# การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์และการแปลงฮ้อนกลับแบบรวดเร็วที่มีขนาดเล็ก

# DESIGN OF A COMPACT FFT/IFFT PROCESSOR

รัฐประชา ชาญชลสมุทร
RATAPRACHA CHARNCHONSAMUTRA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2549

ISBN 974-15-2164-2

# การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์และการแปลงย้อนกลับแบบรวดเร็วที่มีขนาดเล็ก

# DESIGN OF A COMPACT FFT/IFFT PROCESSOR


รัฐประชา ชาญชลสมุทร

RATAPRACHA CHARNCHONSAMUTRA



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2549

ISBN 974 – 15 – 2164 – 2

# DESIGN OF A COMPACT FFT/IFFT PROCESSOR

RATAPRACHA CHARNCHONSAMUTRA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRONIC ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG



2006

ISBN 974 – 15 – 2164 – 2

**บัณฑิตวิทยาลัย**

**สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง**

**ใบรับรองวิทยานิพนธ์**

---

**หัวข้อวิทยานิพนธ์** การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์และการแปลงย้อนกลับแบบรวดเร็วที่มีขนาดเล็ก

DESIGN OF A COMPACT FFT/IFFT PROCESSOR

**นักศึกษา** นายรัฐประชา ชาญชลสมุทร

**รหัสประจำตัว** 44611300

**ปริญญา** วิศวกรรมศาสตรมหาบัณฑิต

**สาขาวิชา** วิศวกรรมอิเล็กทรอนิกส์

**อาจารย์ผู้ควบคุมวิทยานิพนธ์** รศ.ดร.อภินันท์ ธนชยานนท์

| คณะกรรมการสอบวิทยานิพนธ์ | | ลายมือชื่อ |
|---|---|---|
| ศ.ดร.วัลลภ | สุระกำพลธร | |
| ดร.กิติพล | ชิตสกุล | |
| ผศ.ดร.สุพันธุ์ | ตั้งจิตกุศลมั่น | |
| ดร.กสิน | วิเชียรชม | |
| รศ.ดร.อภินันท์ | ธนชยานนท์ | |

**วัน / เดือน/ ปี ที่สอบ** 20 ธันวาคม 2548 เวลา 14.15-16.15 น.

**สถานที่สอบ** ณ อาคาร 12 ชั้น ชั้น 4 (E12-404)

**บัณฑิตวิทยาลัยรับรองแล้ว**

**(ผศ.ดร.บำรุงตร. เจริญชน)**

**คณบดีบัณฑิตวิทยาลัย**

**วันที่.....ไ.....เดือน.....ม.ค.๒๕๔๖.....พ.ศ.๒๕๔๖...**

| | |
|---|---|
| **หัวข้อวิทยานิพนธ์** | การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์และการแปลงย้อนกลับแบบรวดเร็วที่มีขนาดเล็ก |
| **นักศึกษา** | นายรัฐประชา ชาญชลสมุทร |
| **รหัสนักศึกษา** | 44611300 |
| **ปริญญา** | วิศวกรรมศาสตรมหาบัณฑิต |
| **สาขาวิชา** | วิศวกรรมอิเล็กทรอนิกส์ |
| **พ.ศ.** | 2549 |
| **อาจารย์ผู้ควบคุมวิทยานิพนธ์** | รศ.ดร. อภินันท์ ธนชยานนท์ |

## บทคัดย่อ

วิทยานิพนธ์ฉบับนี้เป็นการนำเสนอ การออกแบบและการทดสอบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่มีขนาดเล็ก สำหรับการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน โดยรูปแบบของสถาปัตยกรรมที่นำเสนอจะใช้โครงสร้างการทำงานแบบ 1 หน่วยความจำ นำไปสู่การใช้อุปกรณ์ที่มีขนาดเล็ก โดยลำดับของข้อมูลที่เข้ามาจะถูกส่งเข้าไปยังส่วนประมวลผลของแผนภาพผีเสื้อเพื่อคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด โดยที่ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถคำนวณค่าได้อย่างมีประสิทธิภาพ สำหรับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วออกแบบและจำลองการทำงาน โดยใช้ภาษา VHDL รวมทั้งทดสอบการทำงานโดยใช้ชิพ XILINX SPARTAN-II FPGA โดยที่ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วใช้เกทในการทำงานทั้งหมด 37,281 เกท ขณะที่ส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วใช้เกทในการทำงานทั้งหมด 37,442 เกท ไม่รวมขนาดของหน่วยความจำ ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วสามารถทำงานได้ที่ความถี่เท่ากับ 53 MHz และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถทำงานได้ที่ความถี่เท่ากับ 52 MHz ทำให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อนให้เสร็จได้ภายในเวลา77 μsec และ 78 μsec ตามลำดับ

| | |
|---|---|
| **Thesis Title** | Design of a compact FFT/IFFT processor |
| **Student** | Mr. Ratapracha Charnchonsamutra |
| **Student ID** | 44611300 |
| **Degree** | Master of Engineering |
| **Programme** | Electronic Engineering |
| **Year** | 2006 |
| **Thesis** | Assoc. Prof. Dr. Apinunt Thanachayanont |

## ABSTRACT

This thesis presents the design and implementation of a compact 256-point complex FFT/IFFT processor. The proposed architecture uses a single-memory structure. To achieve a compact hardware realization, input data words are sequentially fetched into the main 2-point radix-2 butterfly processor so that the FFT/IFFT calculation can perform. The FFT/IFFT processor has been designed and simulated by VHDL, and implemented by using XILINX SPARTAN-II FPGA. The realized FFT processor occupies 37,281 gates where IFFT processor occupies 37,442 gates, excluding RAM. The FFT processor can operate at 53 MHz and the IFFT processor can operate at 52 MHz. The FFT/IFFT processor calculates a 256-point complex in 77 μsec and 78 μsec respectively.

# กิตติกรรมประกาศ

การที่วิทยานิพนธ์เล่มนี้สำเร็จลุล่วงลงได้ต้องขอขอบพระคุณ บิดา มารดา ทุกคนภายในครอบครัวของข้าพเจ้าที่คอยให้กำลังใจ เอาใจใส่ และให้การสนับสนุนที่ดีตลอดมา ขอขอบพระคุณ รองศาสตราจารย์ ดร. อภินันท์ ธนชยานนท์ และอาจารย์ทุกท่านเป็นอย่างสูง ที่คอยให้คำปรึกษา คำแนะนำ ตลอดจนแนวทางการแก้ไขปัญหาต่างๆ

ขอขอบพระคุณพี่ๆ เพื่อนๆ และน้องๆ ทุกท่านใน LAB MDRD ที่ให้กำลังใจ และความช่วยเหลือตลอดมา

ขอขอบพระคุณโครงการสำนักวิจัยการสื่อสารและเทคโนโลยีสารสนเทศ (ReCCIT) ที่ให้การสนับสนุนการทำวิจัยมาโดยตลอด

ขอขอบพระคุณบัณฑิตวิทยาลัย ที่ได้สนับสนุนการทำวิจัยครั้งนี้

คุณค่าและประโยชน์อันใดอันพึงมีจากวิทยานิพนธ์เล่มนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

รัฐประชา ชาญชลสมุทร

# สารบัญ

หน้า

# สารบัญ (ต่อ)

หน้า

# สารบัญ (ต่อ)

หน้า

# สารบัญตาราง

# สารบัญภาพ

# สารบัญภาพ (ต่อ)

# สารบัญภาพ (ต่อ)

# บทที่ 1
# บทนำ

## 1.1 กล่าวนำ

การแปลงฟูริเยร์แบบไม่ต่อเนื่อง (Discrete Fourier Transform : DFT) เป็นหนึ่งในวิธีการที่นิยมนำมาใช้ในงานที่เกี่ยวกับการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) เนื่องจากการแปลงฟูริเยร์แบบไม่ต่อเนื่องมีประโยชน์ในการใช้งานอย่างมาก จึงได้มีความพยายามคิดค้นหาวิธีที่จะคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่องให้เร็วขึ้น และมีประสิทธิภาพมากขึ้น ซึ่งก็คือ การแปลงฟูริเยร์แบบรวดเร็ว (Fast Fourier Transform : FFT) โดยมีการนำเอาการแปลงฟูริเยร์แบบรวดเร็ว ไปประยุกต์ใช้งานกันอย่างหลากหลายเช่น การสื่อสาร และเครื่องมือวัด

ในการสื่อสารทั้งการสื่อสารแบบมีสายและไร้สาย โดยทั่วไปการแปลงฟูริเยร์แบบรวดเร็วและการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว (Inverse Fast Fourier Transform : IFFT) มีการนำไปใช้งานที่ภาครับข้อมูล และภาคส่งข้อมูลของระบบตามลำดับ โดยการทำงานของการแปลงฟูริเยร์แบบรวดเร็วนั้นจะนำมาใช้ในการแปลงข้อมูล โดยแปลงจากข้อมูลในเชิงเวลามาเป็นข้อมูลในเชิงความถี่ที่ภาครับข้อมูล และการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วจะใช้ในการแปลงสัญญาณในเชิงความถี่กลับมาเป็นสัญญาณในเชิงเวลาที่ภาคส่งข้อมูล

ซึ่งปัจจุบันระบบต่างๆ เหล่านี้ได้มีการพัฒนาให้มีขนาดที่เล็กลง เพื่อความสะดวกในการนำไปใช้งาน และประหยัดพลังงาน จึงทำให้วงจรต่างๆ ภายในระบบได้รับการพัฒนาให้มีขนาดที่เล็กลง ดังนั้นส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว (FFT processor) และส่วนประมวลผลการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว (IFFT processor) ที่จะนำไปใช้งานจึงต้องมีการปรับปรุงให้มีขนาดที่เล็กลงตามไปด้วย

## 1.2 วัตถุประสงค์ของวิทยานิพนธ์

วัตถุประสงค์ของวิทยานิพนธ์เล่มนี้ ต้องการที่จะลดขนาดของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็ก เมื่อพิจารณาการแปลงฟูริเยร์แบบรวดเร็ววิธีการคำนวณแบบต่างๆ พบว่า การแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 (Radix-2) แบบแตกเป็นส่วนย่อยทางฝั่งเวลา (Decimation In Time : DIT) มีสมการสำหรับการคำนวณทางคณิตศาสตร์ที่มีความซับซ้อนไม่มากนัก ทำให้สามารถนำมาออกแบบให้มีโครงสร้างการทำงานที่มีขนาดเล็กได้ จึงได้นำการคำนวณวิธีระบบฐาน 2 มาใช้เป็นโครงสร้างพื้นฐานในปรับปรุงและการออกแบบ ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็ก เมื่อพิจารณาโครงสร้างการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วพบว่า ส่วนประมวลผลของแผนภาพผีเสื้อ (Butterfly processor) จะมีขนาดใหญ่ที่สุด ดังนั้นถ้าสามารถลดขนาดส่วนประมวลผลของแผนภาพผีเสื้อให้มีขนาดที่เล็กลงได้ ก็จะมีผลทำให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วมีขนาดที่เล็กลง ดังนั้นวิทยานิพนธ์เล่มนี้จึงได้นำเสนอการวิเคราะห์ และการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็ก โดยการลดขนาดส่วนประมวลผลของแผนภาพผีเสื้อ ซึ่งเลือกใช้พื้นฐานการทำงานจาก วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลามาใช้ในการออกแบบ ซึ่งการทำงานทั้งหมดออกแบบโดยใช้ภาษา VHDL และนำไปทดสอบการทำงานบนชิพ Field Programmable Gate Arrays (FPGAs)

## 1.3 รายละเอียดของวิทยานิพนธ์

ในเนื้อหาของวิทยานิพนธ์เล่มนี้ประกอบด้วย 6 บท และ 6 ภาคผนวก โดยกล่าวถึงการวิเคราะห์ และการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดที่เล็ก โดยเลือกใช้พื้นฐานการทำงานจากวิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา โดยรายละเอียดในแต่ละบทที่นำเสนอมีดังนี้

บทที่ 2 กล่าวถึงวิธีการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว และการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว รวมทั้งการนำไปประยุกต์ใช้ในการสื่อสาร

บทที่ 3 เป็นการอธิบายถึงการออกแบบวงจรคำนวณทางคณิตศาสตร์คือ วงจรบวก วงจรลบ วงจรคูณ และวงจรหาร เพื่อรองรับการทำงานของระบบเลขอิงดรรชนีขนาด 32 บิต สำหรับนำไปใช้งานในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

บทที่ 4 เป็นการอธิบายถึงการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็ก โดยปรับปรุงส่วนประมวลผลของแผนภาพผีเสื้อให้มีขนาดเล็กลง ด้วยการลดขนาดโครงสร้างการทำงานของส่วนประมวลผลของแผนภาพผีเสื้อให้เหลือเพียง 2 วงจรคูณ และ 2 วงจรบวก รวมทั้งการออกแบบวงจรต่างๆ ที่ประกอบอยู่ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว



บทที่ 5 เป็นการเอาส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว รวมทั้งวงจรต่างๆ ที่ออกแบบเสร็จแล้วมาจำลองการทำงาน (Simulation) บนเครื่องคอมพิวเตอร์เพื่อตรวจสอบการทำงาน หลังจากนั้นเป็นการนำส่วน

ประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วมาโปรแกรมลงบนบอร์ด FPGA เพื่อทดสอบการทำงาน

บทที่ 6 เป็นบทที่กล่าวถึง การสรุปงานวิจัยของวิทยานิพนธ์เล่มนี้และยังกล่าวถึงข้อเสนอแนะต่างๆรวมถึงแนวทางในการพัฒนาต่อไป

# บทที่ 2
# การแปลงฟูริเยร์แบบรวดเร็ว

## 2.1 บทนำ

การแปลงฟูริเยร์แบบรวดเร็วได้รับการพัฒนาขึ้นมาจากการแปลงฟูริเยร์แบบไม่ต่อเนื่อง เพื่อให้สามารถทำการแปลงฟูริเยร์แบบไม่ต่อเนื่องได้เร็วมากขึ้น เนื่องจากการแปลงฟูริเยร์แบบไม่ต่อเนื่องสามารถที่จะกระทำการแปลงนี้โดยใช้การคำนวณ (การคูณ และการบวก) ทางดิจิตอลได้ ซึ่งสามารถนำไปประยุกต์ใช้ได้สะดวกมากในคอมพิวเตอร์ หรือในฮาร์ดแวร์โดยตรง [1] จึงเป็นที่นิยมในการนำไปประยุกต์ใช้งานภายในระบบต่างๆ รวมทั้งการสื่อสาร ซึ่งปัจจุบันได้มีการพัฒนาให้ระบบมีขนาดที่เล็กลง เพื่อสะดวกต่อการนำไปใช้งาน และการใช้พลังงานที่ต่ำลง

ในบทนี้ได้นำเสนอเกี่ยวกับการพัฒนาการแปลงฟูริเยร์แบบไม่ต่อเนื่อง มาเป็นการแปลงฟูริเยร์แบบรวดเร็ว และการนำไปประยุกต์ใช้ในภาครับและภาคส่งของระบบสื่อสาร โดยหัวข้อที่ 2.2 จะอธิบายรูปแบบ และการคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่อง หัวข้อที่ 2.3 นำเสนอเกี่ยวกับการนำการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา ไปประยุกต์ใช้งานในรูปแบบต่างๆ รวมทั้งข้อได้เปรียบ และวิธีการคำนวณของการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา หัวข้อที่ 2.4 นำเสนอวิธีการคำนวณการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว หัวข้อที่ 2.5 นำเสนอเทคนิคการเติมศูนย์สำหรับการแก้ไขข้อจำกัดของการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 หัวข้อที่ 2.6 นำเสนอการแปลงฟูริเยร์แบบรวดเร็วและการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว กับการนำมาประยุกต์ใช้ในภาครับและภาคส่งของระบบสื่อสาร หัวข้อที่ 2.7 เป็นบทสรุป

## 2.2 การแปลงฟูริเยร์แบบไม่ต่อเนื่อง (Discrete Fourier Transform : DFT)

การแปลงฟูริเยร์แบบไม่ต่อเนื่อง เป็นการแปลงสัญญาณจากสัญญาณในเชิงเวลาไปเป็นสัญญาณในเชิงความถี่ หรือทำการแปลงย้อนกลับจากสัญญาณในเชิงความถี่ไปเป็นสัญญาณในเชิงเวลา ดังนั้นการแปลงฟูริเยร์แบบไม่ต่อเนื่อง จะมีทั้งสัญญาณในเชิงเวลา และในเชิงความถี่เป็นแบบไม่ต่อเนื่อง ซึ่งจุดนี้เป็นจุดที่สำคัญมาก เพราะเป็นจุดที่บ่งบอกว่า สามารถที่จะทำการแปลงฟูริเยร์แบบไม่ต่อเนื่องได้โดยใช้การคำนวณ (การคูณ และการบวก) ทางดิจิตอลได้ ซึ่งสามารถนำมาประยุกต์ใช้งานได้อย่างสะดวกในคอมพิวเตอร์ หรือในฮาร์ดแวร์โดยตรง

การแปลงฟูริเยร์แบบไม่ต่อเนื่อง มีความหมายเหมือนกับ อนุกรมฟูริเยร์แบบไม่ต่อเนื่อง (Discrete Fourier Series : DFS) มาก ทั้งการแปลงฟูริเยร์แบบไม่ต่อเนื่อง และอนุกรมฟูริเยร์แบบไม่ต่อเนื่องจะมีสมการที่ใช้งานเหมือนกัน จุดที่ต่างกันก็คือ ที่มาและความหมายของทั้งสอง อนุกรมฟูริเยร์แบบไม่ต่อเนื่องคือ การใช้งานในกรณีที่สัญญาณในเชิงเวลาเป็นแบบไม่ต่อเนื่อง และเป็นคาบ ซึ่งก็จะได้สัญญาณในเชิงความถี่เป็นแบบไม่ต่อเนื่อง และเป็นคาบเช่นเดียวกัน ส่วนการแปลงฟูริเยร์แบบไม่ต่อเนื่องนั่น เป็นการเอาความจริงที่เกิดขึ้นจากอนุกรมฟูริเยร์แบบไม่ต่อเนื่องมาใช้ คือ สัญญาณที่ได้จากการแปลงทั้งสัญญาณในเชิงเวลา และสัญญาณในเชิงความถี่เป็นจะแบบไม่ต่อเนื่อง รวมทั้งสัญญาณที่ได้จะเป็นรายคาบทั้งในเชิงเวลา และและในเชิงความถี่

ดังนั้นการแปลงฟูริเยร์แบบไม่ต่อเนื่อง ก็คือ อนุกรมฟูริเยร์แบบไม่ต่อเนื่อง เพียงคาบเดียวเท่านั้น ซึ่งสัญญาณในเชิงเวลาเป็นสัญญาณไม่ต่อเนื่องมีความยาวจำกัดเท่ากับ N และเป็นสัญญาณที่มีรูปร่างใดๆ ก็ตาม เมื่อทำการแปลงฟูริเยร์แบบไม่ต่อเนื่องแล้ว จะได้สัญญาณในเชิงความถี่เป็นสัญญาณไม่ต่อเนื่อง และมีความยาวจำกัดเท่ากับ N เท่ากัน

สมมติให้ x(n) เป็นสัญญาณในเชิงเวลา และ X(k) เป็นสัญญาณในเชิงความถี่ที่เกิดจากการแปลงฟูริเยร์แบบไม่ต่อเนื่อง โดย k แทนตัวชี้ลำดับของสัญญาณทางด้านความถี่ ทั้งสองสัญญาณมีความยาวเท่ากันคือ N ทำให้เขียนสัญลักษณ์ได้ว่า

$$\text{DFT; N}$$

$$x(n) \text{ <----> } X(k)$$

ซึ่งจะได้ความสัมพันธ์กันของ x(n) และ X(k) ดังนี้

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad (2.1)$$

เพื่อจัดรูปสมการให้ง่ายขึ้นจึงนิยามให้ $W_N = e^{-j2\pi/N}$ เป็นค่าที่ขึ้นกับ N เท่านั้น สำหรับในการแปลงครั้งหนึ่งๆ N จะมีค่าคงที่ ดังนั้น $W_N$ จึงเสมือนเป็นค่าคงที่ สามารถเขียนการแปลงฟูริเยร์แบบไม่ต่อเนื่อง ได้เป็น

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad \text{โดยที่ } W_N = e^{-j2\pi/N} \text{ และ } k = 0, 1, 2, \ldots, N\text{-}1 \qquad (2.2)$$

หรือเขียนในรูปของเมตริกซ์ได้เป็น

$$\begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & \cdots & W_N^0 \\ W_N^0 & W_N^{1x1} & W_N^{1x2} & \cdots & W_N^{1x(N-1)} \\ W_N^0 & W_N^{2x1} & W_N^{2x2} & \cdots & W_N^{2x(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ W_N^0 & W_N^{(N-1)x1} & W_N^{(N-1)x2} & \cdots & W_N^{(N-1)x(N-1)} \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix} \quad (2.3)$$

## 2.3 การแปลงฟูริเยร์แบบรวดเร็ว (Fast Fourier Transform : FFT)

เนื่องจากการแปลงฟูริเยร์แบบไม่ต่อเนื่อง มีประโยชน์ในการใช้งานเป็นอย่างมาก จึงได้มีความพยายามคิดค้นหาวิธีที่จะคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่องให้เร็วขึ้น และมีประสิทธิภาพมากขึ้น ดังนั้นการแปลงฟูริเยร์แบบรวดเร็ว ก็คือชื่อที่ใช้เรียก วิธีการคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่องอย่างรวดเร็วกว่าการคิดปกตินั้นเอง เพราะฉะนั้นเมื่อกล่าวถึงการแปลงฟูริเยร์แบบรวดเร็วโดยหลักการแล้วก็คือ การแปลงฟูริเยร์แบบไม่ต่อเนื่องนั่นเอง และการแปลงฟูริเยร์แบบรวดเร็ว ก็ไม่ใช่การแปลงชนิดใหม่แต่อย่างใด

ปัจจุบันมีการคิดค้นวิธีการคำนวณ การแปลงฟูริเยร์แบบไม่ต่อเนื่องอย่างรวดเร็วได้หลายวิธี ดังนั้นคำว่า การแปลงฟูริเยร์แบบรวดเร็ว จึงเป็นชื่อกลางๆ ไม่ได้บ่งบอกว่าเป็นวิธีไหน แต่ในวิทยานิพนธ์เล่มนี้เลือกใช้ การแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 (Radix-2) แบบแตกเป็นส่วนย่อยทางฝั่งเวลา (Decimation In Time : DIT) มาเป็นพื้นฐานในการออกแบบ ซึ่งวิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลานี้ ได้มีการนำไปใช้งานกันอย่างหลากหลาย เช่น

การนำไปประยุกต์ใช้ในเครื่องมือวัดแบบ Real-Time Spectrum Analysis [2] เป็นการออกแบบมาเพื่อรองรับการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน และสามารถคำนวณข้อมูลทั้งหมดให้เสร็จได้ภายในเวลา 102.4 μs (ไม่คิดเวลาช่วงที่มีการรับ และส่งข้อมูลจากภายนอก) โดยสามารถทำงานที่ความถี่นาฬิกาสูงสุดเท่ากับ 40 MHz

การนำไปออกแบบเพื่อให้วงจรมีประสิทธิภาพสูง (high performance) และใช้พลังงานต่ำ (low power) [3] เป็นการออกแบบให้สามารถรองรับการคำนวณข้อมูลจำนวน 1,024 จุดแบบจำนวนเชิงซ้อน ผลที่ได้คือสามารถคำนวณข้อมูลทั้งหมดให้เสร็จได้ภายในเวลา 330 μs โดยใช้กำลังงาน 9.5 mW ที่ 3.3 V และสามารถทำงานที่ความถี่นาฬิกาสูงสุดคือ 173 MHz



การนำไปประยุกต์ใช้ในการสื่อสารแบบระบบโครงข่ายแบบไร้สายภายในประเทศ (WLAN) ของระบบ OFDM [4] เป็นการออกแบบมาเพื่อรองรับการคำนวณข้อมูลจำนวน 64 จุดแบบจำนวนเชิงซ้อน โดยมีความกว้างของข้อมูลจำนวน 20 บิต และได้มีการออกแบบมาเพื่อให้

วงจรใช้พลังงานต่ำ และสามารถรองรับการคำนวณข้อมูลจำนวน 64 จุดเช่นกัน [5] โดยผลที่ได้จากการออกแบบสามารถทำงานที่ความถี่นาฬิกาสูงสุดคือ 33 MHz และสามารถคำนวณข้อมูลทั้งหมดได้เสร็จภายใน 92 สัญญาณนาฬิกา (clock cycle)

เนื่องจากโครงสร้างการคำนวณทางคณิตศาสตร์ของการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลามีความซับซ้อนไม่มากนัก รวมทั้งตัวแปรต่างๆ มีจำนวนน้อย ทำให้ง่ายต่อการนำมาออกแบบ เมื่อเทียบกับการแปลงฟูริเยร์แบบรวดเร็ววิธีอื่นๆ [6] ด้วยเหตุนี้จึงสามารถออกแบบให้มีโครงสร้างการทำงานที่มีขนาดเล็ก และใช้พลังงานต่ำได้ [3,7] สามารถอธิบายการคำนวณของการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา ได้ดังนี้

จากการแปลงฟูริเยร์แบบไม่ต่อเนื่อง ในสมการที่ 2.2

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad \text{โดยที่ } W_N = e^{-j2\pi/N} \text{ และ } k = 0, 1, 2, \ldots, N-1$$

ถ้าให้ N เป็นเลขคู่ ทำให้สามารถกระจาย X(k) ให้อยู่ในรูปของผลบวกของเทอมที่ n เป็นคู่และเทอมที่ n เป็นคี่ได้

$$X(k) = \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2kn}}_{\text{เทอมคู่}} + \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{(2n+1)k}}_{\text{เทอมคี่}} \tag{2.4}$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{2nk} W_N^{k} \tag{2.5}$$

พิจารณาเทอม $W_N^{ab}$ ที่มี a และ b เป็นจำนวนใดๆ ที่ไม่เท่ากับ 0 จะพบว่า สามารถย้ายตัวยกกำลังของ $W$ ไปเป็นตัวหารของ N ได้ดังนี้

$$W_N^{ab} = e^{-j\frac{2\pi}{N}ab} = e^{-j\frac{2\pi}{N/b}a} = W_{N/b}^{a} \tag{2.6}$$

จึงใช้ความจริงข้อนี้ แทนค่าเทอม $W_N^{2nk}$ ด้วย $W_{N/2}^{nk}$ ในสมการที่ 2.5 จะได้

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_{N/2}^{nk} \times W_N^{k} \tag{2.7}$$



จะเห็นได้ว่า X(k) ได้กลายเป็นผลบวกของสองเทอม แต่ละเทอมเป็นรูปแบบของการคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่อง N/2 จุด โดยเทอมแรกกระทำกับสัญญาณ x(0),

x(2),...,x(N-2) และเทอมที่สองกระทำกับสัญญาณ x(1), x(3),...,x(N-1) โดยที่ค่า $W_N^k$ จะเป็นค่าสัมประสิทธิ์ที่ใช้สำหรับคูณเทอมดี่

ซึ่งถ้าทำการแตกเทอมของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง N/2 จุดที่อยู่ในสมการที่ 2.7 ต่อไป แต่ละเทอมก็จะสามารถกระจายให้กลายเป็นผลบวกของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง N/4 จุดสองเทอม และสามารถกระจายเช่นนี้ต่อไปเรื่อยๆ จนกระทั่งทุกตัวอยู่ในรูปของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด ซึ่งการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุดสามารถคำนวณได้ดังนี้

