
in0	44	out0	47

จากนั้นทำการ Compile ใหม่อีกครั้งก่อนที่จะนำไปใช้

10.การโปรแกรมลงชิปทำได้โดยเลือกเมนูคำสั่ง MAX+plusII | Programmer



จากนั้นเลือกเมนูคำสั่ง JTAG | Multi-Device JTAG Chain Setup ทำการเลือก JTAG Device Attributes เป็น EPF10K10 กด OK และเลือก Select Programming file เป็น da1.sof กด OK จะปรากฏสิ่งที่เลือกที่ Device Name และ Programming File Name แล้วกดเลือก ADD

จากนั้นตรวจสอบว่าโปรแกรม MAX+plusII มองเห็นตัวชิปหรือไม่ โดยคลิกตรวจสอบได้ที่ Detect JTAG Chain Info หลังจากนั้นกด Configure เพื่อโปรแกรมข้อมูลลงบนชิป

4.6 ผลการออกแบบวงจรกรองสัญญาณเชิงเลขที่ได้ด้วย Altera FPGA เบอร์ EPF10K10LC84-4

ผลการออกแบบวงจรกรองสัญญาณเชิงเลขที่ได้ด้วย Altera FPGA เบอร์ EPF10K10LC84-4 แสดงจำนวนของ สัญญาณอินพุท เอาท์พุท ที่ใช้ รวมทั้งจำนวนของลอจิกเซลล์ ที่ใช้งานทั้งหมด แสดงได้ดังนี้

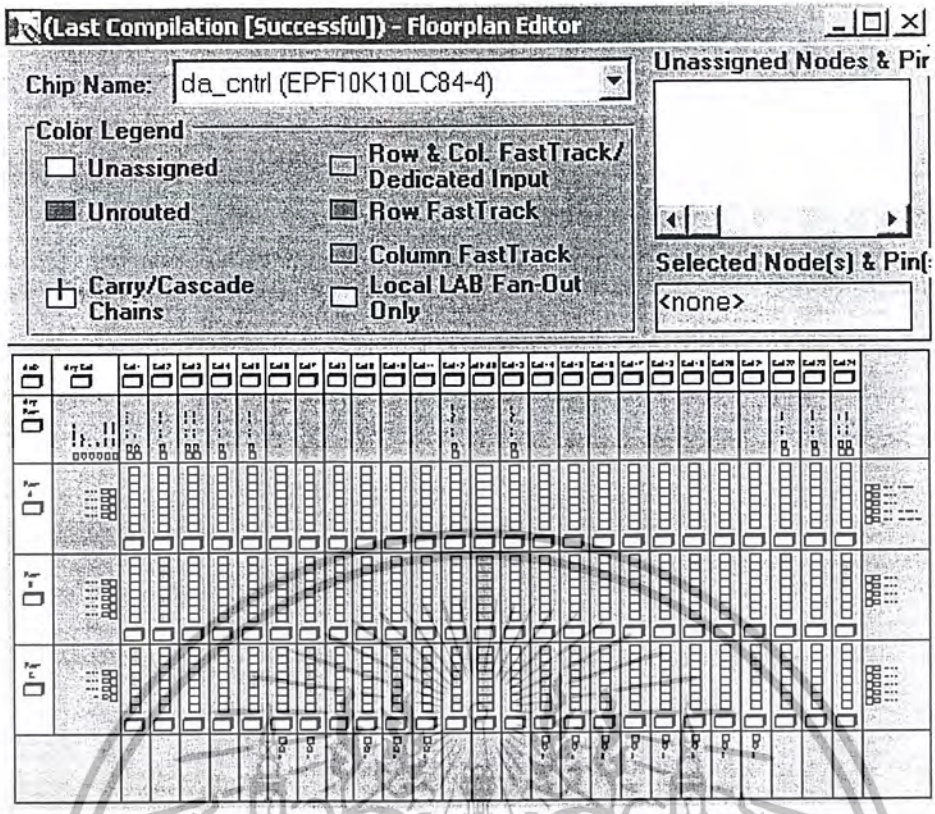
4.6.1 โครงสร้างแบบ Controllable Canonical form

da_cntrl.rpt - Text Editor

```
** DEVICE SUMMARY **

Chip/                                Input  Output  Bidir  Memory  Memory              LCs
POF       Device                     Pins   Pins    Pins   Bits    % Utilized  LCs     % Utilized

da_cntrl  EPF10K10LC84-4             9      9       0      128     2 %         112     19 %

User Pins:                           9      9       0

Line 1    Col 1    INS
```

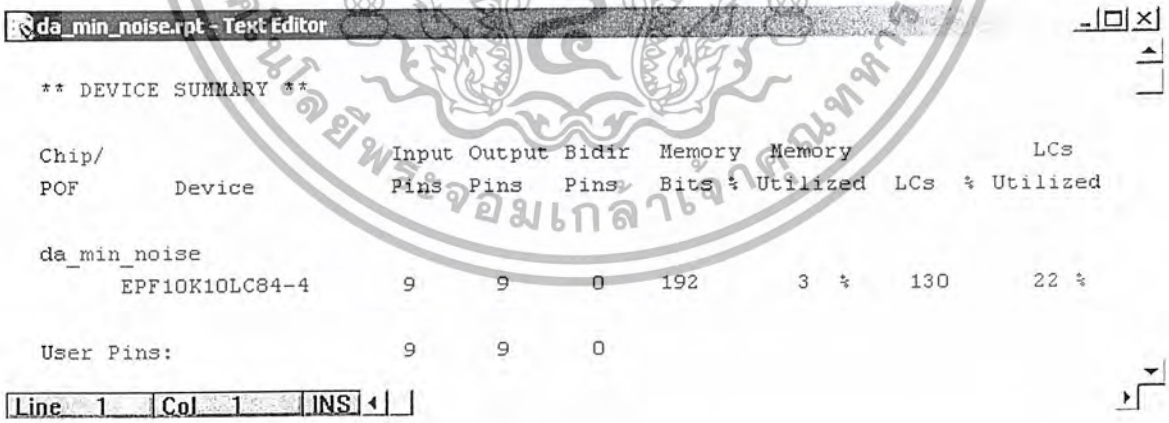
รูปที่ 4.37 จำนวนของลอจิกเซลล์ที่ใช้งานทั้งหมดของโครงสร้างแบบ Controllable Canonical form

รูปที่ 4.38 ความเร็วสูงสุดของโครงสร้างแบบ Controllable Canonical form

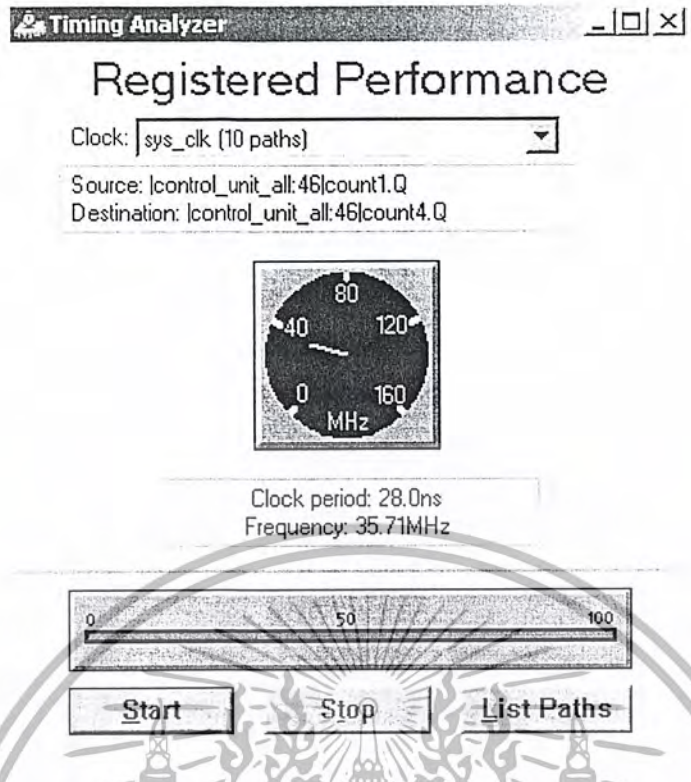


รูปที่ 4.39 ปริมาณ logic cells ที่ใช้งานทั้งหมดของโครงสร้างแบบ Controllable Canonical form

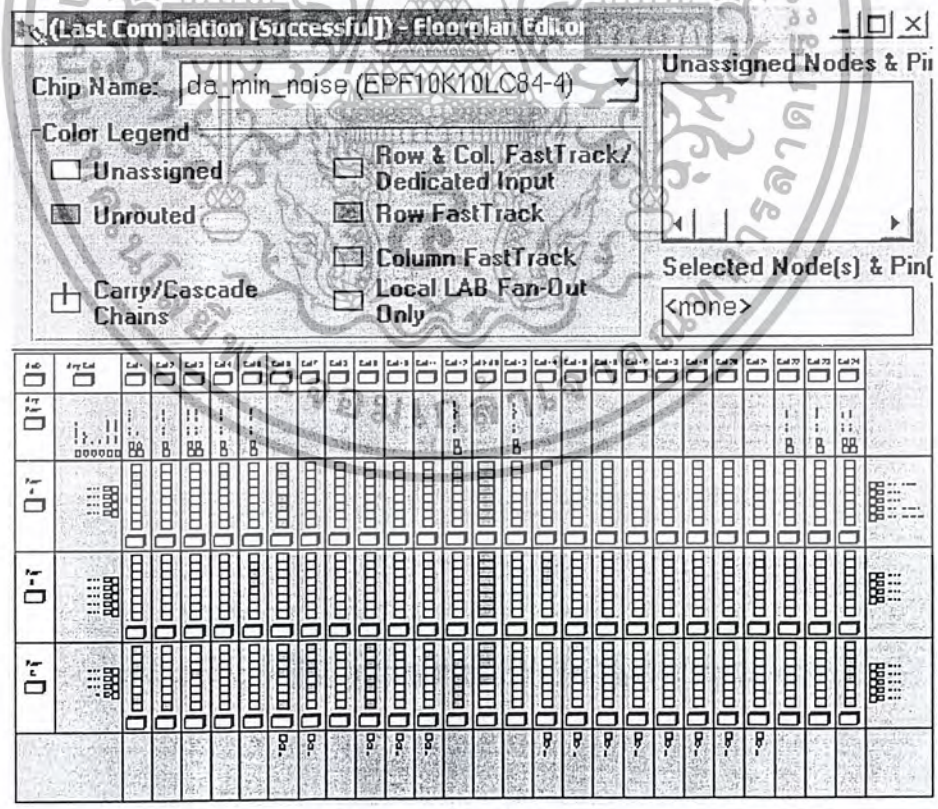
4.6.2 โครงสร้างแบบ Minimum Noise Structure



รูปที่ 4.40 จำนวนของลอจิกเซลล์ ที่ใช้งานทั้งหมดของโครงสร้างแบบ Minimum Noise Structure



รูปที่ 4.41 ความเร็วสูงสุดของโครงสร้างแบบ Minimum Noise Structure



รูปที่ 4.42 ปริมาณ logic cells ที่ใช้งานทั้งหมดของโครงสร้างแบบ Minimum Noise Structure

บทที่ 5
การทดลองและผลการทดลอง

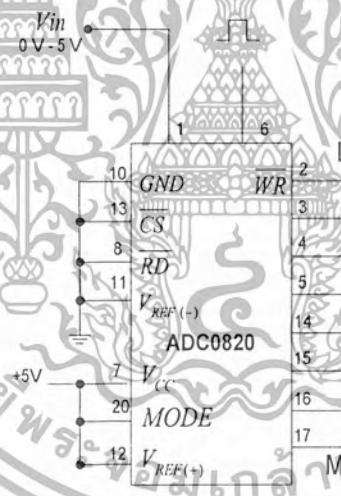
ในการทดลองจะแบ่งออกเป็น 2 ส่วน คือ ส่วนของการทดลองของโครงสร้างแบบ Controllable Canonical form และโครงสร้างแบบ Minimum Noise โดยในแต่ละโครงสร้างจะแบ่งการทดลองออกเป็น 2 ส่วน คือ จะเป็นการทดลองเพื่อดูถึงคุณลักษณะของวงจรกรองสัญญาณที่ออกแบบขึ้นว่าไปตามที่ออกแบบหรือไม่ การดูถึงผลการทำงานของวงจรที่ออกแบบขึ้นเทียบกับผลการจำลองการทำงานจากโปรแกรม Matlab

โดยในการทดลองจะมีอุปกรณ์ที่ใช้อยู่ 3 ส่วนดังนี้

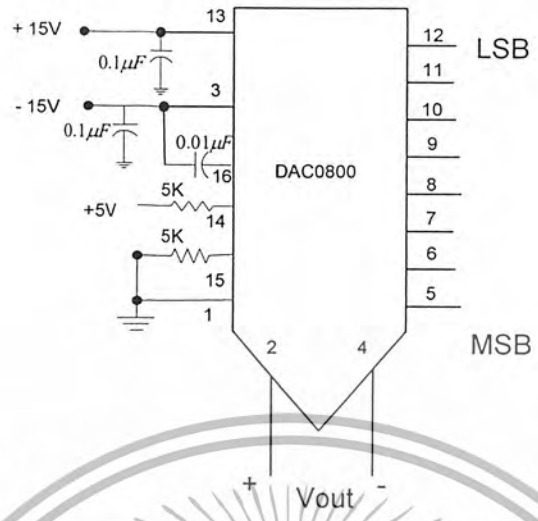
1.วงจรแปลงสัญญาณเชิงอนาลอก (Analog) เป็นสัญญาณดิจิตอล (Digital) หรือ Analog To Digital Converter (ADC)

2.วงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณเชิงอนาลอก หรือ Digital To Analog Converter (DAC)

3.อุปกรณ์ FPGA ของบริษัท Alter ตระกูล FLEX 10k เบอร์ EPF10K10LC84-4 โดยวงจรแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล และ วงจรแปลงสัญญาณเลขเป็นสัญญาณเชิงอนาลอกนี้



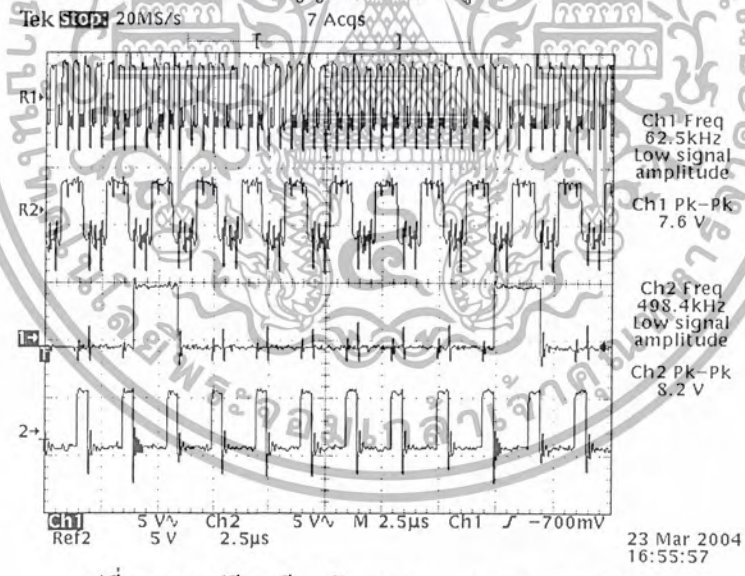
รูปที่ 5.1 วงจรแปลงสัญญาณเชิงอนาลอกเป็นสัญญาณดิจิตอล



รูปที่ 5.2 วงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณเชิงอนาลอก

5.1 ผลการออกแบบการทดลองของสัญญาณควบคุมภายใน

ภายในโครงสร้างแบบ Controllable Canonical Form และโครงสร้างแบบ Minimum Noise โดยในทั้ง 2 โครงสร้างนี้ จะมีสัญญาณควบคุมภายในเหมือนกัน ซึ่งประกอบด้วยสัญญาณ sys_clk, clk, s_a, lacc, lr, clacc และ sc ซึ่งจะแสดงผลการเปรียบเทียบสัญญาณเหล่านี้ ดังรูปที่ 5.3 ถึง 5.6



