# การออกแบบและสร้างฟิลเตอร์แบงก์เพื่อใช้ในการประมวลผลสัญญาณภาพ

# FILTER BANKS DESIGN AND IMPLEMENTATION FOR IMAGE SIGNAL PROCESSING


โดย

นายพานเพชร พินเรา

นายพีรเดช มีเจริญ

นายพีรภัทร์ กิจเจริญวงศ์


ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

# การออกแบบและสร้างฟิลเตอร์แบงก์เพื่อใช้ในการประมวลผลสัญญาณภาพ

# FILTER BANKS DESIGN AND IMPLEMENTATION FOR IMAGE SIGNAL PROCESSING


โดย

นายพานเพชร พินเรา

นายพีรเดช มีเจริญ

นายพีรภัทร์ กิจเจริญวงศ์


ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

ปริญญานิพนธ์ปีการศึกษา 2556

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบและสร้างฟิลเตอร์แบงก์เพื่อใช้ในการประมวลผลสัญญาณภาพ
FILTER BANKS DESIGN AND IMPLEMENTATION FOR IMAGE SIGNAL PROCESSING


ผู้จัดทำ

1. นายพานเพชร พินเรา 53011113
2. นายพีรเดช มีเจริญ 53011155
3. นายพีรภัทร์ กิจเจริญวงศ์ 53011159


อัครพล ตรีรัตน์ ..... อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ อัครพล ตรีรัตน์)

..... อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ดร.ศรวัฒน์ ชิวปรีชา)

# กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ประสบความสำเร็จได้ด้วยความกรุณาจากท่านอาจารย์ที่ปรึกษา ผู้ช่วยศาสตราจารย์ อัครพล ตรีรัตน์ และอาจารย์ที่ปรึกษาร่วม ผู้ช่วยศาสตราจารย์ ดร.ศรวัฒน์ ชิวปรีชา ซึ่งทางผู้จัดทำปริญญานิพนธ์ขอกราบขอบพระคุณอย่างยิ่งที่กรุณาให้คำปรึกษาชี้แนะแนวทางให้ความช่วยเหลือและปรับปรุงแก้ไขจนได้ปริญญานิพนธ์ฉบับนี้ด้วยความเอาใจใส่อย่างดียิ่งตลอดมา

ผู้จัดทำปริญญานิพนธ์ขอกราบขอบพระคุณบิดาและมารดาของผู้จัดทำที่ให้โอกาส ให้การสนับสนุน และกำลังใจในการเข้ารับการศึกษาในระดับต่าง ๆ ตลอดมา นอกจากนี้ยังมีคณาจารย์ภาควิชาวิศวกรรมโทรคมนาคมทุกท่านที่ให้ความรู้ในสาขาที่เรียน ในวิชาต่าง ๆ ซึ่งเป็นประโยชน์โดยสามารถนำความรู้มาประกอบการทำโครงงานนี้จนสำเร็จ

ประโยชน์และคุณค่าที่พึงมีจากโครงงานนี้ผู้จัดทำปริญญานิพนธ์ ขอมอบให้แก่ผู้มีส่วนร่วมทุกท่าน ตลอดทั้งผู้ที่ต้องการศึกษาในเรื่องที่ทางผู้จัดทำได้ศึกษาต่อไป

นายพานเพชร พินเรา
นายพีรเดช มีเจริญ
นายพีรภัทร์ กิจเจริญวงศ์
ผู้จัดทำ

การออกแบบและสร้างฟิลเตอร์แบงก์เพื่อใช้ในการประมวลผลสัญญาณภาพ

FILTER BANKS DESIGN AND IMPLEMENTATION FOR IMAGE SIGNAL PROCESSING

โดย นายพานเพชร พินเรา 53011113
นายพีรเดช มีเจริญ 53011155
นายพีรภัทร์ กิจเจริญวงศ์ 53011159

**อาจารย์ที่ปรึกษา** ผู้ช่วยศาสตราจารย์ อัครพล ตรีรัตน์
ผู้ช่วยศาสตราจารย์ ดร.ศรวัฒน์ ชิวปรีชา

**บทคัดย่อ**

ปริญญานิพนธ์นี้เป็นการศึกษาและออกแบบฟิลเตอร์แบงก์ (Filter Banks) เพื่อใช้ในการประมวลผลภาพ โดยที่สัญญาณภาพที่เป็นอินพุตจะถูกจัดเก็บไว้ในหน่วยความจำบนบอร์ด (ROM On Board) เมื่อประมวลผลเสร็จจะแสดงผลบนจอแสดงผลวีจีเอ (VGA Monitor) ในโครงงานนี้จะทำการสร้างเป็นฮาร์ดแวร์ผ่านทาง FPGA (Field-Programmable Gate Array) ทั้งส่วนของการแสดงผลและฟิลเตอร์แบงก์ โดยในส่วนของการออกแบบฟิลเตอร์แบงก์จะอาศัยวงจรกรองสัญญาณเชิงเลขแบบอิมพัลส์จำกัด (FIR Filter) โดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู (LUT-less Distributed Arithmetic Structure) ซึ่งจะออกแบบและจำลองการทำงานของฟิลเตอร์แบงก์ผ่านทางโปรแกรมแมตแล็บ (MATLAB) รวมถึงทำการทดสอบวงจรย่อยภายในฟิลเตอร์แบงก์ด้วยโปรแกรม ModelSim แล้วทดสอบการทำงานของระบบทั้งหมด

ABSTRACT

This project is to study and design filter banks for image processing. First, the input image will be stored in ROM on board then it will be fed to filter

banks. Finally, filter banks output will be displayed on VGA monitor. All parts of this project will be implemented via FPGA environment. In filter banks, we use FIR filter using LUT-less distributed arithmetic structure. Most of the simulations carry out via MATLAB except all hardware designs will be simulated by ModelSim.

# สารบัญ

# สารบัญ (ต่อ)

# สารบัญรูป

หน้า

รูปที่

# สารบัญรูป (ต่อ)

หน้า

รูปที่

# สารบัญรูป (ต่อ)

# สารบัญรูป (ต่อ)

หน้า

รูปที่

# สารบัญรูป (ต่อ)

หน้า

รูปที่

# สารบัญรูป (ต่อ)

# สารบัญรูป (ต่อ)

หน้า

รูปที่

# สารบัญรูป (ต่อ)

# สารบัญรูป (ต่อ)

หน้า

รูปที่

# สารบัญตาราง

หน้า

ตารางที่

# บทที่ 1

# บทนำ

## 1.1 ความเป็นมาและความสำคัญของปัญหา

ฟิลเตอร์แบงก์ (Filter Banks) ได้ถูกนำมาประยุกต์ใช้งานในการประมวลผลสัญญาณ เช่น การบีบอัดสัญญาณเสียง การบีบอัดรูปภาพ และการบีบอัดภาพเคลื่อนไหว อีกทั้งการใช้งานในอุปกรณ์พกพา ซึ่งถูกจำกัดด้วยขนาดที่เล็ก และการทำงานถูกจำกัดด้วยขนาดของแบตเตอรี่ที่ ดังนั้นในการออกแบบและสร้างฟิลเตอร์แบงก์จึงต้องมีคุณสมบัติที่เล็ก สามารถทำงานได้ที่ความเร็วสูง ประหยัดพลังงาน รวมทั้งมีประสิทธิภาพที่ดี

ในปริญญานิพนธ์นี้จะศึกษาการนำฟิลเตอร์แบงก์มาใช้ในการประมวลผลสัญญาณภาพ โดยจะใช้ในการเข้ารหัสในระบบการเข้ารหัสมาตรฐาน JPEG2000 ซึ่งจะเป็นชุดค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 สำหรับการบีบอัดข้อมูลแบบมีการสูญเสีย (Lossy) ในส่วนของโครงสร้างฟิลเตอร์แบงก์จะเลือกใช้โครงสร้างทางฮาร์ดแวร์แบบปราศจากตัวคูณ (Multiplier-less) ที่มีคุณสมบัติที่สอดคล้องกับความต้องการข้างต้น เพราะในการออกแบบด้วยโครงสร้างแบบทั่วไปต้องใช้จำนวนอุปกรณ์ตัวคูณและดีเลย์ที่เมื่อมีจำนวนมากจะส่งผลให้ต้องใช้พลังงานสูงและทำให้ประมวลผลได้ช้าลง ดังนั้นปริญญานิพนธ์จึงเน้นในการออกแบบเพื่อให้ได้วงจรที่มีขนาดเล็ก ใช้พลังงานต่ำ และใช้ในการประมวลผลความเร็วสูง

## 1.2 วัตถุประสงค์

ปริญญานิพนธ์นี้จะทำการศึกษา วิเคราะห์ และทำการสร้างฟิลเตอร์แบงก์ด้วยอุปกรณ์ FPGA เพื่อให้ได้ฮาร์ดแวร์ที่ประมวลผลสัญญาณภาพได้โดยมีวัตถุประสงค์ดังนี้

1) ศึกษาทฤษฎีและหลักการออกแบบฟิลเตอร์แบงก์

2) ศึกษาการจำลองการทำงานในการประมวลผลสัญญาณภาพของฟิลเตอร์แบงก์ด้วยโปรแกรมแมตแล็บ

3) ศึกษาการนำโครงสร้างแบบเลขคณิตกระจายมาประยุกต์ใช้กับฟิลเตอร์แบงก์

4) ศึกษาการทำงานในการประมวลผลสัญญาณภาพของฟิลเตอร์แบงก์บนอุปกรณ์ FPGA



5) ศึกษาการใช้งานหน่วยความจำภายนอก (ROM) ร่วมกับอุปกรณ์ FPGA

6) ศึกษาการแสดงผลบนจอแสดงผลวีจีเอ (VGA Monitor)

## 1.3 ขอบเขตของปริญญานิพนธ์

ในปริญญานิพนธ์นี้มีโครงสร้างโดยรวมของชิ้นงานแสดงได้ดังรูปที่ 1.1 เริ่มต้นด้วยจะทำการออกแบบและสร้างฟิลเตอร์แบงก์โดยใช้วงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดที่ออกแบบด้วยโครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู ซึ่งจะใช้ชุดค่าสัมประสิทธิ์เป็นแบบ CDF 9/7 แล้วทำการออกแบบโครงสร้างฮาร์ดแวร์ด้วยภาษา VHDL จำลองการทำงานอุปกรณ์ย่อยภายในฟิลเตอร์แบงก์ด้วยโปรแกรม ModelSim และจำลองการทำงานในการประมวลผลสัญญาณภาพผ่านโปรแกรมแมตแล็บ ซึ่งผลการจำลองการทำงานที่ได้จะนำมาเปรียบเทียบกับผลการทดสอบการทำงานจริงบนอุปกรณ์ FPGA โดยวงจรฟิลเตอร์แบงก์จะเรียกภาพที่ถูกเก็บไว้ในหน่วยความจำภายนอกนำมาประมวลผลแล้วแสดงผลบนจอแสดงผลวีจีเอ


รูปที่ 1.1 โครงสร้างโดยรวมของชิ้นงานในปริญญานิพนธ์

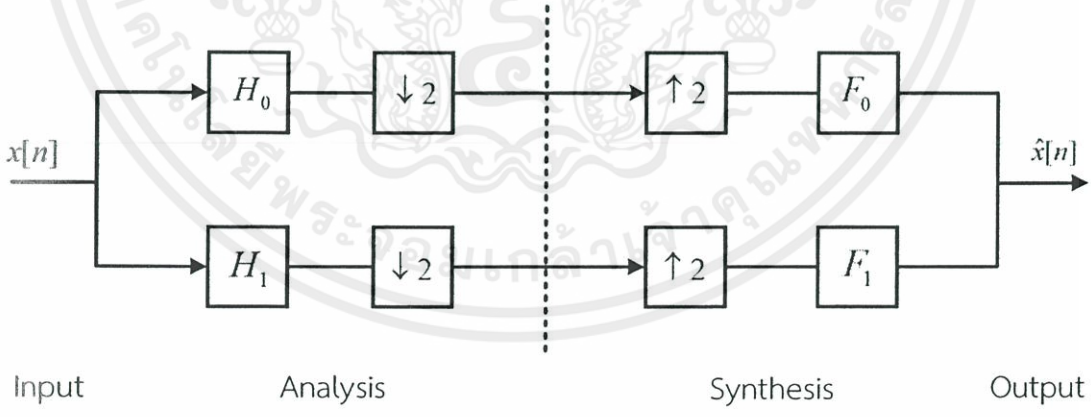
# บทที่ 2

# ทฤษฎีและหลักการที่เกี่ยวข้อง

ในการออกแบบและสร้างฟิลเตอร์แบงก์เพื่อใช้ในการประมวลผลสัญญาณภาพ จะทำการประยุกต์ใช้ความรู้พื้นฐาน และทฤษฎีเบื้องต้นซึ่งจะประกอบไปด้วยความรู้เบื้องต้นเกี่ยวกับฟิลเตอร์แบงก์ การประยุกต์ใช้งานฟิลเตอร์แบงก์กับการประมวลผลสัญญาณภาพ หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจาย และการออกแบบวงจรดิจิตอลด้วยภาษา VHDL ดังต่อไปนี้

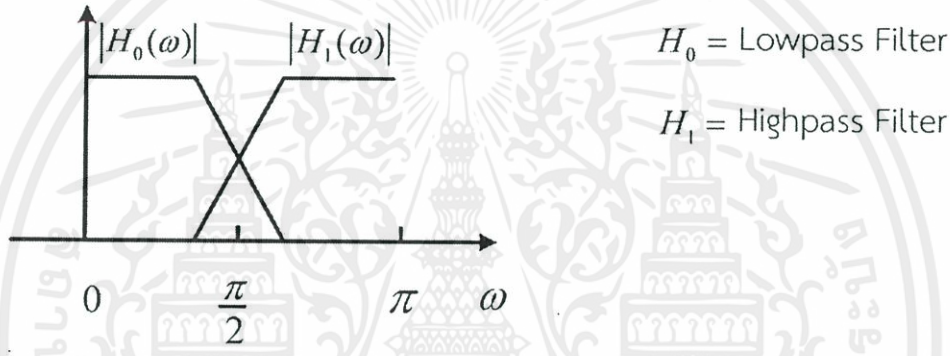
## 2.1 ความรู้เบื้องต้นเกี่ยวกับฟิลเตอร์แบงก์ (Filter Banks) [1,3,4,9]

ฟิลเตอร์แบงก์คือเซตของวงจรกรองความถี่ที่เชื่อมต่อกับตัวสุ่มค่าสัญญาณหรือบางครั้งอาจเป็นตัวหน่วงสัญญาณ (Delay) โดยที่ตัวสุ่มค่าสัญญาณจะมี 2 ลักษณะคือตัวสุ่มค่าสัญญาณที่ทำการลดค่าสุ่มตัวอย่างของสัญญาณอินพุต (Decimator) และตัวสุ่มค่าสัญญาณที่ทำการเพิ่มค่าสุ่มตัวอย่างของสัญญาณอินพุต (Expander) ของฟิลเตอร์แบงก์ ซึ่งส่วนของวงจรกรองความถี่ด้านวิเคราะห์ (Analysis Filter) จะประกอบด้วยวงจรกรองความถี่ต่ำและวงจรกรองความถี่สูงโดยจะแทนด้วย $H_0$ และ $H_1$ ตามลำดับดังรูปที่ 2.1




รูปที่ 2.1 ฟิลเตอร์แบงก์ [9]

จากรูปที่ 2.1 $H_0$ คือวงจรกรองความถี่ต่ำ และ $H_1$ คือวงจรกรองความถี่สูง โดยมีผลตอบสนองเชิงความถี่แสดงดังรูปที่ 2.2 ซึ่งจะเห็นได้ว่าทั้งสองวงจรไม่เป็นวงจรกรองความถี่ในอุดมคติเนื่องจากมีผลตอบสนองเชิงความถี่บางส่วนเกิดการซ้อนทับกัน (Overlap) จึงทำให้สเปกตรัมมีการซ้อนทับกัน (Aliasing) ในแต่ละช่องสัญญาณรวมถึงยังเกิดการลดทอนของสัญญาณในเชิงขนาดและเฟส โดยในที่นี้จะกล่าวถึงเฉพาะการลดทอนสัญญาณในเชิงขนาดเท่านั้น (Amplitude Distortion) ดังนั้นในส่วนของวงจรกรองความถี่ด้านสังเคราะห์ (Synthesis Filter) $F_0$ และ $F_1$ จึงต้องปรับให้เข้ากับวงจรกรองความถี่ด้านวิเคราะห์ด้วยเพื่อกำจัดค่าผิดพลาดที่เกิดขึ้น




รูปที่ 2.2 ผลตอบสนองเชิงความถี่ของวงจรกรองความถี่ต่ำและสูง [9]

จุดสำคัญของฟิลเตอร์แบงก์คือการหาเงื่อนไขการกู้กลับอย่างสมบูรณ์ (Perfect Reconstruction Conditions) ซึ่งหมายถึงว่าฟิลเตอร์แบงก์ต้องเป็นฟังก์ชันเชิงตั้งฉากคู่ (Biorthogonal) นั่นคือวงจรกรองความถี่ด้านสังเคราะห์ที่ประกอบด้วย $F_0$, $F_1$ และ $\uparrow 2$ จะต้องเป็นส่วนกลับของวงจรกรองความถี่ด้านวิเคราะห์

หลักการของการกู้กลับอย่างสมบูรณ์เป็นคุณสมบัติที่สำคัญมากซึ่งประกอบด้วยสองเงื่อนไขคือการกำจัดการซ้อนทับกันของสเปกตรัมและการป้องกันไม่ให้เกิดการผิดเพี้ยนของสัญญาณ โดยถ้าในกรณีที่ตัวสุ่มค่าสัญญาณทั้ง $(\uparrow 2)$ และ $(\downarrow 2)$ ไม่ถูกแสดงในสมการ การกู้กลับอย่างสมบูรณ์แบบมีค่าความหน่วง l ขั้นสามารถเขียนอยู่ในรูป Z-domain ได้ดังสมการที่ 2.1

$$F_0(z)H_0(z)+F_1(z)H_1(z)=z^{-l} \tag{2.1}$$

แต่เมื่อนำ $(\uparrow 2)$ และ $(\downarrow 2)$ มาพิจารณาด้วย จะเป็นการกำจัดค่าศูนย์ออกจากพจน์คี่ จึงคงเหลือไว้เฉพาะพจน์คู่ดังสมการที่ 2.2 และในส่วนของด้านความถี่สูงก็จะเป็นเช่นเดียวกันดังสมการที่ 2.3

$$(\downarrow 2)(\uparrow 2)H_0x=\frac{1}{2}\left[H_0(z)X(z)+H_0(-z)X(-z)\right] \tag{2.2}$$

$$(\downarrow 2)(\uparrow 2)H_1x=\frac{1}{2}\left[H_1(z)X(z)+H_1(-z)X(-z)\right] \tag{2.3}$$

สำหรับส่วนของวงจรกรองความถี่ด้านสังเคราะห์จะทำการคูณสมการที่ 2.2 ด้วย $F_0$ และคูณสมการที่ 2.3 ด้วย $F_1$ ทำให้ได้สมการที่ 2.4 และสมการที่ 2.5 ตามลำดับ

วงจรกรองความถี่ต่ำ คือ $\frac{1}{2}F_0(z)\left[H_0(z)X(z)+H_0(-z)X(-z)\right]$ (2.4)

วงจรกรองความถี่สูง คือ $\frac{1}{2}F_1(z)\left[H_1(z)X(z)+H_1(-z)X(-z)\right]$ (2.5)

หลังจากนั้นนำสัญญาณเอาต์พุตที่ได้จากทั้งสองวงจรมารวมเข้าด้วยกันเพื่อหาสัญญาณ $\hat{x}[n]$ หรือในรูป Z-domain คือ $\hat{X}(z)$ ดังสมการที่ 2.6

$$\begin{aligned}\hat{X}(z)=&\frac{1}{2}\left[F_0(z)H_0(z)+F_1(z)H_1(z)\right]X(z)\\&+\frac{1}{2}\left[F_0(z)H_0(-z)+F_1(z)H_1(-z)\right]X(-z)\end{aligned} \tag{2.6}$$

จากสมการที่ 2.6 จะพบว่าในส่วนเทอมหลังจะเป็นส่วนของการซ้อนทับของความถี่ที่เกิดขึ้นที่ต้องกำจัดทำให้ได้เงื่อนไขการกำจัดการซ้อนทับ (Alias Cancellation) ดังสมการที่ 2.7

$$F_0(z)H_0(-z)+F_1(z)H_1(-z)=0 \tag{2.7}$$

ดังนั้นได้เอาต์พุตดังสมการที่ 2.8

$$\hat{X}(z)=\frac{1}{2}\left[F_0(z)H_0(z)+F_1(z)H_1(z)\right]X(z) \tag{2.8}$$

จากคุณสมบัติของฟิลเตอร์แบงก์ที่มีการกู้กลับอย่างสมบูรณ์นี้ทำให้ได้ความสัมพันธ์ระหว่างสัญญาณอินพุตและเอาต์พุตในรูป Z-domain คือ $\hat{X}(z)=z^{-l}X(z)$ ทำให้ได้สมการใหม่ดังสมการที่ 2.9

$$\frac{1}{2}\left[F_0(z)H_0(z)+F_1(z)H_1(z)\right]X(z)=z^{-l}X(z) \tag{2.9}$$

ดังนั้นจะได้เงื่อนไขการปราศจากการลดทอน (No Distortion) ดังสมการที่ 2.10

$$F_0(z)H_0(z)+F_1(z)H_1(z)=2z^{-l} \tag{2.10}$$

จากเงื่อนไขของการกู้กลับอย่างสมบูรณ์ทั้งสองข้อทำให้ได้ความสัมพันธ์ของฟิลเตอร์แต่ละตัวดังสมการที่ 2.11 และสมการที่ 2.12

$$F_0(z)=H_1(-z) \tag{2.11}$$

$$F_1(z)=-H_0(-z) \tag{2.12}$$

## 2.2 หลักการแยกการทำคอนโวลูชั่นแบบ 2 มิติ (Separable 2D Convolution)

ในการประมวลผลสัญญาณภาพเชิงทฤษฎีจะอาศัยหลักการทำคอนโวลูชั่นแบบ 2 มิติ (2D Convolution) ซึ่งจะเป็นการคำนวณค่าอินพุตกับค่าสัมประสิทธิ์ของตัวกรองในลักษณะเมตริกซ์ $M\times N$ มิติ ที่จะคำนวณเมตริกซ์ที่เป็นแถวและหลักพร้อมกัน ดังสมการที่ 2.13



$$y[m,n]=h[m,n]*x[m,n]=\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}h[i,j]x[m-i,n-j] \tag{2.13}$$

โดยที่ $M$ แทนมิติเมตริกซ์ในแนวแถว $N$ แทนมิติเมตริกซ์ในแนวหลัก

แต่เมื่อมองการประมวลผลสัญญาณภาพในเชิงของฮาร์ดแวร์จะพบว่าการทำตัวอุปกรณ์การทำคอนโวลูชั่นแบบ 2 มิติ ที่มีขนาด $N \times N$ มิติ จะต้องใช้ตัวคูณและตัวบวกจำนวนมากในลักษณะที่เพิ่มแบบเอกซ์โพเนนเชียล (Exponential) ด้วยเหตุนี้ทำให้ต้องมีการลดจำนวนตัวคูณและตัวบวกลงด้วยการใช้หลักการแยกการทำคอนโวลูชั่นแบบ 2 มิติ ให้มาเป็นการทำคอนโวลูชั่นแบบ 1 มิติ (1D Convolution) จำนวน 2 ครั้ง โดยอาศัยคุณสมบัติที่สำคัญของการทำคอนโวลูชั่นคือ Separable Convolution ซึ่งจะทำคอนโวลูชั่นในแนวแถวของเมตริกซ์ แล้วจึงนำมาทำคอนโวลูชั่นในแนวคอลัมน์ของเมตริกซ์ตามลำดับ

ดังนั้นจากสมการที่ 2.13 ถ้า $h[m,n]$ มีคุณสมบัติการแยกตัวได้ (Separable) เป็นเมตริกซ์ที่มีมิติ $M \times 1$ และ $1 \times N$ จะทำให้ได้สมการที่ 2.14

$$h[m,n] = h_1[m] * h_2[n] \tag{2.14}$$

ทำการแทนค่า $h[m,n]$ จากสมการที่ 2.14 ลงในสมการที่ 2.13 แล้วจัดรูปสมการได้ดังสมการที่ 2.15

$$\begin{aligned} y[m,n] &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} h[i,j]x[m-i,n-j] \\ &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} \left(h_1[i]h_2[j]\right)x[m-i,n-j] \\ &= \sum_{j=-\infty}^{\infty} h_2[j]\left[\sum_{i=-\infty}^{\infty} h_1[i]x[m-i,n-j]\right] \end{aligned} \tag{2.15}$$

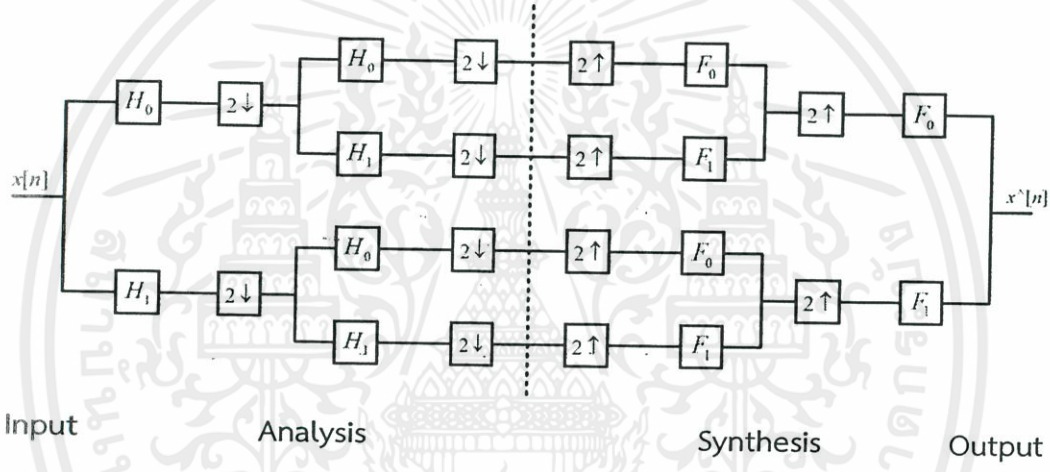
จากสมการผลต่างสืบเนื่องแบบ 1 มิติ ตามสมการที่ 2.16

$$y[n] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{2.13}$$

เมื่อนำสมการที่ 2.16 เทียบกับสมการที่ 2.15 ทำให้จัดรูปสมการได้สมการที่ 2.17

$$y[m,n] = h_2[n] * (h_1[m] * x[m,n]) \tag{2.17}$$

จากสมการที่ 2.17 จะพบว่า $y[m,n]$ จะเกิดจากการทำคอนโวลูชั่นแบบ 1 มิติ 2 ครั้ง โดยเริ่มจากการทำคอนโวลูชั่นในแนวแถว แล้วตามด้วยการทำคอนโวลูชั่นในแนวคอลัมน์ตามลำดับ ดังนั้นในการออกแบบตัวกรองที่จะใช้จึงจำเป็นต้องมีคุณสมบัติการแยกตัวได้ด้วย ซึ่งพบว่ามีหนึ่งในตัวกรองที่มีคุณสมบัตินี้คือ ฟิลเตอร์แบงก์แบบ 2 ระดับ (Two-Stage Filter Banks) ที่มีลักษณะดังรูปที่ 2.3




รูปที่ 2.3 ฟิลเตอร์แบงก์แบบ 2 ระดับ

## 2.3 หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจาย [8]

โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) หรือเรียกอย่างย่อว่า DA เป็นการจัดรูปแบบทางคณิตศาสตร์ของสมการที่อยู่ในรูปผลบวกของผลคูณ (Sum of Product) ของเลขฐานสิบให้กระจายออกเป็นบิตหรือในรูปแบบของเลขฐานสอง เพื่อให้การคำนวณทางคณิตศาสตร์สามารถแปลงเป็นวงจรดิจิตอลอิเล็กทรอนิกส์ได้ โดยจะปรากฏอยู่ในงานทางด้านประมวลผลสัญญาณเชิงเลข หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจายที่นำมาใช้งานด้านการประมวลผลสัญญาณเชิงเลขคือ การแปลงฟังก์ชันถ่ายโอน (Transfer Function) ซึ่งเป็นสมการที่อยู่ในรูปผลบวกของผลคูณระหว่างสัญญาณอินพุตกับค่าสัมประสิทธิ์ของฟังก์ชันถ่ายโอนของระบบ โดยในการคูณกันระหว่างค่าสัมประสิทธิ์ของฟังก์ชั่นถ่ายโอนกับสัญญาณอินพุตจะใช้การคูณเลขฐานสอง

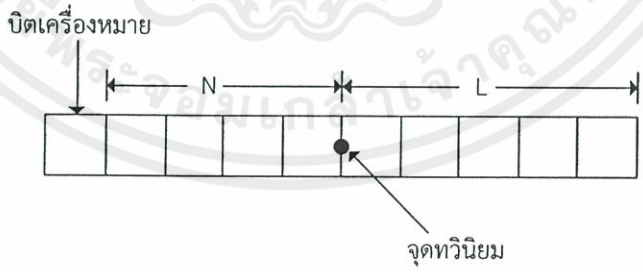
แบบส่วนเติมเต็มสอง (2's Complement) และการคูณจะใช้แบบเปิดตาราง (Look-Up Table) โดยค่าผลบวกของผลคูณระหว่างสัมประสิทธิ์และสัญญาณอินพุตจะถูกเก็บไว้ในหน่วยความจำ EPROM และจะใช้สัญญาณอินพุตเป็นแอดเดรสของ EPROM โดยตรง ทั้งค่าสัมประสิทธิ์ของวงจรกรองและสัญญาณอินพุตจะถูกแทนด้วยเลขส่วนเติมเต็มสอง เพราะฉะนั้นโครงสร้างเลขคณิตกระจายจึงมีทฤษฎีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง

### 2.3.1 ระบบตัวเลข

สำหรับระบบเชิงเลขตัวเลขจะถูกแทนด้วยเลขฐานสองซึ่งโดยทั่วไปมีรูปแบบที่ นิยมใช้กันอยู่ 2 รูปแบบ คือ รูปแบบจำนวนโดยตรง (Fixed Point Format) และรูปแบบจำนวนอิงดรรชนี (Floating Point Format) ซึ่งรูปแบบจำนวนโดยตรงจะมีวงจรฮาร์ดแวร์ที่ใช้ในการคำนวณที่ง่ายกว่า แต่ให้ค่าจากการคูณค่อนข้างจำกัด ส่วนรูปแบบจำนวนอิงดรรชนีจะสามารถแทน ค่าของสัญญาณคือให้ย่านพลวัตร (Dynamic Range) ได้มากกว่า แต่ต้องใช้วงจรฮาร์ดแวร์ที่สลับซับซ้อน แพงกว่า และให้ความเร็วในการประมวลผลที่ลดลง

#### 2.3.1.1 รูปแบบจำนวนโดยตรง

รูปแบบจำนวนโดยตรงปกติจะประกอบไปด้วย 3 ส่วนคือ บิตเครื่องหมาย (Sign Bit) 1 บิต บิตจำนวนเต็ม (Inteleger Bit) N บิต และบิตเศษส่วน (Fractional Bit) L บิต โดยจะมีจุดทวินิยม (Binary Point) อยู่ระหว่างบิตจำนวนเต็มและบิตเศษส่วนดังแสดงในรูปที่ 2.4

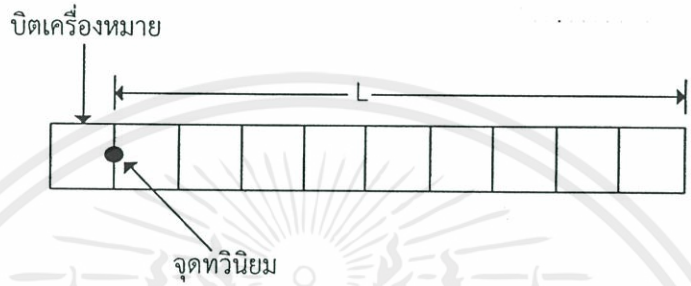



รูปที่ 2.4 การจัดรูปแบบจำนวนโดยตรงที่ประกอบด้วยบิตจำนวนเต็มและบิตเศษส่วน [8]



จำนวนบิต N เป็นตัวกำหนดย่านพลวัตร ที่ต้องการโดยถ้าเลือกให้มีจำนวนน้อยอาจทำให้เกิดการล้น (Overflow) จากการคำนวณได้ แต่ถ้าเลือกให้มีจำนวนมากความ

เที่ยงตรงก็จะน้อยลง ซึ่งในการสร้างวงจรกรองสัญญาณเชิงเลขโดยการแทนด้วยรูปแบบจำนวนโดยตรงนั้นนิยมที่จะทำมาตราส่วน (Scaling) เพื่อให้ขนาดของสัญญาณมีค่าอยู่ระหว่าง $-1 \le x < 1$ คือมีบิตเครื่องหมาย 1 บิต และบิตเศษส่วน L บิต ดังแสดงในรูปที่ 2.5




รูปที่ 2.5 การจัดรูปแบบจำนวนโดยตรงที่มีแต่บิตเศษส่วน [8]

โดยทั่วไปเลขฐานสองแบบจำนวนโดยตรงแบ่งออกได้เป็น 3 รูปแบบด้วยกัน คือ (1) แบบขนาดและเครื่องหมาย (Sign Magnitude) (2) แบบส่วนเติมเต็มหนึ่ง (1's Complement) และ (3) แบบส่วนเติมเต็มสอง (2's Complement) โดยคุณลักษณะที่สำคัญบางประการของ การแทนตัวเลขด้วยเลขฐานสองแบบจำนวนโดยตรงทั้ง 3 รูปแบบสามารถสรุปได้ดังตารางที่ 2.1

ตารางที่ 2.1 คุณสมบัติที่สำคัญของรูปแบบจำนวนโดยตรง [8]

| Features | Sign and magnitude | 2's complement | 1's complement |
|---|---|---|---|
| Range | $-\left(1-2^{-L}\right) \le x \le \left(1-2^{-L}\right)$ | $-1 \le x \le \left(1-2^{-L}\right)$ | $-\left(1-2^{-L}\right) \le x \le \left(1-2^{-L}\right)$ |
| Representation of zero | 0.000 and 1.000 | 0.000 | 0.000 and 1.111 |

ตารางที่ 2.1 คุณสมบัติที่สำคัญของรูปแบบจำนวนโดยตรง [8] (ต่อ)

| Features | Sign and magnitude | 2's complement | 1's complement |
|---|---|---|---|
| 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 | Not so good | Excellent | Good |

ในรูปแบบนี้ตัวเลขแบบส่วนเติมเต็มสองเป็นที่นิยมใช้กันมากในระบบการประมวลผลสัญญาณเชิงเลข ทั้งนี้เนื่องมาจากมีการแทนค่าเลขศูนย์ได้เพียงค่าเดียว การสร้างวงจรฮาร์ดแวร์สำหรับการบวก ลบ และคูณของเลขส่วนเติมเต็มสองทำได้ง่ายโดยในการคูณสามารถใช้หลักการเลื่อนและบวก (Shift and Add) และในระหว่างผลการบวกย่อย (Partial Sum) ของการบวกเลขส่วนเติมเต็มสอง สามหรือสี่ จำนวนถึงแม้อาจจะเกิดการล้น (ตัวทดจากผลการบวกล้นข้ามไปทับบิตเครื่องหมาย) 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 |

2.3.1.2 รูปแบบจำนวนอิงดรรชนี

รูปแบบจำนวนโดยตรงมีข้อเสียที่สำคัญ 2 ประการ คือ

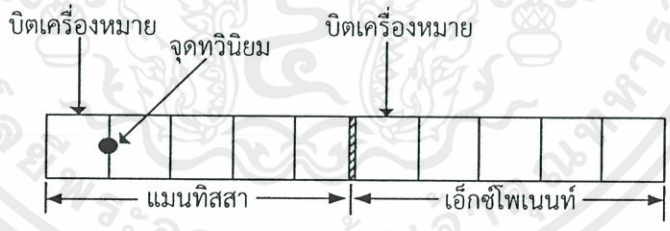
1) ย่านพลวัตรของตัวเลขมีค่าน้อย เช่น การแทนด้วยเลขส่วนเติมเต็มสอง ค่าที่น้อยที่สุดคือ $-1$ และค่าที่มากที่สุด คือ $1-2^{-L}$

2) เปอร์เซ็นต์ความผิดพลาดที่เกิดจากการตัด (Truncation) หรือการปัด (Rounding) จะเพิ่มมากขึ้นเมื่อขนาดของตัวเลขมีค่าลดลง ตัวอย่างเช่น ถ้าจำนวน 0.11011010 และ 0.000110101 ถูกตัดให้จำนวนบิตเศษส่วนเหลือเพียง 4 บิต เปอร์เซ็นต์ความผิดพลาดจะเป็น 4.59 % และ 39.6 % ตามลำดับ โดยข้อเสียนี้สามารถแก้ไขได้โดยการใช้รูปแบบจำนวนอิงดรรชนี ซึ่งตัวเลข $X$ แสดงได้ดังสมการที่ 2.18

$$X = M \times 2^{e} \tag{2.18}$$

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

$M$ และ $e$ เรียกว่า แมนทิสสา (Mantissa) และเอกซ์โพเนนท์ (Exponent) ตามลำดับ ตัวอย่างเช่น จำนวน 0.00110101 และ 01001.11 สามารถแทนได้โดย $0.110101 \times 2^{-2}$ และ $0.100111 \times 2^{4}$ ตามลำดับ ส่วนจำนวนที่มีค่าเป็นลบก็ทำในลักษณะเดียวกัน รูปแบบจำนวนอิงดรรชนีสามารถแสดงได้ดังรูปที่ 2.6 โดยแบ่งเป็น 2 ส่วน คือส่วนหนึ่งสำหรับแมนทิสสา และอีกส่วนสำหรับเอกซ์โพเนนท์




รูปที่ 2.6 การจัดรูปแบบจำนวนอิงดรรชนี [8]

ข้อดีของการใช้จำนวนอิงดรรชนีคือแทนค่าของสัญญาณได้ละเอียดกว่าและแม่นยำกว่าแบบจำนวนโดยตรง แต่การบวก ลบ หรือคูณจะยุ่งยากกว่ามาก วงจรจึงซับซ้อนและแพงกว่าแบบจำนวนโดยตรงมาก นอกจากนี้ความเร็วในการประมวลผลยังช้ากว่าด้วย ดังนั้นสำหรับการประมวลผลแบบเวลาจริง (Real Time) จึงนิยมใช้ระบบตัวเลขแบบจำนวนโดยตรง

### 2.3.2 ทฤษฎีเลขคณิตกระจาย

จากที่ได้กล่าวมาแล้วว่าโครงสร้างเลขคณิตกระจายมีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง ดังนั้นในหัวข้อนี้จะได้อธิบายถึงหลักการของการคูณเลขส่วนเติมเต็มสอง

ให้เลขส่วนเติมเต็มสองของ $X$ ซึ่งแทนด้วย $\bar{X}$ และนิยามตามสมการที่ 2.19 โดย

$$\bar{X} = \begin{cases} X, & X \geq 0, \\ 2 - |X|, & X < 0. \end{cases} \tag{2.19}$$

โดย $X$ เป็นเลขที่เป็นเศษส่วน (Fractional Number)

ในระบบเลขส่วนเติมเต็มสองจะใช้บิตที่มีนัยสำคัญสูงสุด (MSB) เป็นบิตแสดงเครื่องหมาย โดยถ้าเป็นบวกแทนด้วย “0” และถ้าเป็นลบแทนด้วย “1” ถ้าให้ $X$ แทนด้วยเลขฐานสองขนาด $L+1$ บิต ดังนั้นรูปแบบของเลขส่วนเติมเต็มสองจะเขียนได้ดังสมการที่ 2.20

$$\bar{X} = X_0.X_1X_2\ldots X_L \tag{2.20}$$

ค่าของ $\bar{X}$ ในรูปของเลขฐานสิบสามารถหาได้ตามสมการที่ 2.21 ดังนี้

$$X = -X_0 + \sum_{i=1}^{L} X_i 2^{-i} \tag{2.21}$$

จากนั้นพิจารณาผลคูณของสมการที่ 2.22 ต่อไปนี้

$$Y = Xm \tag{2.22}$$

ให้ $\bar{Y}$, $\bar{X}$ และ $\bar{m}$ เป็นเลขส่วนเติมเต็มสองของ $Y$, $X$ และ $m$ ตามลำดับ จากนั้นพิจารณาจากสมการที่ 2.21 และสมการที่ 2.22 จะได้สมการที่ 2.23 และสมการที่ 2.24

$$Y = -Y_0 + \sum_{i=1}^{L} Y_i 2^{-i} \tag{2.23}$$

$$Y = -X_0 m + \sum_{i=1}^{L} X_i m 2^{-i} \tag{2.24}$$

ดังนั้นเมื่อทำการกระจายสมการที่ 2.24 จะได้สมการที่ 2.25

$\bar{Y}$ คือส่วนเติมเต็มสองของ $\left(-X_0 m + 2^{-1} X_1 m + 2^{-2} X_2 m + 2^{-3} X_3 m + \ldots + 2^{-L} X_L m\right)$ (2.25)

ทำการจัดรูปสมการที่ 2.25 ได้สมการที่ 2.26 ดังนี้

$\bar{Y}$ คือส่วนเติมเต็มสองของ $\left(-X_0 m + 2^{-1}\left(X_1 m + \ldots + 2^{-1}\left(X_{L-1} m + 2^{-1}\left(X_L m\right)\right)\right)\right)$ (2.26)

ต่อไปพิจารณาส่วนเติมเต็มสองของ $2^{-1}U$ สมการที่ 2.27

$$\bar{U} = U_0 . U_1 U_2 \ldots U_M \tag{2.27}$$

สำหรับ $U \geq 0$ (หรือ $U_0 = 0$) จะได้สมการที่ 2.28

ส่วนเติมเต็มสองของ $\left(2^{-1}U\right) = 2^{-1}\bar{U}$ (2.28)

และสำหรับ $U < 0$ (หรือ $U_0 = 1$) จะได้สมการที่ 2.29

ส่วนเติมเต็มสองของ $\left(2^{-1}U\right) = 2 - \left|2^{-1}U\right| = 1 + 2^{-1}\left(2 - |U|\right) = 1 + 2^{-1}\bar{U}$ (2.29)

ดังนั้นสรุปได้ดังสมการที่ 2.30 ว่า

ส่วนเติมเต็มสองของ $\left(2^{-1}U\right) = \begin{cases} 2^{-1}\bar{U}, & U_0 = 0, \\ 1 + 2^{-1}\bar{U}, & U_0 = 1. \end{cases}$ (2.30)

สมการที่ 2.30 นี้แสดงให้เห็นได้ว่า ส่วนเติมเต็มสองของ $\left(2^{-1}U\right)$ เป็นการเลื่อนข้อมูลของ $\bar{U}$ ไปทางขวา 1 บิต ดังนั้นสรุปได้เป็นสมการที่ 2.31

$$\therefore \text{ส่วนเติมเต็มสองของ } \left(2^{-1}U\right)=2_2^{-1}\bar{U} \tag{2.31}$$

จากสมการที่ 2.31 $2_2^{-1}\bar{U}$ แสดงถึงการเลื่อนข้อมูลของ $\bar{U}$ ไปทางขวา 1 บิต แบบเลขส่วนเติมเต็มสองซึ่งสัญลักษณ์ $2_2^{-1}$ (ซึ่งโดยทั่วไปนิยมเขียนเป็น $2^{-1}$) เป็นการแสดงว่าในกรณีที่ $\bar{U}$ เป็นเลขบวก ซึ่งบิตเครื่องหมายจะเป็นเลขศูนย์ โดยหลังจากเลื่อนข้อมูลไปทางขวา 1 บิตแล้ว บิตที่มาแทนบิต เครื่องหมายก็ยังคงเป็นเลขศูนย์ ส่วนในกรณีที่ $\bar{U}$ เป็นเลขลบ หลังจากเลื่อนข้อมูลไปทางขวา 1 บิต แล้ว บิตที่มาแทนบิตเครื่องหมายจะเป็นเลขหนึ่ง (จาก $1+2^{-1}\bar{U}$) ซึ่งในการสร้างเพื่อใช้งานจริง จำเป็นต้องมีวงจรที่ใช้ในการตรวจสอบบิตเครื่องหมาย (Sign Digit) ทุกครั้งที่มีการเลื่อนข้อมูล

จากนั้นพิจารณาสมการที่ 2.26 และสมการที่ 2.30 จะได้สมการที่ 2.32

$$\bar{Y}=-X_0\bar{m}+2^{-1}X_1\bar{m}+2^{-2}X_2\bar{m}+2^{-3}X_3\bar{m}+\ldots+2^{-L}X_L\bar{m} \tag{2.32}$$

ทำการจัดรูปสมการที่ 2.32 ได้สมการที่ 2.33

$$\bar{Y}=-X_0\bar{m}+2^{-1}\left(X_1\bar{m}+\ldots+2^{-1}\left(X_{L-1}\bar{m}+2^{-1}\left(X_L\bar{m}\right)\right)\right) \tag{2.33}$$

จากสมการที่ 2.33 จะเห็นได้ว่าผลคูณจากสมการที่ 2.22 สามารถหาได้โดยการใช้หลักการเลื่อนและบวก (Shift and Add) โดยผลลัพธ์ที่ได้จากการคูณแบบเลขส่วนเติมเต็มสองสามารถหาได้ตามขั้นตอนดังนี้

1) เคลียร์ค่าข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์

2) บวก $X_L\bar{m}$ กับค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์

3) เลื่อนค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ไปทางขวา 1 บิต

4) ทำซ้ำข้อ 2 และ 3 สำหรับค่า $X_{L-1},\ldots,X_1$

5) ลบค่า $X_0\bar{m}$ ออกจากค่าที่อยู่ในแอคคิวมูเลเตอร์รีจิสเตอร์ (ลบแบบเลขส่วนเติมเต็มสอง)

ตัวอย่างการทำงานตามอัลกอริธึมนี้ $Y = Xm = 0.8125(-0.390625)$ โดยสมมุติให้ใช้แอคคิวมูเลเตอร์รีจิสเตอร์ขนาด 12 บิต

$m = -0.390625$

$\bar{m} = 2 - |m|$ $\quad \because m$ เป็นเลขลบ

$\bar{m} = 2 - 0.390625$

$\bar{m} = 1.609375$

$\therefore \bar{m} = 1.100111$

$X = 0.8125 = \bar{X}$ $\quad \because X$ เป็นเลขบวก

$\therefore X = 0.1101 = X_0.X_1X_2X_3X_4$

โดยมีขั้นตอนการทำงาน ดังตารางที่ 2.2 ต่อไปนี้

ตารางที่ 2.2 ขั้นตอนการคูณเลขส่วนเติมเต็มสอง [8]

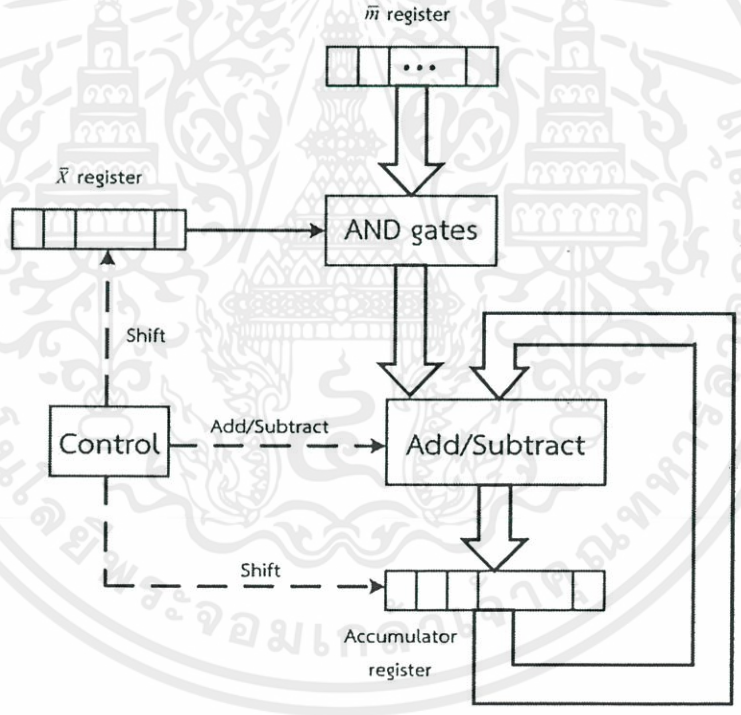
| การดำเนินการ | ข้อมูลในแอคคิวมูเลเตอร์รีจิสเตอร์ |
|---|---|
| เคลียร์ ACC | 0.000 0000 0000 |
| $ACC + X_4\bar{m}$ | 1.100 1110 0000 |
| เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิต | 1.110 0111 0000 |
| $ACC + X_3\bar{m}$ | 1.110 0111 0000 |
| เลื่อนข้อมูลใน ACC ไปทางขวา 1 บิต | 1.111 0011 1000 |
| $ACC + X_2\bar{m}$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

$$\therefore \overline{Y} = 1.101\ 0111\ 0110 = Y_0.Y_1Y_2\ldots Y_{11} \tag{2.34}$$

จากสมการที่ 2.34 จะได้

$$\begin{aligned} Y &= -Y_0 + \sum_{i=1}^{11} Y_i 2^{-i} \\ &= -1 + \left(2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-9} + 2^{-10}\right) \\ &= -0.3173828125 \end{aligned}$$

จากอัลกอริธึมดังกล่าวสามารถออกแบบการทำงานและสร้างวงจรแสดงได้ดังรูปที่ 2.7




รูปที่ 2.7 การคูณแบบเลขส่วนเติมเต็มสองโดยใช้เลขคณิตกระจาย [8]

ที่ผ่านมาเป็นหลักการคูณแบบเลขส่วนเติมเต็มสอง ส่วนทฤษฎีเลขคณิตกระจายก็อาศัยหลักการดังกล่าวมาใช้ โดยทำการกระจายสมการที่อยู่ในรูปผลบวกของผลคูณให้แตกออกมาอยู่ในระดับบิต (Bit Level) พิจารณาผลบวกของผลคูณของสมการที่ 2.35 ต่อไปนี้

$$Y = \sum_{i=0}^{N} m_i X_i \tag{2.35}$$

โดย $m_i$ คือค่าสัมประสิทธิ์ที่ซึ่งเป็นค่าคงที่

$X_i$ คือข้อมูลอินพุต

ถ้า $X_i$ แต่ละค่าเป็นเลขส่วนเติมเต็มสอง โดย $|X_i| < 1$ สามารถแสดง $X_i$ แต่ละค่าได้ดังสมการที่ 2.36

$$X_j = -X_{i0} + \sum_{j=1}^{L} X_{ij} 2^{-j} \tag{2.36}$$

โดย $X_{ij}$ คือบิตของข้อมูล $X_i$ มีค่าเป็น 0 หรือ 1

$X_{i0}$ คือบิตแสดงเครื่องหมาย

$X_{iL}$ คือบิตที่มีนัยสำคัญต่ำสุด (LSB)

$L+1$ คือจำนวนบิตที่แทนข้อมูลอินพุต

แทนค่า $X_i$ ในสมการที่ 2.36 ลงในสมการที่ 2.35 จะได้สมการที่ 2.37

$$Y = \sum_{i=0}^{N} m_i \left[ -X_{i0} + \sum_{j=1}^{L} X_{ij} 2^{-j} \right] \tag{2.37}$$

เมื่อจัดเทอมของผลบวกใหม่จะได้ดังสมการที่ 2.38

$$Y = -\sum_{i=0}^{N} X_{i0} m_i + \sum_{i=0}^{N} \sum_{j=1}^{L} X_{ij} m_i 2^{-j} \tag{2.38}$$

จากนั้นกระจายออกให้เป็นระดับบิต ได้สมการที่ 2.39 ดังนี้

$$
\begin{aligned}
Y = &-\left(X_{00}m_0 + X_{10}m_1 + X_{20}m_2 + \ldots + X_{N0}m_N\right) \\
&+2^{-1}\left(X_{01}m_0 + X_{11}m_1 + X_{21}m_2 + \ldots + X_{N1}m_N\right) \\
&+2^{-2}\left(X_{02}m_0 + X_{12}m_1 + X_{22}m_2 + \ldots + X_{N2}m_N\right) \\
&+\ldots+2^{-L}\left(X_{0L}m_0 + X_{1L}m_1 + X_{2L}m_2 + \ldots + X_{NL}m_N\right)
\end{aligned} \tag{2.39}
$$

จากสมการที่ 2.39 นี้ถูกกระจายออกให้อยู่ในรูปผลบวกของผลคูณระหว่างสัมประสิทธิ์กับข้อมูลอินพุตในระดับบิตซึ่งเป็นนิยามของการคำนวณแบบเลขคณิตกระจาย และเมื่อเปรียบเทียบสมการที่ 2.32 กับสมการที่ 2.27 จะเห็นว่าการคำนวณหาค่า $Y$ ก็ใช้เลขคณิตกระจายนั่นเองเพียงแต่นำค่าผลคูณย่อย (Partial Product) ที่คำนวณไว้ล่วงหน้าแล้วสำหรับแต่ละค่าที่สอดคล้องกับแต่ละบิตของข้อมูลอินพุตไปเก็บไว้ในตารางเปิดดู ซึ่งเป็นหน่วยความจำ EPROM และใช้ข้อมูลอินพุตเป็นแอดเดรสของหน่วยความจำเพื่อนำค่าในตารางเปิดดูมาผ่านขั้นตอนการคำนวณตามบู๊ทอัลกอริธึมซึ่งค่าในตารางเปิดดูสามารถแสดงในตารางที่ 2.3 ได้ดังนี้

ตารางที่ 2.3 ค่าผลคูณย่อยที่เก็บไว้ในตารางเปิดดูที่กำหนดโดยข้อมูลอินพุต [8]

| Bit Pattern ของข้อมูลอินพุต $X_{Nj} \ldots X_{2j}\ X_{1j}\ X_{0j}$ | 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 | $m_N + m_{N-1} + \ldots + m_2 + m_1 + m_0$ |

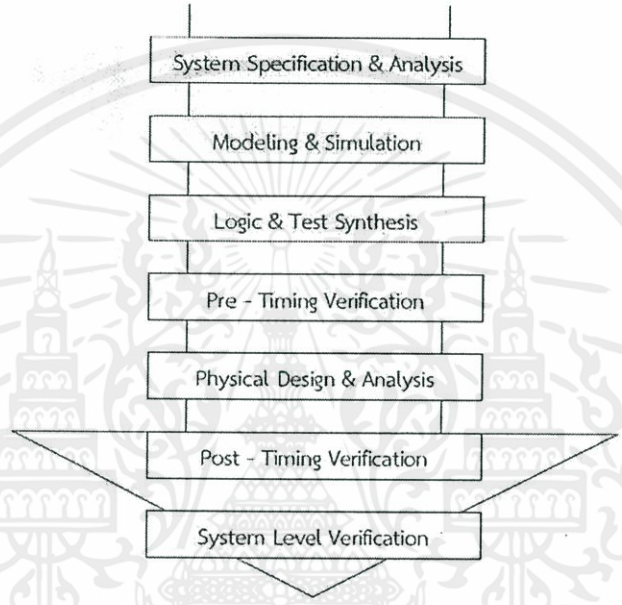
## 2.4 การออกแบบวงจรดิจิตอลด้วยภาษา VHDL [7,8]

ในการออกแบบวงจรดิจิตอล (Digital Circuit) นั้น ในปัจจุบันก้าวหน้าไปอย่างมาก โดยการใช้ภาษาบรรยายการทำงานของวงจร (Hardware Description Language: HDL) ซึ่งเป็นภาษาที่ใช้สำหรับออกแบบฮาร์ดแวร์โดยภาษาที่เป็นมาตรฐานสากล เช่น 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 มาบรรยายการทำงานของ วงจรที่ได้ออกแบบไว้ ซึ่งในโครงงานนี้ได้ใช้ภาษา VHDL มาทำการออกแบบวงจรกรองสัญญาณ เชิงเลขทำให้ลดความยุ่งยากในการนำเอาอุปกรณ์มาเชื่อมต่อให้เป็นวงจร รวมทั้งลดเวลาที่ใช้ในการ ออกแบบและทดสอบการทำงาน ซึ่งมีความแตกต่างเป็นอย่างมากเมื่อเปรียบเทียบกับการออกแบบ ในอดีตที่ผ่านมาคือผู้ออกแบบจะต้องนำเอาอุปกรณ์แต่ละตัวที่ทำการออกแบบไว้ มาทำการต่อ ทดลองในแผงวงจรจริงและทำการทดสอบวงจรเพื่อหาข้อผิดพลาด ซึ่งต้องใช้เวลานานกับการแก้ ปัญหาแต่ละอย่างที่เกิดขึ้น แต่ในการออกแบบด้วยภาษา VHDL ผู้ออกแบบเพียงแต่เขียนซอสโค้ด (Source Code) บรรยายการทำงานของวงจร หลังจากนั้นก็ทำการคอมไพล์ (Compile) แล้ว จำลองการทำงาน (Simulate) ดูว่าได้ฟังก์ชันการทำงานและไทม์มิ่ง (Timing) ตามที่ต้องการหรือ ไม่ จากนั้นก็นำซอสโค๊ดที่ได้ไปทำการสังเคราะห์ด้วยโปรแกรมสังเคราะห์ (Synthesis Tool) สุดท้ายนำวงจรที่ได้จากการสังเคราะห์ไปทำการแม็ป (Map) ลงไปยัง FPGA (Field-Programmable Gate Array) เพื่อเป็นชิป (Chip) ต้นแบบสำหรับการนำไปทดสอบการทำงาน

### 2.4.1 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมเชิงเลขขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของบล็อกไดอะแกรมก่อน จากนั้นจึงวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายการทำงานของแต่ละบล็อกและวิเคราะห์การทำงานแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์ เพื่อให้ได้การทำงานตามที่ต้องการโดยการออกแบบในลักษณะนี้ เรียกว่าหลักการออกแบบจากบนลงล่าง (Top-Down Design) ดังรูปที่ 2.8 ซึ่งถ้าเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาในการออกแบบมากกว่า เพราะเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบจำลองการทำงานตรวจสอบ

ความถูกต้อง ซึ่งใช้เวลามากและถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากในการออกแบบลักษณะนี้ ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นวิธีการที่เหมาะสมสำหรับการออกแบบและพัฒนาวงจรที่มีความซับซ้อนมากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ




รูปที่ 2.8 ขั้นตอนการออกแบบ [8]

จากรูปที่ 2.8 แสดงให้เห็นถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย โดยขั้นตอนของการออกแบบจากบนลงล่างมีรายละเอียดดังนี้