สมมติให้ x(n) ยาว 2 จุด จะได้

$$X(k) = \sum_{n=0}^{1} x(n) W_2^{kn} \tag{2.8}$$

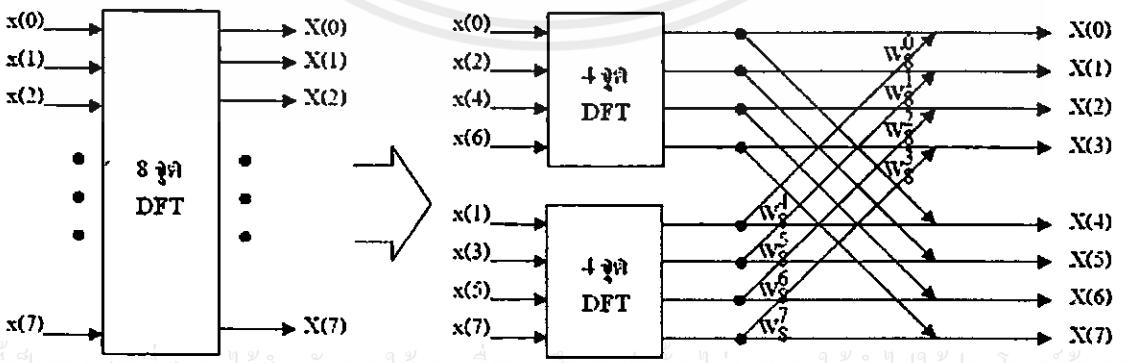
อาศัยความจริงที่ว่า $W_2^0 = 1$ และ $W_2^1 = e^{-j\pi} = -1$ จะได้ว่า

$$\begin{aligned} X(0) &= x(0) + x(1) \\ X(1) &= x(0) - x(1) \end{aligned} \tag{2.9}$$

ขั้นตอนที่อธิบายทั้งหมดนี้รวมเรียกว่า การแปลงฟูริเยร์แบบรวดเร็ว โดยทั่วไปนิยมเขียนการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว โดยใช้แผนภาพที่เรียกว่า แผนภาพผีเสื้อ (butterfly diagram)

แสดงขั้นตอนการคิดแผนภาพผีเสื้อสำหรับการแปลงฟูริเยร์แบบรวดเร็ว เมื่อ N = 8

เริ่มจากกระจายการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 8 จุด ให้อยู่ในรูปของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 4 จุด สองเทอมบวกกัน สามารถเขียนเป็นแผนภาพแสดงในรูปที่ 2.1




**รูปที่ 2.1** แสดงการกระจายการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 8 จุด เป็นการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 4 จุด

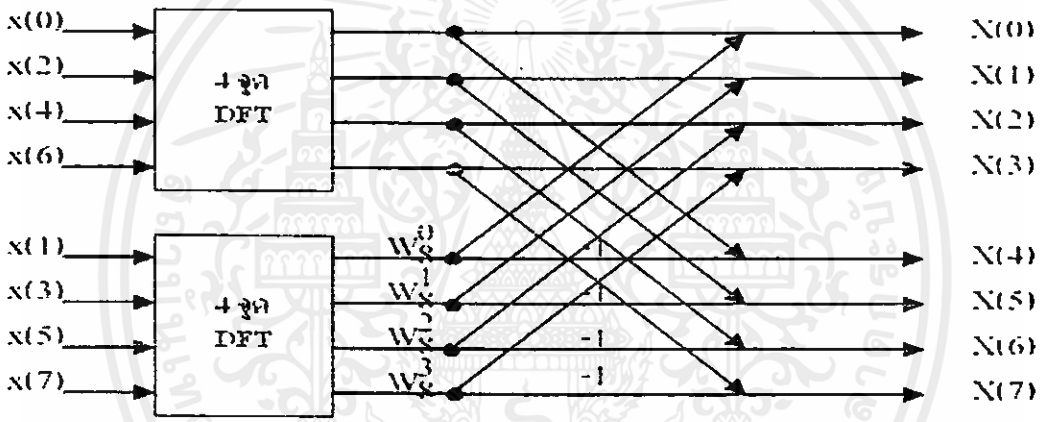
จากรูปที่ 2.1 สามารถทำค่าสัมประสิทธิ์ที่คูณอยู่ในแผนภาพให้ง่ายลงได้ โดยใช้คุณสมบัติความสมมาตรของ $W_N$ ดังนี้

$$W_N^{k+N/2} = W_N^k W_N^{N/2} = W_N^k(-1) = -W_N^k \tag{2.10}$$

ใช้คุณสมบัติตามสมการที่ 2.10 จะได้ว่า

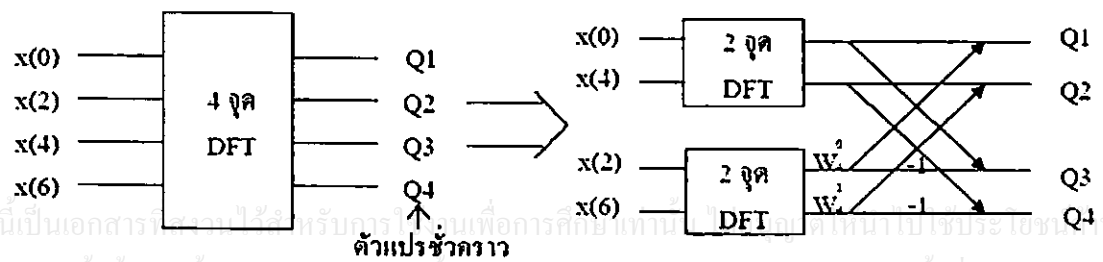
$$W_8^4 = -W_8^0,\ W_8^5 = -W_8^1,\ W_8^6 = -W_8^2 \text{ และ } W_8^7 = -W_8^3 \tag{2.11}$$

แทนค่าทั้งหมดลงในแผนภาพรูปที่ 2.1 จะได้แผนภาพ แสดงในรูปที่ 2.2




**รูปที่ 2.2** แสดงการกระจายการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 8 จุด เป็นการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 4 จุด หลังจากใช้คุณสมบัติของความสมมาตร

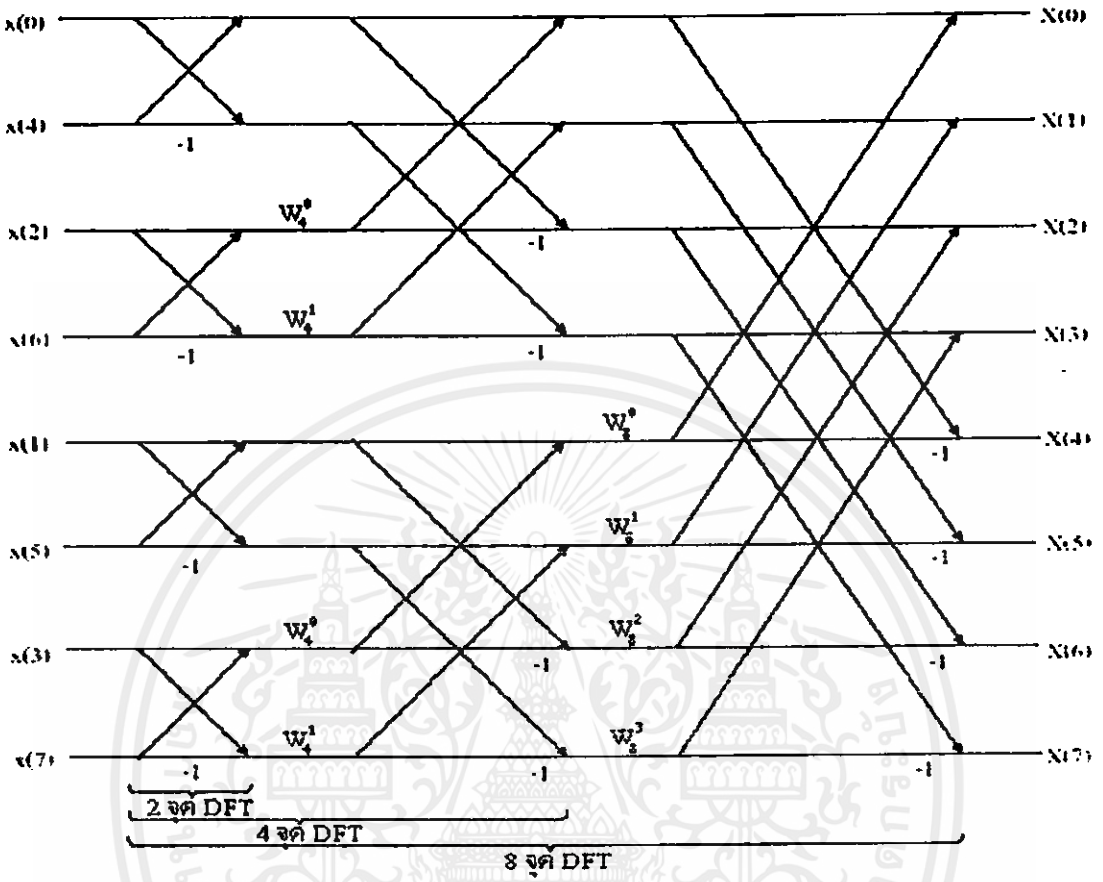
ในทำนองเดียวกันการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 4 จุด ก็สามารถกระจายเป็นการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุดได้ แสดงในรูปที่ 2.3






**รูปที่ 2.3** แสดงการกระจายการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 4 จุด เป็นการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด

เมื่อรวมผลลัพธ์ของแต่ละส่วนเข้าเป็นแผนภาพเดียวกัน แสดงในรูปที่ 2.4




**รูปที่ 2.4** แผนภาพรวมของการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว 8 จุด

จากรูปที่ 2.4 สามารถนำแผนภาพของการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว 8 จุดไปใช้เป็นแนวทางในการเขียนแผนภาพสำหรับการแปลงฟูริเยร์แบบรวดเร็ว N จุดใดๆ ได้ รวมทั้งใช้เป็นแนวทางในการเขียนโปรแกรมเพื่อคำนวณการแปลงฟูริเยร์แบบรวดเร็ว N จุดใดๆ ได้ด้วยเช่นกัน

ข้อสังเกตจากแผนภาพผีเสื้อของการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว มีดังนี้คือ

1. ถ้าต้องการได้ผลลัพธ์ในเชิงความถี่เรียงตามลำดับจาก X(0), X(1),..., X(7) ต้องทำการเรียงลำดับสัญญาณขาเข้าใหม่เป็น x(0), x(4), x(2), x(6), x(1), x(5), x(3) และ x(7) แสดงการเขียนของลำดับเหล่านี้ในเลขฐานสอง แสดงในตารางที่ 2.1 จะเห็นได้ว่าลำดับของสัญญาณใหม่เกิดจากการเรียงลำดับบิตจากหลังไปหน้าของลำดับสัญญาณปกติ (bit reversed order) ซึ่งพบว่าเป็นจริงสำหรับการแปลงฟูริเยร์แบบรวดเร็วที่จำนวน N จุดใดๆ ด้วย

**ตารางที่ 2.1** แสดงการเขียนลำดับของสัญญาณขาเข้า

| ลำดับใหม่ฐานสิบ | ลำดับใหม่ฐานสอง | ลำดับปกติฐานสิบ | ลำดับปกติฐานสอง |
|---|---|---|---|
| 0 | 000 | 0 | 000 |
| 4 | 100 | 1 | 001 |
| 2 | 010 | 2 | 010 |
| 6 | 110 | 3 | 011 |
| 1 | 001 | 4 | 100 |
| 5 | 101 | 5 | 101 |
| 3 | 011 | 6 | 110 |
| 7 | 111 | 7 | 111 |

2. ค่าสัมประสิทธิ์ $W$ ซึ่งเป็นค่าคงที่ที่ใช้คูณกับเทอมดี่ สามารถเปลี่ยนให้อยู่ในรูปของ $W_8^k$ ที่ค่า k ต่างๆ ได้ โดยการคูณตัวห้อย และตัวยกกำลังด้วยค่าเดียวกัน ดังนี้

$$W_4^0 \text{ --> } W_8^0 \text{ และ } W_4^1 \text{ --> } W_8^2 \qquad (2.12)$$

ทำให้สามารถใช้ $W_8^k$ แทนค่าได้ทั้งหมด ซึ่งสามารถคำนวณค่า $W_8^k$ ที่ค่า k ต่างๆ นี้ไว้ล่วงหน้าได้ และใช้เสมือนเป็นค่าคงที่สำหรับการแปลงฟูริเยร์แบบรวดเร็ว 8 จุด ซึ่งก็เป็นจริงเช่นกันสำหรับการแปลงฟูริเยร์แบบรวดเร็วที่จำนวน N จุดใดๆ

3. วิธีระบบฐาน 2 นี้จะใช้ได้ก็ต่อเมื่อค่า N เท่ากับ $2^b$ โดย b เป็นจำนวนเต็มบวกใดๆ ซึ่งข้อนี้ไม่เป็นปัญหา เนื่องจากถ้าไม่สามารถแบ่งสัญญาณให้มีความยาวเท่ากับ $2^b$ ได้ ก็ใช้วิธีเติมศูนย์เพิ่ม (zero padding) [1] เข้าไปในสัญญาณให้ได้ความยาวตามที่ต้องการ

## 2.4 การแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว (Inverse Fast Fourier Transform : IFFT)

เมื่อพิจารณาสูตรของการแปลงฟูริเยร์แบบไม่ต่อเนื่องเทียบกับ การแปลงฟูริเยร์ย้อนกลับแบบไม่ต่อเนื่อง (Inverse Discrete Fourier Transform : IDFT) ก็จะพบว่าการหาการแปลงฟูริเยร์ย้อนกลับแบบไม่ต่อเนื่อง สามารถหาได้โดยการใช้การแปลงฟูริเยร์แบบไม่ต่อเนื่อง ดังสมการต่อไปนี้



$$x(n) = \text{IDFT}(X) = \frac{1}{N}(\text{DFT}(X^*))^* \qquad (2.13)$$

เครื่องหมาย * หมายถึง conjugate สามารถพิสูจน์สมการที่ 2.13 โดยใช้สมการของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง และแทนค่า X* ลงไปจะได้

$$\text{DFT}(X^*) = \sum_{k=0}^{N-1} X(k)^* e^{-j2\pi kn/N} \tag{2.14}$$

ดังนั้นจะได้ค่า $\frac{1}{N}(\text{DFT}(X^*))^*$ เป็น

$$\frac{1}{N}(\text{DFT}(X^*))^* = \frac{1}{N}\sum_{k=0}^{N-1}\{X(k)^* e^{-j2\pi kn/N}\}^* = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi kn/N} \tag{2.15}$$

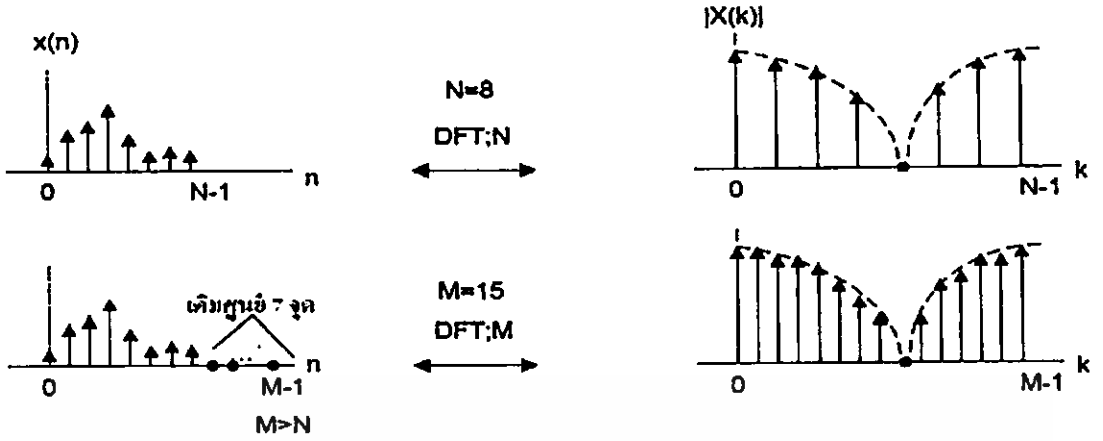
ซึ่งมีค่าเท่ากับ IDFT(X) = x(n) จริงตามสมการที่ 2.13

ในทำนองเดียวกันการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ซึ่งมีความหมายเดียวกับการแปลงฟูริเยร์ย้อนกลับแบบไม่ต่อเนื่อง สามารถหาได้จากการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว ดังสมการต่อไปนี้

$$x(n) = \text{IFFT}(X) = \frac{1}{N}(\text{FFT}(X^*))^* \tag{2.16}$$

## 2.5 การเติมศูนย์ (zero padding)

การเติมศูนย์ เป็นการเติมจุดที่มีค่าเป็นศูนย์ต่อท้ายเข้าไปในสัญญาณ x(n) ก่อนที่จะทำการแปลงฟูริเยร์แบบไม่ต่อเนื่อง ซึ่งการเติมศูนย์จะช่วยให้มองเห็นรูปร่างได้ละเอียด และชัดเจนขึ้น แต่ในทางทฤษฎีแล้วไม่ได้เป็นการเพิ่มข้อมูลใดๆ ให้กับสัญญาณเลย เส้นประที่แสดงในรูปที่ 2.5 คือเส้นที่แสดงผลลัพธ์ที่เกิดจากการแปลงฟูริเยร์แบบไม่ต่อเนื่อง ซึ่งไม่ว่าเราจะเติมศูนย์เข้าไปเท่าไรเส้นนี้ก็จะคงเดิม [1]

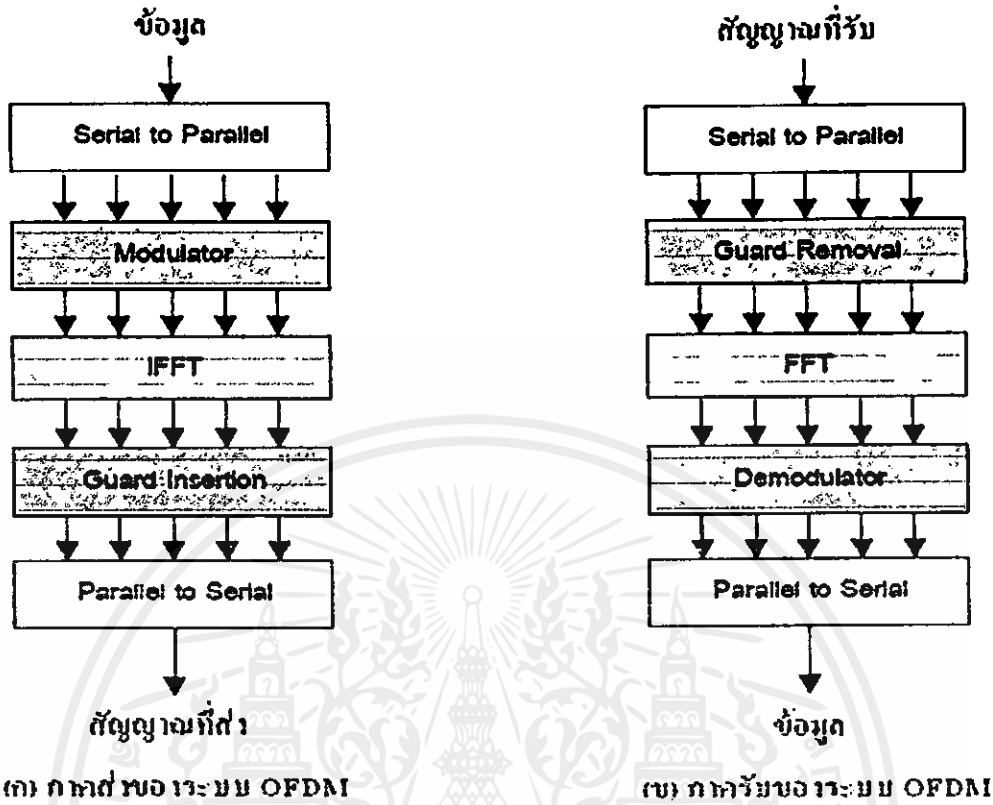



**รูปที่ 2.5** ผลของการเติมศูนย์กับการแปลงฟูริเยร์แบบไม่ต่อเนื่อง

## 2.6 การแปลงฟูริเยร์แบบรวดเร็วกับการนำมาประยุกต์ใช้ในการสื่อสาร

การสื่อสารด้วยความเร็วสูงได้มีการพัฒนาเทคนิคการส่งสัญญาณขึ้นมาหลายวิธี โดยเริ่มมีการคิดค้นและวิจัยกันมาตั้งแต่ช่วงปี ค.ศ. 1960-1970 แต่กว่าจะสามารถนำมาใช้งานจริงได้ ก็เป็นเวลาอีกประมาณ 30-40 ปีต่อมา เนื่องจากต้องอาศัยเทคโนโลยีต่างๆ ที่พัฒนาขึ้นมาเพื่อสร้างส่วนประกอบทางด้านฮาร์ดแวร์ที่สำคัญ เช่น มีการนำการแปลงฟูริเยร์แบบรวดเร็วไปประยุกต์ใช้งานที่ภาคส่งและภาครับ ของการสื่อสารระบบ OFDM

จากลักษณะการทำงานของระบบ OFDM ที่ภาคการส่งสัญญาณนั้น สัญญาณข้อมูลที่มีอัตราบิตสูงที่ต้องการส่งออกไปจะถูกแบ่งออกเป็นสัญญาณข้อมูลที่มีอัตราบิตต่ำลงหลายๆ อัน โดยอาศัยเทคนิคการแปลงข้อมูลแบบอนุกรมไปเป็นข้อมูลแบบขนาน (serial to parallel) แล้วจึงนำมามอดูเลต (modulate) เข้ากับคลื่นพาห์ของช่องสัญญาณต่างๆ ด้วยเหตุนี้เราจึงเรียก OFDM ว่าเป็นเทคนิคการมอดูเลตแบบหลายคลื่นพาห์ (Multi-Carrier Modulation Technique : MCM) ซึ่งสามารถป้องกันการเกิด สัญญาณเฟดเฉพาะความถี่ (frequency-selective fading) และการรบกวนแถบแคบ (narrow-band interference) ได้ จากนั้นจึงทำการแปลงสัญญาณในเชิงความถี่มาเป็นสัญญาณในเชิงเวลาโดยอาศัยการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว แล้วทำการแทรกช่วงเวลาป้องกัน (Guard Interval) เพื่อแก้ปัญหาการรบกวนของสัญญาณ อันเนื่องมาจากการหน่วงเวลาซึ่งเกิดจากการสะท้อนของคลื่นหลายเส้นทาง (Multipath delay) ของช่องสัญญาณ แล้วรวมสัญญาณย่อยๆ เข้าด้วยกันโดยอาศัยเทคนิคการแปลงข้อมูลแบบขนานไปเป็นข้อมูลแบบอนุกรม (parallel to serial) สำหรับที่ภาครับสัญญาณนั้นก็จะมีขั้นตอนการทำงานที่ตรงกันข้ามกับที่ภาคส่งสัญญาณ และในภาครับสัญญาณนั้นจะใช้การแปลงฟูริเยร์แบบรวดเร็วในการแปลงสัญญาณในเชิงเวลากลับไปเป็นสัญญาณในเชิงความถี่อีกครั้ง บล็อกไดอะแกรมของภาคส่ง และภาครับ แสดงในรูปที่ 2.6




(ก) ภาคส่งของระบบ OFDM (ข) ภาครับของระบบ OFDM

**รูปที่ 2.6** แสดงบล็อกไดอะแกรมของระบบ OFDM (ก) ภาคส่ง (ข) ภาครับ

จากคุณสมบัติและเทคนิคต่างๆ ที่ได้กล่าวมา ทำให้ปัจจุบันนี้ได้มีการนำเทคนิคการสื่อสารระบบ OFDM มาพัฒนาเพิ่มเติมจนกลายเป็น W-OFDM (Wide-band OFDM), V-OFDM (Vector OFDM) และ Flash OFDM เป็นต้น เพื่อนำมาประยุกต์ใช้งานในรูปแบบต่างๆ คือ ระบบโครงข่ายแบบไร้สายภายในประเทศ ระบบการส่งสัญญาณดิจิตอลแบบไม่เท่ากันในสายสัญญาณ ระบบการส่งสัญญาณดิจิตอลแบบรวดเร็วในสายสัญญาณ ระบบการกระจายสัญญาณเสียงแบบดิจิตอล และระบบการกระจายสัญญาณวีดีโอแบบดิจิตอล

การสื่อสารระบบ OFDM นำการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว และการแปลงฟูริเยร์แบบรวดเร็ว มาใช้ในการแปลงรูปแบบของสัญญาณที่ภาคส่ง และภาครับตามลำดับนั้น ด้วยเหตุผลที่ว่า การแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว และการแปลงฟูริเยร์แบบรวดเร็วสามารถนำมาประยุกต์ใช้ในกับการส่ง และรับข้อมูลแบบหลายช่องทาง (parallel channels) ได้ [11] ซึ่งเป็นลักษณะการส่ง และรับข้อมูลของระบบ OFDM ที่จะส่งข้อมูลไปในหลายๆ คลื่นพาห์ ที่เรียกกันว่า การส่งข้อมูลแบบขนาน (parallel transmission) รวมทั้งยังสามารถที่จะออกแบบให้ส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ให้มีการใช้พลังงานที่ต่ำ (minimize power consumption) [3,7,12] และสามารถคำนวณ

ได้รวดเร็ว (high throughput) [12] ดังนั้นจึงได้มีการพัฒนาส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ขึ้นมาหลายวิธีเพื่อให้สามารถนำมาประยุกต์ใช้งานกับระบบของ OFDM ได้ เช่น

การนำไปประยุกต์ใช้ในการสื่อสารระบบโครงข่ายแบบไร้สาย (WLAN) [13] ซึ่งออกแบบมาจากการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 เพื่อรองรับการคำนวณข้อมูลจำนวน 16 จุดแบบจำนวนเชิงซ้อน โดยมีความกว้างของข้อมูล (word length) ขนาด 9 บิต และผลที่ได้จากการออกแบบสามารถทำงานได้ที่ความถี่นาฬิกาสูงสุดคือ 50 MHz และใช้ทรานซิสเตอร์ในการทำงานทั้งหมด 110,000 ทรานซิสเตอร์

การนำไปประยุกต์ใช้ในการสื่อสารระบบการส่งสัญญาณดิจิตอลแบบรวดเร็วในสายสัญญาณ (ADSL) [14] ซึ่งเป็นการออกแบบจากการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 4 (radix-4) โดยความกว้างของข้อมูลมีขนาด 18 บิต และรับข้อมูลเข้ามาคำนวณได้ 2,048 จุด โดยสามารถคำนวณข้อมูลทั้งหมดให้เสร็จได้ภายในเวลา 80 μs ที่ความถี่นาฬิกาสูงสุดคือ 100 MHz ซึ่งใช้เกทในการทำงานทั้งหมดประมาณ 150,000 เกทโดยไม่รวมหน่วยเก็บความจำ

การนำไปประยุกต์ใช้ในการสื่อสารระบบการกระจายสัญญาณวีดีโอแบบดิจิตอล (DVB) [15] ซึ่งเป็นการออกแบบจากการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 4 ในการทำงาน 6 สถานะแรกผสมกับวิธีระบบฐาน 2 ในการทำงานที่สถานะสุดท้าย เพื่อรองรับข้อมูลเข้ามาคำนวณได้ 8,192 จุดแบบจำนวนเชิงซ้อน โดยใช้เกทในการทำงานทั้งหมด 700,000 เกท

การออกแบบเพื่อนำไปประยุกต์ใช้ในการสื่อสารของระบบ OFDM [16] ซึ่งเป็นการออกแบบจากการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 4 โดยมีความกว้างของข้อมูลจำนวน 20 บิต และรับข้อมูลเข้ามาคำนวณได้ 256 จุดแบบจำนวนเชิงซ้อน โดยสามารถคำนวณข้อมูลทั้งหมดให้เสร็จได้ภายในเวลา 6 μs ที่ความถี่นาฬิกาสูงสุดคือ 42 MHz ซึ่งใช้เกทในการทำงานทั้งหมดประมาณ 98,326 เกทโดยไม่รวมหน่วยเก็บความจำ