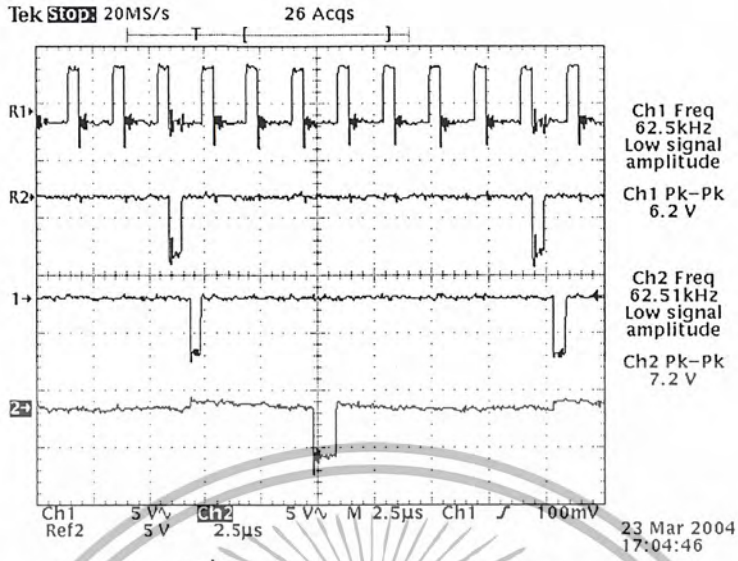
รูปที่ 5.3 การเปรียบเทียบสัญญาณ sys_clk, clk, s_a และ lacc

CH R1 เป็นสัญญาณ sys_clk

CH R2 เป็นสัญญาณ clk

CH 1 เป็นสัญญาณ s_a

CH 2 เป็นสัญญาณ lacc



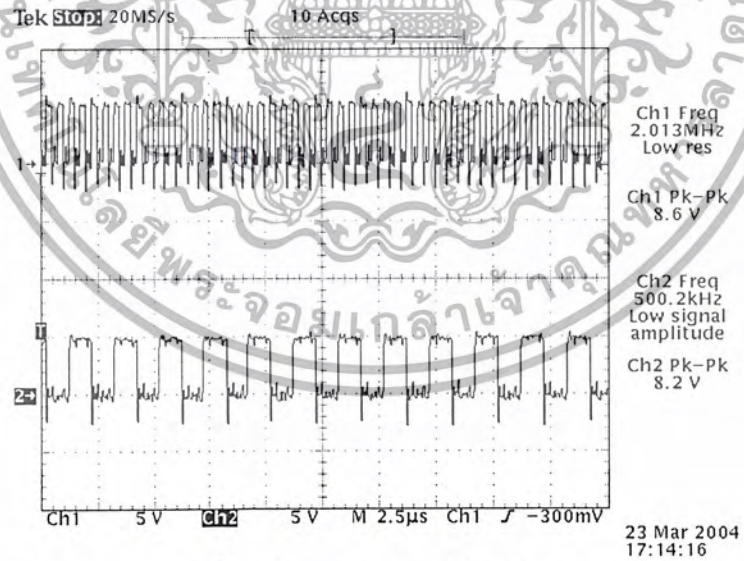
รูปที่ 5.4 สัญญาณ lacc, lr, clacc และ sc

CH R1 เป็นสัญญาณ lacc

CH R2 เป็นสัญญาณ lr

CH 1 เป็นสัญญาณ clacc

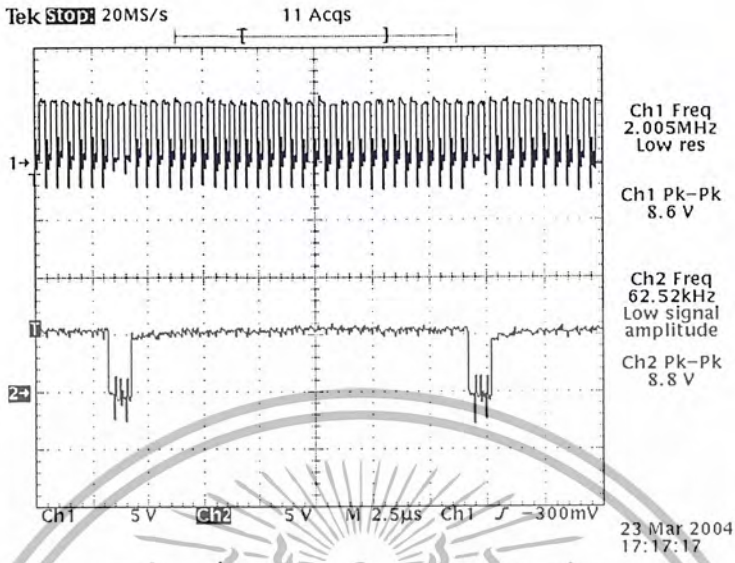
CH 2 เป็นสัญญาณ sc



รูปที่ 5.5 สัญญาณ sys_clk เทียบกับ clk

CH 1 เป็นสัญญาณ sys_clk

CH 2 เป็นสัญญาณ clk



รูปที่ 5.6 สัญญาณ sys_clk เทียบกับ sc

CH 1 เป็นสัญญาณ sys_clk

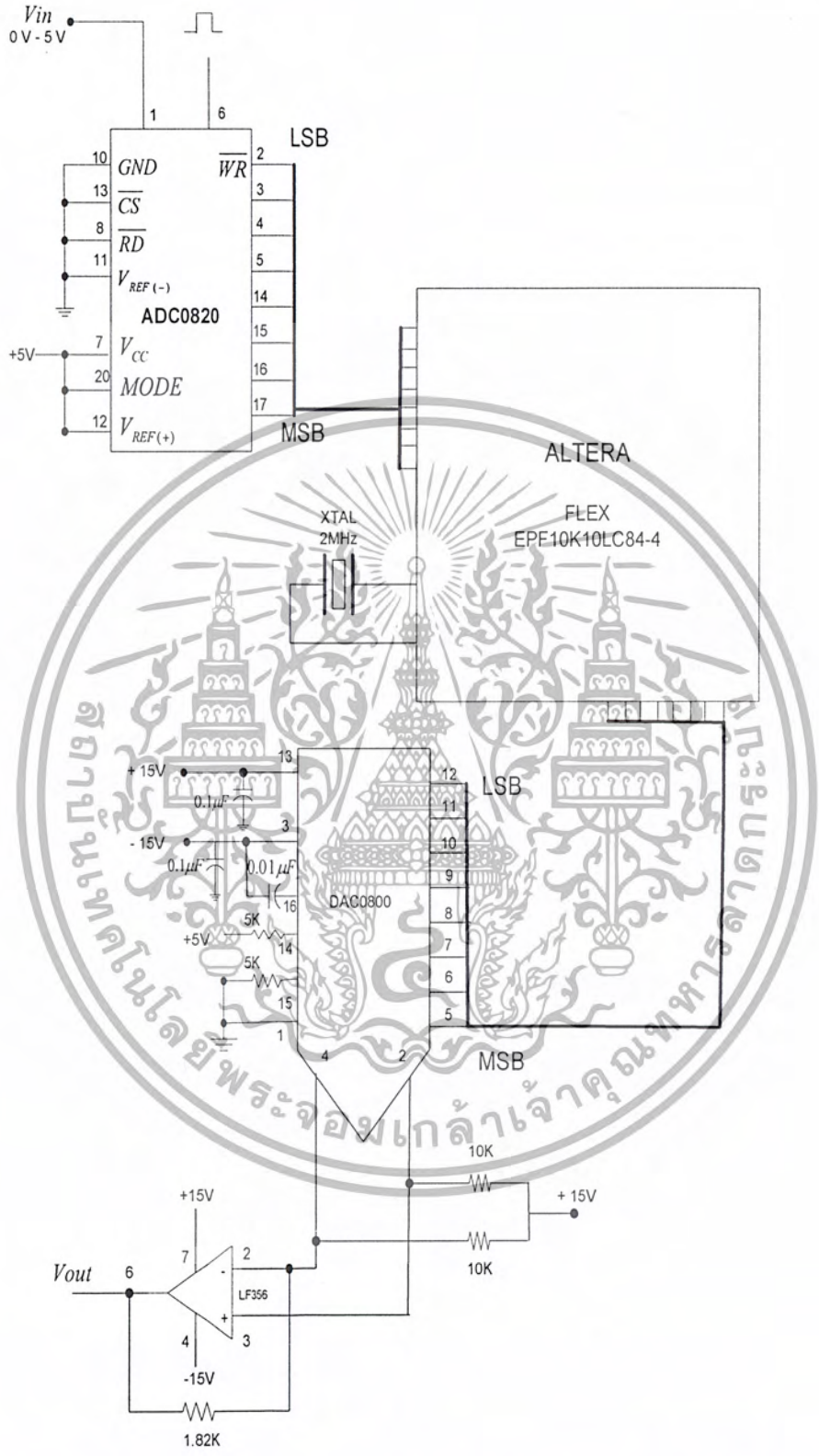
CH 2 เป็นสัญญาณ sc

5.2 การออกแบบการทดลองสำหรับโครงสร้างแบบ Controllable Canonical Form

การทดลองในส่วนโครงสร้างแบบ Controllable Canonical Form จะประกอบไปด้วย 2 ส่วน คือ การทดลองเพื่อวัดคุณลักษณะของวงจร และการศึกษาถึงผลการทำงานของวงจรที่ออกแบบขึ้นเทียบกับผลการจำลองการทำงานจากโปรแกรม Matlab

5.2.1 การออกแบบการทดลองสำหรับการวัดคุณลักษณะของวงจรกรองสัญญาณที่ออกแบบ

เราจะนำวงจรเปลี่ยนสัญญาณอนาลอกเป็นสัญญาณดิจิตอลมาต่อกับวงจร FPGA เพื่อที่จะทำให้สัญญาณอนาลอกถูกเปลี่ยนให้อยู่ในรูปสัญญาณดิจิตอลจึงส่งเข้าไปให้ FPGA ซึ่งทำหน้าที่เป็นวงจรกรองสัญญาณ แล้วจึงใช้วงจรดิจิตอลเปลี่ยนเป็นอนาลอกเปลี่ยนกลับอีกครั้งหนึ่งเพื่อให้ได้สัญญาณกลับมาอยู่ในรูปอนาลอกที่เอาท์พุตอีกครั้งหนึ่ง สำหรับการวัดคุณสมบัติของวงจรกรองสัญญาณที่ออกแบบขึ้นสามารถแสดงการเตรียมอุปกรณ์ ได้ดังรูปที่ 5.7

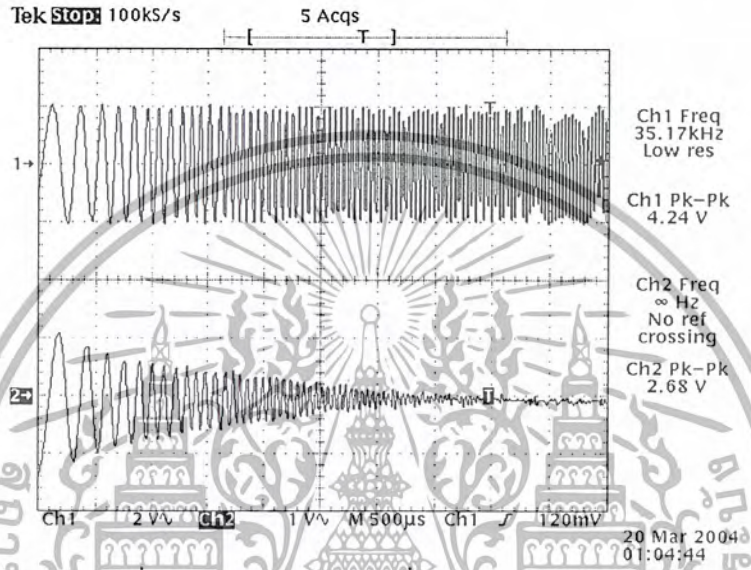


รูปที่ 5.7 การเตรียมอุปกรณ์สำหรับการทดลองวัดคุณลักษณะของวงจรกรองสัญญาณที่ออกแบบขึ้น

5.2.2 ผลการทดลองและการวัดคุณลักษณะของวงจรกรองสัญญาณที่ออกแบบ

1. ผลการ sweep ความถี่เพื่อแสดงการทำงานของวงจรกรองสัญญาณที่ออกแบบ

ในการทดลองส่วนนี้จะทำการป้อนสัญญาณอินพุตเป็นสัญญาณ Sine แล้ว Sweep ความถี่จาก 100 Hz ถึง 31.25 kHz ขยายในช่วงเวลา 5 ms โดยป้อนสัญญาณอินพุตด้วยขนาด 4 Vpp สามารถแสดงผลการทดลองได้ดังรูป



รูปที่ 5.8 ผลการ Sweep ความถี่

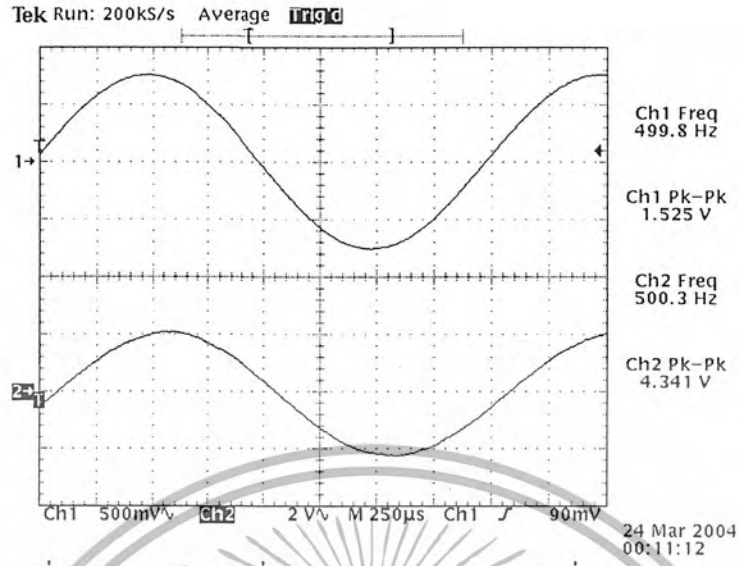
CH1 input ที่ป้อนให้กับวงจร

CH2 output ที่ออกจากวงจร

จากรูปที่ 5.8 จะเห็นได้ว่าวงจรกรองสัญญาณที่ออกแบบขึ้นเป็นวงจรกรองสัญญาณความถี่ต่ำผ่าน เนื่องจากเมื่อความถี่ของสัญญาณอินพุตสูงขึ้น ขนาดของสัญญาณเอาท์พุตจะมีขนาดลดลง

2. ผลการทดลองปรับความถี่สัญญาณอินพุตไปที่ค่าต่าง ๆ

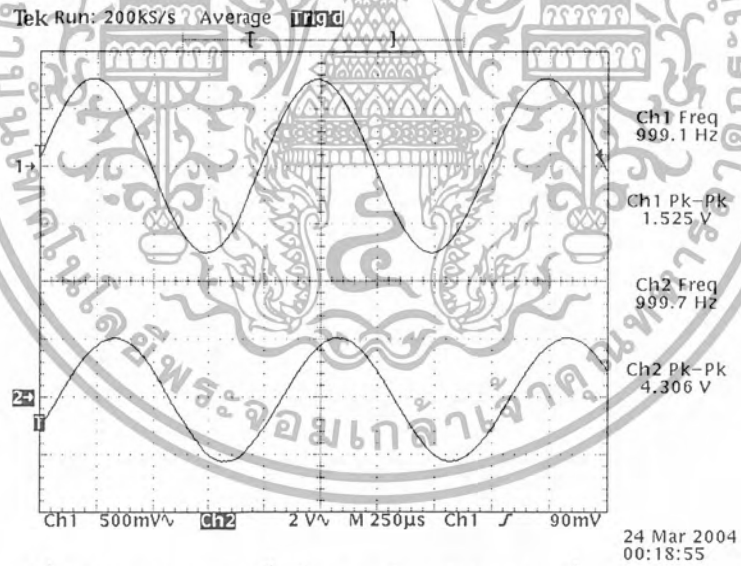
ในการทดลองนี้จะทำการป้อนสัญญาณอินพุตเป็นสัญญาณ sine ขนาด 1.5Vpp แล้วทำการปรับความถี่ของสัญญาณอินพุตไปที่ค่าต่าง ๆ เพื่อดูถึงขนาดของสัญญาณเอาท์พุตนี้ได้แต่ละความถี่ รวมทั้งค่าความต่างเฟสระหว่างสัญญาณอินพุตและสัญญาณเอาท์พุตด้วย เพื่อใช้ในการ plot ผลตอบสนองทางความถี่ของวงจรกรองสัญญาณ ได้ผลการทดลองดังนี้



รูปที่ 5.9 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 500 Hz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