1) ขั้นตอนการสร้างข้อกำหนดของความต้องการและวิเคราะห์ระบบเพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2) ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงานเพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3) ขั้นตอนการสังเคราะห์ซึ่งจะต้องทำการกำหนดเทคโนโลยีที่จะมารองรับวงจรที่ออกแบบ และระบบช่วยออกแบบจะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่

ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท (Gate Level) และการเชื่อมต่อกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของเน็ทลิสต์ (Net List) ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้

4) หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือเน็ทลิสต์แล้ว ข้อมูลที่ได้นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงานในเรื่องของความถูกต้องของฟังก์ชันแล้ว ยังมีข้อมูลที่เกี่ยวกับเวลาด้วย ซึ่งจากความจริงที่ว่าอุปกรณ์อิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของการเคลื่อนผ่าน (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับนาโนวินาที แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันจำนวน 10,000 เกท ขึ้นไปเวลาดังกล่าวนี้จะสะสมกันมากขึ้นจนอาจจะทำให้การทำงานของวงจรทั้งหมดผิดไป หรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาสูงได้

5) ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and Device Mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของอุปกรณ์ FPGA หรือวงจรรวม ASIC

6) หลังจากที่ได้วงจรจริงมาแล้ว ยังต้องมีความจำเป็นที่จะต้องตรวจสอบการทำงานที่คำนึงถึงเวลาด้วยเพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นให้เป็นระบบ เพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วยอินพุตและเอาต์พุตแพด (Pad) ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก

7) หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นให้เป็นระบบแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นอีกครั้ง ซึ่งเป็นการทดสอบการทำงานจริงขั้นสุดท้าย

### 2.4.2 ภาษา VHDL และส่วนประกอบของภาษา

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

ในการเขียนรูปแบบบรรยายระบบเชิงเลขในลักษณะของการออกแบบจากบนลงล่าง จะต้องทำความเข้าใจในเรื่องของโครงสร้างและส่วนประกอบต่างๆ ของรูปแบบภาษา VHDL เสียก่อน ซึ่งส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียนมี 4 หน่วยคือ

1) หน่วยการออกแบบเอนทิตี (Entity Design Unit)

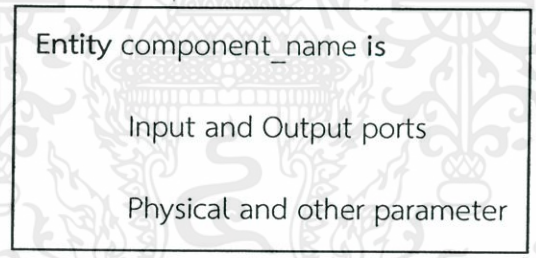
2) หน่วยการออกแบบสถาปัตยกรรม (Architecture Design Unit)

3) หน่วยการออกแบบแพ็กเก็ต (Package Design Unit)

4) หน่วยการออกแบบโครงแบบ (Configuration Design Unit)

2.4.2.1 หน่วยการออกแบบเอนทิตี

หน่วยการออกแบบนี้เป็นส่วนที่ใช้สำหรับติดต่อระหว่างภายนอกกับรูปแบบที่เขียนขึ้นโดยเป็นการกำหนดจุดเชื่อมต่อของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆ ของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น ซึ่งจะมีรูปแบบการเขียนดังรูปที่ 2.9




รูปที่ 2.9 โครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี [8]

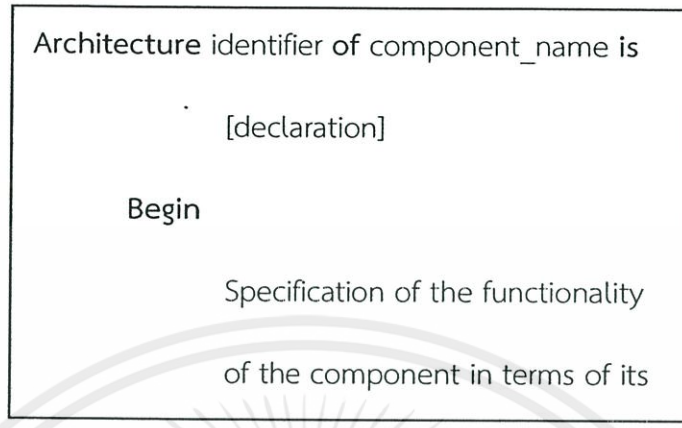
จากรูปที่ 2.9 ส่วนนี้จะขึ้นต้นด้วยคำว่า Entity และ is ระหว่างคำทั้งสองเป็นส่วนสำหรับชื่อรูปแบบที่ต้องการจะเขียน component_name หลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและออกของข้อมูล (Input-Output) รวมทั้งพารามิเตอร์อื่น และที่สำคัญคือหน่วยการออกแบบเอนทิตีจะต้องปิดท้ายด้วยคำว่า End และเครื่องหมายอัฒภาค (;) เสมอ

2.4.2.2 หน่วยการออกแบบสถาปัตยกรรม

หน่วยการออกแบบสถาปัตยกรรมคือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบในมุมมองของการจำลองการทำงานพฤติกรรมที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออกตรงช่องทาง ตลอดจนพารามิเตอร์ที่กำหนดในหน่วยการออกแบบเอนทิตี

ในหน่วยนี้มีรูปแบบการเขียนดังรูปที่ 2.10 ส่วนของหน่วยการออกแบบสถาปัตยกรรมเริ่มต้นด้วยคำว่า Architecture และตามด้วยชื่อ identifier สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า Architecture นั้นใช้บรรยายหน่วยการออกแบบเอนทิตีใดใด of<entity design unit> is ส่วนที่อยู่ระหว่าง Architecture และ Begin เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (Architecture Declaration Area) ที่เป็นส่วนเผื่อเลือก (Option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ อาทิเช่น ประเภท (Type) ต่างๆ (ตัวอย่างเช่น Bit, Bit_vector), สัญญาณ (Signal), ค่าคงที่ (Constant), โปรแกรมย่อย (ได้แก่ Function และ Procedure) และอุปกรณ์ (Component) ส่วนที่ใช้บรรยายความสำพันธ์ระหว่างข้อมูลที่ไหลเข้าและไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง Port) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า Begin กับ End ของหน่วยการออกแบบสถาปัตยกรรม และนอกจากนั้นชุด คำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (Concurrent Statement) เท่านั้นคือทุก statement จะทำงานพร้อมกันลำดับก่อนหลังจะไม่มีผลต่อการทำงานของรูปแบบ หน่วยการออกแบบสถาปัตยกรรมจะต้องปิดท้ายคำสั่ง End และชื่อของสถาปัตยกรรมนั้น โดยทั่วไปการเขียนรูปแบบระบบเชิงเลขด้วยภาษา VHDL สามารถเขียนได้ในลักษณะต่างๆ ดังนี้

1) ลักษณะการไหลของข้อมูล (Dataflow Style)
2) ลักษณะพฤติกรรม (Behavioral Style)
3) ลักษณะโครงสร้าง (Structural Style)
4) ลักษณะผสม (Mixed Model Style)

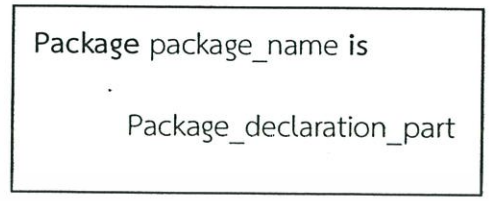



รูปที่ 2.10 โครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม [8]

2.4.2.3 หน่วยการออกแบบแพ็กเก็ต

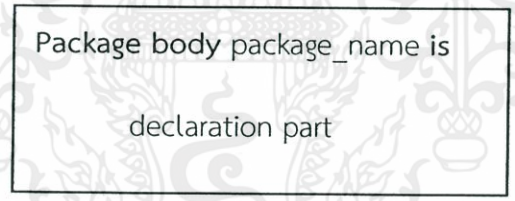
ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบเชิงเลขสามารถเก็บไว้ในส่วนของแพ็กเก็ตได้ และข้อมูลเหล่านี้สามารถเรียกไปใช้ได้ โดยหน่วยการออกแบบเอนทิตี หน่วยการออกแบบสถาปัตยกรรม หรือจากหน่วยการออกแบบแพ็กเก็ตอื่น โดยปกติแล้วแพ็กเก็ตจะแบ่งออกเป็น 2 ส่วน คือ การประกาศแพ็กเก็ต (Package Declaration) และการประกาศบอดี้แพ็กเก็ต (Package Body) เนื่องจากแพ็กเก็ตถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้นการที่จะนำแพ็กเก็ตไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อนซึ่งในภาษา VHDL สามารถกระทำได้ด้วยชุดคำสั่ง USE

Package Declaration ส่วนที่มีความสำคัญที่สุดของแพ็กเก็ต (ถ้ามองในแง่การนำไปใช้จากภายนอก) ได้แก่ส่วนการประกาศแพ็กเก็ต เพราะจะเป็นส่วนที่กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็กเก็ตสำหรับนำไปใช้ภายนอกตัวของแพ็กเก็ตเอง สิ่งใดใดที่ถูกประกาศไว้ในส่วนของบอดี้แพ็กเก็ต แต่ไม่ได้ถูกประกาศไว้ในส่วนการประกาศแพ็กเก็ตจะไม่สามารถถูกนำค่าและพฤติกรรมไปใช้ส่วนนอกได้ซึ่งสามารถเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศเอนทิตี คือจุดเชื่อมต่อหรือพอร์ทที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเก็ตสามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้ดังแสดงในรูปที่ 2.11 และยังสามารถถูกนำไปใช้จากรูปแบบภายนอกได้ เช่น ใช้สำหรับประกาศชนิด (Type) หรือสัญญาณ เช่นเดียวกันกับส่วนบอดี้แพ็กเก็ตที่ไม่จำเป็นต้อง มีส่วนของการประกาศแพ็กเก็ต แต่แพ็กเก็ตนั้นจะไม่สามารถถูกนำไปใช้จากรูปแบบอื่นได้




รูปที่ 2.11 โครงสร้างโดยทั่วไปของส่วนการประกาศแพ็กเก็ต [8]

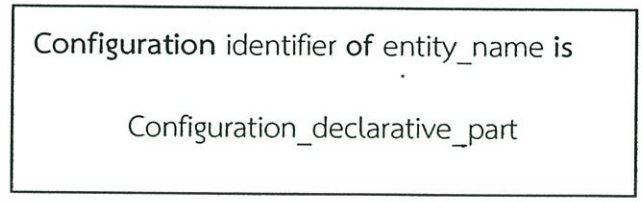
Package Body ดังรูปที่ 2.12 เป็นโครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ (Sequence) ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (Subprogram) ทั้งหลาย ที่ชื่อของโปรแกรมย่อยนั้นถูกประกาศไปในส่วนของการประกาศแพ็กเก็ตแล้วจะถูกเก็บไว้ในส่วนบอดี้แพ็กเก็ต ทั้งนี้รวมทั้งการกำหนดค่าคงที่ อันได้แก่ค่าคงที่ที่ถูกประกาศชื่อก่อนในส่วนของการประกาศแพ็กเก็ต แต่ถูกกำหนดค่าในส่วนของบอดี้แพ็กเก็ต ฉะนั้นส่วนบอดี้แพ็กเก็ตจึงไม่จำเป็นต้องมีถ้าในส่วนของการประกาศแพ็กเก็ตไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อยหรือค่าคงที่




รูปที่ 2.12 โครงสร้างโดยทั่วไปของบอดี้แพ็กเก็ต [8]

2.4.2.4 หน่วยการออกแบบโครงแบบ

ดังที่ทราบกันแล้วว่ารูปแบบหนึ่งของระบบเชิงเลขไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนิตีได้เพียงหนึ่งเดียวเท่านั้น แต่หน่วยการออกแบบเอนทิตีหนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมีหน่วยการออกแบบโครงแบบมาเพื่อกำหนดการใช้โครงแบบ (Configuration) ประกอบเอนทิตีกับหน่วยการออกแบบสถาปัตยกรรมหน่วยไหนเข้าด้วยกัน มีรูปแบบการเขียนดังรูปที่ 2.13

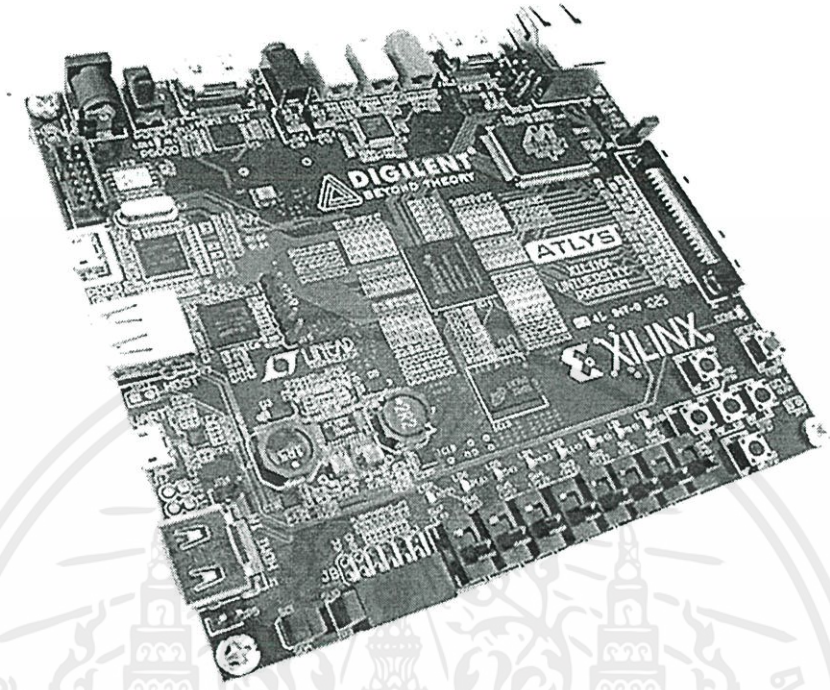



รูปที่ 2.13 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ [8]

### 2.4.3 การออกแบบวงจรเชิงเลขด้วยอุปกรณ์ FPGA

อุปกรณ์ FPGA (Field-Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามที่ออกแบบไว้ ในการทำ FPGA ซึ่งเป็นวิธีการออกแบบ IC (Integrated Circuit) แบบ Semicustom อีกวิธีหนึ่ง เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuit) แล้วนั้นก็มีทั้งข้อดีและข้อเสียคือการทำ FPGA จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในอุปกรณ์ FPGA จะมีจำนวนเกท (Gate) ให้ใช้จำนวนจำกัด และการทำ FPGA ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิต ในปริมาณต่ำส่วนข้อดีของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (Code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (Download) นั้นน้อยกว่าการทำ ASIC มาก และการตรวจสอบหรือแก้ไขการออกแบบก็ทำได้สะดวก

การทำ FPGA ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตอุปกรณ์ FPGA ได้เพิ่มความสามารถของอุปกรณ์ FPGA โดยเพิ่มจำนวนองค์ประกอบภายในหรือปรับปรุงโครงสร้างสถาปัตยกรรมภายใน และยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ PPR (Partitioning, Placement and Routing) สำหรับอุปกรณ์นั้น ในการใช้งานอุปกรณ์ FPGA สามารถไปประยุกต์ใช้งานได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP: Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์เป็นต้น ลักษณะของตัว FPGA แสดงได้ดังรูปที่ 2.14




รูปที่ 2.14 ลักษณะของตัว FPGA [11]

ในการออกแบบวงจรเชิงเลขด้วยอุปกรณ์มีขั้นตอนต่อไปนี้

2.4.3.1 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์

ในการออกแบบวงจรเชิงเลขนั้นทำได้โดยการวาดวงจรหรือใช้ภาษาอธิบายฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASIC ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะแตกต่างกัน โดยที่การทำวิธีนี้จะต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นได้ว่าการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ทำได้สะดวกกว่า เพราะการทำด้วยวิธีนี้ไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ (Technology Independence) และที่สำคัญการออกแบบด้วยวิธีนี้สามารถที่จะแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่าเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

ในการเขียนโค้ดสิ่งที่ต้องคำนึงถึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และให้คุณสมบัติของวงจรตามที่กำหนดเพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับ

วงจรที่ได้เนื่องจากในการสังเคราะห์วงจรนั้นซอฟต์แวร์สังเคราะห์วงจร (Synthesis Tools) จะทำการสังเคราะห์ตามโค้ดที่เขียน ถ้าอธิบายการทำงานของวงจรเดียวกันแต่เขียนโค้ดในลักษณะที่ต่างกัน เมื่อสังเคราะห์แล้วจะได้วงจรที่ต่างกันและจากวงจรที่ต่างกัน เมื่อนำไปทำต้นแบบด้วย FPGA หรือการทำ ASIC แล้วจะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านของขนาดหรือความเร็ว (Area and Time) ส่วนการเขียนโค้ดลักษณะใดเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ในการออกแบบ

2.4.3.2 การจำลองการทำงานของวงจร (Simulation)

ขั้นตอนนี้เป็นขั้นตอนที่สำคัญเพราะเป็นขั้นตอนที่ใช้ตรวจสอบฟังก์ชันการทำงานของวงจรว่าถูกต้องหรือไม่มีข้อผิดพลาดตรงไหน เพื่อที่จะได้ทำการแก้ไข้ให้ถูกต้อง ในขั้นตอนนี้จะใช้ซอฟต์แวร์สำหรับทำการจำลองการทำงานของวงจร เช่น V-System และ ModelSim ของบริษัท Model Technology

2.4.3.3 การสังเคราะห์วงจร

ในขั้นตอนนี้จะใช้ซอฟต์แวร์สังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์โค้ดเพื่อให้ได้เป็นวงจรขึ้นมา แต่ต้องตรวจสอบด้วยว่าซอฟต์แวร์ สนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการใช้หรือไม่ โดย FPGA ที่นิยมใช้งานเช่นของบริษัท Xilinx และบริษัท Altera ซอฟต์แวร์สังเคราะห์วงจรจะแปลงโค้ดและทำการออปติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้นอกจากนี้ยังสามารถกำหนดข้อบังคับสำหรับวงจรได้เช่น ข้อบังคับเรื่องของเวลา (Time Constraints) หรือข้อบังคับในเรื่องของพื้นที่ ซึ่งข้อบังคับเหล่านี้จะถูกนำไป ใช้ในขั้นตอนออปติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการออปติไมซ์คือการเทียบ (Mapping) วงจรให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์ FPGA ในกรณีของ Xilinx จะเทียบโดยใช้วิธี LUT (Look-Up Table) เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอฟต์แวร์สังเคราะห์วงจรก็จะมีการรายงานผลว่าวงจรที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วง (Delay) เท่าไร ใช้ทรัพยากรใน FPGA อะไรบ้าง เป็นต้น

2.4.3.4 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากกระบวนการสังเคราะห์ให้เป็นหน่วยย่อย สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (Routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำ โดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจร เช่น เกท (Gate), ฟลิปฟลอป (Flipflop) ลงในทรัพยากรต่างๆ ที่มีอยู่ในอุปกรณ์ FPGA (CLBs, IOBs, BUFT และ Edge Decoder) หลังจากทำขั้นตอนนี้เสร็จแล้วสามารถที่ทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนซอฟต์แวร์ที่ใช้ในขั้นตอนนี้ขึ้นอยู่กับตัว FPGA ที่ใช้งานเช่น FPGA ของบริษัท Xilinx จะใช้ Xilinx Foundation Series 2.1i ซึ่งซอฟต์แวร์ตัวนี้จะรวมเอาซอฟต์แวร์ย่อยอื่นอีก เพื่อให้การทำ PPR (Partitioning, Placement and Routing) เป็นไปอย่างต่อเนื่อง ส่วน FPGA ของบริษัท Altera จะใช้ Altera MAX+II

2.4.3.5 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควรจะอยู่ในตำแหน่งใดในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่น วงจรส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (Route) ได้ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือตัว Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

2.4.3.6 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA เช่น ระหว่าง CLBs หรือระหว่าง CLBs กับ IOBs ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมดหรือเกิดความ หน่วงเกินค่าที่กำหนดในข้อบังคับโดยสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟต์แวร์เช่นกัน หรือจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (Manual Layout) ก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่าโดยให้ทำการค้นหาเส้นทางหลายครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการ

กำหนดข้อบังคับทางเวลา (Time Constraints) จะช่วยให้ผลที่ได้จากการทำการเชื่อมต่อสัญญาณดีขึ้นได้

2.4.3.7 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่วงจรผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement and Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (Download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้ให้เป็นข้อมูลวงจร (Configuration Data) ซึ่งอยู่ในรูปบิตสตรีม (Bit-Stream) ก่อน แล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามวงจรที่ออกแบบไว้

จากที่อธิบายมาทั้งหมดจะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้น ทำได้สะดวกกว่าการทำ ASIC มากเพราะใช้เวลาน้อยกว่ามาก ส่วนสำคัญที่ใช้ในการทำ FPGA คือ ซอฟต์แวร์ที่ใช้ตั้งแต่การเขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงในอุปกรณ์ FPGA ซึ่งซอฟต์แวร์ที่ใช้ต้องเป็นซอฟต์แวร์ที่ใช้งานต่อเนื่องกัน

## 2.5 Field-Programmable Gate Array (FPGA) [10]

FPGA เป็นอุปกรณ์สารกึ่งตัวนำชนิดโปรแกรมได้ที่มีโครงข่ายการเชื่อมต่อภายในแบบเมตริกซ์ โครงสร้างภายในของ FPGA นั้นสามารถโปรแกรมให้มีหน้าที่การทำงานเหมือนลอจิกเกตพื้นฐาน เช่น AND, OR, XOR, NOT หรือรวมกันหลายชนิด (Combinational Logic Gate) เพื่อให้ทำหน้าที่ที่มีความซับซ้อนเพิ่มขึ้น เช่น Decoders หรือฟังก์ชันทางคณิตศาสตร์ใน FPGA ทั่วไป นอกจากจะประกอบด้วยส่วนของวงจรลอจิกแบบโปรแกรมได้แล้ว ยังมีบล็อกของหน่วยความจำ ซึ่งอาจจะสร้างจากฟลิบฟลอปอย่างง่าย หรือใช้พื้นที่ของสารกึ่งตัวนำสร้างเป็นหน่วยความจำจริงอยู่ภายในก็ได้

ในการออกแบบวงจรเชิงเลขอิเล็กทรอนิกส์ที่มี FPGA อยู่บนแผงวงจรด้วยนั้น จะช่วยให้ผู้ออกแบบสามารถลดขนาดของแผงวงจร รวมทั้งสามารถออกแบบได้รวดเร็ว ไม่ต้องทดสอบรายละเอียดภายในให้เสร็จสมบูรณ์ก่อน ก็สามารถออกแบบแผงวงจรได้ เมื่อได้รับแผงวงจรและประกอบอุปกรณ์ต่างๆ เสร็จแล้ว จึงค่อยกำหนดหน้าที่การทำงานของ FPGA ได้ในภายหลัง ต่างจากการออกแบบด้วยลอจิกเกตขนาดเล็กที่ต้องออกแบบทางเดินของลายทองแดงให้เสร็จสมบูรณ์ก่อน และไม่สามารถแก้ไขได้ในภายหลัง นอกจากนี้การใช้งาน FPGA สามารถโปรแกรมการทำงาน

ได้ในทุกขณะแม้แต่ขณะที่จัดทำชิ้นงานแล้ว ก็ยังสามารถแก้ไขวงจรได้โดยง่าย จึงเป็นที่มาของคำว่า "Field-Programmable" ซึ่งหมายถึงโปรแกรมได้ในภาคสนามหรือที่หน้างานนั่นเอง อย่างไรก็ตาม รูปแบบของวงจรที่สังเคราะห์ (Configuration) ของ FPGA จะหายไปหลังจากปิดไฟเลี้ยง ดังนั้น จะต้องมีหน่วยความจำภายนอก (Flash) มาเก็บรูปแบบของวงจรที่สังเคราะห์บน FPGA ไว้ ซึ่ง FPGA จะมีกระบวนการอ่านรูปแบบของวงจรที่สังเคราะห์นั้นโดยอัตโนมัติหลังจากได้รับไฟเลี้ยง

### 2.5.1 ประเภทของ FPGA

ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรมที่แตกต่างไป นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบคือการโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรมโดยการใช้หน่วยความจำ

#### 2.5.1.1 การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

1) Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียว ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อมต่อจะขาดจากกัน

2) Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกันแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรมแล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

#### 2.5.1.2 การโปรแกรมโดยใช้หน่วยความจำ

1) EEPROM Based FPGA

FPGA ที่ใช้การโปรแกรมแบบนี้มักเรียกว่า CPLD ซึ่งเทคโนโลยีที่ใช้จะเหมือนกับ EEPROM ทำให้มีความจุของเกตต่ำ โดยทั่วไปจะน้อยกว่า 20,000 เกต แต่ข้อดีของ EEPROM Based FPGA คือสามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง และในโปรแกรมจะใช้ทรานซิสเตอร์ 1 ตัวต่อ 1 บิต ซึ่งการโปรแกรมสามารถทำได้ประมาณ 10,000 ครั้ง

2) SRAM Based FPGA

FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM) ทำให้สามารถโปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกตในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 – 1,000,000 เกต) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการโปรแกรมน้อย (ระดับ nanosecond) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และเหมาะสำหรับการออกแบบวงจรที่มีความซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมในภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อใช้เก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิฟในขณะที่เริ่มต้นใช้งาน

### 2.5.2 ข้อได้เปรียบของการใช้ FPGA

1) ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพเพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึงภาษาแอสแซมบลี (Assembly) ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2) การออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจรหรือเอชดีแอล เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็วและไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มันจากนั้นตัวซอฟต์แวร์จะสังเคราะห์ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตรฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3) การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของคอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพ ขณะที่อยู่ในระบบได้โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอกและที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสียค่าใช้จ่ายเพิ่มแต่อย่างใด

### 2.5.3 เครื่องมือสำหรับการออกแบบ FPGA

จะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่า ASIC มากเพราะใช้เวลาน้อยกว่ามากด้วย ส่วนสำคัญที่ใช้ในการทำ FPGA คือซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาว์นโหลดลงในอุปกรณ์ FPGA ซึ่งซอฟต์แวร์ที่ใช้ต้องเป็นซอฟต์แวร์ที่ทำงานต่อเนื่องกัน

ได้ สำหรับซอฟต์แวร์ที่ใช้ทำการจำลองการทำงานของวงจรนั้นต้องสามารถใช้งานต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบเพราะโมเดลที่ได้จากการทำขั้นตอนต่างๆ ด้วยซอฟต์แวร์ต้องเอามาจำลองการทำงานได้ และในการจำลองการทำงานของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดทั้งระบบเพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ที่ใช้บนพีซี (PC) มากขึ้นซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

# บทที่ 3

# การออกแบบและการจัดทำปริญญานิพนธ์

ในบทนี้จะนำเสนอการออกแบบและการสร้างโครงสร้างทางฮาร์ดแวร์ของฟิลเตอร์แบงก์ โดยจะใช้ค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 เป็นค่าสัมประสิทธิ์ให้กับวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด (Finite Impulse Response Digital Filter; FIR Filter) ภายในฟิลเตอร์แบงก์ และใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู ออกแบบโดยบรรยายพฤติกรรมด้วยภาษา VHDL แล้วทำการสังเคราะห์ลงบน FPGA ซึ่งในภาคการเรียนที่ 1 จะออกแบบเป็นฟิลเตอร์แบงก์แบบ 1 มิติ (1D Filter Banks) และในภาคการเรียนที่ 2 จะออกแบบเป็นฟิลเตอร์แบงก์แบบ 2 มิติ (2D Filter Banks) รวมทั้งส่วนควบคุมการแสดงผลบนจอแสดงผลวีจีเอ

## 3.1 การออกแบบวงจรกรองเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด

### 3.1.1 การนำค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 มาใช้กับวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด

ในหัวข้อนี้การออกแบบวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดจะใช้ค่าสัมประสิทธิ์จากตัวกรอง CDF 9/7 [5] ซึ่งมีสัมประสิทธิ์ของตัวกรอง $H_0(z)$ และ $H_1(z)$ ดังตารางที่ 3.1 โดย $h_0[n]$ คือสัมประสิทธิ์ของตัวกรองความถี่ต่ำผ่าน $H_0(z)$ และ $h_1[n]$ คือสัมประสิทธิ์ของตัวกรองความถี่สูงผ่าน $H_1(z)$ ซึ่งจะพบว่าค่าสัมประสิทธิ์จะมีคุณสมบัติที่มีความเป็นสมมาตรกัน

ตารางที่ 3.1 ค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 ของ $H_0(z)$ และ $H_1(z)$ [5]

| $n$ | $h_0[n]$ | $h_1[n]$ |
|---|---|---|
| 0 | 0.026748757410810 | 0.0912717631142495 |
| 1 | -0.016864118442875 | -0.057543526228500 |
| 2 | -0.078223266528988 | -0.591271763114247 |
| 3 | 0.266864118442872 | 1.115087052456994 |

ตารางที่ 3.1 ค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 ของ $H_0(z)$ และ $H_1(z)$ [5] (ต่อ)

| $n$ | $h_0[n]$ | $h_1[n]$ |
|---|---|---|
| 4 | 0.602949018236358 | -0.591271763114247 |
| 5 | 0.266864118442872 | -0.057543526228500 |
| 6 | -0.078223266528988 | 0.0912717631142495 |
| 7 | -0.016864118442875 | |
| 8 | 0.026748757410810 | |

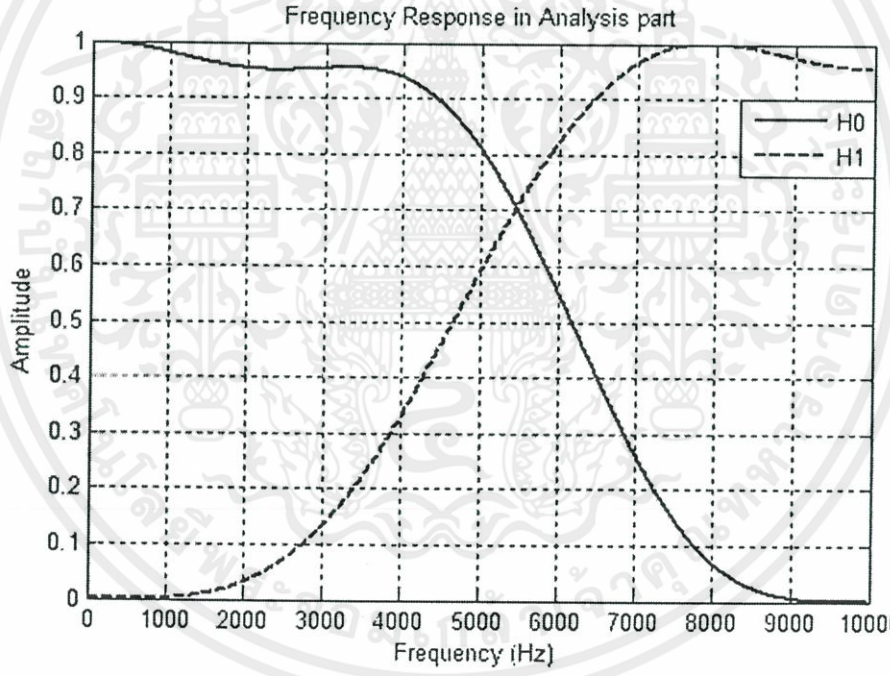
สำหรับสัมประสิทธิ์ของตัวกรอง $F_0(z)$ และ $F_1(z)$ สามารถหาได้จากความสัมพันธ์ของฟิลเตอร์แต่ละตัวตามเงื่อนไขของการกู้กลับอย่างสมบูรณ์ดังสมการที่ 2.11 และสมการที่ 2.12 ทำให้ได้สัมประสิทธิ์ของตัวกรองดังตารางที่ 3.2

ตารางที่ 3.2 ค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 ของ $F_0(z)$ และ $F_1(z)$ [5]

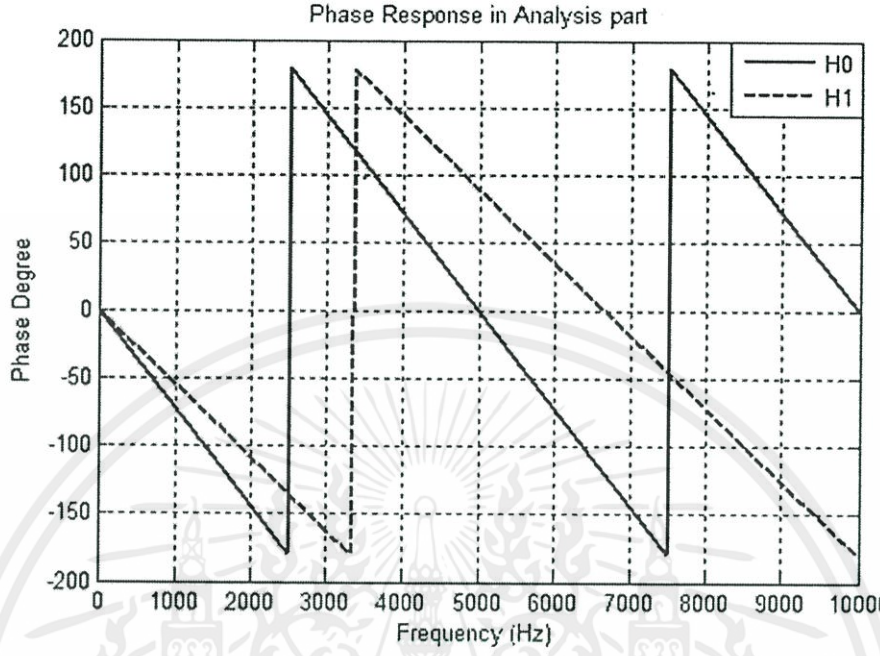
| $n$ | $f_0[n]$ | $f_1[n]$ |
|---|---|---|
| 0 | -0.0912717631142495 | 0.026748757410810 |
| 1 | -0.057543526228500 | 0.016864118442875 |
| 2 | 0.591271763114247 | -0.078223266528988 |
| 3 | 1.115087052456994 | -0.266864118442872 |
| 4 | 0.591271763114247 | 0.602949018236358 |
| 5 | -0.057543526228500 | -0.266864118442872 |
| 6 | -0.0912717631142495 | -0.078223266528988 |
| 7 | | 0.016864118442875 |
| 8 | | 0.026748757410810 |

จากตารางที่ 3.2 $f_0[n]$ คือสัมประสิทธิ์ของตัวกรองความถี่ต่ำผ่าน $F_0(z)$ และ $f_1[n]$ คือสัมประสิทธิ์ของตัวกรองความถี่สูงผ่าน $F_1(z)$ ซึ่งจะพบว่าค่าสัมประสิทธิ์จะมีคุณสมบัติสมมาตรเช่นกัน สำหรับการตรวจสอบความถูกต้องตามเงื่อนไขการกู้กลับอย่างสมบูรณ์ทั้งการกำจัดการซ้อนทับ และการปราศจากการลดทอนจะกล่าวในหัวข้อถัดไป

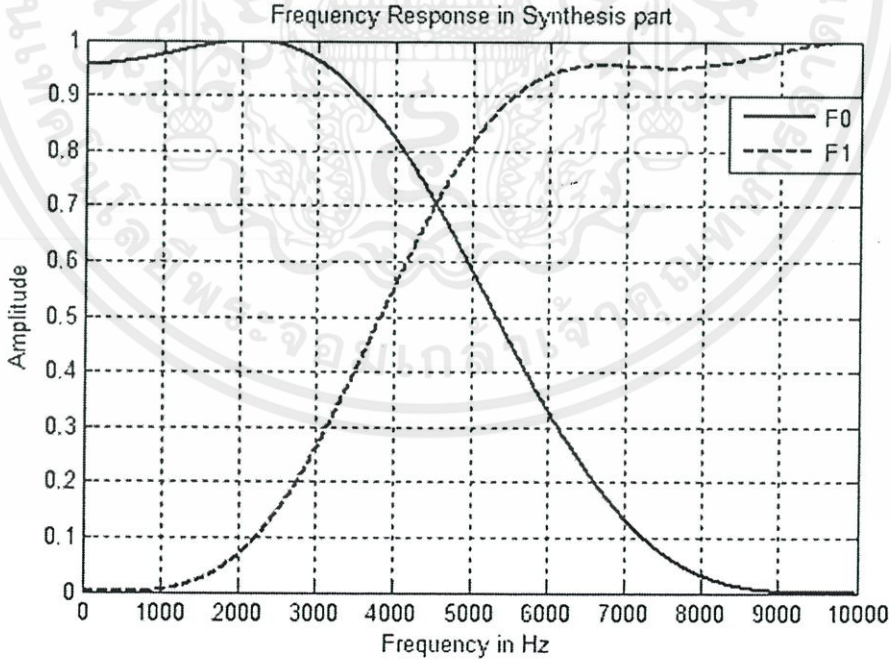
เมื่อนำค่าสัมประสิทธิ์ของตัวกรองทั้ง 4 ตัวมาเขียนกราฟผลตอบสนองเชิงความถี่ โดยกำหนดให้มีค่าความถี่สุ่มตัวอย่างอยู่ที่ 20 kHz ซึ่งจะแสดงผลตอบสนองเชิงความถี่ได้ดังรูปที่ 3.1 และรูปที่ 3.2 และผลตอบสนองเชิงเฟสได้ดังรูปที่ 3.3 และรูปที่ 3.4 โดยมีเงื่อนไขการออกแบบว่าตัวกรองต้องมีลักษณะเป็นตัวกรองแบบ Half-band นั้นจะทำการตรวจสอบความถูกต้องของการออกแบบตัวกรองในหัวข้อถัดไป




รูปที่ 3.1 ผลตอบสนองเชิงความถี่ของตัวกรองด้านวิเคราะห์ $H_0(z)$ และ $H_1(z)$

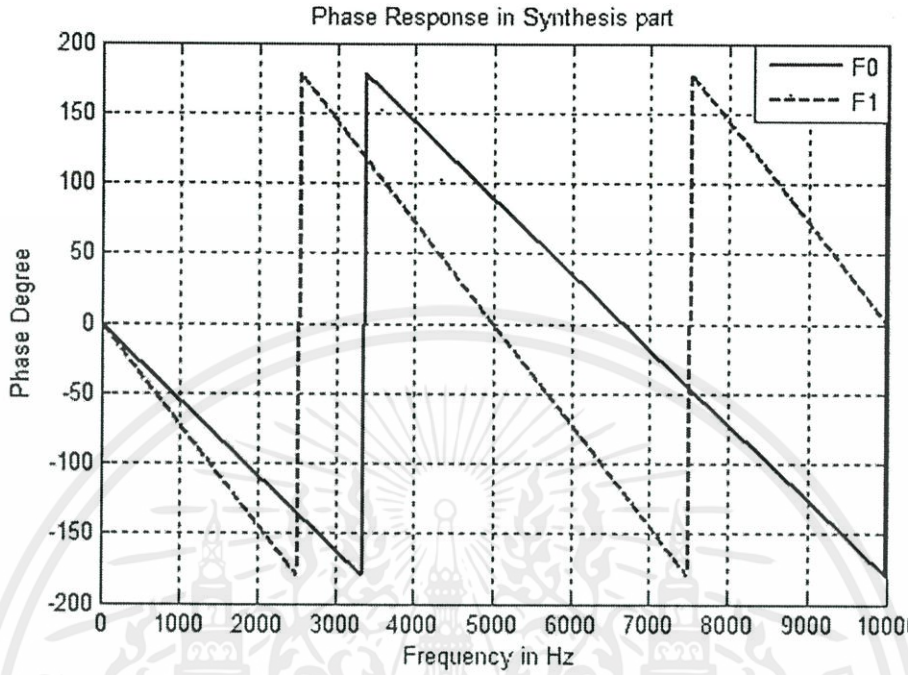



รูปที่ 3.2 ผลตอบสนองเชิงเฟสของตัวกรองด้านวิเคราะห์ $H_0(z)$ และ $H_1(z)$






รูปที่ 3.3 ผลตอบสนองเชิงความถี่ของตัวกรองด้านสังเคราะห์ $F_0(z)$ และ $F_1(z)$




รูปที่ 3.4 ผลตอบสนองเชิงเฟสของตัวกรองด้านสังเคราะห์ $F_0(z)$ และ $F_1(z)$

เนื่องจากค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 ที่นำมาใช้เป็นค่าทศนิยม ในการคำนวณภายในฮาร์ดแวร์จะใช้การคำนวณในรูปเลขฐานสอง ดังนั้นจึงต้องมีการแปลงค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 ทั้ง 4 ตัวให้อยู่ในรูปเลขฐานสองแบบส่วนเติมเต็มสองก่อน กำหนดให้ค่าสัมประสิทธิ์ถูกแทนเป็นเลขฐานสองจำนวน 8 บิต โดยจะมีค่าสูงสุดเพียง $2^4$ หรือ 16 เพื่อให้ได้ค่าที่เมื่อนำไปคำนวณภายในฮาร์ดแวร์แล้วไม่เกิดการล้น แล้วหลังจากนั้นทำการแปลงเป็นเลขฐานสอง ได้ดังตารางที่ 3.3 ในส่วนนี้จะอาศัยการคำนวณผ่านทางโปรแกรมแมตแล็บซึ่งโค้ดจะแสดงในภาคผนวก ก

ตารางที่ 3.3 ค่าสัมประสิทธิ์ของตัวกรอง CDF 9/7 ในรูปเลขฐานสองแบบส่วนเติมเต็มสอง

| $n$ | $h_0[n]$ | $h_1[n]$ | $f_0[n]$ | $f_1[n]$ |
|---|---|---|---|---|
| 0 | 00000001 | 00000001 | 11111111 | 00000001 |
| 1 | 00000000 | 11111111 | 11111111 | 00000000 |
| 2 | 11111110 | 11111000 | 00001000 | 11111110 |
| 3 | 00000111 | 00001111 | 00001111 | 11111001 |
| 4 | 00001111 | 11111000 | 00001000 | 00001111 |
| 5 | 00000111 | 11111111 | 11111111 | 11111001 |
| 6 | 11111110 | 00000001 | 11111111 | 11111110 |
| 7 | 00000000 | | | 00000000 |
| 8 | 00000001 | | | 00000001 |

### 3.1.2 การตรวจสอบความถูกต้องตามเงื่อนไขการกู้กลับอย่างสมบูรณ์

ในการตรวจสอบความถูกต้องตามเงื่อนไขการกู้กลับอย่างสมบูรณ์จะพิสูจน์โดยอาศัยสมการเงื่อนไขการกำจัดการซ้อนทับสมการที่ 2.7 และสมการเงื่อนไขการปราศจากการลดทอนสมการที่ 2.10 ซึ่งค่าสัมประสิทธิ์ที่นำมาใช้ตามตารางที่ 3.1 และ 3.2 จะถูกเขียนอยู่ในรูปพหุนาม $z^{-n}$ โดยที่ค่าสัมประสิทธิ์ของตัวกรองจะเป็นค่าสัมประสิทธิ์ของตัวแปร หลังจากนั้นจึงแทนลงสมการทั้งสองข้างต้นเพื่อทำการตรวจสอบ

จากสมการเงื่อนไขการกำจัดการซ้อนทับสมการที่ 2.7 ทำการหาค่า $F_0(z)H_0(-z)$ และ $F_1(z)H_1(-z)$ ได้สมการผลลัพธ์ดังสมการที่ 3.1 และ 3.2 ตามลำดับ

$$\begin{aligned} F_0(z)H_0(-z) = {} & -0.002z^{-14} - 0.003z^{-13} + 0.022z^{-12} + 0.069z^{-11} \\ & -0.051z^{-10} - 0.247z^{-9} + 0.032z^{-8} + 0.363z^{-7} \\ & +0.032z^{-6} - 0.247z^{-5} - 0.051z^{-4} + 0.069z^{-3} \\ & +0.022z^{-2} - 0.003z^{-1} - 0.002 \end{aligned} \tag{3.1}$$

$$F_1(z)H_1(-z) = 0.002z^{-14} + 0.003z^{-13} - 0.022z^{-12} - 0.069z^{-11}$$
$$+0.051z^{-10} + 0.247z^{-9} - 0.032z^{-8} - 0.363z^{-7}$$
$$-0.032z^{-6} + 0.247z^{-5} + 0.051z^{-4} - 0.069z^{-3}$$
$$-0.022z^{-2} + 0.003z^{-1} + 0.002 \quad (3.2)$$

จากนั้นเมื่อนำผลลัพธ์ทั้งสองส่วนมารวมกันตามสมการเงื่อนไขการกำจัดการซ้อนทับจะได้สมการที่ 3.3

$$F_0(z)H_0(-z) + F_1(z)H_1(-z) = 0 \quad (3.3)$$

จากสมการที่ 3.3 จะพบว่าผลลัพธ์ทั้งสองส่วนที่นำมารวมกันทำให้เงื่อนไขการกำจัดการซ้อนทับเป็นจริง สำหรับเงื่อนไขการปราศจากการลดทอนนั้นจะทำการหาค่า $F_0(z)H_0(z)$ และ $F_1(z)H_1(z)$ ตามสมการที่ 2.10 ได้สมการผลลัพธ์ดังสมการที่ 3.4 และ 3.5 ตามลำดับ

$$F_0(z)H_0(z) = -0.002z^{-14} + 0.024z^{-12} - 0.120z^{-10} + 0.598z^{-8} + z^{-7}$$
$$+0.598z^{-6} - 0.120z^{-4} + 0.024z^{-2} - 0.002 \quad (3.4)$$

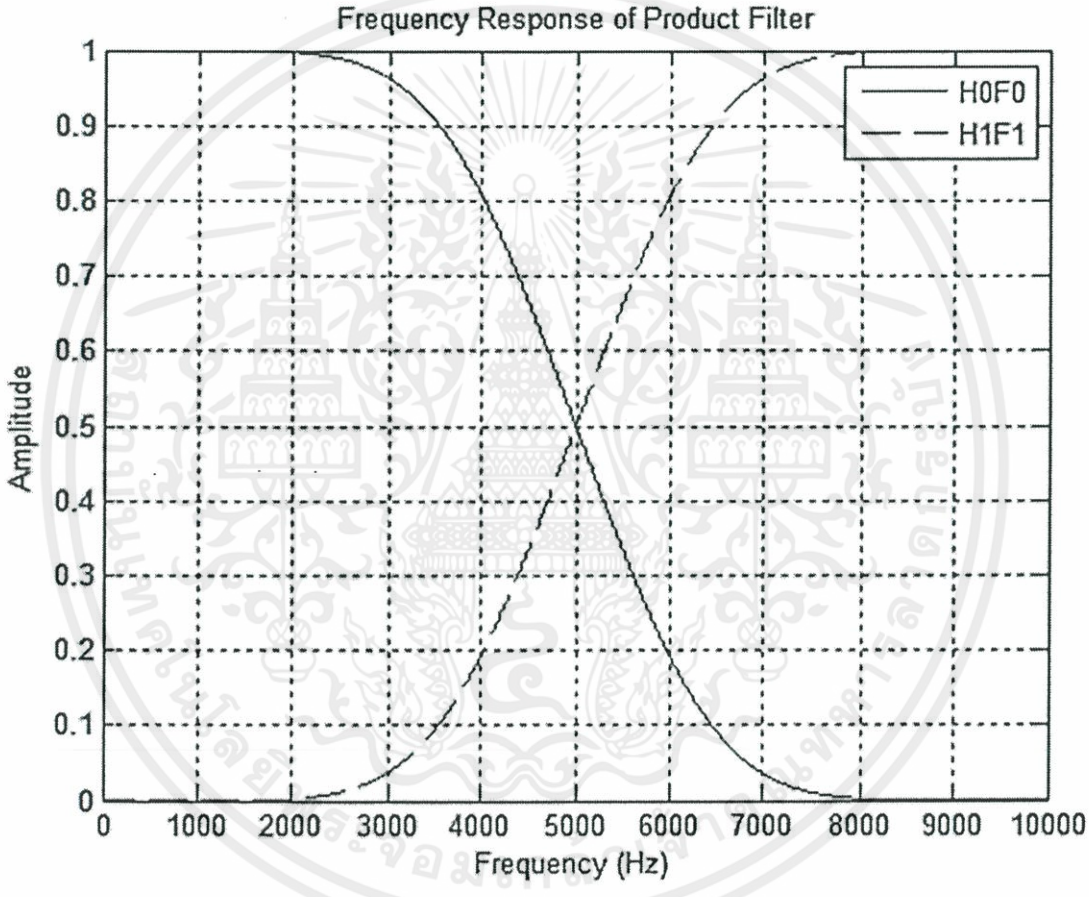
$$F_1(z)H_1(z) = 0.002z^{-14} - 0.024z^{-12} + 0.120z^{-10} - 0.598z^{-8} + z^{-7}$$
$$-0.598z^{-6} + 0.120z^{-4} - 0.024z^{-2} + 0.002 \quad (3.5)$$

จากนั้นเมื่อนำผลลัพธ์ตามสมการที่ 3.4 และ 3.5 มารวมกันตามสมการเงื่อนไขการปราศจากการลดทอนจะได้สมการที่ 3.6

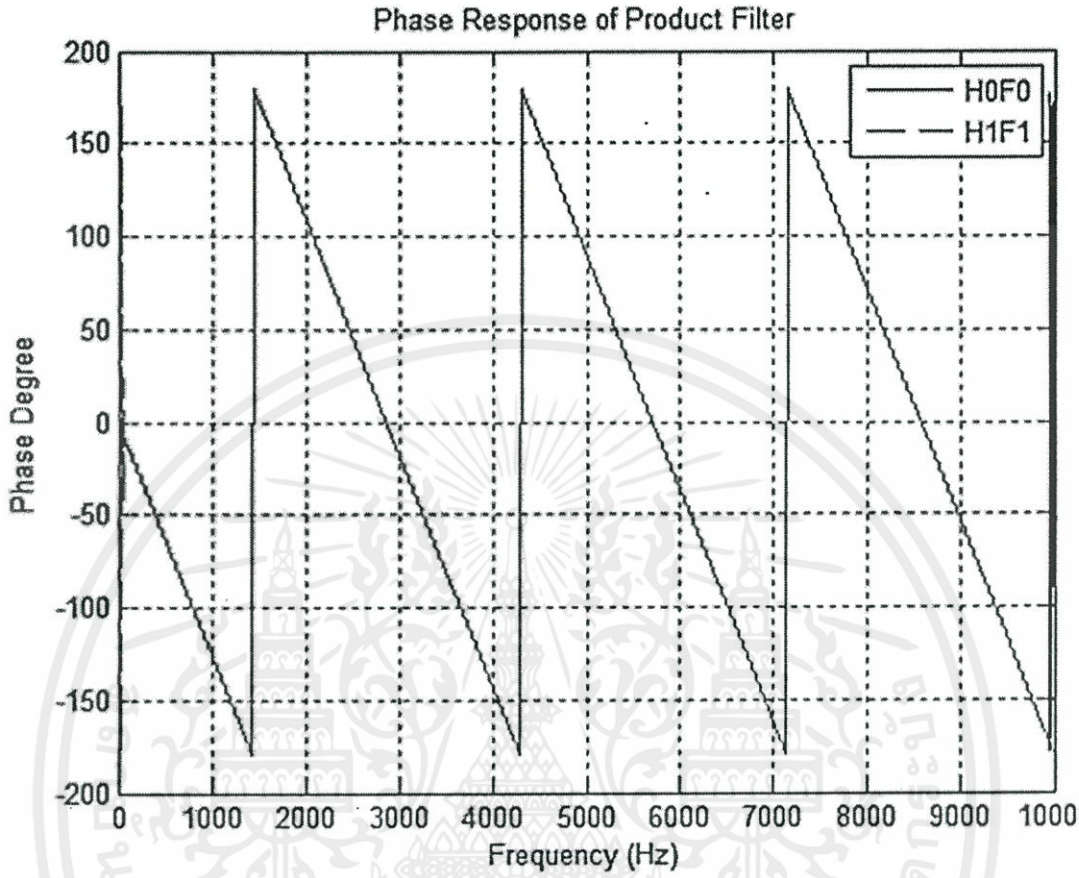
$$F_0(z)H_0(z) + F_1(z)H_1(z) = 2z^{-l}$$
$$2z^{-7} = 2z^{-l} \quad (3.6)$$

จากสมการที่ 3.6 จะพบว่าเราสามารถหาค่า $l$ ที่ทำให้เงื่อนไขการปราศจากการลดทอนเป็นจริงได้ คือค่า $l = 7$ และในเมื่อหาค่า $l$ ได้ทำให้ทราบอีกด้วยว่าค่าสัมประสิทธิ์ที่นำมาใช้จะทำให้เอาต์พุตที่ได้จากการประมวลผลเกิดการดีเลย์ไป 7 ตำแหน่ง

ดังนั้นจากการพิสูจน์เงื่อนไขทั้งสองพบว่าค่าสัมประสิทธิ์ CDF 9/7 ที่นำมาใช้นั้นทำให้เงื่อนไขการกำจัดการซ้อนทับ และเงื่อนไขการปราศจากการลดทอนเป็นจริง จึงสรุปได้ว่าค่าสัมประสิทธิ์ชุดนี้มีคุณสมบัติการกู้กลับอย่างสมบูรณ์ โดยสามารถตรวจสอบเพิ่มเติมได้ด้วยการนำค่า $F_0(z)H_0(z)$ และ $F_1(z)H_1(z)$ มาเขียนกราฟผลตอบสนองเชิงความถี่ และผลตอบสนองเชิงเฟสของฟิลเตอร์ผลลัพธ์ (Product Filter) ได้ดังรูปที่ 3.5 และ 3.6




รูปที่ 3.5 ผลตอบสนองเชิงความถี่ของฟิลเตอร์ผลลัพธ์



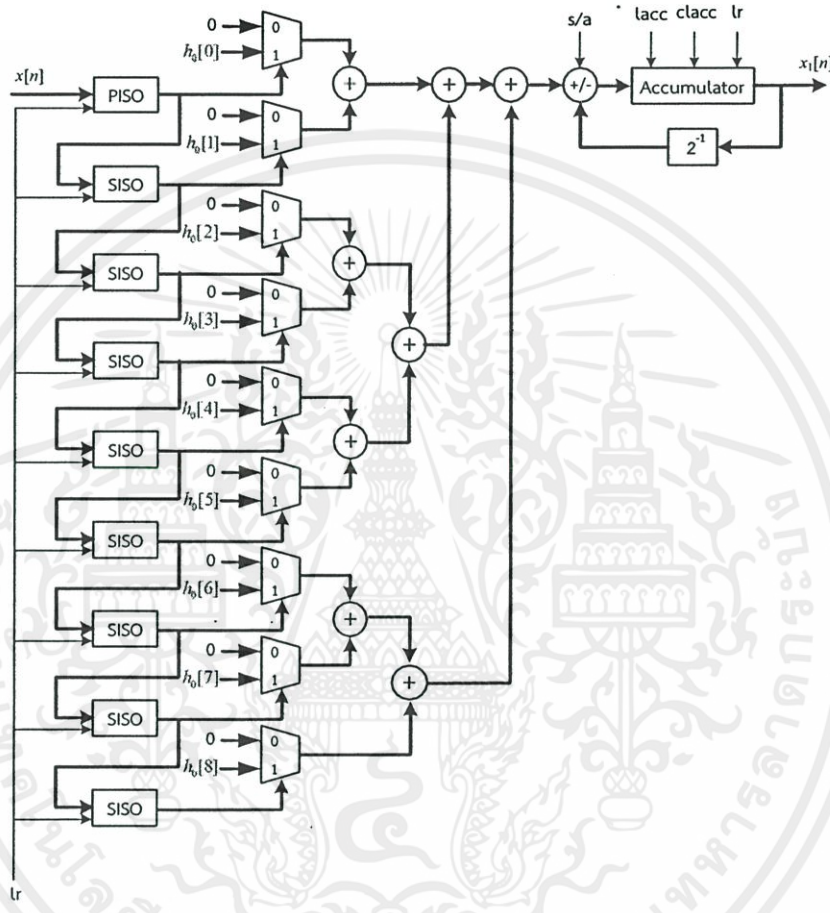

รูปที่ 3.6 ผลตอบสนองเชิงเฟสของฟิลเตอร์ผลลัพธ์

จากรูปที่ 3.5 ทำให้พบว่าค่าสัมประสิทธิ์ที่นำมาใช้เมื่อสร้างเป็นฟิลเตอร์แล้วจะมีลักษณะเป็น Half-band ซึ่งตรงตามเงื่อนไขการออกแบบ และนอกจากนี้จากรูปที่ 3.6 ทำให้ทราบว่าฟิลเตอร์ที่ออกแบบมีเฟสเป็นแบบเชิงเส้นด้วย

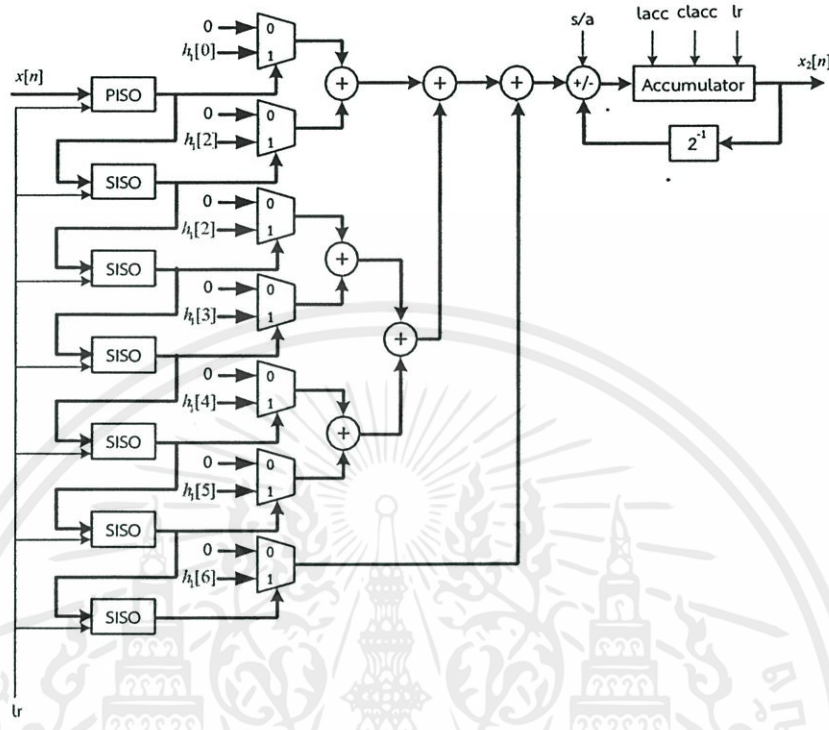
### 3.1.3 โครงสร้างที่นำมาใช้กับวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด

สำหรับการออกแบบวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดภายในฟิลเตอร์แบงก์ จะใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู (LUT-less DA Structure) [6] ซึ่งจะแบ่งโครงสร้างหลักออกเป็นสองแบบคือแบบ 9-Tap และ 7-Tap โดยแบ่งตาม