การออกแบบโดยใช้การแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 4 ในการทำงาน 2 สถานะแรกผสมกับวิธีระบบฐาน 2 ในการทำงานสถานะสุดท้าย [17] โดยให้สามารถรับข้อมูลเข้ามาคำนวณได้ 256 จุดแบบจำนวนเชิงซ้อน โดยมีความกว้างของข้อมูลขนาด 16 บิต ผลที่ได้คือสามารถคำนวณข้อมูลทั้งหมดให้เสร็จได้ภายในเวลา 6.6 μs ที่ความถี่นาฬิกาสูงสุดคือ 97 MHz และใช้เกทในการทำงานทั้งหมดประมาณ 37,000 เกทโดยไม่รวมหน่วยเก็บความจำ

จากบทความต่างๆ แสดงให้เห็นว่าส่วนประมวลผลของแปลงฟูริเยร์แบบรวดเร็ว ที่นำมาประยุกต์ใช้ภายในภาคส่ง และภาครับข้อมูลของการสื่อสารนั้น มีวิธีการคำนวณที่แตกต่างกันออกไปหลายวิธี และทุกวิธีที่มีการนำมาใช้งานต่างก็ได้รับการพัฒนาให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วนั้นมีขนาดที่เล็กลง

## 2.7 สรุป

เนื้อหาในบทนี้ได้แสดงถึง ที่มาของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง และการพัฒนาให้สามารถทำงานได้เร็วขึ้น และมีประสิทธิภาพมากขึ้น นั่นคือการแปลงฟูริเยร์แบบรวดเร็ว โดยอธิบายวิธีการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว และการคำนวณการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา รวมทั้งการนำไปประยุกต์ใช้ที่ภาคส่ง และภาครับข้อมูลของการสื่อสาร ซึ่งการแปลงฟูริเยร์แบบรวดเร็วที่นำมาใช้ที่ภาคส่ง และภาครับข้อมูลของการสื่อสารนั้น มีอยู่ด้วยกันหลายวิธี และทุกวิธีต่างก็มีการพัฒนาให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วมีขนาดที่เล็กลง โดยจากการศึกษาพบว่าวิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา สามารถนำมาปรับปรุงให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วมีขนาดเล็กได้ง่ายกว่าวิธีการคำนวณแบบอื่น

# บทที่ 3
# การออกแบบวงจรคำนวณทางคณิตศาสตร์

## 3.1 บทนำ

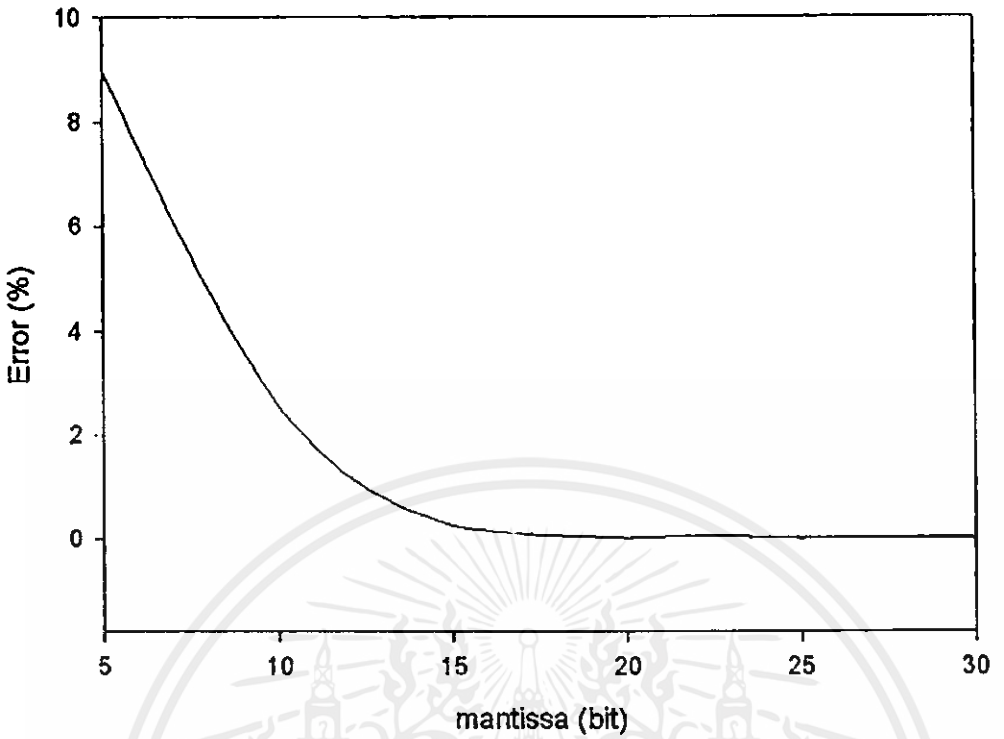
โดยทั่วไปการคำนวณทางคณิตศาสตร์โดยใช้คอมพิวเตอร์เป็นการคำนวณบนเลขจำนวนเต็มฐาน 2 (binary integer) แต่เมื่อนำมาใช้งานก็จะพบข้อจำกัดคือ ไม่สามารถใช้แทนค่าที่มีขนาดใหญ่มากๆ หรือน้อยมากๆ ได้ซึ่งถูกจำกัดโดยขนาดความยาวของคำ (word size) ตัวอย่างเช่น กำหนดค่าตัวเลขทั้งหมดเป็น 32 บิต จะได้ค่าแบบดิดเดรื่องหมายมีค่าสูงสุดเท่ากับ $2^{31}$-1 เท่านั้น และขอบเขตของจำนวนจะมีขนาดลดลงถ้าใช้จำนวนบิตที่เท่ากันไปใช้ในการแทนค่าเศษส่วน ซึ่งเป็นปัญหาที่พบในการเขียนโปรแกรมที่ใช้ในการคำนวณทางคณิตศาสตร์ จึงได้มีการคิดหาวิธีการเพื่อแก้ปัญหาข้อจำกัดในส่วนนี้

ในปี ค.ศ. 1985 the Institute of Electrical and Electronic Engineers (IEEE) ได้ประกาศมาตรฐาน (IEEE Floating-Point Standard 754) [18] สำหรับกำหนดค่าตัวเลขในการใช้งานคือตัวเลขอิงดรรชนี (floating-point number) เพื่อนำมาใช้แก้ไขข้อจำกัดดังกล่าว

ในบทนี้ได้นำเสนอเกี่ยวกับการออกแบบวงจรคำนวณทางคณิตศาสตร์ด้วยภาษา VHDL โดยใช้มาตรฐานการกำหนดค่าของระบบเลขอิงดรรชนี โดยหัวข้อที่ 3.2 เป็นการอธิบายรูปแบบและการกำหนดค่าของระบบเลขอิงดรรชนีตามมาตรฐาน จากนั้นจะเข้าไปสู่การออกแบบวงจรคำนวณทางคณิตศาสตร์ที่ประกอบด้วยหัวข้อต่อไปนี้ หัวข้อที่ 3.3 นำเสนอการออกแบบวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี หัวข้อที่ 3.4 นำเสนอการออกแบบวงจรคูณของระบบเลขอิงดรรชนี หัวข้อที่ 3.5 นำเสนอการออกแบบวงจรหารของระบบเลขอิงดรรชนี หัวข้อที่ 3.6 อธิบายการเกิดโอเวอร์โฟลว์ (overflow) และอันเดอร์โฟลว์ (underflow) หัวข้อที่ 3.7 เป็นการสรุป

## 3.2 ระบบเลขอิงดรรชนีตามมาตรฐาน 754 (IEEE Floating-Point Standard 754)

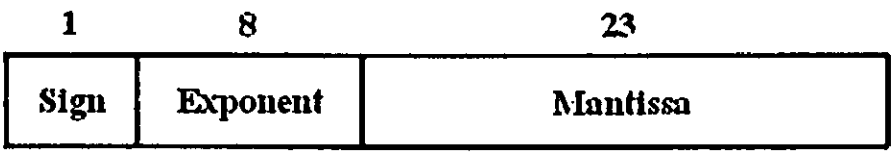
IEEE ได้มีการพัฒนาระบบเลขอิงดรรชนีขึ้นมา 2 แบบคือ แบบ 32 บิต ที่เป็นแบบพื้นฐานและแบบ 64 บิต [19] ซึ่งเป็นแบบที่มีการขยายค่าของข้อมูลให้สามารถรับข้อมูลที่มีขนาดกว้างมากขึ้นได้ ทำให้สามารถเลือกใช้ตามความต้องการที่จะออกแบบให้สามารถรับค่าของข้อมูลได้กว้างขนาดไหน ในการออกแบบครั้งนี้เลือกใช้รูปแบบพื้นฐาน 32 บิต เนื่องจากได้ทำการทดสอบการแปลงค่าของเลขอิงดรรชนีที่ได้จากการคำนวณกลับมาเป็นเลขฐานสิบ โดยใช้จำนวนบิตของค่าจุดทศนิยม (mantissa) ที่ต่างกันเพื่อเปรียบเทียบความผิดพลาดที่เกิดขึ้น แสดงดังกราฟรูปที่ 3.1




**รูปที่ 3.1** แสดงความผิดพลาดที่เกิดขึ้นจากการใช้จำนวนบิตของจุดทศนิยมที่ต่างกัน

จากกราฟรูปที่ 3.1 แสดงให้เห็นว่าความผิดพลาดในการแปลงค่าจากเลขอิงดรรชนีไปเป็นเลขฐานสิบจะลดลงเมื่อใช้จำนวนบิตของจุดทศนิยมที่มากขึ้น และความผิดพลาดจะลดต่ำลงจนเกือบเป็นศูนย์เมื่อใช้จำนวนบิตของจุดทศนิยมมากกว่า 20 บิตขึ้นไป

ซึ่งจากรูปแบบของเลขอิงดรรชนีขนาด 32 บิต ที่นำมาใช้งานจะมีการแบ่งการเก็บข้อมูลออกเป็น 3 ส่วนคือ บิตที่ 0 ไปถึงบิตที่ 22 สำหรับค่า Mantissa บิตที่ 23 ไปถึงบิตที่ 30 สำหรับค่า Exponent และ บิตที่ 31 สำหรับค่า Sign แสดงในรูปที่ 3.2 ซึ่งค่าของทั้ง 3 ส่วนนี้จะมาจากการกำหนดค่าของระบบเลขอิงดรรชนี




**รูปที่ 3.2** แสดงการแบ่งส่วนของระบบเลขอิงดรรชนีแบบ 32 บิต

การปรับค่าจากเลขฐาน 10 มาเป็นระบบเลขอิงดรรชนีแบบ 32 บิต สามารถปรับค่าได้ตามสมการต่อไปนี้

$$\text{Value} = (-1)^{S} \times 2^{E-127} \times 1.M \qquad (3.1)$$

ตัวแปร S หมายถึงค่า Sign มีขนาด 1 บิต โดยที่ค่า Sign จะมีค่าเป็น 0 ก็ต่อเมื่อค่าของเลขฐาน 10 มีค่าเป็นบวก และจะมีค่าเป็น 1 ก็ต่อเมื่อค่าของเลขฐาน 10 มีค่าเป็นลบ

ตัวแปร E หมายถึงค่า Exponent มีขนาด 8 บิต ทำให้สามารถนับค่าได้ตั้งแต่ 0 ถึง 255 แต่จะถูกชดเชย ด้วยค่า 127 ทำให้สามารถนับค่าได้ตั้งแต่ $2^{-127}$ ถึง $2^{128}$ เพื่อเป็นการแบ่งค่าที่น้อยกว่า 1 และ มากกว่า 1

ตัวแปร M หมายถึงค่า Mantissa มีขนาด 23 บิต ซึ่งเป็นค่าที่ได้มาจากการปรับค่าของเลขฐาน 10 (ที่เหลือจากการทำให้อยู่ในรูปของ 2 ยกกำลัง) ทำให้อยู่ในรูปของหนึ่งส่วนสองยกกำลัง ในกรณีที่จะปรับค่าของเลขอิงดรรชนีจะทำให้กลับไปเป็นเลขฐาน 10 หรือการนำเลขอิงดรรชนีไปคำนวณทางคณิตศาสตร์นั้น จะต้องใส่ค่า 1 หน้าค่า Mantissa เสียก่อน เนื่องจากบิตหน้าสุดของค่า Mantissa ที่ใช้เวลาปรับค่าหรือเวลาคำนวณต้องเป็น 1 เสมอ เรียกว่า การนอร์มัลไลซ์ (Normalize) และบิตนี้ไม่จำเป็นต้องแสดง ดังนั้นจากตัวเลขจำนวน 23 บิต จึงมีความละเอียดเป็น 24 บิต

ค่า 0 จะเป็นตัวเลขพิเศษคือ ค่า Exponent และค่า Mantissa จะมีค่าเป็น 0 ทั้งหมด โดยที่ค่า Sign สามารถมีค่าได้ทั้ง 0 และ 1

**ตัวอย่างการปรับค่าจากเลขฐาน 10 ไปเป็นระบบเลขอิงดรรชนีขนาด 32 บิต**

ค่า 3 ในเลขฐาน 10 สามารถเขียนให้อยู่ในเลข 2 ยกกำลังได้เป็น $1.5 \times 2^{1}$

ค่า 1.5 จะปรับให้อยู่ในรูปของหนึ่งส่วนสองยกกำลังแบบ 1 + 1/2 + 1/4 + 1/8 ได้เป็น

$$1.5 = 1 + 1/2 + 0/4 + 0/8 = 1.100$$

เมื่อนำค่าที่ได้ทั้งหมดกลับไปแทนค่าในสมการที่ 3.1 จะได้เป็น

$$3 = (-1)^{0} \times 2^{128} \times 1.100$$

ค่า Sign มีค่าเป็น 0 เนื่องจากเลขฐาน 10 มีค่าเป็นบวก

ค่า Exponent มีค่าเท่ากับ 127+1 = 128 = 10000000

ค่า Mantissa มีค่าเท่ากับค่าหลังจุดทศนิยมคือ 10000000000000000000000

ทำให้ได้ค่าในระบบเลขอิงดรรชนีขนาด 32 บิตเป็น

$$3 = 0\ 10000000\ 10000000000000000000000$$

**ตัวอย่างการปรับค่าจากระบบเลขอิงดรรชนีขนาด 32 บิตไปเป็นเลขฐาน 10**

ค่าในระบบเลขอิงดรรชนี 32 บิตคือ 0 10000010 11000000000000000000000

ค่า Sign มีค่าเป็น 0 แสดงว่าเลขฐาน 10 มีค่าเป็นบวก

ค่ายกกำลังของ 2 มีค่าเท่ากับ 130-127 = 3

ค่า 1.M มีค่าเท่ากับ 1.1100 เมื่อเปลี่ยนค่ากลับมาได้เป็น 1.75

เมื่อนำมารวมกันตามสมการที่ 3.1 ได้เป็น

$$(-1)^0 \times 2^3 \times 1.75 = 14$$

ดังนั้นค่าที่ได้ในเลขฐาน 10 จะมีค่าเท่ากับ 14

## 3.3 การออกแบบวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี

การออกแบบวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี ขบวนการทำงานจะเป็นไปตามขั้นตอนดังต่อไปนี้

- ตรวจสอบข้อมูลทั้ง 2 ค่าที่เข้ามาว่าค่าใดมีค่าน้อยกว่า โดยตรวจสอบจากค่า exponent
- เลื่อนค่า mantissa ของค่าน้อยไปทางขวาตามค่าความต่างของค่า exponent
- กำหนดค่า exponent ของผลลัพธ์ให้มีค่าเท่ากับค่า exponent ของค่ามาก
- บวกหรือลบค่า mantissa ค่ามากกับค่า mantissa ค่าน้อยที่ผ่านการเลื่อนค่าแล้ว การบวกหรือลบจะขึ้นอยู่กับค่า sign ของข้อมูลทั้ง 2 ที่เข้ามาคือ ถ้าค่า sign เหมือนกันนำค่า mantissa มากบวกกัน แต่ถ้าค่า sign ต่างกันจะนำค่า mantissa มาลบกัน
- สรุปค่า sign ให้กับผลลัพธ์ โดยกำหนดค่า sign ตามข้อมูลค่ามาก พร้อมกับตรวจสอบว่ามีการเกิดโอเวอร์โฟลว์ หรืออันเดอร์โฟลว์ หรือไม่

**ตัวอย่างการคำนวณ**

**แสดงการบวกของระบบเลขอิงดรรชนี พิจารณาในกรณีที่บิต n+1 มีค่าเป็น 0**

| | | | |
|---|---|---|---|
| ค่าที่ 1 | 10.5 | $1.0101 \times 2^3$ | sign 0 |
| ค่าที่ 2 | 2.25 | $1.0010 \times 2^1$ | sign 0 |

ค่า mantissa ของค่าที่ 1 เป็นการเก็บค่า 0101 และของค่าที่ 2 เป็นการเก็บค่า 0010 จากนั้นเป็นการเลื่อนบิตของค่า mantissa ค่าน้อยตามค่าความต่างของค่า exponent คือ 2 ได้เป็น

| | |
|---|---|
| 1.010100 + | ค่า sign เหมือนกันนำมาบวกกัน |
| 0.010010 | เลื่อนค่า 2 บิต |
| 1.100110 | |

ผลลัพธ์ที่ได้คือ $1.100110 \times 2^3$ ซึ่งจะมีค่าเท่ากับ 12.75 ในกรณีนี้ค่า mantissa ของผลลัพธ์คือ 100110 ค่า exponent ของผลลัพธ์คือ 127+3 มีค่าเท่ากับ 130 และค่า sign มีค่าเท่ากับ 0

**แสดงการบวกของระบบเลขอิงดรรชนี พิจารณาในกรณีที่บิต n+1 มีค่าเป็น 1 มีการเกิดโอเวอร์โฟลว์**

| | | | |
|---|---|---|---|
| ค่าที่ 1 | 5.5 | $1.0110 \times 2^2$ | sign 0 |
| ค่าที่ 2 | 14.5 | $1.1101 \times 2^3$ | sign 0 |

ค่า mantissa ของค่าที่ 1 เป็นการเก็บค่า 0110 และของค่าที่ 2 เป็นการเก็บค่า 1101 จากนั้นเป็นการเลื่อนบิตของค่า mantissa ค่าน้อยตามค่าความต่างของค่า exponent คือ 1 ได้เป็น

| | |
|---|---|
| 1.1101+ | ค่า sign เหมือนกันนำมาบวกกัน |
| 0.1011 | เลื่อนค่า 1 บิต |
| 10.1000 | |

ผลลัพธ์ที่ได้คือ $10.1000 \times 2^3$ ซึ่งจะมีค่าเท่ากับ 20 กรณีนี้เกิดโอเวอร์โฟลว์ ทำให้ต้องปรับค่าใหม่เป็น $1.01000 \times 2^4$ กรณีนี้ค่า mantissa ของผลลัพธ์คือ 01000 ค่า exponent ของผลลัพธ์คือ 127+4 มีค่าเท่ากับ 131 และค่า sign มีค่าเท่ากับ 0

**แสดงการลบของระบบเลขอิงดรรชนี พิจารณาในกรณีที่บิต n+1 และ n มีค่าเป็น 0 มีการเกิดอันเดอร์โฟล์ว์**

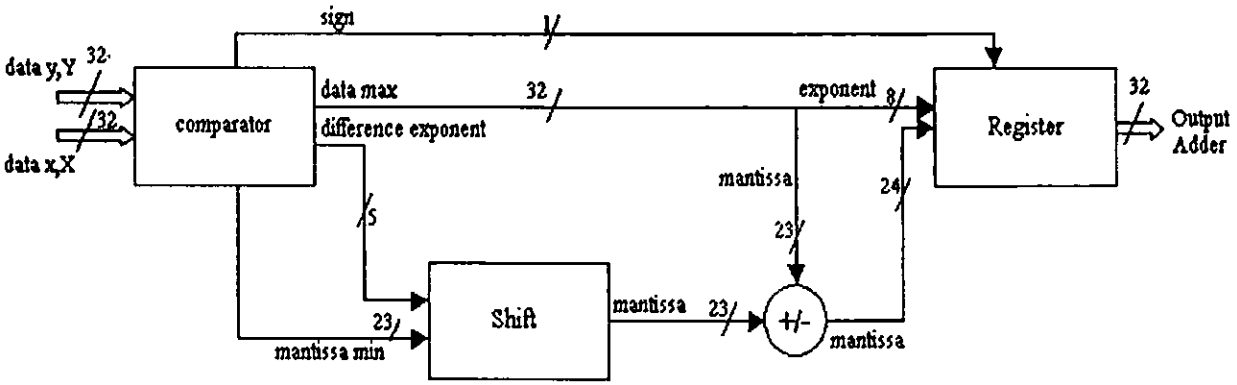
| | | | |
|---|---|---|---|
| ค่าที่ 1 | 7.5 | $1.1110 \times 2^2$ | sign 0 |
| ค่าที่ 2 | -6.5 | $1.1010 \times 2^2$ | sign 1 |

ค่า mantissa ของค่าที่ 1 เป็นการเก็บค่า 1110 และของค่าที่ 2 เป็นการเก็บค่า 1010 จากนั้นเป็นการเลื่อนบิตค่า mantissa แต่กรณีนี้ค่า exponent มีขนาดเท่ากันไม่จำเป็นต้องเลื่อนค่า mantissa

| | |
|---|---|
| 1.1110- | ค่า sign ต่างกันนำมาลบกัน |
| 1.1010 | |
| 0.0100 | |

ผลลัพธ์ที่ได้คือ $0.0100 \times 2^2$ ซึ่งจะมีค่าเท่ากับ 1 กรณีนี้เกิดอันเดอร์โฟลว์ ทำให้ต้องปรับค่าใหม่เป็น $1.0000 \times 2^0$ กรณีนี้ค่า mantissa ของผลลัพธ์คือ 00000 ค่า exponent ของผลลัพธ์คือ 127+0 มีค่าเท่ากับ 127 และค่า sign มีค่าเท่ากับ 0

จากขั้นตอนการทำงานทั้งหมดสามารถนำมาออกแบบการทำงานของวงจรบวกและวงจรลบของระบบเลขอิงดรรชนีได้ดัง แสดงในรูปที่ 3.3




**รูปที่ 3.3** บล็อกไดอะแกรมของวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี

จากรูปที่ 3.3 หลักการทำงานคือ เมื่อรับข้อมูลทั้ง 2 ค่าเข้ามาวงจรเปรียบเทียบข้อมูล (comparator) จะทำการเปรียบเทียบข้อมูลที่เข้ามาว่าข้อมูลใดมีค่าน้อยกว่า จากนั้นจะส่งค่า mantissa ของค่าน้อยไปเลื่อนบิตที่วงจรเลื่อนบิต (Shift) ตามจำนวนความต่างของ exponent เมื่อเลื่อนบิตค่า mantissa ของค่าน้อยเสร็จก็จะส่งไปที่วงจรบวก/ลบ (Add/Sub) เพื่อทำการบวกหรือลบกับ mantissa ของค่ามาก การที่จะบวกหรือลบจะตรวจสอบจากค่า sign ของข้อมูลทั้ง 2 ที่เข้ามา เมื่อบวกหรือลบเสร็จแล้วค่า mantissa จะถูกส่งไปพักที่วงจรสรุปผลการคำนวณ (Register) รวมทั้งค่า sign และค่า exponent ของผลลัพธ์ ซึ่งจะมีค่าเป็นไปตามข้อมูลค่าที่มากกว่า และตรวจสอบค่า mantissa ว่ามีการเกิดโอเวอร์โฟลว์หรืออันเดอร์โฟลว์ หรือไม่จากนั้นจะส่งค่าผลลัพธ์ออกไป สามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี คือ

### 3.3.1 วงจรเปรียบเทียบข้อมูล (Comparator)

เป็นวงจรที่คอยตรวจสอบข้อมูลที่เข้ามาว่า ข้อมูลใดมีค่ามากกว่าโดยตรวจสอบจากค่า exponent ของข้อมูล พร้อมทั้งส่งค่าความต่างของค่า exponent รวมทั้ง mantissa ของข้อมูลค่าน้อยไปให้วงจรเลื่อนบิต จากนั้นจะส่งค่า exponent และค่า sign ของข้อมูลค่ามากไปให้วงจรสรุปผลการคำนวณ เพื่อให้เป็นค่า exponent และค่า sign ของผลลัพธ์ โปรแกรมแสดงในภาคผนวก ง.1

### 3.3.2 วงจรเลื่อนบิต (Shift)

เป็นวงจรที่ทำหน้าที่เลื่อนบิตค่า mantissa ของข้อมูลค่าน้อยไปทางขวา ตามจำนวนค่าความต่างของค่า exponent พร้อมทั้งส่งค่า mantissa ของข้อมูลค่าน้อยที่เลื่อนบิตแล้วไปให้วงจรบวก/ลบ โปรแกรมแสดงในภาคผนวก ง.3

**3.3.3 วงจรบวก/ลบ (Add/Sub)**

เป็นวงจรที่ทำหน้าที่บวกหรือลบค่า mantissa ของข้อมูลที่เข้ามา วงจรจะทำการบวกเมื่อค่า sign ของข้อมูลที่เข้ามามีค่าเหมือนกัน แต่วงจรจะทำการลบเมื่อค่า sign ของข้อมูลที่เข้ามามีค่าต่างกัน เมื่อบวกหรือลบข้อมูลเสร็จก็จะส่งผลที่ได้ไปให้กับวงจรสรุปผลการคำนวณ โปรแกรมแสดงในภาคผนวก ง.2

**3.3.4 วงจรสรุปผลการคำนวณ (Register)**

เป็นวงจรที่รับข้อมูลทั้งหมดมาพักไว้ก่อนที่จะส่งผลลัพธ์ออกไป เพื่อตรวจสอบว่าผลที่ได้จากวงจรบวก/ลบ มีการเกิดโอเวอร์โฟลว์หรืออันเดอร์โฟลว์หรือไม่ โปรแกรมแสดงในภาคผนวก ง.4

## 3.4 การออกแบบวงจรคูณของระบบเลขอิงดรรชนี

การออกแบบวงจรคูณของระบบเลขอิงดรรชนี ขบวนการทำงานจะเป็นไปตามขั้นตอนดังต่อไปนี้

- ตรวจสอบข้อมูลทั้ง 2 ค่าที่เข้ามาว่ามีค่าใดมีค่าเท่ากับ 0 หรือไม่ ถ้ามีจะส่งผลลัพธ์ที่มีค่าเท่ากับ 0 ออกไป
- ถ้าพบว่าไม่มีค่าใดเท่ากับ 0 แล้ว จะนำค่า mantissa ของข้อมูลทั้ง 2 มาคูณกัน ส่วนค่า exponent จะนำมาบวกกัน
- นำค่า sign ของทั้ง 2 ข้อมูลมาเปรียบเทียบกันเพื่อหา sign ของผลลัพธ์ โดยถ้าค่า sign ของทั้ง 2 ข้อมูลที่เข้ามามีค่าเหมือนกัน sign ของผลลัพธ์จะมีค่าเป็น 0 แต่ถ้าไม่เหมือนกัน sign ของผลลัพธ์จะมีค่าเป็น 1
- ตรวจสอบว่ามีการเกิดโอเวอร์โฟลว์ หรืออันเดอร์โฟลว์ หรือไม่

**ตัวอย่างการคำนวณ**

**แสดงการคูณของระบบเลขอิงดรรชนี**

| | | | |
|---|---|---|---|
| ค่าที่ 1 | 5.5 | $1.0110 \times 2^{2}$ | sign = 0 |
| ค่าที่ 2 | 2.25 | $1.0010 \times 2^{1}$ | sign = 0 |