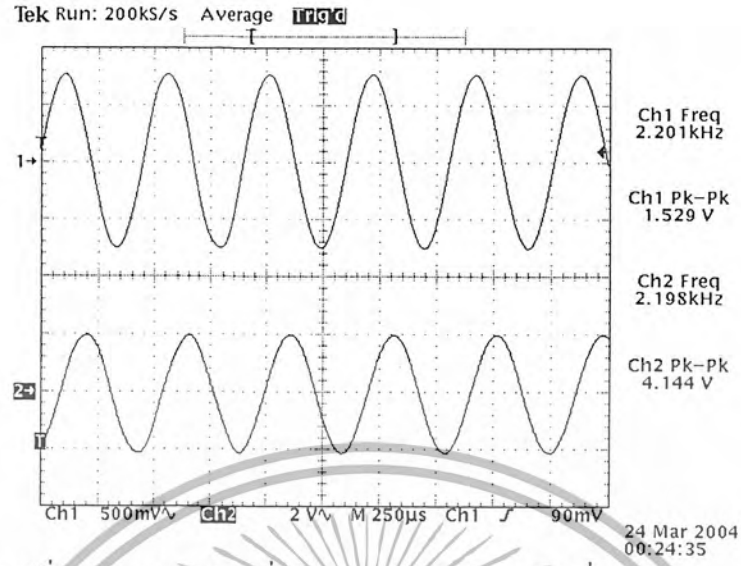
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.10 ผลการทดลองเมื่อป้อน แรงดัน 1.5Vpp ความถี่ 1 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

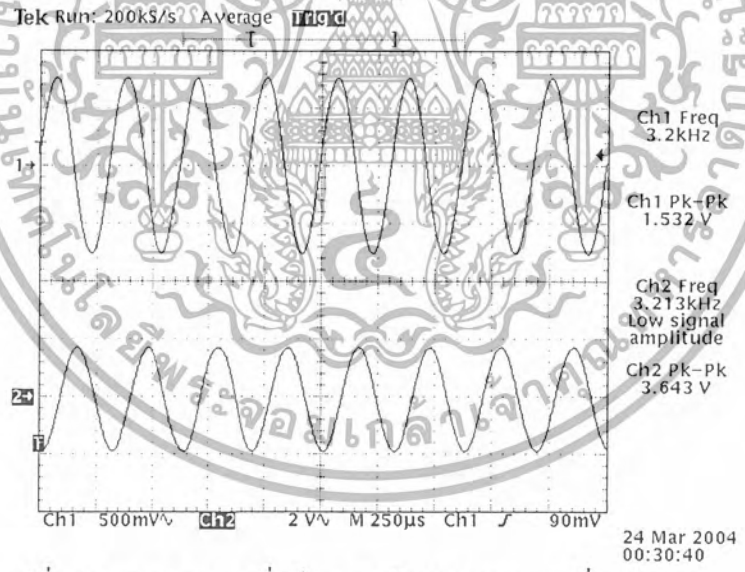
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.11 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 2.2 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

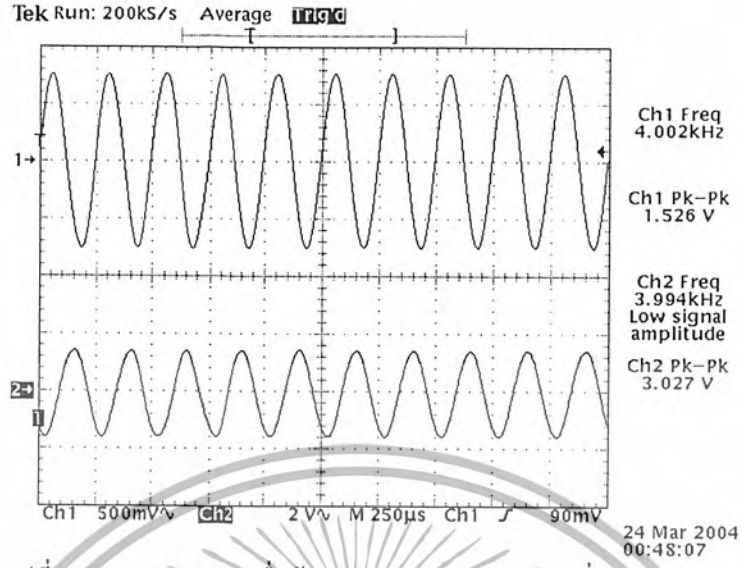
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.12 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 3.2 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

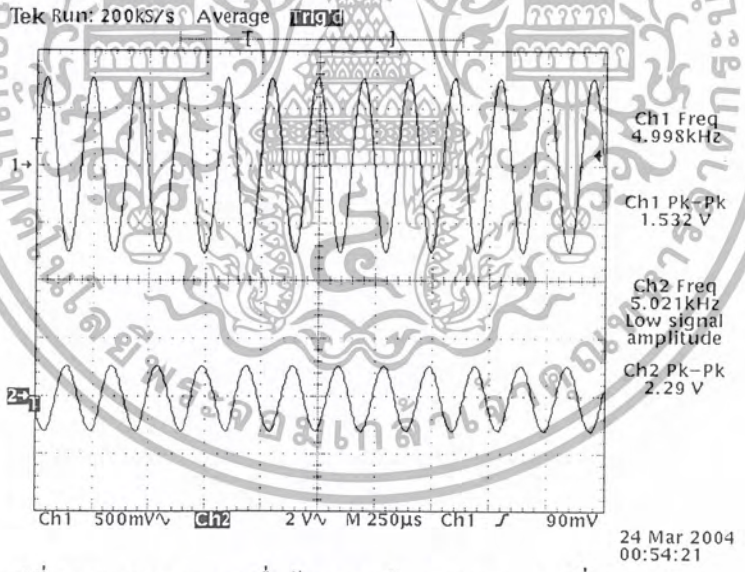
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.13 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 4 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

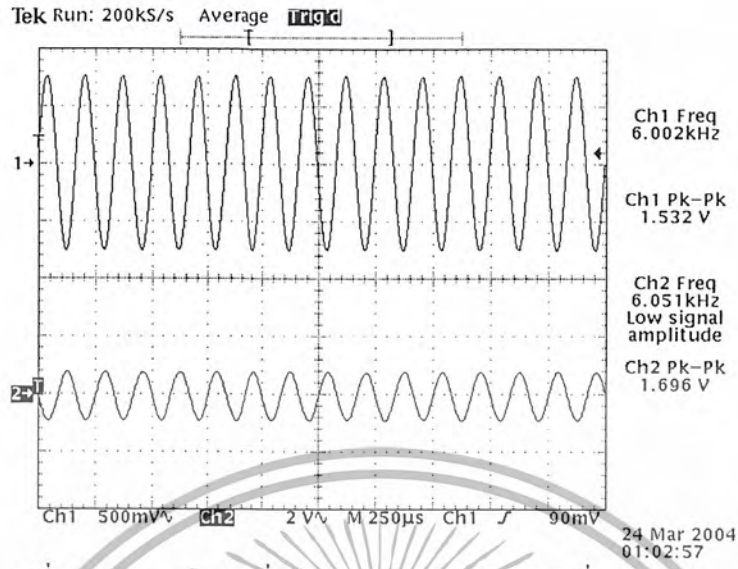
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.14 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 5 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

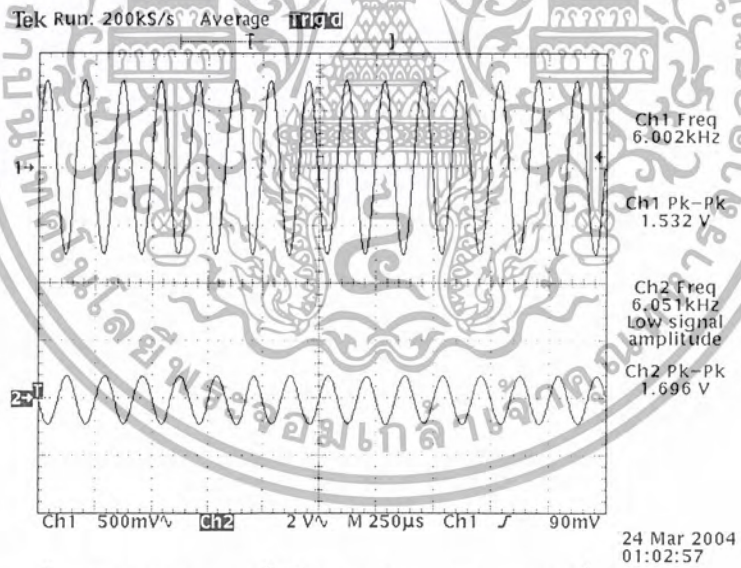
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.15 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 6 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

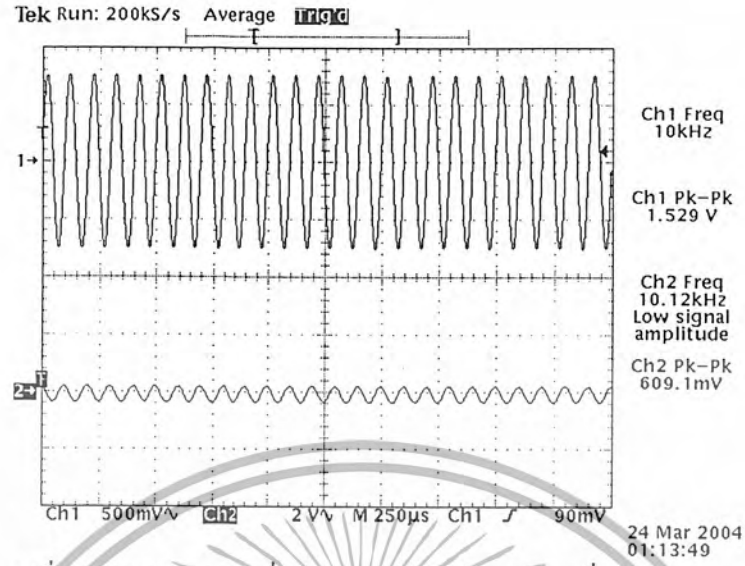
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.16 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 7.5 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.17 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 10 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

รูปที่ 5.18 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 20 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

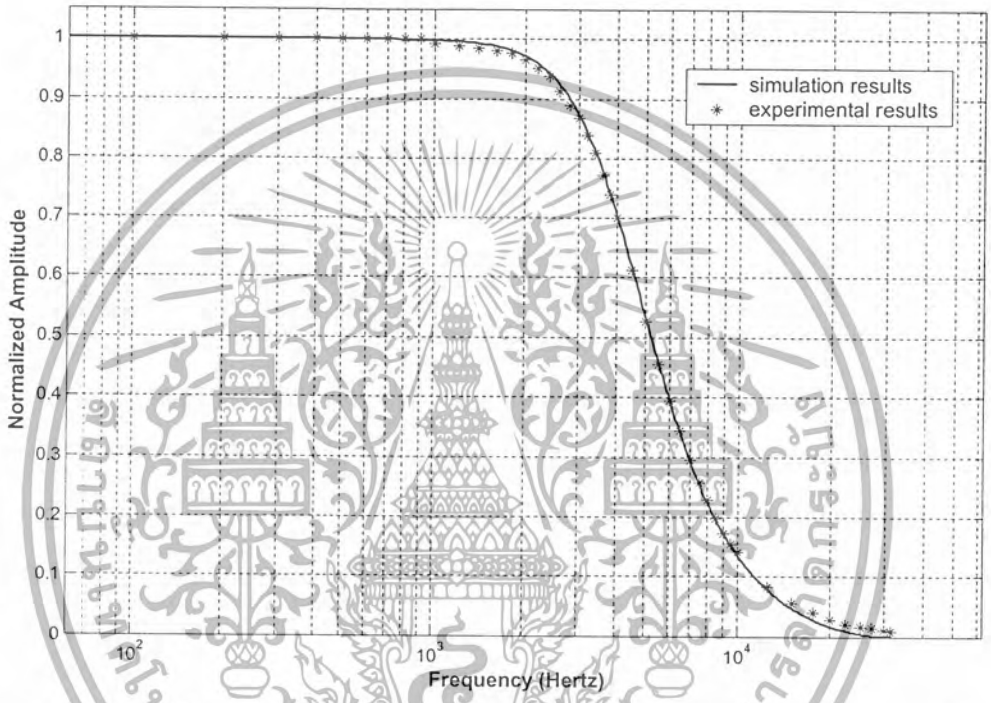
5.2.3 ผลตอบสนองความถี่ของวงจรกรองสัญญาณที่ออกแบบ

ในการทดลองนี้จะใช้ผลที่ได้จากการทดลอง 5.2.2.2 ในการ plot ผลตอบสนองทางความถี่ของวงจร ซึ่งสามารถแสดงได้ดังนี้

ตารางที่ 5.1 ค่าแรงดันของสัญญาณเอาท์พุตที่อ่านได้ ณ ความถี่ต่าง ๆ

Input Frequency Hz	Output Amplitude Vpp	Normalized Amplitude	Input Frequency Hz	Output Amplitude Vpp	Normalized Amplitude
100	4.34	1.0000	3800	3.21	0.7396
200	4.34	1.0000	4000	3.04	0.7005
300	4.34	1.0000	4500	2.66	0.6129
400	4.34	1.0000	5000	2.29	0.5276
500	4.34	1.0000	5500	1.97	0.4539
600	4.34	1.0000	6000	1.71	0.3940
700	4.34	1.0000	6500	1.49	0.3433
800	4.34	1.0000	7000	1.28	0.2949
900	4.34	1.0000	7500	1.12	0.2581
1000	4.31	0.9931	8000	0.99	0.2281
1200	4.29	0.9885	8500	0.86	0.1982
1400	4.27	0.9839	9000	0.75	0.1728
1600	4.26	0.9816	9500	0.67	0.1544
1800	4.24	0.9770	10000	0.61	0.1406
2000	4.19	0.9654	12500	0.36	0.0829
2200	4.13	0.9516	15000	0.24	0.0553
2400	4.07	0.9378	17500	0.17	0.0392
2600	3.96	0.9124	20000	0.12	0.0276
2800	3.86	0.8894	22500	0.09	0.0207
3000	3.77	0.8687	25000	0.07	0.0161
3200	3.64	0.8387	27500	0.06	0.0138
3400	3.51	0.8088	30000	0.05	0.0115
3600	3.35	0.7719	31250	0.04	0.0092

จากตารางที่ 5.1 จะแสดงค่าของแรงดันของสัญญาณเอาท์พุตที่อ่านได้ ณ ความถี่ต่าง ๆ จากนั้นจะทำการ Normalize ค่าที่ได้จากการทดลอง เพื่อใช้ในการ Plot Amplitude Response ของวงจรกรองสัญญาณที่ได้นำมาเปรียบเทียบกับผลที่ได้จากการเลียนแบบด้วยโปรแกรม Matlab ของวงจรกรองสัญญาณดิจิตอลชนิดป้อนกลับอันดับที่ 2 ที่ความถี่ต่ำผ่าน f_c=4 kHz, f_s=62.5 kHz Normalized frequency = 0.128 π ซึ่งสามารถแสดงได้ดังรูป 5.19



รูปที่ 5.19 กราฟผลตอบสนองทางขนาดที่ได้จากการทดลองเปรียบเทียบกับผลที่ได้จากการเลียนแบบด้วยโปรแกรม Matlab

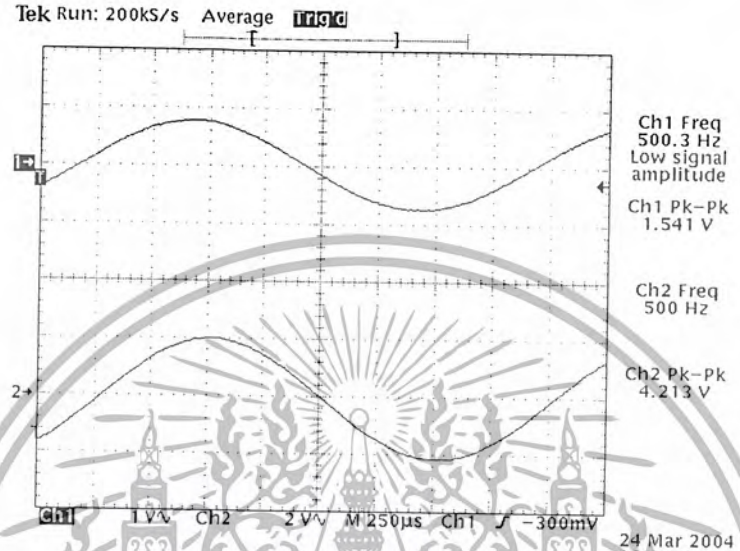
5.3 การออกแบบการทดลองสำหรับโครงสร้างแบบ Minimum Noise

การทดลองในส่วนโครงสร้างแบบ Minimum Noise จะประกอบไปด้วย 2 ส่วน คือ การทดลองเพื่อวัดคุณลักษณะของวงจร และการศึกษาถึงผลการทำงานของวงจรที่ออกแบบขึ้นเทียบกับผลการจำลองการทำงานจากโปรแกรม Matlab

5.3.1 ผลการทดลองและการวัดคุณลักษณะของวงจรกรองสัญญาณที่ออกแบบ

สำหรับการทดลองการวัดคุณสมบัติของวงจรกรองสัญญาณที่ออกแบบขึ้น สามารถแสดงการเตรียมอุปกรณ์ ได้ดังรูปที่ 5.7 เช่นเดียวกับการทดลองวัดคุณสมบัติของโครงสร้างแบบ Controllable Canonical Form