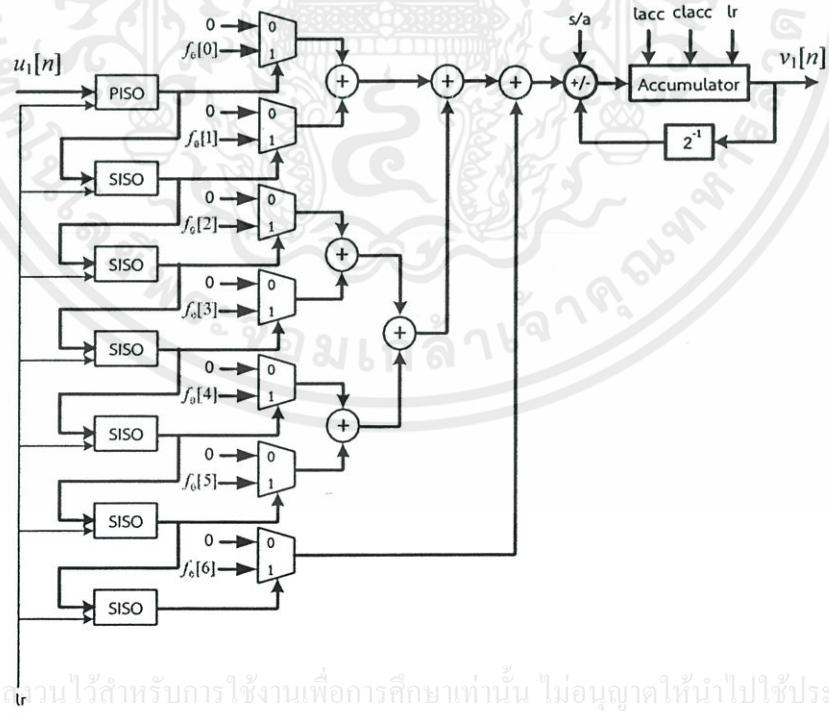
ขนาดของฟิลเตอร์จากการดูจำนวนค่าสัมประสิทธิ์ CDF 9/7 ทำให้ได้โครงสร้างของตัวกรอง $H_0(z)$, $H_1(z)$, $F_0(z)$ และ $F_1(z)$ ดังรูปที่ 3.7, 3.8, 3.9 และ 3.10 ตามลำดับ




รูปที่ 3.7 โครงสร้างของตัวกรอง $H_0(z)$

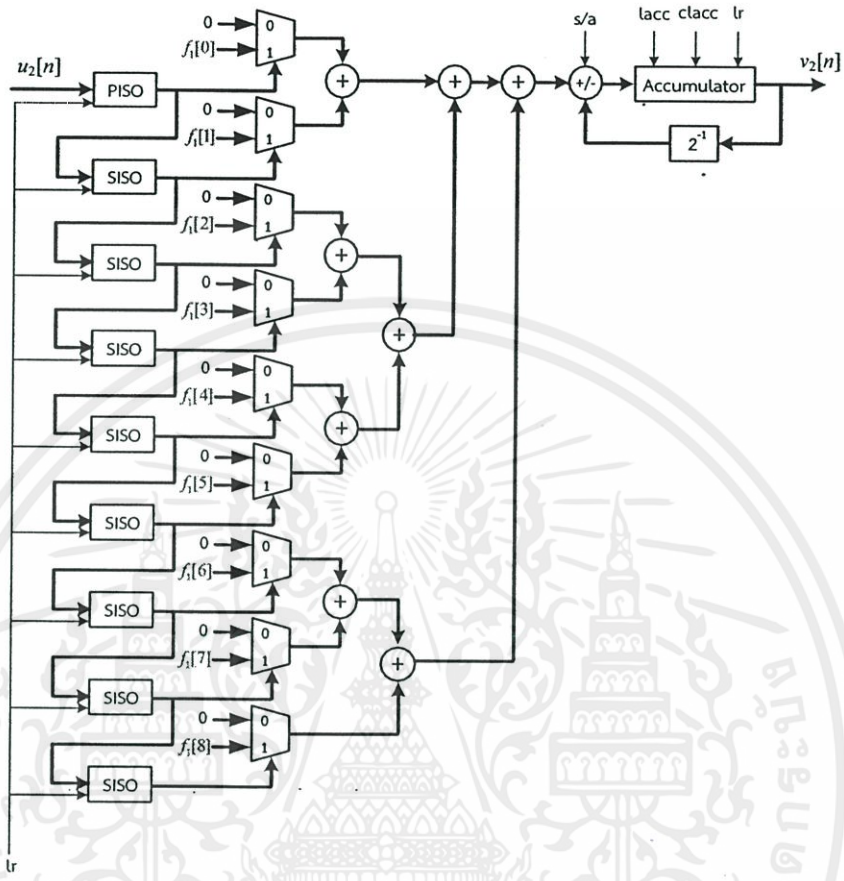



รูปที่ 3.8 โครงสร้างของตัวกรอง $H_1(z)$






รูปที่ 3.9 โครงสร้างของตัวกรอง $F_0(z)$




รูปที่ 3.10 โครงสร้างของตัวกรอง $F_1(z)$

โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูที่นำมาใช้นี้มีข้อได้เปรียบจากโครงสร้างเลขคณิตกระจายแบบอาศัยตารางเปิดดู (Conventional LUT-based DA Structure) ตรงที่ไม่ได้มีการเก็บค่าผลคูณย่อยระหว่างสัญญาณอินพุตกับค่าสัมประสิทธิ์ของตัวกรองที่คำนวณไว้ล่วงหน้าแล้ว เก็บเอาไว้ในหน่วยความจำ (ROM) ซึ่งเมื่อฟิลเตอร์มีอันดับที่สูงมากขึ้น จะเห็นได้ว่าโครงสร้างต้องใช้ขนาดหน่วยความจำในการเก็บค่าผลคูณย่อยเพิ่มขึ้นแบบเอกซ์โพเนนเชียลตามอันดับของตัวกรอง และต้องทำการคำนวณค่าผลคูณย่อยเพื่อจัดเก็บในหน่วยความจำใหม่ [9] ดังนั้นโครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูจึงถูกพิจารณานำมาใช้แทนโดยจะเปรียบเทียบลักษณะโครงสร้างทั้ง 2 แบบในภาคผนวก ง

ลักษณะโครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูจากรูปที่ 3.7, 3.8, 3.9 และ 3.10 จะทำการแยกสัญญาณอินพุตออกในรูปแบบบิตและนำมาเป็นตัวควบคุม มัลติเพล็กซ์

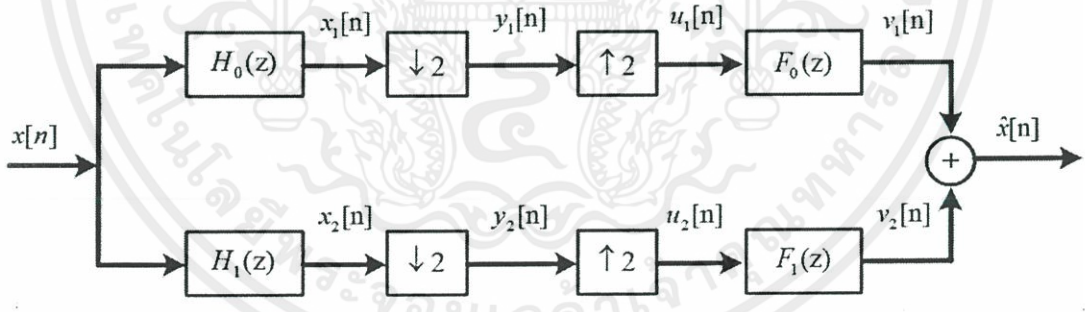
เซอร์ (Multiplexer) ที่จะเลือกค่าสัมประสิทธิ์ที่เก็บไว้ตามตารางที่ 3.3 ออกมาใช้ หลังจากนั้นจะนำผลจากมัลติเพล็กซ์เซอร์ทุกตัวมาบวกรวมกัน โดยจะพบว่าทำให้ลดพื้นที่ของหน่วยความจำลง และจำนวนตัวบวกกับมัลติเพล็กซ์เซอร์จะเพิ่มเป็นแบบเชิงเส้นเมื่ออันดับของตัวกรองเพิ่มขึ้น [10]

## 3.2 การออกแบบโครงสร้างของฟิลเตอร์แบงก์ด้วยภาษา VHDL

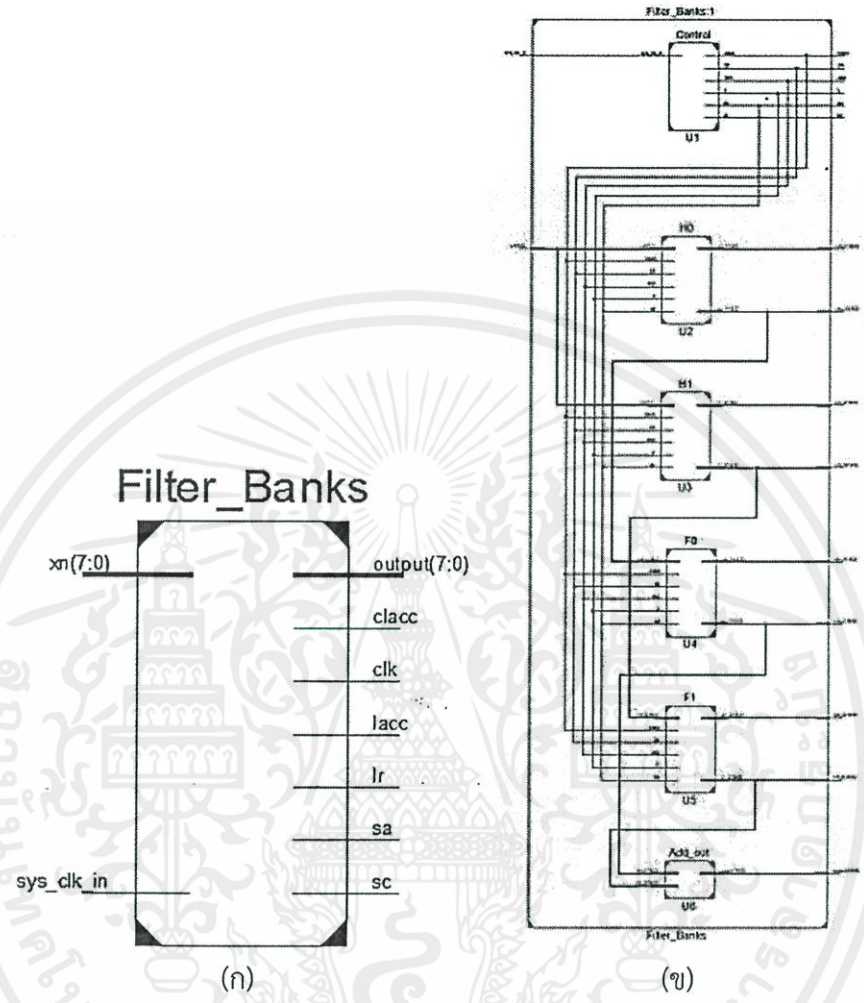
โครงสร้างของฟิลเตอร์แบงก์ที่ออกแบบจะถูกบรรยายพฤติกรรมการทำงานด้วยภาษา VHDL โดยใช้โปรแกรม Xilinx ISE Design Suite 14.6 ในการสังเคราะห์ และจำลองการทำงานวงจรย่อยต่างๆ ผ่านโปรแกรม ISim 14.6

### 3.2.1 การออกแบบฟิลเตอร์แบงก์แบบ 1 มิติ

ในการออกแบบฟิลเตอร์แบงก์แบบ 1 มิติ จะออกแบบด้วยการเขียนโปรแกรมเพื่อสร้างแต่ละวงจรย่อยที่ต้องทำงานร่วมกัน โดยมีโครงสร้างของตัวกรองเป็นแบบ Two-Channel Filter Banks ตามรูปที่ 3.11 หลังจากนั้นทำการสังเคราะห์วงจรจะทำให้ได้วงจรโดยแสดงเป็น Schematic Diagram ดังรูปที่ 3.12




รูปที่ 3.11 Two-Channel Filter Banks [2]



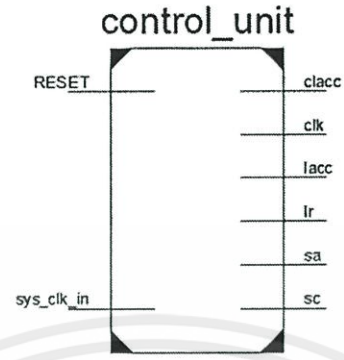

(ก) (ข)

รูปที่ 3.12 (ก) สัญลักษณ์ของฟิลเตอร์แบงก์แบบ 1 มิติที่ออกแบบด้วยภาษา VHDL
(ข) Schematic Diagram ของฟิลเตอร์แบงก์แบบ 1 มิติ

ดังนั้นในการออกแบบและสร้างฟิลเตอร์แบงก์แบบ 1 มิติ จะมีแต่ละวงจรย่อยได้ดังนี้

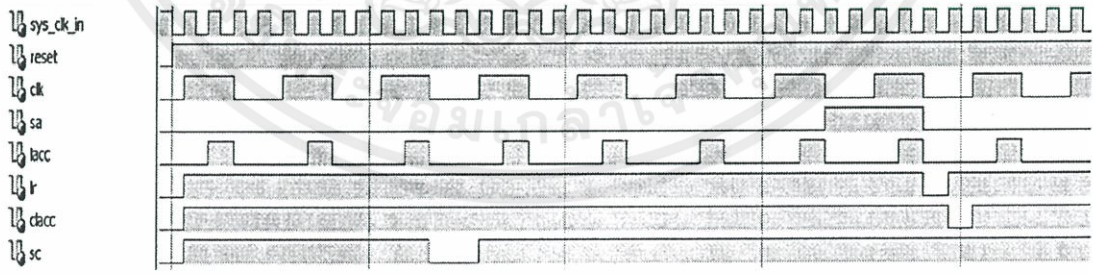
3.2.1.1 วงจรควบคุม (Control Unit)

วงจรควบคุมใช้ในการสร้างสัญญาณควบคุมอุปกรณ์ภายในต่างๆ ทั้งหมดในระบบเพื่อให้ทำงานสอดคล้องกัน โดยวงจรควบคุมจะสร้างสัญญาณควบคุมตามการทำงานที่กำหนดไว้ สามารถแทนวงจรควบคุมด้วยสัญลักษณ์ได้ดังรูปที่ 3.13




รูปที่ 3.13 สัญลักษณ์ของวงจรควบคุม

การทำงานของวงจรควบคุมจะรับสัญญาณนาฬิกา sys_clk จากภายนอกผ่านทางขา sys_clk_in ซึ่งจะทำให้เริ่มทำงานตามจังหวะของสัญญาณนาฬิกาที่ความถี่ 640 kHz เพื่อสร้างสัญญาณควบคุมได้แก่ reset, clacc, clk, lacc, lr, sa และ sc ที่จะนำไปควบคุมวงจรย่อยภายในฟิลเตอร์แบงก์ โดยสัญญาณ reset ใช้ควบคุมการรีเซ็ตระบบหลังจากครบรอบการทำงาน สัญญาณ clacc, lacc และ sa ใช้ควบคุมวงจรเลื่อนและบวกสะสม (Scaling Accumulator) สัญญาณ clk ใช้ควบคุมวงจรเลื่อนบิต สัญญาณ lr ใช้ควบคุมการโหลดชุดข้อมูลเข้าและออก สัญญาณ sc ใช้ควบคุมวงจรแปลงสัญญาณเชิงอุปมานเป็นสัญญาณเชิงเลข (Analog to Digital Converter) เป็นความถี่สุ่มตัวอย่างโดยมีค่าความถี่อยู่ที่ 20 kHz การทำงานนี้แสดงผลการจำลองการทำงานในรูปที่ 3.14

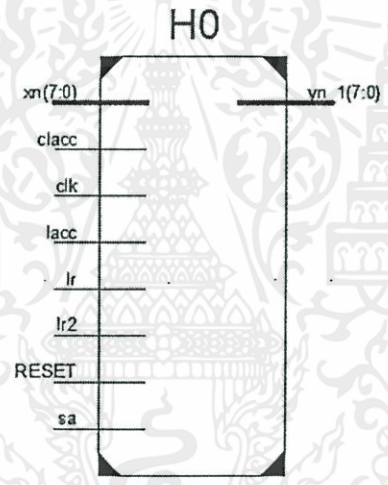



รูปที่ 3.14 ผลการจำลองการทำงานของวงจรควบคุม

3.2.1.2 วงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด

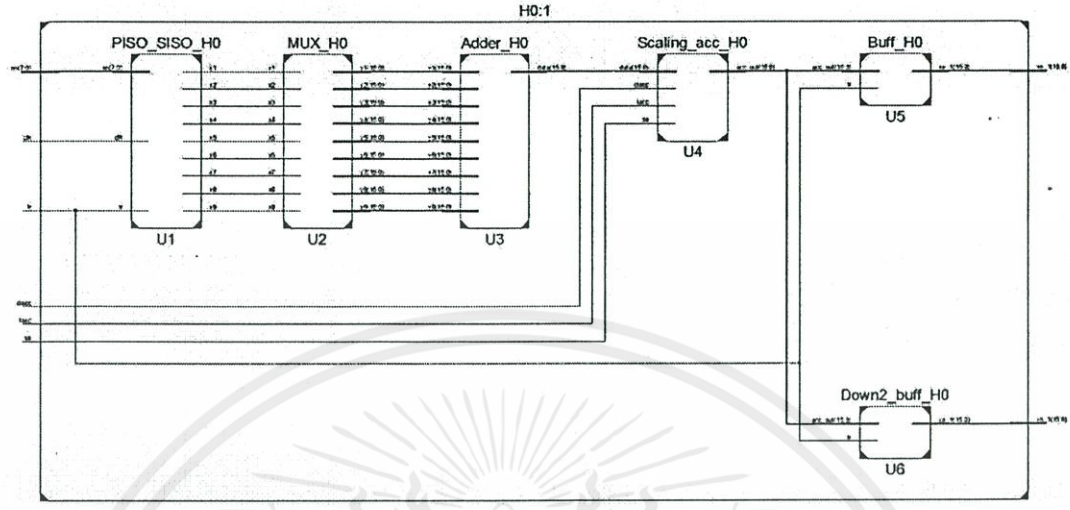
ในหัวข้อนี้จะยกตัวอย่างการออกแบบเพียงเฉพาะตัวกรอง $H_0(z)$ เนื่องจากหลักการในการทำงานและวงจรย่อยภายในของตัวกรองอีกสามตัวมีลักษณะคล้ายกัน แต่จะต่างกันแค่เพียงบางส่วนเท่านั้นซึ่งจะอธิบายเพิ่มเติมในส่วนนี้ภายหลัง

วงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดของตัวกรอง $H_0(z)$ ที่ออกแบบจะเป็นตัวกรองแบบ 9-Tap ซึ่งจะรับสัญญาณอินพุตทางขา xn ในรูปแบบบิต และรับสัญญาณควบคุมจากวงจรควบคุมผ่านทางขา clacc, clk, lacc, lr, reset และ sa แล้วส่งเอาต์พุตออกทางขา yn สามารถเขียนตัวกรองแทนด้วยสัญลักษณ์ดังรูปที่ 3.15




รูปที่ 3.15 สัญลักษณ์ของตัวกรอง $H_0(z)$

จากโครงสร้างของตัวกรองรูปที่ 3.7 ทำการเขียนโปรแกรมบรรยายพฤติกรรมการทำงานของวงจรย่อยต่างๆ โดยเริ่มจากวงจรเลื่อนบิตแบบเข้าขนานออกอนุกรม (PISO Shift Register), วงจรเลื่อนบิตเข้าอนุกรมออกอนุกรม (SISO Shift Register), มัลติเพล็กซ์เซอร์, วงจรบวก สุดท้ายที่วงจรเลื่อนและบวกสะสม (Scaling Accumulator) หลังจากนั้นทำการสังเคราะห์และจำลองการทำงานในแต่ละวงจร ซึ่งสามารถแสดงได้เป็น Schematic Diagram ดังรูปที่ 3.16

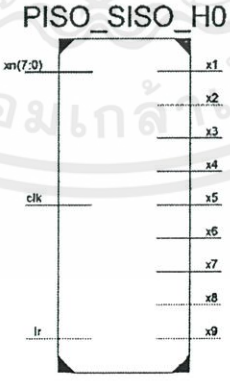



รูปที่ 3.16 Schematic Diagram ของวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด

จากรูปที่ 3.16 วงจรภายในวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดมีรายละเอียดการออกแบบและการทำงานดังนี้

1) วงจรเลื่อนบิต (Shift Register)

วงจรเลื่อนบิตใช้สำหรับควบคุมการส่งข้อมูลเข้าวงจร ซึ่งจะใช้สัญลักษณ์แทนในรูปที่ 3.17

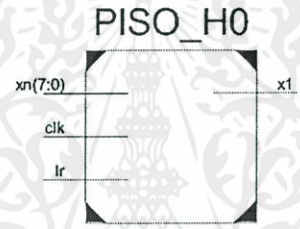



รูปที่ 3.17 สัญลักษณ์ของวงจรเลื่อนบิต

วงจรเลื่อนบิตที่ออกแบบจะมีอยู่ด้วยกัน 2 แบบ ได้แก่

1.1) วงจรเลื่อนบิตแบบเข้าขนานออกอนุกรม

วงจรเลื่อนบิตแบบเข้าขนานออกอนุกรมเป็นวงจรที่ใช้รับข้อมูลเข้าพร้อมกัน 8 บิต ซึ่งถูกควบคุมด้วยสัญญาณ lr ที่ควบคุมการโหลดข้อมูลเข้า จากนั้นจะทำการเลื่อนข้อมูลออกมาทีละ 1 บิต โดยเริ่มจากบิต LSB จนถึง MSB ตามสัญญาณ clk เพื่อนำไปชี้มัลติเพล็กซ์เซอร์ ซึ่งจะใช้จำนวน 1 วงจร ทั้งตัวกรองแบบ 9-Tap และ 7-Tap สามารถเขียนสัญลักษณ์แทนวงจรได้ดังรูปที่ 3.18




รูปที่ 3.18 สัญลักษณ์ของวงจรเลื่อนบิตแบบเข้าขนานออกอนุกรม

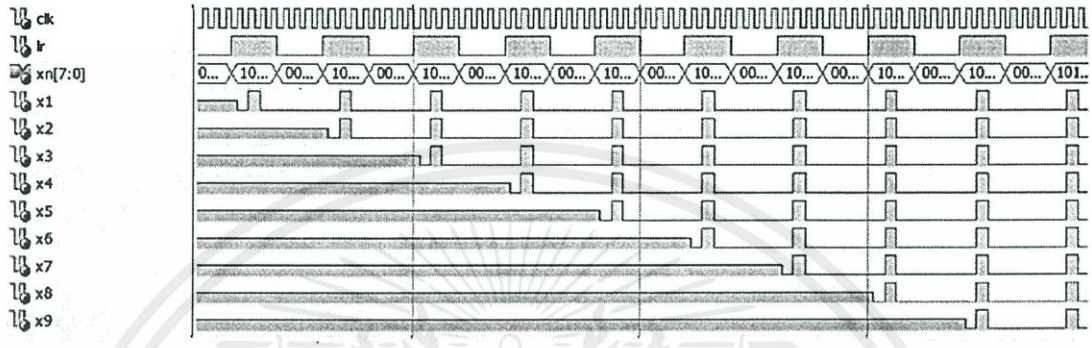
1.2) วงจรเลื่อนบิตแบบเข้าอนุกรมออกอนุกรม

วงจรเลื่อนบิตแบบเข้าอนุกรมออกอนุกรมเป็นวงจรที่ใช้รับข้อมูลเข้าทีละ 1 บิตจากวงจรเลื่อนบิตแบบเข้าขนานออกอนุกรมข้างต้น แล้วจะทำการเลื่อนข้อมูลออกมาทีละ 1 บิต ตามสัญญาณ clk เพื่อนำไปชี้มัลติเพล็กซ์เซอร์ ซึ่งจะใช้จำนวน 8 วงจร สำหรับตัวกรองแบบ 9-Tap และจะใช้จำนวน 6 วงจร สำหรับตัวกรองแบบ 7-Tap สามารถเขียนสัญลักษณ์แทนวงจรได้ดังรูปที่ 3.19




รูปที่ 3.19 สัญลักษณ์ของวงจรเลื่อนบิตแบบเข้าอนุกรมออกอนุกรม

หลังจากทำการออกแบบวงจรครบแล้ว ก็นำมารวมเป็นวงจรเลื่อนบิตตามรูปที่ 3.17 แล้วทำการจำลองการทำงานของวงจรเลื่อนบิต ได้ผลตามรูปที่ 3.20

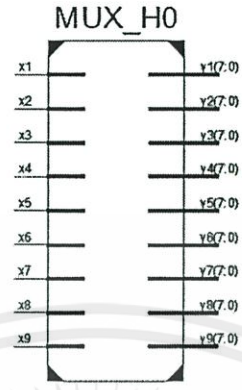



รูปที่ 3.20 ผลการจำลองการทำงานของวงจรเลื่อนบิต

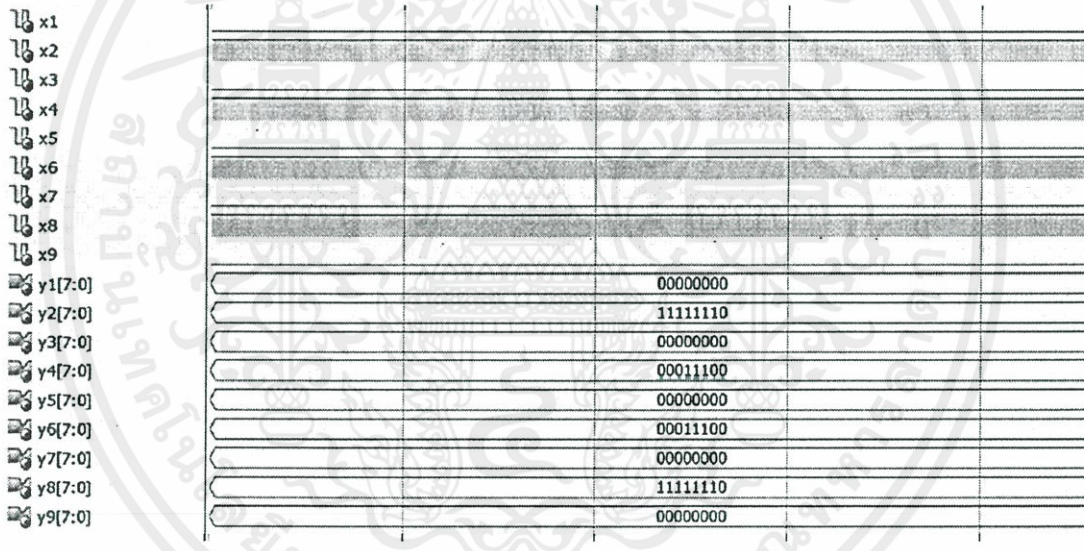
จากผลการจำลองการทำงานจะเห็นว่าเมื่อวงจรเลื่อนบิตแบบเข้าขนานออกอนุกรมได้รับสัญญาณ lr เป็นศูนย์ วงจรจะเริ่มโหลดข้อมูลเข้าและเริ่มส่งข้อมูลออกมาเป็นบิตตามสัญญาณ clk และจะโหลดข้อมูลใหม่ทุกครั้งเมื่อได้รับสัญญาณ lr ที่เป็นศูนย์ ยกตัวอย่างเช่นสัญญาณขาเข้า xn มีค่า 00000010 จะได้สัญญาณขาออกที่เริ่มจากบิต LSB ก่อนตามสัญญาณ clk ได้สัญญาณขาออกเป็น 00000010 จากนั้นจะถูกส่งเข้าวงจรเลื่อนบิตแบบเข้าอนุกรมออกอนุกรมทีละบิตและเลื่อนออกทีละบิต ไปจนครบ 8 วงจร ตามสัญญาณ clk

2) มัลติเพล็กซ์เซอร์ (Multiplexer)

มัลติเพล็กซ์เซอร์เป็นวงจรที่ใช้ควบคุมการเลือกค่าสัมประสิทธิ์ที่ถูกเก็บไว้ออก ซึ่งจะมีสัญญาณข้อมูลที่ถูกเลื่อนบิตมาเป็นตัวชี้มัลติเพล็กซ์เซอร์ โดยถ้าสัญญาณข้อมูลมีค่าเป็นศูนย์จะส่งค่าสัมประสิทธิ์ที่เป็นศูนย์ และถ้ามีค่าเป็นหนึ่งจะส่งค่าสัมประสิทธิ์ที่เก็บไว้ตามตารางที่ 3.3 ออกมา ดังนั้นจำนวนขามัลติเพล็กซ์เซอร์จะขึ้นอยู่กับจำนวนค่าสัมประสิทธิ์ ซึ่งสามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.21 และมีผลการจำลองการทำงานได้ดังรูปที่ 3.22




รูปที่ 3.21 สัญลักษณ์ของมัลติเพล็กซ์เซอร์

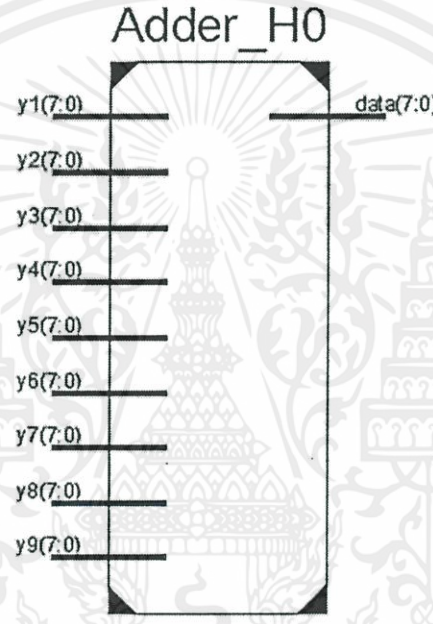



รูปที่ 3.22 ผลการจำลองการทำงานของมัลติเพล็กซ์เซอร์

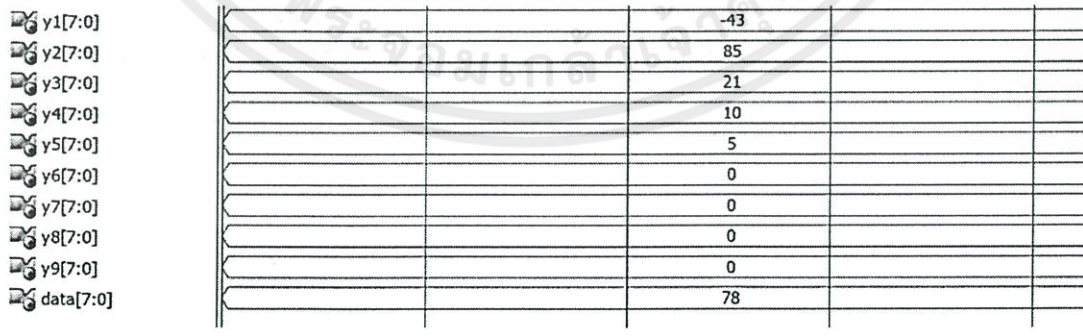
จากผลการจำลองการทำงานเมื่อป้อนสัญญาณข้อมูลที่ถูกเลื่อนบิตที่ขาเข้า x1 ถึง x9 เป็น 010101010 มัลติเพล็กซ์เซอร์จะเลือกค่าสัมประสิทธิ์ที่เป็น 00000000 ออกมาเมื่อได้รับบิตที่เป็นศูนย์ และเมื่อสัญญาณข้อมูลมีบิตเป็นหนึ่ง มัลติเพล็กซ์เซอร์จึงจะเลือกค่าสัมประสิทธิ์ที่ถูกเก็บไว้ออกมาเป็น เช่น ที่ x2 ได้รับบิตมีค่าเป็นหนึ่งจะส่งข้อมูลออกทางขา y2 เป็น 11111110 จะเห็นได้ว่าวงจรทำงานได้ตามที่กำหนดไว้

3) วงจรบวก (Adder)

วงจรบวกเป็นวงจรที่ใช้ในการรวมค่าสัมประสิทธิ์ที่ถูกปล่อยออกมาจากมัลติเพล็กซ์เซอร์แต่ละตัว โดยเมื่อมีค่าข้อมูลเข้ามาก็จะทำการรวมด้วยการบวกในรูปเลขฐานสองแบบส่วนเติมเต็มสอง วงจรสามารถเขียนแทนด้วยสัญลักษณ์ดังรูปที่ 3.23 และมีผลการจำลองการทำงานตามรูปที่ 3.24




รูปที่ 3.23 สัญลักษณ์ของวงจรบวก

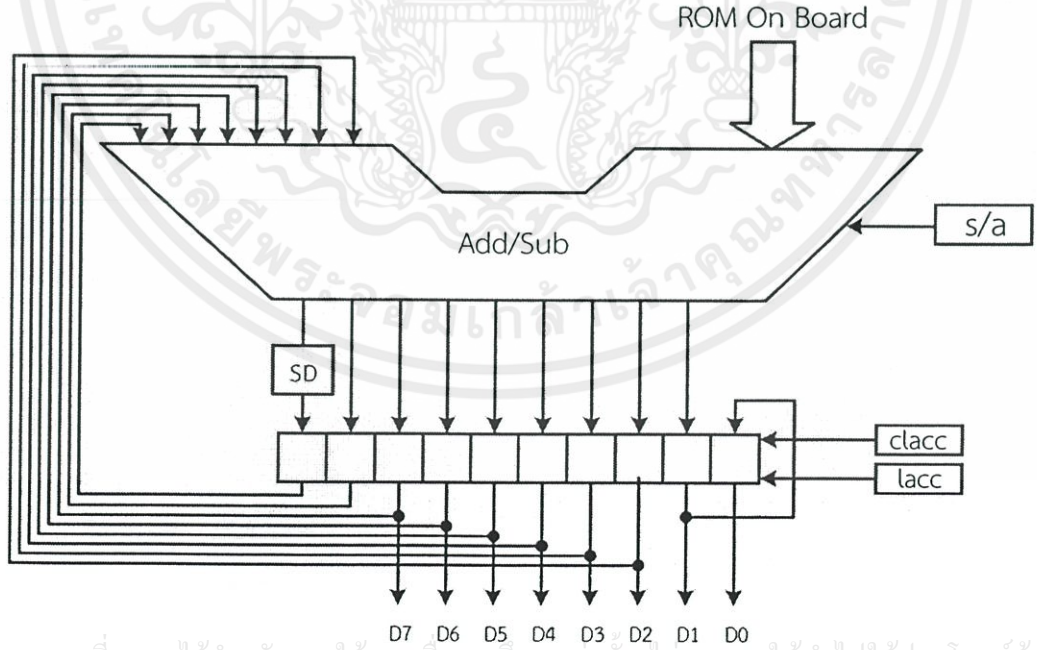



รูปที่ 3.24 ผลการจำลองการทำงานของวงจรบวก

จากผลการจำลองการทำงานจะทำการป้อนข้อมูลเข้าที่ขา y1 ถึง y5 มีค่า -43, 85, 21, 10 และ 5 ตามลำดับ นอกจากนั้นทำการป้อนค่าเป็นศูนย์ จะเห็นว่าวงจรจะทำการรวมค่าทั้งหมดของขาเข้าได้เป็น 78 ซึ่งเป็นค่าที่ถูกต้อง ดังนั้นวงจรบวกที่ออกแบบสามารถทำงานได้ถูกต้องตามที่กำหนดไว้

4) วงจรเลื่อนและบวกสะสม (Scaling Accumulator) [10]

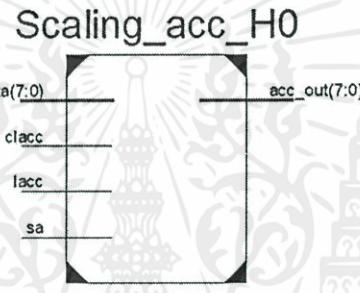
วงจรเลื่อนและบวกสะสมเป็นวงจรที่อาศัยคุณสมบัติในการบวกลบเลขฐานสองแบบส่วนเติมเต็มสองแทนการคูณโดยตรง ซึ่งประกอบด้วยวงจรบวก/ลบ (Add/Sub) และรีจิสเตอร์แอคคิวมูเลเตอร์ โดยสัญญาณอินพุตหนึ่งของวงจรบวก/ลบ จะถูกออกแบบเป็นฮาร์ดแวร์สเกลลิ่ง (Hardware Scaling) ด้วย 1/2 ก่อนจะนำมาคำนวณในวงจรบวก/ลบต่อไป และเพื่อเป็นการหลีกเลี่ยงการเลื่อนบิตผิดพลาดไปทับตำแหน่งของบิตเครื่องหมายในกรณีที่มีการล้น (Overflow) ดังนั้นต้องมีวงจรเพิ่มเติมสำหรับตรวจสอบบิตเครื่องหมายโดยใช้ XOR เกต ซึ่งทำการตรวจสอบภายในวงจรบวก/ลบ ลักษณะการทำงานนี้สามารถเขียนโครงสร้างของวงจรเลื่อนและบวกสะสมได้ตามรูปที่ 3.25



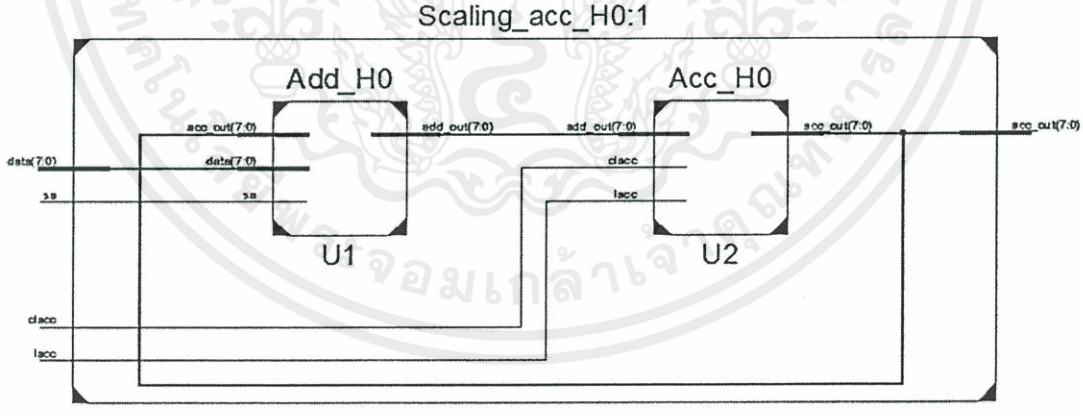



รูปที่ 3.25 โครงสร้างของวงจรเลื่อนและบวกสะสม

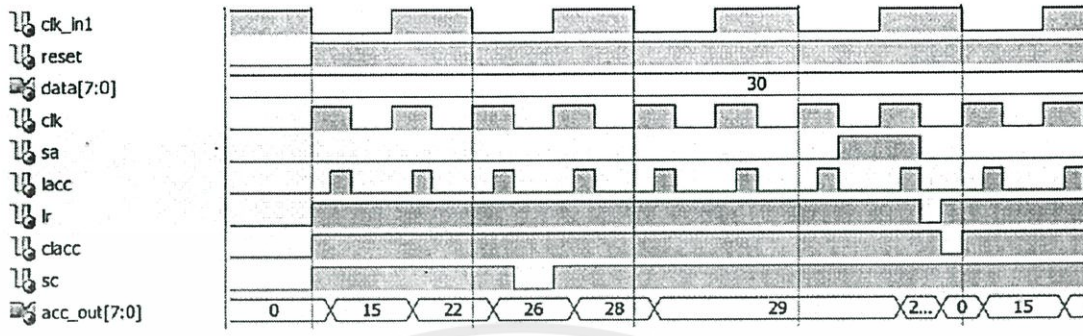
จากรูปที่ 3.25 จะแสดงลักษณะวงจรเลื่อนและบวกสะสมโดยมีวงจร SD ที่ใช้เพื่อป้องกันการเกิดการล้นของข้อมูลแล้วไปทับตำแหน่งบิตเครื่องหมาย และแสดงการเชื่อมโยงของสัญญาณในลักษณะฮาร์ดแวร์สเกลลิ่งด้วย 1/2 หรือเลื่อนข้อมูลไปทางขวา 1 บิต ซึ่งวงจร SD เสมือนเป็นวงจรที่ใช้เติมบิตเครื่องหมายหลังจากการเลื่อนข้อมูลไปทางขวา โดยภายในวงจรเลื่อนและบวกสะสมสามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.26 ส่วนวงจรที่ถูกสังเคราะห์สามารถแสดงเป็น Schematic Diagram ได้ดังรูปที่ 3.27 และมีผลการจำลองการทำงานได้ตามรูปที่ 3.28




รูปที่ 3.26 สัญลักษณ์ของวงจรเลื่อนและบวกสะสม




รูปที่ 3.27 Schematic Diagram ของวงจรเลื่อนและบวกสะสม

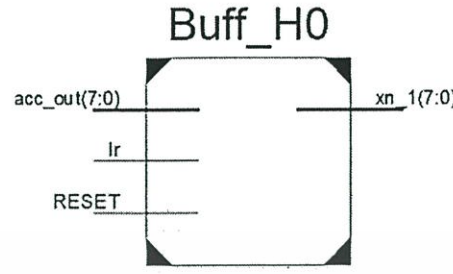



รูปที่ 3.28 ผลการจำลองการทำงานของวงจรเลื่อนและบวกสะสม

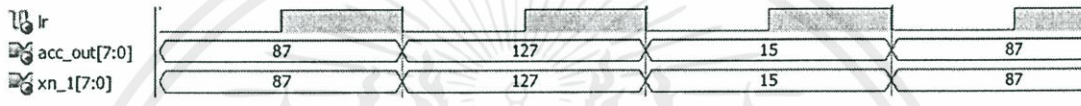
จากผลการจำลองการทำงานจะเห็นว่าเมื่อสัญญาณ sa เป็นศูนย์จะทำให้วงจรเลื่อนและบวกสะสมเริ่มทำงาน ในขณะเดียวกันสัญญาณ claac เป็นศูนย์จะทำการเคลียร์ค่าภายในแอคคิวมูเลเตอร์ แล้วเมื่อวงจรได้รับสัญญาณ lacc เป็นหนึ่ง ก็จะทำการโหลดค่าแอคคิวมูลแตอร์ที่มีอยู่ในขณะนั้นมาคำนวณ ยกตัวอย่างเช่น ป้อนสัญญาณเข้าขา data มีค่าเป็น 30 เมื่อวงจรโหลดค่าเข้าด้วยสัญญาณ lacc วงจรจะทำการบวกค่า 30+0 = 30 นำไปสเกลค่าได้ 30/2 = 15 จากนั้นเมื่อมีสัญญาณ lacc อีกครั้ง วงจรโหลดค่า 30 เข้ามา ก็จะทำการบวกค่า 30+15 = 45 นำไปสเกลค่า 45/2 = 22.5 แต่ค่าที่แสดงเป็น 22 เนื่องจากการจำลองการทำงานนี้มองเลขฐานสองเป็นจำนวนเต็ม วงจรจะคำนวณไปจนกว่าจะมีการเคลียร์ค่าภายในแอคคิวมูเลเตอร์ด้วยสัญญาณ clacc

5) วงจรบัฟเฟอร์ (Buffer)

วงจรบัฟเฟอร์เป็นวงจรที่ใช้ในการโหลดค่าสัญญาณออกของวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด โดยจะรับค่าข้อมูลขาเข้ามาจากวงจรเลื่อนและบวกสะสม ซึ่งจะโหลดข้อมูลเมื่อครบรอบการทำงานของวงจรเลื่อนและบวกสะสมโดยมีสัญญาณ lr เป็นตัวควบคุม เมื่อได้รับสัญญาณ lr วงจรบัฟเฟอร์จะโหลดข้อมูลเข้ามาและเป็นสัญญาณออกของวงจรกรองต่อไป สัญญาณที่ใช้แทนวงจรบัฟเฟอร์มีลักษณะตามรูปที่ 3.29 และมีผลการจำลองการทำงานดังรูปที่ 3.30




รูปที่ 3.29 สัญลักษณ์ของวงจรบัฟเฟอร์

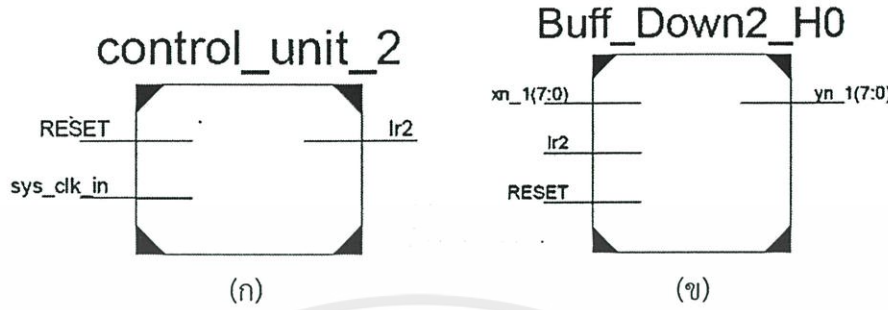



รูปที่ 3.30 ผลการจำลองการทำงานของวงจรบัฟเฟอร์

จากผลการจำลองการทำงานจะเห็นว่าเมื่อได้รับสัญญาณ lr ทุกครั้งที่เป็นศูนย์ จะเริ่มทำการโหลดข้อมูลเข้าสู่วงจรและปล่อยออกในเวลาเดียวกัน ดังตัวอย่างเช่น เมื่อได้รับสัญญาณ lr วงจรจะโหลดค่า 32704 เข้ามาและปล่อยออกไปพร้อมกัน และเมื่อได้รับสัญญาณ lr อีกครั้งก็โหลดข้อมูลเข้ามาใหม่คือค่า 3855 เป็นอย่างนี้ไปทุกครั้งที่ได้รับสัญญาณ lr มีค่าเป็นศูนย์

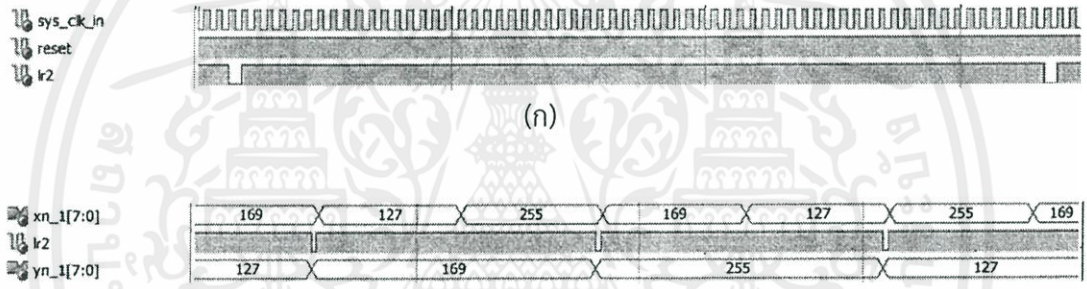
3.2.1.3 วงจรลดค่าสุ่มสัญญาณ (Decimator)

วงจรลดค่าสุ่มสัญญาณออกแบบให้ลดค่าสุ่มสัญญาณลงสอง ประกอบไปด้วย 2 วงจร ได้แก่ วงจรสร้างสัญญาณ lr2 และวงจรบัฟเฟอร์ ซึ่งสัญญาณ lr2 มีความถี่ของสัญญาณ lr โดยสัญญาณ lr2 ถูกนำไปควบคุมการโหลดข้อมูลออกจากวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดของวงจรบัฟเฟอร์ สามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.31 และมีผลการจำลองการทำงานดังรูปที่ 3.32




(ก) (ข)

รูปที่ 3.31 (ก) สัญลักษณ์ของวงจรสร้างสัญญาณควบคุม lr2
(ข) สัญลักษณ์ของวงจรบัฟเฟอร์สำหรับลดค่าสุ่มสัญญาณ




(ก)

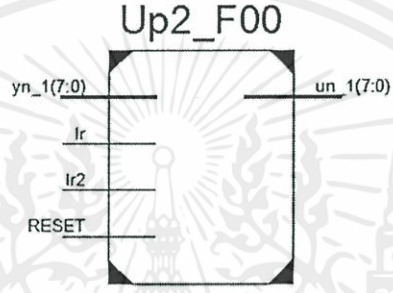
(ข)

รูปที่ 3.32 (ก) ผลการจำลองการทำงานของวงจรสร้างสัญญาณควบคุม lr2
(ข) ผลการจำลองการทำงานของวงจรบัฟเฟอร์สำหรับลดค่าสุ่มสัญญาณ

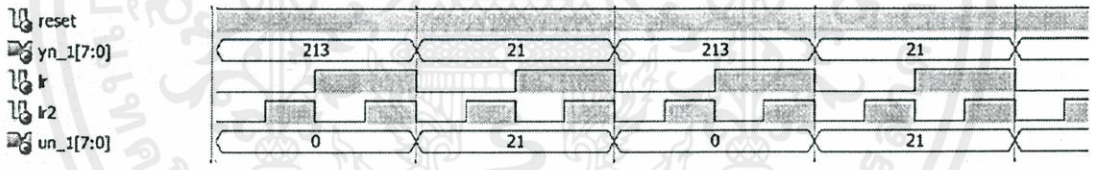
จากผลการจำลองการทำงานจะเห็นว่าสัญญาณ lr2 ที่ถูกสร้างขึ้นจากสัญญาณ sys_clk_in จำนวน 64 คาบ ดังรูปที่ 3.32 (ก) แต่สัญญาณ lr ถูกสร้างขึ้นจากสัญญาณ sys_clk_in จำนวน 32 คาบ ดังรูปที่ 3.14 แสดงให้เห็นว่าสัญญาณ lr2 มีความถี่ลดลงสองเท่าของสัญญาณ lr ดังนั้นสัญญาณ lr2 จึงควบคุมการทำงานของวงจรบัฟเฟอร์สำหรับลดค่าสุ่มสัญญาณลง 2 ได้ ดังแสดงได้ในรูปที่ 3.32 (ข) โดยเมื่อวงจรบัฟเฟอร์ได้รับสัญญาณ lr2 ทุกครั้งที่เป็นศูนย์ จะเริ่มทำการโหลดข้อมูลเข้าสู่วงจรและปล่อยออกในเวลาเดียวกัน ดังตัวอย่างเช่น เมื่อได้รับสัญญาณ lr2 วงจรจะโหลดค่า 169 เข้ามาและปล่อยออกไปพร้อมกัน และเมื่อได้รับสัญญาณ lr2 อีกครั้งก็โหลดข้อมูลเข้ามาใหม่คือค่า 255 เป็นอย่างนี้ไปทุกครั้งที่ได้รับสัญญาณ lr2 มีค่าเป็นศูนย์ จะเห็นว่าวงจรจะไม่นำค่า 127 ปล่อยออกมา

3.2.1.4 วงจรเพิ่มค่าสุ่มสัญญาณ (Expander)

วงจรเพิ่มค่าสุ่มสัญญาณออกแบบให้เพิ่มค่าสุ่มสัญญาณขึ้นสอง โดยอาศัยการแทรกค่าศูนย์ระหว่างค่าของตัวอย่าง โดยออกแบบให้ทำการนับตามสัญญาณ lr ถ้าเป็นลำดับเลขคี่เอาต์พุตจะนำค่าอินพุตส่งออกไป แต่ถ้าเป็นลำดับเลขคู่เอาต์พุตให้ส่งค่าศูนย์ออก สามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.33 และมีผลการจำลองการทำงานดังรูปที่ 3.34




รูปที่ 3.33 สัญลักษณ์ของวงจรเพิ่มค่าสุ่มสัญญาณ

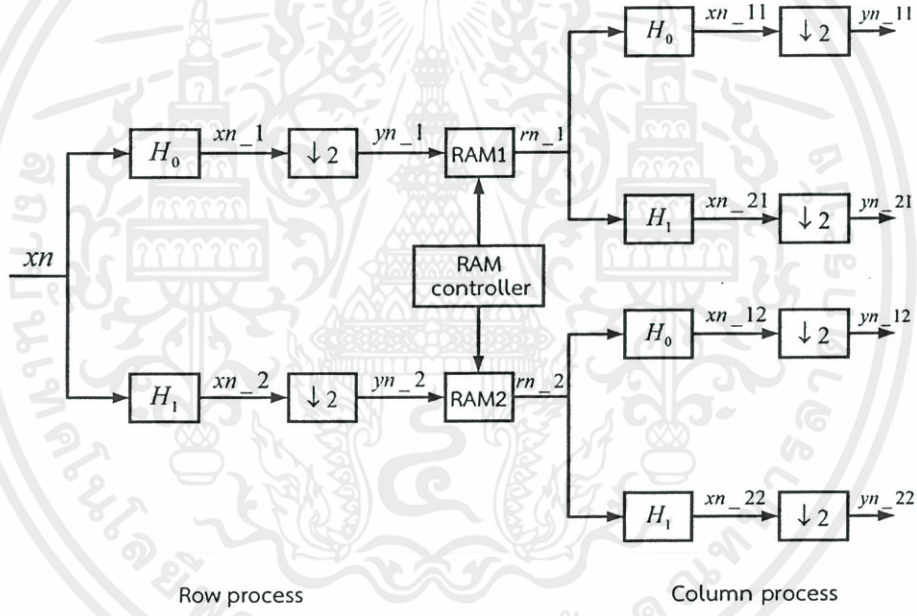



รูปที่ 3.34 ผลการจำลองการทำงานของวงจรเพิ่มค่าสุ่มสัญญาณ

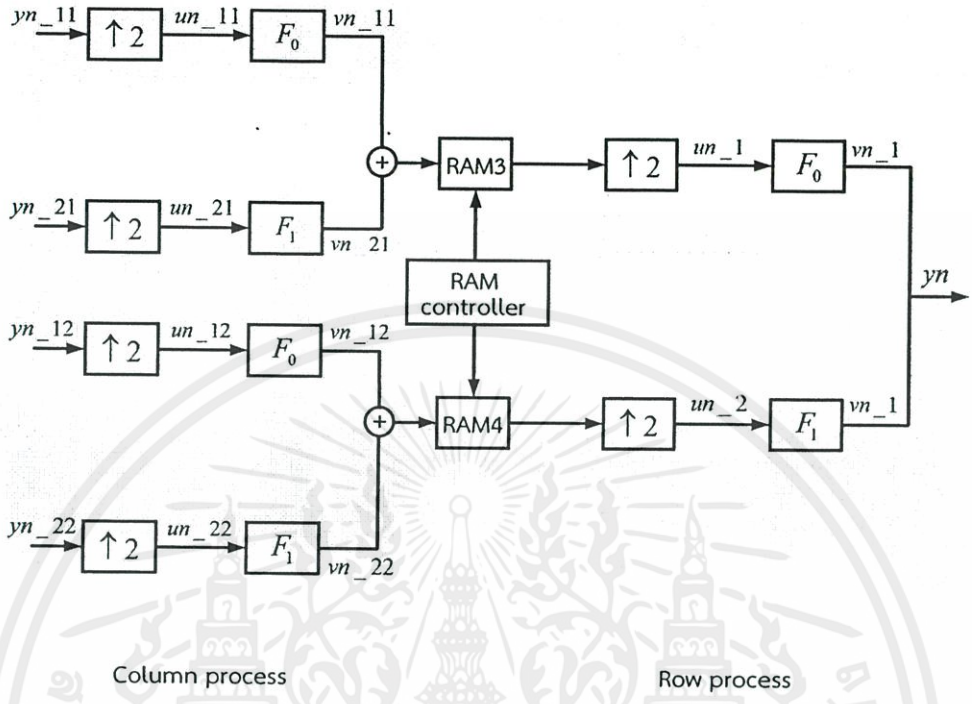
จากผลการจำลองการทำงานจะเห็นว่าเมื่อได้รับสัญญาณ lr ที่มีค่าเป็นศูนย์จะทำการโหลดค่าข้อมูล ดังตัวอย่างตามรูปจะโหลดค่า 21 ซึ่งเป็นลำดับเลขคู่แล้วส่งออกไป และเมื่อได้รับสัญญาณควบคุมอีกครั้งก็จะโหลดค่า 0 แล้วส่งออกไป ทำสลับกันไปก็จะได้สัญญาณที่ถูกเพิ่มค่าสุ่มสัญญาณขึ้นสอง ซึ่งวงจรในส่วนนี้จะนำไปขาเข้าของวงจรต่อกับขาสัญญาณที่ออกจากตัวกรอง $H_0(z)$ และ $H_1(z)$ ส่วนขาออกของวงจรจะต่อกับขาเข้าของตัวกรอง $F_0(z)$ และ $F_1(z)$

### 3.2.2 การออกแบบฟิลเตอร์แบงก์แบบ 2 มิติ

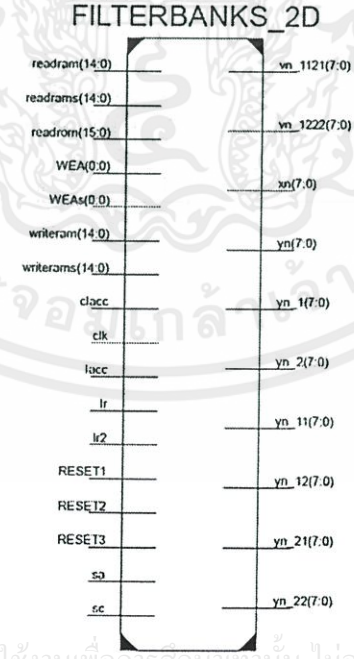
ในการออกแบบฟิลเตอร์แบงก์แบบ 2 มิติ จะอาศัยพื้นฐานการออกแบบฟิลเตอร์แบงก์แบบ 1 มิติ แต่จะมีการเพิ่มเติมรายละเอียดอุปกรณ์ภายในบางส่วน การเพิ่มอุปกรณ์ควบคุมและสัญญาณควบคุม รวมทั้งเพิ่มระบบการแสดงผลบนจอภาพวีจีเอ การสร้างฟิลเตอร์แบงก์จะใช้โครงสร้างของตัวกรองเป็นฟิลเตอร์แบงก์แบบ 2 ระดับ ตามรูปที่ 2.3 ซึ่งการสร้างเป็นฮาร์ดแวร์นั้นจะมีรายละเอียดของอุปกรณ์ที่ทำการสังเคราะห์แบ่งเป็นสองส่วนคือโครงสร้างในส่วนวิเคราะห์ (Analysis Part) และส่วนสังเคราะห์ (Synthesis Part) ดังรูปที่ 3.35 และ 3.36 ตามลำดับ สำหรับวงจรที่สังเคราะห์ได้แสดงเป็น Schematic Diagram ได้ดังรูปที่ 3.37