นำค่า mantissa ของทั้ง 2 ค่ามาคูณกัน และนำค่า exponent ของทั้ง 2 ค่ามาบวกกัน

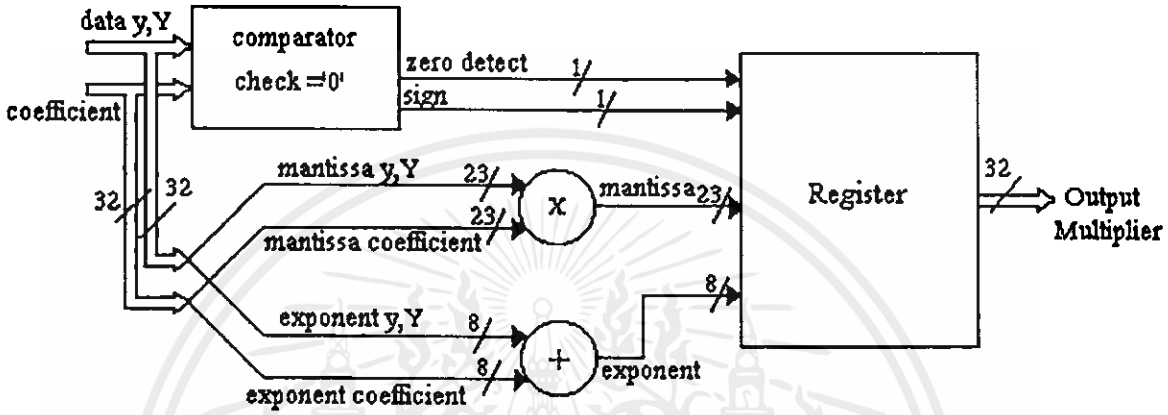
การคำนวณค่า mantissa

$$\begin{array}{r} 1.011 \times \\ \underline{1.001} \\ 1.100011 \end{array}$$

การคำนวณค่า exponent 2 + 1 = 3

ผลลัพธ์ที่ได้คือ 1.100011 x $2^3$ ซึ่งจะมีค่าเท่ากับ 12.375 ในกรณีนี้ค่า mantissa ของผลลัพธ์คือ 100011 ค่า exponent ของผลลัพธ์คือ 127+3 มีค่าเท่ากับ 130 และค่า sign มีค่าเท่ากับ 0

จากขั้นตอนการทำงานทั้งหมดสามารถนำมาออกแบบการทำงานของวงจรคูณของระบบเลขอิงดรรชนี แสดงในรูปที่ 3.4




รูปที่ 3.4 บล็อกไดอะแกรมของวงจรคูณของระบบเลขอิงดรรชนี

จากรูปที่ 3.4 หลักการทำงานคือ เมื่อรับข้อมูลทั้ง 2 ค่าเข้ามาวงจรเปรียบเทียบข้อมูล (comparator) จะตรวจสอบว่าข้อมูลทั้ง 2 ค่านั้นมีค่าใดเป็น 0 หรือไม่ ถ้าพบว่ามีค่าใดค่าหนึ่งเป็น 0 ก็จะส่งสัญญาณ zero detect ไปยังวงจรสรุปผลการคำนวณ (Register) ให้ส่งผลลัพธ์ค่า 0 ออกไปทันที แต่ถ้าไม่พบว่ามีค่าใดเป็น 0 ค่า mantissa ของทั้ง 2 ข้อมูลจะส่งไปคูณกันที่วงจรคูณค่า mantissa (x) ส่วนค่า exponent ของทั้ง 2 ข้อมูลจะส่งไปบวกกันที่วงจรบวกค่า exponent (+) จากนั้นส่งผลลัพธ์ที่ได้ไปที่วงจรสรุปผลการคำนวณ (Register) เพื่อสรุปค่า sign ของผลลัพธ์ และตรวจสอบว่าค่า mantissa มีการเกิดโอเวอร์โฟลว์ หรืออันเดอร์โฟลว์ หรือไม่ จากนั้นส่งค่าผลลัพธ์ออกไป สามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในวงจรคูณของระบบเลขอิงดรรชนี คือ

### 3.4.1 วงจรเปรียบเทียบข้อมูล (Comparator)

เป็นวงจรที่รับข้อมูลทั้งสองค่าเข้ามาแล้วตรวจสอบว่ามีข้อมูลใดมีค่าเท่ากับศูนย์หรือไม่ ถ้ามีข้อมูลค่าใดค่าหนึ่งมีค่าเท่ากับศูนย์แล้วจะส่งสัญญาณ zero detect ที่มีค่าเป็น 1 ไปให้วงจรสรุปผลการคำนวณ แต่ถ้าไม่มีค่าใดเป็นศูนย์แล้วค่า exponent ทั้งสองจะส่งไปที่วงจรบวกค่า exponent และค่า mantissa ทั้งสองจะส่งไปที่วงจรคูณค่า mantissa พร้อมทั้งส่งค่า sign ของผลลัพธ์ไปให้วงจรสรุปผลการคำนวณ โปรแกรมแสดงในภาคผนวก ง.7

**3.4.2 วงจรคูณค่า mantissa (x)**

เป็นวงจรที่รับค่า mantissa ขนาด 23 บิตของทั้งสองข้อมูลเข้ามาคูณกัน โดยผลที่ได้จากการคูณค่า mantissa ขนาด 23 บิต สองจำนวนเข้าด้วยกันทำให้ได้ผลออกมาเป็นค่า mantissa ขนาด 46 บิต ทำให้วงจรต้องตัดค่า mantissa ส่วนที่เกินจาก 23 บิต ออกไปก่อนที่จะส่งผลไปให้วงจรสรุปผลการคำนวณ โปรแกรมแสดงในภาคผนวก ง.5

**3.4.3 วงจรบวกค่า exponent (+)**

เป็นวงจรที่รับค่า exponent ของข้อมูลทั้งสองเข้ามาบวกกัน ผลที่ได้จากการบวกจะส่งไปให้กับวงจรสรุปผลการคำนวณ โปรแกรมแสดงในภาคผนวก ง.5

**3.4.4 วงจรสรุปผลการคำนวณ (Register)**

เป็นวงจรที่รับข้อมูลทั้งหมดมาพักไว้ก่อนที่จะส่งผลลัพธ์ออกไป และวงจรจะคอยตรวจสอบว่าสัญญาณ zero detect มีค่าเป็น 1 หรือไม่ ถ้าสัญญาณ zero detect มีค่าเป็น 1 ก็จะส่งผลลัพธ์ที่มีค่าเป็นศูนย์ออกไป โปรแกรมแสดงในภาคผนวก ง.6

## 3.5 การออกแบบวงจรหารของระบบเลขอิงดรรชนี

การออกแบบวงจรหารของระบบเลขอิงดรรชนีครั้งนี้ วงจรหารที่ต้องการออกแบบจะเป็นการหารด้วยค่าคงที่เพียงค่าเดียว ทำให้สามารถลดขั้นตอนการทำงานลงได้ ซึ่งจะเหลือขั้นตอนการทำงาน ดังต่อไปนี้

- ตรวจสอบข้อมูลที่เข้ามาว่ามีค่าเท่ากับ 0 หรือไม่ ถ้ามีจะส่งผลลัพธ์ที่มีค่าเท่ากับ 0 ออกไป
- ถ้าพบว่าข้อมูลมีค่าไม่เท่ากับ 0 แล้วจะนำค่า exponent ของข้อมูลที่เข้ามาลบด้วยค่า 8 เนื่องการหารในครั้งนี้เราต้องการหารด้วยค่า 256
- ค่า sign และค่า mantissa ของผลลัพธ์จะยังคงเป็นค่าเดิมเนื่องจากเป็นการหารด้วยค่าคงที่

**ตัวอย่างการคำนวณ**

**แสดงการหารของระบบเลขอิงดรรชนี ด้วยตัวหารที่เป็นค่าคงที่**

ค่าที่ 1 10 $1.0100 \times 2^3$

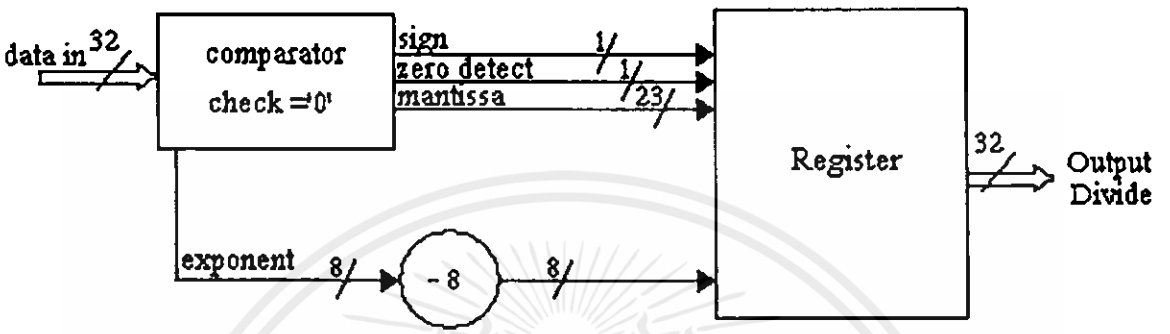
ค่าคงที่ 256 $1.0000 \times 2^8$

นำค่า exponent ของข้อมูลที่เข้ามาลบด้วยค่า 8

ค่า exponent $3 - 8 = -5$

ผลลัพธ์ที่ได้คือ $1.0100 \times 2^{-5}$ ซึ่งจะมีค่าเท่ากับ 0.0390625 ในกรณีนี้ค่า mantissa ของผลลัพธ์คือ 0100 ค่า exponent ของผลลัพธ์คือ 127-5 มีค่าเท่ากับ 122 และค่า sign มีค่าเท่ากับ 0

จากขั้นตอนการทำงานทั้งหมดสามารถนำมาออกแบบการทำงานของวงจรหารของระบบเลขอิงดรรชนี แสดงในรูปที่ 3.5




รูปที่ 3.5 บล็อกไดอะแกรมของวงจรหารของระบบเลขอิงดรรชนี

จากรูปที่ 3.5 หลักการทำงานคือ เมื่อรับข้อมูลเข้ามาวงจรเปรียบเทียบข้อมูล (comparator) จะตรวจสอบว่าข้อมูลนั้นมีค่าเป็น 0 หรือไม่ ถ้าพบว่าข้อมูลมีค่าเป็น 0 ก็จะส่งสัญญาณ zero detect ไปยังวงจรสรุปผลการคำนวณ (Register) ให้ส่งผลลัพธ์ค่า 0 ออกไปทันที แต่ถ้าไม่พบว่ามีค่าเป็น 0 ก็จะนำค่า exponent ไปที่วงจรลบ 8 (-8) เพื่อลบค่า exponent ด้วยค่าคงที่ 8 จากนั้นส่งไปที่วงจรสรุปผลการคำนวณ (Register) พร้อมกับค่า sign และค่า mantissa ที่จะเป็นค่าเดิมของข้อมูลที่เข้ามาและทำการส่งค่าผลลัพธ์ออกไป สามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในวงจรหารของระบบเลขอิงดรรชนี คือ

### 3.5.1 วงจรเปรียบเทียบข้อมูล (Comparator)

เป็นวงจรที่รับข้อมูลเข้ามาแล้วตรวจสอบว่าข้อมูลมีค่าเท่ากับศูนย์หรือไม่ ถ้ามีค่าเท่ากับศูนย์ ก็จะส่งสัญญาณ zero detect ที่มีค่าเป็น 1 ไปให้วงจรสรุปผลการคำนวณ แต่ถ้าไม่มีค่าเป็นศูนย์แล้วจะส่งค่า exponent ของข้อมูลที่เข้ามาไปที่วงจรลบ 8 พร้อมทั้งส่งค่า sign และค่า mantissa ของข้อมูลที่เข้ามาไปให้วงจรสรุปผลการคำนวณ โปรแกรมแสดงในภาคผนวก ง.9



### 3.5.2 วงจรลบ 8 (-8)

เป็นวงจรที่นำค่า exponent ของข้อมูลที่เข้ามาลบด้วยค่าคงที่ 8 ค่า exponent ที่ลบด้วย 8 แล้วจะส่งไปที่วงจรสรุปผลการคำนวณ โปรแกรมแสดงในภาคผนวก ง.8

**3.5.3 วงจรสรุปผลการคำนวณ (Register)**

เป็นวงจรที่รับข้อมูลทั้งหมดมาพักไว้ก่อนที่จะส่งผลลัพธ์ออกไป และวงจรจะคอยตรวจสอบว่าสัญญาณ zero detect มีค่าเป็น 1 หรือไม่ ถ้าสัญญาณ zero detect มีค่าเป็น 1 ก็จะส่งผลลัพธ์ที่มีค่าเป็นศูนย์ออกไป โปรแกรมแสดงในภาคผนวก ง.10

## 3.6 การเกิดโอเวอร์โฟลว์ (Overflow) และการเกิดอันเดอร์โฟลว์ (Underflow)

การเกิดโอเวอร์โฟลว์ คือ จำนวนฐาน 2 ที่มีความกว้าง n บิต 2 จำนวนเมื่อทำการบวกจะทำให้ผลลัพธ์เกิดเป็นความกว้างขนาด n+1 บิต ซึ่งในกรณีบิตที่ n+1 มีค่าเป็น 1 จะทำให้มีการเลื่อนบิตของผลลัพธ์ ให้เริ่มจากบิตที่ n+1 เป็นบิตแรก ดังนั้นเมื่อเลื่อนบิตในส่วนของค่า mantissa แล้วจะต้องเพิ่มค่าให้กับค่า exponent ตามจำนวนการเลื่อนจุดทศนิยม

การเกิดอันเดอร์โฟลว์ คือ เกิดขึ้นในกรณีที่มีการลบกันแล้วบิตที่ n+1 และบิตที่ n มีค่าเป็น 0 คือพบบิตที่มีค่าเป็น 1 ที่บิต n-1 เป็นบิตแรก จะต้องทำการเลื่อนบิตของผลลัพธ์ให้เริ่มจากบิตที่ n-1 ดังนั้นเมื่อเลื่อนบิตในส่วนของค่า mantissa แล้วจะต้องลดค่าให้กับค่า exponent ตามจำนวนการเลื่อนจุดทศนิยม

## 3.7 สรุป

เนื้อหาในบทนี้ได้อธิบายถึง รูปแบบที่ใช้ในการกำหนดค่าของระบบเลขอิงดรรชนีขนาด 32 บิต ตามมาตรฐานของ IEEE Standard 754 รวมทั้งแสดงการปรับค่าจากเลขฐาน 10 ไปเป็นระบบเลขอิงดรรชนีขนาด 32 บิต และการปรับค่ากลับ แสดงตัวอย่างวิธีการคำนวณทางคณิตศาสตร์ของระบบเลขอิงดรรชนี และการออกแบบวงจรคำนวณทางคณิตศาสตร์ในรูปแบบต่างๆ ได้แก่ วงจรบวก วงจรลบ วงจรคูณ และวงจรหาร เพื่อรองรับการคำนวณของระบบเลขอิงดรรชนีขนาด 32 บิต สำหรับการนำไปใช้ในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

# บทที่ 4

# การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

## 4.1 บทนำ

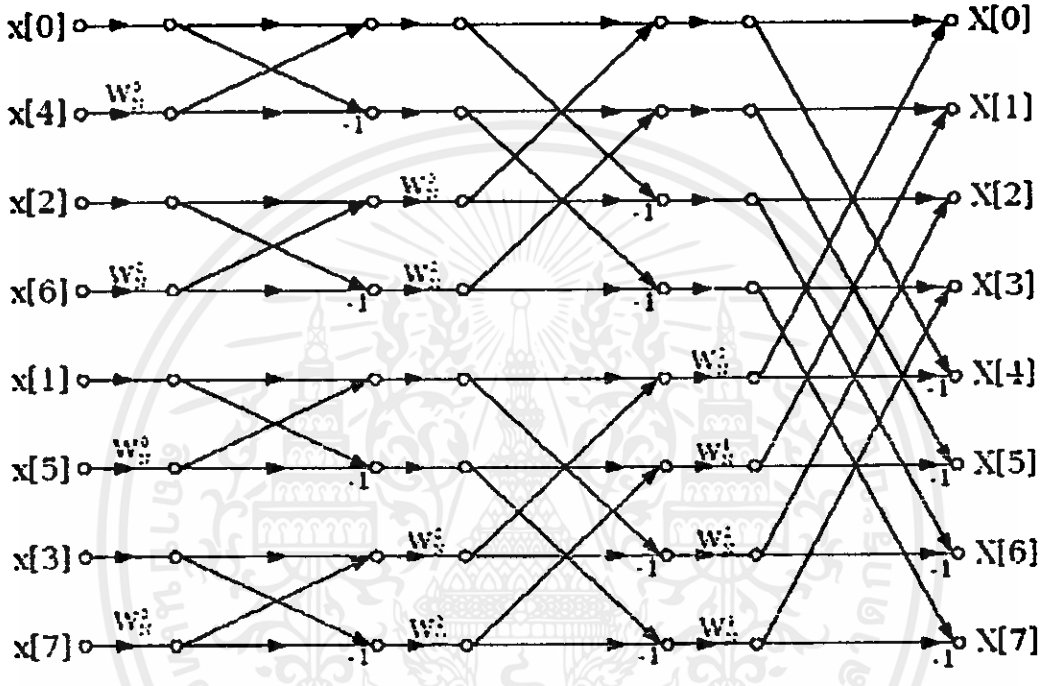
การแปลงฟูริเยร์แบบรวดเร็ว และการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วนิยมคำนวณโดยการใช้แผนภาพ ที่เรียกว่าแผนภาพผีเสื้อ (butterfly diagram) ดังนั้นจึงนำแผนภาพผีเสื้อมาใช้ในการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว จากการออกแบบพบว่าส่วนประมวลผลของแผนภาพผีเสื้อจะมีขนาดใหญ่ที่สุดภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว เนื่องจากว่าเป็นส่วนที่ใช้ในการคำนวณทางคณิตศาสตร์ และขนาดของส่วนประมวลผลของแผนภาพผีเสื้อจะมีขนาดใหญ่ขึ้น เมื่อมีการรับข้อมูลเข้ามาคำนวณมากขึ้น ดังนั้นเพื่อที่จะออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็กได้นั้น จึงจำเป็นที่จะต้องลดโครงสร้างการทำงานภายในส่วนประมวลผลของแผนภาพผีเสื้อลง และนำหน่วยความจำ (RAM) มาช่วยในการพักข้อมูลระหว่างการคำนวณ รวมทั้งการออกแบบการรับข้อมูลเท่าที่จำเป็น เนื่องจากจำนวนข้อมูลที่รับเข้ามาก็มีผลกับขนาด และเวลาที่ใช้ในการคำนวณ ซึ่งการออกแบบส่วนต่างๆ เหล่านี้จะเป็นไปตามข้อกำหนดของการนำไปใช้งาน

ซึ่งการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ในวิทยานิพนธ์เล่มนี้เป็นการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็กด้วยภาษา VHDL

โดยในบทนี้มีการนำเสนอเกี่ยวกับการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็ก โดยการลดโครงสร้างการทำงานภายในส่วนประมวลผลแผนภาพผีเสื้อให้มีขนาดที่เล็กลง หัวข้อที่ 4.2 เป็นการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว หัวข้อที่ 4.3 เป็นการออกแบบการทำงานของส่วนประมวลผลของแผนภาพผีเสื้อให้มีขนาดที่เล็กลง หัวข้อที่ 4.4 เป็นการออกแบบส่วนประกอบต่างๆ ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว หัวข้อที่ 4.5 เป็นการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว หัวข้อที่ 4.6 เป็นบทสรุป

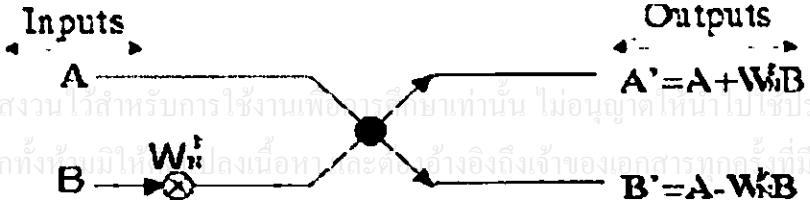
## 4.2 การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว (FFT processor)

การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว เป็นการออกแบบการทำงานมาจากแผนภาพผีเสื้อเช่นเดียวกับการคำนวณในรูปแบบปกติ โดยการออกแบบในบทนี้ จะเป็นการนำเอาแผนภาพผีเสื้อของการคำนวณวิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา มาใช้ในการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว แสดงในรูปที่ 4.1




**รูปที่ 4.1** แผนภาพรวมของการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว 8 จุด

จากรูปที่ 4.1 เป็นการแสดงการคำนวณข้อมูลจำนวน 8 จุด ซึ่งถ้ามีจำนวนข้อมูลมากกว่านี้ โครงสร้างการคำนวณของแผนภาพผีเสื้อก็จะมีการขยายขนาดมากขึ้นตามไปด้วย ทำให้เมื่อนำไปออกแบบการทำงานแล้วจะมีขนาดใหญ่ แต่จากการสังเกตพบว่าโครงสร้างดังกล่าวมีการทำงานที่ซ้ำกันอยู่ภายในของการคำนวณการแปลงฟูริเยร์แบบรวดเร็วนั่นก็คือ โครงสร้างการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด แสดงในรูปที่ 4.2






**รูปที่ 4.2** โครงสร้างการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด

จากรูปที่ 4.2 สามารถแสดงรายละเอียดการคำนวณได้ดังนี้

ข้อมูล A มีค่าเท่ากับ x + jX ที่ x เป็นค่าจริงและ jX เป็นค่าจินตภาพ

ข้อมูล B มีค่าเท่ากับ y + jY ที่ y เป็นค่าจริงและ jY เป็นค่าจินตภาพ

ค่า $W_N^k$ เป็นค่าตัวเลขสัมประสิทธิ์ (coefficient) มีค่าเท่ากับ $\cos(2\pi k/N) - j\sin(2\pi k/N)$

ผลลัพธ์ที่ได้จากการคำนวณ สามารถแสดงได้ดังนี้

$$A' = A + W_N^k B \tag{4.2}$$

$$B' = A - W_N^k B \tag{4.3}$$

จากค่า A, B และ $W_N^k$ สามารถแทนค่าในสมการที่ 4.2 และ 4.3 ได้เป็น

$$A' = [(x + y\cos(2\pi k/N) + Y\sin(2\pi k/N)) + j(X + Y\cos(2\pi k/N) - y\sin(2\pi k/N))] \tag{4.4}$$

$$B' = [(x - y\cos(2\pi k/N) - Y\sin(2\pi k/N)) + j(X - Y\cos(2\pi k/N) + y\sin(2\pi k/N))] \tag{4.5}$$

จากโครงสร้างการคำนวณของแผนภาพผีเสื้อที่มีการนำเอาโครงสร้างการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุดมาทำงานซ้ำกัน ทำให้สามารถออกแบบการทำงานภายในส่วนประมวลผลของแผนภาพผีเสื้อได้จากการใช้โครงสร้างการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุดเพียงชุดเดียว โดยนำหน่วยความจำมาช่วยในการพักข้อมูลระหว่างการคำนวณ เพื่อให้สามารถคำนวณได้อย่างสมบูรณ์ จากการทำงานในลักษณะนี้ทำให้สามารถออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วให้ที่มีขนาดเล็กได้ โดยการออกแบบในครั้งนี้เป็นการออกแบบเพื่อรองรับการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน

สำหรับการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว เพื่อรองรับการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน ในการออกแบบสามารถที่จะคำนวณจำนวนสถานะ (stage) ที่ต้องใช้ในการคำนวณได้จากสมการต่อไปนี้

$$\text{stage} = \log_2 N \tag{4.6}$$



โดยที่ stage คือ จำนวนสถานะ

N คือ จำนวนข้อมูลที่รับเข้ามา (จุด)

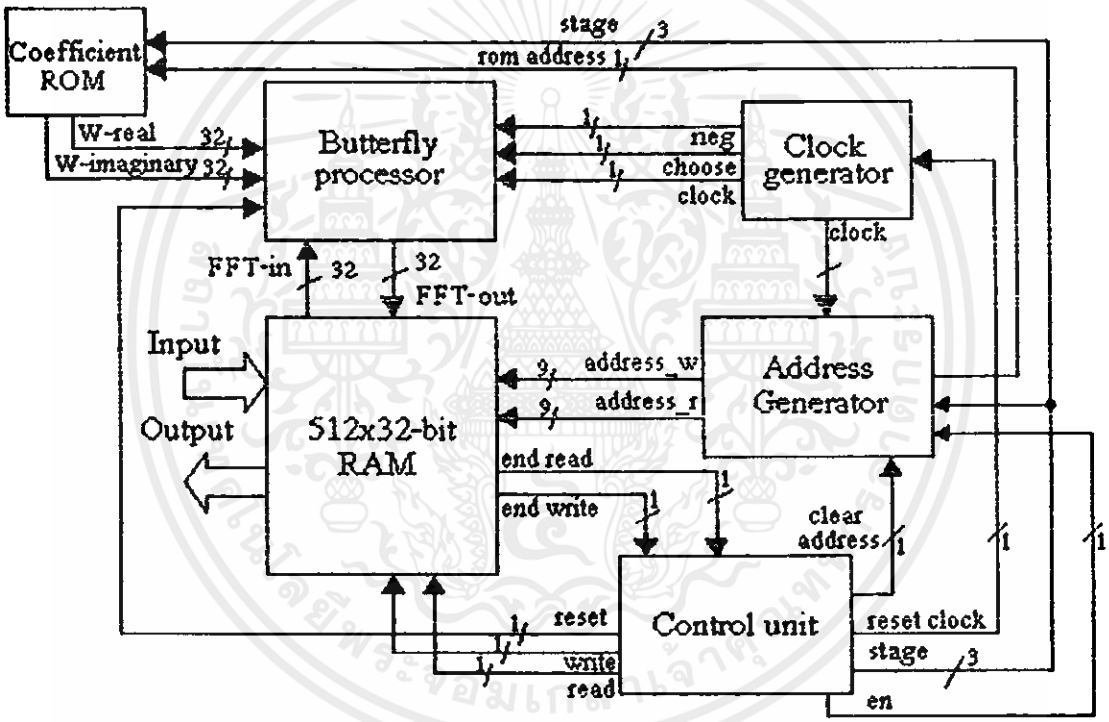
นำจำนวนข้อมูลที่รับเข้ามา (ข้อมูล 256 จุด) แทนค่าในสมการที่ 4.6 ได้

$$stage = \log_2 256$$

$$stage = \log_2 2^8$$

$$stage = 8 \log_2 2 = 8$$

ดังนั้นจำนวนสถานะที่ต้องใช้ในการคำนวณข้อมูลจำนวน 256 จุด จึงมีจำนวนเท่ากับ 8 สถานะ เมื่อนำมาออกแบบเป็นส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว สำหรับการรองรับการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน และข้อมูลที่รับเป็นแบบเลขอิงดรรชนีขนาด 32 บิตแล้ว จะได้บล็อกไดอะแกรมของการทำงาน ดังแสดงในรูปที่ 4.3




**รูปที่ 4.3** บล็อกไดอะแกรมส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

จากรูปที่ 4.3 สามารถอธิบายการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วได้คือ เมื่อข้อมูลเริ่มเข้ามาในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ส่วนการควบคุม (Control unit) จะสั่งให้วงจรจ่ายที่อยู่ของข้อมูล (Address Generator) จ่ายค่าที่อยู่สำหรับการเขียนข้อมูล (address_w) ไปให้หน่วยความจำ (RAM) เพื่อเก็บข้อมูลที่เข้ามาตามที่อยู่ที่กำหนดไว้ เมื่อข้อมูลถูกเก็บเข้าไปในหน่วยความจำครบหมดแล้ว หน่วยความจำจะส่งสัญญาณ (end write) มาที่ส่วนของการควบคุม เพื่อให้ส่วนของการควบคุมรู้ว่าข้อมูลได้เก็บครบแล้วพร้อมที่จะเริ่มการทำงานสถานะที่ 1 (stage 1) จากนั้นเริ่มการทำงานสถานะที่ 1 ส่วนของการควบคุมจะสั่งให้วงจร