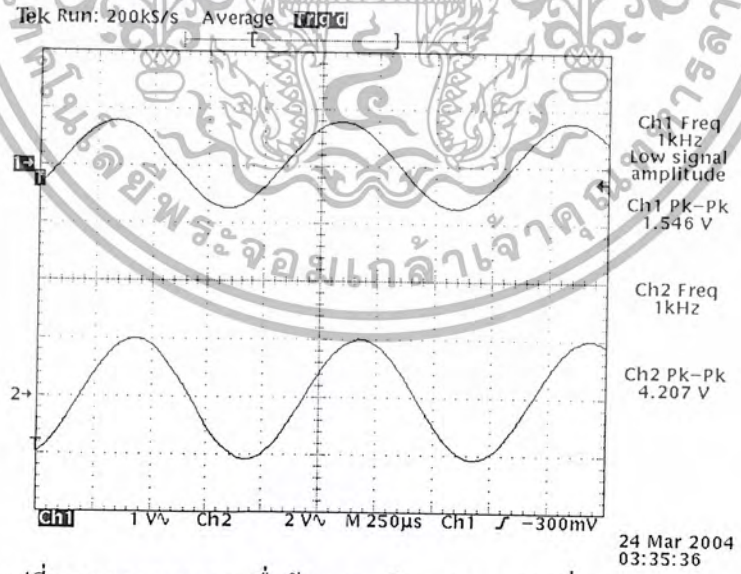
ในการทดลองนี้จะทำการป้อนสัญญาณอินพุตเป็นสัญญาณ sine ขนาด 1.5Vpp แล้วทำการปรับความถี่ของสัญญาณอินพุตไปที่ค่าต่าง ๆ เพื่อดูถึงขนาดของสัญญาณเอาท์พุตนี้ได้แต่ละความถี่ ระหว่าง เพื่อใช้ในการ plot ผลตอบสนองทางความถี่ของวงจรกรองสัญญาณ ได้ผลการทดลองดังนี้



รูป 5.20 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 500 Hz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

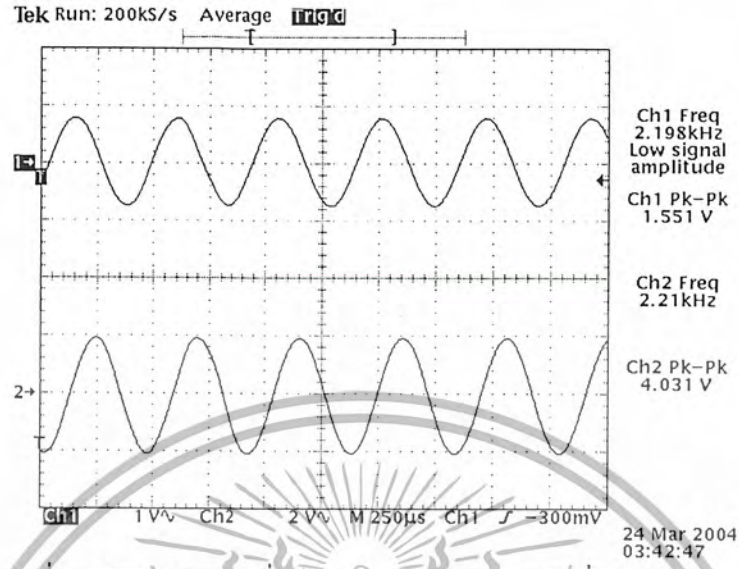
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.21 ผลการทดลองเมื่อป้อน แรงดัน 1.5Vpp ความถี่ 1 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

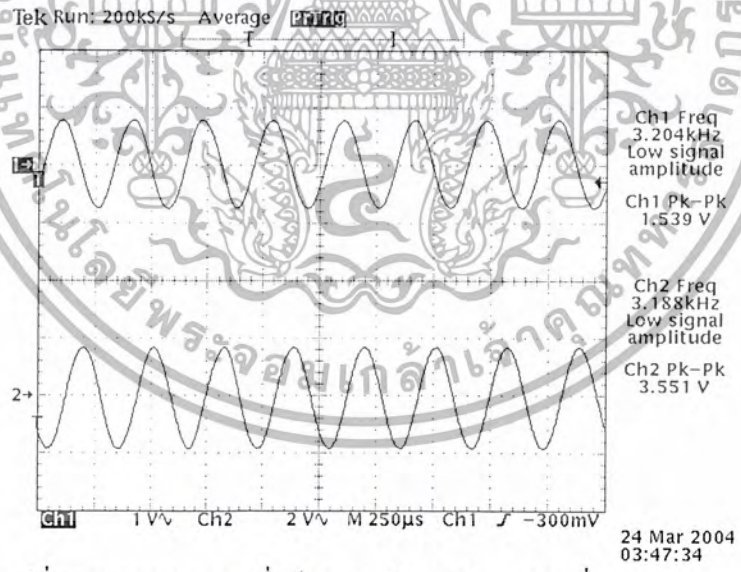
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.22 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 2.2 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

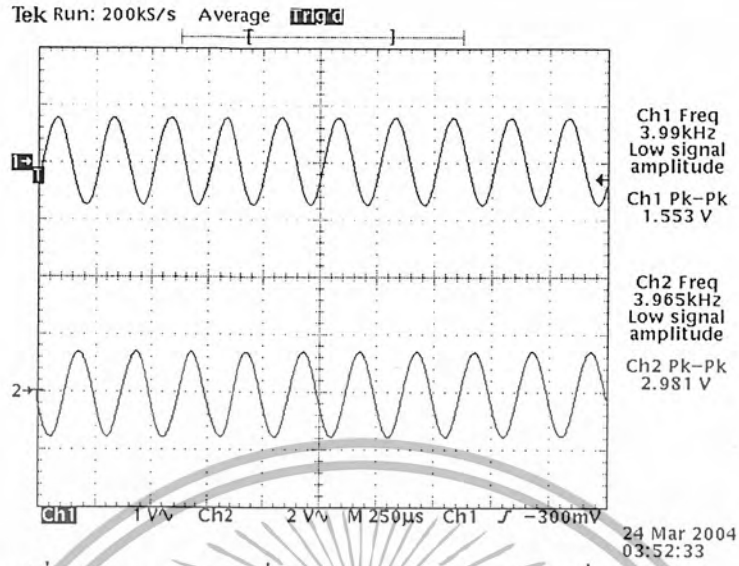
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.23 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 3.2 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

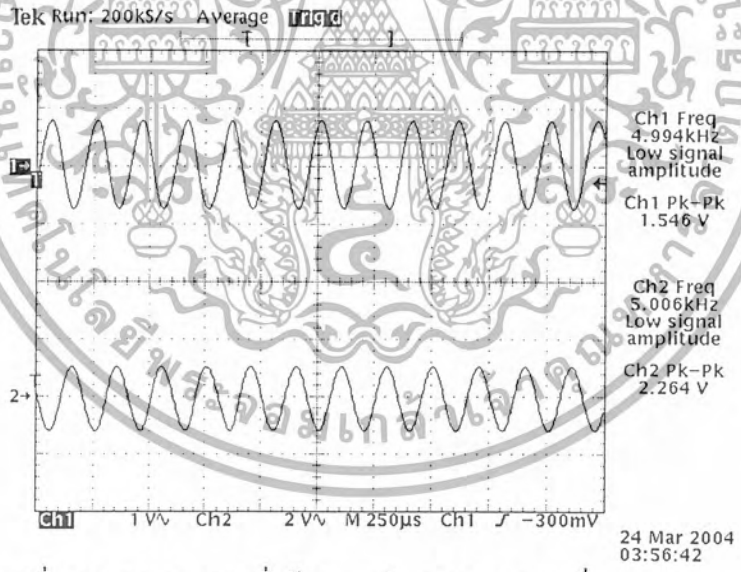
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.24 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 4 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

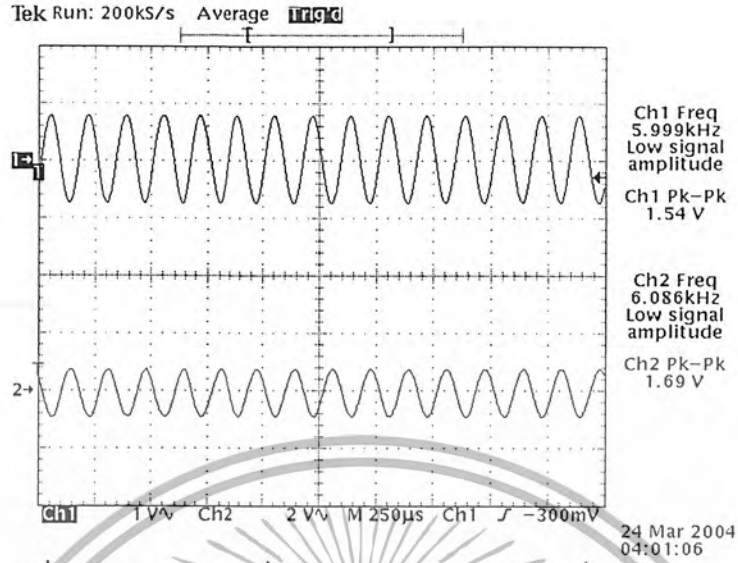
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.25 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 5 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

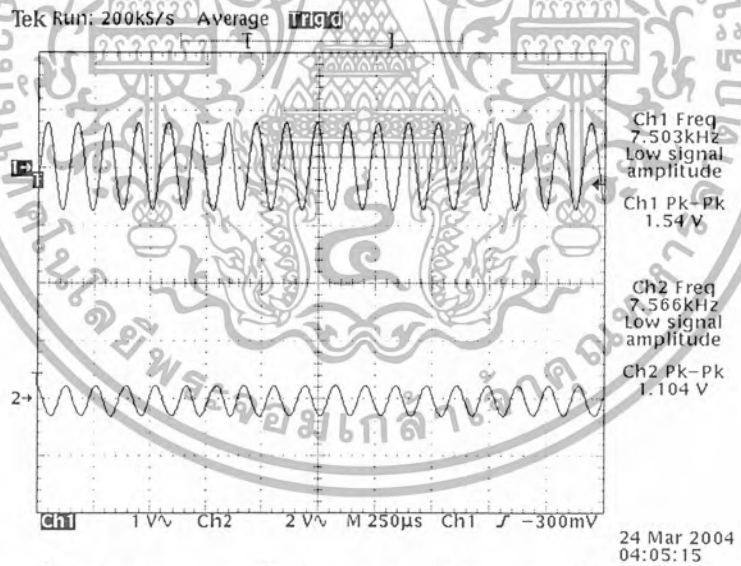
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.26 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 6 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

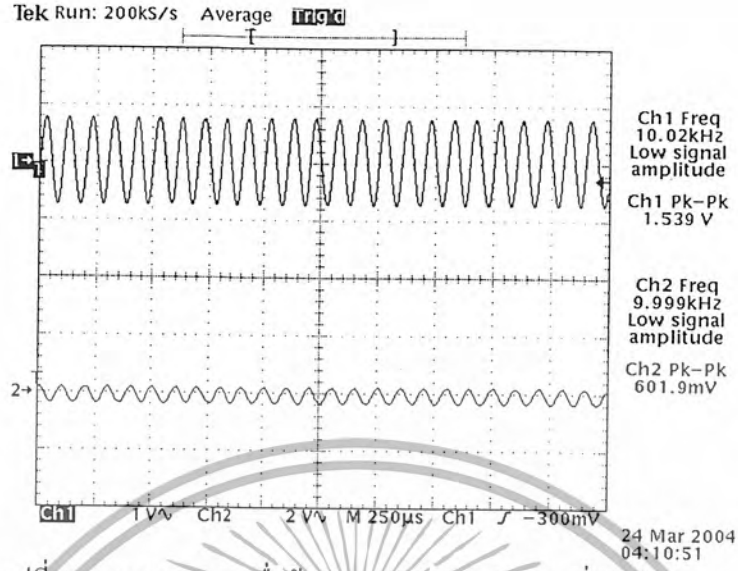
CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.27 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 7.5 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.28 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 10 kHz

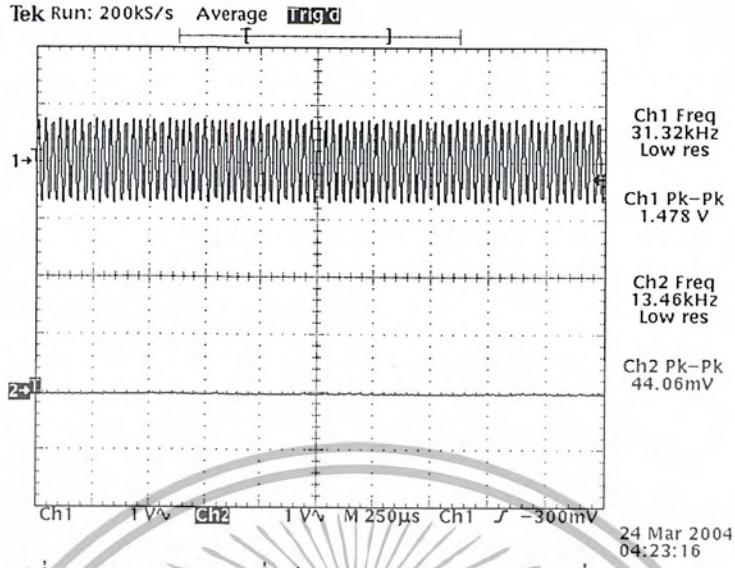
CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

รูปที่ 5.29 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 20 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



รูปที่ 5.30 ผลการทดลองเมื่อป้อน แรงดัน 1.5 Vpp ความถี่ 31.25 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

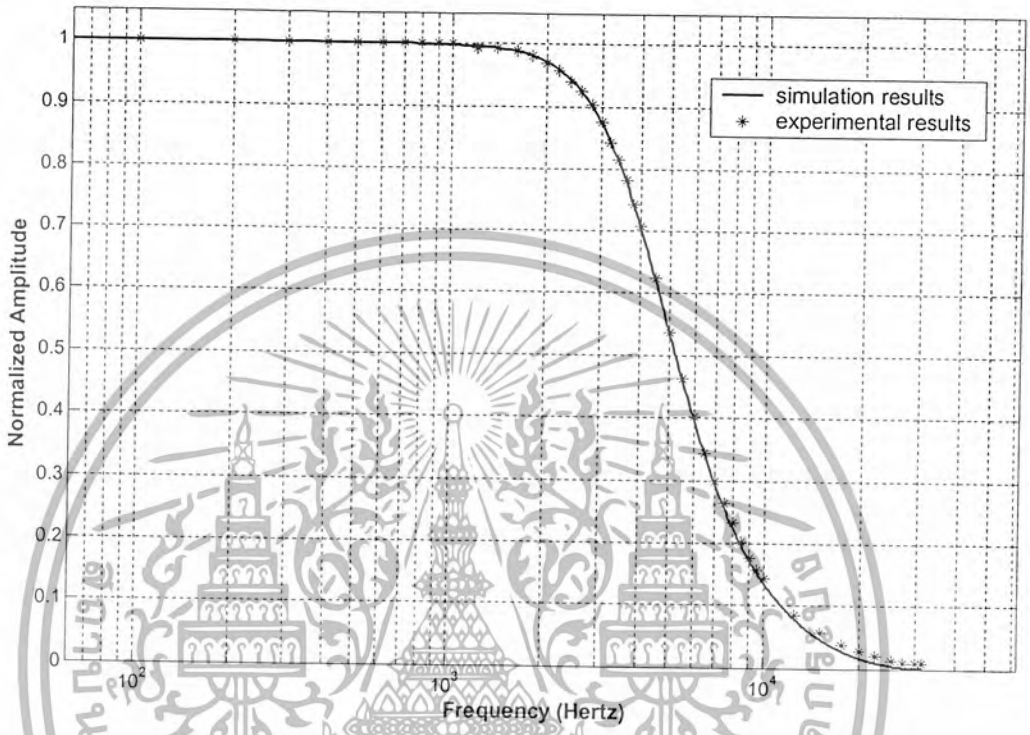
5.3.2 ผลตอบสนองความถี่ของวงจรกรองสัญญาณที่ออกแบบ

ในการทดลองนี้จะใช้ผลที่ได้จากการทดลอง 5.3.1 ในการ plot ผลตอบสนองทางความถี่ของวงจรซึ่งสามารถแสดงได้ดังนี้