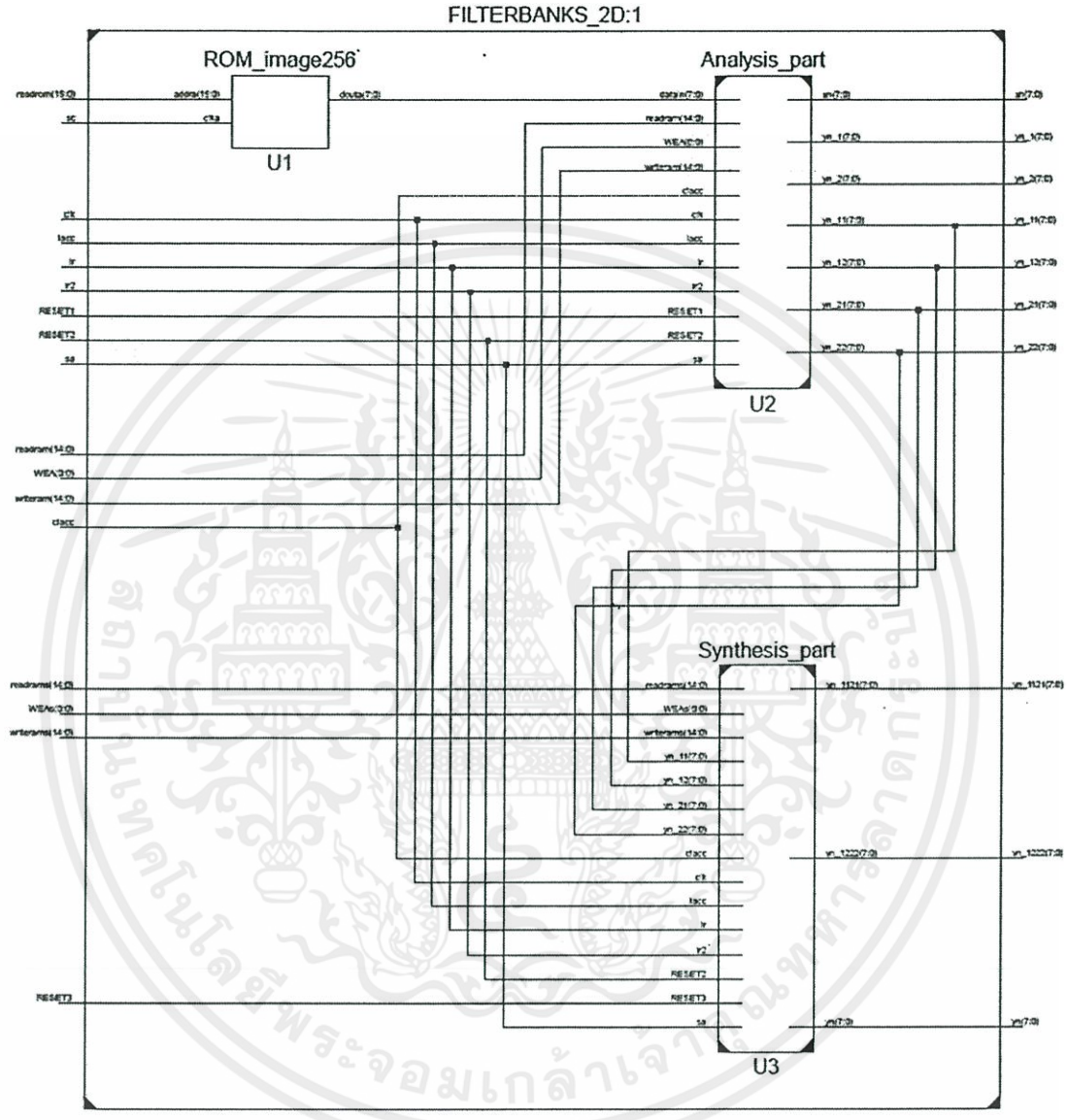

รูปที่ 3.35 โครงสร้างของฟิลเตอร์แบงก์แบบ 2 มิติในส่วนวิเคราะห์ (Analysis Part)




รูปที่ 3.36 โครงสร้างของฟิลเตอร์แบงก์แบบ 2 มิติในส่วนสังเคราะห์ (Synthesis Part)




(ก)




(ข)

รูปที่ 3.37 (ก) สัญลักษณ์ของฟิลเตอร์แบงก์แบบ 2 มิติที่ออกแบบด้วยภาษา VHDL

(ข) Schematic Diagram ของฟิลเตอร์แบงก์แบบ 2 มิติ

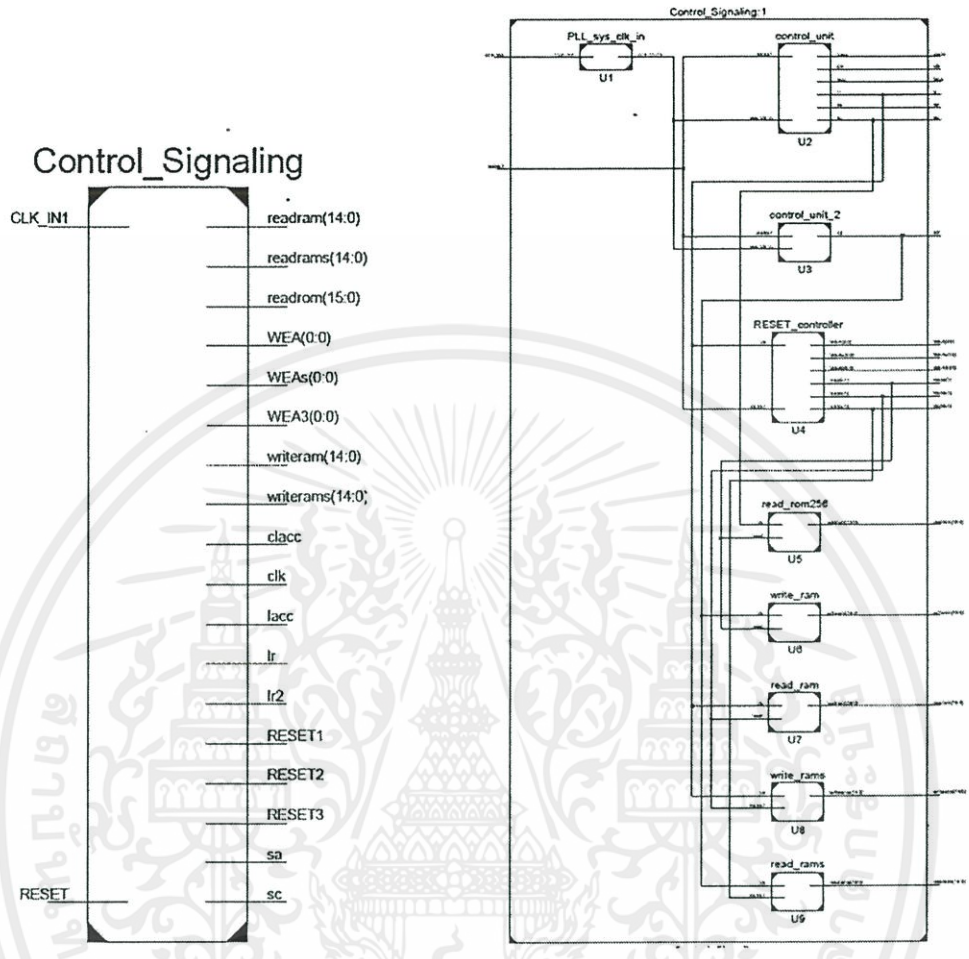
หลักการการทำงานของฟิลเตอร์แบงก์แบบ 2 มิติที่ออกแบบนั้น จะเริ่มต้นด้วยการจัดเก็บข้อมูลภาพลงในหน่วยความจำบนบอร์ดแล้วนำไปประมวลผลโดยอาศัยหลักการแยกการทำคอนโวลูชั่นแบบ 2 มิติ ซึ่งจะใช้การประมวลผลการคำนวณแถว (Row Processing) ในสเตจที่ 1 แล้วจึงนำไปประมวลผลการคำนวณคอลัมน์ (Column Processing) ในสเตจที่ 2 หลังจากนั้นทำการสังเคราะห์ย้อนกลับจนได้ภาพผลลัพธ์ กระบวนการนี้ถือเป็นการประมวลผลระดับ 1 (Level 1 Process) ซึ่งในปริญญานิพนธ์นี้จะทำการจำลองภาพผลลัพธ์ตามทฤษฎีจนถึงการประมวลผลระดับ 3 (Level 3 Process) เฉพาะบนโปรแกรมแมตแล็บ แต่สำหรับการจำลองการทำงานของฮาร์ดแวร์และการสร้างฮาร์ดแวร์นั้นจะทำเพียงแค่การประมวลผลระดับ 1

ในการประมวลผลการคำนวณทั้งแถวและคอลัมน์จะมีอยู่ในส่วนวิเคราะห์และสังเคราะห์ และระหว่างการทำงานจะมีการจัดเก็บผลในแต่ละขั้นของการคำนวณด้วยหน่วยความจำชั่วคราว (RAM) เพื่อนำไปใช้ในการคำนวณขั้นต่อไป หรือใช้แสดงผลบนจอวีจีเอ โดยหน่วยความจำชั่วคราวนี้ก็จะมีอยู่ในทั้งสองส่วนเช่นกัน ดังนั้นในปริญญานิพนธ์จะกล่าวถึงเพียงการออกแบบอุปกรณ์ในส่วนวิเคราะห์เท่านั้น เพราะส่วนสังเคราะห์มีลักษณะคล้ายกัน

การสร้างฟิลเตอร์แบงก์แบบ 2 มิติจะมีอุปกรณ์ที่เพิ่มเติมมาดังนี้

3.2.2.1 วงจรสร้างสัญญาณควบคุม (Control Signaling)

วงจรสร้างสัญญาณควบคุมจะสร้างสัญญาณที่ใช้ควบคุมทั้งระบบ โดยสร้างสัญญาณควบคุมวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด (FIR Filter) วงจรควบคุมการรีเซ็ต (Reset Controller) วงจรควบคุมการทำงานของหน่วยความจำบนบอร์ด (ROM Controller) และวงจรควบคุมการทำงานของหน่วยความจำชั่วคราว (RAM Controller) สามารถแสดงวงจรที่สังเคราะห์ได้เป็น Schematic Diagram ดังรูปที่ 3.38




รูปที่ 3.38 (ก) สัญลักษณ์ของวงจรสร้างสัญญาณควบคุม
(ข) Schematic Diagram ของวงจรสร้างสัญญาณควบคุม

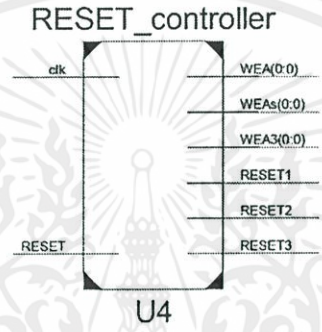
จากรูปที่ 3.38 มีรายละเอียดอุปกรณ์ภายในดังต่อไปนี้

1) วงจรควบคุมวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด (Control Unit)

วงจรควบคุมนี้จะทำงานเหมือนกับวงจรควบคุมในฟิลเตอร์แบงก์แบบ 1 มิติ แตกต่างกันตรงที่สัญญาณนาฬิกาที่มีความถี่ 800 MHz เพื่อให้ได้สัญญาณ sc ที่ความถี่ 25 MHz ซึ่งจะสัมพันธ์กับความถี่ที่ใช้ส่งข้อมูลไปแสดงผลบนจอวีจีเอ

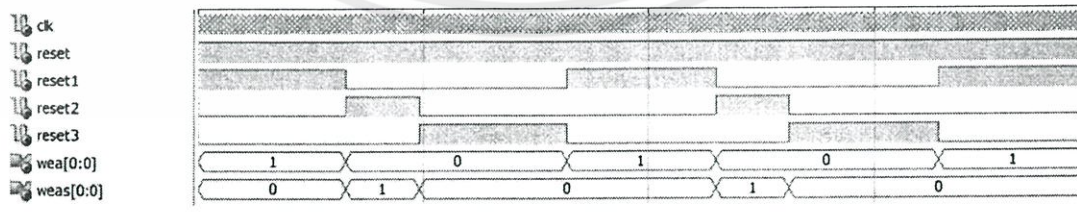
2) วงจรควบคุมการรีเซ็ต (Reset Controller)

วงจรควบคุมการรีเซ็ตใช้เพื่อสร้างสัญญาณควบคุมการคำนวณภายในให้เริ่มทำงาน (wea, weas) และจบกระบวนการทำงาน (reset, reset1, reset2, reset3) เมื่อทำการประมวลผลครบ 1 รอบการทำงาน ซึ่งจะนับตามจำนวนพิกเซลทั้งหมดที่ประกอบกันเป็นภาพขนาด 256 แถว 256 คอลัมน์ สามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.39




รูปที่ 3.39 สัญลักษณ์ของวงจรควบคุมการรีเซ็ต

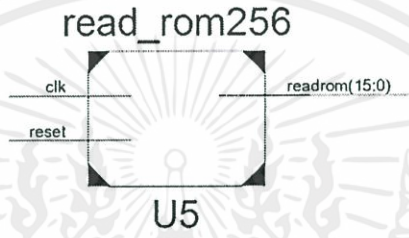
จากรูปที่ 3.39 วงจรจะสร้างสัญญาณจากการรับสัญญาณ lr จากวงจรควบคุมข้างต้นผ่านทางขา clk แล้วสร้างสัญญาณ reset, reset1, reset2 และ reset3 ที่จะไปควบคุมการคำนวณแต่ละส่วนได้แก่ วงจรควบคุมวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัด ส่วนประมวลผลการคำนวณแถวฝั่งวิเคราะห์ ส่วนประมวลผลการคำนวณคอลัมน์ทั้งฝั่งวิเคราะห์และสังเคราะห์ และส่วนประมวลผลคำนวณแถวฝั่งสังเคราะห์ รวมทั้ง wea และ weas ไปควบคุมหน่วยความจำชั่วคราวและวงจรแสดงผล มีผลการจำลองการทำงานได้ดังรูปที่ 3.40




รูปที่ 3.40 ผลการจำลองการทำงานของวงจรควบคุมการรีเซ็ต

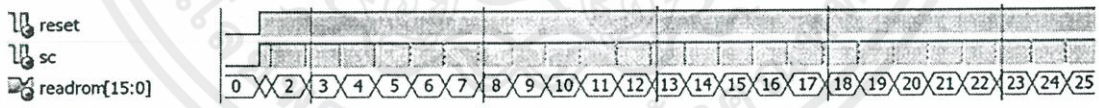
3) วงจรควบคุมการทำงานของหน่วยความจำบนบอร์ด (ROM Controller)

วงจรควบคุมการทำงานของหน่วยความจำบนบอร์ดใช้สร้างสัญญาณควบคุมจังหวะการอ่านข้อมูลภาพ ซึ่งจะนำไปชี้เป็นค่าตำแหน่ง (Address) ของข้อมูลภาพที่จัดเก็บไว้ในหน่วยความจำบนบอร์ด สามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.41




รูปที่ 3.41 สัญลักษณ์ของวงจรควบคุมการทำงานของหน่วยความจำบนบอร์ด

จากรูปที่ 3.41 วงจรจะสร้างสัญญาณค่าตำแหน่งเริ่มจากค่า 0000000000000000 จนถึง 1111111111111111 โดยการรับสัญญาณ sc จากวงจรควบคุมข้างต้นผ่านทางขา clk แล้วส่งค่าตำแหน่งที่ได้จำนวน 16 บิตออกทางขา readrom โดยมีผลการจำลองการทำงานได้ดังรูปที่ 3.42

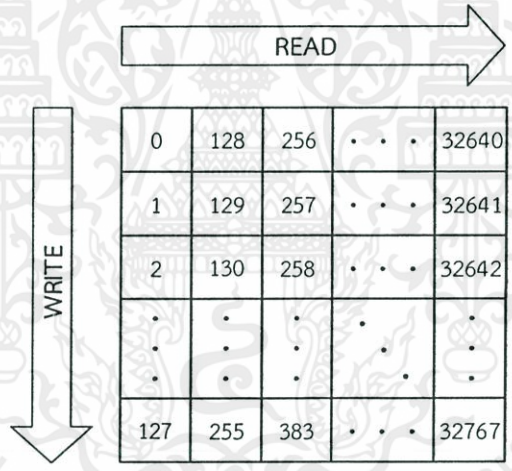



รูปที่ 3.42 ผลการจำลองการทำงานของวงจรควบคุมการทำงานของหน่วยความจำบนบอร์ด

จากผลการจำลองการทำงานตามรูปที่ 3.45 จะพบว่าวงจรควบคุมหน่วยความจำบนบอร์ดจะทำงานเมื่อได้รับสัญญาณควบคุม sc ที่มีค่าเป็นศูนย์ โดยจะสร้างค่าตำแหน่งเรียงจากตำแหน่งแรกคือ 0 ไปจนถึงค่าสุดท้ายคือ 65535 ได้อย่างถูกต้องตรงตามที่ได้ออกแบบไว้

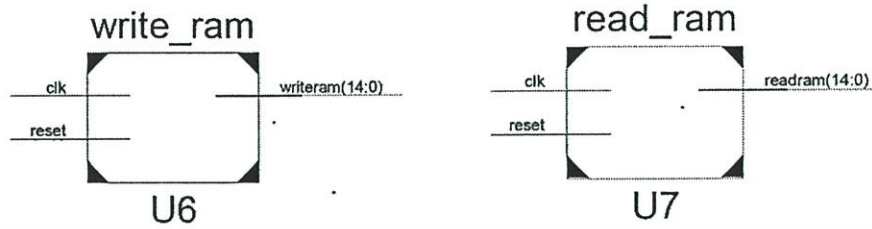
4) วงจรควบคุมการทำงานของหน่วยความจำชั่วคราว (RAM Controller)

วงจรควบคุมการทำงานของหน่วยความจำชั่วคราวจะประกอบด้วยวงจรย่อย 2 วงจร คือวงจรสร้างค่าตำแหน่งสำหรับการเขียน (Write RAM) และสำหรับการอ่าน (Read RAM) ทั้ง 2 วงจรจะทำหน้าที่สร้างค่าตำแหน่งซึ่งจะถูกใช้เป็นค่าอ้างอิงสำหรับการใช้เขียนและอ่านข้อมูลภาพบนหน่วยความจำชั่วคราวตามลำดับ โดยจะสร้างค่าตำแหน่งตามสัญญาณควบคุม lr และ lr2 ผ่านทางขา clk และจะสร้างค่าตำแหน่งเป็นเลขฐานสองไปจนถึงค่าสุดท้ายในทิศทางการสร้างค่าตำแหน่งในส่วนวิเคราะห์ตามรูปที่ 3.43 คือค่าตำแหน่งสำหรับการเขียนสร้างในแนวคอลัมน์ และค่าตำแหน่งสำหรับการอ่านสร้างในแนวแถว ที่จะส่งออกเป็นข้อมูล 15 บิตผ่านทางขา writeram และ readram ตามลำดับ



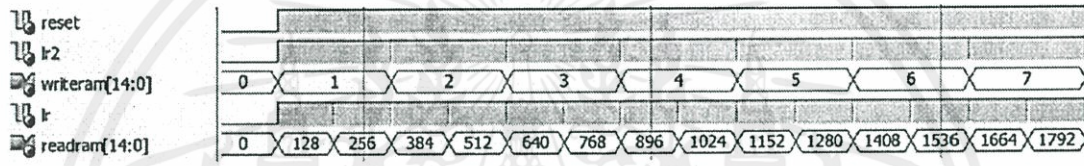

รูปที่ 3.43 ทิศทางการสร้างค่าตำแหน่งในส่วนวิเคราะห์

วงจรทั้งสองจะมีทั้งหมด 2 ชุด โดยจะทำงานกันตรงข้ามกัน แต่ในปริญญานิพนธ์จะแสดงแค่ในส่วนวิเคราะห์ ซึ่งสามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.44 และมีผลการจำลองการทำงานดังรูปที่ 3.45




(ก) (ข)

รูปที่ 3.44 (ก) สัญลักษณ์ของวงจรสร้างค่าตำแหน่งสำหรับการเขียนในส่วนวิเคราะห์
(ข) สัญลักษณ์ของวงจรสร้างค่าตำแหน่งสำหรับการอ่านในส่วนวิเคราะห์

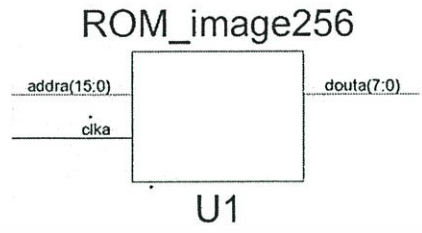



รูปที่ 3.45 ผลการจำลองการทำงานของวงจรควบคุมการทำงานหน่วยความจำชั่วคราวฝั่งวิเคราะห์

จากผลจำลองการทำงานรูปที่ 3.45 วงจรควบคุมหน่วยความจำชั่วคราวสามารถสร้างค่าตำแหน่งสำหรับการอ่านในทิศทางแนวแถวได้อย่างถูกต้องคือเมื่อวงจรได้รับสัญญาณควบคุมจังหวะ lr หรือ lr2 ที่มีค่าเป็นศูนย์จะสร้างค่าตำแหน่งเรียงไปจนจบที่ค่าสุดท้ายตามรูปที่ 3.43

3.2.2.2 หน่วยความจำบนบอร์ด (ROM On Board)

หน่วยความจำบนบอร์ดจะออกแบบให้จัดเก็บข้อมูลภาพขนาด 256 แถว 256 คอลัมน์ แต่ละพิกเซลขนาด 8 บิต ทำให้ต้องการพื้นที่ทั้งหมดประมาณ 65 กิโลไบต์ สามารถเขียนแทนด้วยสัญลักษณ์ตามรูปที่ 3.46



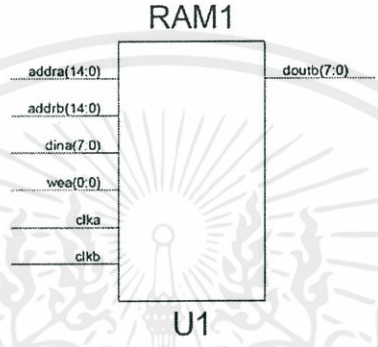

รูปที่ 3.46 สัญลักษณ์ของหน่วยความจำบนบอร์ด

จากรูปที่ 3.46 หน่วยความจำบนบอร์ดจะทำการรับค่าตำแหน่งจากวงจรควบคุมการทำงานของหน่วยความจำบนบอร์ดภายในวงจรสร้างสัญญาณควบคุมผ่านทางขา addra ที่จะใช้เรียกข้อมูลภาพทีละพิกเซลเป็นค่าขนาด 16 บิต เริ่มจากค่า 0 จนถึง 65535 ซึ่งจะทำงานสัมพันธ์สัญญาณ sc ที่เข้าทางขา clka โดยเรียกข้อมูลในอัตราเดียวกันกับความถี่สุ่มตัวอย่าง และส่งข้อมูลเอาต์พุตออกทางขา douta เป็นข้อมูลขนาด 8 บิต

3.2.2.3 หน่วยความจำชั่วคราว (RAM)

หน่วยความจำชั่วคราวเป็นส่วนที่ใช้เก็บข้อมูลภาพแต่ละพิกเซลในระหว่างกระบวนการทำงาน เพื่อพักข้อมูลก่อนนำไปใช้ในการคำนวณในขั้นต่อไป หรือนำไปแสดงผลบนจอแสดงผลวีจีเอ ซึ่งสามารถแบ่งออกได้เป็นสองส่วนตามลักษณะการทำงานข้างต้นคือ ส่วนแรกจะอยู่ภายในวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดมีจำนวน 4 ตัว แบ่งเป็น 2 ชุด โดยจะเก็บภาพขนาด 256 X 128 X 8 บิต เป็นพื้นที่รวมทั้งหมด 32 กิโลไบต์ และส่วนที่สองจะอยู่ภายในวงจรควบคุมการแสดงผลข้อมูล (Display Controller) มีจำนวน 10 ตัว โดยแบ่งเก็บภาพตามขนาดดังนี้ เก็บภาพอินพุตและเอาต์พุตขนาด 256 X 256 X 8 บิต จำนวน 2 ตัว เป็นพื้นที่รวมทั้งหมด 131 กิโลไบต์ เก็บภาพที่ผ่านสเตจที่ 1 ฝั่งวิเคราะห์และภาพที่ผ่านสเตจ ที่ 2 ฝั่งสังเคราะห์ขนาด 256 X 128 X 8 บิต จำนวน 4 ตัว เป็นพื้นที่รวมทั้งหมดประมาณ 131 กิโลไบต์ เก็บภาพที่ผ่านสเตจที่ 2 ฝั่งวิเคราะห์ก่อนเข้าประมวลผลในสเตจ ที่ 2 ฝั่งสังเคราะห์ขนาด 128 X 128 X 8 บิต จำนวน 4 ตัว เป็นพื้นที่รวมทั้งหมด 65 กิโลไบต์ หน่วยความจำชั่วคราวทั้งหมดนี้ข้อมูลแต่ละพิกเซลจะถูกเก็บโดยมีค่าตำแหน่งบนหน่วยความจำชั่วคราวเรียงตัวตามรูปที่ 3.43 ซึ่งจะรับค่าตำแหน่งของการเขียนและอ่านขนาด 15 บิตเหล่านี้จากวงจรควบคุมการทำงานของหน่วยความจำชั่วคราวในแต่ละส่วนการประมวลผล ผ่านทางขา addra และ addrb ตามลำดับ สำหรับข้อมูลภาพ

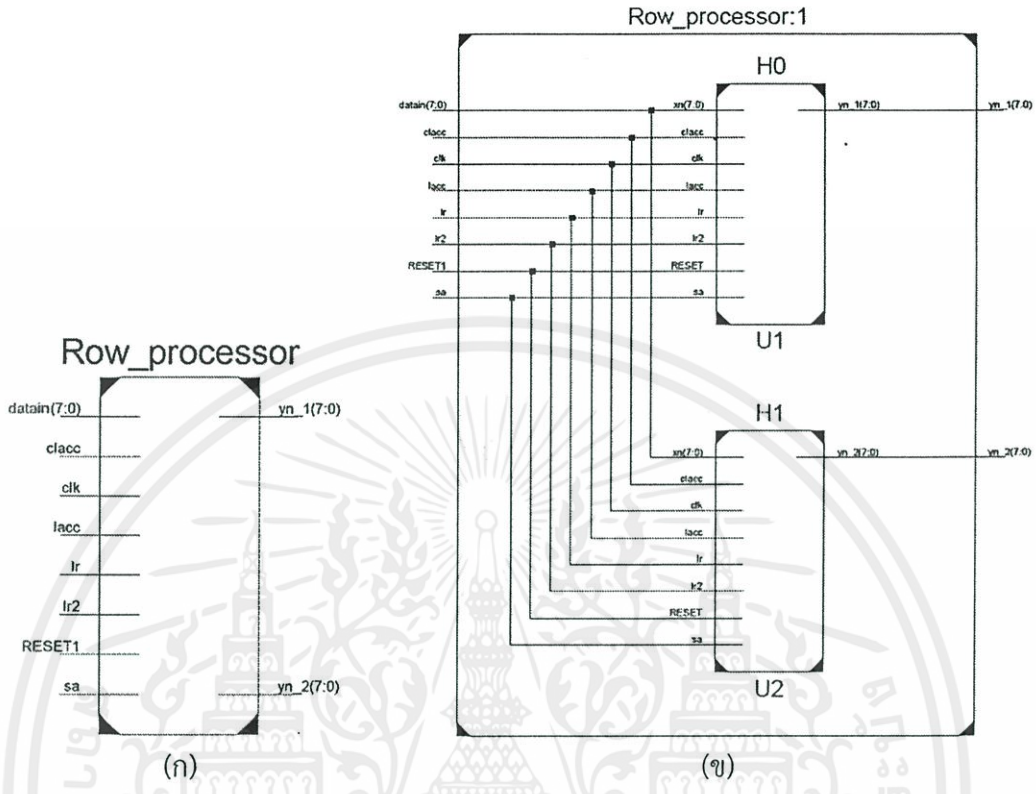
ขนาด 8 บิตรับเข้าทางขา dina และข้อมูลภาพที่อ่านขนาด 8 บิตจะถูกส่งออกทางขา doutb รวมทั้งรับสัญญาณควบคุมการทำงานได้แก่สัญญาณ wea หรือ weas จากวงจรสร้างสัญญาณควบคุม สัญญาณ lr และ lr2 ผ่านทางขา wea, clka และ clkb ตามลำดับ สามารถเขียนสัญลักษณ์แทนวงจรได้ดังภาพที่ 3.47




รูปที่ 3.47 สัญลักษณ์ของหน่วยความจำชั่วคราว

3.2.2.4 วงจรประมวลผลการคำนวณแถว (Row Processor)

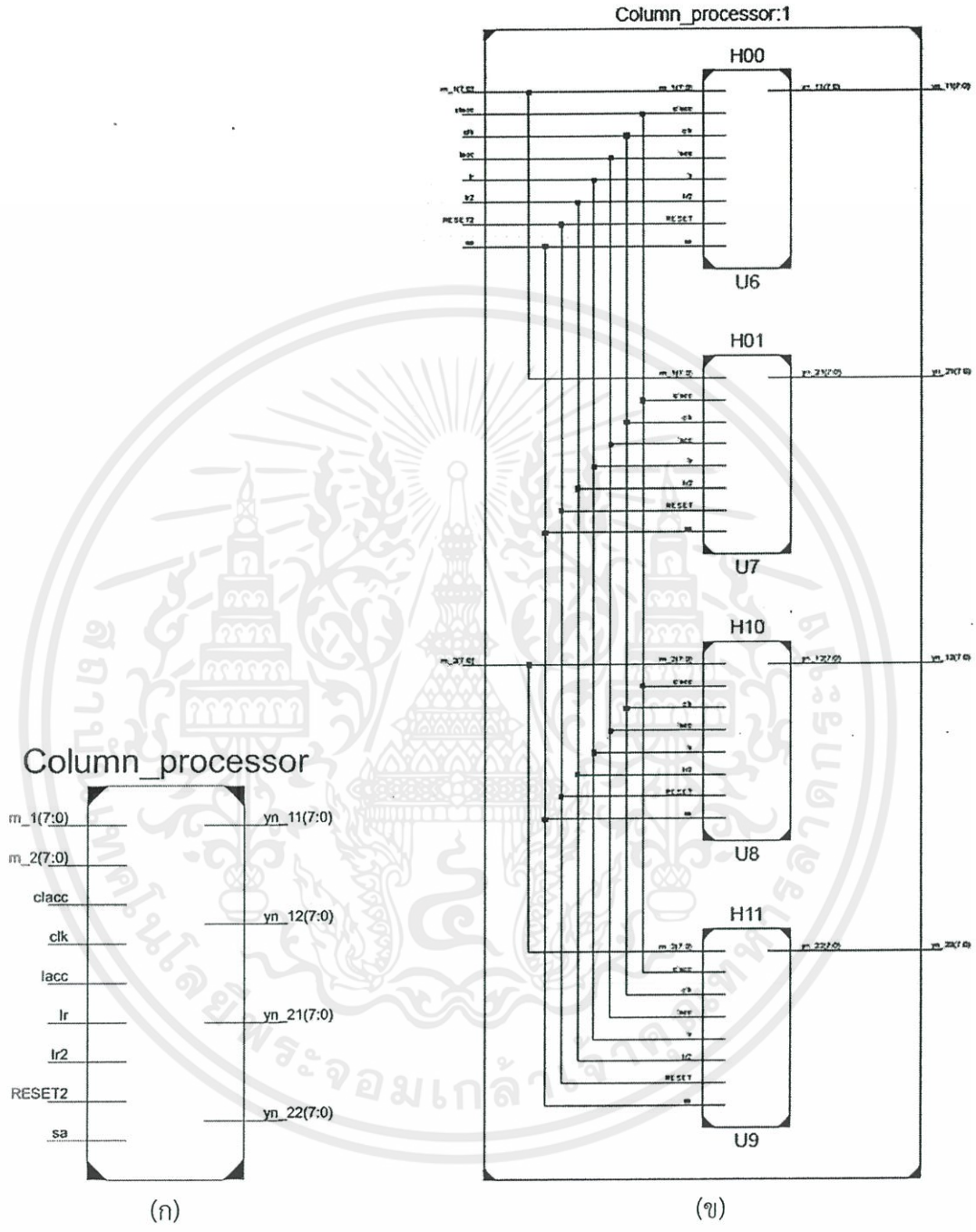
วงจรประมวลผลการคำนวณแถวเป็นวงจรที่ใช้ในการคำนวณในสเตจแรกของทั้งส่วนวิเคราะห์และสังเคราะห์ ซึ่งจะคำนวณโดยการรับค่าข้อมูลที่เก็บไว้ในหน่วยความจำทั้งสองชนิดด้วยการอ่านข้อมูลมาในลักษณะแถว วงจรจะบรรจุตัวกรอง $H_0(z)$, $H_1(z)$, $F_0(z)$ และ $F_1(z)$ แบ่งเป็น 2 ชุด อยู่ในส่วนวิเคราะห์ 1 ชุด ภายในประกอบด้วย $H_0(z)$ และ $H_1(z)$ และอยู่ในส่วนสังเคราะห์อีก 1 ชุด ภายในประกอบด้วย $F_0(z)$ และ $F_1(z)$ ลักษณะของตัวกรองออกแบบตามฟิลเตอร์แบงก์แบบ 1 มิติ ที่ใช้ค่าสัมประสิทธิ์ CDF 9/7 ตามตารางที่ 3.3 สามารถแสดงวงจรที่สังเคราะห์ได้เป็น Schematic Diagram ดังรูปที่ 3.48




รูปที่ 3.48 (ก) สัญลักษณ์ของวงจรประมวลผลการคำนวณแถว
(ข) Schematic Diagram ของวงจรประมวลผลการคำนวณแถว

3.2.2.5 วงจรประมวลผลการคำนวณคอลัมน์ (Column Processor)

วงจรประมวลผลการคำนวณคอลัมน์เป็นวงจรที่ใช้ในการคำนวณในสเตจที่ 2 ของทั้งส่วนวิเคราะห์และสังเคราะห์ ซึ่งจะคำนวณโดยการรับค่าข้อมูลที่เก็บไว้ในหน่วยความจำทั้งสองชนิดด้วยการอ่านข้อมูลมาในลักษณะคอลัมน์ วงจรจะบรรจุตัวกรอง $H_0(z)$, $H_1(z)$, $F_0(z)$ และ $F_1(z)$ แบ่งเป็น 2 ชุด อยู่ในส่วนวิเคราะห์ 1 ชุด ภายในประกอบด้วย $H_0(z)$ และ $H_1(z)$ จำนวน 2 ชุด และอยู่ในส่วนสังเคราะห์อีก 1 ชุดภายในประกอบด้วย $F_0(z)$ และ $F_1(z)$ จำนวน 2 ชุด ลักษณะของตัวกรองออกแบบตามฟิลเตอร์แบงก์แบบ 1 มิติ ที่ใช้ค่าสัมประสิทธิ์ CDF 9/7 ตามตารางที่ 3.3 สามารถแสดงวงจรที่สังเคราะห์ได้เป็น Schematic Diagram ดังรูปที่ 3.49

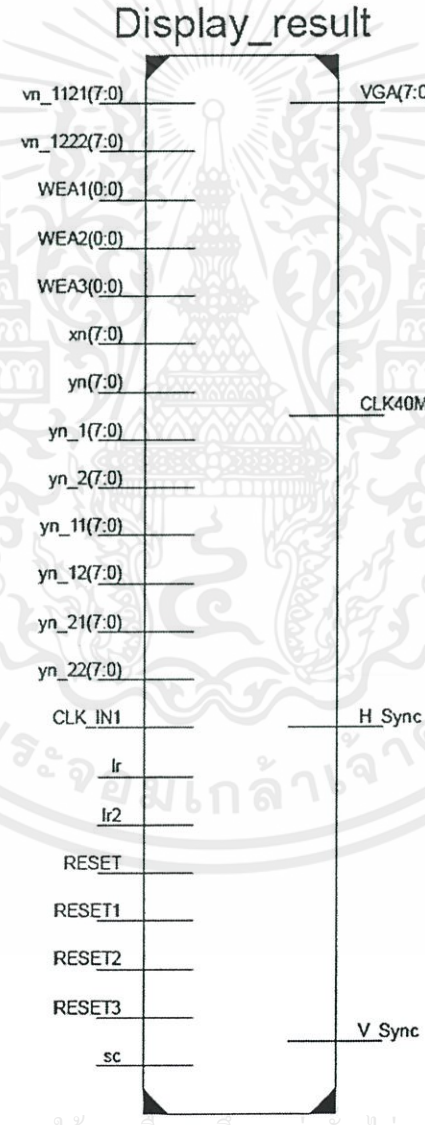



รูปที่ 3.49 (ก) สัญลักษณ์ของวงจรประมวลผลการคำนวณคอลัมน์

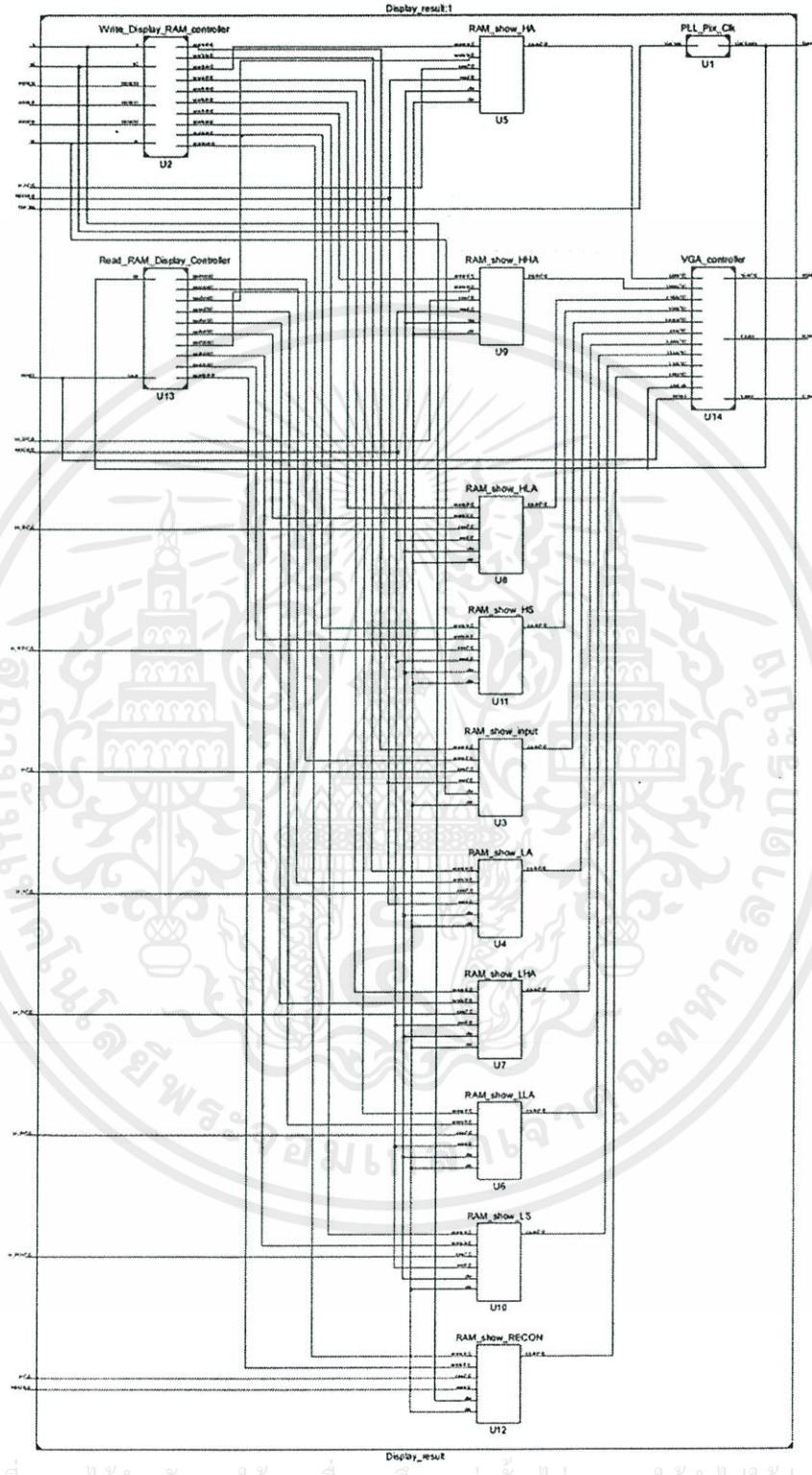
(ข) Schematic Diagram ของวงจรประมวลผลการคำนวณคอลัมน์

### 3.2.3 การออกแบบวงจรควบคุมการแสดงผลข้อมูล (Display Controller)

วงจรควบคุมการแสดงผลข้อมูลที่ออกแบบจะมี 4 ส่วนหลักได้แก่ วงจรสร้างสัญญาณนาฬิกาพิกเซล (Pixel Clock Generator) หน่วยความจำชั่วคราวจำนวน 10 ตัวที่กล่าวในหัวข้อข้างต้น วงจรควบคุมการเขียนและอ่านหน่วยความจำชั่วคราว และวงจรควบคุมการแสดงผลบนจอแสดงผลวีจีเอ (VGA Controller) สามารถเขียนแทนสัญลักษณ์ได้ดังรูปที่ 3.50 และแสดงวงจรที่สังเคราะห์ได้เป็น Schematic Diagram ดังรูปที่ 3.51




รูปที่ 3.50 สัญลักษณ์ของวงจรควบคุมการแสดงผลข้อมูล






รูปที่ 3.51 Schematic Diagram ของวงจรควบคุมการแสดงผลข้อมูล

หลักการทำงานของวงจรควบคุมการแสดงผลข้อมูลจะทำหน้าที่ในการบันทึกผลที่ได้ระหว่างขั้นตอนการคำนวณโดยรับข้อมูลในจุดที่ต้องการจะศึกษาเข้ามาจัดเก็บในหน่วยความจำชั่วคราวภายในวงจรควบคุมการแสดงผลข้อมูล ซึ่งจะมีการควบคุมการเขียนและอ่านข้อมูลด้วยวงจรควบคุมการเขียนและอ่านหน่วยความจำชั่วคราวให้สัมพันธ์กับการทำงานกับฟิลเตอร์แบงก์แบบ 2 มิติ แล้วจึงนำไปแสดงผลบนจอแสดงผลวีจีเอที่ถูกควบคุมด้วยวงจรควบคุมการแสดงผลบนจอแสดงผลวีจีเอ โดยทั้งหมดนี้จะมีรายละเอียดการออกแบบแต่ละวงจรดังนี้

3.2.3.1 วงจรสร้างสัญญาณนาฬิกาพิกเซล (Pixel Clock Generator)

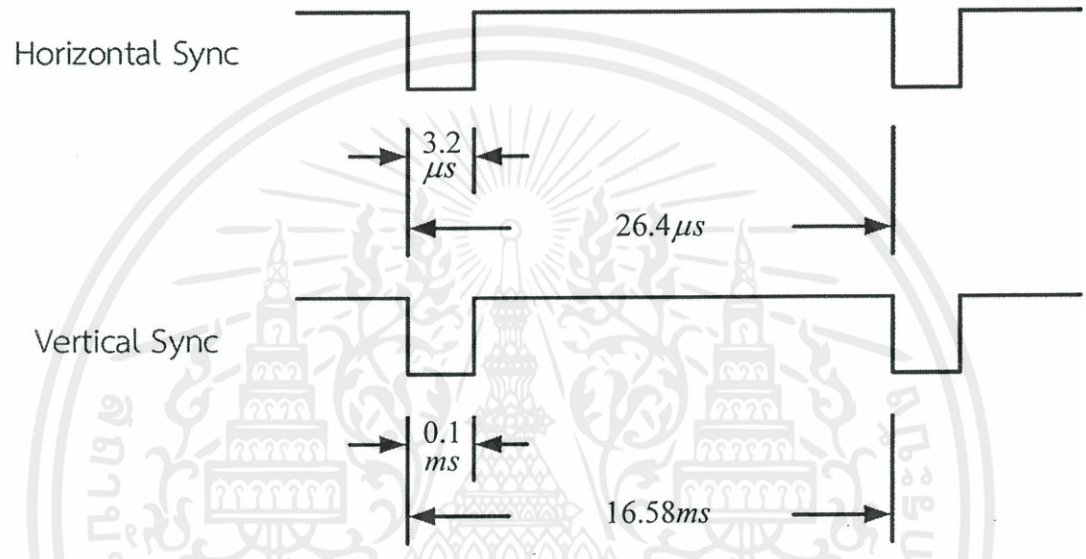
วงจรสร้างสัญญาณนาฬิกาพิกเซลเป็นวงจรที่ใช้ในการสร้างสัญญาณที่ใช้นับจำนวนพิกเซลตามมาตรฐานการแสดงผล สำหรับในปริญญานิพนธ์นี้เลือกที่จะแสดงผลภาพขนาด 800 แถว 600 คอลัมน์ ที่ 60 Hz ดังนั้นค่าความถี่สัญญาณนาฬิกาพิกเซลที่ต้องใช้มีค่า 40 MHz ที่ส่งออกทางขา clk_out1 โดยเกิดจากการหารความถี่สัญญาณของออสซิลเลเตอร์บนบอร์ดที่มีความถี่ 100 MHz ซึ่งได้รับผ่านทางขา clk_in1 สามารถเขียนแทนสัญลักษณ์ได้ดังรูปที่ 3.52


รูปที่ 3.52 สัญลักษณ์ของวงจรสร้างสัญญาณนาฬิกาพิกเซล

3.2.3.2 วงจรควบคุมการแสดงผลบนจอแสดงผลวีจีเอ (VGA Controller)

วงจรควบคุมการแสดงผลบนจอแสดงผลวีจีเอเป็นวงจรที่จะสร้างสัญญาณควบคุมจอแสดงผลโดยจะสร้างสัญญาณทั้งหมด 5 ชนิดคือสัญญาณ HSync (Horizontal Sync) เป็นสัญญาณซิ้งค์ที่ควบคุมการกวาดภาพในแนวนอน สัญญาณ VSync (Vertical Sync) เป็นสัญญาณซิ้งค์ที่ควบคุมการกวาดภาพในแนวตั้ง สัญญาณ VGA_Red เป็นสัญญาณข้อมูลภาพในกลุ่มสีแดง สัญญาณ VGA_Green เป็นสัญญาณข้อมูลภาพในกลุ่มสีเขียว และสัญญาณ VGA_Blue เป็นสัญญาณข้อมูลภาพในกลุ่มสีน้ำเงิน ซึ่งสัญญาณทั้งหมดจะถูกส่งออกทางพอร์ตเชื่อมต่อ DE-15 ที่เป็นพอร์ตวิดีโอสำหรับต่อสายสัญญาณภาพกับจอแสดงผลโดยลักษณะพอร์ตที่จะส่งสัญญาณออกจะอยู่ในภาคผนวก ค

ในการสร้างสัญญาณควบคุม HSync และ VSync จะอาศัยวงจรสร้างสัญญาณนาฬิกาพิกเซลที่สร้างสัญญาณ Pixel Clock ที่ความถี่ 40 MHz ทำการนับช่วงเวลาของการซิ้งค์จากมาตรฐานของสัญญาณซิ้งค์สำหรับภาพขนาด 800 แถว 600 คอลัมน์ เพื่อใช้เป็นตัวชี้ตำแหน่งในหน่วยความจำชั่วคราว ดังแสดงรูปที่ 3.53



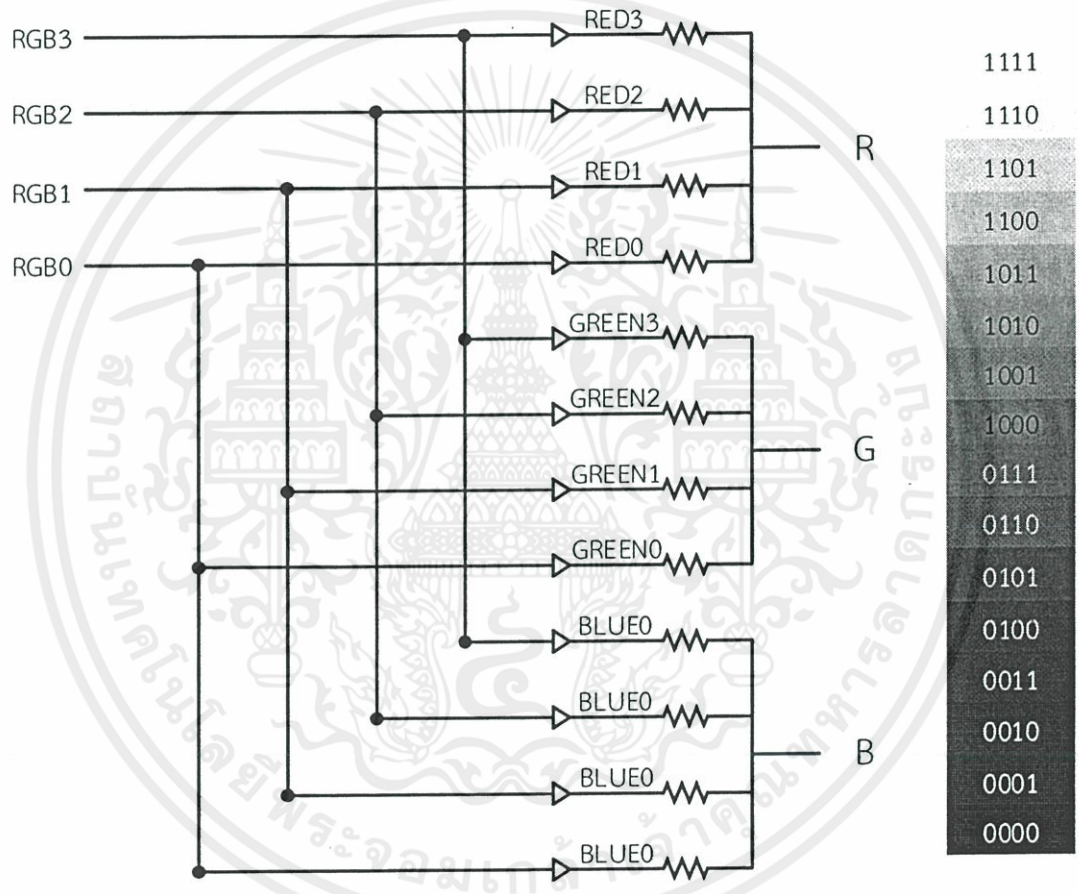

รูปที่ 3.53 ช่วงเวลาของสัญญาณซิ้งค์

จากรูปที่ 3.53 เราสามารถแจกแจงช่วงเวลาของสัญญาณและการนับของสัญญาณ HSync ที่มีความถี่ 37.88 Hz และ VSync ที่มีความถี่ 60.31 Hz ได้ตามตารางที่ 3.4

ตารางที่ 3.4 ช่วงเวลาของสัญญาณซิ้งค์และการนับ

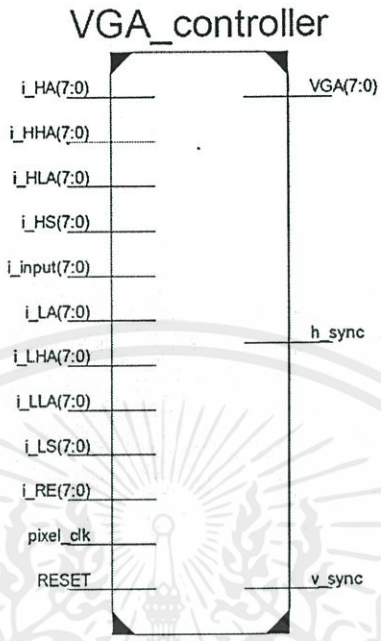
| สัญญาณ | ช่วงเวลา | การนับ |
|---|---|---|
| Horizontal Scan Time | 26.4 $\mu s$ | 1056 |
| Horizontal Sync Pulse | 3.2 $\mu s$ | 128 |
| Vertical Scan Time | 16.58 $ms$ | 663200 |
| Vertical Sync Pulse | 0.1 $ms$ | 4000 |

สำหรับสัญญาณ VGA_Red, VGA_Green และ VGA_Blue ในปริญญานิพนธ์นี้เนื่องจากใช้ข้อมูลภาพอินพุตเป็นภาพสเกลสีเทา (Grayscale) ดังนั้นในการออกแบบจึงต้องทำการเชื่อมบิตข้อมูลทั้งสามสีแต่ละบิตที่ตำแหน่งเดียวเข้าด้วยกัน ดังแสดงตามรูปที่ 3.54 ก่อนส่งออกไปแสดงผล ซึ่งอุปกรณ์ FPGA ที่ใช้จะส่งข้อมูลภาพออกด้วยจำนวนสีละ 4 บิต รวมเป็น 12 บิตต่อการส่ง 1 ครั้ง



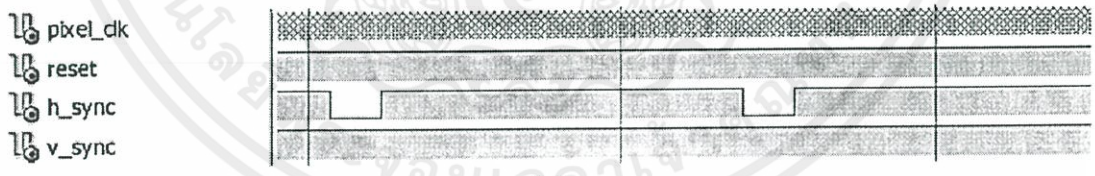

รูปที่ 3.54 ลักษณะการเชื่อมต่อบิตข้อมูล RGB ให้เป็นข้อมูลสเกลสีเทา

จากการออกแบบข้างต้นทำให้ได้วงจรควบคุมการแสดงผลบนจอแสดงผลวีจีเอ ที่มีสัญลักษณ์แทนตามรูปที่ 3.55 โดยจะรับค่าสัญญาณเอาต์พุตที่ประมวลผลภาพแล้วเก็บไว้ในหน่วยความจำชั่วคราว ซึ่งทำงานตามสัญญาณ pixel_clk

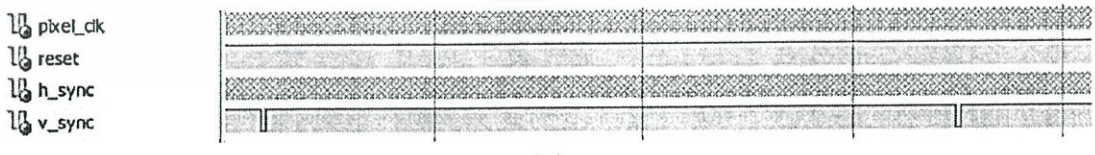



รูปที่ 3.55 สัญลักษณ์ของวงจรควบคุมการแสดงผลบนจอแสดงผลวีจีเอ

จากรูปที่ 3.55 วงจรจะรับข้อมูลภาพจำนวน 8 บิตจากหน่วยความจำชั่วคราวผ่านทางขา i_HA, i_HHA, i_HLA, i_HS, i_input, i_LA, i_LHA, i_LLA, i_LS และ i_RE แล้วส่งออกทางขา VGA มีผลการจำลองการสร้างสัญญาณซิ้งค์ได้ดังรูปที่ 3.56




(ก)

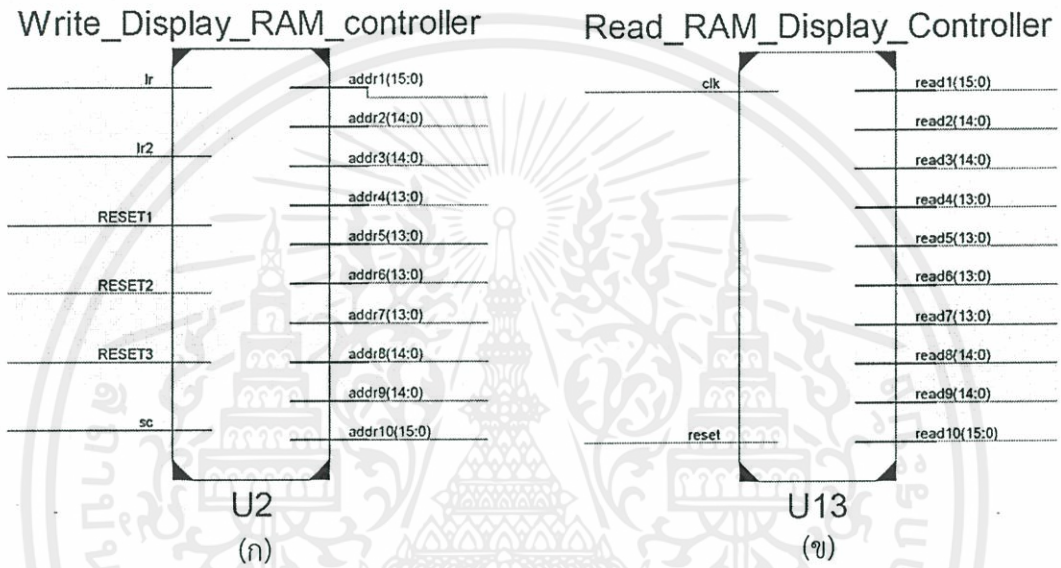



(ข)

รูปที่ 3.56 ผลการจำลองการสร้างสัญญาณซิ้งค์ (ก) สัญญาณควบคุม HSync
(ข) สัญญาณควบคุม VSync

3.2.3.3 วงจรควบคุมการเขียนและอ่านหน่วยความจำชั่วคราว

วงจรควบคุมการเขียนและอ่านหน่วยความจำชั่วคราวเป็นวงจรที่ใช้ควบคุมการเริ่มต้นและหยุดการเขียนและอ่านหน่วยความจำชั่วคราวภายในวงจรควบคุมการแสดงผลข้อมูล สามารถเขียนสัญลักษณ์แทนได้ดังภาพที่ 3.57




รูปที่ 3.57 (ก) สัญลักษณ์ของวงจรควบคุมการเขียนหน่วยความจำชั่วคราว
(ข) สัญลักษณ์ของวงจรควบคุมการอ่านหน่วยความจำชั่วคราว

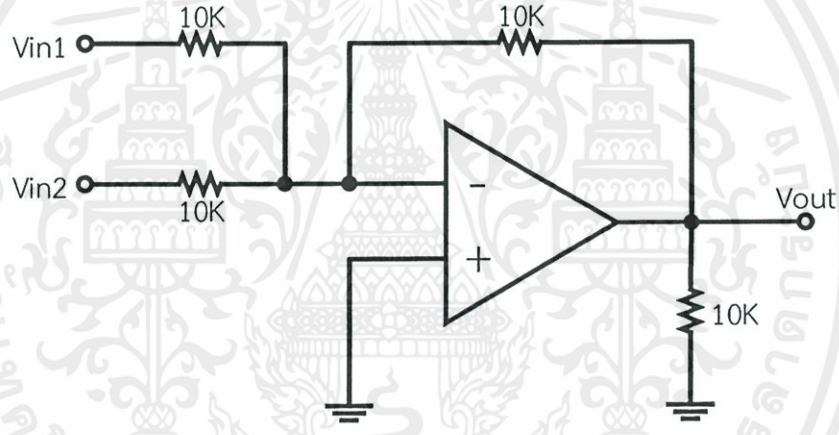
จากรูปที่ 3.57 วงจรควบคุมการเขียนหน่วยความจำชั่วคราวจะทำงานตามจังหวะสัญญาณ lr, lr2, reset1, reset2, reset3 และ sc โดยส่งสัญญาณค่าตำแหน่งไปชี้ตำแหน่งที่ต้องการเก็บในหน่วยความจำชั่วคราวแต่ละตัว แต่วงจรควบคุมการอ่านหน่วยความจำชั่วคราวจะทำงานตามสัญญาณ reset และ clk ที่ทำงานที่ความถี่ 40 MHz โดยส่งสัญญาณค่าตำแหน่งไปชี้ตำแหน่งที่ต้องการอ่านในหน่วยความจำชั่วคราวแต่ละตัว