จ่ายที่อยู่ของข้อมูลจ่ายที่อยู่สำหรับการอ่านข้อมูล (address_r) สำหรับการทำงานสถานะที่ 1 ไปให้หน่วยความจำเพื่อที่จะอ่านข้อมูลออกมาตามลำดับที่กำหนดไว้ตามค่าที่อยู่ของข้อมูล ข้อมูลที่อ่านออกมาจะส่งไปยังส่วนประมวลผลของแผนภาพผีเสื้อ (Butterfly processor) เพื่อทำการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด จากนั้นส่วนของการควบคุมจะสั่งให้วงจรจ่ายค่าสัมประสิทธิ์ (Coefficient ROM) จ่ายค่าสัมประสิทธิ์ออกมาตามลำดับที่ได้กำหนดเอาไว้ เพื่อที่จะนำเข้าไปใช้ในการคำนวณภายในส่วนประมวลผลของแผนภาพผีเสื้อ เมื่อคำนวณเสร็จส่วนของการควบคุมก็สั่งให้วงจรจ่ายที่อยู่ของข้อมูล จ่ายค่าที่อยู่สำหรับการเขียนข้อมูลไปให้หน่วยความจำ เพื่อที่จะเก็บผลลัพธ์ที่ได้จากการคำนวณไปไว้ที่ที่อยู่เดียวกับที่อ่านข้อมูลออกมา การทำงานทั้งหมดนี้เป็นการทำงานของสถานะที่ 1 และจะเป็นการทำงานซ้ำๆ กันไปจนครบตามจำนวนสถานะที่ออกแบบไว้ เนื่องจากมีการรับข้อมูลเข้ามาทั้งหมด 256 จุดแบบจำนวนเชิงซ้อน ทำให้ต้องมีการทำงานทั้งหมด 8 สถานะ เป็นไปตามสมการที่ 4.6 เมื่อทำการคำนวณครบทั้ง 8 สถานะแล้ว ก็จะส่งผลลัพธ์ที่ได้จากการคำนวณออกไป พร้อมทั้งรับข้อมูลชุดใหม่เข้ามา จากการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ออกแบบ สามารถนำมาเขียนเป็นสมการสำหรับคำนวณเวลาที่ใช้ในการแปลงฟูริเยร์แบบรวดเร็วได้ คือ

$$\frac{(N\times 2)+(((14+(N\times 2))\times stage)}{f} = \text{เวลาที่ใช้ในการแปลงฟูริเยร์แบบรวดเร็ว}$$

โดยที่ $N$ คือจำนวนข้อมูล

$f$ คือความถี่ที่ใช้ในการทำงาน

$stage$ คือจำนวนสถานะที่ใช้ในการงาน

และสามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบอยู่ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว คือ

**4.2.1 ส่วนประมวลผลแผนภาพผีเสื้อ (Butterfly processor)**

ส่วนประมวลผลแผนภาพผีเสื้อเป็นส่วนที่จะทำการคำนวณข้อมูลที่ส่งเข้ามา ซึ่งผลที่ได้ออกมาจากการคำนวณภายในส่วนประมวลผลแผนภาพผีเสื้อหนึ่งครั้ง จะมีค่าเท่ากับการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด



**4.2.2 ส่วนของการควบคุม (Control unit)**

เป็นวงจรที่คอยควบคุมการทำงานวงจรต่างๆ ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ทั้งหมดให้สามารถทำงานร่วมกันได้ตามสถานะที่กำหนดไว้

**4.2.3 วงจรจ่ายที่อยู่ของข้อมูล (Address Generator)**

เป็นวงจรที่จ่ายที่อยู่ของข้อมูลสำหรับการเขียน และการอ่านข้อมูลจากหน่วยความจำมาคำนวณ ซึ่งค่าที่อยู่ของข้อมูลจะมีค่าเปลี่ยนไปตามสถานะต่างๆ ของการทำงาน

**4.2.4 วงจรจ่ายค่าสัมประสิทธิ์ (Coefficient ROM)**

เป็นวงจรที่จ่ายค่าสัมประสิทธิ์ที่ใช้ในการคำนวณภายในส่วนประมวลผลแผนภาพผีเสื้อออกมา ตามค่าที่อยู่ของข้อมูลที่กำหนดไว้ ซึ่งค่าสัมประสิทธิ์ต่างๆ จะเก็บไว้ในลักษณะของค่าคงที่

**4.2.5 วงจรจ่ายสัญญาณนาฬิกา (Clock Generator)**

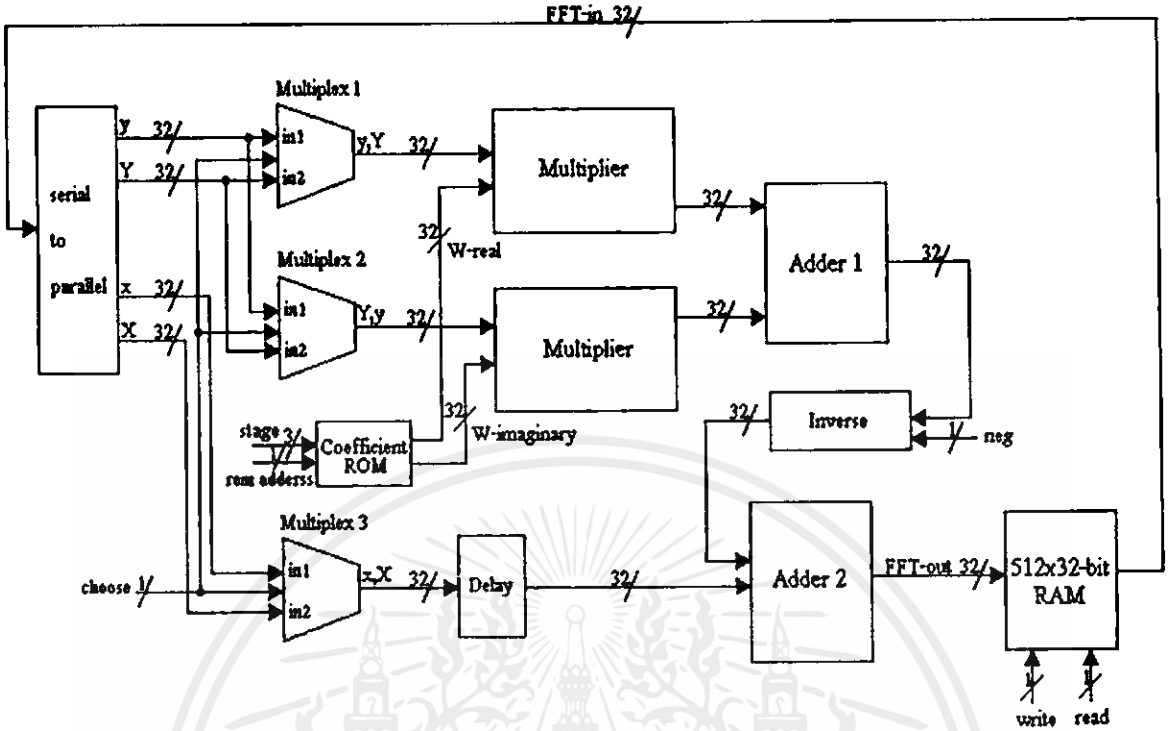
เป็นวงจรที่จ่ายสัญญาณนาฬิกาออกมาสำหรับใช้งานภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ซึ่งจะมีสัญญาณนาฬิกาออกมา 3 ความถี่สำหรับการใช้งานที่แตกต่างกัน

**4.2.6 วงจรหน่วยความจำ (RAM)**

เป็นลักษณะที่ใช้วงจรหน่วยความจำตัวเดียว (single ram) ซึ่งเป็นส่วนที่ใช้สำหรับการเก็บข้อมูลชั่วคราวระหว่างทำการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว โดยออกแบบให้มีลักษณะเป็นหน่วยความจำแบบ 2 ทาง (Dual port)

## 4.3 การออกแบบส่วนประมวลผลของแผนภาพผีเสื้อ (Butterfly processor)

การออกแบบส่วนประมวลผลของแผนภาพผีเสื้อ เป็นการออกแบบมาจากโครงสร้างของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด ตามรูปที่ 4.2 ซึ่งปกติแล้วในการออกแบบส่วนประมวลผลของแผนภาพผีเสื้อโดยใช้โครงสร้างของการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุดทั่วไปจะต้องการ 4 วงจรคูณและ 3 วงจรบวก [3,20] สำหรับใช้ในการคำนวณภายในส่วนประมวลผลของแผนภาพผีเสื้อ แต่เมื่อนำมาออกแบบการทำงานจริงแล้วพบว่ายังสามารถลดโครงสร้างการทำงานลงได้อีก เนื่องจากว่าส่วนประมวลผลของแผนภาพผีเสื้อที่ออกแบบการทำงานโดยใช้ 4 วงจรคูณและ 3 วงจรบวกนั้น โครงสร้างของส่วนประมวลผลของแผนภาพผีเสื้อจะแยกการคำนวณออกเป็น 2 ส่วน ส่วนแรกสำหรับคำนวณค่าจริง และส่วนที่สองสำหรับคำนวณค่าจินตภาพ ซึ่งทั้งสองส่วนนี้จะมีโครงสร้างการทำงานที่เหมือนกัน ทำให้สามารถลดโครงสร้างการทำงานดังกล่าวให้เหลือการทำงานเพียงส่วนเดียวได้ โดยให้ข้อมูลทั้งค่าจริง และค่าจินตภาพ เข้าไปคำนวณในส่วนของวงจรที่ใช้ในการคำนวณเดียวกัน ซึ่งทำให้โครงสร้างใหม่นี้สามารถลดโครงสร้างการทำงานให้เหลือเพียง 2 วงจรคูณและ 2 วงจรบวก สำหรับใช้ในการคำนวณภายในส่วนประมวลผลของแผนภาพผีเสื้อได้ แสดงในรูปที่ 4.4




รูปที่ 4.4 บล็อกไดอะแกรมส่วนประมวลผลของแผนภาพผีเสื้อ

จากรูปที่ 4.4 สามารถอธิบายการทำงานของส่วนประมวลผลของแผนภาพผีเสื้อได้คือ เมื่อรับข้อมูลแบบอนุกรมขนาด 32 บิต (FFT-in) มาจากหน่วยความจำแล้วจะส่งไปที่บล็อกการทำงานที่แปลงจากข้อมูลอนุกรมไปเป็นข้อมูลขนานจำนวน 4 ค่า (serial to parallel) โดยข้อมูลขนานทั้ง 4 ค่า จะแบ่งเป็น ค่าจริง x, ค่าจินตภาพ X, ค่าจริง y และค่าจินตภาพ Y โดยข้อมูลแบบขนานที่ออกมานั้นจะต้องขยายขนาดออกมาเป็น 4 เท่าของข้อมูลที่เข้าไป จากนั้นค่าจริง y และค่าจินตภาพ Y จะถูกส่งไปยังมัลติเพล็กซ์ 1 และมัลติเพล็กซ์ 2 ส่วนค่าจริง x และค่าจินตภาพ X จะถูกส่งไปยังมัลติเพล็กซ์ 3 โดยที่มัลติเพล็กซ์ 1 และมัลติเพล็กซ์ 3 จะมีการทำงานที่เหมือนกันคือ เมื่อสัญญาณ choose เข้ามาเป็น 0 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in1 แต่ถ้าสัญญาณ choose เข้ามาเป็น 1 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in2 ส่วนมัลติเพล็กซ์ 2 จะทำงานตรงข้ามกับมัลติเพล็กซ์ 1 และมัลติเพล็กซ์ 3 คือ เมื่อสัญญาณ choose เข้ามาเป็น 0 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in2 แต่ถ้าสัญญาณ choose เข้ามาเป็น 1 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in1 ดังนั้นเมื่อมีข้อมูลเข้ามาที่มัลติเพล็กซ์ ผลที่จะได้ออกมาคือ เป็นค่าจริง y สลับกับค่าจินตภาพ Y จากมัลติเพล็กซ์ 1, ค่าจินตภาพ Y สลับกับค่าจริง y จากมัลติเพล็กซ์ 2 และค่าจริง x สลับกับค่าจินตภาพ X จากมัลติเพล็กซ์ 3 จากนั้นค่าที่ได้ออกมาจากมัลติเพล็กซ์ 1 และมัลติเพล็กซ์ 2 จะถูกส่งไปคูณ (Multiplier) กับค่าสัมประสิทธิ์ (W) ที่จ่ายค่ามาจากวงจรจ่ายค่าสัมประสิทธิ์ (Coefficient Rom) โดยข้อมูลที่ได้ออกมาจากมัลติเพล็กซ์ 1 คือค่าจริง y สลับกับค่าจินตภาพ Y จะคูณกับค่าสัมประสิทธิ์ที่

เป็นค่าจริง (W-real) ส่วนข้อมูลที่ได้ออกมาจากมัลติเพล็กซ์ 2 คือ ค่าจินตภาพ Y สลับกับค่าจริง y จะคูณกับค่าสัมประสิทธิ์ที่เป็นค่าจินตภาพ (W-imaginary) เมื่อทำการคูณค่าเรียบร้อยแล้วผลลัพธ์ที่ได้จากการคูณทั้ง 2 ค่าจะถูกส่งไปบวกกันที่วงจรบวกที่ 1 (Adder1) เมื่อบวกกันเสร็จแล้วผลที่ได้จะส่งไปทำการกลับค่าสัญญาณ sign ที่บล็อก inverse ตามสัญญาณ neg ที่รับเข้ามา ซึ่งจะทำการกลับค่า sign จากบวกเป็นลบ หรือลบเป็นบวกทุกๆ 2 สัญญาณนาฬิกา ผลลัพธ์ที่ได้ออกมาจะส่งไปบวกกับค่าจริง x สลับกับค่าจินตภาพ X จาก มัลติเพล็กซ์ 3 ที่ถูกหน่วงเวลารอการคำนวณของค่าจริง y และค่าจินตภาพ Y ที่วงจรบวกที่ 2 (Adder2) ผลลัพธ์ที่ได้จากวงจรบวกที่ 2 คือ ผลลัพธ์ที่ได้จากการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุด ซึ่งผลลัพธ์ (FFT-out) ที่ได้ออกมาทั้งหมดจะถูกส่งไปเก็บไว้ในหน่วยความจำที่ที่อยู่เดียวกับที่อ่านข้อมูลออกมาจึงถือว่าเป็นการทำงานครบ 1 สถานะ และสามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในส่วนประมวลผลของแผนภาพผีเสื้อ คือ

**4.3.1 วงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน (Serial to parallel)**

เป็นวงจรที่ใช้ในการแปลงข้อมูลแบบอนุกรมที่อ่านออกมาจากหน่วยความจำ โดยทำการแปลงจากข้อมูลอนุกรมไปเป็นข้อมูลขนาน สำหรับส่งเข้าไปคำนวณ

**4.3.2 วงจรมัลติเพล็กซ์ (Multiplex)**

เป็นวงจรที่จะโหลดข้อมูลที่เข้ามาตามจังหวะของสัญญาณที่ใช้ในการเลือกโหลดข้อมูล โดยที่มัลติเพล็กซ์ 1 และมัลติเพล็กซ์ 3 จะมีการทำงานที่เหมือนกันคือ เมื่อสัญญาณ choose เข้ามาเป็น 0 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in1 แต่ถ้าสัญญาณ choose เข้ามาเป็น 1 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in2 ส่วนมัลติเพล็กซ์ 2 จะทำงานตรงข้ามกับมัลติเพล็กซ์ 1 และมัลติเพล็กซ์ 3 คือ เมื่อสัญญาณ choose เข้ามาเป็น 0 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in2 แต่ถ้าสัญญาณ choose เข้ามาเป็น 1 มัลติเพล็กซ์จะโหลดค่าที่เข้ามาทาง in1

**4.3.3 วงจรคูณ (Multiplier)**

เป็นวงจรที่ใช้ในการคูณข้อมูลที่เข้ามาในรูปแบบของเลขอิงดรรชนีขนาด 32 บิต กับค่าสัมประสิทธิ์แบบเลขอิงดรรชนีขนาด 32 บิต ที่ส่งมาจากวงจรจ่ายค่าสัมประสิทธิ์

**4.3.4 วงจรบวก (Adder)**

เป็นวงจรที่ใช้ในการบวกหรือลบข้อมูลในรูปแบบของเลขอิงดรรชนีขนาด 32 บิต โดยการบวกหรือลบนั้นจะขึ้นอยู่กับค่าเครื่องหมายของข้อมูลที่เข้ามา คือ ถ้าข้อมูลมีค่าของเครื่องหมายเหมือนกันจะนำข้อมูลมาบวกกัน แต่ถ้าค่าเครื่องหมายต่างกันจะนำข้อมูลมาลบกัน

**4.3.5 วงจรกลับค่าสัญลักษณ์ (Inverse)**

เป็นวงจรที่จะทำหน้าที่กลับค่าสัญลักษณ์ของข้อมูลจากสัญลักษณ์บวกไปเป็นสัญลักษณ์ลบ หรือจากสัญลักษณ์ลบไปเป็นสัญลักษณ์บวก

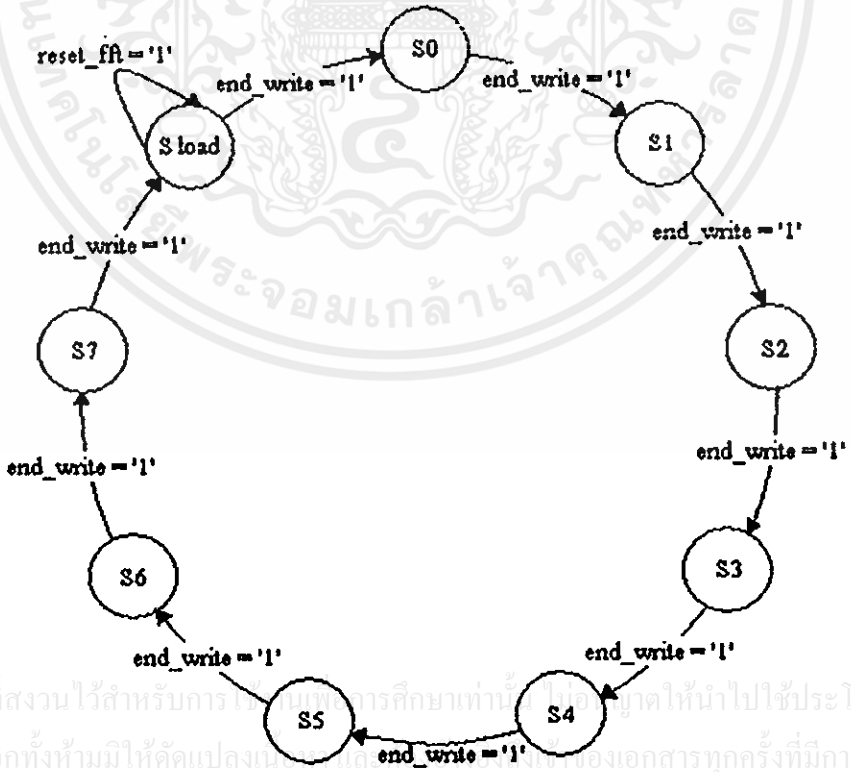
**4.3.6 วงจรหน่วงเวลา (Delay)**

เป็นวงจรที่ทำหน้าที่หน่วงเวลาของข้อมูล ตามเวลาที่กำหนดเอาไว้เพื่อรอการคำนวณข้อมูลในส่วนอื่นๆ

## 4.4 การออกแบบส่วนประกอบภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

**4.4.1 การออกแบบส่วนการควบคุม (Control unit)**

การออกแบบส่วนการควบคุม จะนำรูปแบบการทำงานของ Finitc-Statc Machinc (FSM) มาใช้ในการออกแบบ [22] โดยการทำงานทั้งหมดแบ่งออกเป็น 9 สถานะ คือจะมีสถานะที่ใช้ในการคำนวณทั้งหมด 8 สถานะ และจะมีสถานะที่ใช้ในการรับข้อมูล (S load) เพิ่มเข้ามาในการทำงานด้วยอีก 1 สถานะ โดยสามารถบรรยายด้วยแผนภาพไดอะแกรม แสดงในรูปที่ 4.5




**รูปที่ 4.5** แสดงแผนภาพไดอะแกรมส่วนการควบคุม

จากรูปที่ 4.5 สามารถอธิบายการทำงานได้ดังนี้คือ การทำงานจะเริ่มจากสถานะ (S load) เป็นสถานะแรก เมื่อรับสัญญาณ reset_fft ที่มีค่าเป็น 0 จะเริ่มเขียนข้อมูลที่เข้ามาไปเก็บไว้ในหน่วยความจำ และเมื่อรับสัญญาณ end_write ที่มีค่าเป็น 1 แสดงว่าข้อมูลทั้งหมดได้ถูกเก็บเข้าไปในหน่วยความจำครบหมดแล้ว สถานะการทำงานก็จะเปลี่ยนไปเป็นสถานะที่ 2 (S0) เมื่อการทำงานอยู่ที่สถานะที่ 2 (S0) ข้อมูลก็จะถูกอ่านออกจากหน่วยความจำเพื่อไปคำนวณ และเมื่อสัญญาณรับ end_write ที่มีค่าเป็น 1 แสดงว่าข้อมูลที่อ่านออกมาจากหน่วยความจำเพื่อส่งไปคำนวณได้คำนวณเสร็จ และเก็บเข้าไปในหน่วยความจำครบหมดแล้ว สถานะการทำงานก็จะเปลี่ยนไปเป็นสถานะที่ 3 (S1) ซึ่งการทำงานจะเป็นไปในลักษณะนี้จนถึงสถานะที่ 9 (S7) ซึ่งเป็นการคำนวณสถานะสุดท้าย และเมื่อคำนวณเสร็จผลลัพธ์ที่ได้ก็จะส่งออกไปจากส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว โดยที่การทำงานของแต่ละสถานะจะมีการส่งสัญญาณออกไปเพื่อควบคุมการทำงานของวงจรต่างๆ ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว สามารถอธิบายได้ดังต่อไปนี้ โปรแกรมแสดงในภาคผนวก ง.19

สัญญาณ io_mode เป็นสัญญาณที่ใช้ควบคุมหน่วยความจำ โดยเมื่อมีค่าเป็น 1 แสดงว่ามีการรับหรือส่งข้อมูลจากภายนอกส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ซึ่งเป็นเฉพาะสถานะแรก (S load) เท่านั้น ส่วนสถานะอื่นๆ จะมีค่าเป็น 0 แสดงว่ามีการรับหรือส่งข้อมูลจากภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

สัญญาณ reset จะมีค่าเป็น 1 ขึ้นมา 1 สัญญาณนาฬิกาแล้วกลับไปเป็นค่า 0 เพื่อยกเลิกการทำงานทั้งหมดของส่วนประมวลผลของแผนภาพผีเสื้อ ก่อนที่จะเริ่มการทำงานใหม่ทุกๆ สถานะ

สัญญาณ clear จะมีค่าเป็น 0 ขึ้นมา 1 สัญญาณนาฬิกาแล้วกลับไปเป็นค่า 1 เพื่อยกเลิกการทำงานของวงจรสร้างสัญญาณนาฬิกาทุกๆ สถานะ เพื่อให้ทุกครั้งที่มีการเปลี่ยนสถานะสัญญาณนาฬิกาจะเริ่มการทำงานใหม่ที่เหมือนกัน

สัญญาณ enable จะมีค่าเป็น 1 เพื่อให้วงจรต่างๆ ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วทำงาน

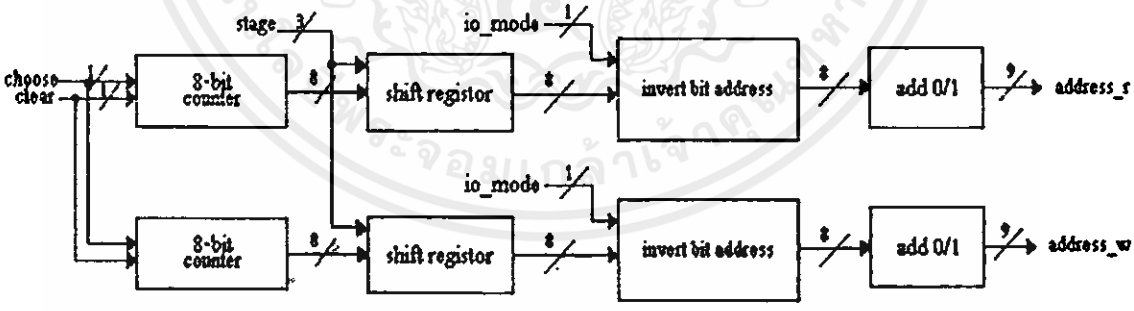
สัญญาณ read จะมีค่าเป็น 1 เพื่ออ่านข้อมูลจากหน่วยความจำทุกครั้งที่มีการเปลี่ยนสถานะใหม่และจะเป็น 0 เมื่ออ่านข้อมูลสุดท้ายเสร็จ

สัญญาณ write จะมีค่าเป็น 1 เมื่อสัญญาณ start_w มีค่าเป็น 1 (ข้อมูลคู่แรกคำนวณเสร็จ) เพื่อเขียนผลลัพธ์ที่ได้ลงหน่วยความจำ และจะเป็น 0 เมื่อเขียนผลลัพธ์สุดท้ายเสร็จ

สัญญาณ stage จะมีค่าเป็น 000 ที่ S load และที่ S0, 001 ที่ S1, 010 ที่ S2, 011 ที่ S3, 100 ที่ S4, 101 ที่ S5, 110 ที่ S6, 111 ที่ S7 เพื่อใช้สำหรับการจ่ายที่อยู่ของข้อมูล

**4.4.2 การออกแบบวงจรจ่ายที่อยู่ของข้อมูล (Address generator)**

การออกแบบวงจรจ่ายที่อยู่ของข้อมูล จะนำวงจรนับ (counter) ขนาด 8 บิตมาใช้ในการทำงาน โดยวงจรนับขนาด 8 บิตจะนับค่าออกมาได้ 256 ค่า จากนั้นจะส่งค่าที่นับออกมาไปที่วงจรเลื่อนบิต (shift register) เพื่อทำการเลื่อนบิตค่าที่อยู่ของข้อมูลที่นับออกมา โดยจะเลื่อนบิตตามจำนวนค่าสถานะที่ส่งมาจากส่วนการควบคุม เพื่อให้ได้ค่าบิตที่อยู่ของข้อมูลของแต่ละสถานะตามที่ได้ออกแบบไว้ แต่จากแผนภาพผีเสื้อรูปที่ 4.1 แสดงให้เห็นว่ามีการกลับค่าบิตที่อยู่ของข้อมูลก่อนที่จะเข้าไปทำงานในแผนผีเสื้อ เพื่อให้ผลลัพธ์เรียงตามลำดับจาก X(0), X(1),..., X(7) ดังนั้นจึงต้องใส่บล็อกที่ทำการกลับค่าบิตที่อยู่ของข้อมูลที่วงจรกลับค่าบิตที่อยู่ของข้อมูล (invert bit address) เพื่อทำการกลับค่าบิตที่อยู่ของข้อมูล (ช่วงเวลาที่รับข้อมูลจากภายนอกเข้ามานั้นจะไม่มีการทำการกลับค่าบิตที่อยู่ของข้อมูล ซึ่งจะแยกการทำงานโดยใช้สัญญาณ io_mode คือ เมื่อสัญญาณ io_mode มีค่าเป็น 1 บล็อกการกลับค่าบิตที่อยู่ของข้อมูลจะไม่มีการทำงาน) จากนั้นจะใส่ค่า 0 หรือ 1 เข้าไปในบิตหน้าสุดทางด้านซ้ายมือ (MSB) ที่วงจรใส่ 0/1 (add 0/1) ทำให้ได้ที่อยู่ของข้อมูลเป็น 512 ค่า และจะแบ่งการเก็บข้อมูลโดยที่อยู่ของข้อมูลที่บิตหน้าสุดทางด้านซ้ายมือเป็น 0 สำหรับเก็บข้อมูลค่าจริง และที่อยู่ของข้อมูลที่บิตหน้าสุดทางด้านซ้ายมือเป็น 1 สำหรับเก็บข้อมูลค่าจินตภาพ ซึ่งจะมีวงจรที่มีการทำงานลักษณะนี้ 2 ชุดคือสำหรับการอ่านข้อมูล และสำหรับการเขียนข้อมูล เนื่องจากว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วจะมีช่วงเวลาที่อ่านข้อมูล และเขียนข้อมูลในเวลาเดียวกันอยู่ด้วย แสดงในรูปที่ 4.6