ตารางที่ 5.2 ผลตอบสนองความถี่ของวงจรกรองสัญญาณที่ออกแบบ

Input Frequency Hz	Output Amplitude Vpp	Normalized Amplitude	Input Frequency Hz	Output Amplitude Vpp	Normalized Amplitude
100	4.21	1.0000	3800	3.13	0.7435
200	4.21	1.0000	4000	2.98	0.7078
300	4.21	1.0000	4500	2.63	0.6247
400	4.21	1.0000	**5000**	2.26	0.5368
500	4.21	1.0000	5500	1.95	0.4632
600	4.21	1.0000	**6000**	1.69	0.4014
700	4.21	1.0000	6500	1.45	0.3444
800	4.21	1.0000	7000	1.26	0.2993
900	4.21	1.0000	**7500**	1.11	0.2637
1000	4.21	1.0000	8000	0.98	0.2328
1200	4.18	0.9929	8500	0.85	0.2019
1400	4.18	0.9929	9000	0.74	0.1758
1600	4.16	0.9881	9500	0.66	0.1568
1800	4.13	0.9810	**10000**	0.60	0.1425
2000	4.08	0.9691	12500	0.36	0.0855
2200	4.03	0.9572	15000	0.24	0.0570
2400	3.96	0.9406	17500	0.16	0.0380
2600	3.89	0.9240	**20000**	0.12	0.0285
2800	3.80	0.9026	22500	0.09	0.0214
3000	3.69	0.8765	25000	0.05	0.0119
3200	3.55	0.8432	27500	0.04	0.0095
3400	3.43	0.8147	30000	0.04	0.0095
3600	3.29	0.7815	**31250**	0.04	0.0095

จากตารางที่ 5.2 จะแสดงค่าของแรงดันของสัญญาณเอาท์พุตที่อ่านได้ ณ ความถี่ต่าง ๆ จากนั้นจะทำการ Normalize ค่าที่อ่านได้ เพื่อใช้ในการ Plot Amplitude Response ของวงจรกรองสัญญาณที่ได้

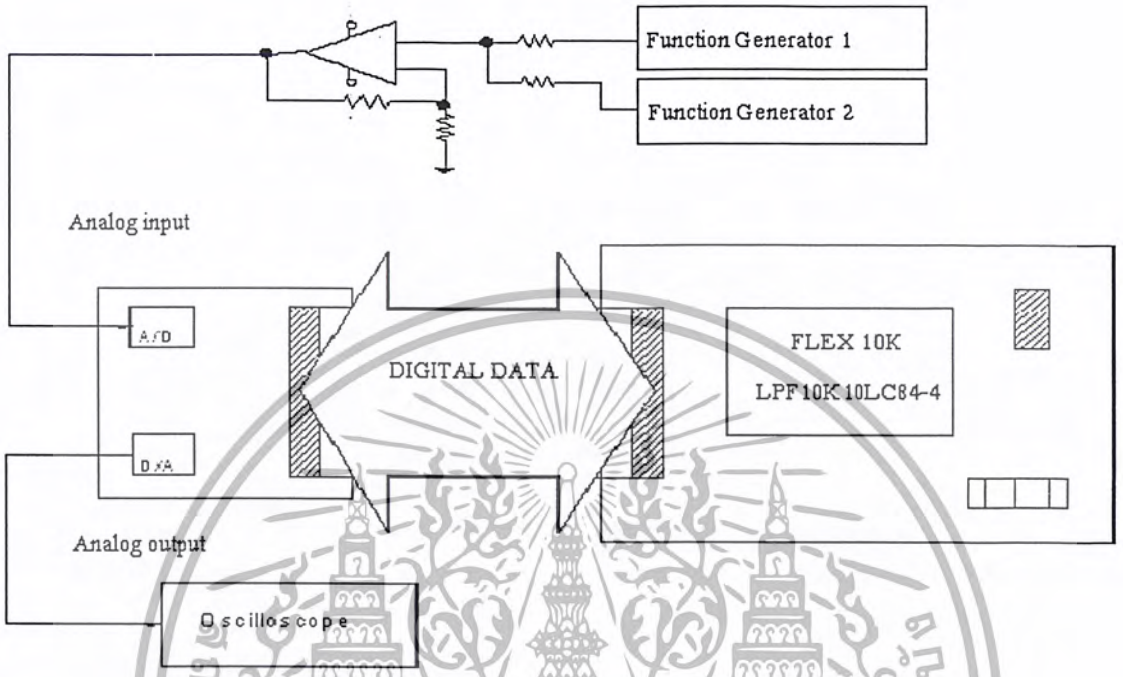


รูปที่ 5.31 กราฟผลตอบสนองทางขนาดที่ได้จากการทดลองเปรียบเทียบกับผลที่ได้จากการเลียนแบบด้วยโปรแกรม Matlab

จากรูปที่ 5.31 จะเป็นการแสดงผลตอบสนองทางขนาดที่ได้จากการทดลองเทียบกับผลที่ได้จากการเลียนแบบด้วยโปรแกรม Matlab ของวงจรกรองสัญญาณดิจิตอลชนิดป้อนกลับอันดับที่ 2 ที่ความถี่ต่ำผ่าน f_c=4 kHz, f_s=62.5 kHz คิดเป็นค่า normalized frequency = 0.128π

5.3.3 การออกแบบการทดลองสำหรับดูผลการทำงานของวงจรเปรียบเทียบกับการจำลองการทำงานด้วยโปรแกรม Matlab

สำหรับการดูผลการทำงานของวงจรกรองสัญญาณที่ออกแบบขึ้นสามารถแสดงการ Set อุปกรณ์สำหรับการทดลองได้ดังรูป



รูปที่ 5.32 การเตรียมอุปกรณ์สำหรับการทดลองดูผลการทำงานของวงจร

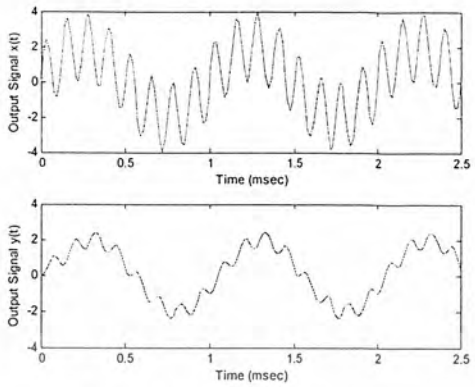
จากรูปที่ 5.32 จะเห็นได้ว่าจะมีส่วนเพิ่มเติมขึ้นมาคือวงจร Summing Amplifier เพื่อใช้ในการรวมสัญญาณ f_1 ความถี่ f_2 เข้าด้วยกันโดย ความถี่ f_1 จะเป็นความถี่ที่อยู่ในย่าน Passband และความถี่ f_2 ซึ่งจะกำหนดให้เป็นสัญญาณความถี่สูง ทำหน้าที่เสมือนสัญญาณรบกวน ผลรวมที่ได้จะถูกผ่านเข้าไปยังวงจรกรองสัญญาณเพื่อดูผลการทำงาน

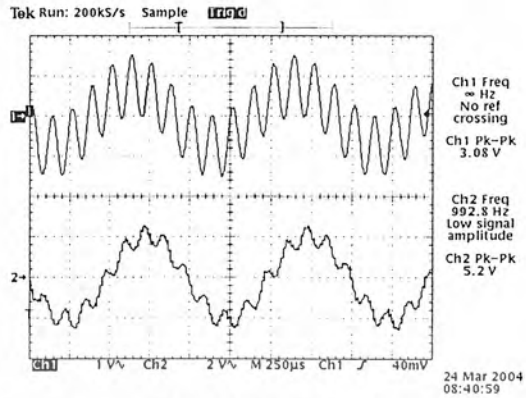
5.3.4 ผลการทำงานของวงจรเปรียบเทียบกับการจำลองการทำงานด้วยโปรแกรม Matlab

ในส่วนนี้จะเป็นผลการทำงานเทียบกับที่จำลองการทำงานที่ได้จากโปรแกรม Matlab โดยจะแบ่งการทดลองออกเป็น 3 ตอน

ตอนที่ 1 ให้สัญญาณองค์ประกอบความถี่สูงมีขนาดเท่ากับสัญญาณองค์ประกอบความถี่ต่ำ(f_1=1 kHz)

- โดยค่าจาก Function Generator ของ f_1 คือ Amplitude = 0.75 V และ Offset = 0.625 V
- โดยค่าจาก Function Generator ของ f_2 คือ Amplitude = 0.75 V และ Offset = 0.625 V



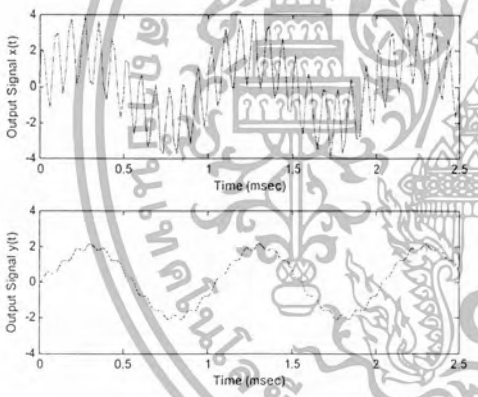


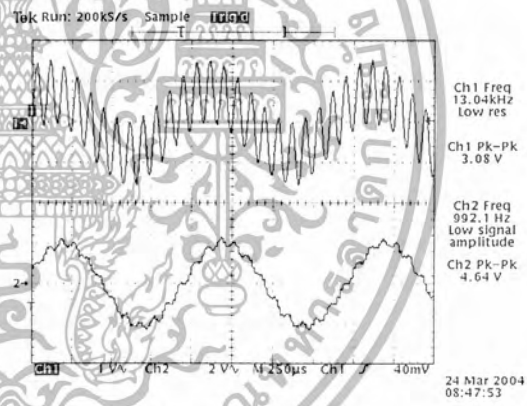
(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.33 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 8 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



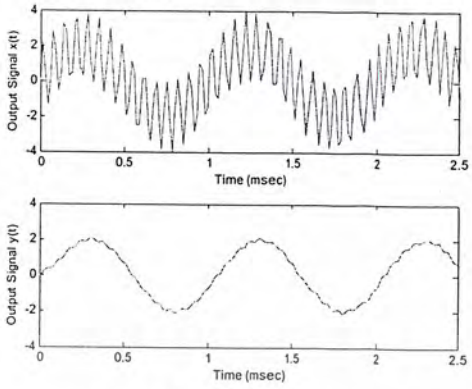


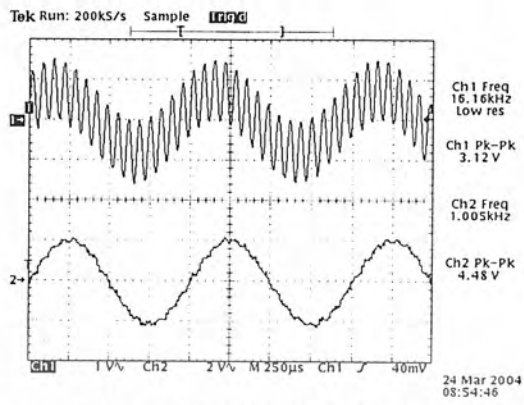
(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.34 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 12 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



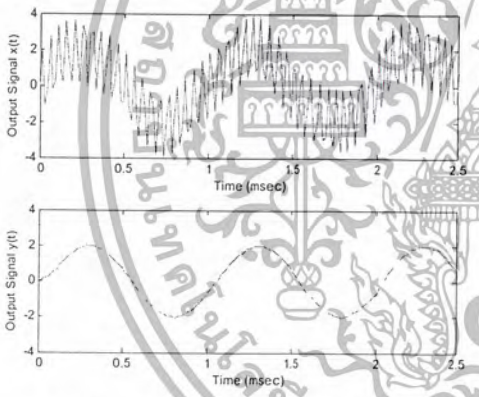


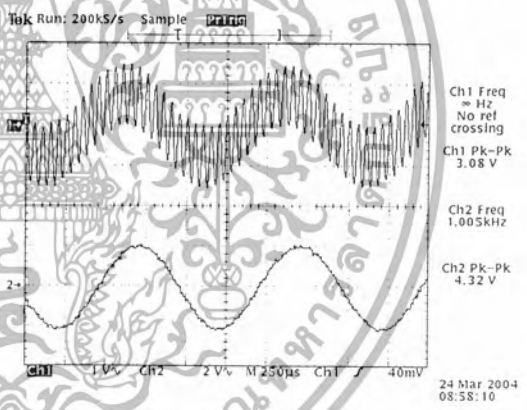
(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.35 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 15 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



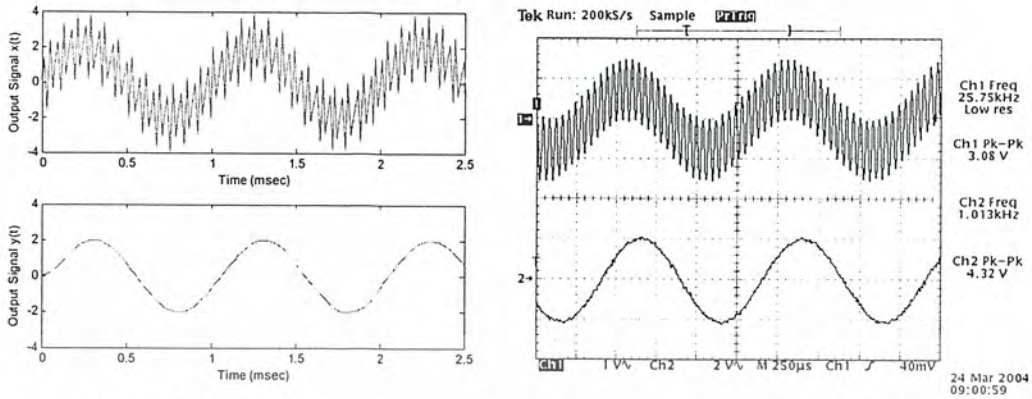


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.36 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 20 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

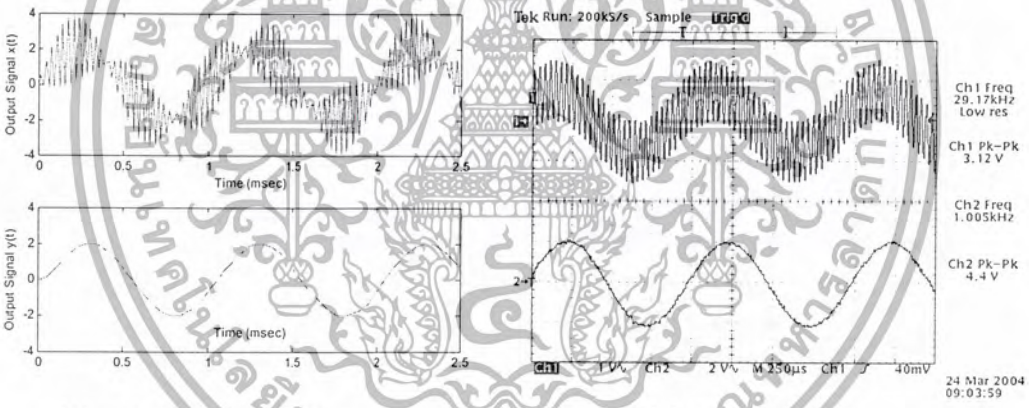


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.37 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 25 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

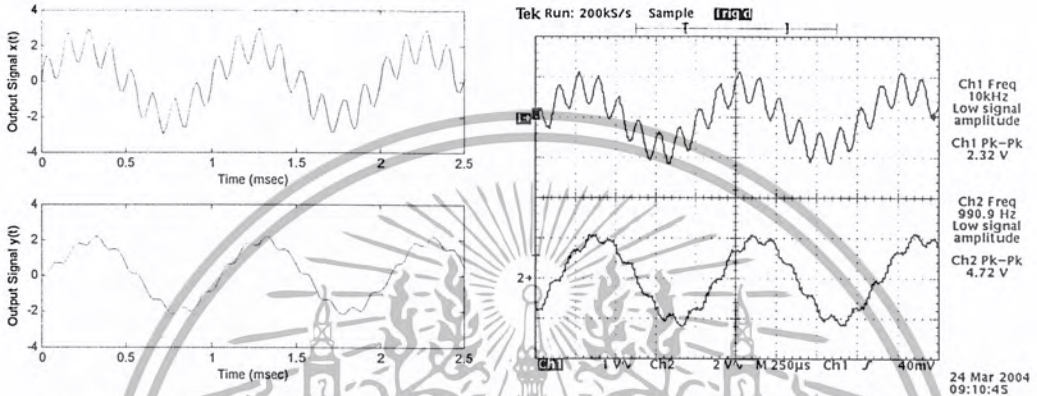
รูปที่ 5.38 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 30 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

ตอนที่ 2 ให้สัญญาณองค์ประกอบความถี่สูงมีขนาดเป็นครึ่งหนึ่งของสัญญาณองค์ประกอบความถี่ต่ำ (f_1=1 kHz)

- โดยค่าจาก Function Generator ของ f_1 คือ Amplitude = 0.75 V และ Offset = 0.625 V
- โดยค่าจาก Function Generator ของ f_2 คือ Amplitude = 0.375 V และ Offset = 0.625 V