## 3.3 อุปกรณ์ที่ใช้ในการทดลองฟิลเตอร์แบงก์

ขั้นตอนการออกแบบข้างต้นเป็นการออกแบบโดยใช้ภาษา VHDL และจำลองการทำงาน ในส่วนต่อไปจะเป็นอุปกรณ์ที่ใช้ในการทดลองร่วมกับฟิลเตอร์แบงก์ที่ถูกสร้างเป็นฮาร์ดแวร์ ประกอบด้วย 4 ส่วนหลักดังนี้

### 3.3.1 วงจรขยายรวมสัญญาณ (Summing Amplifier)

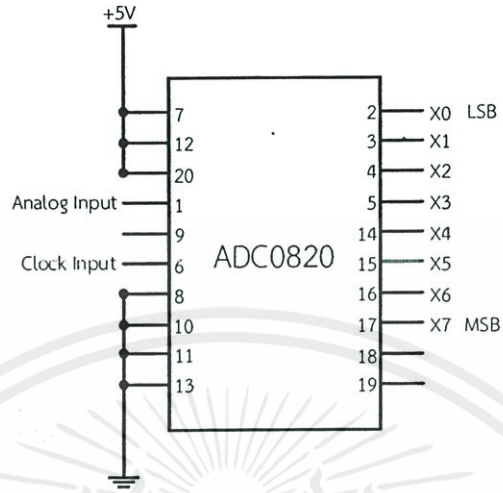
วงจรขยายรวมสัญญาณเป็นวงจรรวมสัญญาณเพื่อใช้เป็นสัญญาณอินพุตให้กับฟิลเตอร์แบงก์แบบ 1 มิติ สามารถแสดงวงจรได้ดังรูปที่ 3.58




รูปที่ 3.58 วงจรขยายรวมสัญญาณ

### 3.3.2 วงจรแปลงสัญญาณเชิงอุปมานเป็นสัญญาณเชิงเลข (Analog to Digital Converter)

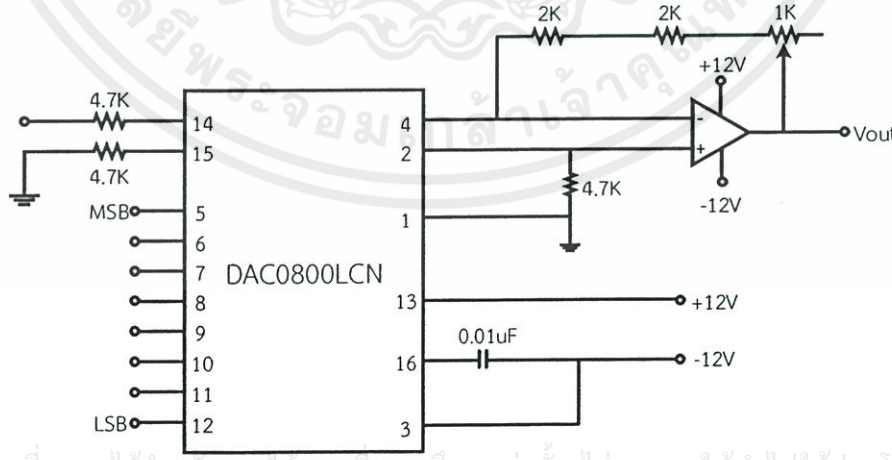
วงจรแปลงสัญญาณเชิงอุปมานเป็นสัญญาณเชิงเลขเป็นอุปกรณ์ที่แปลงสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอล ในปริญญานิพนธ์นี้จะเลือกใช้ไอซี ADC0820 ซึ่งเมื่อแปลงข้อมูลอนาล็อกแล้วจะได้ข้อมูลเป็นดิจิตอล 8 บิต เพื่อป้อนเข้าสู่ฟิลเตอร์แบงก์แบบ 1 มิติ โดยมีวงจรแสดงดังรูปที่ 3.59




รูปที่ 3.59 วงจรแปลงสัญญาณเชิงอุปมานเป็นสัญญาณเชิงเลข

**3.3.3 วงจรแปลงสัญญาณเชิงเลขเป็นสัญญาณเชิงอุปมาน (Digital to Analog Converter)**

วงจรแปลงสัญญาณเชิงเลขเป็นสัญญาณเชิงอุปมานเป็นอุปกรณ์ที่แปลงสัญญาณดิจิตอลให้เป็นสัญญาณอนาล็อก ในปริญญานิพนธ์นี้จะเลือกใช้ไอซี DAC0800LCN ซึ่งจะรับข้อมูลดิจิตอลจากฟิลเตอร์แบงก์แบบ 1 มิติ มาเป็นข้อมูล 8 บิตเพื่อนำมาแปลงเป็นสัญญาณอนาล็อก โดยมีวงจรแสดงดังรูปที่ 3.60

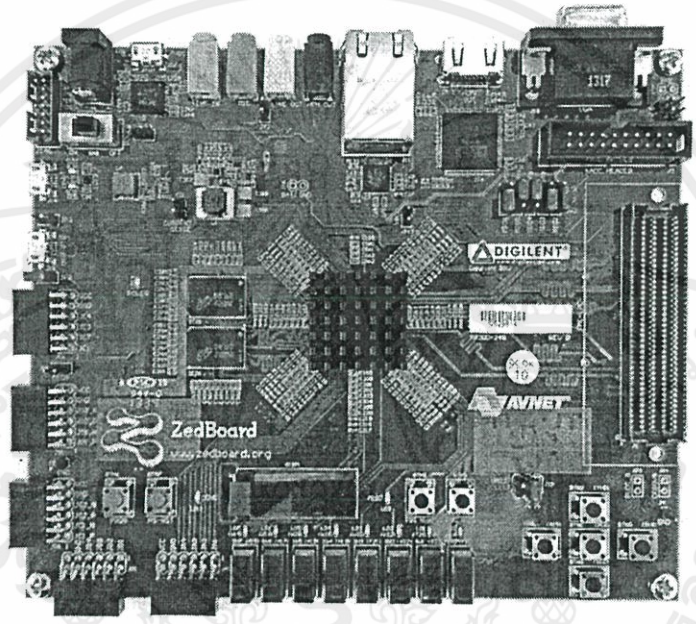





รูปที่ 3.60 วงจรแปลงสัญญาณเชิงเลขเป็นสัญญาณเชิงอุปมาน

### 3.3.4 อุปกรณ์ FPGA

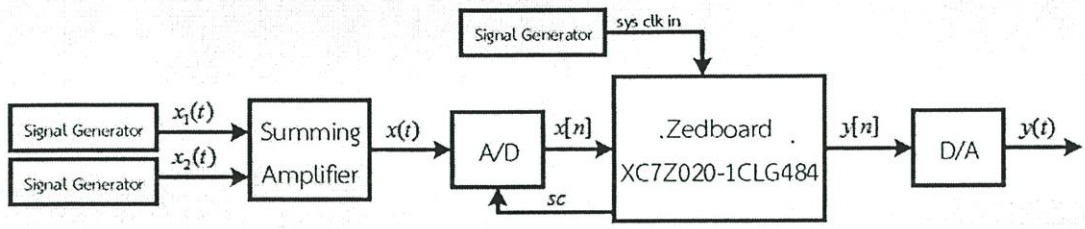
อุปกรณ์ FPGA ในปริญญานิพนธ์นี้จะเลือกใช้ของบริษัท Xilinx ตระกูล Zynq Zedboard เบอร์ XC7Z020-1CLG484 โดยใช้สร้างฟิลเตอร์แบงก์เป็นฮาร์ดแวร์ซึ่งเป็นการเขียนภาษา VHDL จากที่ออกแบบในหัวข้อก่อนหน้าลงบน FPGA เพื่อให้ทำงานเป็นฟิลเตอร์แบงก์ทั้งแบบ 1 มิติ และ 2 มิติ ตามต้องการ ลักษณะของอุปกรณ์ FPGA มีรูปแบบตามรูปที่ 3.61



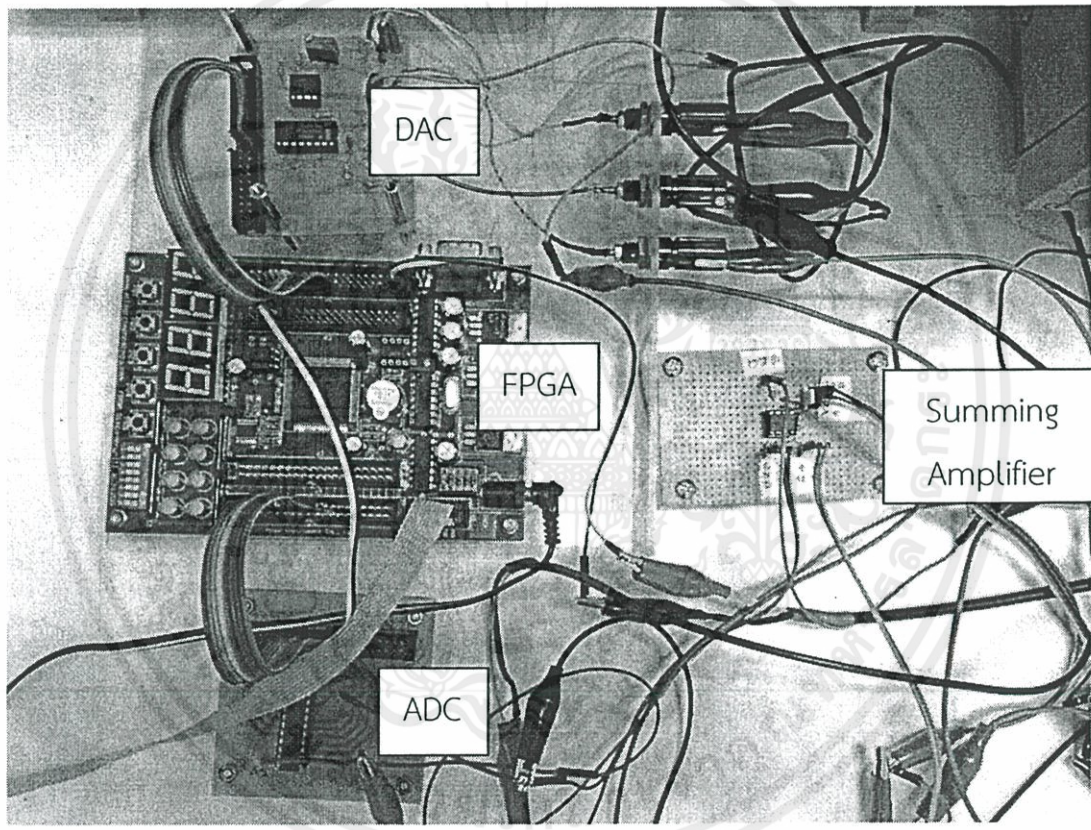

รูปที่ 3.61 อุปกรณ์ FPGA

## 3.4 การจัดเก็บผลการทดลองฟิลเตอร์แบงก์

ในส่วนนี้จะเริ่มจากการนำอุปกรณ์ทั้งหมดมาเชื่อมต่อเข้าด้วยกัน สำหรับฟิลเตอร์แบงก์แบบ 1 มิติสามารถแสดงเป็นบล็อกไดอะแกรมได้ดังรูปที่ 3.62 โดยจะมีการต่อสัญญาณควบคุม sc จากอุปกรณ์ FPGA มาเป็นสัญญาณสุ่มตัวอย่างให้กับวงจรแปลงสัญญาณเชิงอุปมานเป็นสัญญาณเชิงเลข และรับสัญญาณอินพุตจากฟังก์ชันเจเนอเรเตอร์ 2 ตัว ซึ่งจะถูกรวมกันก่อนด้วยวงจรขยายรวมสัญญาณแล้วแปลงเป็นสัญญาณเชิงเลขด้วยวงจรแปลงสัญญาณเชิงอุปมานเป็นสัญญาณเชิงเลขก่อนเข้า FPGA และสัญญาณเอาต์พุตจะถูกแปลงเป็นสัญญาณอนาล็อกด้วยวงจรแปลงสัญญาณเชิงเลขเป็นสัญญาณเชิงอุปมานแล้วจึงเก็บผล แสดงอุปกรณ์ต้นแบบได้ตามรูปที่ 3.63

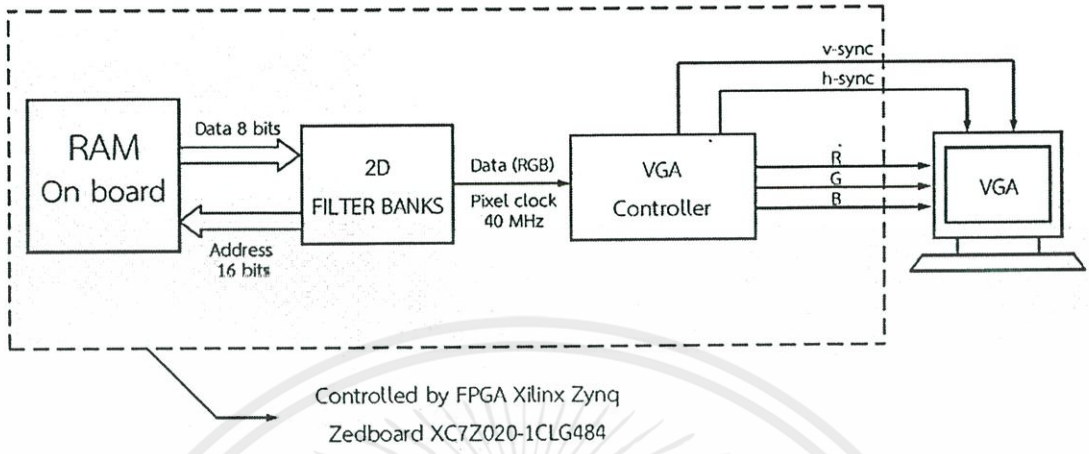



รูปที่ 3.62 บล็อกไดอะแกรมของการทดสอบฟิลเตอร์แบงก์แบบ 1 มิติ

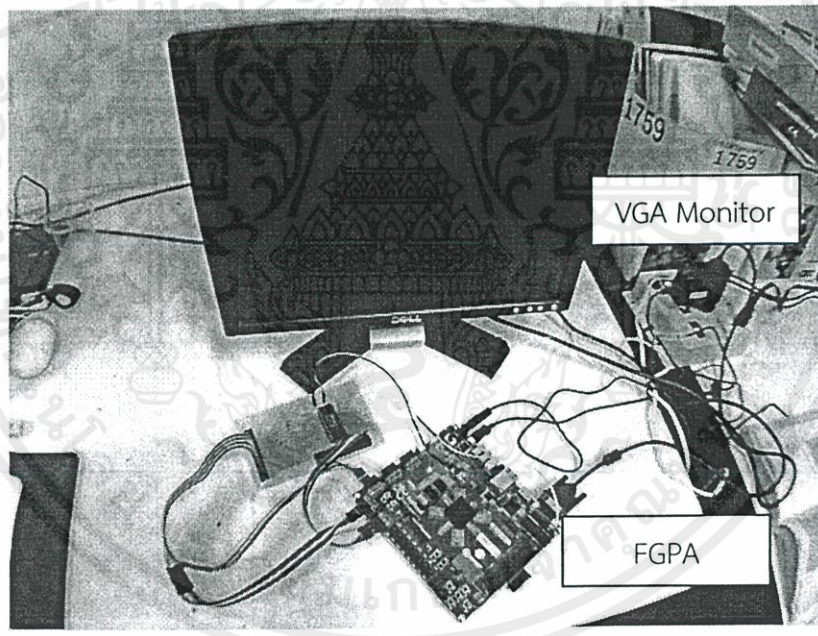



รูปที่ 3.63 อุปกรณ์ต้นแบบฟิลเตอร์แบงก์แบบ 1 มิติ

สำหรับฟิลเตอร์แบงก์แบบ 2 มิติสามารถแสดงเป็นบล็อกไดอะแกรมได้ดังรูปที่ 3.64 จะทำการเรียกข้อมูลที่เก็บไว้ในหน่วยความจำบนบอร์ดนำไปประมวลผล แล้วจะส่งภาพเอาต์พุตพร้อมสัญญาณควบคุมไปแสดงผลบนจอแสดงวีจีเอที่ทำการต่อสายสัญญาณภาพกับอุปกรณ์ FPGA ไว้ สามารถแสดงอุปกรณ์ต้นแบบได้ตามรูปที่ 3.65




รูปที่ 3.64 บล็อกไดอะแกรมของการทดสอบฟิลเตอร์แบงก์แบบ 2 มิติ




รูปที่ 3.65 อุปกรณ์ต้นแบบฟิลเตอร์แบงก์แบบ 2 มิติ

ในส่วนการจัดเก็บผลการทดลองจะถูกแบ่งออกเป็น 4 ส่วนตามการทดลองคือการทดลองโดยใช้การจำลองการทำงานของฟิลเตอร์แบงก์ตามทฤษฎี (Software Simulation) ผ่านโปรแกรมแมตแล็บ การทดลองการจำลองการทำงานของวงจรย่อยภายในฟิลเตอร์แบงก์ที่ออกแบบ

ด้วยภาษา VHDL ผ่านโปรแกรม ISim 14.6 การทดลองการจำลองการทำงานของฮาร์ดแวร์ (Hardware Simulation) และการทดลองการทำงานจริงของฟิลเตอร์แบงก์บน FPGA

**3.4.1 การทดลองการจำลองการทำงานของฟิลเตอร์แบงก์ตามทฤษฎี (Software Simulation) ผ่านโปรแกรมแมตแล็บ**

ในส่วนของการทดลองฟิลเตอร์แบงก์แบบ 1 มิติ จะทำโดยการป้อนสัญญาณอินพุตที่สร้างเป็นสัญญาณคลื่นรูปไซน์ซึ่งเป็นการรวมคลื่นรูปไซน์ 2 ลูกเข้าด้วยกัน โดยมีความถี่ 2 ค่าคือ ความถี่ต่ำอยู่ที่ 200 Hz และความถี่สูงอยู่ที่ 8 kHz เข้าสู่ฟิลเตอร์แบงก์ที่ออกแบบไว้ แล้วทำการบันทึกผลเพื่อใช้เปรียบเทียบแต่ละจุดที่ต้องการศึกษาในส่วนของการทดลองจริง โดยโค้ดที่ใช้ในการทดลองนี้จะอยู่ในภาคผนวก ก

สำหรับการทดลองฟิลเตอร์แบงก์แบบ 2 มิติ จะทำโดยการป้อนสัญญาณอินพุตเป็นข้อมูลภาพแบบสเกลสีเทา (Grayscale) ขนาด 256 แถว 256 คอลัมน์ ที่เป็นรูปทดสอบตามมาตรฐานด้านการประมวลผลสัญญาณภาพ ชื่อภาพคือ lena_gray_256.tif เข้าสู่ฟิลเตอร์แบงก์แบบ 2 มิติที่ออกแบบไว้ แล้วทำการบันทึกผลแต่ละจุดที่ต้องการศึกษาเพื่อใช้เปรียบเทียบกับการจำลองการทำงานของฮาร์ดแวร์ผ่านโปรแกรมแมตแล็บ และผลการทำงานจริงของฟิลเตอร์แบงก์ โดยโค้ดที่ใช้ในการทดลองนี้จะอยู่ในภาคผนวก ก

**3.4.2 การทดลองการจำลองการทำงานของวงจรย่อยภายในฟิลเตอร์แบงก์ที่ออกแบบด้วยภาษา VHDL ผ่านโปรแกรม ISim 14.6**

การออกแบบฟิลเตอร์แบงก์ด้วยภาษา VHDL ในหัวข้อ 3.2 นำโค้ดบรรยายพฤติกรรมของวงจรที่ทำในโปรแกรม Xilinx ISE Design Suite 14.6 มาทำการจำลองการทำงานผ่านโปรแกรม ISim 14.6 โดยใช้การสร้างโค้ด VHDL Test Bench ทำการทดลองในแต่ละวงจรย่อย แล้วทำการบันทึกผลเพื่อตรวจสอบความถูกต้องในการทำงานของแต่ละส่วนย่อยก่อนนำไปทำเป็นวงจรรวมจริงบน FPGA ซึ่งโค้ดที่ใช้ในการบรรยายพฤติกรรมและที่ใช้ในการจำลองการทำงานบางส่วนจะอยู่ในภาคผนวก ข

### 3.4.3 การทดลองการจำลองการทำงานของฮาร์ดแวร์ (Hardware Simulation)

ในส่วนของการทดลองฟิลเตอร์แบงก์แบบ 1 มิติ นำโค้ดบรรยายพฤติกรรมของวงจรมาทำการจำลองการทำงานผ่านโปรแกรม ModelSim SE 6.5 โดยใช้การสร้างโค้ด VHDL Test Bench ทำการทดลองด้วยการป้อนสัญญาณอินพุตที่เหมือนกับในส่วนของการจำลองการทำงานผ่านโปรแกรมแมตแล็บคือเป็นสัญญาณคลื่นรูปไซน์ 2 ความถี่ ที่มีความถี่เดียวกัน แล้วทำการบันทึกผลเพื่อใช้เปรียบเทียบในส่วนการทำงานจริง ซึ่งโค้ดที่ใช้ในการบรรยายพฤติกรรมและที่ใช้ในการจำลองการทำงานบางส่วนจะอยู่ในภาคผนวก ข

สำหรับการทดลองฟิลเตอร์แบงก์แบบ 2 มิติ จะจำลองการทำงานของฮาร์ดแวร์ผ่านโปรแกรมแมตแล็บ ทดสอบเพื่อหาข้อจำกัดของฮาร์ดแวร์ (Hardware Limitation) โดยการสร้างฟังก์ชั่นในการปรับค่าความละเอียดของการควอนไตซ์ (Quantization) ที่ทำให้เกิดผลกระทบของความยาวคำจำกัด (Finite Word Length Effect) และเพื่อจำลองค่าความผิดพลาดของการควอนไตซ์ (Quantization Error) ในการคำนวณแต่ละขั้นตอน ด้วยการนำฟังก์ชันนี้ไปคั้นกลางระหว่างการคำนวณทุกขั้นตอน แล้วทำการบันทึกผลในแต่ละจุดที่ต้องการศึกษา จากนั้นนำไปเปรียบเทียบกับผลการทดลองจริงที่ได้ ซึ่งฟังก์ชันที่ใช้ในการจำลองส่วนนี้จะอยู่ในส่วนภาคผนวก ก

### 3.4.4 การทดลองการทำงานจริงของฟิลเตอร์แบงก์บน FPGA

การทดลองในส่วนนี้จะนำโค้ดบรรยายพฤติกรรมฟิลเตอร์แบงก์ที่ทดสอบการจำลองการทำงานผ่านแล้วมาเขียนลงบน FPGA ซึ่งต้องมีการกำหนดพอร์ตอินพุตและเอาต์พุตให้กับวงจรก่อนด้วยโปรแกรม PlanAhead 14.5 แล้วจึงทำการเขียนลงบน FPGA ด้วยโปรแกรม iMPACT ผ่านพอร์ต JTAG (J17) สำหรับแผนผังวงจรภายในของ FPGA ที่แสดงลักษณะพอร์ตการใช้งานจะอยู่ในภาคผนวก ค เมื่อฟิลเตอร์แบงก์พร้อมใช้งานแล้วทำการต่อรวมอุปกรณ์ต่างๆ ตามรูปที่ 3.63 และ 3.64 แล้วจึงทำการบันทึกผลเพื่อใช้เปรียบเทียบกับผลการจำลองการทำงานของฟิลเตอร์แบงก์จากโปรแกรมแมตแล็บ

# บทที่ 4

## ผลการทดลอง

จากโครงสร้างฟิลเตอร์แบงก์ที่นำเสนอไว้หลังจากทำการออกแบบฮาร์ดแวร์ด้วยภาษา VHDL และสังเคราะห์ลงบน FPGA ในบทนี้จะแสดงผลการทดสอบวงจร คือผลการสังเคราะห์วงจร ผลการทดลองของฟิลเตอร์แบงก์แบบ 1 มิติ และผลการทดลองของฟิลเตอร์แบงก์แบบ 2 มิติ โดยจะเปรียบเทียบผลการทำงานจริงกับผลการจำลองการทำงานตามทฤษฎีบนโปรแกรมแมตแล็บ และผลการจำลองการทำงานของฮาร์ดแวร์ นอกจากนี้ยังมีผลการจำลองฟิลเตอร์แบงก์หลายสเตจ ซึ่งมีผลการทดลองดังนี้

### 4.1 ผลการสังเคราะห์วงจร

จากการออกแบบและเขียนโปรแกรมภาษา VHDL เมื่อสังเคราะห์ฟิลเตอร์แบงก์แล้วสามารถตรวจสอบคุณสมบัติทางด้านฮาร์ดแวร์ เช่นการใช้ทรัพยากรในการสังเคราะห์วงจร ความถี่สัญญาณนาฬิกาสูงสุดที่วงจรยังสามารถทำงานได้ ซึ่งคุณสมบัติเหล่านี้เป็นตัวบ่งบอกประสิทธิภาพของวงจรที่ออกแบบ โดยผลของการสังเคราะห์โครงสร้างทางฮาร์ดแวร์ของฟิลเตอร์แบงก์ทั้งสองแบบที่ออกแบบด้วยโปรแกรม Xilinx ISE Design Suite 14.6 บนอุปกรณ์ Xilinx Zynq Zedboard XC7Z020-1CLG484 จะแสดงผลการสังเคราะห์ดังรูปที่ 4.1

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 448 | 106400 | 0% |
| Number of Slice LUTs | 884 | 53200 | 1% |
| Number of fully used LUT-FF pairs | 315 | 1017 | 30% |
| Number of bonded IOBs | 127 | 200 | 63% |
| Number of BUFG/BUFGCTRLs | 4 | 32 | 12% |

(ก)

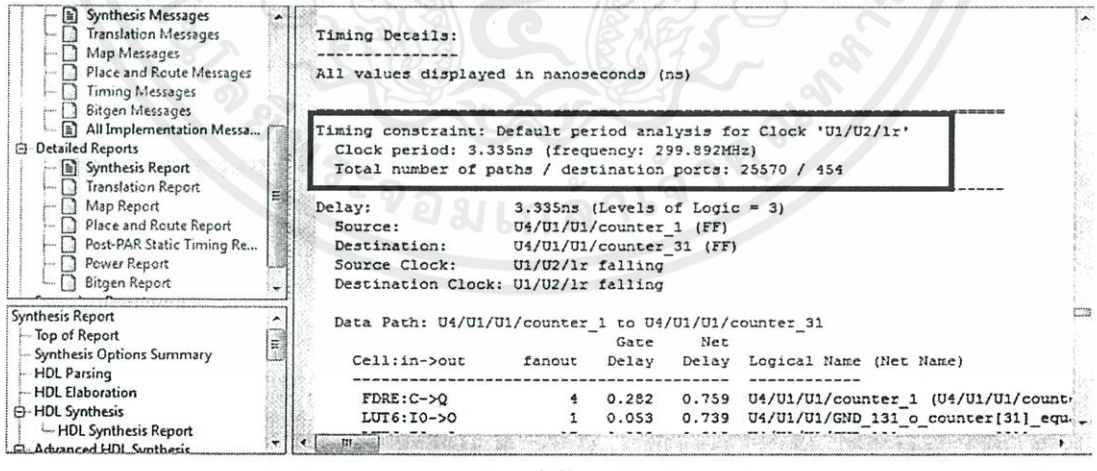
| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 1263 | 106400 | 1% |
| Number of Slice LUTs | 2150 | 53200 | 4% |
| Number of fully used LUT-FF pairs | 1094 | 2319 | 47% |
| Number of bonded IOBs | 29 | 200 | 14% |
| Number of Block RAM/FIFO | 128 | 140 | 91% |
| Number of BUFG/BUFGCTRLs | 11 | 32 | 34% |

(ข)

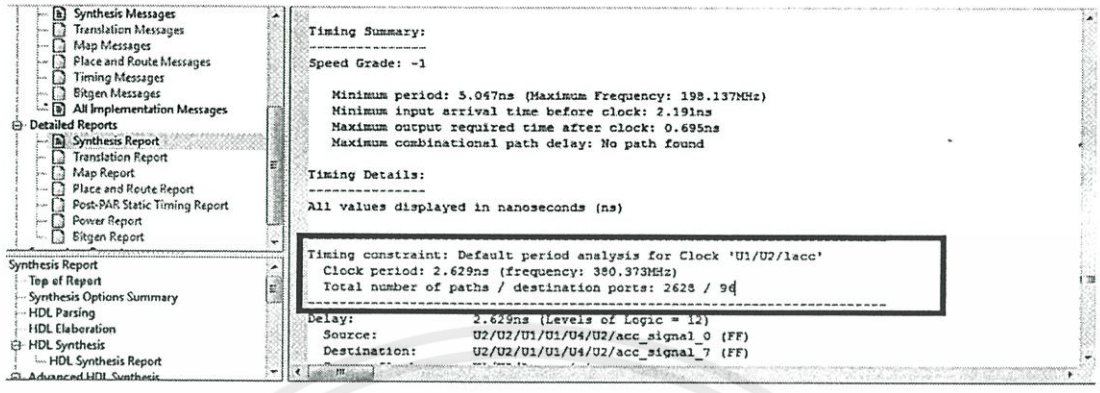
รูปที่ 4.1 ผลการสังเคราะห์วงจรบนโปรแกรม Xilinx ISE Design Suite 14.6
(ก) ฟิลเตอร์แบงก์แบบ 1 มิติ (ข) ฟิลเตอร์แบงก์แบบ 2 มิติ

จากรูปที่ 4.1 จะพบว่าการสังเคราะห์ฟิลเตอร์แบงก์ทั้งสองแบบใช้ทรัพยากรไม่ถึงครึ่งหนึ่งของอุปกรณ์ และการเพิ่มจำนวนหน่วยประมวลผลจากฟิลเตอร์แบงก์แบบ 1 มิติ เป็นฟิลเตอร์แบงก์แบบ 2 มิติ ไม่ได้ทำให้การใช้ทรัพยากรเพิ่มขึ้นเป็นทวีคูณ ด้วยผลของการใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู

ในการสังเคราะห์วงจรสามารถแสดงค่าความถี่สัญญาณนาฬิกาสูงสุดของวงจรที่ยังสามารถทำงานได้ ซึ่งมีค่าความถี่อยู่ที่ 299.892 MHz สำหรับฟิลเตอร์แบงก์แบบ 1 มิติ และที่ความถี่ 380.373 MHz สำหรับฟิลเตอร์แบงก์แบบ 2 มิติ ดังแสดงในรูปที่ 4.2




(ก)

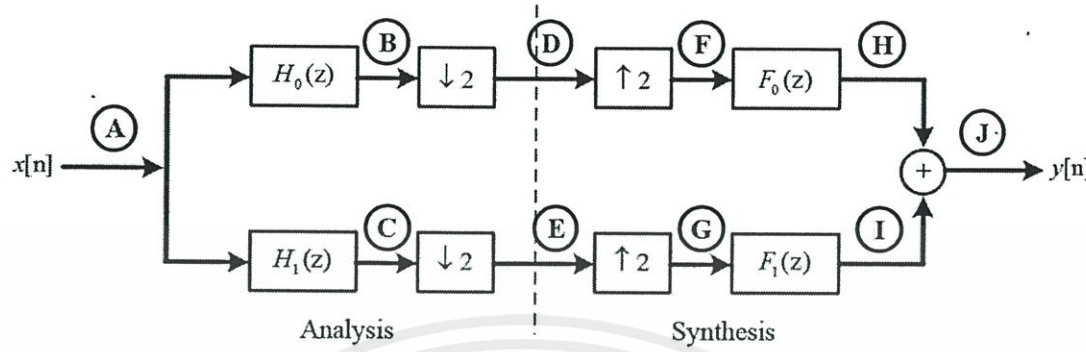



(ข)

รูปที่ 4.2 ค่าความถี่สัญญาณนาฬิกาสูงสุด (ก) ฟิลเตอร์แบงก์แบบ 1 มิติ
(ข) ฟิลเตอร์แบงก์แบบ 2 มิติ

## 4.2 ผลการทดลองของฟิลเตอร์แบงก์แบบ 1 มิติ

จากโครงสร้างฮาร์ดแวร์ของฟิลเตอร์แบงก์แบบ 1 มิติบนพื้นฐานของวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูในการออกแบบตามที่กล่าวมาในบทที่ 3 ได้ทำการออกแบบและเขียนโปรแกรมด้วยภาษา VHDL แล้วทำการจำลองการทำงานด้วยโปรแกรม ModelSim ด้วยการป้อนสัญญาณรวมความถี่ต่ำและความถี่สูง ซึ่งสัญญาณความถี่ต่ำมีค่า 200 Hz โดยอยู่ในย่านความถี่ต่ำผ่านของวงจรกรองความถี่ต่ำผ่าน และสัญญาณความถี่สูงมีค่า 8 kHz โดยอยู่ในย่านความถี่สูงผ่านของวงจรกรองความถี่สูงผ่าน ให้กับฟิลเตอร์แบงก์ที่ออกแบบ แล้วทำการเปรียบเทียบผลการจำลองการทำงานของฮาร์ดแวร์ที่ได้จาก ModelSim กับการจำลองการทำงานบนโปรแกรมแมตแล็บเพื่อตรวจสอบความถูกต้อง ก่อนนำไปทำเป็นฮาร์ดแวร์จริง แล้วทำการเทียบผลทั้งสาม โดยมีจุดทดสอบแต่ละจุดบนบล็อกไดอะแกรมจะถูกกำหนดขึ้นมาเพื่อให้ง่ายในการเปรียบเทียบดังรูปที่ 4.3




รูปที่ 4.3 บล็อกไดอะแกรมจุดทดสอบบนฟิลเตอร์แบงก์แบบ 1 มิติ [9]

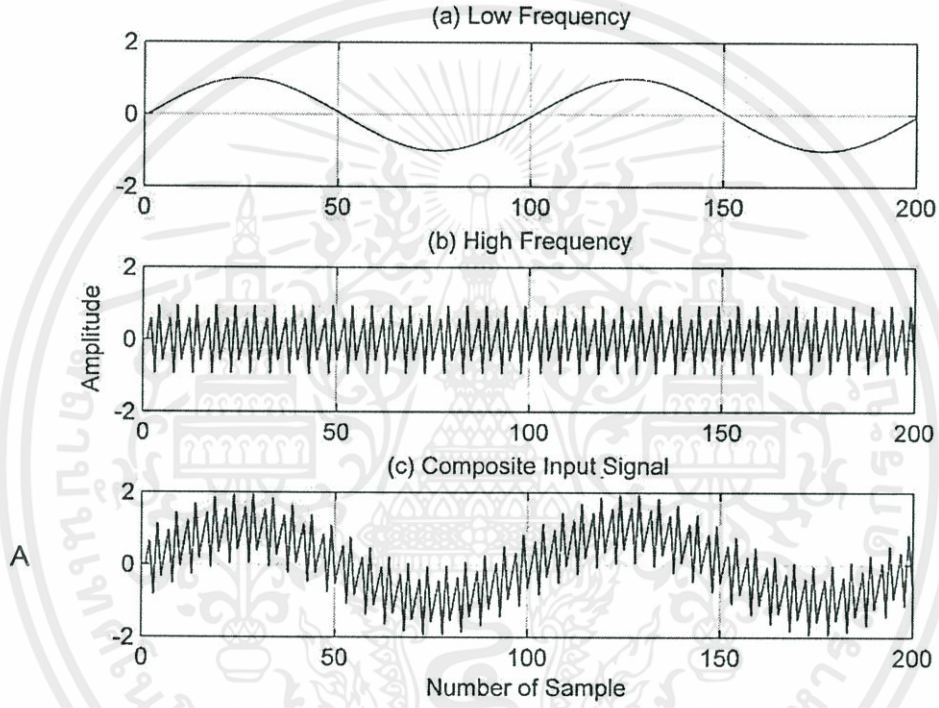
จากบล็อกไดอะแกรมจุดทดสอบ สัญญาณที่จุดต่างๆ จะถูกเปรียบเทียบเพื่อตรวจสอบความถูกต้องของฟิลเตอร์แบงก์ที่ออกแบบ สัญญาณอินพุตที่เป็นสัญญาณรวมความถี่ต่ำและสัญญาณความถี่สูง (Composite Signal) จะถูกป้อนเข้าสู่ฟิลเตอร์แบงก์ที่จุด A สัญญาณจะผ่านเข้าตัวกรองความถี่ต่ำผ่าน $H_0(z)$ ได้สัญญาณที่จุด B ในขณะเดียวกันสัญญาณที่ป้อนก็จะผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ ได้สัญญาณที่จุด C สัญญาณที่ผ่านตัวกรองทั้งสองจะถูกลดค่าสุ่มตัวอย่างลงสอง ได้สัญญาณเอาต์พุตภาควิเคราะห์เป็นสัญญาณที่จุด D และ E ตามลำดับ ส่วนในภาคสังเคราะห์สัญญาณที่ถูกลดค่าสุ่มตัวอย่างทั้งสองจะถูกเพิ่มค่าสุ่มตัวอย่างก่อนทำให้ได้สัญญาณที่จุด F และ G จากนั้นสัญญาณที่จุด F จะป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ ได้สัญญาณที่จุด H ส่วนสัญญาณที่จุด G จะถูกป้อนสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ ได้สัญญาณที่จุด I สัญญาณทั้งสองจุดจะถูกรวมเป็นเอาต์พุตได้สัญญาณที่จุด J

จากการระบุตำแหน่งตามรูปที่ 4.3 บนบล็อกไดอะแกรมทำให้ได้ผลการทดลองฟิลเตอร์แบงก์แบบ 1 มิติ ตามจุดเหล่านั้นดังต่อไปนี้

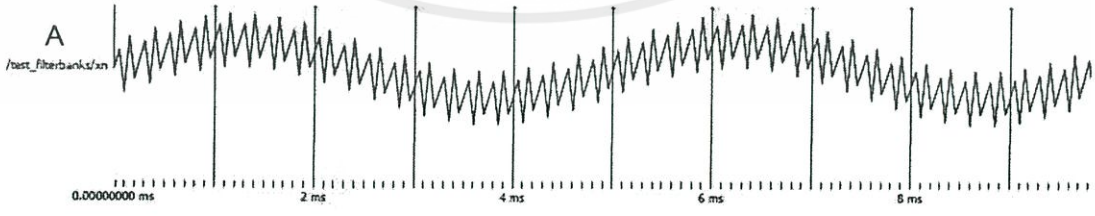
1) สัญญาณที่จุด A

ในการทดสอบการทำงานของฟิลเตอร์แบงก์แบบ 1 มิติสำหรับจุดนี้จะสร้างสัญญาณอินพุตที่เป็นสัญญาณรวมระหว่างความถี่ต่ำและความถี่สูงเพื่อทำการป้อนให้กับฟิลเตอร์แบงก์ โดยจะทำการสร้างสัญญาณนี้โดยใช้ฟังก์ชันไซน์ในโปรแกรมแมตแล็บเพื่อตรวจสอบการทำงาน ดังรูปที่ 4.4 แล้วทำการแปลงค่าให้อยู่ในรูปของเลขฐานสองแบบส่วนเติมเต็มสอง เพื่อที่จะนำค่าที่ได้ไปใช้ในการป้อนเป็นสัญญาณอินพุตให้กับการจำลองการทำงานของฮาร์ดแวร์ที่ออกแบบในโปรแกรม

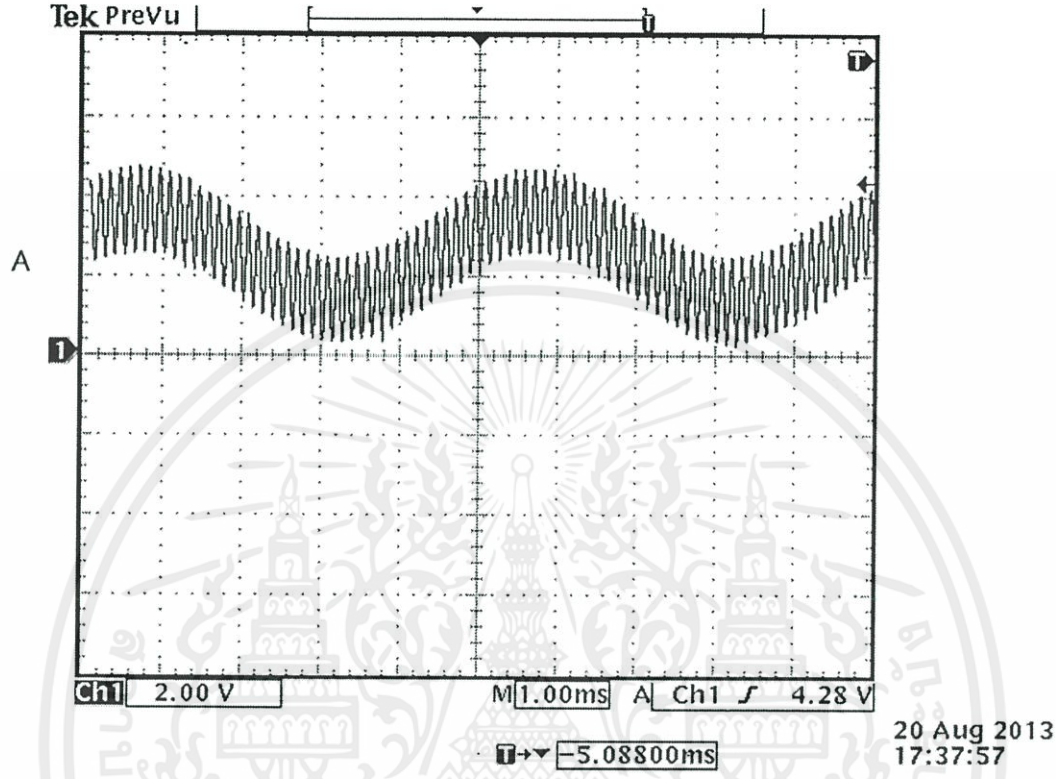
VHDL และจำลองการทำงานบนโปรแกรม ModelSim ได้ผลการจำลองการทำงานดังรูปที่ 4.5 จากนั้นนำไปทำเป็นฮาร์ดแวร์ เริ่มทำการทดสอบวงจรด้วยการป้อนสัญญาณความถี่ต่ำและสัญญาณความถี่สูงจากฟังก์ชันเจเนอเรเตอร์เข้าวงจรขยายรวมสัญญาณเพื่อรวมเป็นสัญญาณผสม 2 ความถี่ ได้สัญญาณเอาต์พุต Ch.1 ก่อนทำการป้อนสัญญาณอินพุตเข้าสู่ฟิลเตอร์แบงก์ที่จุด A ได้ผลการทดลองการทำงานจริงตามรูปที่ 4.6




รูปที่ 4.4 สัญญาณที่จุด A จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



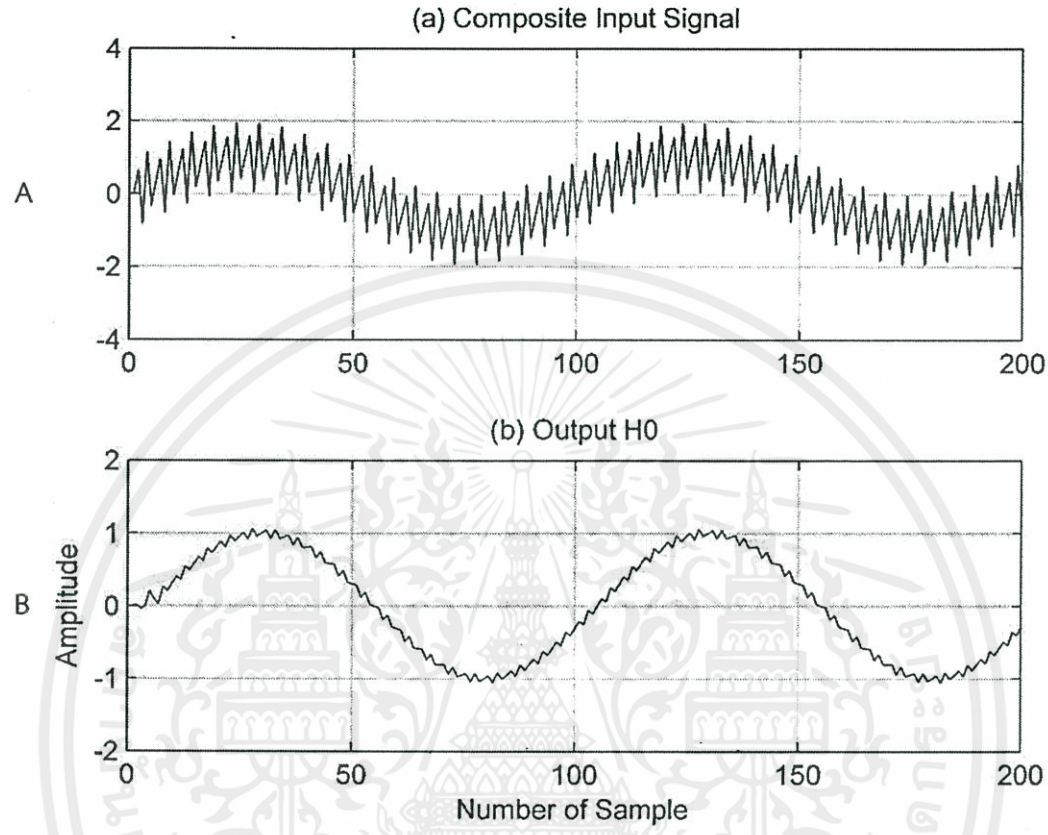

รูปที่ 4.5 สัญญาณที่จุด A จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim



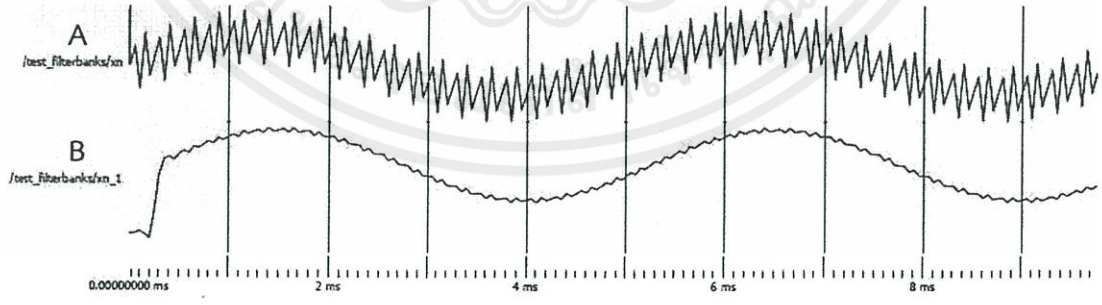

รูปที่ 4.6 สัญญาณที่จุด A จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

2) สัญญาณที่จุด B

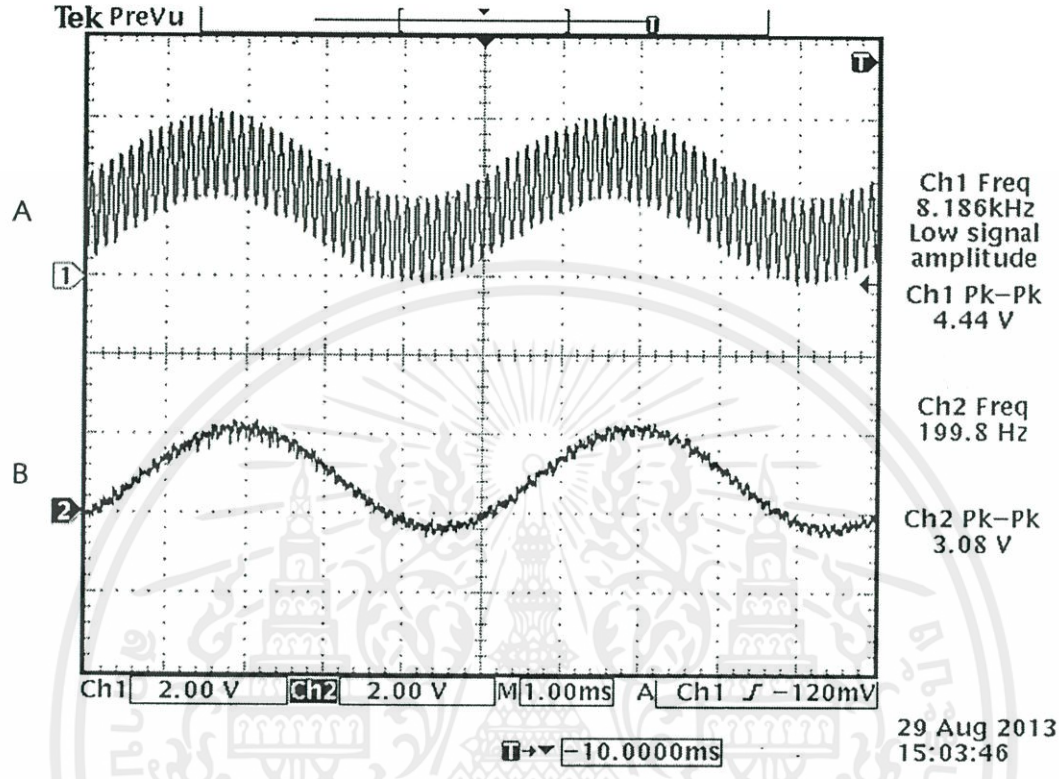
สัญญาณที่จุด A จะถูกป้อนให้กับตัวกรองความถี่ต่ำผ่าน $H_0(z)$ ซึ่งจะทำการหาผลคูณของอินพุตกับสัมประสิทธิ์ของตัวกรอง (Convolution) ได้สัญญาณเอาต์พุตออกจากตัวกรองเป็นสัญญาณที่จุด B ได้ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ดังรูปที่ 4.7 นำมาเปรียบเทียบกับผลการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.8 และในส่วนการทำงานจริงของวงจร หลังจากสัญญาณผสมที่จุด A (Ch.1) ผ่านเข้าสู่ตัวกรองความถี่ต่ำผ่าน $H_0(z)$ จะทำให้สัญญาณเฉพาะในย่านความถี่ต่ำออกมาที่ Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.9




รูปที่ 4.7 สัญญาณที่จุด B จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



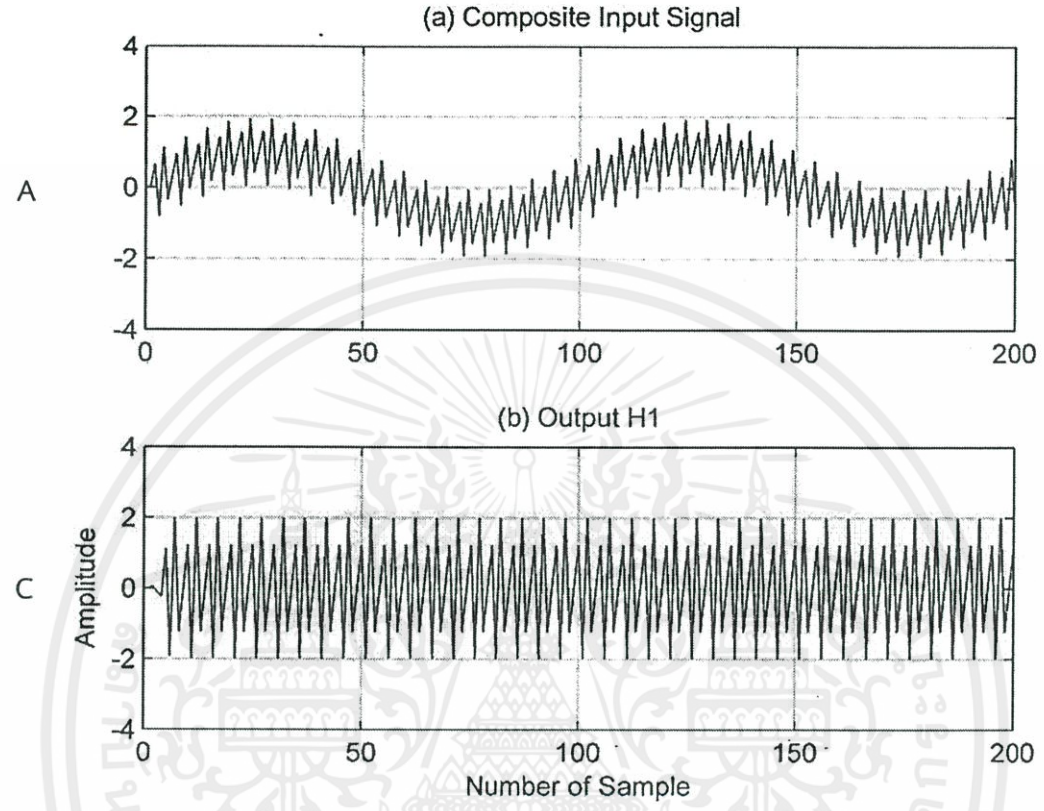

รูปที่ 4.8 สัญญาณที่จุด B จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim



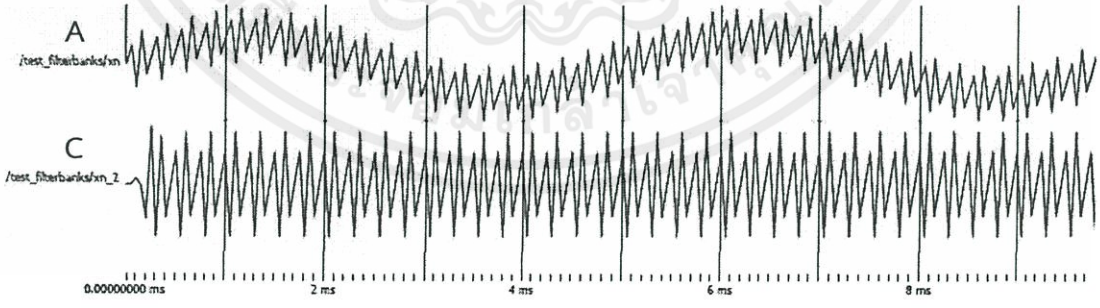

รูปที่ 4.9 สัญญาณที่จุด B จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

3) สัญญาณที่จุด C

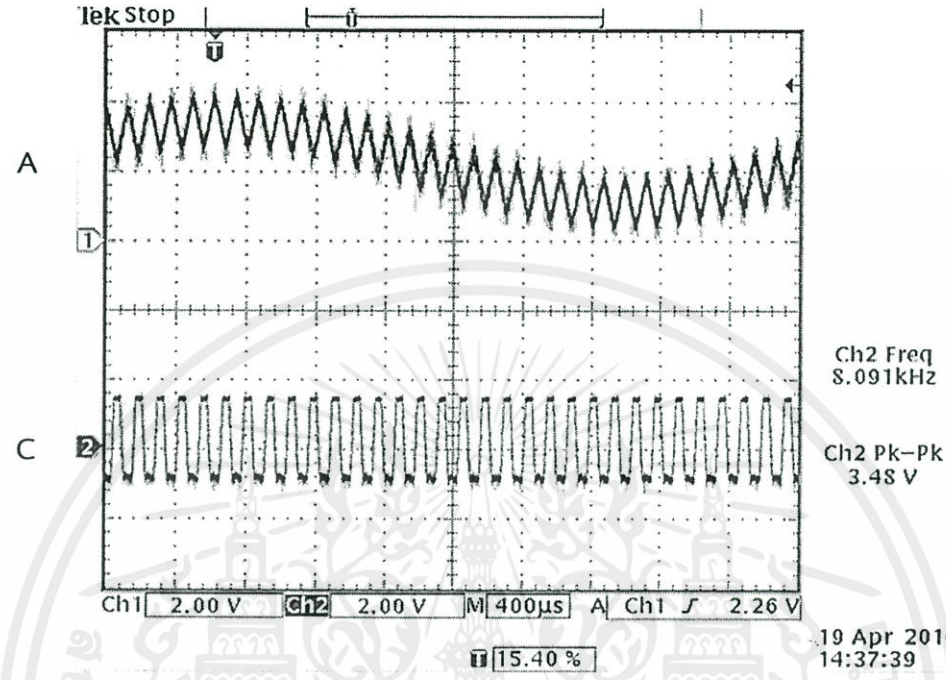
สัญญาณที่จุด A จะถูกป้อนให้กับตัวกรองความถี่สูงผ่าน $H_1(z)$ ซึ่งจะทำการหาผลคูณของอินพุตกับสัมประสิทธิ์ของตัวกรอง (Convolution) ได้สัญญาณเอาต์พุตออกจากตัวกรองเป็นสัญญาณที่จุด C ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ได้ดังรูปที่ 4.10 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.11 และในส่วนการทำงานจริงของวงจร หลังจากสัญญาณผสมที่จุด A (Ch.1) ผ่านเข้าสู่ตัวกรองความถี่สูงผ่าน $H_1(z)$ จะทำให้สัญญาณเฉพาะในย่านความถี่สูงออกมาที่ Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.12




รูปที่ 4.10 สัญญาณที่จุด C จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



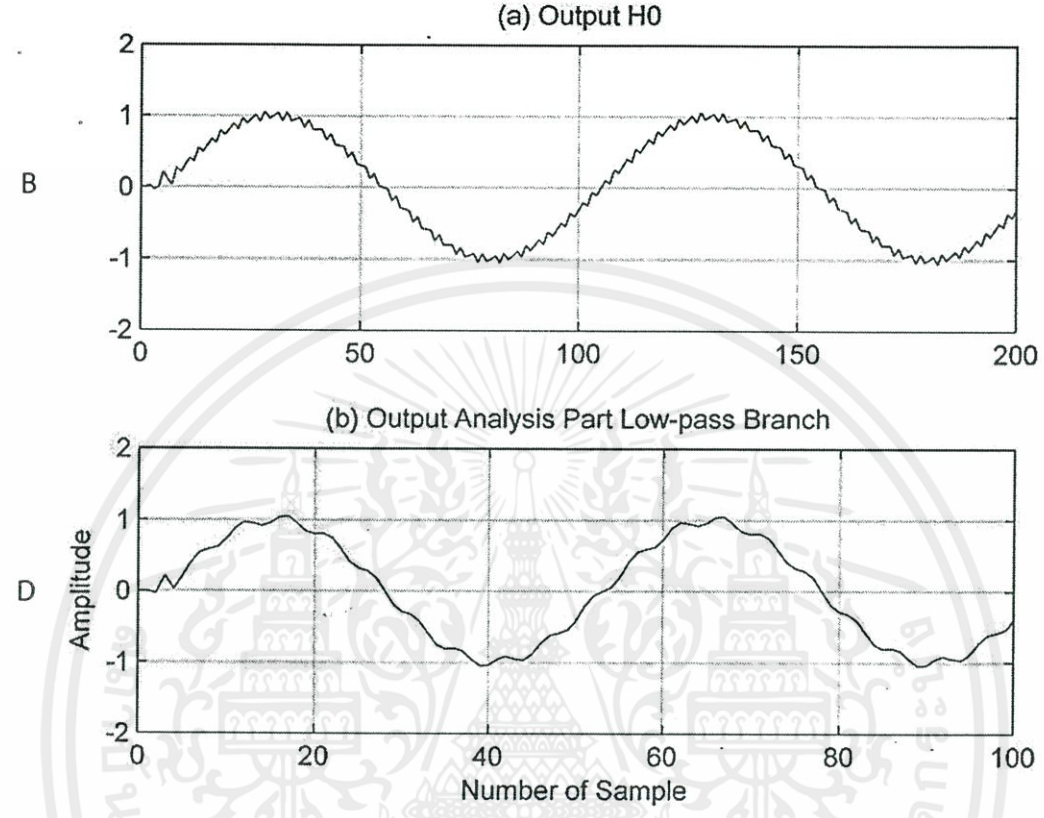

รูปที่ 4.11 สัญญาณที่จุด C จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim



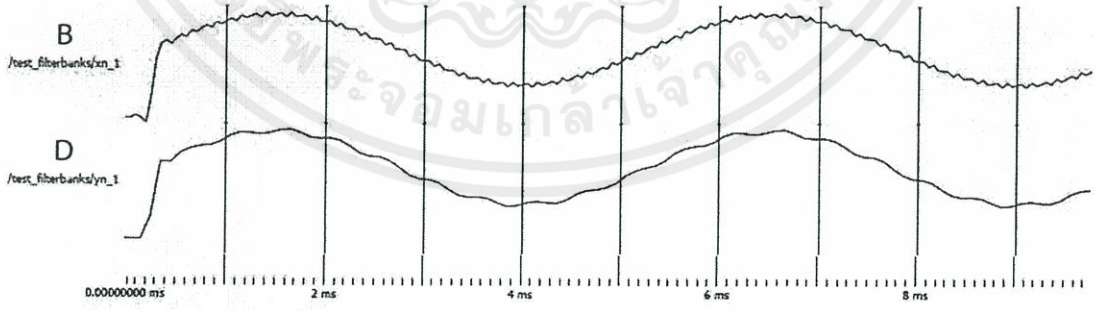

รูปที่ 4.12 สัญญาณที่จุด C จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

4) สัญญาณที่จุด D

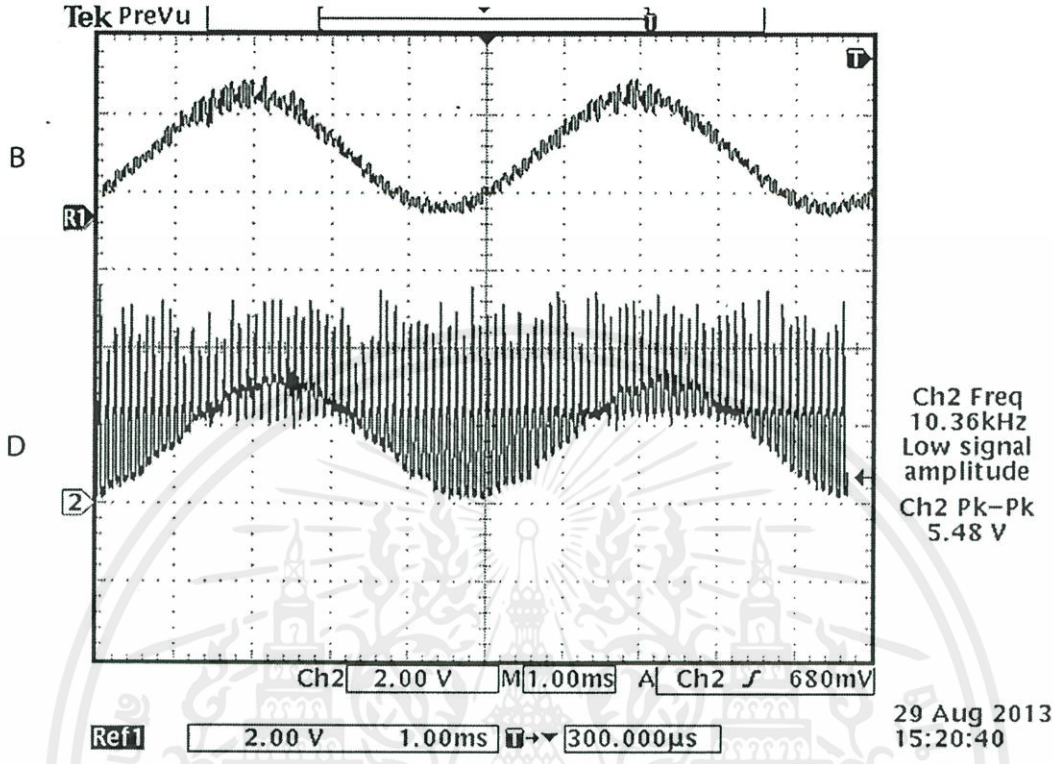
สัญญาณอินพุตเมื่อผ่านตัวกรองความถี่ต่ำผ่าน $H_0(z)$ จะถูกลดค่าสุ่มตัวอย่างลงสอง ได้สัญญาณเอาต์พุตออกจากตัวลดค่าสุ่มสัญญาณได้เป็นสัญญาณที่จุด D ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ดังรูปที่ 4.13 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.14 และในส่วนการทำงานจริงของวงจร หลังจากสัญญาณที่จุด B ผ่านตัวกรองความถี่ต่ำผ่าน $H_0(z)$ (Ch.R1) จะสู่ขั้นตอนลดค่าสุ่มตัวอย่าง ได้สัญญาณเอาต์พุต Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.15 โดยจะพบว่าได้สัญญาณเอาต์พุตที่มีลักษณะไม่ตรงกับทางทฤษฎี เนื่องจากว่าในการจำลองการทำงานด้วยโปรแกรมแมตแล็บการลดค่าสุ่มตัวอย่างลงสองจะใช้การทิ้งค่าข้อมูล แต่วงจรจริงนั้นจะลดค่าสุ่มตัวอย่างลงสองแล้วเติมค่าที่เป็นศูนย์แทนในตำแหน่งที่ค่าข้อมูลที่จะถูกทิ้ง เนื่องจากการทำงานจริงเราไม่สามารถหยุดการทำงานตามสัญญาณควบคุมเป็นจังหวะแล้วจัดเรียงข้อมูลใหม่ได้เหมือนกับบนโปรแกรมแมตแล็บ จึงได้สัญญาณที่มีลักษณะกระโดดจากค่าที่เป็นอยู่สลับกับค่าศูนย์นั่นเอง




รูปที่ 4.13 สัญญาณที่จุด D จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



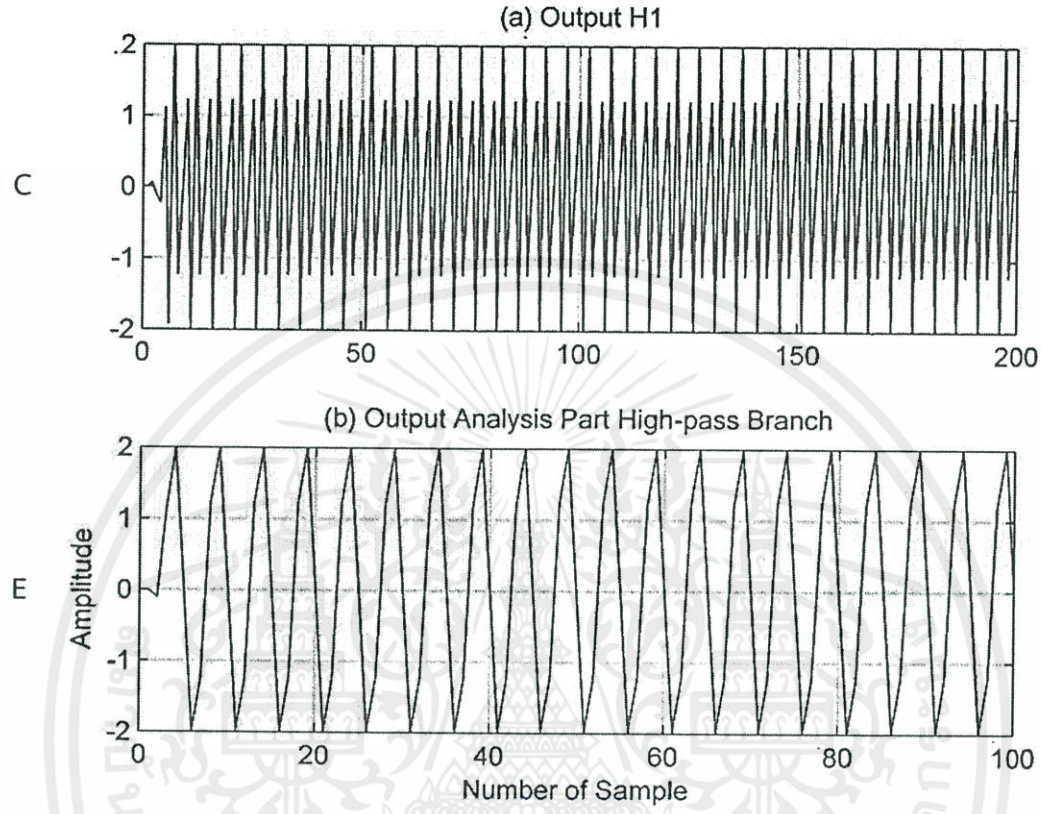

รูปที่ 4.14 สัญญาณที่จุด D จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim



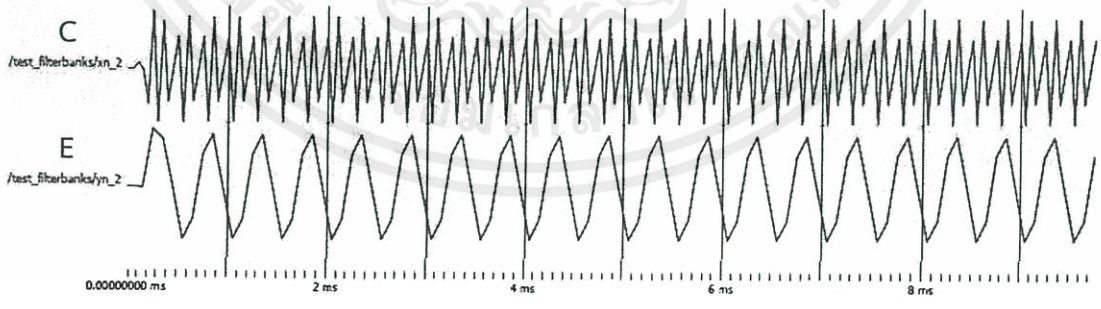

รูปที่ 4.15 สัญญาณที่จุด D จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

5) สัญญาณที่จุด E

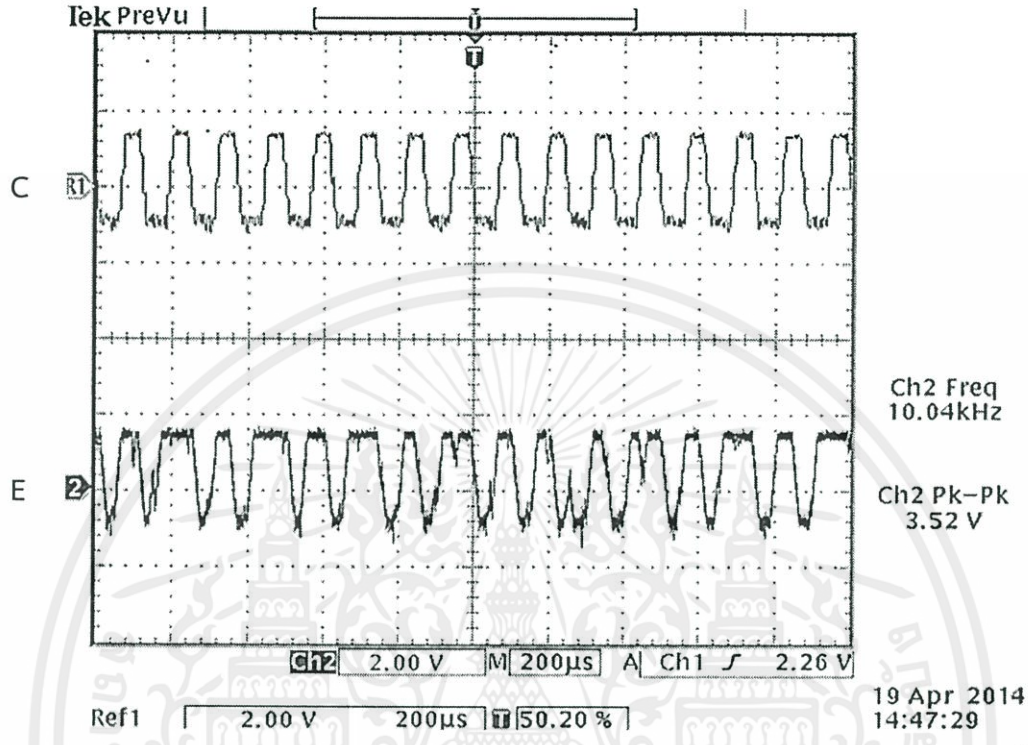
สัญญาณอินพุตเมื่อผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ จะถูกลดค่าสุ่มตัวอย่างลงสอง ได้สัญญาณเอาต์พุตออกจากตัวลดค่าสุ่มสัญญาณได้เป็นสัญญาณที่จุด E ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ได้ดังรูปที่ 4.16 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.17 และในส่วนการทำงานจริงของวงจร หลังจากสัญญาณที่จุด C ผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ (Ch.R1) จะสู่ขั้นตอนลดค่าสุ่มตัวอย่าง ได้สัญญาณเอาต์พุต Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.18 โดยจะพบว่าได้สัญญาณเอาต์พุตที่มีลักษณะไม่ตรงกับทางทฤษฎี เนื่องจากว่าในการจำลองการทำงานด้วยโปรแกรมแมตแล็บการลดค่าสุ่มตัวอย่างลงสองจะใช้การทิ้งค่าข้อมูล แต่วงจรจริงนั้นจะลดค่าสุ่มตัวอย่างลงสองแล้วเติมค่าที่เป็นศูนย์แทนในตำแหน่งที่ค่าข้อมูลที่จะถูกทิ้ง เนื่องจากการทำงานจริงเราไม่สามารถหยุดการทำงานตามสัญญาณควบคุมเป็นจังหวะแล้วจัดเรียงข้อมูลใหม่ได้เหมือนกับบนโปรแกรมแมตแล็บ จึงได้สัญญาณที่มีลักษณะกระโดดจากค่าที่เป็นอยู่สลับกับค่าศูนย์ หรือบางตำแหน่งอาจคงที่นั่นเอง




รูปที่ 4.16 สัญญาณที่จุด E จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



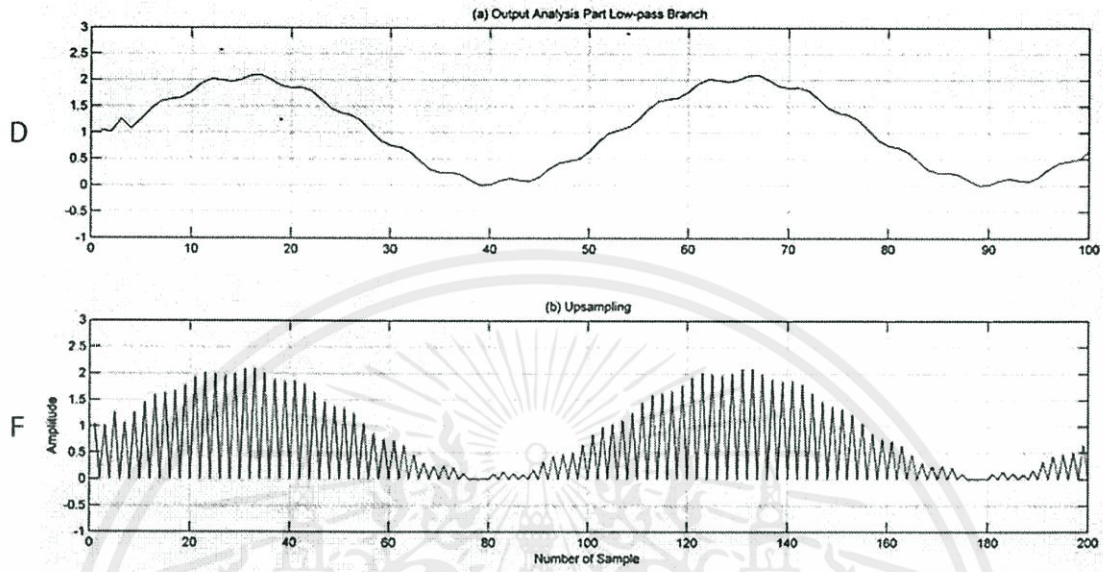

รูปที่ 4.17 สัญญาณที่จุด E จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim



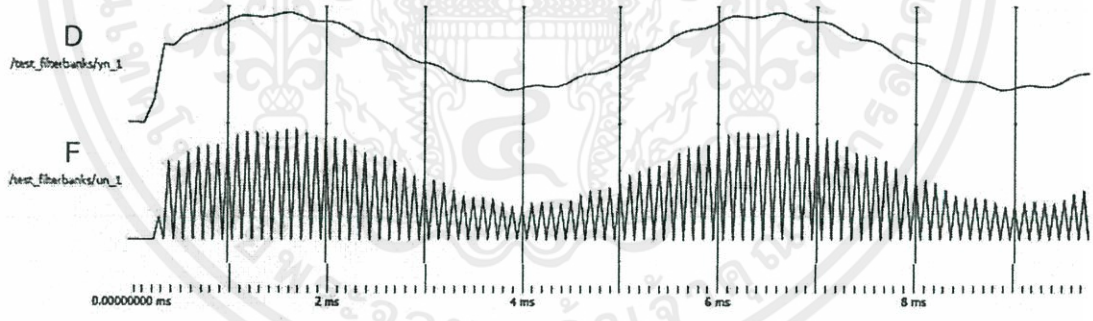

รูปที่ 4.18 สัญญาณที่จุด E จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

6) สัญญาณที่จุด F

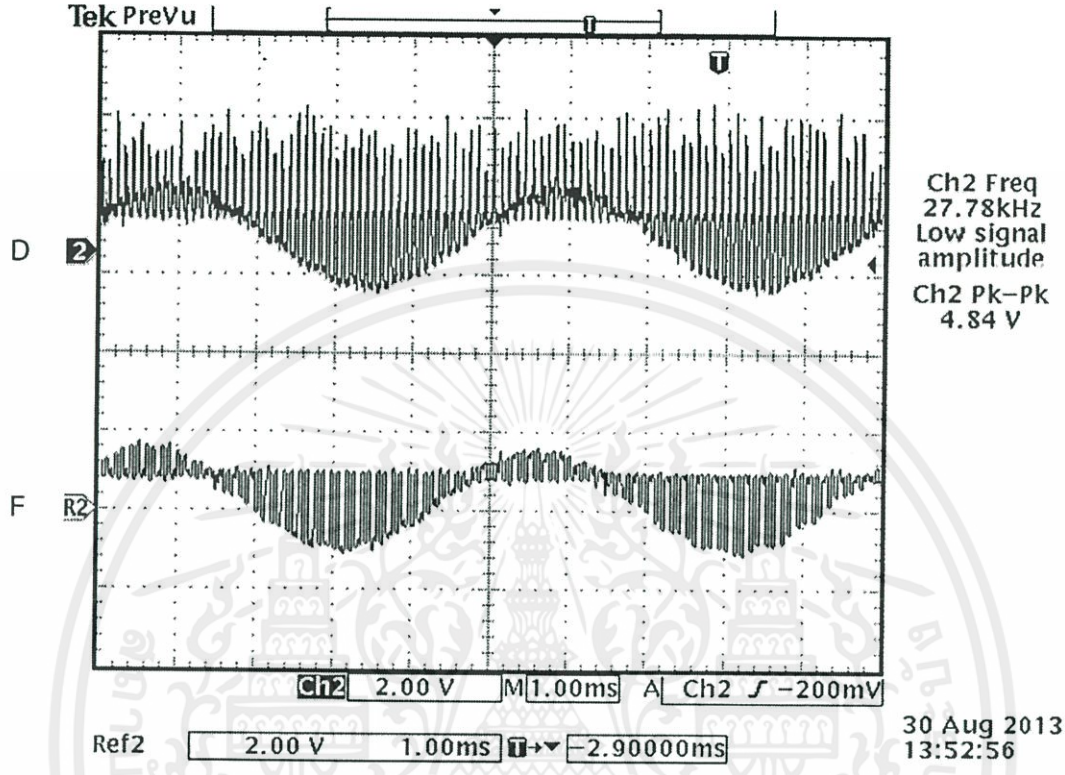
สัญญาณเอาต์พุตของภาควิเคราะห์ที่จุด D จะถูกป้อนเข้าสู่ภาคสังเคราะห์ เพื่อทำการกู้กลับสัญญาณเดิมคืนมา โดยเริ่มจากการเพิ่มค่าสุ่มตัวอย่างขึ้นสอง ได้สัญญาณเอาต์พุตออกจากตัวเพิ่มค่าสุ่มสัญญาณได้เป็นสัญญาณที่จุด F ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บได้ดังรูปที่ 4.19 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.20 และในส่วนการทำงานจริงของวงจร หลังจากได้สัญญาณภาควิเคราะห์ที่จุด D (Ch.2) ก่อนเข้าสู่ภาคสังเคราะห์จะทำการเพิ่มค่าสุ่มตัวอย่างก่อน ได้สัญญาณเอาต์พุต Ch.R2 ซึ่งมีลักษณะตามรูปที่ 4.21 โดยจะพบว่าได้สัญญาณเอาต์พุตที่มีลักษณะคล้ายกับทางทฤษฎี เนื่องจากสัญญาณที่ทำในกระบวนจริงไม่ได้ถูกยกระดับสัญญาณเหมือนกับการจำลองการทำงานบนโปรแกรมแมตแล็บ




รูปที่ 4.19 สัญญาณที่จุด F จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



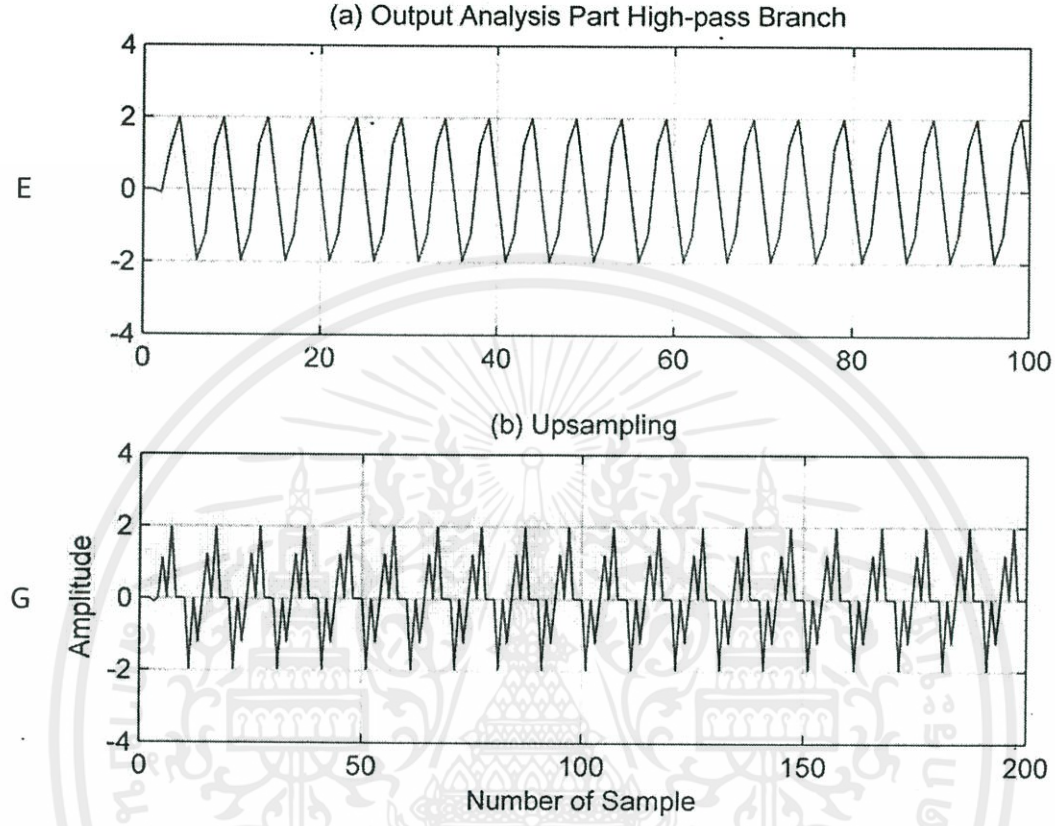

รูปที่ 4.20 สัญญาณที่จุด F จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim



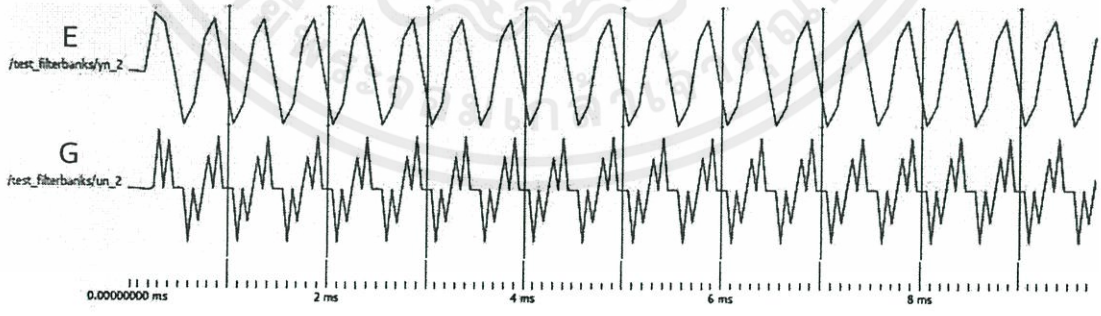

รูปที่ 4.21 สัญญาณที่จุด F จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

7) สัญญาณที่จุด G

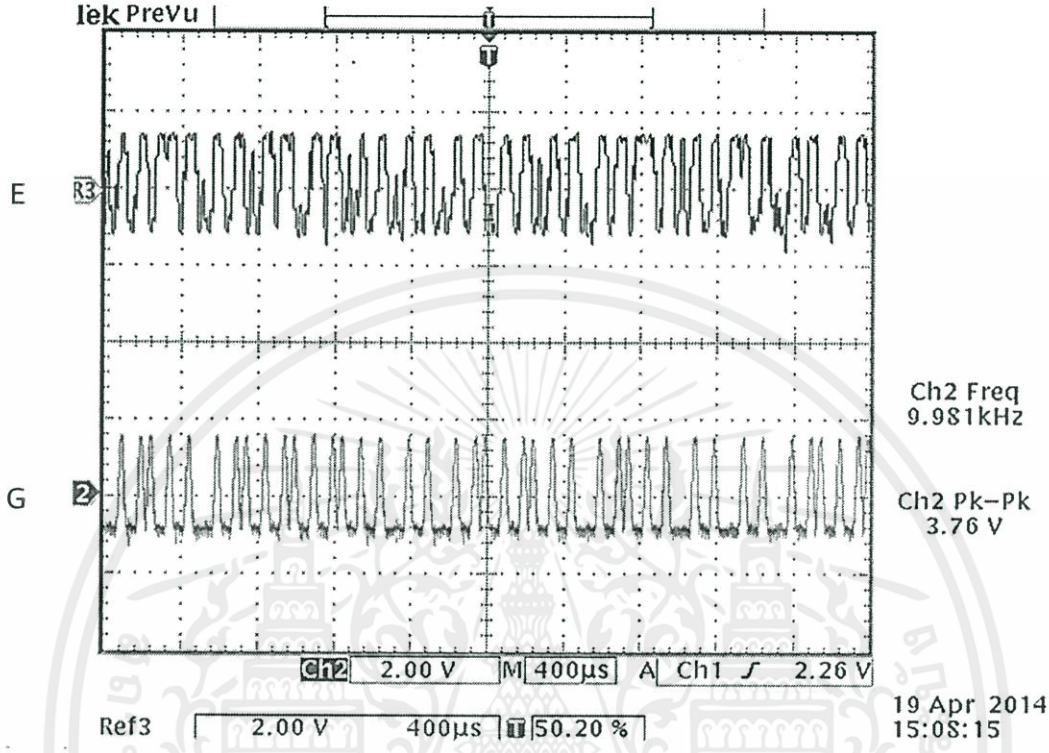
สัญญาณเอาต์พุตของภาควิเคราะห์ที่จุด E จะถูกป้อนเข้าสู่ภาคสังเคราะห์ เพื่อทำการกู้กลับสัญญาณเดิมคืนมา โดยเริ่มจากการเพิ่มค่าสุ่มตัวอย่างขึ้นสอง ได้สัญญาณเอาต์พุตออกจากตัวเพิ่มค่าสุ่มสัญญาณได้เป็นสัญญาณที่จุด G ผลการจำลองการทำงานบนโปรแกรม ModelSim ได้ดังรูปที่ 4.22 และเปรียบเทียบกับการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ดังรูปที่ 4.23 และในส่วนการทำงานจริงของวงจร หลังจากได้สัญญาณภาควิเคราะห์ที่จุด E (Ch.R3) ก่อนเข้าสู่ภาคสังเคราะห์จะทำการเพิ่มค่าสุ่มตัวอย่างก่อน ได้สัญญาณเอาต์พุต Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.24




รูปที่ 4.22 สัญญาณที่จุด G จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ
ด้วยโปรแกรมแมตแล็บ



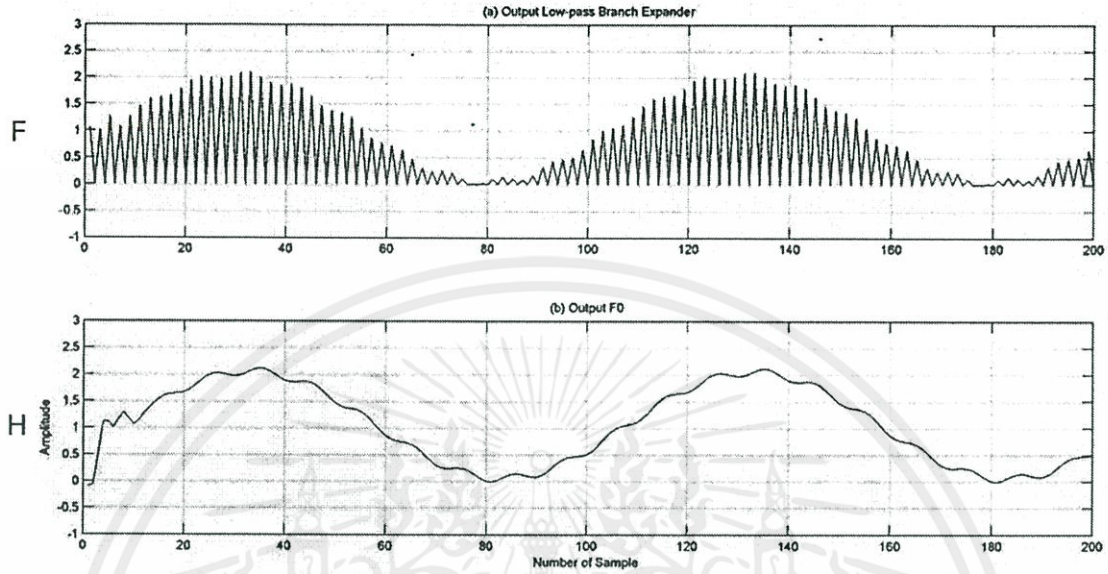

รูปที่ 4.23 สัญญาณที่จุด G จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ
ด้วยโปรแกรม ModelSim



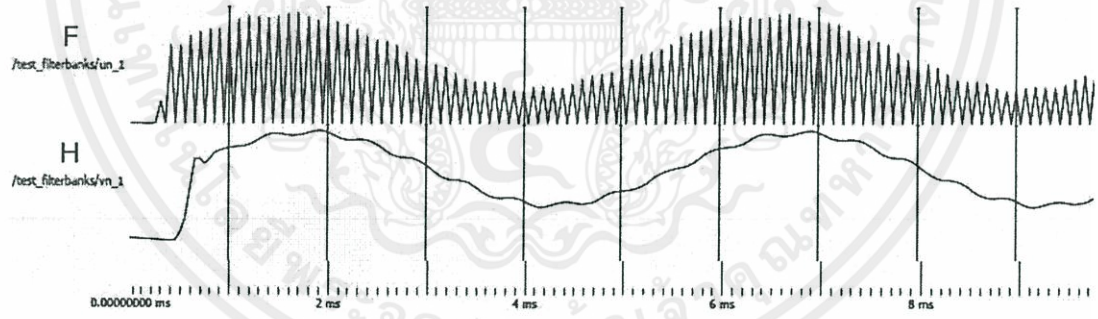

รูปที่ 4.24 สัญญาณที่จุด G จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

8) สัญญาณที่จุด H

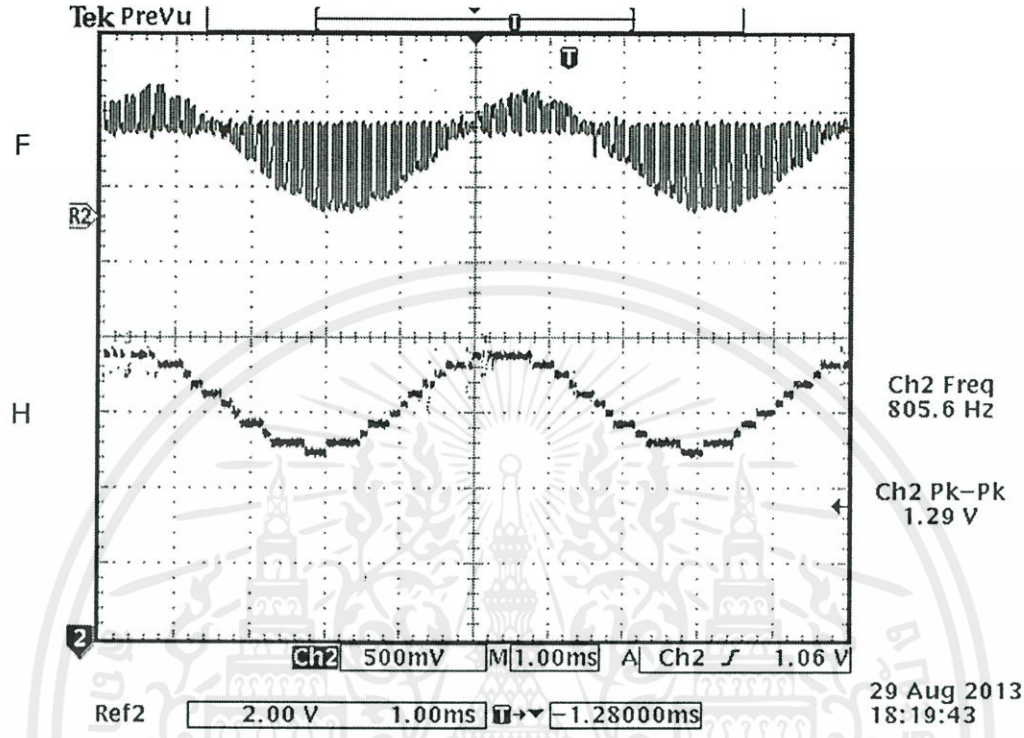
สัญญาณเอาต์พุตที่ได้จากการเพิ่มค่าสุ่มตัวอย่างที่จุด F จะถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ ได้สัญญาณเอาต์พุตออกจากตัวกรองได้เป็นสัญญาณที่จุด H ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ได้ดังรูปที่ 4.25 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.26 และในส่วนการทำงานจริงของวงจร หลังจากได้สัญญาณที่เพิ่มค่าสุ่มตัวอย่างที่จุด F (Ch.R2) จะถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ ได้สัญญาณเอาต์พุต Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.27




รูปที่ 4.25 สัญญาณที่จุด H จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ
ด้วยโปรแกรมแมตแล็บ



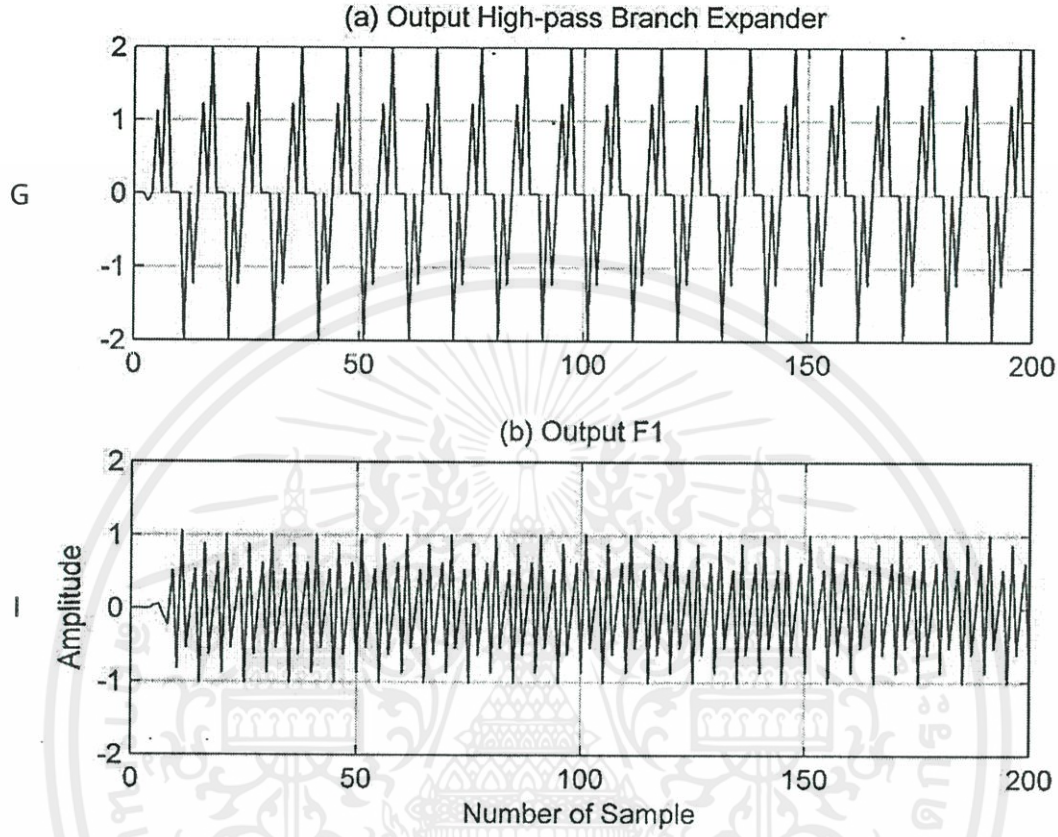

รูปที่ 4.26 สัญญาณที่จุด H จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ
ด้วยโปรแกรม ModelSim



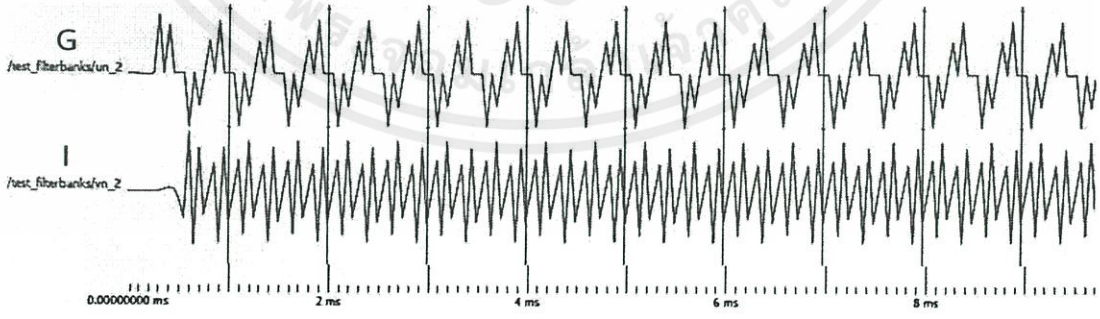

รูปที่ 4.27 สัญญาณที่จุด H จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

9) สัญญาณที่จุด I

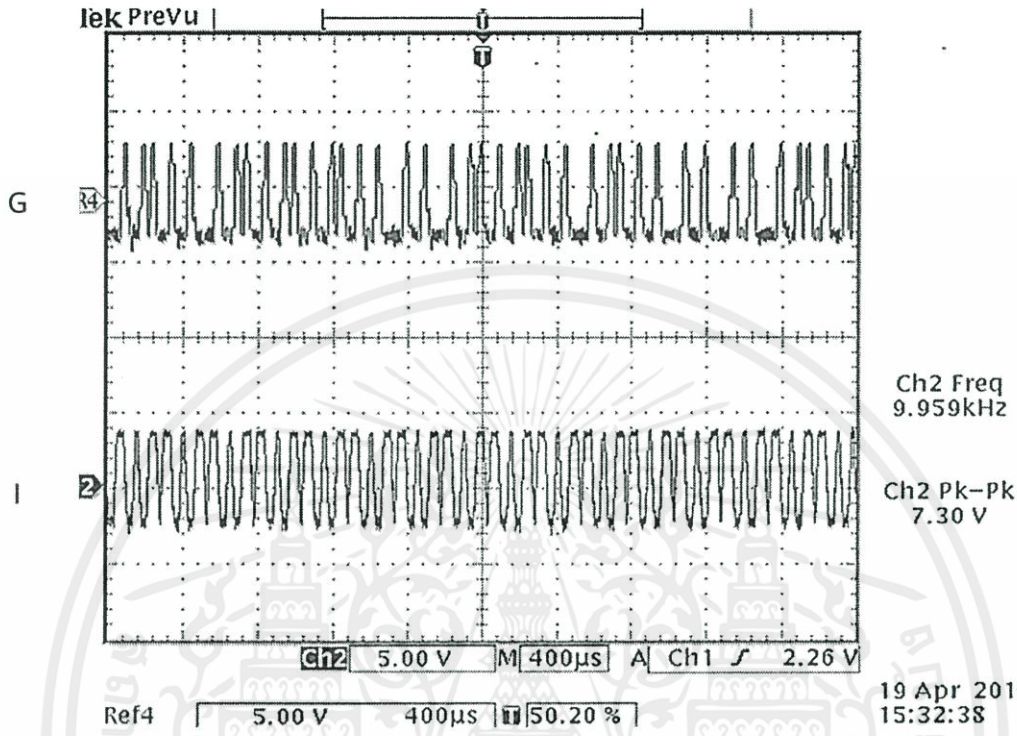
สัญญาณเอาต์พุตที่ได้จากการเพิ่มค่าสุ่มตัวอย่างที่จุด G จะถูกป้อนเข้าสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ ได้สัญญาณเอาต์พุตออกจากตัวกรองได้เป็นสัญญาณที่จุด I ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ได้ดังรูปที่ 4.28 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.29 และในส่วนการทำงานจริงของวงจร หลังจากได้สัญญาณที่เพิ่มค่าสุ่มตัวอย่างที่จุด G (Ch.R4) จะถูกป้อนเข้าสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ ได้สัญญาณเอาต์พุต Ch.2 ซึ่งมีลักษณะตามรูปที่ 4.30




รูปที่ 4.28 สัญญาณที่จุด I จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ
ด้วยโปรแกรมแมตแล็บ



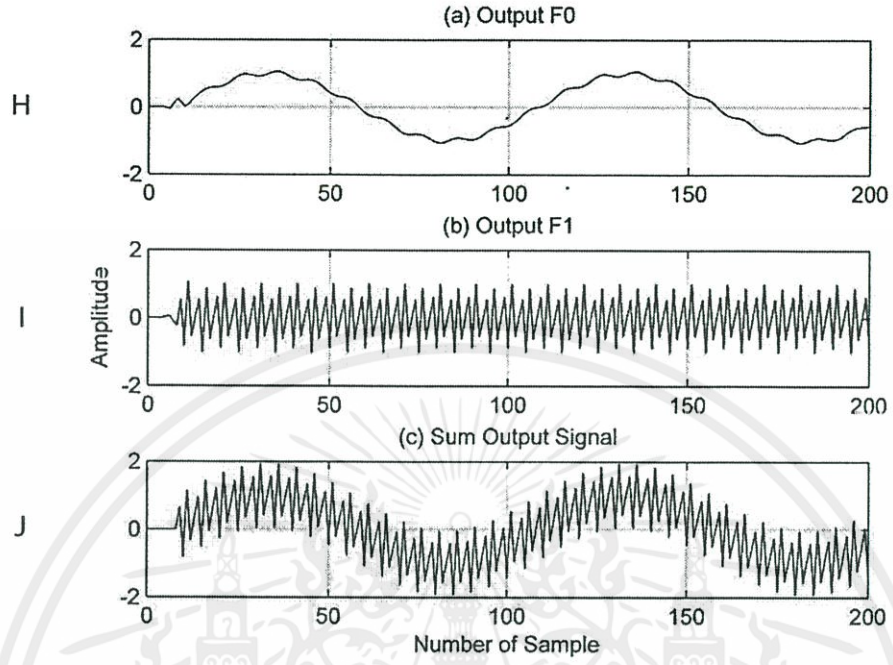

รูปที่ 4.29 สัญญาณที่จุด I จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ
ด้วยโปรแกรม ModelSim



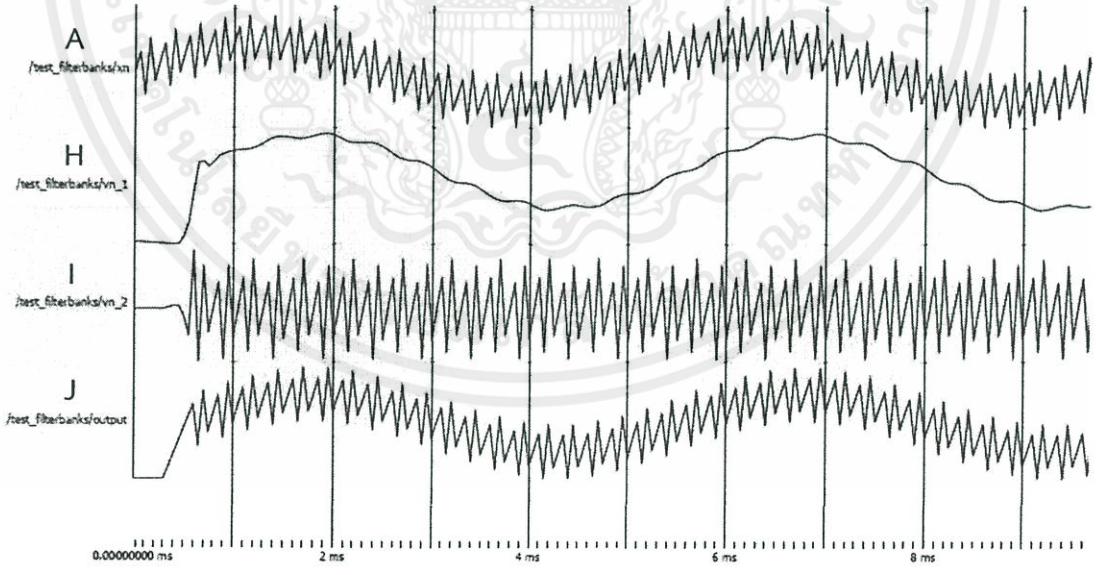

รูปที่ 4.30 สัญญาณที่จุด I จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

10) สัญญาณที่จุด J

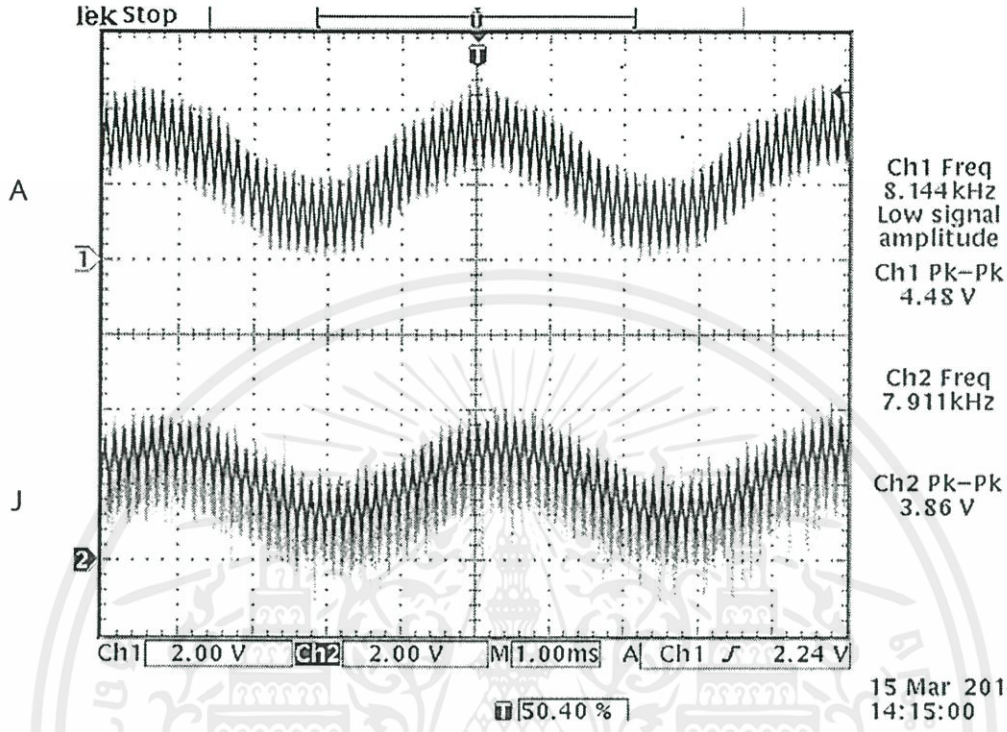
สัญญาณเอาต์พุตที่ได้จากตัวกรองความถี่ต่ำผ่าน $F_0(z)$ ที่จุด H และจากตัวกรองความถี่สูงผ่าน $F_1(z)$ ที่จุด I จะถูกนำมารวมกันได้สัญญาณเอาต์พุตออกจากฟิลเตอร์แบงก์ได้เป็นสัญญาณที่จุด J ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บ ดังรูปที่ 4.31 และเปรียบเทียบกับการจำลองการทำงานบนโปรแกรม ModelSim ดังรูปที่ 4.32 และในส่วนการทำงานจริงของวงจรสัญญาณเอาต์พุตที่ได้จากฟิลเตอร์แบงก์แบบ 1 มิติ ที่จุด J (Ch.2) พบว่ามีลักษณะคล้ายสัญญาณอินพุต (Ch.1) โดยแสดงได้ตามรูปที่ 4.33




รูปที่ 4.31 สัญญาณที่จุด J จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรมแมตแล็บ



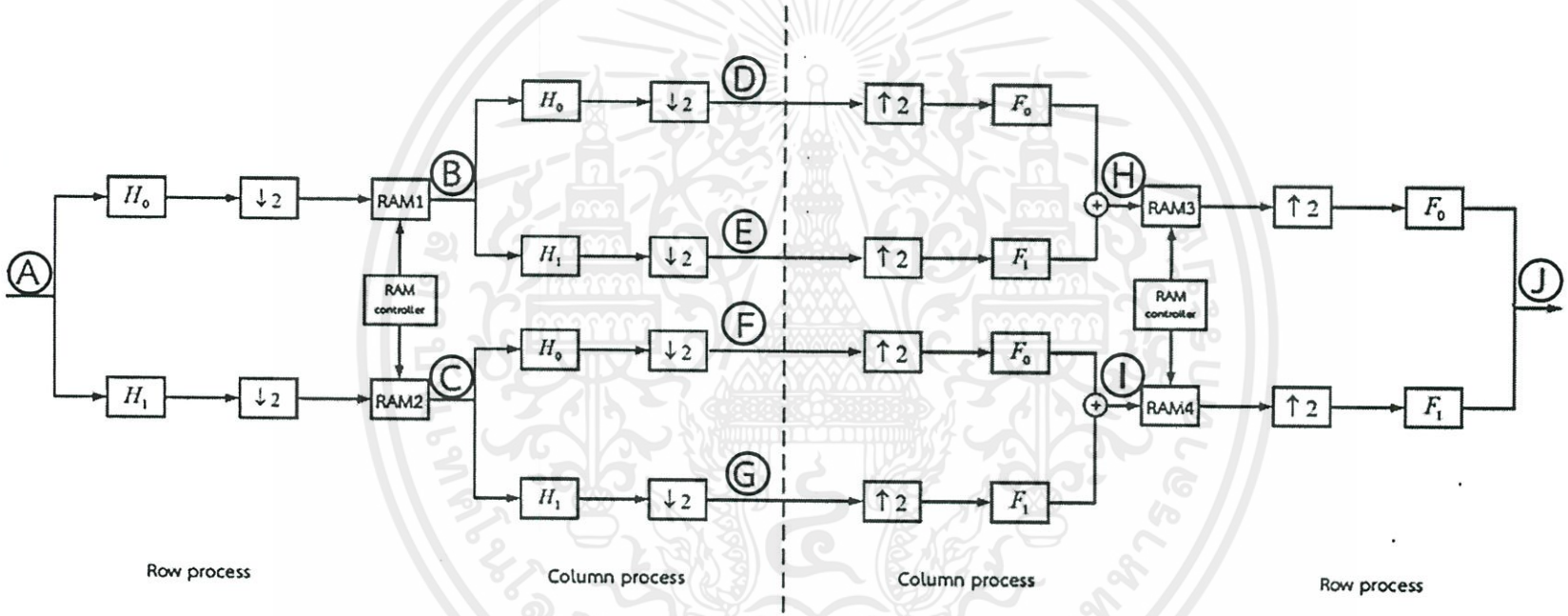

รูปที่ 4.32 สัญญาณที่จุด J จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 1 มิติ ด้วยโปรแกรม ModelSim




รูปที่ 4.33 สัญญาณที่จุด J จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 1 มิติ

## 4.3 ผลการทดลองของฟิลเตอร์แบงก์แบบ 2 มิติ

จากโครงสร้างฮาร์ดแวร์ของฟิลเตอร์แบงก์บนแบบ 2 มิติบนพื้นฐานของวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูในการออกแบบตามที่กล่าวมาในบทที่ 3 เมื่อได้ทำการออกแบบและเขียนโปรแกรมด้วยภาษา VHDL แล้วทำการจำลองการทำงานวงจรย่อยด้วยโปรแกรม ISim เพื่อตรวจสอบความถูกต้องของการทำงาน จากนั้นนำไปทำเป็นฮาร์ดแวร์จริง แล้วทำการเปรียบเทียบการจำลองการทำงานบนโปรแกรมแมตแล็บกับผลการจำลองการทำงานของฮาร์ดแวร์ โดยมีจุดทดสอบแต่ละจุดบนบล็อกไดอะแกรมจะถูกกำหนดขึ้นมาเพื่อให้ง่ายในการเปรียบเทียบดังรูปที่ 4.34




รูปที่ 4.34 บล็อกไดอะแกรมจุดทดสอบบนฟิลเตอร์แบงก์แบบ 2 มิติ

จากบล็อกไดอะแกรมจุดทดสอบรูปที่ 4.34 สัญญาณที่จุดต่างๆ จะถูกเปรียบเทียบเพื่อตรวจสอบความถูกต้องของฟิลเตอร์แบงก์ที่ออกแบบ สัญญาณอินพุตที่เป็นสัญญาณภาพ จะถูกป้อนเข้าสู่ฟิลเตอร์แบงก์ที่จุด A สัญญาณจะผ่านเข้าตัวกรองความถี่ต่ำผ่าน $H_0(z)$ ของสเตจที่ 1 จากนั้นจะถูกลดค่าสุ่มตัวอย่างลงสอง ระหว่างการประมวลผลในแต่ละพิกเซลจะถูกจัดเก็บเข้าในหน่วยความจำชั่วคราวตัวที่ 1 เพื่อทำการจัดเรียงเป็นภาพอีกก่อนนำไปประมวลผลต่อที่จุด B ในขณะเดียวกันสัญญาณที่ป้อนก็จะผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ ของสเตจที่ 1 จากนั้นจะถูกลดค่าสุ่มตัวอย่างลงสองและจะถูกจัดเก็บเข้าที่หน่วยความจำชั่วคราวตัวที่ 2 ก่อนนำไปใช้ต่อที่จุด C จากนั้นสัญญาณที่จุด B และ C จะกระทำการเช่นเดียวกับจุด A โดยผ่านเข้าตัวกรองความถี่ต่ำผ่าน $H_0(z)$ ของสเตจที่ 2 จากนั้นจะถูกลดค่าสุ่มตัวอย่างลงสอง หรือผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ ของสเตจที่ 2 จากนั้นจะถูกลดค่าสุ่มตัวอย่างลงสอง ตามลำดับ ได้สัญญาณเอาต์พุตภาควิเคราะห์เป็นสัญญาณที่จุด D, E, F และ G ตามลำดับ ส่วนในภาคสังเคราะห์สัญญาณ D และ F จะถูกเพิ่มค่าสุ่มตัวอย่างจากนั้นจะถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ ของสเตจที่ 2 ได้สัญญาณที่จุดก่อนเข้าวงจรบวก ส่วนสัญญาณที่จุด E และ G จะถูกเพิ่มค่าสุ่มตัวอย่างเช่นกัน จากนั้นจะถูกป้อนสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ ของสเตจที่ 2 ได้สัญญาณที่จุดก่อนเข้าวงจรบวก สัญญาณทั้งสี่จุดก่อนเข้าวงจรบวกจะถูกรวมเป็นสัญญาณที่จุด H และ I จากนั้นสัญญาณที่จุด H จะถูกเพิ่มค่าสุ่มตัวอย่างจากนั้นจะถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ ของสเตจที่ 1 ได้สัญญาณเอาต์พุตของสเตจที่ 1 ส่วนสัญญาณที่จุด I จะถูกเพิ่มค่าสุ่มตัวอย่างจากนั้นจะถูกป้อนสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ ของสเตจที่ 1 ได้สัญญาณเอาต์พุตของสเตจที่ 1 หลังจากนั้นสัญญาณเอาต์พุตของสเตจที่ 1 ฝั่งสังเคราะห์ จะถูกรวมเป็นสัญญาณเอาต์พุตสุดท้ายที่จุด J

จากการระบุตำแหน่งตามรูปที่ 4.34 บนบล็อกไดอะแกรมทำให้ได้ผลจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ตามจุดเหล่านั้นดังต่อไปนี้

1) สัญญาณภาพที่จุด A

ในการทดสอบการทำงานของฟิลเตอร์แบงก์สำหรับจุดนี้คือจะทำการป้อนสัญญาณอินพุตที่เป็นภาพให้กับฟิลเตอร์แบงก์ โดยใช้ภาพ lena ชนิดภาพสเกลสีเทา ขนาด 256 X 256 พิกเซล พิกเซลละ 8 บิต ได้ผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บดังรูปที่ 4.35 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์