**รูปที่ 4.6** บล็อกไดอะแกรมวงจรจ่ายที่อยู่ของข้อมูล

จากรูปที่ 4.6 สามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในวงจรจ่ายที่อยู่ของข้อมูล คือ



**4.4.2.1 วงจรนับขนาด 8 บิต (8-bit counter)**

เป็นวงจรนับขนาด 8 บิต โดยวงจรนับจะทำการนับตั้งแต่ค่า 0 จนถึงค่า 256 ซึ่งค่าที่นับได้จะส่งให้กับวงจรเลื่อนบิต โปรแกรมแสดงในภาคผนวก ง.11

**4.4.2.2 วงจรเลื่อนบิต (Shift Register)**

เป็นวงจรที่จะนำค่าที่อยู่ของข้อมูลขนาด 8 บิต มาเลื่อนบิตที่อยู่ของข้อมูลไปทางขวาตามค่าสถานะที่รับมาจากส่วนของการควบคุม โดยบิตทางขวาที่ถูกเลื่อนออกไปจะถูกนำมาต่อท้ายของบิตทางซ้าย โปรแกรมแสดงในภาคผนวก ง.12

**4.4.2.3 วงจรกลับค่าบิตที่อยู่ของข้อมูล (Invert Bit Address)**

เป็นวงจรที่จะทำหน้าที่เรียงลำดับของบิตที่อยู่ของข้อมูลใหม่จากบิตหลังไปบิตหน้าของลำดับบิตปกติ โปรแกรมแสดงในภาคผนวก ง.13

**4.4.2.4 วงจรใส่ 0/1 (add 0/1)**

เป็นวงจรที่ใส่ค่า 0 หรือ 1 เพิ่มเข้าไปในบิตหน้าสุดของที่อยู่ของข้อมูลขนาด 8 บิต เมื่อใส่ค่า 0 หรือ 1 ให้กับที่อยู่ของข้อมูลแล้วก็จะส่งออกไปใช้งาน โปรแกรมแสดงในภาคผนวก ง.14

**ตัวอย่างวิธีการจ่ายค่าที่อยู่ของข้อมูล**

กรณีที่ วงจรนับ นับ 2 คือ 00000010

ถ้าค่าสถานะที่รับมามีค่าเท่ากับ 001 (สถานะที่ S1) ดังนั้นจะทำการเลื่อนข้อมูลไปทางซ้าย 1 บิต และบิตซ้ายสุดก็จะถูกเลื่อนมาอยู่ทางด้านขวาสุดแทน

เมื่อผ่านการเลื่อนบิต (shift register) แล้วได้เป็น 00000100

จากนั้นทำการกลับค่าบิต (invert bit address) แล้วได้เป็น 00100000

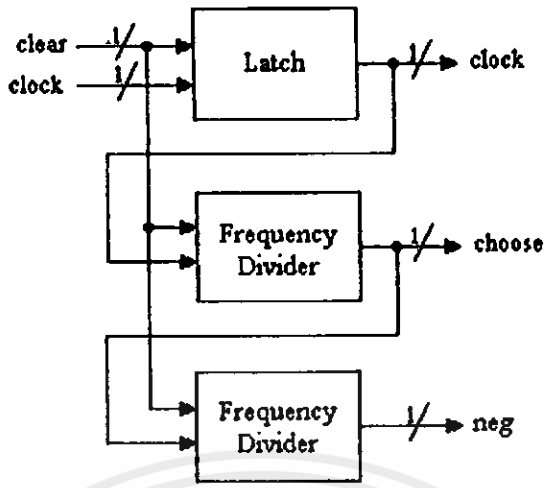
เมื่อใส่ 0 หรือ 1 (add0/1) เข้าไปจะได้ค่าที่อยู่ของข้อมูลเป็น

000100000 มีค่าเท่ากับ 32 ที่อยู่สำหรับข้อมูลค่าจริง

100100000 มีค่าเท่ากับ-288 ที่อยู่สำหรับข้อมูลค่าจินตภาพ

**4.4.3 การออกแบบวงจรสร้างสัญญาณนาฬิกา (Clock Generator)**

สัญญาณนาฬิกาที่ใช้ภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว มีทั้งหมด 3 สัญญาณ โดยสัญญาณนาฬิกา (clock) จะใช้กับวงจรคำนวณทั้งหมดภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ส่วนอีก 2 สัญญาณที่เหลือ จะใช้กับการทำงานในวงจรต่างๆ ของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว เช่น วงจรจ่ายค่าที่อยู่ของข้อมูล วงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน และวงจรอื่นๆ โดยขนาดของสัญญาณ choose จะมีขนาดใหญ่เป็น 2 เท่าของสัญญาณนาฬิกา และสัญญาณ neg จะมีขนาดใหญ่เป็น 4 เท่าของสัญญาณนาฬิกา แสดงในรูปที่ 4.7




รูปที่ 4.7 บล็อกไดอะแกรมวงจรสร้างสัญญาณนาฬิกา

จากรูปที่ 4.7 สามารถอธิบายการทำงานคือ สัญญาณนาฬิกาจะเป็นสัญญาณนาฬิกาปกติที่ป้อนเข้ามา แต่จะมีวงจรส่งผ่านสัญญาณ (Latch) ไว้สำหรับการยกเลิก (clear) การทำงาน และจะมีวงจรหารความถี่ (Frequency Divider) ที่จะหารความถี่ลงครึ่งหนึ่ง เพื่อสร้างสัญญาณ choose จากสัญญาณนาฬิกา และสร้างสัญญาณ neg จากสัญญาณ choose สามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในวงจรสร้างสัญญาณนาฬิกา คือ

**4.4.3.1 วงจรส่งผ่านสัญญาณ (Latch)**

เป็นวงจรที่รับสัญญาณความถี่นาฬิกาเข้ามา และส่งผ่านออกไปถ้าสัญญาณ clear มีค่าเป็น 1 แต่ถ้าสัญญาณ clear มีค่าเป็น 0 แล้วจะไม่มีการส่งสัญญาณนาฬิกาที่เข้ามาผ่านออกไป โปรแกรมแสดงในภาคผนวก ง.15

**4.4.3.2 วงจรหารความถี่ (Frequency Divider)**

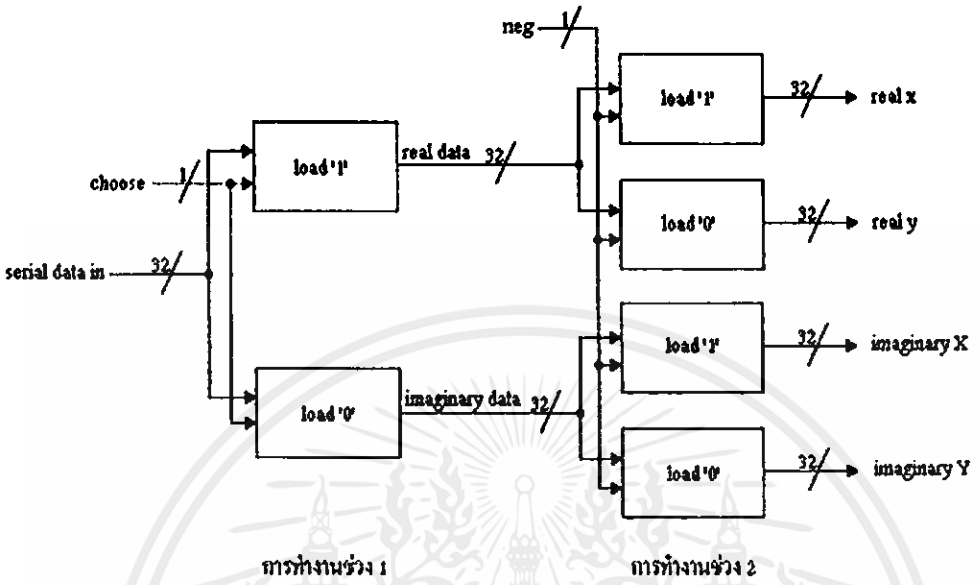
เป็นวงจรที่หารความถี่ของสัญญาณนาฬิกาที่เข้ามาลงครึ่งหนึ่ง เพื่อให้ได้สัญญาณนาฬิกาที่มีความถี่ต่ำลง โปรแกรมแสดงในภาคผนวก ง.16

**4.4.4 การออกแบบวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน (serial to parallel)**



การออกแบบวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน ส่วนประมวลผลของแผนภาพผีเสื้อต้องการข้อมูลในแบบขนานที่ได้จากการแปลงข้อมูลแบ่งออกมาเป็น 4 ส่วนคือ ค่าจริง x, ค่าจริง y, ค่าจินตภาพ X และค่าจินตภาพ Y ซึ่งข้อมูลที่ได้จากการแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนานต้องมีขนาดใหญ่ขึ้นเป็น 4 เท่าของข้อมูลที่เข้าไปเพื่อนำไปใช้ในการคำนวณ ดังนั้นการ

ออกแบบวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนานคือ ต้องใช้บล็อกการทำงานมีการโหลดข้อมูลเมื่อสัญญาณ (สัญญาณ choose กับสัญญาณ neg) ที่เข้ามามีค่าเป็น 0 หรือเป็น 1 แสดงในรูปที่ 4.8




รูปที่ 4.8 บล็อกไดอะแกรมวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน

จากรูปที่ 4.8 สามารถอธิบายการทำงานได้คือ เมื่อรับข้อมูลแบบอนุกรมเข้ามาแล้วการทำงานช่วงที่ 1 คือ วงจรดึงข้อมูลเมื่อเป็น 1 (load '1') จะโหลดข้อมูลเมื่อสัญญาณ choose มีค่าเป็น 1 ซึ่งค่าที่ได้จากการโหลดจะเป็นค่าจริงทั้งหมด ส่วนวงจรดึงข้อมูลเมื่อเป็น 0 (load '0') จะโหลดข้อมูลเมื่อสัญญาณ choose มีค่าเป็น 0 ซึ่งค่าที่ได้จากการโหลดจะเป็นค่าจินตภาพทั้งหมด จากนั้นข้อมูลจริงและข้อมูลจินตภาพที่แยกออกจากกันแล้วจะถูกส่งเข้าไปในการทำงานช่วงที่ 2 คือ ข้อมูลที่เป็นค่าจริงจะถูกส่งไปทำงานที่บล็อกการทำงานคู่บนคือ วงจรดึงข้อมูลเมื่อเป็น 1 (load '1') จะโหลดข้อมูลเมื่อสัญญาณ neg มีค่าเป็น 1 ซึ่งค่าที่ได้จากการโหลดจะเป็นค่าจริง x ส่วนวงจรดึงข้อมูลเมื่อเป็น 0 (load '0') จะโหลดข้อมูลเมื่อสัญญาณ neg มีค่าเป็น 0 ซึ่งค่าที่ได้จากการโหลดจะเป็นค่าจริง y ส่วนข้อมูลที่เป็นค่าจินตภาพจะถูกส่งไปทำงานที่บล็อกการทำงานคู่ล่างคือ วงจรดึงข้อมูลเมื่อเป็น 1 (load '1') จะโหลดข้อมูลเมื่อสัญญาณ neg มีค่าเป็น 1 ซึ่งค่าที่ได้จากการโหลดจะเป็นค่าจินตภาพ X ส่วนวงจรดึงข้อมูลเมื่อเป็น 0 (load '0') จะโหลดข้อมูลเมื่อสัญญาณ neg มีค่าเป็น 0 ซึ่งค่าที่ได้จากการโหลดจะเป็นค่าจินตภาพ Y ซึ่งข้อมูลแบบขนานทั้ง 4 ที่ได้ออกมา ก็จะมีขนาดใหญ่ขึ้นเป็น 4 เท่าของข้อมูลแบบอนุกรมที่ส่งเข้าไป สามารถอธิบายการทำงานของวงจรต่างๆ ที่ประกอบภายในวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน คือ

**4.4.4.1 วงจรดึงข้อมูลเมื่อเป็น 1 (load '1')**

เป็นวงจรที่จะดึงข้อมูลเข้ามาแล้วส่งออกไป เมื่อสัญญาณที่ควบคุมการทำงานมีค่าเป็น 1 โปรแกรมแสดงในภาคผนวก ง.17

**4.4.4.1 วงจรดึงข้อมูลเมื่อเป็น 0 (load '0')**

เป็นวงจรที่จะดึงข้อมูลเข้ามาแล้วส่งออกไป เมื่อสัญญาณที่ควบคุมการทำงานมีค่าเป็น 0 โปรแกรมแสดงในภาคผนวก ง.18

**4.4.5 การออกแบบวงจรจ่ายค่าสัมประสิทธิ์ (Coefficient Rom)**

ค่าสัมประสิทธิ์ที่ใช้สำหรับการคำนวณภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วจะเป็นค่าคงที่ ($W_N^k$) ซึ่งสามารถคำนวณหาค่าสัมประสิทธิ์ไว้ล่วงหน้าก่อนได้ การออกแบบครั้งนี้ใช้โปรแกรม Matlab สำหรับการคำนวณหาค่าสัมประสิทธิ์ (ภาคผนวก ก.) ตามคำสั่งต่อไปนี้

$$W = \exp(-j*2*pi*[0:N/2-1]/N)\ ; \qquad (4.7)$$

การจ่ายค่าสัมประสิทธิ์เพื่อไปคูณกับข้อมูลนั้น จะจ่ายค่าสัมประสิทธิ์ตามค่าที่อยู่ของข้อมูลที่กำหนดไว้ ซึ่งค่าสัมประสิทธิ์ที่จ่ายออกไปจะมีทั้งสัมประสิทธิ์ค่าจริง และค่าจินตภาพ ซึ่งทั้ง 2 ค่าจะจ่ายออกไปไม่เหมือนกันคือ ค่าสัมประสิทธิ์ที่เป็นค่าจริงสามารถจ่ายออกไปได้ตามปกติค่าละ 4 สัญญาณนาฬิกา แต่ถ้าเป็นการส่งค่าสัมประสิทธิ์ที่เป็นค่าจินตภาพแล้ว จะส่งค่าออกไปสลับกันระหว่างค่าจินตภาพปกติ กับค่าจินตภาพที่คูณด้วยค่า -1 สลับกันไปทุกๆ 1 สัญญาณนาฬิกาจนครบ 4 สัญญาณนาฬิกา ซึ่งการคูณค่าจินตภาพด้วยค่า -1 นั้นเพื่อเป็นการชดเชยกรณีที่ค่าจินตภาพมีการคูณกันเอง เนื่องจาก $j^2$ จะมีค่าเท่ากับ -1

**4.4.6 การออกแบบหน่วยความจำ (RAM)**

การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว จะออกแบบการใช้งานหน่วยความจำเป็นแบบหน่วยความจำตัวเดียว (single ram) เนื่องจากเป็นหน่วยความจำที่มีขนาดเล็กที่สุด และยังสามารถออกแบบส่วนการควบคุมการทำงานได้ง่าย [3] ซึ่งหน่วยความจำที่ออกแบบจะมีลักษณะเป็นแบบ 1 ทางเข้าและ 1 ทางออก (dual-port ram) ที่มีขนาดเท่ากับ 512x32 บิต เนื่องจากว่าหน่วยความจำลักษณะนี้สามารถออกแบบให้มีการอ่านข้อมูล และเขียนข้อมูลในเวลาเดียวกันได้แต่ต้องไม่ใช่ที่อยู่ของข้อมูลเดียวกัน ซึ่งทำให้สามารถที่จะเขียนข้อมูลที่คำนวณเสร็จแล้วกลับไปเก็บในหน่วยความจำที่อ่านออกมาได้เลย ไม่จำเป็นต้องรอให้อ่านข้อมูลออกมาคำนวณทั้งหมดก่อนแล้วค่อยเก็บข้อมูลลงหน่วยความจำ ซึ่งช่วยลดเวลาการทำงานของส่วนประมวลผล

ของการแปลงฟูริเยร์แบบรวดเร็วลงได้ และมีวงจรที่คอยตรวจสอบค่าที่อยู่ของข้อมูลคือ เมื่อที่อยู่ของข้อมูลมีค่าเท่ากับ 011111111 (255) ก็จะส่งสัญญาณ end_write หรือ end_read มีค่าเท่ากับ 1 ออกมา 1 สัญญาณนาฬิกา ขึ้นอยู่กับว่าที่อยู่ของข้อมูลนั้นเป็นที่อยู่สำหรับการเขียนหรือการอ่าน เพื่อบอกว่าการเขียนหรือการอ่านที่ทำอยู่ได้ทำเสร็จแล้ว และมีการตรวจสอบค่าสัญญาณ io_mode คือ เมื่อสัญญาณ io_mode มีค่าเป็น 1 จะเป็นการเขียนข้อมูลที่รับมาจากภายนอก แต่เมื่อสัญญาณ io_mode มีค่าเป็น 0 จะเป็นการเขียนผลลัพธ์ที่ได้มาจากการคำนวณในส่วนประมวลผลของแผนภาพผีเสื้อ

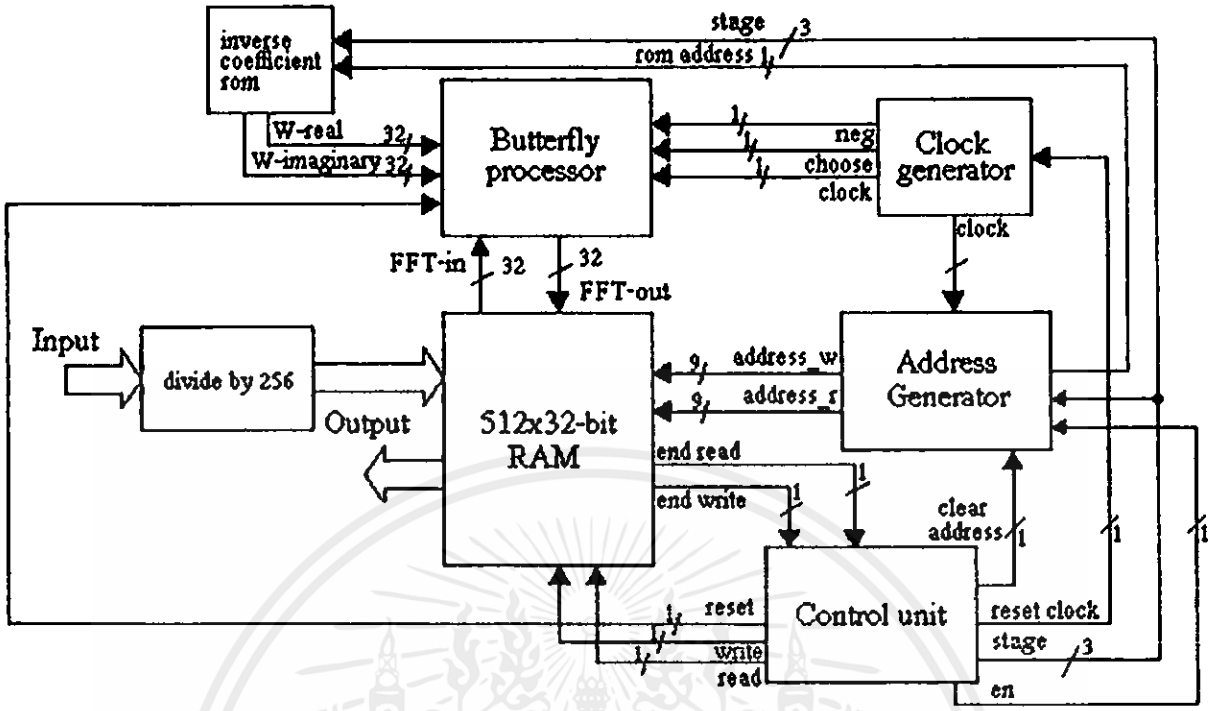
## 4.5 การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว (IFFT processor)

การออกแบบส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วนั้น วงจรหลักเกือบทั้งหมดสามารถใช้ร่วมกันกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วได้ เนื่องจากว่าส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วจะมีการทำงานที่คล้ายกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วมาก ดังนั้นเมื่อออกแบบส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วจึงสามารถที่จะใช้วงจรต่างๆ ร่วมกันกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วได้ แต่จะพบข้อแตกต่างกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วคือ จะมีวงจรหารค่าเท่ากับจำนวน N เพิ่มขึ้นมา และค่าสัมประสิทธิ์ที่ใช้สำหรับการคำนวณจะมีค่าไม่เหมือนกับค่าสัมประสิทธิ์ที่ใช้สำหรับการคำนวณภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ซึ่งค่าสัมประสิทธิ์ที่ใช้สำหรับการคำนวณภายในส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถคำนวณได้ โดยใช้โปรแกรม Matlab ตามคำสั่งต่อไปนี้

$$W = \exp(j*2*pi*[0:N/2-1]/N)\ ; \qquad (4.8)$$

ค่าสัมประสิทธิ์ที่ได้จากสมการที่ 4.8 (ภาคผนวก ก.) จะจ่ายออกมาในลักษณะเดียวกับการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

เมื่อนำมาออกแบบเป็นส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว สำหรับการรองรับการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน และข้อมูลที่รับเป็นแบบเลขอิงดรรชนีขนาด 32 บิต จะได้บล็อกไดอะแกรมของการทำงาน ดังแสดงในรูปที่ 4.9




รูปที่ 4.9 บล็อกไดอะแกรมส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

จากรูปที่ 4.9 จะสังเกตเห็นว่าสิ่งที่เปลี่ยนแปลงไปจากส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วคือ มีวงจรหารด้วยค่า 256 เพิ่มขึ้นมา และจะมีการเปลี่ยนค่าสัมประสิทธิ์ที่ใช้ในการคำนวณเท่านั้น ส่วนอื่นๆ จะไม่มีการเปลี่ยนแปลง ซึ่งสามารถอธิบายการทำงานได้ดังนี้

เมื่อรับข้อมูลเข้ามาภายในส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ข้อมูลทั้งหมดจะถูกส่งไปที่วงจรหาร (Divide) เพื่อทำการหารด้วยค่าคงที่ ที่มีค่าเท่ากับจำนวน N ซึ่งวงจรนี้มีค่าเท่ากับ 256 ผลที่ได้จากการหารก็จะนำไปเก็บไว้ในหน่วยความจำ เช่นเดียวกับการเริ่มต้นการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และขั้นตอนการทำงานต่อจากนี้ไปก็จะเหมือนกับการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว แต่จะต่างกันเพียงค่าสัมประสิทธิ์ที่ส่งออกมาคำนวณจะเป็นค่าสัมประสิทธิ์ชุดใหม่

จากการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่ปรับปรุง พบว่าสามารถลดการใช้งานวงจรคูณ และวงจรบวกลงไปได้ โดยเปรียบเทียบการใช้วงจรคูณ และวงจรบวกของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลาที่ปรับปรุงกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้โครงสร้างการทำงาน และวิธีการคำนวณแบบอื่นๆ ที่รับข้อมูล $N$ จุดได้ดัง แสดงในตารางที่ 4.1

ตารางที่ 4.1 เปรียบเทียบการใช้วงจรคูณและวงจรบวกของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ปรับปรุงกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้โครงสร้างการทำงาน และวิธีการคำนวณแบบอื่นๆ ที่รับข้อมูล $N$ จุด

| | วงจรคูณ(Multipliers) | วงจรบวก(Adders) |
|---|---|---|
| Radix-2 (วิธีระบบฐาน 2)[16] | $\frac{3}{2}\log_2 N-\frac{7}{2}N+8$ | $\frac{5}{2}\log_2 N-\frac{7}{2}N+8$ |
| Radix-4 (วิธีระบบฐาน 4)[16] | $\frac{9}{8}\log_2 N-3N+3$ | $\frac{25}{8}\log_2 N-3N+3$ |
| Radix-2 Multi-path Delay Commutator[20] | $2(\log_4 N-1)$ | $4\log_4 N$ |
| Radix-2 Single-path Delay Feedback[20] | $2(\log_4 N-1)$ | $4\log_4 N$ |
| Radix-4 Single-path Delay Feedback[20] | $\log_4 N-1$ | $8\log_4 N$ |
| Radix-4 Multi-path Delay Commutator[20] | $3(\log_4 N-1)$ | $8\log_4 N$ |
| Radix-4 Single-path Delay Commutator[20] | $(\log_4 N-1)$ | $3\log_4 N$ |
| Radix-$2^2$ Single-path Delay Feedback[20] | $(\log_4 N-1)$ | $4\log_4 N$ |
| วิธีระบบฐาน 2 ที่ปรับปรุง | 2 | 2 |

จากตารางที่ 4.1 แสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ปรับปรุงไม่ว่าจะรับข้อมูลเข้ามาคำนวณจำนวนเท่าไหร่ ก็จะใช้วงจรคำนวณทางคณิตศาสตร์ในจำนวนที่คงที่ คือ 2 วงจรคูณ และ 2 วงจรบวก เมื่อเทียบกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้โครงสร้างการทำงาน และวิธีการคำนวณแบบอื่นๆ วิธีพบว่าจำนวนวงจรคำนวณทางคณิตศาสตร์จะขึ้นอยู่กับจำนวนข้อมูล N ที่เข้ามาคือ ยิ่งรับข้อมูลเข้ามาคำนวณมาจำนวนมากขึ้นเท่าไหร่ ก็จำเป็นต้องใช้จำนวนวงจรคำนวณทางคณิตศาสตร์เพิ่มมากขึ้นตามสมการที่แสดงไว้ในตารางที่ 4.1 ซึ่งแสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ปรับปรุง จะใช้วงจรคำนวณทางคณิตศาสตร์ที่น้อยกว่า ซึ่งในส่วนนี้ก็จะส่งผลถึงขนาดของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วด้วย

## 4.6 สรุป

เนื้อหาในบทนี้ได้แสดงถึงการออกแบบ และการปรับปรุงส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดที่เล็กโดยการลดขนาดโครงสร้างการทำงานในส่วนประมวลผลของแผนภาพผีเสื้อ เนื่องจากเป็นส่วนการทำงานที่มีขนาดใหญ่ที่สุดภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว โดยสามารถลดโครงสร้างการทำงานภายในของส่วนประมวลผลของแผนภาพผีเสื้อจากปกติที่ต้องใช้ 4 วงจรคูณและ 3 วงจรบวก ให้เหลือเพียง

2 วงจรคูณและ 2 วงจรบวก ซึ่งมีผลทำให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วมีขนาดที่เล็กลง และเมื่อเทียบกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้โครงสร้างการทำงาน และวิธีการคำนวณแบบอื่นๆ พบว่าจำนวนวงจรคำนวณทางคณิตศาสตร์ที่ต้องใช้ของวิธีการคำนวณแบบอื่นนั้น จะขึ้นอยู่กับจำนวนข้อมูลที่เข้ามา

# บทที่ 5
# การทดสอบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

## 5.1 บทนำ

เมื่อออกแบบการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วด้วยภาษา VHDL แล้ว ในบทนี้จะเป็นการกล่าวถึงการทดสอบการทำงานวงจรที่ออกแบบว่าสามารถทำงานได้ตามที่ออกแบบไว้หรือไม่ โดยขั้นแรกจะเป็นการจำลองการทำงาน (Simulation) ของวงจรก่อนที่จะโปรแกรมลงบอร์ด FPGA เพื่อดูผลตอบสนองต่างๆ ของวงจร ขั้นตอนที่สองเป็นการทดสอบการทำงานบนบอร์ด FPGA เพื่อเป็นการตรวจสอบว่า ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่ได้ออกแบบไว้นั้น สามารถทำงานได้ตรงตามที่ได้ออกแบบไว้หรือไม่ โดยใช้เครื่องวัดสัญญาณดิจิตอล (Logic Analyzer) รวมทั้งตรวจสอบความผิดพลาดที่เกิดขึ้น จากนั้นจะนำไปทดสอบการทำงานโดยการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่ภาครับและภาคส่งข้อมูล