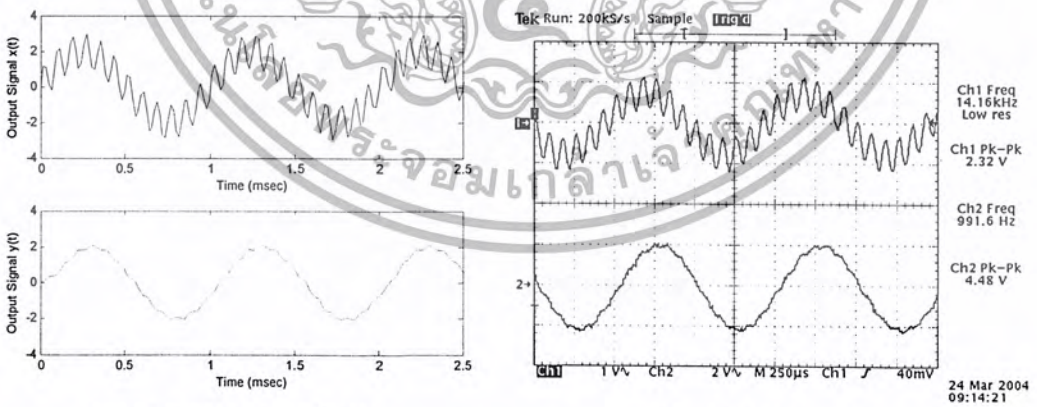


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.39 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 8 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

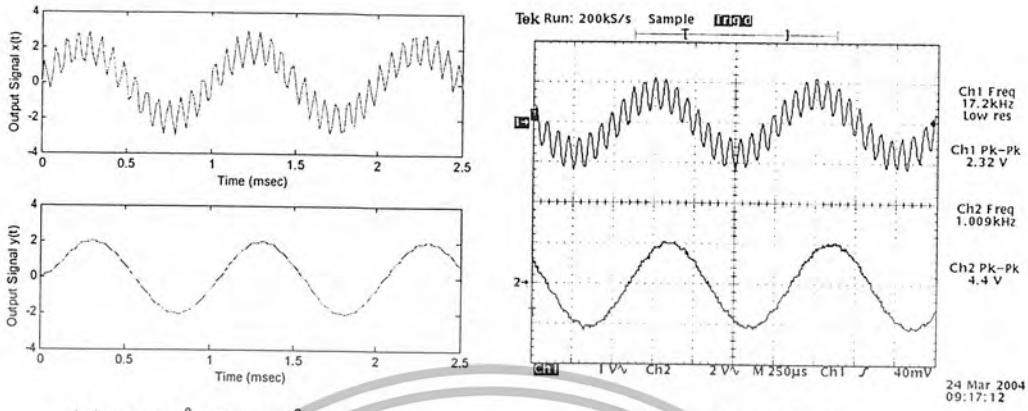


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.40 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 12 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

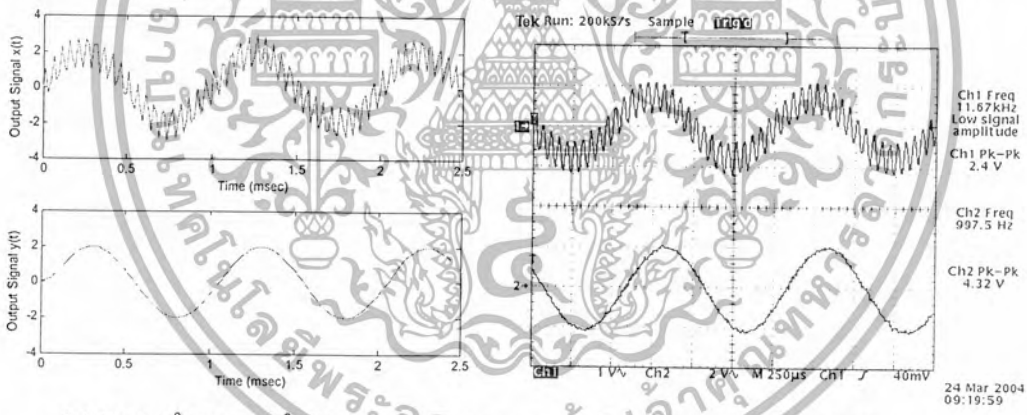


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.41 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 15 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

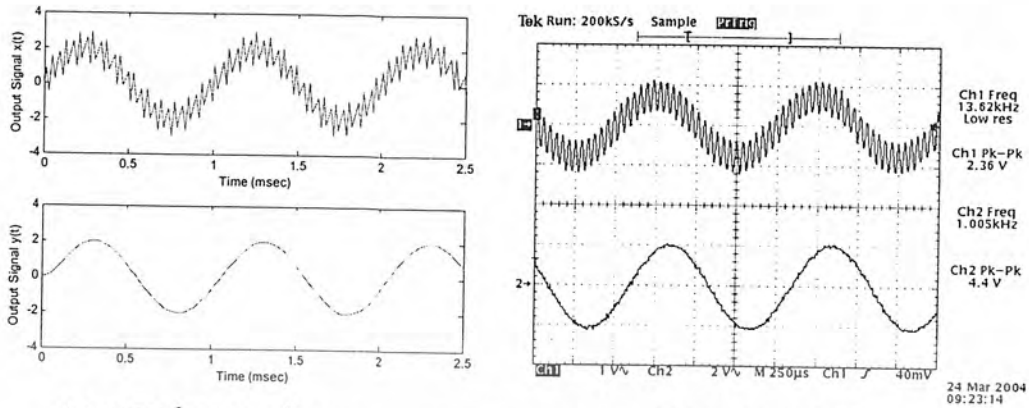


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.42 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 20 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

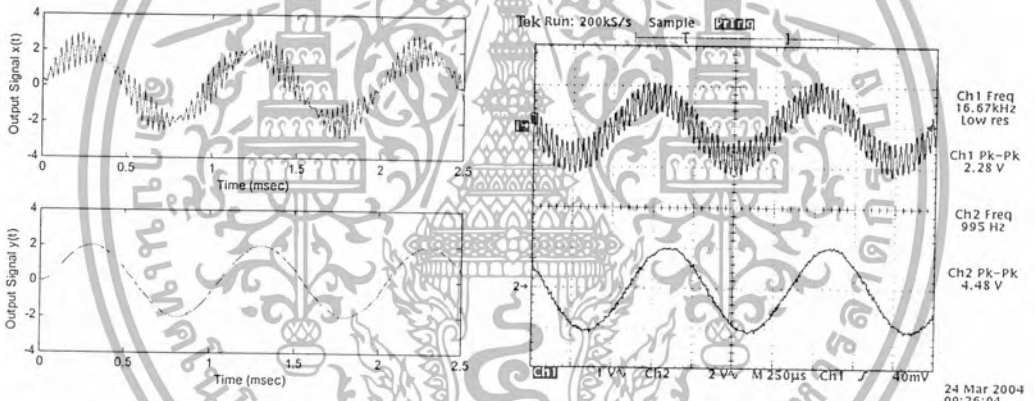


(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.43 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 25 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter



(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

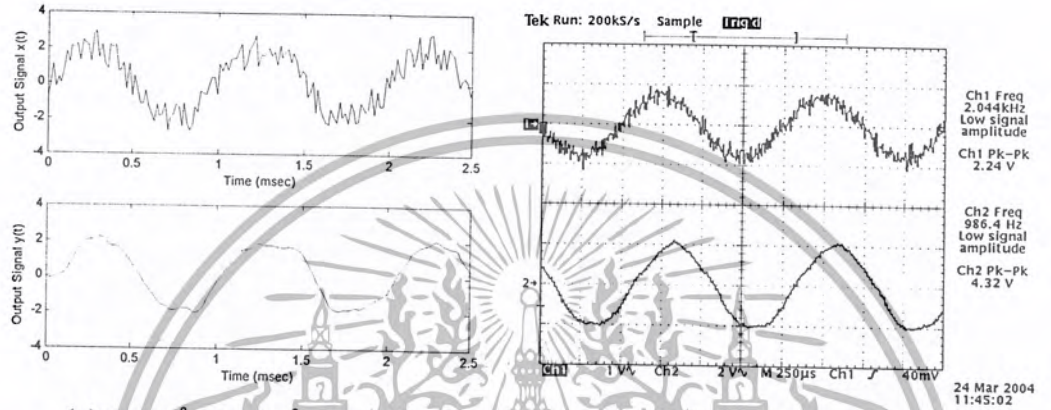
รูปที่ 5.44 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย f_1=1 kHz และ f_2= 30 kHz

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

ตอนที่ 3 สัญญาณองค์ประกอบความถี่ต่ำ (f_l=1 kHz) รวมกับสัญญาณรบกวนแบบสุ่ม (Random) โดยกำหนดค่า Function Generator ของ f_l คือ Amplitude = 0.75 V และ Offset = 0.625 V

1. กำหนดค่า Function Generator ของสัญญาณ Random คือ Amplitude = 1.5 V และ Offset = 0.625 V



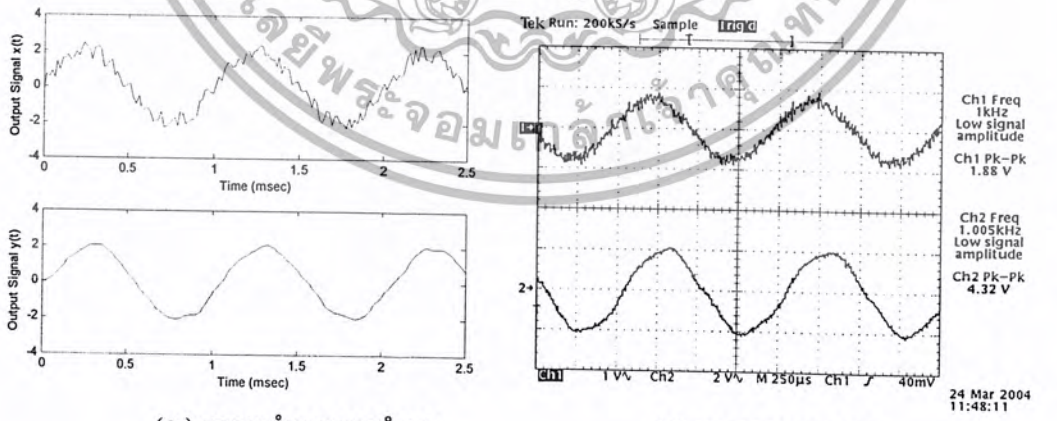
(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.45 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย Amplitude ของ Noise เป็น2 เท่าของ f_l

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

2. กำหนดค่า Function Generator ของสัญญาณ Random คือ Amplitude = 0.75 V และ Offset = 0.625 V



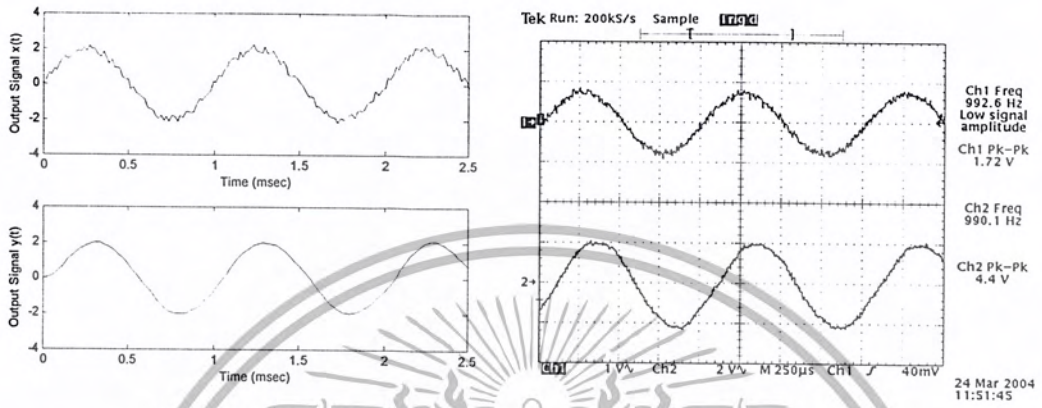
(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.46 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย Amplitude ของ Noise เป็น1 เท่าของ f_l

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

3. กำหนดค่า Function Generator ของสัญญาณ Random คือ Amplitude = 0.375 V และ Offset = 0.625 V



(ก) ผลการจำลองการทำงาน (ข) ผลจากการทดลอง

รูปที่ 5.47 ผลการจำลองการทำงานเทียบกับผลการทดลอง โดย Amplitude ของ Noise เป็น 0.5 เท่าของ f_1

CH1 สัญญาณอินพุตที่ป้อนให้กับวงจร Low Pass Filter

CH2 สัญญาณเอาท์พุตที่ออกจากวงจร Low Pass Filter

บทที่ 6
วิจารณ์และสรุป

วงจรกรองสัญญาณเชิงเลขที่ออกแบบในการทดลอง จะเป็นวงจรกรองสัญญาณเชิงเลขชนิดป้อนกลับในอันดับที่ 2 และเป็นวงจรกรองความถี่ต่ำผ่านโดยในการออกแบบได้ใช้โครงสร้างของเลขคณิตกระจาย (Distributed Arithmetic) โดยใช้เทคนิคการแทนสมการผลต่างสืบเนื่องด้วยปริภูมิสเตท (State-Space Representation) โดยใช้โครงสร้างแบบ Controllable Canonical Form ซึ่งส่งผลให้จำนวนของสัญญาณอินพุทที่ใช้ในการอ้างอิงตำแหน่งของหน่วยความจำลดลง ทำให้พื้นที่ของ Chip FPGA ที่ใช้มีขนาดลดลงและความเร็วหรือ Maximum Frequency สูงขึ้นกว่ากรณีที่สร้างจากโครงสร้างโดยตรง

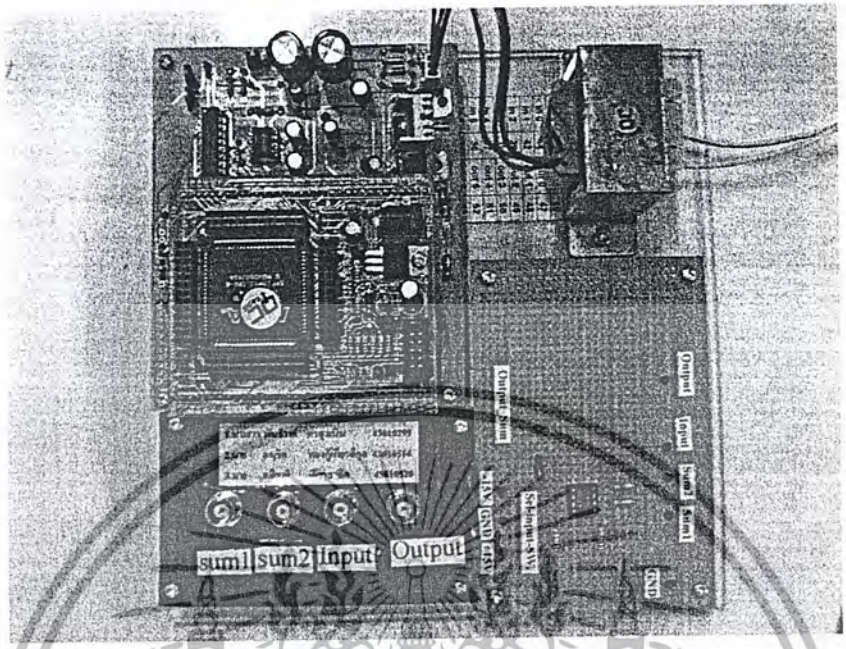
แต่ว่าในการใช้โครงสร้างแบบ Controllable Canonical Form จะมีปัญหาที่เกิดขึ้นในการออกแบบ คือการกำหนดขนาดของความยาวคำที่ใช้ในหน่วยความจำที่เหมาะสม เนื่องจากการประมวลผลเป็นแบบ 8 บิท คือสัญญาณที่ใช้เป็นแอดเดรสของหน่วยความจำมีขนาด 8 บิท ถึงแม้ว่าขนาดของความยาวคำที่ใช้ในหน่วยความจำและวงจรสเกลลิ่งแอคคิวมูเลเตอร์จะมีจำนวนบิทมากๆ ก็ตามก็ไม่ได้ช่วยให้ผลลัพธ์จากการคำนวณถูกต้องขึ้น เนื่องจากผลลัพธ์ที่ได้จากการคำนวณจะถูกตัดให้เหลือเพียง 8 บิท ซึ่งจะทำให้เกิดความผิดพลาดของสัมประสิทธิ์มากกว่าการกำหนดให้ขนาดของความยาวคำมีขนาด $8+n$ บิท และแอคคิวมูเลเตอร์รีจิสเตอร์มีขนาด $9+n$ บิท โดย 2^n คือค่าที่ใช้ในการหารสัมประสิทธิ์ในหน่วยความจำไม่ให้มีขนาดเกินหนึ่ง เพื่อไม่ให้เกิดการล้นของสัมประสิทธิ์ ผลลัพธ์จากการคำนวณจะต้องทำการคูณ 2^n กลับ หรือเลื่อนข้อมูลไปทางซ้าย n บิท ซึ่งจะทำให้ผลลัพธ์ที่โหลดออกมามีขนาด 8 บิทพอดีเทียบได้กับเป็นการปัดเศษของผลลัพธ์