ดังรูปที่ 4.36 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.37

รูปที่ 4.35 สัญญาณภาพที่จุด A จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโปรแกรมแมตแล็บ

รูปที่ 4.36 สัญญาณภาพที่จุด A จากการจำลองการทำงานของฮาร์ดแวร์ ด้วยโปรแกรมแมตแล็บ

รูปที่ 4.37 สัญญาณภาพที่จุด A จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

2) สัญญาณภาพที่จุด B

สัญญาณภาพที่จุด A จะถูกป้อนให้กับตัวกรองความถี่ต่ำผ่าน $H_0(z)$ ซึ่งจะทำการหาผลคูณของอินพุตกับสัมประสิทธิ์ของตัวกรอง (Convolution) จากนั้นจะถูกลดค่าสุ่มตัวอย่างลงสอง ได้สัญญาณภาพเอาต์พุตที่จุด B ซึ่งจะมีขนาดลดลงเหลืออยู่ที่ 256 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.38 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.39 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.40 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.38 สัญญาณภาพที่จุด B จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโปรแกรมแมตแล็บ

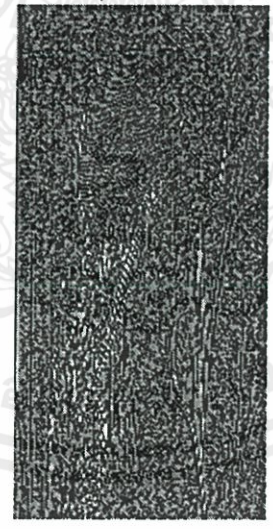
รูปที่ 4.39 สัญญาณภาพที่จุด B จากการจำลองการทำงานของฮาร์ดแวร์ ด้วยโปรแกรมแมตแล็บ

รูปที่ 4.40 สัญญาณภาพที่จุด B จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

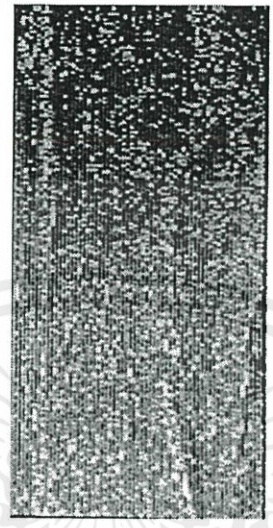
3) สัญญาณภาพที่จุด C

สัญญาณภาพที่จุด A จะถูกป้อนให้กับตัวกรองความถี่สูงผ่าน $H_1(z)$ ซึ่งจะทำการหาผลคูณของอินพุตกับสัมประสิทธิ์ของตัวกรอง จากนั้นจะถูกลดค่าสุ่มตัวอย่างลงสอง ได้สัญญาณภาพเอาต์พุตที่จุด C ซึ่งจะมีขนาดลดลงเหลืออยู่ที่ 256 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแมตแล็บดังรูปที่ 4.41 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.42 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.43 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.41 สัญญาณภาพที่จุด C จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโปรแกรมแมตแล็บ



รูปที่ 4.42 สัญญาณภาพที่จุด C จากการจำลองการทำงานของฮาร์ดแวร์ ด้วยโปรแกรมแมตแล็บ



รูปที่ 4.43 สัญญาณภาพที่จุด C จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

4) สัญญาณภาพที่จุด D

สัญญาณที่ภาพจุด B จะผ่านเข้าตัวกรองความถี่ต่ำผ่าน $H_0(z)$ และถูกลดค่าสุ่มตัวอย่างลงสองอีกครั้ง ได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด D ซึ่งจะมีขนาดลดลงเหลืออยู่ที่ 128 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.44 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.45 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.46 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.44 สัญญาณภาพที่จุด D จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโปรแกรมแมตแล็บ

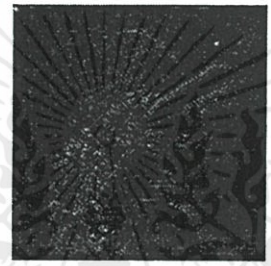
รูปที่ 4.45 สัญญาณภาพที่จุด D จากการจำลองการทำงานของฮาร์ดแวร์ ด้วยโปรแกรมแมตแล็บ

รูปที่ 4.46 สัญญาณภาพที่จุด D จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

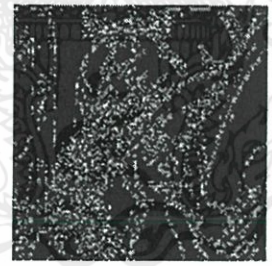
5) สัญญาณภาพที่จุด E

สัญญาณที่จุด B จะผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ จะถูกลดค่าสุ่มตัวอย่างลงสองอีกครั้ง ได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด E ซึ่งจะมีขนาดลดลงเหลืออยู่ที่ 128 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.47 สำหรับผล

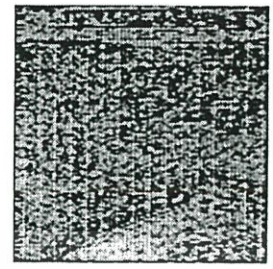
การจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.48 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.49 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์



รูปที่ 4.47 สัญญาณภาพที่จุด E จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติด้วยโปรแกรมแมตแล็บ



รูปที่ 4.48 สัญญาณภาพที่จุด E จากการจำลองการทำงานของฮาร์ดแวร์ด้วยโปรแกรมแมตแล็บ



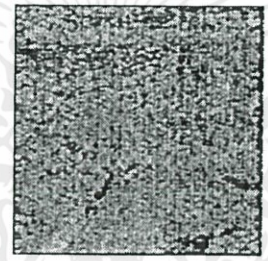
รูปที่ 4.49 สัญญาณภาพที่จุด E จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

6) สัญญาณภาพที่จุด F

สัญญาณภาพที่จุด C จะผ่านตัวกรองความถี่ต่ำผ่าน $H_0(z)$ และถูกลดค่าสุ่มตัวอย่างลงสองอีกครั้ง ได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด F ซึ่งจะมีขนาดลดลงเหลืออยู่ที่ 128 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.50 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.51 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.52 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.50 สัญญาณภาพที่จุด F จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ
ด้วยโปรแกรมแมตแล็บ

รูปที่ 4.51 สัญญาณภาพที่จุด F จากการจำลองการทำงานของฮาร์ดแวร์ด้วยโปรแกรมแมตแล็บ

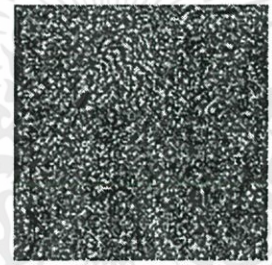


รูปที่ 4.52 สัญญาณภาพที่จุด F จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

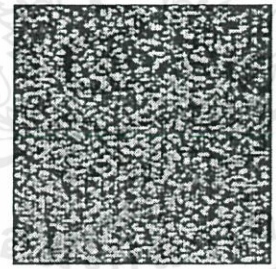
7) สัญญาณภาพที่จุด G

สัญญาณภาพที่จุด C เมื่อผ่านตัวกรองความถี่สูงผ่าน $H_1(z)$ จะถูกลดค่าสุ่มตัวอย่างลงสองอีกครั้ง ได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด G ซึ่งจะมีขนาดลดลงเหลืออยู่ที่ 128 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.53 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.54 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.55 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.53 สัญญาณภาพที่จุด G จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโปรแกรมแมตแล็บ



รูปที่ 4.54 สัญญาณภาพที่จุด G จากการจำลองการทำงานของฮาร์ดแวร์ ด้วยโปรแกรมแมตแล็บ



รูปที่ 4.55 สัญญาณภาพที่จุด G จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

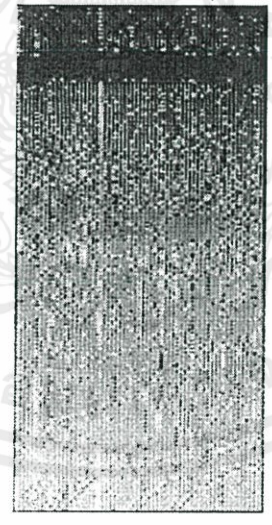
8) สัญญาณภาพที่จุด H

สัญญาณภาพที่จุด D เมื่อถูกเพิ่มค่าสุ่มตัวอย่างขึ้นสอง และถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ และสัญญาณภาพที่จุด E เมื่อถูกเพิ่มค่าสุ่มตัวอย่างขึ้นสอง และถูกป้อนเข้าสู่

ตัวกรองความถี่สูงผ่าน $F_1(z)$ รวมกันได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด H โดยจะมีขนาดเพิ่มขึ้นด้วยการแทรกพิกเซลสีดำ ซึ่งอยู่ที่ 256 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.56 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.57 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.58 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.56 สัญญาณภาพที่จุด H จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติด้วยโปรแกรมแมตแล็บ

รูปที่ 4.57 สัญญาณภาพที่จุด H จากการจำลองการทำงานของฮาร์ดแวร์
ด้วยโปรแกรมแมตแล็บ

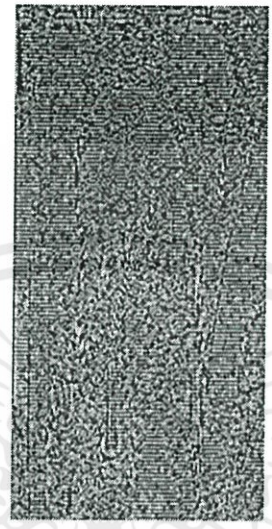


รูปที่ 4.58 สัญญาณภาพที่จุด H จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

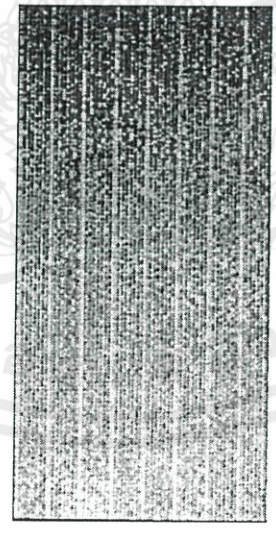
9) สัญญาณภาพที่จุด I

สัญญาณภาพที่จุด F เมื่อถูกเพิ่มค่าสุ่มตัวอย่างขึ้นสอง และถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ และสัญญาณภาพที่จุด G เมื่อถูกเพิ่มค่าสุ่มตัวอย่างขึ้นสอง และถูกป้อนเข้าสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ รวมกันได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด I โดยจะมีขนาดเพิ่มขึ้นด้วยการแทรกพิกเซลสีดำ ซึ่งอยู่ที่ 256 X 128 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.59 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.60 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.61 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.59 สัญญาณภาพที่จุด I จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโปรแกรมแมตแล็บ



รูปที่ 4.60 สัญญาณภาพที่จุด I จากการจำลองการทำงานของฮาร์ดแวร์
ด้วยโปรแกรมแมตแล็บ



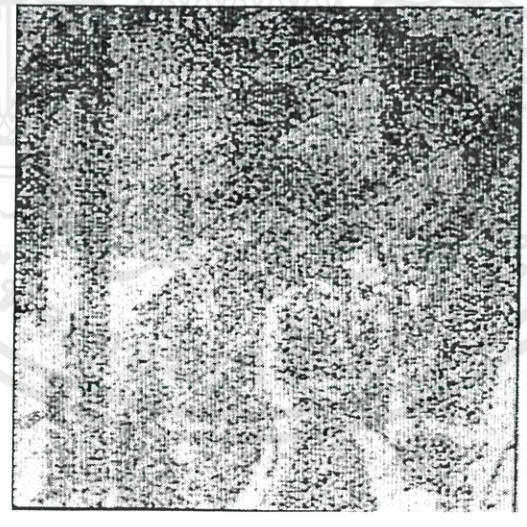
รูปที่ 4.61 สัญญาณภาพที่จุด I จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

10) สัญญาณภาพที่จุด J

สัญญาณภาพที่จุด H เมื่อถูกเพิ่มค่าสุ่มตัวอย่างขึ้นสอง และถูกป้อนเข้าสู่ตัวกรองความถี่ต่ำผ่าน $F_0(z)$ และสัญญาณภาพที่จุด I เมื่อถูกเพิ่มค่าสุ่มตัวอย่างขึ้นสอง และถูกป้อนเข้าสู่ตัวกรองความถี่สูงผ่าน $F_1(z)$ รวมกันได้สัญญาณเอาต์พุตเป็นสัญญาณภาพที่จุด J โดยจะมีขนาดเพิ่มขึ้นด้วยการแทรกพิกเซลสีดำ ซึ่งอยู่ที่ 256 X 256 พิกเซล พิกเซลละ 8 บิต มีผลการจำลองการทำงานด้วยโปรแกรมแม็ตแล็บดังรูปที่ 4.62 สำหรับผลการจำลองการทำงานของฮาร์ดแวร์จะทำการปรับค่าความละเอียดของการควอนไตซ์อยู่ที่ 4 บิตต่อพิกเซล ให้สอดคล้องกับการทำงานจริงที่จะเกิดขึ้น ทำให้ได้ผลการจำลองการทำงานของฮาร์ดแวร์ดังรูปที่ 4.63 ซึ่งจะได้ภาพที่มีการไล่ระดับสีเพียง 16 ระดับ และมีสัญญาณรบกวนเพิ่มขึ้นเนื่องจากคำนวณที่มีการสะสมของความผิดพลาด ผลการจำลองทั้งส่วนจะนำมาเปรียบเทียบกับผลการทำงานจริงตามรูปที่ 4.64 ซึ่งจะพบว่าภาพเอาต์พุตที่ได้สอดคล้องกับสมมติฐานที่ได้จากการจำลองการทำงานของฮาร์ดแวร์

รูปที่ 4.62 สัญญาณภาพที่จุด J จากการจำลองการทำงานฟิลเตอร์แบงก์แบบ 2 มิติ
ด้วยโปรแกรมแมตแล็บ

รูปที่ 4.63 สัญญาณภาพที่จุด J จากการจำลองการทำงานของฮาร์ดแวร์
ด้วยโปรแกรมแมตแล็บ



รูปที่ 4.64 สัญญาณภาพที่จุด J จากการทำงานจริงของฟิลเตอร์แบงก์แบบ 2 มิติ

## 4.4 ผลการจำลองฟิลเตอร์แบงก์หลายสเตจ

เนื่องจากปริญญานิพนธ์นี้สร้างฮาร์ดแวร์ของฟิลเตอร์แบงก์แบบ 2 มิติ ด้วยโครงสร้างของฟิลเตอร์แบงก์แบบ 2 ระดับ เพื่อทำการประมวลผลภาพการประมวลผลระดับ 1 เพียงอย่างเดียว ทางผู้จัดทำจึงได้จำลองผลที่คาดว่าจะได้ทางทฤษฎีหากใช้โครงสร้างของฟิลเตอร์ที่มีสเตจการทำงานมากขึ้น โดยจะจำลองจนถึงการประมวลผลระดับ 3 แล้วนำผลที่ได้มาจัดเรียงเพื่อเทียบความแตกต่างกับการประมวลผลระดับ 1 ที่เป็นพื้นฐาน โดยแสดงผลเป็นภาพผลลัพธ์สุทธิของฝั่งวิเคราะห์ ดังรูปที่ 3.65

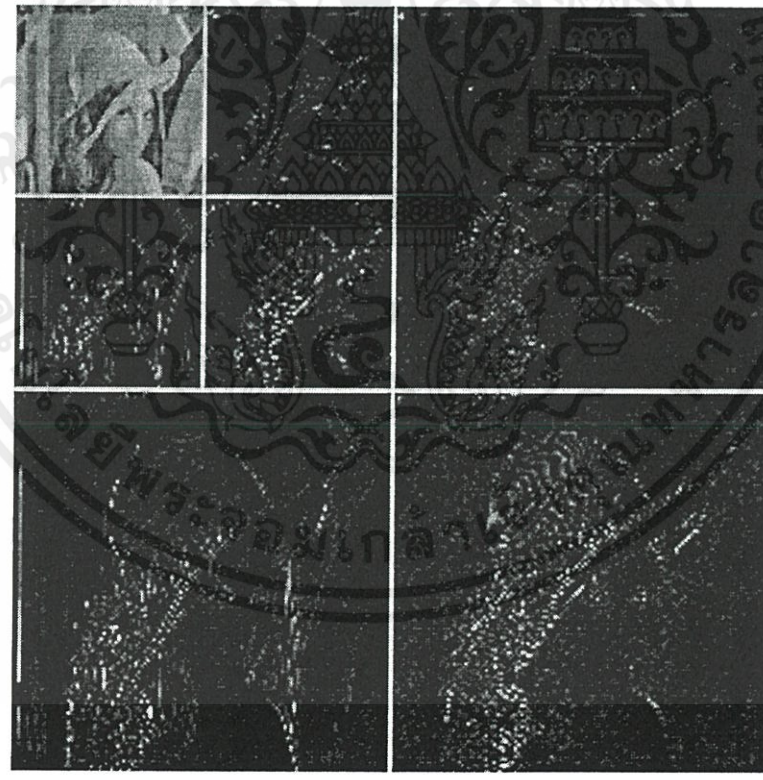
รูปที่ 4.65 ผลผลลัพธ์สุทธิของฝั่งวิเคราะห์ของการประมวลผลระดับ 1

ในการออกแบบให้ฟิลเตอร์แบงก์ทำการประมวลผลในระดับที่มากขึ้นจะทำให้โครงสร้างของฟิลเตอร์แบงก์มีสเตจที่เพิ่มขึ้นด้วย โดยจะมีความสัมพันธ์กันคือ ถ้าการประมวลผลระดับ $l$ ที่เพิ่มขึ้นจะทำให้จำนวนสเตจ $S$ เพิ่มขึ้นด้วยอัตรา $2l$ เฉพาะในกรณีที่ออกแบบโดยใช้

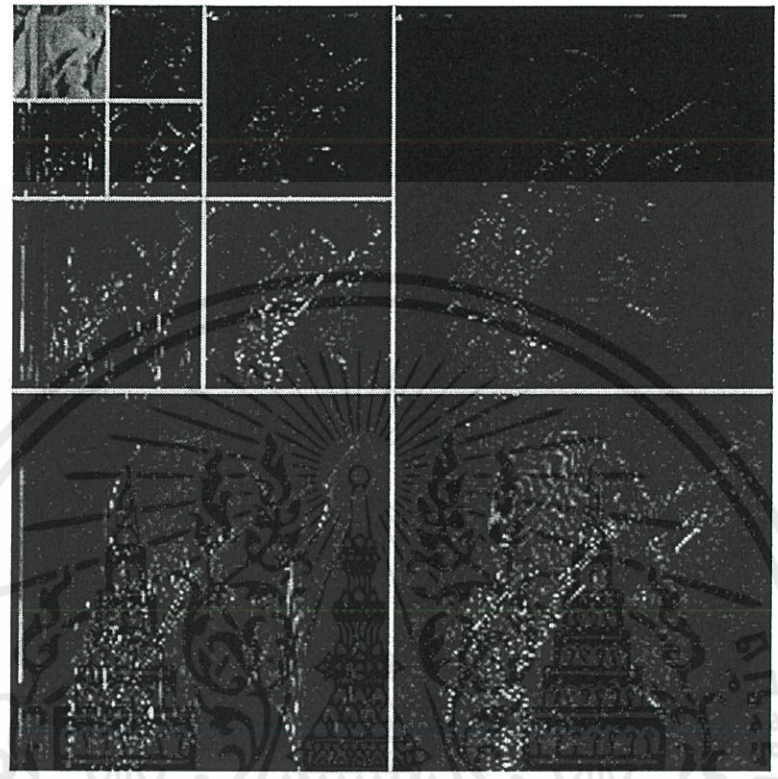
หลักการแยกการทำคอนโวลูชั่นแบบ 2 มิติ ด้วยความสัมพันธ์นี้สามารถเขียนสมการแทนได้ดังสมการที่ 4.1

$$S = 2I \tag{4.1}$$

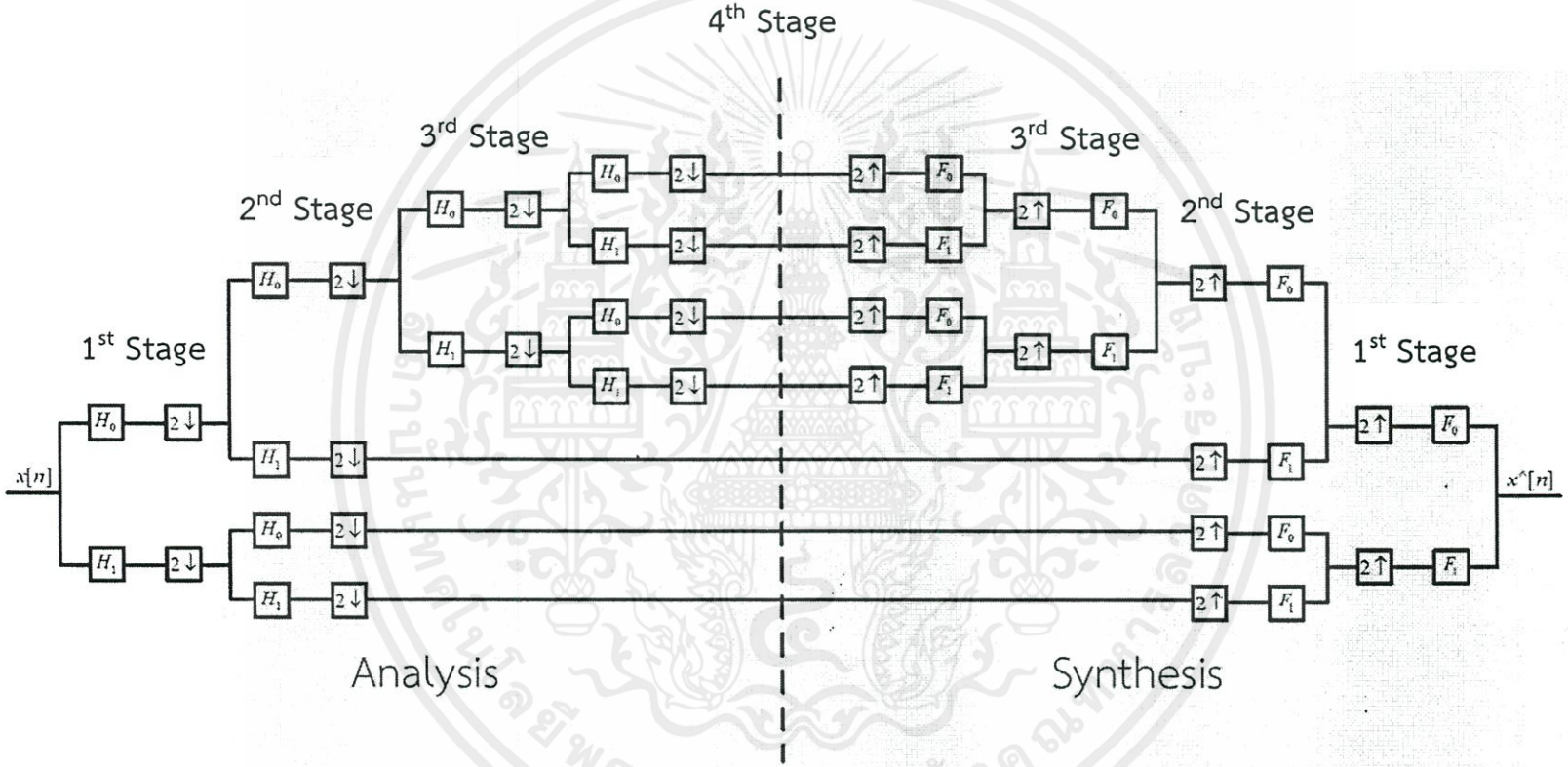
ดังนั้นจะได้ว่าถ้าต้องการประมวลผลระดับ 2 จากสมการที่ 4.1 ต้องออกแบบให้เป็นโครงสร้างของฟิลเตอร์แบงก์แบบ 4 ระดับ เช่นเดียวกันสำหรับการประมวลผลระดับ 3 ต้องออกแบบให้เป็นโครงสร้างของฟิลเตอร์แบงก์แบบ 6 ระดับ ซึ่งมีผลการจำลองภาพผลลัพธ์สุทธิของฝั่งวิเคราะห์ของการประมวลผลระดับ 2 และการประมวลผลระดับ 3 ดังรูปที่ 3.66 และ 3.67 ตามลำดับ และมีลักษณะโครงสร้างของฟิลเตอร์แบงก์แบบ 4 ระดับ และของฟิลเตอร์แบงก์แบบ 6 ระดับ แสดงได้ดังรูปที่ 3.68 และ 3.69 ตามลำดับ



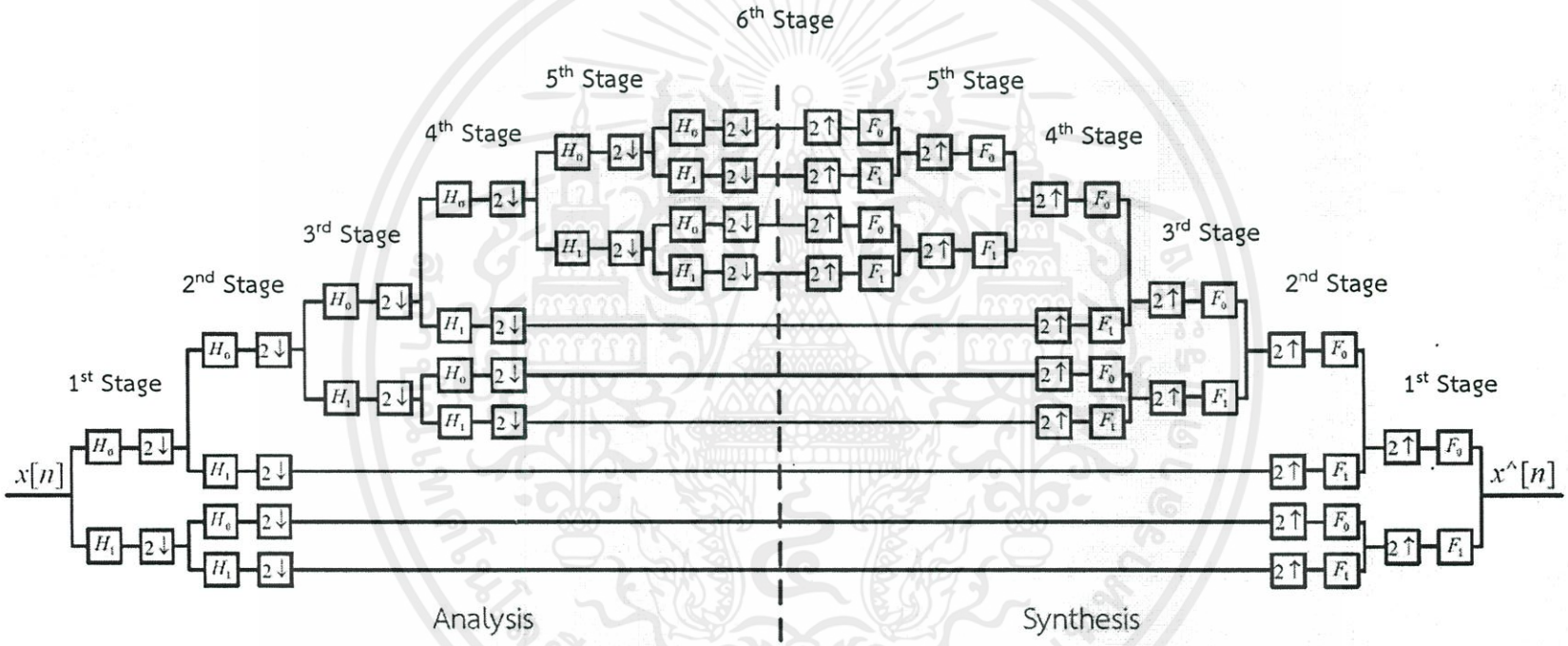
รูปที่ 4.66 ผลผลลัพธ์สุทธิของฝั่งวิเคราะห์ของการประมวลผลระดับ 2



รูปที่ 4.67 ผลผลลัพธ์สุทธิของฝั่งวิเคราะห์ของการประมวลผลระดับ 3




รูปที่ 3.68 โครงสร้างของฟิลเตอร์แบงก์แบบ 4 ระดับ




รูปที่ 3.69 โครงสร้างของฟิลเตอร์แบงก์แบบ 6 ระดับ

# บทที่ 5

# สรุปผลและข้อเสนอแนะ

## 5.1 สรุปผล

ปริญญานิพนธ์นี้นำเสนอการสร้างฟิลเตอร์แบงก์เพื่อใช้ในการประมวลผลสัญญาณภาพ โดยการออกแบบจะใช้วงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดที่ออกแบบด้วยโครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู ซึ่งโครงสร้างนี้ในการออกแบบทำให้โครงสร้างทางฮาร์ดแวร์ที่นำเสนอจะมีคุณสมบัติปราศจากตัวคูณ และยังปราศจากหน่วยความจำ นอกจากนี้ในการเปลี่ยนค่าสัมประสิทธิ์ตัวกรองก็สามารถทำได้โดยไม่ซับซ้อน ทำให้ง่ายในการประยุกต์ใช้กับการออกแบบและสร้างตัวกรองความถี่อันดับสูงขึ้น

ฟิลเตอร์แบงก์ที่นำเสนอนี้มีสองส่วนคือ ฟิลเตอร์แบงก์แบบ 1 มิติ ที่สามารถทำงานได้ที่ความถี่สัญญาณนาฬิกาสูงสุดที่ 299.892 MHz และฟิลเตอร์แบงก์แบบ 2 มิติ ที่สามารถทำงานได้ที่ความถี่สัญญาณนาฬิกาสูงสุดที่ 380.373 MHz และทั้งสองแบบใช้ทรัพยากรไม่ถึงครึ่งหนึ่งของบอร์ด นอกจากนี้เมื่อเพิ่มจำนวนอุปกรณ์ที่ต้องใช้งานสำหรับฟิลเตอร์แบงก์แบบ 2 มิติ จะพบว่าไม่ได้เพิ่มขึ้นเป็นทวีคูณ เนื่องจากประโยชน์ที่ได้จากการใช้โครงสร้างดังกล่าว

ในส่วนของผลการทดลองของฟิลเตอร์แบงก์แบบ 1 มิติ จากผลการจำลองการทำงานของฮาร์ดแวร์ที่ออกแบบบนโปรแกรม ModelSim และผลการจำลองการทำงานบนโปรแกรมแมตแล็บเปรียบเทียบกับผลการทำงานจริงของวงจรพบว่าโครงสร้างฟิลเตอร์แบงก์แบบ 1 มิติที่นำเสนอทำงานได้อย่างถูกต้อง โดยมีสัญญาณเอาต์พุตที่ได้มีรูปร่างเหมือนกับสัญญาณอินพุตที่ป้อนเข้าไป ซึ่งสอดคล้องและเป็นไปตามเงื่อนไขการกู้กลับคืนอย่างสมบูรณ์ตามคุณสมบัติของฟิลเตอร์แบงก์

ในส่วนของผลการทดลองของฟิลเตอร์แบงก์แบบ 2 มิติ จากผลการจำลองการทำงานบนโปรแกรมแมตแล็บ และผลการจำลองการทำงานของฮาร์ดแวร์ที่ออกแบบบนโปรแกรม ModelSim เปรียบเทียบกับผลการทำงานจริงของวงจรพบว่าโครงสร้างฟิลเตอร์แบงก์แบบ 2 มิติที่นำเสนอทำงานได้อย่างถูกต้อง โดยมีสัญญาณภาพเอาต์พุตที่ได้มีรูปร่างสอดคล้องกับสัญญาณภาพอินพุตที่ป้อนเข้าไปและเป็นไปตามเงื่อนไขการกู้กลับคืนอย่างสมบูรณ์ตามคุณสมบัติของฟิลเตอร์แบงก์ แต่ภาพเอาต์พุตที่ได้ไม่เหมือนอย่างสมบูรณ์ เนื่องจากเกิดการสะสมความผิดพลาดของการควอนไตซ์ และในการคำนวณบนวงจรจริงมีค่าความละเอียดการควอนไตซ์น้อย เป็นเหตุที่เกิดจากเรื่องของข้อจำกัดของฮาร์ดแวร์

ในส่วนของการจำลองฟิลเตอร์แบงก์หลายสเตจ เป็นการพิสูจน์ผลการออกแบบตามทฤษฎีที่แสดงให้เห็นว่าฟิลเตอร์แบงก์สามารถถูกนำไปใช้งานประยุกต์ด้านการประมวลผลภาพได้หลากหลาย เช่น การแปลงเวฟเล็ต และการบีบอัดข้อมูล ก่อนนำไปเก็บในหน่วยความจำที่มีอย่างจำกัด อีกทั้งในการทดลองในส่วนฟิลเตอร์แบงก์แบบ 2 มิติ และการจำลองฟิลเตอร์แบงก์หลายสเตจได้ใช้หลักการการแปลงแยกแบบ 2 มิติ ซึ่งจะพบว่าให้เอาต์พุตที่ตรงกับการแปลงแบบ 2 มิติทั่วไป ทำให้โครงสร้างที่ใช้การแปลงแยกแบบ 2 มิติเป็นที่น่าสนใจในการนำไปประยุกต์ในระบบอื่น เพราะมีการคำนวณที่ไม่ซับซ้อนและออกแบบสร้างเป็นฮาร์ดแวร์ได้ง่ายกว่า

## 5.2 ข้อเสนอแนะ

ฟิลเตอร์แบงก์นั้นสามารถนำไปประยุกต์ใช้ในการประมวลผลสัญญาณต่างๆ ทั้งสัญญาณ 1 มิติ และ สัญญาณ 2 มิติ ที่สามารถมีค่าผิดพลาดในระดับที่ยอมรับได้ (Lossy) ได้ โดยผู้ที่สนใจต้องการนำไปใช้สามารถนำโครงสร้างฟิลเตอร์แบงก์นี้ไปปรับเปลี่ยนเพื่อให้เหมาะสมกับงาน เช่นเปลี่ยนอับดับตัวกรอง หรือเปลี่ยนค่าสัมประสิทธิ์ของตัวกรองตามที่ต้องการใช้ แต่ยังคงโครงสร้างหลักเดิมไว้ก็สามารถทำได้โดยง่าย

ถึงแม้ว่าโครงสร้างฟิลเตอร์แบงก์ที่นำเสนอจะมีโครงสร้างที่มีความซับซ้อนน้อยในระดับหนึ่งตามโครงสร้างของเลขคณิตกระจายแบบปราศจากตารางเปิดดู แต่ก็ยังคงมีการใช้ทรัพยากร หรือจำนวนอุปกรณ์ในการสร้างบน FPGA ค่อนข้างสูงในบางส่วน ซึ่งปริญญานิพนธ์นี้ยังไม่ได้นำเสนอการแก้ไขปัญหาดังกล่าว

สำหรับการแก้ปัญหาเรื่องของข้อจำกัดของฮาร์ดแวร์ เนื่องจากในปริญญานิพนธ์นี้ใช้วงจรแปลงสัญญาณเชิงเลขเป็นอุปมานสำหรับภาพที่อยู่บน FPGA ซึ่งจำกัดค่าข้อมูลอยู่ที่ 4 บิตต่อพิกเซล ทำให้เกิดความผิดพลาดของการควอนไตซ์สะสมในการคำนวณ และค่าความละเอียดของการควอนไตซ์ต่ำ ทางผู้จัดทำจึงเสนอแนะว่าหากต้องการจะลดปัญหาเพื่อให้ได้ภาพเอาต์พุตที่สมบูรณ์มากขึ้น ควรใช้วงจรแปลงสัญญาณเชิงเลขเป็นอุปมานสำหรับภาพเป็นอุปกรณ์แยกออกมาจากบอร์ด และควรมีค่าข้อมูลอยู่ที่ 8 บิตต่อพิกเซล ซึ่งคาดว่าจะช่วยเพิ่มประสิทธิภาพให้กับฟิลเตอร์แบงก์แบบ 2 มิติที่ออกแบบได้ดีขึ้น

# บรรณานุกรม


[1] Mitra, Sanjit K. *Digital Signal Processing A Computer-Based Approach.* 4rd ed. Singapore : Mc Graw Hill, 2011.

[2] Vaidyanathan, P.P. *Multirate Systems and Filter Banks.* the United States of America. : Prentice Hall, 1993.

[3] Sayood, Khalid. *Introduction to Data Compression.* 4rd ed. the United States of America, MA. : Morgan Kaufmanm, 2012.

[4] Strang, Gilbert, and Nguyen, Truong. *Wavelets and Filter Banks.* the United States of America, MA. : Wellesley-Cambridge Press, 1996.

[5] Acharya, Tinku, and Tsai, Ping-Sing. *JPEG2000 Standard for Image Compression.* the United States of America. : John Wiley & Sons, 2005.

[6] Yoo, Heejong, and Anderson, David V. *"Hardware-Efficient Distributed Arithmetic Architecture for High-Order Digital Filters." IEEE* (2005) : V-125 - V-128.

[7] ชำนาญ ปัญญาใส, และวัชรากร หนูทอง. *ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล.* กรุงเทพฯ : ซีเอ็ดยูเคชั่น, 2547.

[8] ศรวัฒน์ ชิวปรีชา, "การออกแบบวงจรกรองสัญญาณเชิงเลขที่ใช้โครงสร้างเลขคณิตกระจายโดยการแทนด้วยปริภูมิสเตท". วิทยานิพนธ์ปริญญาวิศวกรรมศาสตรมหาบัณฑิต, สาขาวิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2545

[9] ศุภศิลป์ คำสร้อย. "การสร้างฟิลเตอร์แบงค์สำหรับตัวแปลงเวฟเล็ทแบบไม่ต่อเนื่องบนพื้นฐานตัวกรองไบโนเมียลและโครงสร้างเลขคณิตกระจายแบบกระจายค่าสัมประสิทธิ์." วิทยานิพนธ์ปริญญาวิทยาศาสตรมหาบัณฑิต, สาขาวิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2556.

[10] อนิรุทธิ์ ตระกูลไตรตรึง. "การสร้างวงจรกรองเชิงเลขปรับตัวได้แบบค่าเฉลี่ยกำลังสองน้อยที่สุดโดยใช้โครงสร้างเลขคณิตกระจายซึ่งปราศจากตารางเปิดดู." วิทยานิพนธ์ปริญญาวิทยาศาสตรมหาบัณฑิต, สาขาวิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2556.

[11] "Digilent Atlys Spartan 6 FPGA Development Kit". http://www.robotshop.com/digilent-atlys-spartan-6-fpga-development-kit.html.

# ภาคผนวก ก

# โปรแกรมแมตแล็บ

## โปรแกรมแปลงเลขฐานสองแบบ 2's complement

```
clc;
clear all;
N=7;
h=[-0.0912717631142495,-
0.057543526228500,0.591271763114247,1.115087052456994,0.591271763114247,-
0.057543526228500,-0.0912717631142495];
hd=round((h.*((2^4)-1))/(max(h)));

% Covert into 2's complement binary syntem
for i=1:N;
    if hd(i)>=0;
        hb=dec2bin(hd(i),7);
    else hd(i)<0;
        hb=dec2bin(mod((hd(i)),2^7),7);
    end;
    hbs(i)=cellstr(hb);
    i=i+1;
end;
```

## โปรแกรมแปลงจำลองการทำงานของฟิลเตอร์แบงก์แบบ 1 มิติ

```
clc;
clear all;

% Lowpasss Filter Coefficient in Analysis part
h0=[0.026748757410810,-0.016864118442875,-
0.078223266528988,0.266864118442872,0.602949018236358,0.266864118442872,-
0.078223266528988,-0.016864118442875,0.026748757410810];

% Highpasss Filter Coefficient in Analysis part
h1=[0.0912717631142495,-0.057543526228500,-0.591271763114247,1.115087052456994,-
0.591271763114247,-0.057543526228500,0.0912717631142495];

% Lowpasss Filter Coefficient in Synthesis part
f0=[-0.0912717631142495,-
0.057543526228500,0.591271763114247,1.115087052456994,0.591271763114247,-
0.057543526228500,-0.091271763114249];

% Highpasss Filter Coefficient in Synthesis part
f1=[0.026748757410810,0.016864118442875,-0.078223266528988,-
0.266864118442872,0.602949018236358,-0.266864118442872,-
0.078223266528988,0.016864118442875,0.026748757410810];

Fs=20000;    %Sampling Frequency

% Sine wave Gennerater
f3=200;
f4=8000;
N_point=200;                    % Determine point for ploting
t=[0:1:N_point]/(Fs);           % Determine time vector
xx=sin(2*pi*f3*t);
xxx=sin(2*pi*f4*t);
x=(sin(2*pi*f3*t))+(sin(2*pi*f4*t));          % Generate input signal
%-------------------------------------------------

%-----Analysis Part-----
% Lowpass Filter in Analysis
```

```
x1=filter(h0,1,x);
%Down Sampling of Lowpass Filter in Analysis
x11=x1+(-1*min(x1));
y1=downsample(x11,2);

% Highpass Filter in Analysis
x2=filter(h1,1,x);
%Down Sampling of Highpass Filter in Analysis
y2=downsample(x2,2);
%-----------------------------------------------------

%-----Synthesis Part----
%Up Sampling of F0
u1=upsample(y1,2);
% Lowpass filter of Synthesis
v1=filter(f0,1,u1);

%Up Sampling of F1
u2=upsample(y2,2);
% High filter of Synthesis
v2=filter(f1,1,u2);


%----Display Result----
%plot in analysis part
% Plot input
figure(1);
subplot(5,1,1);
plot(x);grid on;
xlabel('(a) Composite input signal');

axis([0 N_point -2 2]);

% Plot LPF Output
figure(1);subplot(5,1,2);
plot(x1);grid on;
xlabel('(b) Analysis Lowpass Output');

axis([0 N_point -2 2]);

% Plot LPF + Downsampling Output

figure(1);subplot(5,1,3);
plot(y1);grid;
xlabel('(c) Analysis Lowpass Output + Downsampling');
ylabel('Amplitude');
axis([0 N_point/2 -2 2]);

% Plot HPF Output
figure(1);subplot(5,1,4);
plot(x2);grid;
xlabel('(d) Analysis Highpass Output');

axis([0 N_point -2 2]);

%Plot HPF + Downsampling Output
% t1=[0:N_point/2]/(Fs);
figure(1);subplot(5,1,5);
plot(y2);grid;
xlabel('(e) Analysis Highpass Output + Downsampling');

axis([0 N_point/2 -2 2]);

%-----------------------------------------------------

%Plot Synthesis Part
```

```
%Plot Upsampling in LPF Synthesis
% yuu=yu(1:N_point+1);
figure(2);subplot(5,1,1);
plot(u1);grid;
xlabel('(a) Upsampling before LPF in Synthesis Part');

axis([0 N_point -2 2]);

%Plot LPF Synthesis
figure(2);subplot(5,1,2);
plot(v1);grid;
xlabel('(b) Synthesis Lowpass Output ');

axis([0 N_point -2 2]);

%Plot Upsampling HPF
% yuu1=yu1(1:N_point+1);
figure(2);subplot(5,1,3);
plot(u2);grid;
xlabel('(c) Upsampling before HPF in Synthesis Part ');
ylabel('Amplitude');
axis([0 N_point -2 2]);

%Plot HPF Synthesis
figure(2);subplot(5,1,4);
plot(v2);grid;
xlabel('(d) Synthesis Highpass Output');

axis([0 N_point -2 2]);

%Plot Filter Banks Output
data=v1+v2;
figure(2);subplot(5,1,5);
plot(data);grid;
xlabel('(e) Composite Output Signal)');

axis([0 N_point -2 2]);


%Result of A point
figure(3);
subplot(3,1,1);
plot(xx);grid on;
title('(a) Low Frequency');
axis([0 N_point -2 2]);

subplot(3,1,2);
plot(xxx);grid on;
title('(b) High Frequency');ylabel('Amplitude');
axis([0 N_point -2 2]);

subplot(3,1,3);
plot(x);grid on;
title('(c) Composite Input Signal');
xlabel('Number of Sample');
axis([0 N_point -2 2]);

%Result of B point
figure(4);
subplot(2,1,1);
plot(x);grid on;
title('(a) Composite Input Signal');
axis([0 N_point -4 4]);

subplot(2,1,2);
plot(x1);grid on;
title('(b) Output H0');xlabel('Number of Sample');ylabel('Amplitude');
```

```
axis([0 N_point -2 2]);

%Result of C Point
figure(5);
subplot(2,1,1);
plot(x);grid on;
title('(a) Composite Input Signal');
axis([0 N_point -4 4]);

subplot(2,1,2);
plot(x2);grid on;
title('(b) Output H1'),xlabel('Number of Sample'),ylabel('Amplitude');
axis([0 N_point -4 4]);

%Result of D point
figure(6);
subplot(2,1,1);
plot(x1);grid on;
title('(a) Output H0');
axis([0 N_point -2 2]);

subplot(2,1,2);
plot(y1);grid on;
title('(b) Output Analysis Part Low-pass Branch');xlabel('Number of
Sample');ylabel('Amplitude');
axis([0 N_point/2 -2 2]);

%Result of E point
figure(7);
subplot(2,1,1);
plot(x2);grid on;
title('(a) Output H1');
axis([0 N_point -2 2]);

subplot(2,1,2);
plot(y2);grid on;
title('(b) Output Analysis Part High-pass Branch');xlabel('Number of
Sample');ylabel('Amplitude');
axis([0 N_point/2 -2 2]);

%Result of F point
figure(8);
subplot(2,1,1);
y1=y1+(-1*min(y1));
plot(y1);grid on;
title('(a) Output Analysis Part Low-pass Branch');
axis([0 N_point/2 -1 3]);

subplot(2,1,2);
u1=u1+(-1*min(u1));
plot(u1);grid on;
title('(b) Upsampling');xlabel('Number of Sample');ylabel('Amplitude');
axis([0 N_point -1 3]);

%Result of G point
figure(9);
subplot(2,1,1);
plot(y2);grid on;
title('(a) Output Analysis Part High-pass Branch');
axis([0 N_point/2 -4 4]);

subplot(2,1,2);
plot(u2);grid on;
title('(b) Upsampling');xlabel('Number of Sample');ylabel('Amplitude');
axis([0 N_point -4 4]);
%Result of H point
```

```
figure(10);
subplot(2,1,1);
plot(u1);grid on;
title('(a) Output Low-pass Branch Expander');
axis([0 N_point -1 3]);

subplot(2,1,2);
plot(v1);grid on;
title('(b) Output F0');xlabel('Number of Sample');ylabel('Amplitude');
axis([0 N_point -1 3]);

%Result of I point
figure(11);
subplot(2,1,1);
plot(u2);grid on;
title('(a) Output High-pass Branch Expander');
axis([0 N_point -2 2]);

subplot(2,1,2);
plot(v2);grid on;
title('(b) Output F1');xlabel('Number of Sample');ylabel('Amplitude');
axis([0 N_point -2 2]);

%Result of J point
figure(12);
subplot(3,1,1);
plot(v1);grid on;
title('(a) Output F0')
axis([0 N_point -2 2]);

subplot(3,1,2);
plot(v2);grid on;
title('(b) Output F1');ylabel('Amplitude');
axis([0 N_point -2 2]);

subplot(3,1,3);
plot(data);grid on;
title('(c) Sum Output Signal');xlabel('Number of Sample');
axis([0 N_point -2 2]);

figure(13);
N1=0:8;
subplot(4,1,1);
stem(N1,h0);grid on;
title('Coefficient of Low-pass filter in analysis part');

N2=0:6;
subplot(4,1,2);
stem(N2,h1);grid on;
title('Coefficient of High-pass filter in analysis part');

subplot(4,1,3);
stem(N2,f0);grid on;
title('Coefficient of Low-pass filter in synthesis part');

subplot(4,1,4);
stem(N1,f1);grid on;
title('Coefficient of High-pass filter in synthesis part');
```

# โปรแกรมจำลองการทำงานของฟิลเตอร์แบงก์แบบ 2 มิติ

```
clc;
clear all;
% close all;

%% Get image
% get a 2-D uint8 image (size 512x512)
Xread = imread('lena_gray_256.tif');
% uint8 to Double Converter
X = im2double(Xread);
% get size of image
[z,z1]=size(X);
% uint8 to Double Converter
X = im2double(Xread);
col = zeros(256,10);
X1 = horzcat(X,col);

%% Reshape image to 1D
b = reshape(X',1,(256*256));

%% Filter Coefficient
% Analysis LPF Using CDF9/7 as Filter Coefficient
    h0=[0.026748757410810,-0.016864118442875,-
0.078223266528988,0.266864118442872,0.602949018236358,0.266864118442872,-
0.078223266528988,-0.016864118442875,0.026748757410810];

% Analysis HPF Using CDF9/7 as Filter Coefficient
    h1=[0.0912717631142495,-0.057543526228500,-
0.591271763114247,1.115087052456994,-0.591271763114247,-
0.057543526228500,0.0912717631142495];

% Synthesis LPF Using CDF9/7 as Filter Coefficient
    f0=[-0.0912717631142495,-
0.057543526228500,0.591271763114247,1.115087052456994,0.591271763114247,-
0.057543526228500,-0.0912717631142495];

% Synthesis HPF Using CDF9/7 as Filter Coefficient
    f1=[0.026748757410810,0.016864118442875,-0.078223266528988,-
0.266864118442872,0.602949018236358,-0.266864118442872,-
0.078223266528988,0.016864118442875,0.026748757410810];

%% Low - branch
%% LPF #1 (Low-branch)
L=filter(h0,1,b);
% L1 = L(10:262153);

% Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:z
      aL(i,j) = L(m);
      m=m+1;
   end
end

figure(1);
subplot(1,2,1);imshow(Xread);title('Input Image');
subplot(1,2,2);imshow(aL);title('LPF output Image');

%% LPF Downsampling #1 (Low-branch)
bbL = downsample(L,2);

%Reshape data from 1D to 2D
```

```
m =1;
for i=1:z
    for j=1:(z)/2
       bL(i,j) = bbL(m);
       m=m+1;
    end
end
figure(2);
subplot(1,2,1);imshow(aL);title('LPF output Image');
subplot(1,2,2);imshow(bL);title('LPF and Downsampling output Image');

%% Transpose
bLt = bL';

%% Reshape image to 1D
bLr = reshape(bLt',1,((z)/2)*z);

%% LPF #2 (Low-branch)
ccLL = filter(h0,1,bLr);
%Reshape data from 1D to 2D
m =1;
for i=1:z
    for j=1:((z)/2)
       cLL(i,j) = ccLL(m);
       m=m+1;
    end
end
figure(3);
subplot(1,2,1);imshow(bL);title('LPF and Downsampling output Image');
subplot(1,2,2);imshow(cLL);title('LPF #2 output Image');

%% LPF Downsampling #2 (Low-branch)
dLL = downsample(ccLL,2);

%Reshape data from 1D to 2D
m =1;
for i=1:((z)/2)
    for j=1:((z)/2)
       LL(j,i) = dLL(m);
       m=m+1;
    end
end
figure(4);
subplot(1,2,1);imshow(cLL);title('LPF #2 output Image');
subplot(1,2,2);imshow(LL);title('LPF and Downsampling #2 output Image');

%% HPF #2 (Low-branch)
ccLL = filter(h0,1,bLr);
%Reshape data from 1D to 2D
m =1;
for i=1:z
    for j=1:((z)/2)
       cLL(i,j) = ccLL(m);
       m=m+1;
    end
end
figure(5);
subplot(1,2,1);imshow(bL);title('LPF and Downsampling output Image');
subplot(1,2,2);imshow(cLL);title('LPF #2 output Image');

%% LPF Downsampling #2 (Low-branch)
dLL = downsample(ccLL,2);

%Reshape data from 1D to 2D
m =1;
for i=1:((z)/2)
    for j=1:((z)/2)
```

```
      LL(j,i) = dLL(m);
      m=m+1;
   end
end
figure(6);
subplot(1,2,1);imshow(cLL);title('LPF #2 output Image');
subplot(1,2,2);imshow(LL);title('LPF and Downsampling #2 output Image');

%% HPF #2 (Low-branch)
ccLH = filter(h1,1,bLr);
%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:((z)/2)
      cLH(i,j) = ccLH(m);
      m=m+1;
   end
end
figure(7);
subplot(1,2,1);imshow(bL);title('LPF and Downsampling output Image');
subplot(1,2,2);imshow(cLH);title('HPF #2 output Image');

%%  HPF Downsampling #2 (Low-branch)
dLH = downsample(ccLH,2);

%Reshape data from 1D to 2D
m =1;
for i=1:((z)/2)
   for j=1:((z)/2)
      LH(j,i) = dLH(m);
      m=m+1;
   end
end
figure(8);
subplot(1,2,1);imshow(cLH);title('HPF #2 output Image');
subplot(1,2,2);imshow(LH);title('HPF and Downsampling #2 output Image');

%% High-branch
%% HPF #1 (High-branch)
H=filter(h1,1,b);

% Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:z
      aH(i,j) = H(m);
      m=m+1;
   end
end

figure(9);
subplot(1,2,1);imshow(Xread);title('Input Image');
subplot(1,2,2);imshow(aH);title('HPF output Image');

%% LPF Downsampling #1 (High-branch)
bbH = downsample(H,2);

%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:((z)/2)
      bH(i,j) = bbH(m);
      m=m+1;
   end
end
figure(10);
subplot(1,2,1);imshow(aH);title('HPF output Image');
```

```
subplot(1,2,2);imshow(bH);title('HPF and Downsampling output Image');

%% Transpose
bHt = bH';

%% Reshape image to 1D
bHr = reshape(bHt',1,((z)/2)*z);

%% LPF #2 (High-branch)
ccHL = filter(h0,1,bHr);
%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:((z)/2)
      cHL(i,j) = ccHL(m);
      m=m+1;
   end
end
figure(11);
subplot(1,2,1);imshow(bH);title('HPF and Downsampling output Image');
subplot(1,2,2);imshow(cHL);title('LPF #2 output Image');

%% LPF Downsampling #2 (High-branch)
dHL = downsample(ccHL,2);

%Reshape data from 1D to 2D
m =1;
for i=1:((z)/2)
   for j=1:((z)/2)
      HL(j,i) = dHL(m);
      m=m+1;
   end
end
figure(12);
subplot(1,2,1);imshow(cHL);title('LPF #2 output Image');
subplot(1,2,2);imshow(HL);title('LPF and Downsampling #2 output Image');

%% LPF #2 (High-branch)
ccHL = filter(h0,1,bHr);
%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:((z)/2)
      cHL(i,j) = ccHL(m);
      m=m+1;
   end
end
figure(13);
subplot(1,2,1);imshow(bH);title('LPF and Downsampling output Image');
subplot(1,2,2);imshow(cHL);title('LPF #2 output Image');

%% LPF Downsampling #2 (High-branch)
dHL = downsample(ccHL,2);

%Reshape data from 1D to 2D
m =1;
for i=1:((z)/2)
   for j=1:((z)/2)
      HL(j,i) = dHL(m);
      m=m+1;
   end
end
figure(14);
subplot(1,2,1);imshow(cHL);title('HPF #2 output Image');
subplot(1,2,2);imshow(HL);title('HPF and Downsampling #2 output Image');
%% HPF #2 (High-branch)
```

```
ccHH = filter(h1,1,bHr);
%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:((z)/2)
      cHH(i,j) = ccHH(m);
      m=m+1;
   end
end
figure(15);
subplot(1,2,1);imshow(bH);title('LPF and Downsampling output Image');
subplot(1,2,2);imshow(cHH);title('HPF #2 output Image');

%%  HPF Downsampling #2 (High-branch)
dHH = downsample(ccHH,2);

%Reshape data from 1D to 2D
m =1;
for i=1:((z)/2)
   for j=1:((z)/2)
      HH(j,i) = dHH(m);
      m=m+1;
   end
end
figure(16);
subplot(1,2,1);imshow(cHH);title('HPF #2 output Image');
subplot(1,2,2);imshow(HH);title('HPF and Downsampling #2 output Image');

%% Show image LL LH HL HH
figure(17);
subplot(2,2,1);imshow(LL);title('LL');
subplot(2,2,2);imshow(LH);title('LH');
subplot(2,2,3);imshow(HL);title('HL');
subplot(2,2,4);imshow(HH);title('HH');



%% Synthesis Part

%% Synthesis Low filter #1 (Low-branch)

uuLL = upsample(dLL,2); % upsample by 2 befor low filtering
ssLL = filter(f0,1,uuLL); % synthesis low filtering #1

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
   for j=1:z
      uLL(j,i) = uuLL(m);
      m=m+1;
   end
end

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
   for j=1:z
      sLL(j,i) = ssLL(m);
      m=m+1;
   end
end

figure(18);
subplot(1,2,1);imshow(LL);title('Analysis LL');
subplot(1,2,2);imshow(uLL);title('Upsample LL');
figure(19);
subplot(1,2,1);imshow(uLL);title('Upsample LL');
```

```
subplot(1,2,2);imshow(sLL);title('Synthesis LL');

%% Synthesis High filter #1 (Low-branch)

uuLH = upsample(dLH,2); % upsample by 2 befor high filtering
ssLH = filter(f1,1,uuLH); % synthesis high filtering #1

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
    for j=1:z
       uLH(j,i) = uuLH(m);
       m=m+1;
    end
end

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
    for j=1:z
       sLH(j,i) = ssLH(m);
       m=m+1;
    end
end

figure(20);
subplot(1,2,1);imshow(LH);title('Analysis LH');
subplot(1,2,2);imshow(uLH);title('Upsample LH');
figure(21);
subplot(1,2,1);imshow(uLH);title('Upsample LH');
subplot(1,2,2);imshow(sLH);title('Synthesis LH');

%% Synthesis Low filter #1 (High-branch)

uuHL = upsample(dHL,2); % upsample by 2 befor low filtering
ssHL = filter(f0,1,uuHL); % synthesis low filtering #1

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
    for j=1:z
       uHL(j,i) = uuHL(m);
       m=m+1;
    end
end

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
    for j=1:z
       sHL(j,i) = ssHL(m);
       m=m+1;
    end
end

figure(22);
subplot(1,2,1);imshow(HL);title('Analysis HL');
subplot(1,2,2);imshow(uHL);title('Upsample HL');
figure(23);
subplot(1,2,1);imshow(uHL);title('Upsample HL');
subplot(1,2,2);imshow(sHL);title('Synthesis HL');

%% Synthesis High filter #1 (High-branch)
uuHH = upsample(dHH,2); % upsample by 2 befor high filtering
ssHH = filter(f1,1,uuHH); % synthesis high filtering #1
```

```
%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
   for j=1:z
      uHH(j,i) = uuHH(m);
      m=m+1;
   end
end

%Reshape data from 1D to 2D
m =1;
for i=1:(z/2)
   for j=1:z
      sHH(j,i) = ssHH(m);
      m=m+1;
   end
end

figure(24);
subplot(1,2,1);imshow(HH);title('Analysis HH');
subplot(1,2,2);imshow(uHH);title('Upsample HH');
figure(25);
subplot(1,2,1);imshow(uHH);title('Upsample HH');
subplot(1,2,2);imshow(sHH);title('Synthesis HH');

%% Sum matrix in Low-branch

sLLLH = sLL + sLH;

%% Sum matrix in High-branch

sHLHH = sHL + sLH;


%% Upsample in Low-branch

%sLLLHt = sLLLH'; %Transpose in Low-branch to row operation

rLLLH = reshape(sLLLH',1,(z/2)*z); % convert frome 2D to 1D

uLLLH = upsample(rLLLH,2); % Upsample in Low-branch

%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:z
      uLP(i,j) = uLLLH(m);
      m=m+1;
   end
end

%% Upsample in High-branch

%sHLHHt = sHLHH'; %Transpose in High-branch to row operation

rHLHH = reshape(sHLHH',1,(z/2)*z); % convert frome 2D to 1D

uHLHH = upsample(rHLHH,2); % Upsample in Low-branch

%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:z
      uHP(i,j) = uHLHH(m);
      m=m+1;
   end
```

```
end

%% Synthesis Low-filtering#2 (Low-branch)

yyLP = filter(f0,1,uLLLH);

%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:z
      yLP(i,j) = yyLP(m);
      m=m+1;
   end
end

%% Synthesis High-filtering#2 (High-branch)

yyHP = filter(f1,1,uHLHH);

%Reshape data from 1D to 2D
m =1;
for i=1:z
   for j=1:z
      yHP(i,j) = yyHP(m);
      m=m+1;
   end
end

%% Sum matrix to provide the output

Y = yLP + yHP;

%% Show image
figure(26);
subplot(1,2,1);imshow(sLLLH);title('Sum matrix between LL LH synthesis');
subplot(1,2,2);imshow(uLP);title('Upsampled in Low branch');
figure(27);
subplot(1,2,1);imshow(sHLHH);title('Sum matrix between HL HH synthesis');
subplot(1,2,2);imshow(uHP);title('Upsampled in High branch');
figure(28);
subplot(1,2,1);imshow(uLP);title('Upsampled in Low branch');
subplot(1,2,2);imshow(yLP);title('Synthesis Low Filtering output image');
figure(29);
subplot(1,2,1);imshow(uHP);title('Upsampled in High branch');
subplot(1,2,2);imshow(yHP);title('Synthesis High Filtering output image');
figure(30);
subplot(1,2,1);imshow(X);title('Input Image');
subplot(1,2,2);imshow(Y);title('Reconstruction image');
```