ในบทนี้ได้นำเสนอเกี่ยวกับการนำส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว มาทดสอบโดยการจำลองการทำงานบนเครื่องคอมพิวเตอร์ และทดสอบการทำงานบนบอร์ด FPGA โดยหัวข้อที่ 5.2 ผลที่ได้จากการจำลองการทำงาน (Simulation) หัวข้อที่ 5.3 เป็นการแสดงผลที่วัดได้จากการทำงานบนบอร์ด FPGA ด้วยเครื่องวัดสัญญาณดิจิตอล หัวข้อที่ 5.4 เป็นการนำส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่โปรแกรมการทำงานลงบอร์ด FPGA เพื่อทดสอบการทำงานที่จำลองการทำงานมาจากภาครับและภาคส่งข้อมูล หัวข้อที่ 5.5 แสดงผลจากการทดสอบเทียบกับผลจากโปรแกรม Matlab และหัวข้อที่ 5.6 เป็นบทสรุป

## 5.2 ผลจากการจำลองการทำงาน (Simulation)



จากการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว รวมทั้งวงจรต่างๆ โดยใช้ภาษา VHDL และนำมาจำลองการทำงานภายใต้เงื่อนไขของชิพ XILINX Spartan-II FPGA เบอร์ XC2S200 สามารถสรุปผลการออกแบบ และแสดงผลการจำลองการทำงานได้ดังนี้

**5.2.1 แสดงผลการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว (FFT processor)**

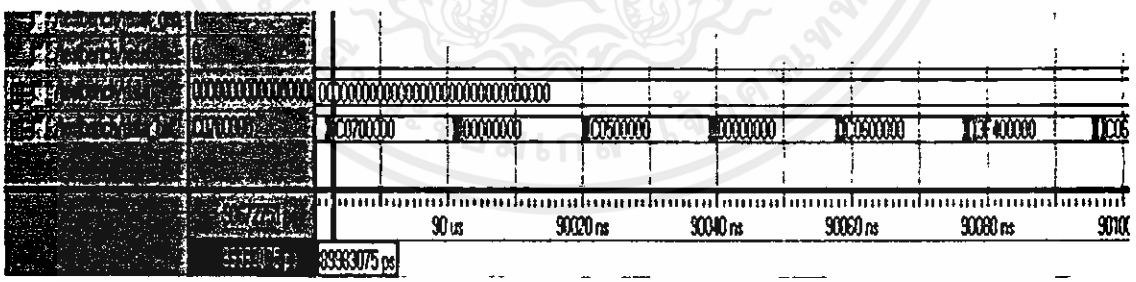
ผลที่ได้จากการจำลองการทำงานการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว แสดงในตารางที่ 5.1

**ตารางที่ 5.1** ผลจากการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 53 MHz |
| จำนวนเกทที่ใช้งาน (ไม่รวมหน่วยความจำ) | 37,281 เกท |
| จำนวน Slice ที่ใช้งาน | 2,141 Slice |
| เวลาทั้งหมดที่ใช้ในการคำนวณ (รวมเวลาที่ใช้รับข้อมูล) | 89 µsec |

จากตารางที่ 5.1 แสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 53 MHz โดยใช้เกทไปจำนวน 37,281 เกท และใช้ Slice ไปจำนวน 2,141 Slice ซึ่งเวลาทั้งหมดที่ใช้ในการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน (รวมเวลาที่ใช้รับข้อมูล) คือ 89 µsec

ผลของสัญญาณที่ได้จากการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว แสดงในรูปที่ 5.1




**รูปที่ 5.1** ผลการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

จากรูปที่ 5.1 ผลการจำลองการทำงานแสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วสามารถทำการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อนให้เสร็จได้ภายในเวลา 89 µsec

**5.2.2 แสดงผลการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว (IFFT processor)**

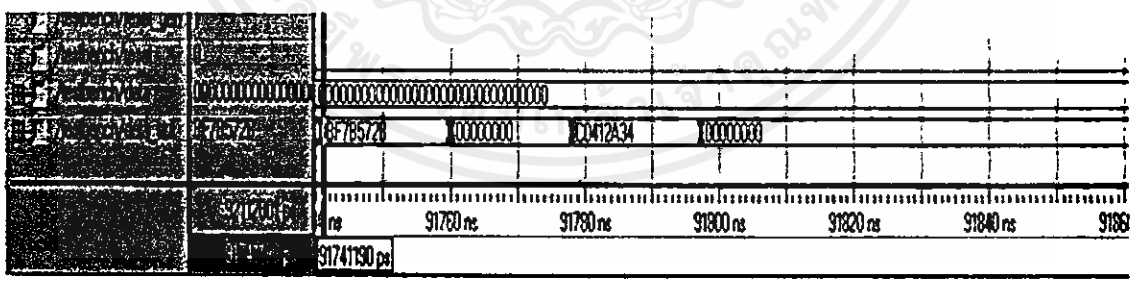
ผลที่ได้จากการจำลองการทำงานการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว แสดงในตารางที่ 5.2

**ตารางที่ 5.2** ผลจากการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด<br>Maximum clock frequency ($f_{max}$) | 52 MHz |
| จำนวนเกทที่ใช้งาน (ไม่รวมหน่วยความจำ) | 37,442 เกท |
| จำนวน Slice ที่ใช้งาน | 2,141 Slice |
| เวลาทั้งหมดที่ใช้ในการคำนวณ (รวมเวลาที่ใช้รับข้อมูล) | 91 µsec |

จากตารางที่ 5.2 แสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 52 MHz โดยใช้เกทไปจำนวน 37,442 เกท และใช้ Slice ไปจำนวน 2,141 Slice ซึ่งเวลาทั้งหมดที่ใช้ในการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อน (รวมเวลาที่ใช้รับข้อมูล) คือ 91 µsec

ผลของสัญญาณที่ได้จากการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว แสดงในรูปที่ 5.2




**รูปที่ 5.2** ผลการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว

จากรูปที่ 5.2 ผลการจำลองการทำงานแสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถทำการคำนวณข้อมูลจำนวน 256 จุดแบบจำนวนเชิงซ้อนให้เสร็จได้ภายในเวลา 91 µsec

**5.2.3 แสดงผลการจำลองการทำงานของส่วนประมวลผลของแผนภาพผีเสื้อ (Butterfly processor)**

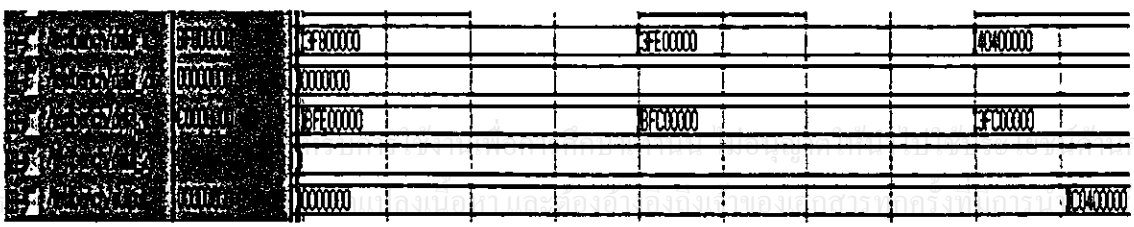
ผลจากการออกแบบของส่วนประมวลผลของแผนภาพผีเสื้อที่ปรับปรุงใหม่เปรียบเทียบกับส่วนประมวลผลของแผนภาพผีเสื้อก่อนการปรับปรุง แสดงในตารางที่ 5.3

**ตารางที่ 5.3** แสดงการเปรียบเทียบผลจากการออกแบบของส่วนประมวลผลของแผนภาพผีเสื้อก่อนและหลังการปรับปรุงโครงสร้างการทำงาน

| ค่าเปรียบเทียบ | ก่อนปรับปรุง | หลังปรับปรุง |
|---|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 57 MHz | 57 MHz |
| จำนวนเกทที่ใช้งาน | 63,462 เกท | 35,076 เกท |
| จำนวน Slice ที่ใช้งาน | 2,214 Slice | 1,496 Slice |
| เวลาที่ใช้ในการคำนวณ (Latency) | 13 สัญญาณนาฬิกา | 13 สัญญาณนาฬิกา |

จากตารางที่ 5.3 แสดงให้เห็นว่าส่วนประมวลผลของแผนภาพผีเสื้อหลังจากการปรับปรุงแล้วจะมีขนาดที่เล็กลงคือ ใช้เกทไปจำนวน 35,076 เกท และใช้ Slice ไปจำนวน 1,496 Slice เมื่อเทียบกับส่วนประมวลผลของแผนภาพผีเสื้อก่อนการปรับปรุงที่ใช้เกทไปจำนวน 63,462 เกท และใช้ Slice ไปจำนวน 2,214 Slice โดยที่ทั้งก่อนและหลังปรับปรุงส่วนประมวลผลของแผนภาพผีเสื้อสามารถทำงานที่ความถี่สูงสุดได้เท่ากันคือ 57 MHz และใช้เวลาในการคำนวณเท่ากันคือ 13 สัญญาณนาฬิกา ดังนั้นจากการปรับปรุงส่วนประมวลผลของแผนภาพผีเสื้อให้มีขนาดเล็กลง ก็จะมีผลทำให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วมีขนาดเล็กลงตามไปด้วย เนื่องจากส่วนประมวลผลของแผนภาพผีเสื้อเป็นส่วนที่มีขนาดใหญ่ที่สุดภายในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว

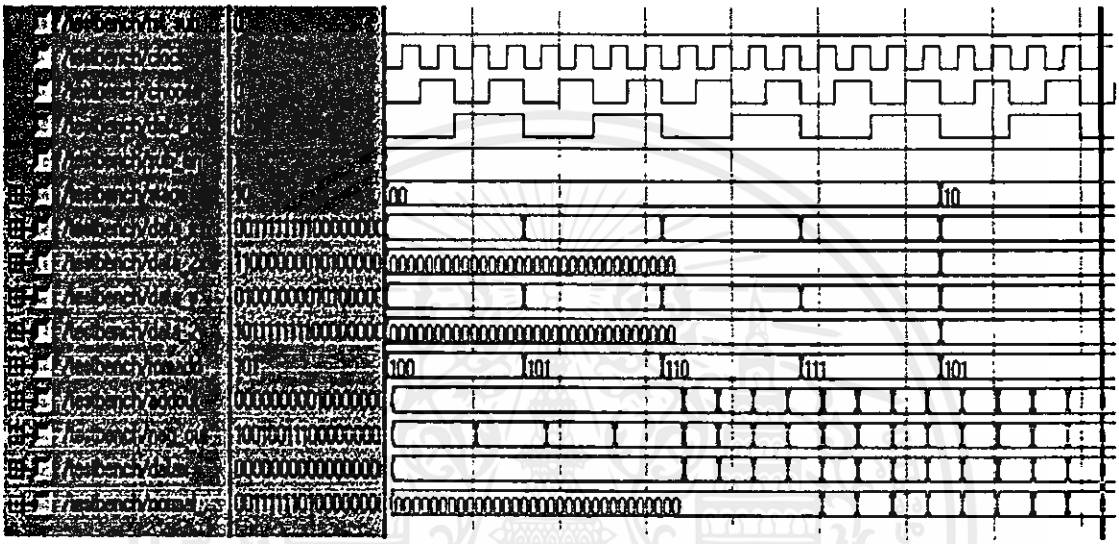
ผลของสัญญาณที่ได้จากการจำลองการทำงานของส่วนประมวลผลของแผนภาพผีเสื้อแสดงในรูปที่ 5.3-5.4



**รูปที่ 5.3** ผลการคำนวณของส่วนประมวลผลของแผนภาพผีเสื้อ

และจากรูปที่ 5.3 สามารถอธิบายผลการคำนวณที่ได้จากการจำลองการทำงาน คือ

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ -1 เปลี่ยนเป็นเลขฐาน 16 คือ BF800000
เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ -2 เปลี่ยนเป็นเลขฐาน 16 คือ C0000000
เมื่อนำไปคำนวณผลที่ได้คือ C0400000 ซึ่งมีค่าเท่า -3 ในระบบเลขอิงดรรชนี




**รูปที่ 5.4** ผลการจำลองการทำงานของส่วนประมวลผลของแผนภาพผีเสื้อ

จากรูปที่ 5.4 แสดงให้เห็นว่าส่วนประมวลผลของแผนภาพผีเสื้อ สามารถคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่อง 2 จุดได้เสร็จภายใน 13 สัญญาณนาฬิกา

**5.2.4 แสดงผลการจำลองการทำงานส่วนการควบคุม (Control unit)**

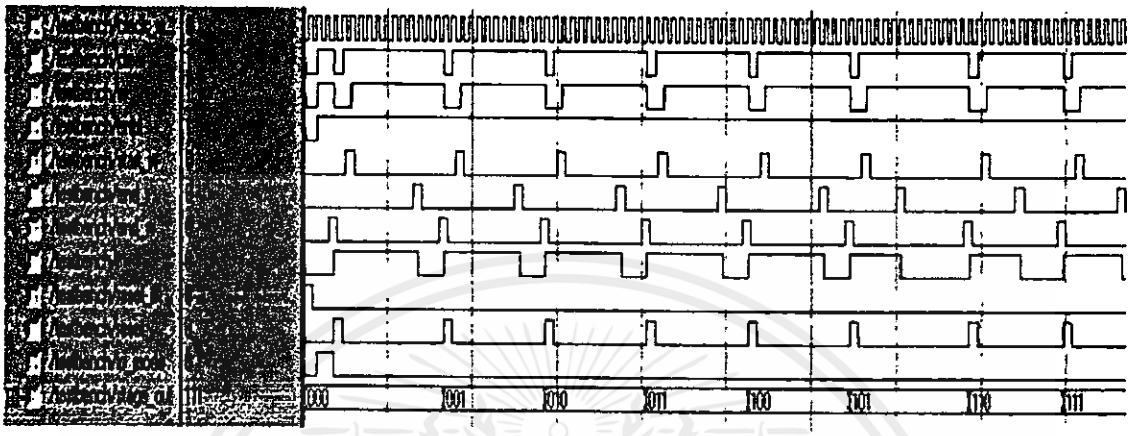
ผลที่ได้จากการจำลองการทำงานการออกแบบส่วนการควบคุม แสดงในตารางที่ 5.4

**ตารางที่ 5.4** ผลจากการออกแบบส่วนการควบคุม

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด<br>Maximum clock frequency ($f_{max}$) | 65 MHz |
| จำนวนเกทที่ใช้งาน | 915 เกท |
| จำนวน Slice ที่ใช้งาน | 78 Slice |

จากตารางที่ 5.4 ส่วนการควบคุมสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 65 MHz โดยใช้เกทไปจำนวน 915 เกท และใช้ Slice ไปจำนวน 78 Slice

ผลของสัญญาณที่ได้จากการจำลองการทำงานส่วนการควบคุม แสดงในรูปที่ 5.5



**รูปที่ 5.5** ผลการจำลองการทำงานส่วนการควบคุม

จากรูปที่ 5.5 แสดงให้เห็นว่าส่วนการควบคุมจะมีการทำงานทั้งหมด 9 สถานะ รวมสถานะที่มีการรับข้อมูลด้วย การทำงานเมื่อมีการรับข้อมูลจากภายนอกเข้ามา ส่วนการควบคุมจะส่งสัญญาณ io_mode ที่มีค่าเป็น 1 จากนั้นเมื่อเขียนข้อมูลเสร็จหน่วยเก็บความจำจะส่งสัญญาณ end_write มีค่าเป็น 1 ขึ้นมา 1 สัญญาณนาฬิกา มาให้ส่วนการควบคุมเพื่อเปลี่ยนสถานะการทำงาน โดยจะเริ่มการคำนวณสถานะที่ 1 จากนั้นส่วนการควบคุมจะเปลี่ยนสัญญาณ io_mode ให้มีค่าเป็น 0 และจะส่งสัญญาณ read มีค่าเป็น 1 เพื่ออ่านข้อมูลจากหน่วยความจำออกมาคำนวณ (เมื่ออ่านข้อมูลออกมาครบแล้วสัญญาณ read จะมีค่าเป็น 0) เมื่อคำนวณข้อมูลคู่ที่ 1 เสร็จส่วนประมวลผลของแผนภาพผีเสื้อก็จะส่งสัญญาณ start_w ที่มีค่าเป็น 1 ขึ้นมา 1 สัญญาณนาฬิกามาให้ส่วนการควบคุมเพื่อที่จะส่งสัญญาณ write ที่มีค่าเป็น 1 ออกไปเพื่อเขียนผลลัพธ์ที่ได้ลงหน่วยความจำ(เมื่อเขียนข้อมูลครบแล้วสัญญาณ write จะมีค่าเป็น 0) ถือเป็นการทำงานครบ 1 สถานะ และส่วนการควบคุมจะจ่ายสัญญาณ reset ที่มีค่าเป็น 1 ขึ้นมา 1 สัญญาณ เพื่อยกเลิกค่าการทำงานที่ค้างอยู่และส่งสัญญาณ clear เพื่อยกเลิกการทำงานของวงจรสร้างสัญญาณนาฬิกา ก่อนที่จะเริ่มการทำงานที่สถานะต่อไปเพื่อให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วสามารถทำงานได้อย่างถูกต้อง



**5.2.5 แสดงผลการจำลองการทำงานของวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี**

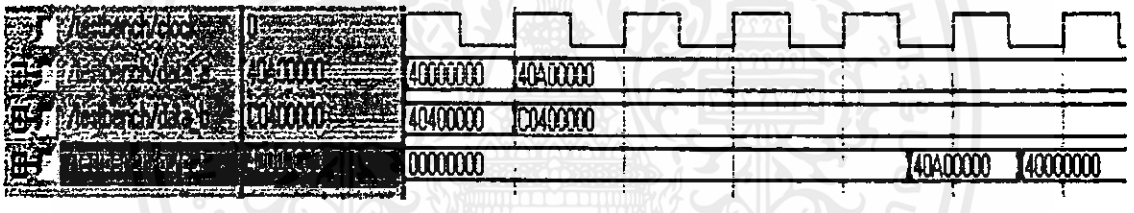
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี แสดงในตารางที่ 5.5

**ตารางที่ 5.5** ผลจากการออกแบบวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด<br>Maximum clock frequency ($f_{max}$) | 60 MHz |
| จำนวนเกทที่ใช้งาน | 7,950 เกท |
| จำนวน Slice ที่ใช้งาน | 377 Slice |
| เวลาที่ใช้ในการคำนวณ (Latency) | 5 สัญญาณนาฬิกา |

จากตารางที่ 5.5 แสดงให้เห็นว่าวงจรบวกและวงจรลบของระบบเลขอิงดรรชนีสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 60 MHz โดยใช้เกทไปจำนวน 7,950 เกท ใช้ Slice ไปจำนวน 337 Slice และใช้เวลาในการคำนวณทั้งหมด 5 สัญญาณนาฬิกา

ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี แสดงในรูปที่ 5.6




**รูปที่ 5.6** ผลการจำลองการทำงานของวงจรบวกและวงจรลบของระบบเลขอิงดรรชนี

จากรูปที่ 5.6 แสดงให้เห็นว่าวงจรบวกและวงจรลบของระบบเลขอิงดรรชนีสามารถที่จะบวก หรือลบข้อมูลได้เสร็จภายใน 5 สัญญาณนาฬิกา (clock cycle) และสามารถอธิบายผลการคำนวณที่ได้จากการจำลองการทำงาน คือ

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ 2 เปลี่ยนเป็นเลขฐาน 16 คือ 40000000

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ 3 เปลี่ยนเป็นเลขฐาน 16 คือ 40400000

ผลจากการบวกได้ 40A00000 ในเลขฐาน 16 เมื่อเปลี่ยนเป็นระบบเลขอิงดรรชนีจะมีค่าเท่ากับ 5



เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ 5 เปลี่ยนเป็นเลขฐาน 16 คือ 40A00000

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ -3 เปลี่ยนเป็นเลขฐาน 16 คือ C0400000

ผลจากการลบได้ 40000000 ในเลขฐาน 16 เมื่อเปลี่ยนเป็นระบบเลขอิงดรรชนีจะมีค่าเท่ากับ 2

**5.2.6 แสดงผลการจำลองการทำงานของวงจรคูณของระบบเลขอิงดรรชนี**

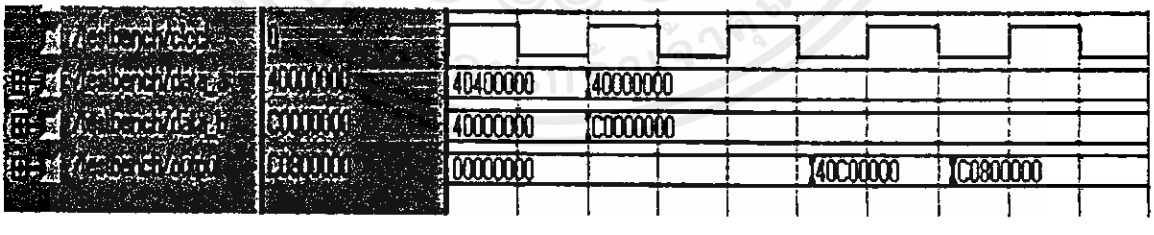
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรคูณของระบบเลขอิงดรรชนี แสดงในตารางที่ 5.6

**ตารางที่ 5.6** ผลจากการออกแบบวงจรคูณของระบบเลขอิงดรรชนี

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 93 MHz |
| จำนวนเกทที่ใช้งาน | 7,525 เกท |
| จำนวน Slice ที่ใช้งาน | 278 Slice |
| เวลาที่ใช้ในการคำนวณ (Latency) | 3 สัญญาณนาฬิกา |

จากตารางที่ 5.6 แสดงให้เห็นว่าวงจรคูณของระบบเลขอิงดรรชนีสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 93 MHz โดยใช้เกทไปจำนวน 7,525 เกท ใช้ Slice ไปจำนวน 278 Slice และใช้เวลาในการคำนวณทั้งหมด 3 สัญญาณนาฬิกา

ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรคูณของระบบเลขอิงดรรชนี แสดงในรูปที่ 5.7




**รูปที่ 5.7** ผลการจำลองการทำงานของวงจรคูณของระบบเลขอิงดรรชนี

จากรูปที่ 5.7 แสดงให้เห็นว่าวงจรคูณของระบบเลขอิงดรรชนีสามารถที่จะคูณข้อมูลได้เสร็จภายใน 3 สัญญาณนาฬิกา (clock cycle) และสามารถอธิบายผลการคำนวณที่ได้จากการจำลองการทำงาน คือ

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ 3 เปลี่ยนเป็นเลขฐาน 16 คือ 40400000

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ 2 เปลี่ยนเป็นเลขฐาน 16 คือ 40000000

ผลจากการคูณได้ 40C00000 ในเลขฐาน 16 เมื่อเปลี่ยนเป็นระบบเลขอิงดรรชนีจะมีค่าเท่ากับ 6

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ 2 เปลี่ยนเป็นเลขฐาน 16 คือ 40000000

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ -2 เปลี่ยนเป็นเลขฐาน 16 คือ C0000000

ผลจากการคูณได้ C0800000 ในเลขฐาน 16 เมื่อเปลี่ยนเป็นระบบเลขอิงดรรชนีจะมีค่าเท่ากับ -4

**5.2.7 แสดงผลการจำลองการทำงานของวงจรหารของระบบเลขอิงดรรชนี**

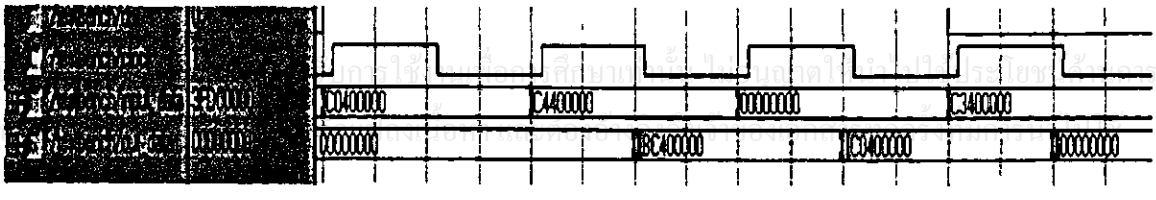
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรหารของระบบเลขอิงดรรชนีที่หารด้วยค่าคงที่ ที่มีค่าเท่ากับ 256 แสดงในตารางที่ 5.7

**ตารางที่ 5.7** ผลจากการออกแบบวงจรหารของระบบเลขอิงดรรชนี

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 161 MHz |
| จำนวนเกทที่ใช้งาน | 683 เกท |
| จำนวน Slice ที่ใช้งาน | 42 Slice |
| เวลาที่ใช้ในการคำนวณ (Latency) | 2 สัญญาณนาฬิกา |

จากตารางที่ 3.3 แสดงให้เห็นว่าวงจรหารของระบบเลขอิงดรรชนีสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 161 MHz โดยใช้เกทไปจำนวน 683 เกท ใช้ Slice ไปจำนวน 42 Slice และใช้เวลาในการคำนวณทั้งหมด 2 สัญญาณนาฬิกา

ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรหารของระบบเลขอิงดรรชนี แสดงในรูปที่ 5.8



**รูปที่ 5.8** ผลการจำลองการทำงานของวงจรหารของระบบเลขอิงดรรชนี

จากรูปที่ 5.8 แสดงให้เห็นว่าวงจรหารของระบบเลขอิงดรรชนีสามารถที่จะหารข้อมูลได้เสร็จภายใน 2 สัญญาณนาฬิกา (clock cycle) และสามารถอธิบายผลการคำนวณที่ได้จากการจำลองการทำงาน คือ

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ -3 เปลี่ยนเป็นเลขฐาน 16 คือ C0400000

ผลจากการหารได้ BC400000 ในเลขฐาน 16 เมื่อเปลี่ยนเป็นระบบเลขอิงดรรชนีจะมีค่าเท่ากับ -0.01171875

เลขจากระบบเลขอิงดรรชนีมีค่าเท่ากับ -768 เปลี่ยนเป็นเลขฐาน 16 คือ C4400000

ผลจากการหารได้ C0400000 ในเลขฐาน 16 เมื่อเปลี่ยนเป็นระบบเลขอิงดรรชนีจะมีค่าเท่ากับ -3

**5.2.8 แสดงผลการจำลองการทำงานของหน่วยความจำ (ram)**

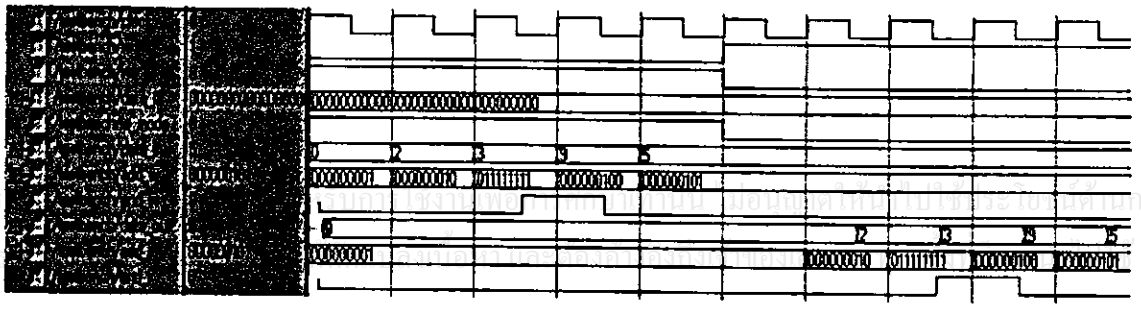
ผลที่ได้จากการจำลองการทำงานการออกแบบหน่วยความจำ แสดงในตารางที่ 5.8

**ตารางที่ 5.8** ผลจากการออกแบบหน่วยความจำ

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 121 MHz |
| จำนวนเกทที่ใช้งาน | 65,792 เกท |
| จำนวน Slice ที่ใช้งาน | 22 Slice |

จากตารางที่ 5.8 หน่วยความจำสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 121 MHz โดยใช้เกทไปจำนวน 65,792 เกท และใช้ Slice ไปจำนวน 22 Slice