เนื่องจากโครงสร้างแบบ Controllable Canonical Form เกิดปัญหาสัญญาณรบกวนเนื่องจากการปัดเศษ ดังนั้น เราจึงได้นำโครงสร้างแบบ Minimum Noise Structure มาใช้ในการแก้ปัญหานี้ โดยมีการคำนวณค่าของสัมประสิทธิ์ที่เก็บภายใน ROM ใหม่

ส่วนผลการทดลองวัดคุณลักษณะของวงจรกรองสัญญาณที่ออกแบบ จะเห็นได้ว่าผลตอบสนองทางขนาดที่ได้มีลักษณะที่เกือบจะเท่ากับผลตอบสนองทางขนาดในทางทฤษฎี ส่วนผลการทำงานของวงจรเมื่อเทียบกับผลการจำลองการทำงานด้วยโปรแกรม Matlab โดยการนำสัญญาณความถี่ต่ำรวมกับสัญญาณความถี่สูงป้อนเป็นอินพุทของวงจร จะเห็นได้ว่าทำงานได้เหมือนกับการจำลองการทำงาน

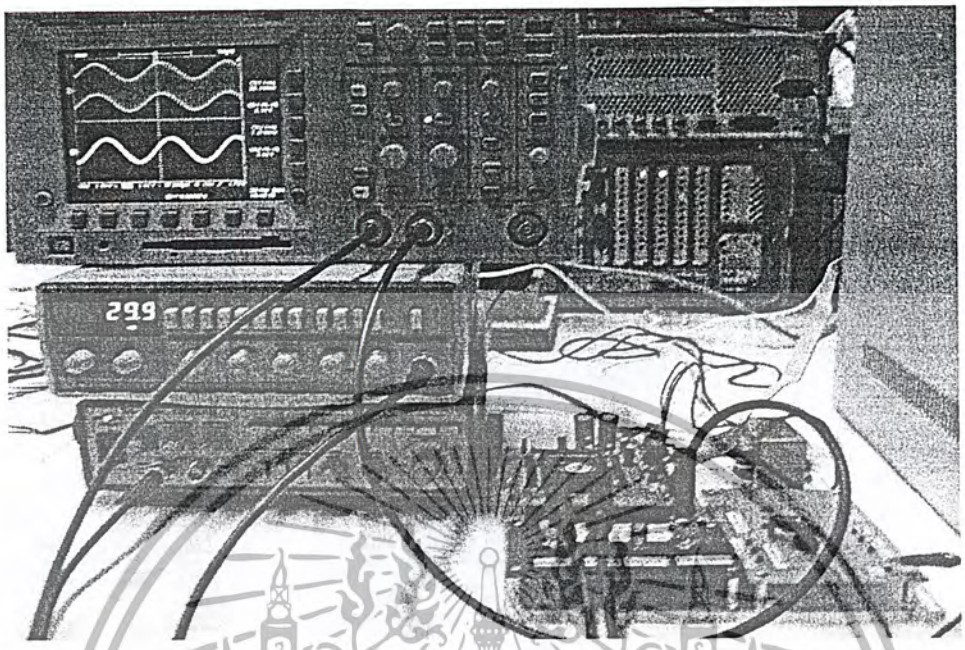
ในส่วนของฝั่งการคำนวณตัวแปรสเตท ซึ่งมองได้ว่าเป็นวงจรกรองแบบ IIR ต้องใช้ขนาดของความยาวคำมากเพื่อป้องกันเรื่องเสถียรภาพของวงจร ส่วนฝั่งการคำนวณเอาท์พุท ซึ่งมองได้ว่าเป็นวงจรกรองแบบ FIR ซึ่งไม่มีปัญหาเรื่องเสถียรภาพ แต่มีปัญหาในเรื่องความถูกต้องของเอาท์พุทเนื่องจากย่านพลวัตรของสัมประสิทธิ์จะมีค่ามาก ดังนั้นถ้าแทนด้วยจำนวนบิทจำกัดแล้ว สำหรับวงจรที่ออกแบบนี้วงจรจะคงมีเสถียรภาพ แต่ผลลัพธ์ที่ได้จากการคำนวณของฝั่งเอาท์พุทจะมีความผิดพลาดเกิดขึ้น ซึ่งทำให้วงจรกรองสัญญาณทำงานได้แต่ให้ผลตอบสนองทางความถี่ผิดพลาดไปจากทางทฤษฎี

ภาคผนวก



ภาพชิ้นงานที่ใช้ในการทดลอง

ภาพชิ้นงานที่ใช้ในการทดลอง



ภาพขณะทำการทดลอง

MATLAB CODE

โปรแกรมที่ใช้ในการเปรียบเทียบอัตราขยายของสัญญาณรบกวน

```
%This program used for plot noise gain for various type of state-
 space digital filter (type_1 to type_5)
%you can change function for design butterworth to chebyshev or
 elliptic
%include type for LPF or HPF
clear all;clc
for n=1:1:98
    i=n/100;
    [num,den]= butter(2,i);
    [Ac,Bc,Cc,Dc]=cntrl(num,den);
    [Am,Bm,Cm,Dm]=min_noise(num,den);
    [Kc,F]=compK(Ac,Bc,Cc,Dc);
    [Wc,F]=compW(Ac,Bc,Cc,Dc);
    [Km,F]=compK(Am,Bm,Cm,Dm);
    [Wm,F]=compW(Am,Bm,Cm,Dm);
    noise_gainC(n)=0.5*trace(Kc)*trace(Wc)
    noise_gainM(n)=0.5*trace(Km)*trace(Wm)
    freq(n)=i;
end
plot(freq,noise_gainC,freq,noise_gainM,'k');
axis([0 1 0.2 300]);
xlabel('Normalized frequency');
ylabel('Noise gain');
```

ฟังก์ชั่นที่ใช้ในการจำลองการทำงาน

```
function [K,F]=compK(A,B,C,D)
%This function used for compute controllability Gramiann matrix for
 controllable pair (A,B)

F=A;
K=B*B';
for i=0:1000
    K=(F*K*F')+K;
    F=F^2;
End
```

ฟังก์ชั่นที่ใช้ในการจำลองการทำงาน

```
function [W,F]=compW(A,B,C,D)
%This function used for compute observability Gramiann matrix for
 observable pair (C,A)

F=A';
W=C'*C;
for i=0:1000
W=(F*W*F')+W;
F=F^2;
end;
```

โปรแกรมที่ใช้ในการหาผลตอบสนองทางความถี่

```
f1=input(' Enter  frequency 1 in KHz :');
f2=input(' Enter  frequency 2 in KHz :');
fc=input(' Enter cutoff frequency in KHz :');
fs=input(' Enter sampling frequency in KHz :');
b=[0.1327 0.264 0.1327];
a=[0.36183 -0.79957 1];
wc=fc/(fs/2);
[b,a]=butter(2,wc);
[amp,q]=freqz(b,a,512,fs*1000);
figure(1);
plot(q,abs(amp));grid;
axis([0 fs*1000 0 1.05])
ylabel('Normalize Amplitude');
xlabel('Frequency (Hertz)');

N_point=250;
t=[0:N_point]/(fs*1000);
x=2*sin(2*pi*f1*1000*t)+1*sin(2*pi*f2*1000*t);
figure(2);
subplot(2,1,1);
plot(t*1000,x);
xlabel('Time (msec)');
ylabel('Output Signal x(t)');

y=filter(b,a,x);
subplot(2,1,2);
plot(t*1000,y);
xlabel('Time (msec)');
ylabel('Output Signal y(t)');

spec_x=fft(x,1024);
spec_y=fft(y,1024);
spectrum_x=abs(spec_x);
spectrum_y=abs(spec_y);
fre=[1:512];
frequency=fre*fs/1024;
figure(3);
subplot(2,1,1);
plot(frequency,spectrum_x(fre));
xlabel(' Frequency in KHz ');
ylabel(' Input Spectrum ');
Subplot(2,1,2);
plot(frequency,spectrum_y(fre));
xlabel(' Frequency in KHz ');
ylabel(' Output Spectrum ');

fss=4000;
n=0:fss-1;

h=2*sin(2*pi*f1*(n/fss))+1*sin(2*pi*f2*(n/fss));

figure(4);
plot(h);

%figure(5);
%stem(n,h);
```

```
h=h;
h=h/max(h);
for j=3001:4000
            coeff=h(j);
            if  coeff>0
                coeff=round(coeff*(2^(8)-1));
                coeff=dec2bin(coeff,8);
            else
                coeff=abs(coeff);
                coeff=round(coeff*(2^(8)-1));
                coeff=dec2bin(coeff,8);
            end
  %fprintf('%.4f>%s \n',j*9.9968,coeff)
end
```

โปรแกรมที่ใช้ในการหาค่าที่เก็บอยู่ใน ROM แบบโครงสร้าง Controllable Canonical form

```
clc;
clc;%clear screen
format long;
fprintf('\n');
disp('*************************************************************');
disp(' This Program for compute coefficient of Butterworth Lowpass
filter in cntrl form');
disp('*************************************************************');
fprintf('\n');

fc=input(' Enter cut-off frequency in kHz =');
fs=input(' Enter sampling frequency in kHz=');
m=input(' Enter Number of bit required =');
n=2;
t=(fc*1000)/((fs*1000)/2);
[num,den]=butter(n,t);
[A,B,C,D]=cntrl(num,den);
      fprintf('\n\n');
      fprintf('*********************************************');
      fprintf('\n                          Matrix A,B,C,D ');
      fprintf('\n*********************************************');
      fprintf('\n    %.15f      %.15f          %i',A(1,1),A(1,2),B(1))
      fprintf('\n   A =                                        B =')
      fprintf('\n    %.15f      %.15f          %i',A(2,1),A(2,2),B(2))
      fprintf('\n\n');
      fprintf('\n   c = %.15f       %.15f       D = %.15f',C(1),C(2),D)
      fprintf('\n\n');
%-------------------------------------------------------------------
a11=A(1,1);
a12=A(1,2);
a21=A(2,1);
a22=A(2,2);
b1=B(1);
b2=B(2);
c1=C(1);
c2=C(2);
d=D;

disp('                        |          |          |                ');
disp('q_2(n) q_2(n-1) x(n)|     F    |   F/2    |      phi       ');
disp('-------------------------------------------------------------');
```

```
      addr(1,1)=0;addr(1,2)=0;addr(1,3)=0;    %generate address
      addr(2,1)=0;addr(2,2)=0;addr(2,3)=1;
      addr(3,1)=0;addr(3,2)=1;addr(3,3)=0;
      addr(4,1)=0;addr(4,2)=1;addr(4,3)=1;
      addr(5,1)=1;addr(5,2)=0;addr(5,3)=0;
      addr(6,1)=1;addr(6,2)=0;addr(6,3)=1;
      addr(7,1)=1;addr(7,2)=1;addr(7,3)=0;
      addr(8,1)=1;addr(8,2)=1;addr(8,3)=1;

    for j=1:8
      phi(j)=(addr(j,1)*a22)+(addr(j,2)*a21)+(addr(j,3)*b2);
      coeff=phi(j)/2;
      if    (coeff>=1|coeff<=-1)          %check condition
disp('!error! this coefficient not suitable for my filter structure
       absolute of coefficient must less than 1')

           %-------------------------------------------------------%
      elseif      coeff>=0;                %for positive value
            coeff=round(coeff*2^(m-1));
            coeff=dec2bin(coeff,m);
           %-------------------------------------------------------%
      else                                 %for negative value
            coeff=round(coeff*2^(m-1));
            coeff=coeff+(2^m);
            coeff=dec2bin(coeff,m);
      end

      fprintf('      %i       %i       %i     |  %.15f  | %.15f    |
%s \n',addr(j,1),addr(j,2),addr(j,3),phi(j),phi(j)/2,coeff);
  end %loop for of j=1:8

      fprintf('\n\n');
         disp('                  |                |                          ');
         disp('q_2(n) q_2(n-1) x(n)|      F       |     zeta                 ');
         disp('___________________|________________|__________________________');

  for j=1:8
      zeta(j)=(addr(j,1)*c2)+(addr(j,2)*c1)+(addr(j,3)*d);
      coeff1=zeta(j);
      if    (coeff1>=1|coeff1<=-1)         %check condition
disp('!error! this filter not suitable for my filter structure
      absolute of coefficient must less than 1')

           %-------------------------------------------------------%
      elseif      coeff1>=0;               %for positive value
            coeff1=round(coeff1*2^(m-1));
            coeff1=dec2bin(coeff1,m);
           %-------------------------------------------------------%
      else                                 %for negative value
            coeff1=round(coeff1*2^(m-1));
            coeff1=coeff1+(2^m);
            coeff1=dec2bin(coeff1,m);
      end
      fprintf('        %i       %i       %i     |  %.15f  |
%s \n',addr(j,1),addr(j,2),addr(j,3),zeta(j),coeff1);

   end %loop for of j=1:8
     fprintf('\n\n\n\n\n\n\n');
```

โปรแกรมที่ใช้ในการหาค่าที่เก็บอยู่ใน ROM แบบโครงสร้าง Minimum Noise Structure

```
clc;
clc;%clear screen
format long;
fprintf('\n');
disp('   ***************************************************');
disp('   This Program for compute coefficient of Butterworth Lowpass
         filter in min_noise form');
disp('   ***************************************************');
fprintf('\n');

fc=input(' Enter cut-off frequency in kHz =');
fs=input(' Enter sampling frequency in kHz=');
m=input (' Enter Number of bit required   =');
n=2;
t=(fc*1000)/((fs*1000)/2);
[num,den]=butter(n,t);
[A,B,C,D]=min_noise(num,den);
      fprintf('\n\n');
      fprintf('*****************************************');
      fprintf('\n                          Matrix A,B,C,D ');
      fprintf('\n*****************************************');
      fprintf('\n    %.15f   %.15f     %i',A(1,1),A(1,2),B(1))
      fprintf('\n  A =                              B =')
      fprintf('\n    %.15f   %.15f     %i',A(2,1),A(2,2),B(2))
      fprintf('\n\n');
      fprintf('\n  C = %.15f   %.15f     D = %.15f',C(1),C(2),D)
      fprintf('\n\n');
%-------------------------------------------------------------------
a11=A(1,1);
a12=A(1,2);
a21=A(2,1);
a22=A(2,2);
b1=B(1);
b2=B(2);
c1=C(1);
c2=C(2);
d=D;

      addr(1,1)=0;addr(1,2)=0;addr(1,3)=0;   %generate address
      addr(2,1)=0;addr(2,2)=0;addr(2,3)=1;
      addr(3,1)=0;addr(3,2)=1;addr(3,3)=0;
      addr(4,1)=0;addr(4,2)=1;addr(4,3)=1;
      addr(5,1)=1;addr(5,2)=0;addr(5,3)=0;
      addr(6,1)=1;addr(6,2)=0;addr(6,3)=1;
      addr(7,1)=1;addr(7,2)=1;addr(7,3)=0;
      addr(8,1)=1;addr(8,2)=1;addr(8,3)=1;

disp('                      |          |          |');
disp(' q_1(n) q_2(n) x(n) |     F    |    F/2   |     phi_1     ');
disp('_____________________|__________|__________|_______________');
```

```
    for j=1:8
      phi(j)=(addr(j,1)*a11)+(addr(j,2)*a12)+(addr(j,3)*b1);
      coeff=phi(j)/2;
      if    (coeff>=1|coeff<=-1)          %check condition
disp('!error! this coefficient not suitable for my filter structure
      absolute of coefficient must less than 1')

%-------------------------------------------------------------------
      elseif      coeff>=0;              %for positive value
            coeff=round(coeff*2^(m-1));
            coeff=dec2bin(coeff,m);
%-------------------------------------------------------------------
      else                               %for negative value
            coeff=round(coeff*2^(m-1));
            coeff=coeff+(2^m);
            coeff=dec2bin(coeff,m);
      end

      fprintf('        %i       %i       %i    |  %.15f  | %.15f   |
%s \n',addr(j,1),addr(j,2),addr(j,3),phi(j),phi(j)/2,coeff);
    end %loop for of j=1:8


disp('                  |        |          |');
disp('q_1(n) q_2(n) x(n)|    F   |     F/2  | phi_2          ');
      disp('_____________________________________________________');

    for j=1:8
      phi(j)=(addr(j,1)*a21)+(addr(j,2)*a22)+(addr(j,3)*b2);
      coeff=phi(j)/2;
      if    (coeff>=1|coeff<=-1)          %check condititon
disp('!error! this coefficient not suitable for my filter structure
      absolute of coefficient must less than 1')

%-------------------------------------------------------------------%
      elseif      coeff>=0;              %for positive value
            coeff=round(coeff*2^(m-1));
            coeff=dec2bin(coeff,m);
%-------------------------------------------------------------------%
      else                               %for negative value
            coeff=round(coeff*2^(m-1));
            coeff=coeff+(2^m);
            coeff=dec2bin(coeff,m);
      end

   fprintf('        %i       %i       %i    |  %.15f  | %.15f   |
%s \n',addr(j,1),addr(j,2),addr(j,3),phi(j),phi(j)/2,coeff);
    end %loop for of j=1:8
   fprintf('\n\n');
disp('                  |                  |                ');
disp(' q_1(n) q_2(n) x(n) |        F         |    zeta        ');
disp('____________________________________________________');

  for j=1:8
      zeta(j)=(addr(j,1)*c1)+(addr(j,2)*c2)+(addr(j,3)*d);
      coeff1=zeta(j);
      if    (coeff1>=1|coeff1<=-1)        %check condititon
disp('!error! this filter not suitable for my filter structure
      absolute of coefficient must less than 1')
```

```
%-------------------------------------------------------------%
      elseif      coeff1>=0;                %for positive value
            coeff1=round(coeff1*2^(m-1));
            coeff1=dec2bin(coeff1,m);
%-------------------------------------------------------------%
      else                                  %for negative value
            coeff1=round(coeff1*2^(m-1));
            coeff1=coeff1+(2^m);
            coeff1=dec2bin(coeff1,m);
      end

      fprintf('        %i       %i       %i     |  %.15f  |
%s \n',addr(j,1),addr(j,2),addr(j,3),zeta(j),coeff1);

   end %loop for of j=1:8
      fprintf('\n\n\n\n\n\n');
```