โปรแกรมฟังก์ชั่นที่ใช้ในจำกัดบิตการควอนไตซ์สำหรับการจำลองการทำงานของฮาร์ดแวร์

```
function coarse = ftoc( x )

x8=ceil(x*6);
xdoub=x8/6;

coarse=xdoub;

end
```

# ภาคผนวก ข

## โปรแกรมภาษา VHDL

## อุปกรณ์เลื่อนบิตแบบเข้าขนานออกอนุกรม (PISO Shift Register)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PISO_H0 is
     port (xn       : in std_logic_vector(7 downto 0);
           clk,lr   : in std_logic;
           x1       : out std_logic);
end PISO_H0;

architecture Behavioral of PISO_H0 is
begin
     process(clk,xn,lr)
     variable sbus : std_logic_vector(7 downto 0);
     begin
       IF (lr = '0') THEN
             sbus := (not xn(7)) & xn(6 downto 0);
       ElSIF (clk'event and clk='1') THEN
             x1              <= 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 Behavioral;
```

## อุปกรณ์เลื่อนบิตแบบเข้าอนุกรมออกอนุกรม (SISO Shift Register)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SISO1_H0 is
     port (x1,clk     : in std_logic;
           x2 : out std_logic);
end SISO1_H0;

architecture Behavioral of SISO1_H0 is
     signal sbus : std_logic_vector(7 downto 1);
begin
     process(clk)
     begin
       if (clk'event and clk='1') then
             x2             <= 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) <= x1;
       end if;
```

```
    end process;
end Behavioral;
```

## อุปกรณ์มัลติเพล็กซ์เซอร์ (Multiplexer)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX_H0 is
    port (x1,x2,x3,x4,x5,x6,x7,x8,x9 : in std_logic;
          y1,y2,y3,y4,y5,y6,y7,y8,y9 : out std_logic_vector(7 downto 0));
end MUX_H0;

architecture Behavioral of MUX_H0 is
begin

    y1 <= "00000001" when x1 = '1' else "00000000";
    y2 <= "00000000" when x2 = '1' else "00000000";
    y3 <= "11111110" when x3 = '1' else "00000000";
    y4 <= "00000111" when x4 = '1' else "00000000";
    y5 <= "00001111" when x5 = '1' else "00000000";
    y6 <= "00000111" when x6 = '1' else "00000000";
    y7 <= "11111110" when x7 = '1' else "00000000";
    y8 <= "00000000" when x8 = '1' else "00000000";
    y9 <= "00000001" when x9 = '1' else "00000000";

end Behavioral;
```

## อุปกรณ์บวก (Adder)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity Adder_H0 is
    port (y1,y2,y3,y4,y5,y6,y7,y8,y9 :in std_logic_vector(7 downto 0);
          data : out std_logic_vector(7 downto 0));
end Adder_H0;

architecture Behavioral of Adder_H0 is

begin


    data <= y1+y2+y3+y4+y5+y6+y7+y8+y9;

end Behavioral;
```

## อุปกรณ์เลื่อนและบวกสะสม (Scaling Accumulator)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Acc_H0 is
      port (add_out    : in    std_logic_vector(7 downto 0);
            clacc,lacc      : in    std_logic;
            acc_out         : out std_logic_vector(7 downto 0));
end Acc_H0;

architecture Behavioral of Acc_H0 is
      signal acc_signal : std_logic_vector(7 downto 0);
begin
      process (lacc,clacc)
      begin
        if      clacc='0' then
                acc_signal<=(others=>'0');
        elsif lacc'event and lacc='1' then
                acc_signal<=add_out;
        end if;
      end process;
        acc_out<=acc_signal;
end Behavioral;
```

## อุปกรณ์บวกและลบภายในวงจรเลื่อนและบวกสะสม (Add/Sub)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Add_H0 is
port( data                    : in  std_logic_vector(7 downto 0);
       acc_out                : in  std_logic_vector(7 downto 0);
      sa              : in  std_logic;
      add_out                 : out std_logic_vector(7 downto 0));
end Add_H0;

architecture Behavioral of Add_H0 is
begin
    process  (sa,data,acc_out)
        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 sa = '0' then
            s := ("0"&data)+acc_out;
            sum(7) :=s(8) xor data(7) xor acc_out(7) ;
            sum(6 downto 0):=s(7 downto 1);
            else
            s_sub:=(data xor "11111111");
            s_sub:=s_sub+1;
            sum(7 downto 0) :=acc_out(7 downto 0)+s_sub(7 downto 0);
        end if;
        add_out<=sum;
     end process ;
end Behavioral;
```

## อุปกรณ์บัฟเฟอร์ (Buffer)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Buff_H0 is
     port(     lr,RESET       : in std_logic;
               acc_out : in std_logic_vector(7 downto 0);
               xn_1           : BUFFER std_logic_vector(7 downto 0));
end Buff_H0;

architecture Behavioral of Buff_H0 is
begin
process(lr,acc_out,RESET)
     begin
     IF lr = '0' THEN
       IF RESET = '0' THEN
               xn_1 <= (others=>'0');
       ELSE
               xn_1 <= acc_out;
       END IF;
     END IF;
end process;
end Behavioral;
```

## อุปกรณ์ลดค่าความถี่สุ่มตัวอย่าง

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_unit_2 is
    port( syn_clk_in : in std_logic;
              RESET          : in std_logic;
          lr2                : out std_logic);
end control_unit_2;

architecture Behavioral of control_unit_2 is
begin
     process(RESET,syn_clk_in)
     variable s : integer range 0 to 63 :=0;
     begin
       IF (syn_clk_in'event AND syn_clk_in='1') THEN
              IF RESET = '0' THEN
                     s := 0;
                     lr2            <= '0';
              ELSE
                     IF s < 63 THEN
                            s := s + 1;
                     else
                            s := 0;
                     end if;

                     if
(s=1)OR(s=5)OR(s=9)OR(s=13)OR(s=17)OR(s=21)OR(s=25)OR(s=29)OR(s=33)OR(s=37)OR(s=41)
OR(s=45)OR(s=49)OR(s=53)OR(s=57) THEN
                                    lr2<='1';
```

```
                        elsif
(s=2)OR(s=6)OR(s=10)OR(s=14)OR(s=18)OR(s=22)OR(s=26)OR(s=30)OR(s=34)OR(s=38)OR(s=42
)OR(s=46)OR(s=50)OR(s=54)OR(s=58) THEN
                                lr2<='1';
                        elsif
(s=3)OR(s=4)OR(s=7)OR(s=8)OR(s=15)OR(s=16)OR(s=19)OR(s=20)OR(s=23)OR(s=24)OR(s=27)O
R(s=28)OR(s=31)OR(s=32)OR(s=35)OR(s=36)OR(s=39)OR(s=40)OR(s=43)OR(s=44)OR(s=47)OR(s
=48)OR(s=51)OR(s=52)OR(s=55)OR(s=56) THEN
                                lr2<='1';
                        elsif (s=11)OR(s=12) then
                                lr2<='1';
                        elsif (s=59)OR(s=60) then
                                lr2<='1';
                        elsif (s=61) then
                                lr2<='1';
                        elsif (s=62) then
                                lr2<='1';
                        elsif (s=63) then
                                lr2<='0';
                        else
                                lr2<='1';
                        end if;
        end if;
        end if;
      end process;
end Behavioral;
```

## อุปกรณ์เพิ่มค่าความถี่สุ่มตัวอย่าง

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Up2_F00 is
     port (RESET    : in std_logic;
           yn_1     : in std_logic_vector(7 downto 0);
           lr,lr2   : in std_logic;
           un_1     : out std_logic_vector(7 downto 0));
end Up2_F00;

architecture Behavioral of Up2_F00 is
begin
     process(RESET,yn_1,lr,lr2)
       VARIABLE sbus       : std_logic_vector(7 downto 0);
       VARIABLE sbus1      : std_logic_vector(7 downto 0);
       VARIABLE counter    : integer range 0 to 1 := 0 ;
     begin
       -- Load data
       IF (lr2'event AND lr2='0') THEN
             IF RESET = '0' THEN
                   sbus := (OTHERS=>'0');
                   sbus1:= (OTHERS=>'0');
                   un_1 <= (OTHERS=>'0');
             ELSE
                   sbus := yn_1;
             END IF;
       END IF;
       -- UP SAMPLING BY 2
```

```
            IF (lr'event AND lr='0') THEN
                  IF counter < 1 THEN
                        counter := counter+1;
                        sbus1 := (OTHERS=>'0');
                  ELSIF counter=1 THEN
                        counter := 0;
                        sbus1 := sbus;
                  END IF;
            END IF;
            un_1 <= sbus1;
      end process;
end Behavioral;
```

## อุปกรณ์ควบคุม (Control Unit)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_unit is
    port( syn_clk_in : in std_logic;
          RESET      : in std_logic;
       clk,sa,lacc,lr,clacc,sc : out std_logic);
end control_unit;

architecture Behavioral of control_unit is
begin
      process(RESET,syn_clk_in)
      variable state : integer range 0 to 31 :=0;
      begin
      if syn_clk_in'event and syn_clk_in='1' then
       if RESET = '0' then
            state := 0;
            clk   <= '0';
            sa    <= '0';
            lacc  <= '0';
            lr    <= '0';
            clacc <= '0';
            sc    <= '0';
            else
                  if state < 31 then
                        state := state + 1;
                  else
                        state := 0;
                  end if;

                  if
(state=1)OR(state=5)OR(state=9)OR(state=13)OR(state=17)OR(state=21)OR(state=25)
then
                        clk<='1';
                        sa<='0';
                        lacc<='0';
                        lr<='1';
                        clacc<='1';
                        sc<='1';
                  elsif
(state=2)OR(state=6)OR(state=10)OR(state=14)OR(state=18)OR(state=22)OR(state=26)
then
                        clk<='1';
                        sa<='0';
```

```
                                lacc<='1';
                                lr<='1';
                                clacc<='1';
                                sc<='1';
                            elsif
(state=3)OR(state=4)OR(state=7)OR(state=8)OR(state=15)OR(state=16)OR(state=19)OR(st
ate=20)OR(state=23)OR(state=24) then
                                clk<='0';
                                sa<='0';
                                lacc<='0';
                                lr<='1';
                                clacc<='1';
                                sc<='1';
                            elsif (state=11)OR(state=12) then
                                clk<='0';
                                sa<='0';
                                lacc<='0';
                                lr<='1';
                                clacc<='1';
                                sc<='0';
                            elsif (state=27)OR(state=28) then
                                clk<='0';
                                sa<='1';
                                lacc<='0';
                                lr<='1';
                                clacc<='1';
                                sc<='1';
                            elsif (state=29) then
                                clk<='1';
                                sa<='1';
                                lacc<='0';
                                lr<='1';
                                clacc<='1';
                                sc<='1';
                            elsif (state=30) then
                                clk<='1';
                                sa<='1';
                                lacc<='1';
                                lr<='1';
                                clacc<='1';
                                sc<='1';
                            elsif (state=31) then
                                clk<='0';
                                sa<='0';
                                lacc<='0';
                                lr<='0';
                                clacc<='1';
                                sc<='1';
                            else
                                clk<='0';
                                sa<='0';
                                lacc<='0';
                                lr<='1';
                                clacc<='0';
                                sc<='1';
                            end if;
        end if;
        end if;
      end process;
end Behavioral;
```

## อุปกรณ์หน่วยความจำชั่วคราว (RAM)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Data_Buffer_RAM is
      PORT (WEA        :       IN      STD_LOGIC_VECTOR(0 DOWNTO 0);
                lr,lr2  :       IN      STD_LOGIC;
                writeram        :       IN STD_LOGIC_VECTOR(14 DOWNTO 0);
                readram :       IN      STD_LOGIC_VECTOR(14 DOWNTO 0);
                yn_1            :       IN      STD_LOGIC_VECTOR(7 DOWNTO 0);
                yn_2            :       IN      STD_LOGIC_VECTOR(7 DOWNTO 0);
                rn_1            :       OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                rn_2            :       OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
end Data_Buffer_RAM;

architecture Behavioral of Data_Buffer_RAM is

COMPONENT RAM1
      PORT (clka        : IN STD_LOGIC;
                wea     : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
                addra : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
                dina    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                clkb    : IN STD_LOGIC;
                addrb   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
                doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

COMPONENT RAM2
      PORT (clka        : IN STD_LOGIC;
                wea     : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
                addra : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
                dina    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                clkb    : IN STD_LOGIC;
                addrb : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
                doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

begin
      U1        :       RAM1    PORT MAP
      (lr2,WEA,writeram,yn_1,lr,readram,rn_1);
      U2        :       RAM2    PORT MAP
      (lr2,WEA,writeram,yn_2,lr,readram,rn_2);
end Behavioral;
```

## อุปกรณ์ควบคุมการอ่านหน่วยความจำชั่วคราว (Read RAM Controller)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity read_ram is
GENERIC(
      addr_l    :       INTEGER   := 15; -- Length of Address (MxN)
    rows        :   INTEGER   := 256; -- Number of row
    cols        :   INTEGER   := 128); -- Number of column
PORT ( reset            : in std_logic;
```

```
        clk                    : in  STD_LOGIC;
        readram       : out std_logic_vector((addr_1 - 1) downto 0)
       );
end read_ram;

architecture Behavioral of read_ram is
     CONSTANT  row_point  :  INTEGER := rows - 1;
     CONSTANT  col_point  :  INTEGER := cqls - 1;
     CONSTANT  matx_point :  INTEGER := (rows * cols)-1;
begin
     process(clk,reset)
     variable addr: integer range 0 to matx_point := 0;
     variable row : integer range 0 to row_point  := 0;
     variable col : integer range 0 to col_point  := 0;
     begin
     IF clk'event and clk='1' THEN
       IF (reset = '0') THEN
              addr := 0;
       ELSE
              IF(row < row_point) THEN
                     row  := row + 1;
                     addr := addr + cols;
                     ELSE
                           row  := 0;
                           addr := 0;
                           IF(col < col_point) THEN
                                 col  := col + 1;
                                 addr := addr + col;
                                 ELSE
                                       col  := 0;
                                       addr := 0;
                           END IF;
                     END IF;
       END IF;
     END IF;
       readram <= conv_std_logic_vector(addr,addr_1);
     END PROCESS;
end Behavioral;
```

## อุปกรณ์ควบคุมการเขียนหน่วยความจำชั่วคราว (Write RAM Controller)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity write_ram is
    Port ( reset : in std_logic;
                     clk : in  STD_LOGIC;
           writeram : out  STD_LOGIC_VECTOR(14 downto 0));
end write_ram;

architecture Behavioral of write_ram is

begin
cnt_process: process(clk,reset)
variable cnt:integer range 0 to 32767 := 0;
     begin
     IF clk'event and clk = '1' THEN
       IF (reset = '0') THEN
             cnt := 0;
             ELSE
                    cnt := cnt + 1;
```

```
        END IF;
      END IF;
      writeram <= conv_std_logic_vector(cnt,15);
END PROCESS;

end Behavioral;
```

## อุปกรณ์ควบคุมหน่วยความจำบนบอร์ด (ROM Controller)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity read_rom256 is

    Port ( reset : in std_logic;
                clk : in  STD_LOGIC;
            readrom : out std_logic_vector(15 downto 0)
                );
end read_rom256;

architecture Behavioral of read_rom256 is
begin
cnt_process: process(clk,reset)
variable cnt:integer range 0 to 65535 := 0;
      begin
      IF clk'event and clk = '1' THEN
        IF (reset = '0') THEN
              cnt := 0;
              ELSE
                    cnt := cnt + 1;
        END IF;
      END IF;
      readrom <= conv_std_logic_vector(cnt,16);
END PROCESS;

end Behavioral;
```

## อุปกรณ์ควบคุมการรีเซ็ต (Reset Controller)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RESET_controller is
     PORT (clk :       IN      STD_LOGIC;
               RESET   :       IN      STD_LOGIC;
               RESET1: OUT STD_LOGIC;
               RESET2: OUT STD_LOGIC;
               RESET3: OUT STD_LOGIC;
               WEA     :       OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
               WEAs    :       OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
               WEA3    :       OUT STD_LOGIC_VECTOR(0 DOWNTO 0));
end RESET_controller;

architecture Behavioral of RESET_controller is
```

```
begin
      PROCESS (clk,RESET)
        VARIABLE cnt : INTEGER RANGE 0 TO 163840 := 0;
      BEGIN
        IF (clk'EVENT AND clk = '1') THEN
                IF (RESET = '0') THEN
                        RESET1 <= '0';
                        RESET2 <= '0';
                        RESET3 <= '0';
                        WEA      <= "0";
                        WEAs     <= "0";
                        WEA3     <= "0";
                        cnt := 0;
                ELSE
                        IF (cnt < 163840) THEN
                                cnt := cnt + 1;
                        ELSE
                                cnt := 1;
                        END IF;

                        IF (cnt < 65535) THEN
                                RESET1 <= '1';
                                RESET2 <= '0';
                                RESET3 <= '0';
                                WEA      <= "1";
                                WEAs     <= "0";
                                WEA3     <= "0";

                        ELSIF (cnt > 65536 AND cnt < 98303) THEN
                                RESET1 <= '0';
                                RESET2 <= '1';
                                RESET3 <= '0';
                                WEA      <= "0";
                                WEAs     <= "1";
                                WEA3     <= "0";

                        ELSIF (cnt > 98304 AND cnt < 163839) THEN
                                RESET1 <= '0';
                                RESET2 <= '0';
                                RESET3 <= '1';
                                WEA      <= "0";
                                WEAs     <= "0";
                                WEA3     <= "1";

                        END IF;
                END IF;
        END IF;
      END PROCESS;
end Behavioral;
```

## อุปกรณ์ควบคุมการแสดงผลบนจอแสดงผลวีจีเอ (VGA Controller)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VGA_controller IS
      GENERIC (h_pulse   :  INTEGER    := 128;    --horiztonal sync pulse width in
pixels
                        h_bp      :  INTEGER    := 88;    --horiztonal back porch width
```

```
in pixels
                        h_pixels :  INTEGER   := 800;  --horiztonal display width in
pixels
                        h_fp     :  INTEGER   := 40;   --horiztonal front porch
width in pixels
                        h_pol    :  STD_LOGIC := '1';   --horizontal sync pulse
polarity (1 = positive, 0 = negative)
                        v_pulse  :  INTEGER   := 4;    --vertical sync pulse width
in rows
                        v_bp     :  INTEGER   := 23;    --vertical back porch width
in rows
                        v_pixels :  INTEGER   := 600;   --vertical display width in
rows
                        v_fp     :  INTEGER   := 1;     --vertical front porch width
in rows
                        v_pol    :  STD_LOGIC := '1');  --vertical sync pulse
polarity (1 = positive, 0 = negative)

     PORT (pixel_clk  :  IN  STD_LOGIC;  --pixel clock at frequency of VGA mode
being used
               RESET                  :  IN  STD_LOGIC;
               --SW00                :  IN  STD_LOGIC;
               i_input                :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_LA                   :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_HA                   :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_LLA                  :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_LHA                  :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_HLA                  :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_HHA                  :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_LS                   :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_HS                   :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               i_RE                   :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
               h_sync                 :  OUT STD_LOGIC;
               v_sync                 :  OUT STD_LOGIC;
               VGA                    :  OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END VGA_controller;

ARCHITECTURE behavior OF VGA_controller IS
  CONSTANT  h_period  :  INTEGER := h_pulse + h_bp + h_pixels + h_fp;  --total
number of pixel clocks in a row
  CONSTANT  v_period  :  INTEGER := v_pulse + v_bp + v_pixels + v_fp;  --total
number of rows in column
BEGIN

  PROCESS(pixel_clk,RESET)
    VARIABLE h_count  :  INTEGER RANGE 0 TO h_period - 1 := 0;  --horizontal
counter (counts the columns)
    VARIABLE v_count  :  INTEGER RANGE 0 TO v_period - 1 := 0;  --vertical counter
(counts the rows)
  BEGIN

    IF(RESET = '0') THEN
      h_count := 0;
      v_count := 0;
       VGA <= (OTHERS=>'0');
      h_sync <= NOT h_pol;
      v_sync <= NOT v_pol;
    ELSIF(pixel_clk'EVENT AND pixel_clk = '1') THEN
      --counters
      IF(h_count < h_period - 1) THEN    --horizontal counter (pixels)
            h_count := h_count + 1;
            ELSE
                h_count := 0;
      IF(v_count < v_period - 1) THEN  --veritcal counter (rows)
          v_count := v_count + 1;
            ELSE
                v_count := 0;
```

```
        END IF;
      END IF;
      --horizontal sync signal
      IF(h_count < h_pixels + h_fp OR h_count > h_pixels + h_fp + h_pulse) THEN
        h_sync <= NOT h_pol;
      ELSE
        h_sync <= h_pol;
      END IF;
      --vertical sync signal
      IF(v_count < v_pixels + v_fp OR v_count > v_pixels + v_fp + v_pulse) THEN
        v_sync <= NOT v_pol;
      ELSE
        v_sync <= v_pol;
      END IF;

                -- Show input image
                IF((h_count>16) AND (h_count<272) AND (v_count>43) AND
(v_count<299))THEN
                        VGA <= i_input(7 DOWNTO 0);
                --show output of LOW filtering and down-sample
                ELSIF((h_count>272) AND (h_count<400) AND (v_count>43) AND
(v_count<299))THEN
                        VGA <= not i_LA(7) & i_LA(6 DOWNTO 0);
                --show output of HIGH filtering and down-sample
                ELSIF((h_count>400) AND (h_count<528) AND (v_count>43) AND
(v_count<299))THEN
                        VGA <= not i_HA(7) & i_HA(6 DOWNTO 0);
                --show output of LOW-LOW filtering and down-sample
                ELSIF((h_count>528) AND (h_count<656) AND (v_count>43) AND
(v_count<171))THEN
                        VGA <= not i_LLA(7) & i_LLA(6 DOWNTO 0);
                --show output of LOW-HIGH filtering and down-sample
                ELSIF((h_count>528) AND (h_count<656) AND (v_count>171) AND
(v_count<299))THEN
                        VGA <= not i_LHA(7) & i_LHA(6 DOWNTO 0);
                --show output of HIGH-LOW filtering and down-sample
                ELSIF((h_count>656) AND (h_count<784) AND (v_count>43) AND
(v_count<171))THEN
                        VGA <= not i_HLA(7) & i_HLA(6 DOWNTO 0);
                --show output of HIGH-HIGH filtering and down-sample
                ELSIF((h_count>656) AND (h_count<784) AND (v_count>171) AND
(v_count<299))THEN
                        VGA <= not i_HHA(7) & i_HHA(6 DOWNTO 0);
                --show output of LOW filtering and up-sample
                ELSIF((h_count>400) AND (h_count<528) AND (v_count>300) AND
(v_count<556))THEN
                        VGA <= not i_LS(7) & i_LS(6 DOWNTO 0);
                --show output of HIGH filtering and up-sample
                ELSIF((h_count>528) AND (h_count<656) AND (v_count>300) AND
(v_count<556))THEN
                        VGA <= not i_HS(7) & i_HS(6 DOWNTO 0);
              --show RECONSTRUC image
              ELSIF((h_count>144) AND (h_count<400) AND (v_count>300) AND
(v_count<556))THEN
                        VGA <= i_RE;
          ELSE
                        VGA <= (OTHERS=>'0');
              END IF;
    END IF;
END PROCESS;

END behavior;
```

# ภาคผนวก ค

## คู่มือการใช้งาน FPGA Xilinx Zynq Zedboard XC7Z020-1CLG484

# ZedBoard

## (Zynq™ Evaluation and Development) Hardware User's Guide


**Version 2.2**
**27 January 2014**

# 1 Introduction

The ZedBoard is an evaluation and development board based on the Xilinx Zynq™-7000 All Programmable SoC (AP SoC). Combining a dual Corex-A9 Processing System (PS) with 85,000 Series-7 Programmable Logic (PL) cells, the Zynq-7000 AP SoC can be targeted for broad use in many applications. The ZedBoard's robust mix of on-board peripherals and expansion capabilities make it an ideal platform for both novice and experienced designers. The features provided by the ZedBoard consist of:

- Xilinx® XC7Z020-1CLG484C Zynq-7000 AP SoC
  - Primary configuration = QSPI Flash
  - Auxiliary configuration options
    - Cascaded JTAG
    - SD Card
- Memory
  - 512 MB DDR3 (128M x 32)
  - 256 Mb QSPI Flash
- Interfaces
  - USB-JTAG Programming using Digilent SMT1-equivalent circuit
    - Accesses PL JTAG
    - PS JTAG pins connected through PS Pmod
  - 10/100/1G Ethernet
  - USB OTG 2.0
  - SD Card
  - USB 2.0 FS USB-UART bridge
  - Five Digilent Pmod™ compatible headers (2x6) (1 PS, 4 PL)
  - One LPC FMC
  - One AMS Header
  - Two Reset Buttons (1 PS, 1 PL)
  - Seven Push Buttons (2 PS, 5 PL)
  - Eight dip/slide switches (PL)
  - Nine User LEDs (1 PS, 8 PL)
  - DONE LED (PL)
- On-board Oscillators
  - 33.333 MHz (PS)
  - 100 MHz (PL)
- Display/Audio
  - HDMI Output
  - VGA (12-bit Color)
  - 128x32 OLED Display
  - Audio Line-in, Line-out, headphone, microphone
- Power
  - On/Off Switch
  - 12V @ 5A AC/DC regulator
- Software
  - ISE® WebPACK Design Software
  - License voucher for ChipScope™ Pro locked to XC7Z020

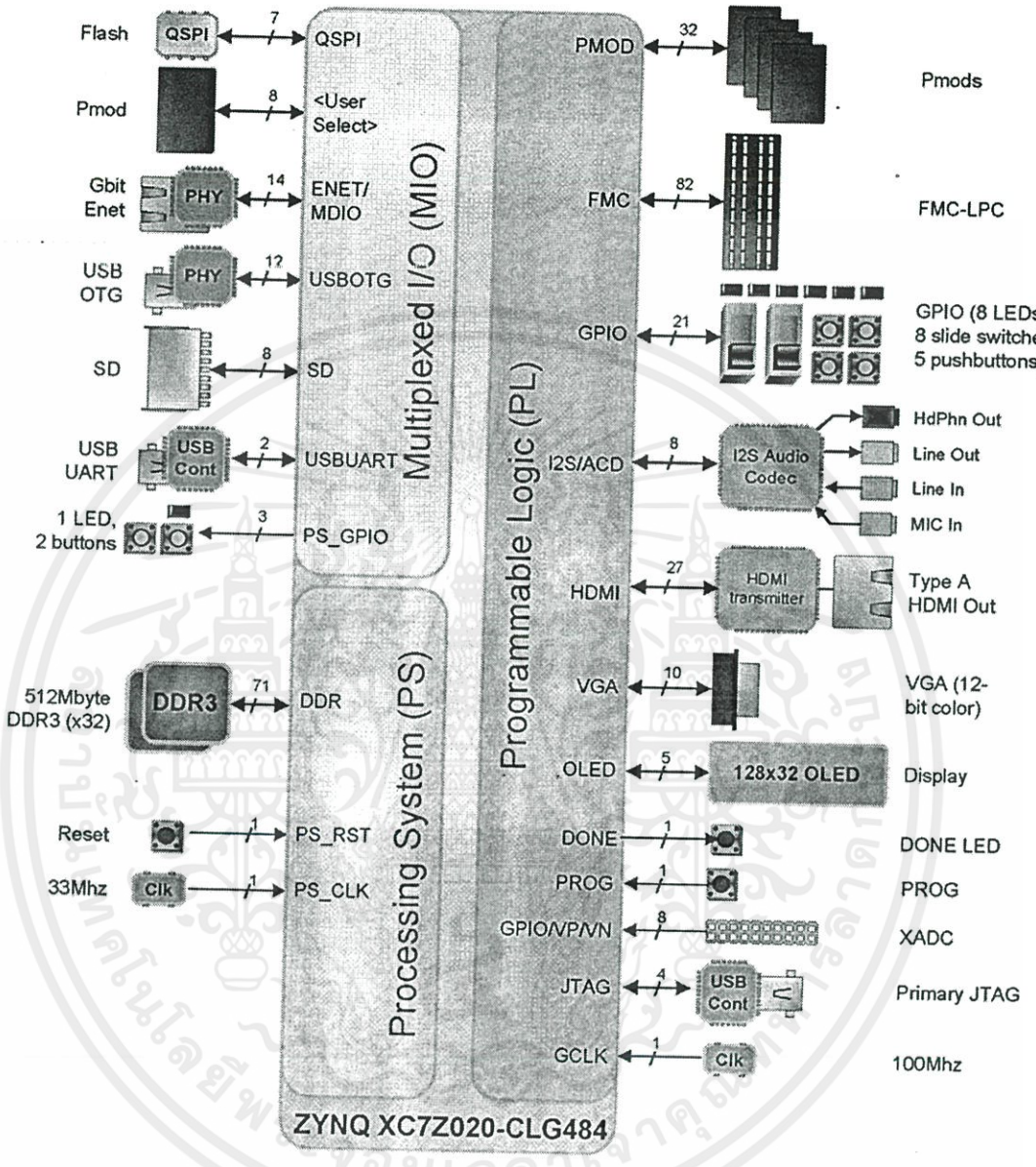



**Figure 1 – ZedBoard Block Diagram**

The UART 1 Zynq PS peripheral is accessed through MIO[48:49] in MIO Bank 1/501 (1.8V). Since the CY7C64225 device requires either 3.3V or 5V signaling, a TI TXS0102 level shifter is used to level shift between 3.3V and 1.8V.

This USB port will not power the board. Therefore, Vbus needs to be connected to 3.3V though a 1KΩ series resistor. The Wake pin, pin 22, connects to GND. A 24Ω series resistor was placed on each of the data lines, D+ and D-.

**Table 6 – CY7C6 Connections**

| UART Function in Zynq | Zynq Pin | MIO | Schematic Net Name | CY7C6 Pin | UART Function in CY7C64225 |
|---|---|---|---|---|---|
| TX, data out | D11 (MIO Bank 1/501) | 48:49 | USB_1_RXD | 23 | RXD, data in |
| RX, data in | C14 (MIO Bank 1/501) | | USB_1_TXD | 4 | TXD, data out |

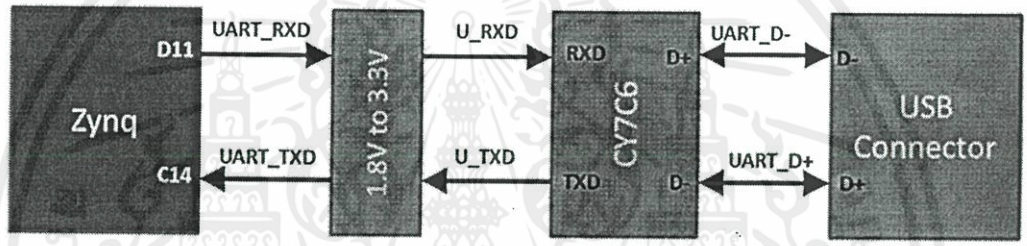



**Figure 7 – USB-UART Bridge Interface**

### 2.3.3 USB-JTAG

The ZedBoard provides JTAG functionality based on the Digilent USB High Speed JTAG Module, SMT1 device. This USB-JTAG circuitry is fully supported and integrated into Xilinx ISE tools, including iMPACT, ChipScope, and SDK Debugger. Designers who want to re-use this circuit on their board can do so by acquiring these modules from Avnet.

http://www.em.avnet.com/en-us/design/drc/Pages/Digilent-JTAG-SMT1-Surface-Mount-Programming-Module.aspx

The JTAG is available through a Micro B USB connector, J17, TE 1981568-1. TCK has a series termination resistor, 20-30Ω, to prevent signal integrity issues.

For the JTAG Chain setup, please refer to the Configuration section.

### 2.4.2 VGA Connector

The ZedBoard also allows 12-bit color video output through a through-hole VGA connector, TE 4-1734682-2. Each color is created from resistor-ladder from four PL pins.

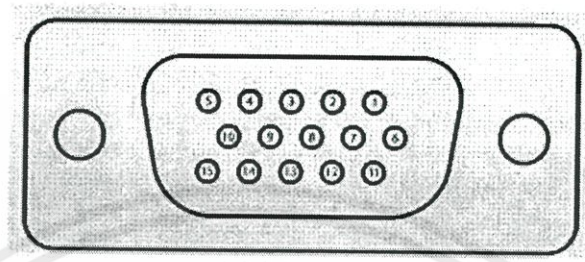


Figure 10 - DB15

Table 8 - VGA Connections

| VGA Pin | Signal | Description | Zynq Pin |
|---|---|---|---|
| 1 | RED | Red video | V20, U20, V19, V18 |
| 2 | GREEN | Green video | AB22, AA22, AB21, AA21 |
| 3 | BLUE | Blue video | Y21, Y20, AB20, AB19 |
| 4 | ID2/RES | formerly Monitor ID bit 2 | NC |
| 5 | GND | Ground (HSync) | NC |
| 6 | RED_RTN | Red return | NC |
| 7 | GREEN_RTN | Green return | NC |
| 8 | BLUE_RTN | Blue return | NC |
| 9 | KEY/PWR | formerly key | NC |
| 10 | GND | Ground (VSync) | NC |
| 11 | ID0/RES | formerly Monitor ID bit 0 | NC |
| 12 | ID1/SDA | formerly Monitor ID bit 1 | NC |
| 13 | HSync | Horizontal sync | AA19 |
| 14 | VSync | Vertical sync | Y19 |
| 15 | ID3/SCL | formerly Monitor ID bit 3 | NC |
| | | | |

### 2.6.2 Program Push Button Switch

A PROG push switch, BTN6, toggles Zynq PROG_B. This initiates reconfiguring the PL-subsection by the processor.

### 2.6.3 Processor Subsystem Reset

Power-on reset, labeled PS_RST/BTN7, erases all debug configurations. The external system reset allows the user to reset all of the functional logic within the device without disturbing the debug environment. For example, the previous break points set by the user remain valid after system reset. Due to security concerns, system reset erases all memory content within the PS, including the OCM. The PL is also reset in system reset. System reset does not re-sample the boot mode strapping pins.

## 2.7 User I/O

### 2.7.1 User Push Buttons

The ZedBoard provides 7 user GPIO push buttons to the Zynq-7000 AP SoC; five on the PL-side and two on the PS-side.

Pull-downs provide a known default state, pushing each button connects to Vcco.

**Table 12 - Push Button Connections**

| Signal Name | Subsection | Zynq pin |
|---|---|---|
| BTNU | PL | T18 |
| BTNR | PL | R18 |
| BTND | PL | R16 |
| BTNC | PL | P16 |
| BTNL | PL | N15 |
| PB1 | PS | D13 (MIO 50) |
| PB2 | PS | C10 (MIO 51) |

### 2.7.2 User DIP Switches

The ZedBoard has eight user dip switches, SW0-SW7, providing user input. SPDT switches connect the I/O through a 10kΩ resistor to the VADJ voltage supply or GND.

**Table 13 - DIP Switch Connections**

| Signal Name | Zynq pin |
|---|---|
| SW0 | F22 |
| SW1 | G22 |
| SW2 | H22 |
| SW3 | F21 |
| SW4 | H19 |
| SW5 | H18 |
| SW6 | H17 |
| SW7 | M15 |

### 2.9.2 Digilent Pmod™ Compatible Headers (2x6)

The ZedBoard has five Digilent Pmod™ compatible headers (2x6). These are right-angle, 0.1" female headers that include eight user I/O plus 3.3V and ground signals as show in the figure below.

Four Pmod connectors interface to the PL-side of the Zynq-7000 AP SoC. These will connect to Bank 13 (3.3V). One Pmod, JE1, connects to the PS-side on MIO pins [0,9-15] in MIO Bank 0/500 (3.3V). Uses for this Pmod include PJTAG access (MIO[10-13]) as well as nine other hardened MIO peripherals (SPI, GPIO, CAN, I2C, UART, SD, QSPI, Trace, Watchdog).

The four PL Pmod connectors are placed in adjacent pairs on the board edge such that the clearance between Pin 6 of header #1 and Pin 1 of header #2 is 10mm.

Two of the Pmods, JC1 and JD1, are aligned in a dual configuration and have their I/O routed differentially to support LVDS running at 525Mbs.

All Pmod data lines, 8 per connector, are protected with two 4-channel TE SESD1004Q4UG-0020-090.

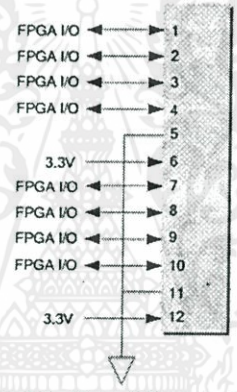



**Figure 12 - Pmod Connections**

A couple links to Pmod examples are provided:

- http://www.em.avnet.com/en-us/design/drc/Pages/Digilent-PmodWiFi-802-11bgn-WiFi-Interface.aspx
- http://www.em.avnet.com/en-us/design/drc/Pages/Digilent-Pmod-RS232-Serial-Converter-and-Interface.aspx

**Table 16 - Pmod Connections**

| Pmod | Signal Name | Zynq pin | Pmod | Signal Name | Zynq pin |
|---|---|---|---|---|---|
| JA1 | JA1 | Y11 | JB1 | JB1 | W12 |
| | JA2 | AA11 | | JB2 | W11 |
| | JA3 | Y10 | | JB3 | V10 |
| | JA4 | AA9 | | JB4 | W8 |
| | JA7 | AB11 | | JB7 | V12 |
| | JA8 | AB10 | | JB8 | W10 |
| | JA9 | AB9 | | JB9 | V9 |
| | JA10 | AA8 | | JB10 | V8 |

| Pmod | Signal Name | Zynq pin | Pmod | Signal Name | Zynq pin |
|---|---|---|---|---|---|
| JC1 Differential | JC1_N | AB6 | JD1 Differential | JD1_N | W7 |
| | JC1_P | AB7 | | JD1_P | V7 |
| | JC2_N | AA4 | | JD2_N | V4 |
| | JC2_P | Y4 | | JD2_P | V5 |
| | JC3_N | T6 | | JD3_N | W5 |
| | JC3_P | R6 | | JD3_P | W6 |
| | JC4_N | U4 | | JD4_N | U5 |
| | JC4_P | T4 | | JD4_P | U6 |

| Pmod | Signal Name | Zynq pin | MIO |
|---|---|---|---|
| JE1 MIO Pmod | JE1 | A6 | MIO13 |
| | JE2 | G7 | MIO10 |
| | JE3 | B4 | MIO11 |
| | JE4 | C5 | MIO12 |
| | JE7 | G6 | MIO0 |
| | JE8 | C4 | MIO9 |
| | JE9 | B6 | MIO14 |
| | JE10 | E6 | MIO15 |

### 2.9.3 Agile Mixed Signaling (AMS) Connector, J2

The XADC header provides analog connectivity for analog reference designs, including AMS daughter cards like Xilinx's AMS Evaluation Card.

The analog header is placed close to the LPC FMC header as shown. Both analog and digital IO can be easily supported for a plug in card. This allows the analog header to be easily connected to the FMC card using a short ribbon cable as shown. The analog header can also be used "stand alone" to support the connection of external analog signals.

The pin out has been chosen to provide tightly coupled differential analog pairs on the ribbon cable and to also provide AGND isolation between channels. The plug in cards which will facilitate a number of reference designs have not yet been designed so this pin out must provide a reasonable degree of freedom while also keeping resource requirements as minimal as possible.

## 2.10 Configuration Modes

Zynq-7000 AP SoC devices use a multi-stage boot process that supports both non-secure and secure boot (note that secure boot is not supported for CES silicon.) The PS is the master of the boot and configuration process. The following table shows the Zynq configuration modes. Upon reset, the device mode pins are read to determine the primary boot device to be used: NOR, NAND, Quad-SPI, SD Card or JTAG.

By default, the ZedBoard uses the SD Card configuration mode. The boot mode pins are MIO[8:2] and are used as follows:

- MIO[2]/Boot_Mode[3] sets the JTAG mode
- MIO[5:3]/Boot_Mode[2:0] select the boot mode
- MIO[6]/Boot_Mode[4] enables the internal PLL
- MIO[8:7]/Vmode[1:0] are used to configure the I/O bank voltages, however these are fixed on ZedBoard and not configurable

The ZedBoard provides jumpers for MIO[6:2]. These are 1x3 jumpers connected as shown below. All mode pins can be pulled high or low through a 20 KΩ resistor.

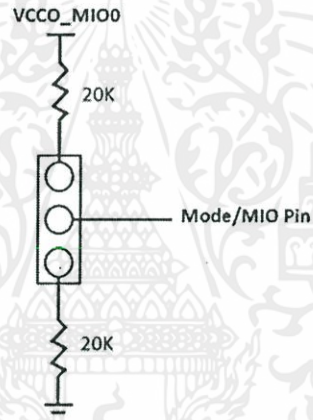



**Figure 15 - Configuration Mode Jumpers**

These jumpers allow users to change the mode options, including using cascaded JTAG configuration as well as using the internal PLL.

As noted above, the VMODE pins are strapped permanently to set Bank 500 and 501 voltages to 3.3V and 1.8V. These are not jumper selectable.

The PS boot mode selections are shown in the table below, default setting highlighted in yellow:

**Table 18 – ZedBoard Configuration Modes**

| Xilinx TRM→ | MIO[6] Boot_Mode[4] | MIO[5] Boot_Mode[0] | MIO[4] Boot_Mode[2] | MIO[3] Boot_Mode[1] | MIO[2] Boot_Mode[3] |
|---|---|---|---|---|---|
| JTAG Mode | | | | | |
| Cascaded JTAG | | | | | 0 |
| Independent JTAG | | | | | 1 |
| Boot Devices | | | | | |
| JTAG | | 0 | 0 | 0 | |
| Quad-SPI | | 1 | 0 | 0 | |
| SD Card | | 1 | 1 | 0 | |
| PLL Mode | | | | | |
| PLL Used | 0 | | | | |
| PLL Bypassed | 1 | | | | |
| Bank Voltages | | | | | |
| MIO Bank 500 | | | 3.3V | | |
| MIO Bank 501 | | | 1.8V | | |

Expected configuration time using a 50MB/s QSPI flash is 250ms.

PUDC_B is pulled high on ZedBoard but can be pulled low via JP5. This active-low input enables internal pull-ups during configuration on all SelectIO pins

A push button labeled "PROG" is connected to the Zynq PROG pin, T11, and pulled up. Pushing the button connects PROG to ground. Pushing this button will clear the PL configuration memory, effectively resetting the entire PL subsection. The PS is responsible for reconfiguring the PL. Zynq will not automatically reconfigure the PL as in standard FPGAs. The user software application must reprogram the PL. An interrupt will indicate a change in the devcfg.INT_STS register and the configuration pin status can be read from the devcfg.STATUS register. When configuration is complete, a blue LED, LD12, labeled DONE, will light.

### 2.10.1 JTAG

As an alternative to using the onboard USB-JTAG interface, the ZedBoard provides traditional Platform Cable JTAG connector, J15, for use with Xilinx Platform Cables and Digilent JTAG HS1 Programming Cables. The JTAG Chain is constructed as follows:

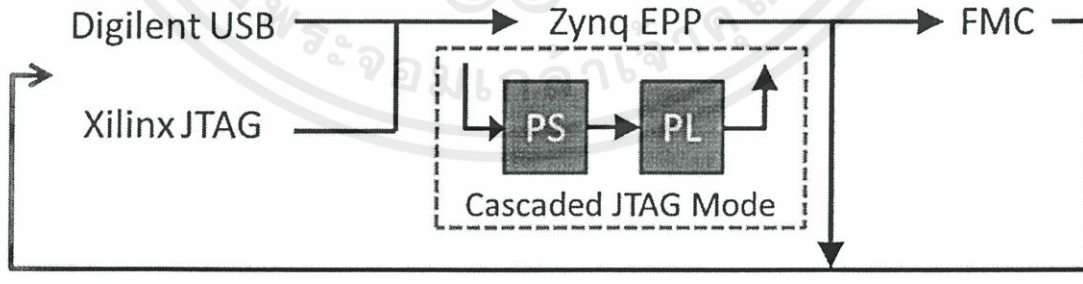



**Figure 16 - ZedBoard JTAG Chain**

# 4 Jumper Settings

**Table 22 - Jumper Settings**

| Ref Designator | Description | Default Setting | Function |
|---|---|---|---|
| JP1 | Microphone Input Bias | Open – No Electret Microphone | Short to enable Bias Voltage for Electret Microphone. Right Channel only. |
| JP2 | Vbus 5V Enable | Open – 5V Disconnected | Short to enable 5V output to USB OTG Connector, J13, for either Host or OTG modes. |
| JP3 | USB Vbus Capacitor Setting | Open – Device Mode | Short for Host mode (>120uF).<br>Open for Device or OTG modes (4.7uF). |
| JP4 | CFGBVS Select | Not Populated | Pre-configuration I/O standard type for the dedicated configuration bank 0. Vcco_0 is 3.3V, Connected to 3.3V through a 10K resistor. This jumper connects to GND and should NOT be used. |
| JP5 | PUDC Select | Not Populated | Active Low input enables internal pull-ups during configuration on all SelectIO pins. Connected to Vadj through 10K resistor. |
| JP6 | PS_MIO0 Pull-Down | Short | Install for SD Card boot on CES silicon. |
| JP7 | Boot_Mode[3]/MIO[2] | GND – Cascaded JTAG | JTAG Mode. GND cascades PS and PL JTAG chains. VCC makes JTAG chains independent. |
| JP8<br>JP9<br>JP10 | Boot_Mode[0]/MIO[3]<br>Boot_Mode[1]/MIO[4]<br>Boot_Mode[2]/MIO[5] | 110 – SD Card | Boot Device Select<br>See Zynq Configuration Modes |
| JP11 | Boot_Mode[4]/MIO[6] | GND – PLL Used | PLL Select. GND uses PS PLLs. VCC bypasses internal PS PLLs |
| JP12 | XADC Ferrite Bead Disable | Open | Short bypasses XADC-GND ferrite bead connection to board GND. |
| JP13 | JTAG PS-RST | Open | Short connects JTAG PROG-RST to PS Reset. |
| J18 | Vadj Select | 1.8V | Selects Vadj (1.8V, 2.5V, or 3.3V) |

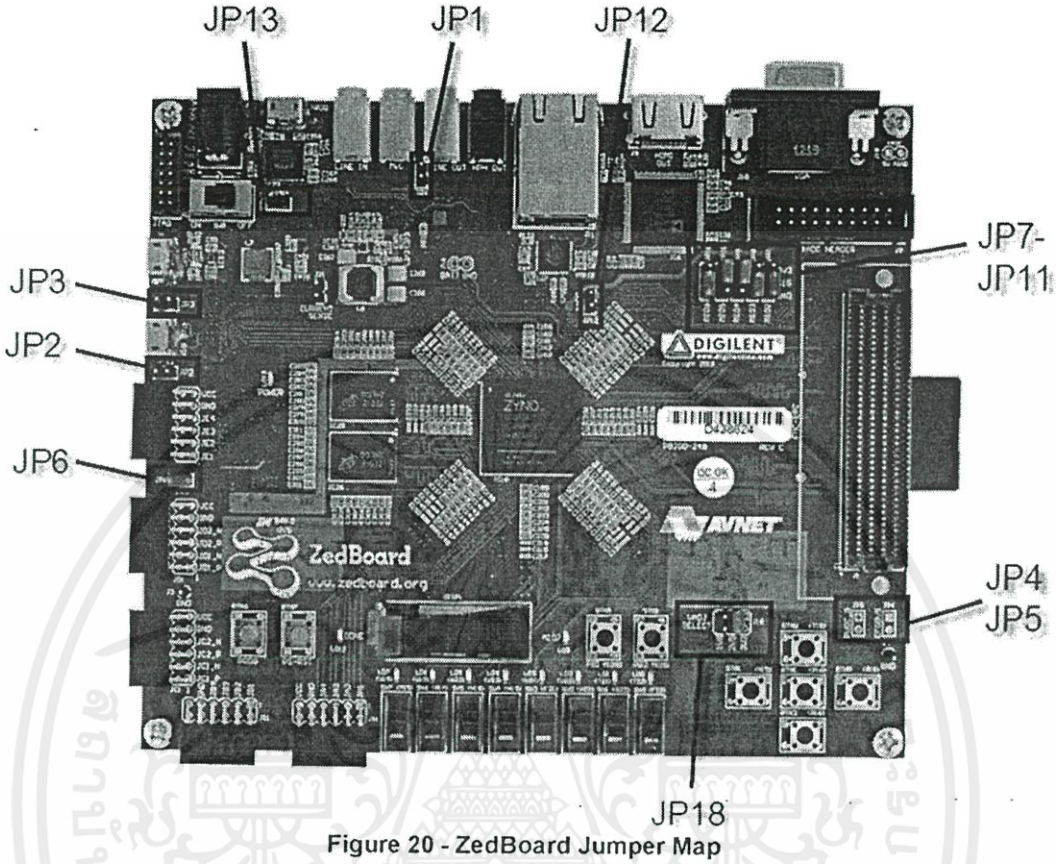



**Figure 20 - ZedBoard Jumper Map**

# ภาคผนวก ง

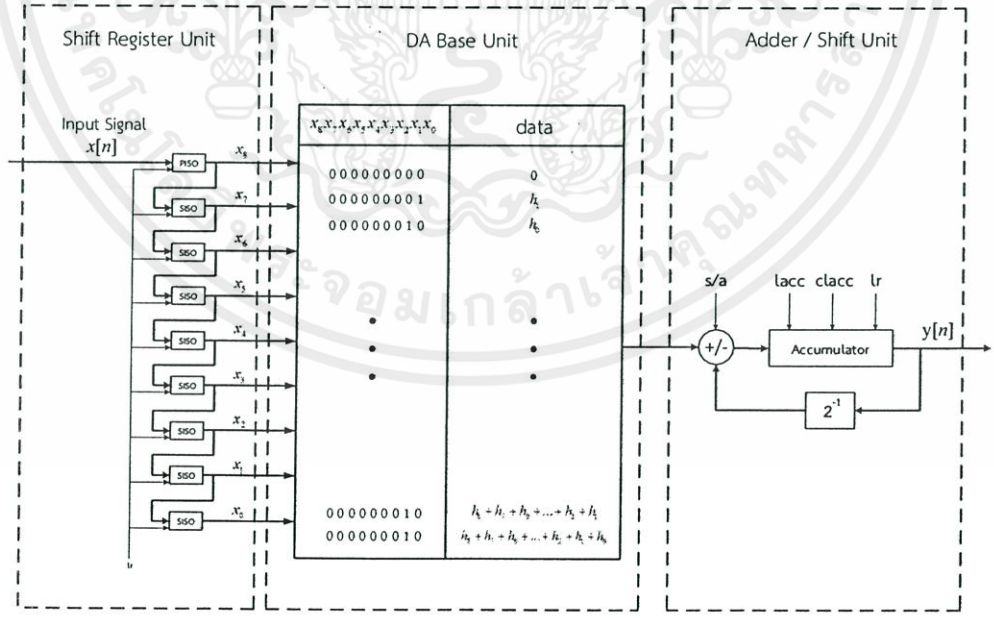
## การเปรียบเทียบประสิทธิภาพของโครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดู กับแบบปราศจากตารางเปิดดู

## 1. วงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดู (Conventional LUT-based Distributed Arithmetic Structure)

$$y[n]=\sum_{i=0}^{k-1} w_i x[n-i] \tag{1}$$

การสร้างวงจรกรองสัญญาณเชิงเลขแบบผลตอบสนองอิมพัลส์จำกัดดังสมการที่ (1) โดยตรงผ่านวงจรรวมขนาดใหญ่มาก (VLSI) ต้องใช้ตัวคูณและตัวเลื่อนบวกสะสมจำนวน K ตัวในการสร้างสัญญาณเอาต์พุต ซึ่งใช้เนื้อที่ฮาร์ดแวร์และการประมวลผลตรรกะค่อนข้างมาก ในการออกแบบ VLSI ผลของการคูณและเลื่อนบวกสะสมดังกล่าวจะถูกเก็บไว้แล้วในหน่วยความจำเรียกว่า “ตารางเปิดดู (LUT)” ทำให้หน่วยความจำมีขนาด $2^K$ -words

การทำงานของวงจรกรองสัญญาณเชิงเลข โดยอาศัยโครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดูประกอบด้วย 3 ส่วน คือ วงจรเลื่อนบิต ตารางเปิดดู และวงจรเลื่อนและบวกสะสม สัญญาณอินพุตจะถูกเลื่อนแต่ละบิตเพื่อเป็นตัวชี้ตารางเปิดดู และนำค่าออกมาผ่านวงจรเลื่อนและบวกสะสมต่อไป ดังรูปที่ 1.1






รูปที่ 1.1 วงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดู

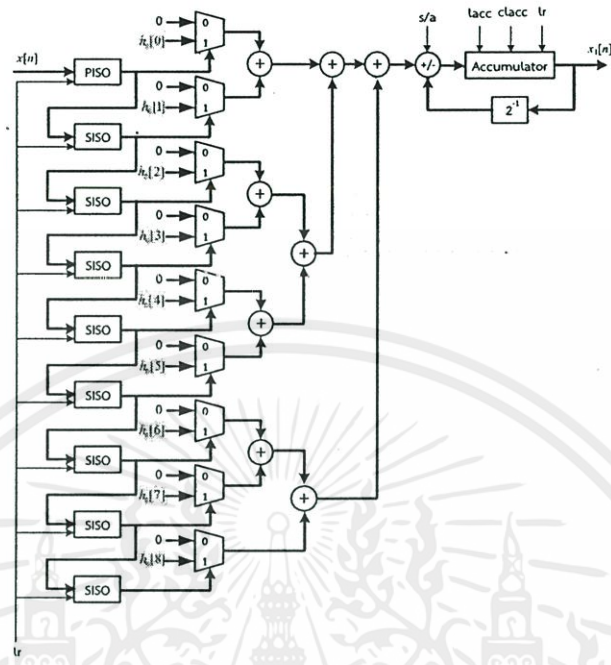
การสร้างวงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดูผ่านวงจร VLSI หรือวงจร FPGA มีจำนวนอุปกรณ์ต่าง ๆ ภายในวงจรดังรูปที่ 1.2 โดยจะพบว่ามีการใช้งานถึง 34 เปอร์เซ็นต์

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice Registers | 88 | 106400 | 0% |
| Number of Slice LUTs | 198 | 53200 | 0% |
| Number of fully used LUT-FF pairs | 73 | 213 | 34% |
| Number of bonded IOBs | 45 | 200 | 22% |
| Number of BUFG/BUFGCTRLs | 3 | 32 | 9% |

รูปที่ 1.2 จำนวน Logic Cell ที่ใช้ในการสร้างวงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดู

## 2. วงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู (LUT-less Distributed Arithmetic Structure)

การสร้างตารางเปิดดูมีข้อเสียคือจะต้องคำนวณค่าที่อยู่ในตารางทั้งหมดไว้ก่อน ซึ่งขนาดของตารางจะเป็นกำลังสองของจำนวนสัมประสิทธิ์ฟิลเตอร์ และเป็นผลทำให้จะต้องใช้หน่วยความจำที่มีขนาดเพิ่มขึ้นด้วย ดังนั้นจึงอาศัยวงจรมัลติเพล็กซ์เซอร์แทนการทำงานของวงจรเปิดดู ซึ่งมีข้อดีคือไม่ต้องคำนวณค่าก่อน และขนาดของหน่วยความจำที่ใช้เก็บค่าสัมประสิทธิ์มีขนาดลดลงด้วย การสร้างวงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูมีโครงสร้างดังรูปที่ 1.3




รูปที่ 1.3 วงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู

การสร้างวงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูผ่านวงจร VLSI หรือวงจร FPGA ใช้จำนวนอุปกรณ์ต่างๆ ภายในวงจรดังรูปที่ 1.4 โดยจะพบว่ามีการใช้งานเพียง 25 เปอร์เซ็นต์ ซึ่งทำให้เห็นว่าโครงโครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดูมีการลดการใช้งานของอุปกรณ์น้อยลงเมื่อเทียบกับโครงสร้างเลขคณิตกระจายแบบใช้ตารางเปิดดู

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 87 | 106400 | 0% |
| Number of Slice LUTs | 259 | 53200 | 0% |
| Number of fully used LUT-FF pairs | 71 | 275 | 25% |
| Number of bonded IOBs | 45 | 200 | 22% |
| Number of BUFG/BUFGCTRLs | 3 | 32 | 9% |

รูปที่ 1.4 จำนวน Logic Cell ที่ใช้ในการสร้างวงจรกรองสัญญาณเชิงเลขโดยใช้โครงสร้างเลขคณิตกระจายแบบปราศจากตารางเปิดดู