ฟังก์ชั่นที่ใช้ในการหาค่าที่เก็บไว้ใน ROM แบบ Minimum Noise Structure

```
function [A,B,C,D]=min_noise(num,den)
%-------------------------------%
%define parameter used for minimumnoise 9 multipliers structure from
transfer function
d=num(1);
alpha1=num(2)-(num(1)*den(2));
alpha2=num(3)-(num(1)*den(3));
beta1=den(2);
beta2=den(3);
%-------------------------------%
%calculate some parameter for used in minimumnoise structure
mu=(((alpha2/alpha1)^2)-((alpha2/alpha1)*beta1)+beta2)^(1/2);
grama=(alpha2/alpha1)-mu;
zi=(alpha2/alpha1)+mu;
lamda=(beta2-1)*(((beta2+1)^2)-(beta1^2));
epsilon=((beta1/2)^2)-beta2;
%-------------------------------%
%compute matrix A, B, C, D
b1=(lamda/((2*beta1*grama)-((beta2+1)*(1+grama^2))))^(1/2);
b2=(lamda/((2*beta1*zi)-((beta2+1)*(1+zi^2))))^(1/2);
a11=-(beta1/2);
a22=-(beta1/2);
a21=(((b2^2+beta2-1)/(b1^2+beta2-1))*epsilon)^(1/2);
a12=epsilon/a21;
c1=alpha1/(2*b1);
c2=alpha1/(2*b2);
A=[a11 a12;a21 a22];
B=[b1;b2];
C=[c1 c2];
D=d;
```

โปรแกรมที่ใช้ในการหาค่าอัตราขยายสัญญาณรบกวน

```
%This program used for plot noise gain for various type of state-
 space digital filter(type_1 to type_5)
%you can change function for design butterworth to chebyshev or
 elliptic
%include type for LPF or HPF
clc
clear all;
for n=1:1:24
    [num,den]=butter(2,0.1);
    [Ac,Bc,Cc,Dc]=cntrl(num,den);
    [Am,Bm,Cm,Dm]=min_noise(num,den);
    [Kc,F]=compK(Ac,Bc,Cc,Dc);
    [Wc,F]=compW(Ac,Bc,Cc,Dc);
    [Km,F]=compK(Am,Bm,Cm,Dm);
    [Wm,F]=compW(Am,Bm,Cm,Dm);
    noise_gainC(n)=0.5*trace(Kc)*trace(Wc)
    noise_powerC(n)=10*(log10(((2^(-2*n))/3)*noise_gainC(n)))
    noise_gainM(n)=0.5*trace(Km)*trace(Wm)
    noise_powerM(n)=10*(log10(((2^(-2*n))/3)*noise_gainM(n)))
    No_bit(n)=n;
end
plot(No_bit,noise_powerC,No_bit,noise_powerM,'k');
axis([6 24 -150 0]);
xlabel('Coefficient wordlength');
ylabel('Noise gain');
```

VHDL CODE

PISO

```
library ieee;
use ieee.std_logic_1164.all;
entity piso_8 is
port(  X          :in std_logic_vector(7 downto 0);
       clk,lr     :in std_logic ;
       Xi         :out std_logic);
end;

architecture a of piso_8 is
      signal sbus : std_logic_vector(7 downto 0);
begin
      process(clk,lr)
      begin
            if lr = '0' then
                  sbus <= X;
            elsif (clk'event and clk='1') then
                  Xi          <= sbus(0);
                  sbus(0)     <= sbus(1);
                  sbus(1)     <= sbus(2);
                  sbus(2)     <= sbus(3);
                  sbus(3)     <= sbus(4);
                  sbus(4)     <= sbus(5);
                  sbus(5)     <= sbus(6);
                  sbus(6)     <= sbus(7);
            end if;
      end process;
end a;
```

SISO

```
library ieee;
use ieee.std_logic_1164.all;

entity siso_8 is
port(qi,clk  : in std_logic;
     qi_1    : out std_logic);
end;

architecture a of siso_8 is
      signal sbus : std_logic_vector(7 downto 1);
begin
      process(clk)
      begin
            if (clk'event and clk='1') then
                  qi_1        <= sbus(1);
                  sbus(1)     <= sbus(2);
                  sbus(2)     <= sbus(3);
                  sbus(3)     <= sbus(4);
                  sbus(4)     <= sbus(5);
                  sbus(5)     <= sbus(6);
                  sbus(6)     <= sbus(7);
                  sbus(7)     <= qi;
            end if;
      end process;
end a;
```

CONTROL_UNIT

```
library ieee;
use ieee.std_logic_1164.all;

entity  control_unit_all  is
port(  sys_clk :in std_logic;
       clk,sa,lacc,lr,clacc,sc :out bit);
end;

architecture  a of control_unit_all is
      type state_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8,s9);
      signal  state : state_type;
      signal count : integer range 0 to 32 ;
begin
      process(sys_clk)
      begin
            if sys_clk'event and sys_clk='1'then
                  case  count  is
                        when 1 | 5 | 9 | 13 | 17 | 21 | 25 => state
                                    <=  s1;
                        clk   <= '1';
                        sa    <= '0';
                        lacc  <= '0';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '1';
                        count <= count + 1 ;
                       when 2 | 6 | 14 | 18 | 22 | 10 | 26 => state
                                    <= s2 ;
                        clk   <= '1';
                        sa    <= '0';
                        lacc  <= '1';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '1';
                        count <=  count + 1 ;
                       when 3 | 4 | 7 | 8 | 15 | 16 | 19 | 20 | 23 |
                       24 => state <= s3 ;
                        clk   <= '0';
                        sa    <= '0';
                        lacc  <= '0';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '1';
                        count <=  count + 1 ;
                      when 11|12 => state <= s4 ;
                        clk   <= '0';
                        sa    <= '0';
                        lacc  <= '0';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '0';
                        count <=  count + 1 ;
                      when 27|28 => state <= s5 ;
                        clk   <= '0';
                        sa    <= '1';
                        lacc  <= '0';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '1';
```

```
                        count <=  count + 1 ;
                  when 29 => state <= s6 ;
                        clk   <= '1';
                        sa    <= '1';
                        lacc  <= '0';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '1';
                        count <=  count + 1 ;
                  when 30 => state <= s7 ;
                        clk   <= '1';
                        sa    <= '1';
                        lacc  <= '1';
                        lr    <= '1';
                        clacc <= '1';
                        sc    <= '1';
                        count <=  count + 1 ;
                  when 31 => state <= s8 ;
                        clk   <= '0';
                        sa    <= '0';
                        lacc  <= '0';
                        lr    <= '0';
                        clacc <= '1';
                        sc    <= '1';
                        count <=  count + 1 ;
                  when others => state <= s9;
                        clk   <= '0';
                        sa    <= '0';
                        lacc  <= '0';
                        lr    <= '1';
                        clacc <= '0';
                        sc    <= '1';
                        count <=  1 ;
                end case;
          end if;
     end process;
end;
```

BUFFER

```
library ieee;
use   ieee.std_logic_1164.all;
use   ieee.std_logic_unsigned.all;

entity      buff_inv is
      port (load : in std_logic;
            d_in : in std_logic_vector(7 downto 0);
            buff_reg :  buffer std_logic_vector(7 downto 0));
end;

architecture rtl of buff_inv is
begin
      process(load)
      begin
              if load='0' then
                    buff_reg<= d_in(7 downto 0);
              end if;
      end process;
end;
```

EPROM 1

```
library ieee;
use ieee.std_logic_1164.all;

entity eprom1 is
port(Xi,qi_1,qi:in std_logic ;
       phi  :out std_logic_vector(7 downto 0));
end;

architecture a of eprom1 is
      signal address : std_logic_vector(2 downto 0);
begin
      process(Xi,qi_1,qi)
      begin
            address <= qi&qi_1&Xi;
            case address is
                  when "000"  => phi <="00000000";
                  when "001"  => phi <="00100010";
                  when "010"  => phi <="11100110";
                  when "011"  => phi <="00001000";
                  when "100"  => phi <="01001001";
                  when "101"  => phi <="01101011";
                  when "110"  => phi <="00101111";
                  when others => phi <="01010001";
            end case;
      end process;
end a;
```

EPROM 2

```
library ieee;
use ieee.std_logic_1164.all;

entity eprom2 is
port(Xi,qi_1,qi:in std_logic ;
       zeta :out std_logic_vector(7 downto 0));
end;

architecture a of eprom2 is
      signal address : std_logic_vector(2 downto 0);
begin
      process(Xi,qi_1,qi)
      begin
            address <= qi&qi_1&Xi;
            case address is
                  when "000"  => zeta <="00000000";
                  when "001"  => zeta <="00001001";
                  when "010"  => zeta <="00001001";
                  when "011"  => zeta <="00010010";
                  when "100"  => zeta <="00110011";
                  when "101"  => zeta <="00111011";
                  when "110"  => zeta <="00111100";
                  when others => zeta <="01000101";
            end case;
      end process;
end a;
```

ADD_SUB

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity addsub is
port( add1_in,add2_in      : in  std_logic_vector(7 downto 0);
      s_a                  : in  std_logic;
      add_out              : out std_logic_vector(7 downto 0));
end;
architecture rtl of addsub is
begin
      process (s_a,add1_in)
         variable s_sub : std_logic_vector(7 downto 0);
         variable sum   : std_logic_vector(7 downto 0);
         variable s     : std_logic_vector(8 downto 0);
    begin
      if  s_a = '0' then
            s:=("0"&add1_in)+add2_in;
            sum(7):=s(8) xor add1_in(7) xor add2_in(7) ;
            sum(6 downto 0):=s(7 downto 1);
            else
            s_sub:=(add1_in xor "11111111");
            s_sub:=s_sub+1;
            sum(7 downto 0):=add2_in(7 downto 0)+s_sub(7 downto 0);
        end if;
        add_out<=sum;
      end process ;
end;
```

ACCUMULATER

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_Unsigned.all;
entity accumulate is
port( acc_in      : in  std_logic_vector(7 downto 0);
      clacc,lacc  : in  std_logic;
      acc_out     : out std_logic_vector(7 downto 0));
end;
architecture accu of accumulate is
      signal acc_signal : std_logic_vector(7 downto 0);
begin
      process (lacc,clacc)
    begin
        if  clacc='1' then
            acc_signal<=(others=>'0');
        elsif lacc'event and lacc='1' then
            acc_signal<=acc_in;
        end if;
    end process ;
      acc_out<=acc_signal;
end;
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดีเนื่องจากได้รับความอนุเคราะห์จาก รศ. ดร. กอบชัย เดชหาญ และอาจารย์ ศรวัฒน์ ชิวปรีชา ที่ได้ให้ข้อคิด ให้การสนับสนุนและคำปรึกษา ตลอดจนการเอื้ออำนวยความสะดวกต่าง ๆ ในการใช้เครื่องมือเพื่อทำการทดลอง จึงขอขอบพระคุณมา ณ โอกาสนี้ด้วย และขอขอบคุณเพื่อนร่วมรุ่นทุกคนที่คอยให้กำลังใจและให้คำปรึกษาในทุก ๆ ด้านด้วยดีเรื่อยมา

คณะผู้จัดทำ

6 เมษายน 2547

เอกสารอ้างอิง

[1] A. Antoniou, Digital filters Analysis and Design, McGraw-Hill, 1979.

[2] A. Peled and B. Liu, "A new hardware realization of Digital filters," IEEE Trans. ASSP., Vol.ASSP-22, No. 6, pp. 456-462, December 1974.

[3] A. Peled and B. Liu, Digital Signal Processing Theory, Design and Implementation, John Wiley & Sons, 1976.

[4] S.A. White, "Applications of Distributed Arithmetic to Digital Signal Processing : A Tutorial Review," IEEE ASSP. Magazine, Vol.6, No.3, pp. 4-13, July 1989.

[5] C.S. Burrus, "Digital Filter Structure Described by Distributed Arithmetic," IEEE Trans. Circuits and Systems, Vol. CAS-24, No.12 pp. 674-680 December 1977.

[6] S. Zohar, "A VLSI Implementation of a Correlator/Digital Filter Based on Distributed Arithmetic," IEEE Trans. ASSP., Vol.37, No.1 pp. 156-160 January 1989.

[7] D.F. Elliott, Handbook of Digital Signal Processing Engineering Applications, Academic Press, 1987.

[8] วินัย ทองต้น, สมยศ จุณณะปิยะ และ กอบชัย เตชาญ, "การออกแบบและสร้างวงจรกรองสัญญาณเชิงเลขแบบบัตเตอร์เวิร์ธ อันดับที่ 6," วิศวสารลาดกระบัง ปีที่ 13 ฉบับที่ 1 หน้า 78-90 กรกฎาคม 2539

[9] S. Tantaratana, "Who Needs Hardware Multipliers," การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 21 หน้า 27-32 พฤศจิกายน 2541

[10] วัลลภ สุระกำพลธร, การประมวลผลสัญญาณเชิงเลข การกรองและการแปลง, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2533

[11] E.C. Ifeachar and B.W. Jervis, Digital Signal Processing A Practical Approach, Addison-Wesley, 1993.

[12] T.W. Parks and C.S. Burrus, Digital Filter Design, John Wiley & Sons, 1987.

[13] B.W. Bomar, "New Second-Order State-Space Structures for Realizing Low Roundoff Noise Digital Filters," IEEE Trans. ASSP., Vol. ASSP-33, No.1 pp. 106-110, February 1985.

[14] W.L. Mills, C.T. Mullis and R.A. Roberts, "Low roundoff noise and normal realizations of fixed point IIR Digital Filters," IEEE Trans. ASSP., Vol. ASSP-29, No. 4., pp. 893-903 August 1981.

[15] M. Kawamata and T. Higuchi, "A Unified Approach to the Optimal Synthesis of fixed-point State-Space Digital Filters," IEEE Trans. ASSP., Vol. ASSP-33, No.4 pp. 911-920, August 1985.

[16] B. Psenicka, F. Garcia-Ugalde, J.Savage, S.Herrera-Garcia and V.Davidek, "Design of State

Digital Filters," IEEE Trans. Signal Processing, Vol.46, No.9, pp. 2544-2549 September 1998.

[17] L. Wanhammar, DSP Integrated Circuits, Academic Press, 1999.

[18] K.K. Parhi, VLSI Digital Signal Processing Systems Design and Implementation, John Wiley & Sons, 1999.

[19] F. J. Taylor, "An Analysis of the Distributed Arithmetic Digital Filter," IEEE Trans. ASSP., Vol. ASSP-34, No.5, pp. 1165-1170, October 1986.

[20] The Role of Distributed Arithmetic in FPGA-based Signal Processing, http://www.xilinx.com

[21] D.E. Ott and T.J. Wilderotter, A Designer's Guide to VHDL Synthesis, Kluwer Academic Publishers, 1994.

[22] S. Sjoholm and L. Lindh, VHDL for Designers, Prentice Hall, 1997.