ผลของสัญญาณที่ได้จากการจำลองการทำงานของหน่วยความจำ แสดงในรูปที่ 5.9



**รูปที่ 5.9** ผลการจำลองการทำงานของหน่วยความจำ

จากรูปที่ 5.9 แสดงให้เห็นว่าเมื่อทำการเขียนข้อมูลจากภายนอกเข้ามาสัญญาณ io_mode ต้องมีค่าเป็น 1 และเมื่อทำการอ่าน หรือเขียนข้อมูลถึงค่าที่อยู่ของข้อมูลเท่ากับ 011111111 (255) แล้วหน่วยความจำจะมีการส่งสัญญาณ end_read หรือ end_write ที่มีค่าเท่ากับ 1 ออกไป

**5.2.9 แสดงผลการจำลองการทำงานของวงจรจ่ายค่าสัมประสิทธิ์ (Coefficient Rom)**

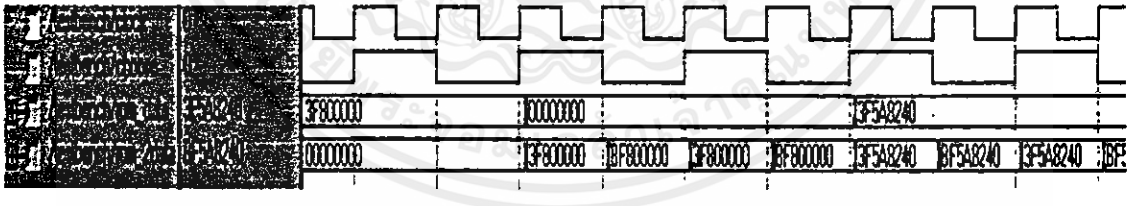
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรจ่ายค่าสัมประสิทธิ์ แสดงในตารางที่ 5.9

**ตารางที่ 5.9** ผลจากการออกแบบวงจรจ่ายค่าสัมประสิทธิ์

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 152 MHz |
| จำนวนเกทที่ใช้งาน | 2,486 เกท |
| จำนวน Slice ที่ใช้งาน | 297 Slice |

จากตารางที่ 5.9 วงจรจ่ายค่าสัมประสิทธิ์สามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 152 MHz โดยใช้เกทไปจำนวน 2,486 เกท และใช้ Slice ไปจำนวน 297 Slice

ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรจ่ายค่าสัมประสิทธิ์ แสดงในรูปที่ 5.10



**รูปที่ 5.10** ผลการจำลองการทำงานของวงจรจ่ายค่าสัมประสิทธิ์

จากรูปที่ 5.10 แสดงการจ่ายค่าสัมประสิทธิ์ให้กับส่วนประมวลผลของแผนภาพผีเสื้อเพื่อใช้ในการคำนวณคือ ค่าสัมประสิทธิ์ที่เป็นค่าจริงจะส่งเป็นค่าคงที่ค่าเดียวออกไปตลอด 4 สัญญาณนาฬิกา ส่วนค่าสัมประสิทธิ์ที่เป็นค่าจินตภาพจะส่งค่าออกไปสลับกันระหว่างค่าจินตภาพปกติ กับค่าจินตภาพที่คูณด้วยค่า -1 สลับกันไปทุกๆ 1 สัญญาณนาฬิกาจนครบ 4 สัญญาณนาฬิกา สามารถอธิบายค่าสัมประสิทธิ์ส่วนที่เป็นค่าจินตภาพที่ส่งออกไปได้คือ

ค่า 3F800000 จะมีค่าเท่ากับ 1 ในระบบเลขอิงดรรชนี

ค่า BF800000 จะมีค่าเท่ากับ -1 ในระบบเลขอิงดรรชนี

ค่า 3F5A8240 จะมีค่าเท่ากับ 0.7071 ในระบบเลขอิงดรรชนี

ค่า BF5A8240 จะมีค่าเท่ากับ -0.7071 ในระบบเลขอิงดรรชนี

### 5.2.10 แสดงผลการจำลองการทำงานของวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน (serial to parallel)

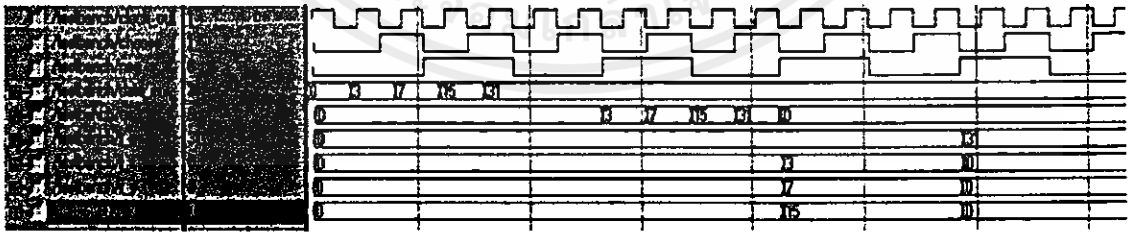
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน แสดงในตารางที่ 5.10

**ตารางที่ 5.10** ผลจากการออกแบบวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 161 MHz |
| จำนวนเกทที่ใช้งาน | 1,350 เกท |
| จำนวน Slice ที่ใช้งาน | 82 Slice |

จากตารางที่ 5.10 วงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนานสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 161 MHz โดยใช้เกทไปจำนวน 1,350 เกท และใช้ Slice ไปจำนวน 82 Slice

ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน แสดงในรูปที่ 5.11



**รูปที่ 5.11** ผลการจำลองการทำงานของวงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน



จากรูปที่ 5.11 แสดงให้เห็นว่าข้อมูลอนุกรมที่อ่านออกมาจากหน่วยความจำผ่านทางออก qq มีขนาด 1 สัญญาณนาฬิกา และเมื่อนำมาแปลงจากข้อมูลอนุกรมไปเป็นข้อมูลขนาน โดยใช้วงจรแปลงข้อมูลอนุกรมไปเป็นข้อมูลขนาน ที่ใช้สัญญาณ choose และสัญญาณ neg ในการแปลง

ข้อมูลอนุกรมไปเป็นข้อมูลขนานแล้ว ข้อมูลขนานที่ได้ออกมาทั้ง 4 ค่าจะมีขนาดใหญ่ขึ้นเป็น 4 เท่าของข้อมูลอนุกรมที่เข้าไป

**5.2.11 แสดงผลการจำลองการทำงานของวงจรสร้างสัญญาณนาฬิกา (Clock Generator)**

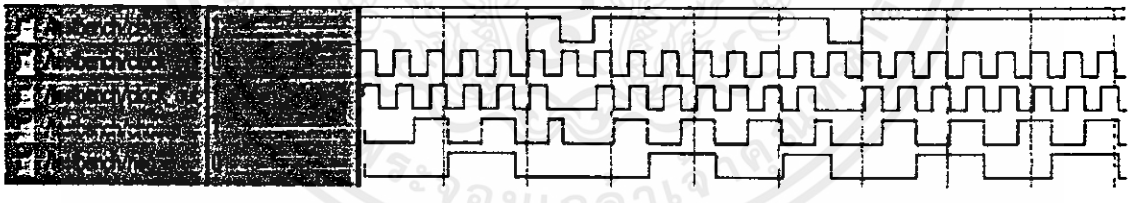
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรสร้างสัญญาณนาฬิกา แสดงในตารางที่ 5.11

**ตารางที่ 5.11** ผลจากการออกแบบวงจรสร้างสัญญาณนาฬิกา

| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด<br>Maximum clock frequency ($f_{max}$) | 183 MHz |
| จำนวนเกทที่ใช้งาน | 232 เกท |
| จำนวน Slice ที่ใช้งาน | 15 Slice |

จากตารางที่ 5.11 วงจรสร้างสัญญาณนาฬิกาสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 183 MHz โดยใช้เกทไปจำนวน 232 เกท และใช้ Slice ไปจำนวน 15 Slice

ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรสร้างสัญญาณนาฬิกา แสดงในรูปที่ 5.12



**รูปที่ 5.12** ผลการจำลองการทำงานของวงจรสร้างสัญญาณนาฬิกา

จากรูปที่ 5.12 แสดงการจ่ายสัญญาณนาฬิกาทั้ง 3 ขนาดออกมา และเมื่อสัญญาณ clear มีค่าเป็น 0 ระบบจะไม่มีการทำงาน และเมื่อสัญญาณ clear มีค่าเป็น 1 ระบบก็จะเริ่มทำงานใหม่อีกครั้ง วงจรสร้างสัญญาณนาฬิกาจะถูกการยกเลิกการทำงานด้วยสัญญาณ clear ทุกๆ ครั้งที่มีการเปลี่ยนสถานะของการทำงาน เพื่อให้สัญญาณนาฬิกาที่ใช้ในการทำงานเหมือนกันทุกสถานะ

**5.2.12 แสดงผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูล (Address generator)**

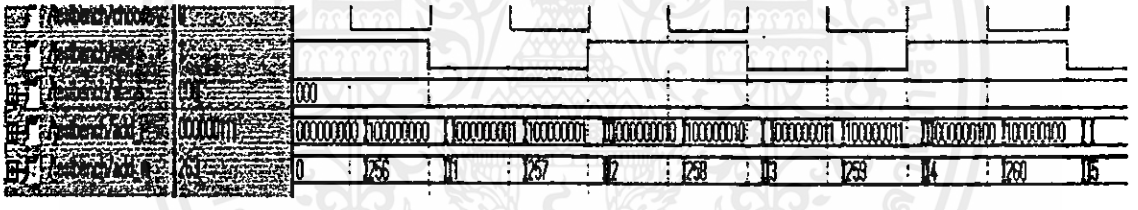
ผลที่ได้จากการจำลองการทำงานการออกแบบวงจรจ่ายที่อยู่ของข้อมูล แสดงในตารางที่ 5.12

**ตารางที่ 5.12** ผลจากการออกแบบวงจรจ่ายที่อยู่ของข้อมูล

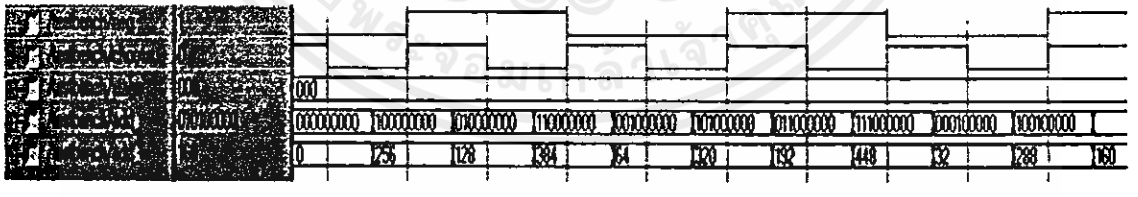
| | |
|---|---|
| ความถี่สัญญาณนาฬิกาสูงสุด<br>Maximum clock frequency ($f_{max}$) | 97 MHz |
| จำนวนเกทที่ใช้งาน | 1,259 เกท |
| จำนวน Slice ที่ใช้งาน | 87 Slice |

จากตารางที่ 5.12 วงจรจ่ายที่อยู่ของข้อมูลสามารถทำงานได้ที่ความถี่สูงสุดเท่ากับ 97 MHz โดยใช้เกทไปจำนวน 1,259 เกท และใช้ Slice ไปจำนวน 87 Slice

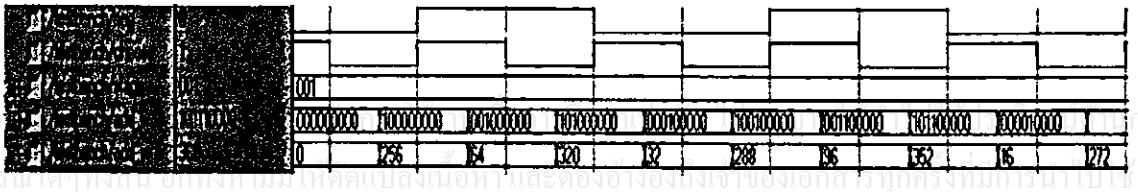
ผลของสัญญาณที่ได้จากการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูล จะมีการทำงานทั้งหมด 9 สถานะ รวมสถานะที่มีการรับข้อมูล แสดงในรูปที่ 5.13 – 5.21



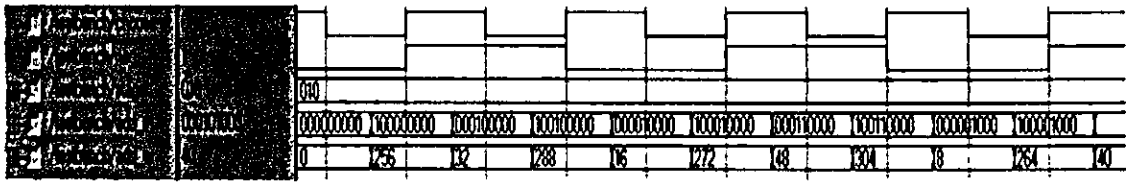
**รูปที่ 5.13** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 1 (S load)



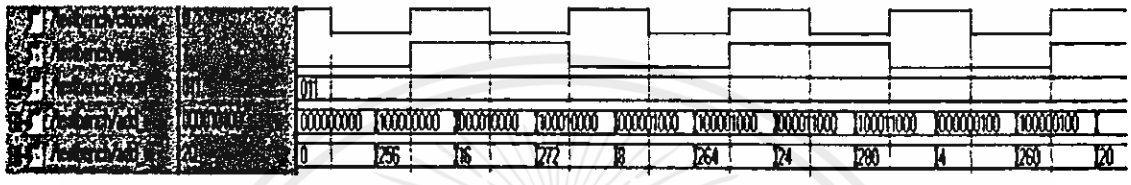
**รูปที่ 5.14** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 2 (S0)



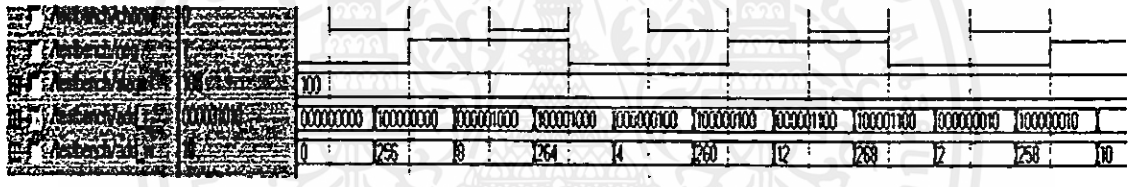
**รูปที่ 5.15** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 3 (S1)



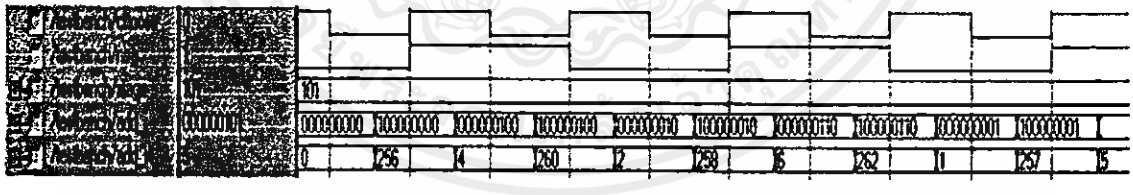
**รูปที่ 5.16** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 4 (S2)



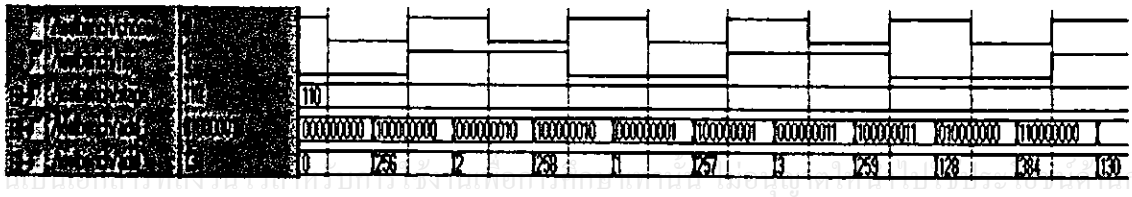
**รูปที่ 5.17** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 5 (S3)



**รูปที่ 5.18** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 6 (S4)

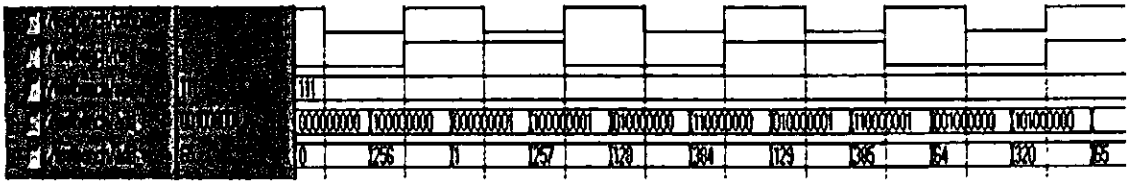


**รูปที่ 5.19** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 7 (S5)





**รูปที่ 5.20** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 8 (S6)



**รูปที่ 5.21** ผลการจำลองการทำงานของวงจรจ่ายที่อยู่ของข้อมูลสถานะที่ 9 (S7)

จากรูปที่ 5.13 แสดงให้เห็นว่าการทำงานของสถานะที่เป็นการรับข้อมูลจากภายนอกเข้ามาในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วนั้น จะไม่มีการทำการกลับบิตค่าที่อยู่ของข้อมูล ส่วนรูปที่ 5.14 – 5.21 แสดงให้เห็นว่าการทำงานของสถานะอื่นๆ จะมีการทำการกลับบิตค่าที่อยู่ของข้อมูล

จากการออกแบบ และการจำลองการทำงานพบว่า เมื่อนำผลที่ได้จากการออกแบบของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ววิธีระบบฐาน 2 ที่ปรับปรุงส่วนประมวลผลของแผนภาพผีเสื้อให้มีขนาดที่เล็กลง มาเปรียบเทียบกับผลการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ที่ใช้วิธีการคำนวณแบบอื่นที่มีการรับข้อมูลขนาด 20 บิต มีผลดังแสดงในตารางที่ 5.13

**ตารางที่ 5.13** เปรียบเทียบผลการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ปรับปรุง กับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้วิธีการคำนวณแบบอื่น ที่มีการรับข้อมูลขนาด 20 บิต

| ค่าต่างๆที่นำมาพิจารณา | วิธีระบบฐาน 4 [16] | วิธีระบบฐาน 2 ที่ปรับปรุง |
|---|---|---|
| บิตของข้อมูลที่ใช้ในวงจร | 20 bits | 20 bits |
| จำนวนเกทที่ใช้งานทั้งหมด (ไม่รวมหน่วยความจำ) | 98,326 เกท | 20,337 เกท |
| เวลาที่ใช้ในการคำนวณ (ไม่รวมเวลาที่ใช้รับและส่งข้อมูล) | 6 µsec | 73 µsec |
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 42 MHz | 56 MHz |
| จำนวนการรับข้อมูลแบบจำนวนเชิงซ้อน | 256 จุด | 256 จุด |

และเมื่อลดความละเอียดในการรับข้อมูลลงเหลือ 16 บิต สามารถนำไปเปรียบเทียบกับผลการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วการคำนวณวิธีอื่น ได้ดังตารางที่ 5.14

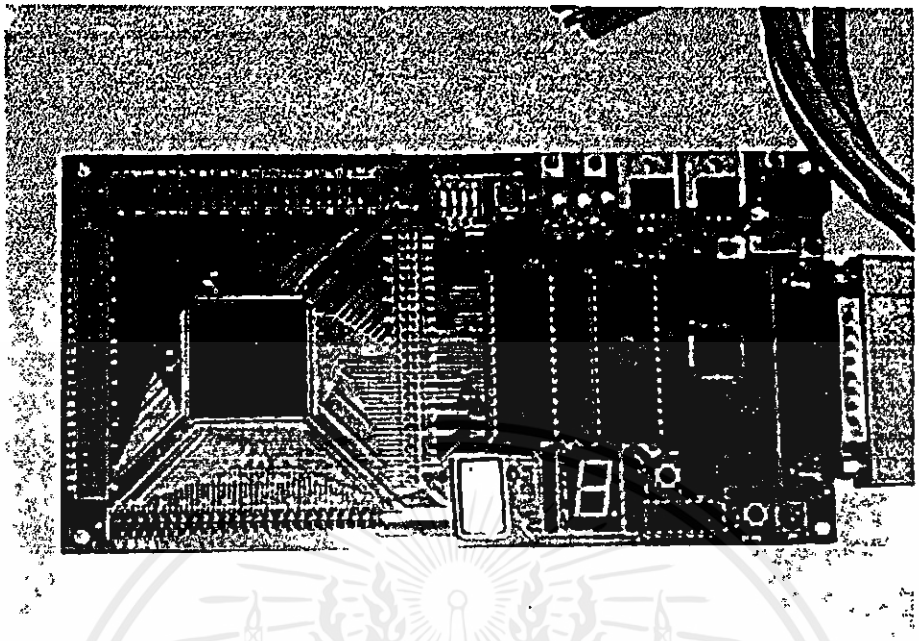
**ตารางที่ 5.14** เปรียบเทียบผลการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ปรับปรุง กับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้วิธีการคำนวณแบบอื่น ที่มีการรับข้อมูลขนาด 16 บิต

| ค่าต่างๆที่นำมาพิจารณา | วิธีระบบฐานผสม [17] | วิธีระบบฐาน 2 ที่ปรับปรุง |
|---|---|---|
| บิตของข้อมูลที่ใช้ในวงจร | 16 bits | 16 bits |
| จำนวนเกทที่ใช้งานทั้งหมด (ไม่รวมหน่วยความจำ) | 37,000 เกท | 14,869 เกท |
| เวลาที่ใช้ในการคำนวณ (ไม่รวมเวลาที่ใช้รับและส่งข้อมูล) | 6.6 µsec | 68 µsec |
| ความถี่สัญญาณนาฬิกาสูงสุด Maximum clock frequency ($f_{max}$) | 79 MHz | 60 MHz |
| จำนวนการรับข้อมูลแบบจำนวนเชิงซ้อน | 256 จุด | 256 จุด |

จากตารางที่ 5.13-5.14 พบว่า ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ววิธีระบบฐาน 2 ที่ปรับปรุงโดยการลดขนาดส่วนประมวลผลของแผนภาพผีเสื้อ จะมีขนาดเล็กเมื่อเทียบกับส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วที่ใช้วิธีการคำนวณแบบอื่น แต่จะใช้เวลาในการคำนวณที่มากกว่าซึ่งเป็นผลมาจากลักษณะของโครงสร้างที่นำมาใช้งาน

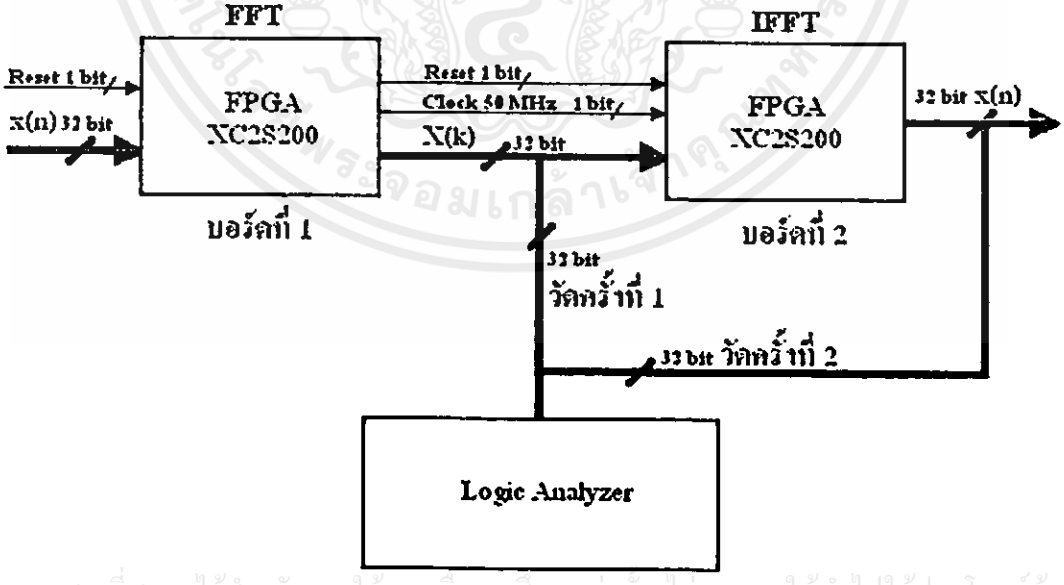
## 5.3 การทดสอบการทำงานบนบอร์ด FPGA

เมื่อทดสอบการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วบนเครื่องคอมพิวเตอร์ ว่าสามารถทำงานได้ตรงตามที่ออกแบบไว้แล้ว ก็จะนำส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่ออกแบบ มาโปรแกรมการทำงานลงบนบอร์ด FPGA และวัดผลการทำงานโดยใช้เครื่องวัดสัญญาณดิจิตอล โดยบอร์ด FPGA ที่นำมาใช้งานแสดงในรูปที่ 5.22



**รูปที่ 5.22** แสดงบอร์ด Xilinx Spartan-II FPGA เบอร์ XC2S200 ที่นำมาใช้ในการทดสอบ

และสามารถแสดงขั้นตอนการวัดผลการทำงานโดยใช้เครื่องวัดสัญญาณดิจิตอลได้ ดังรูปที่ 5.2

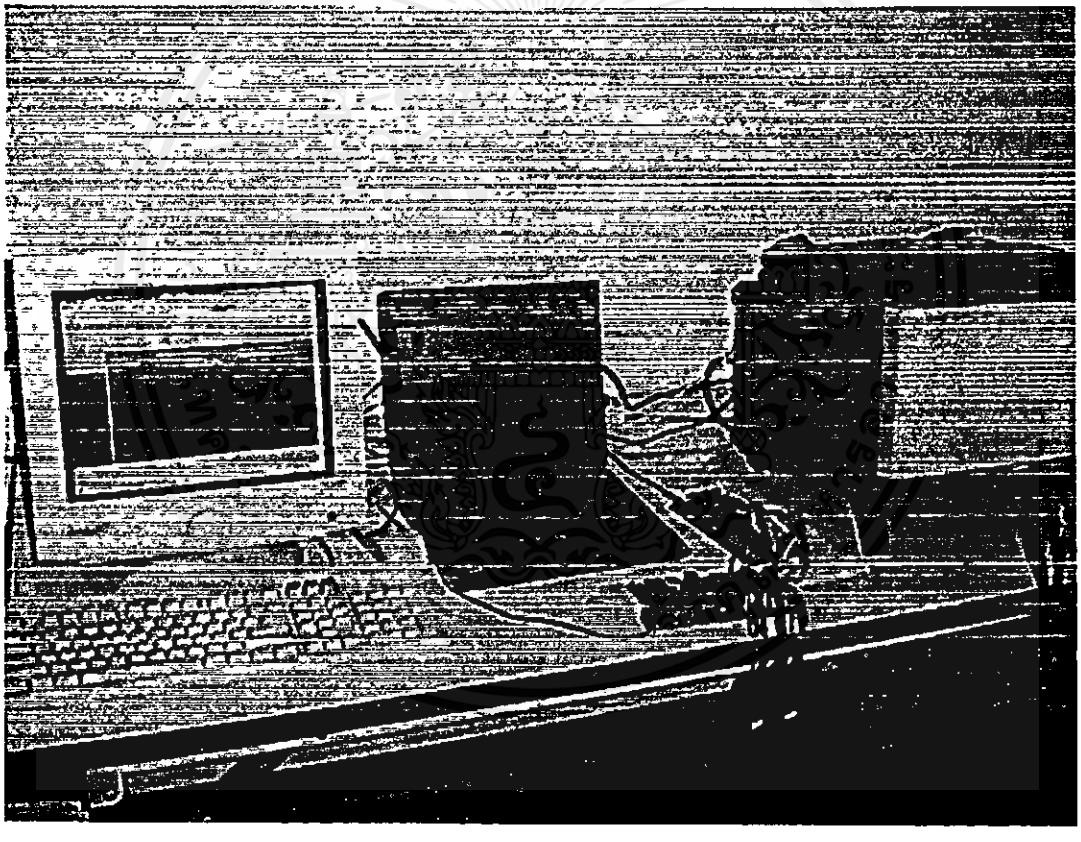




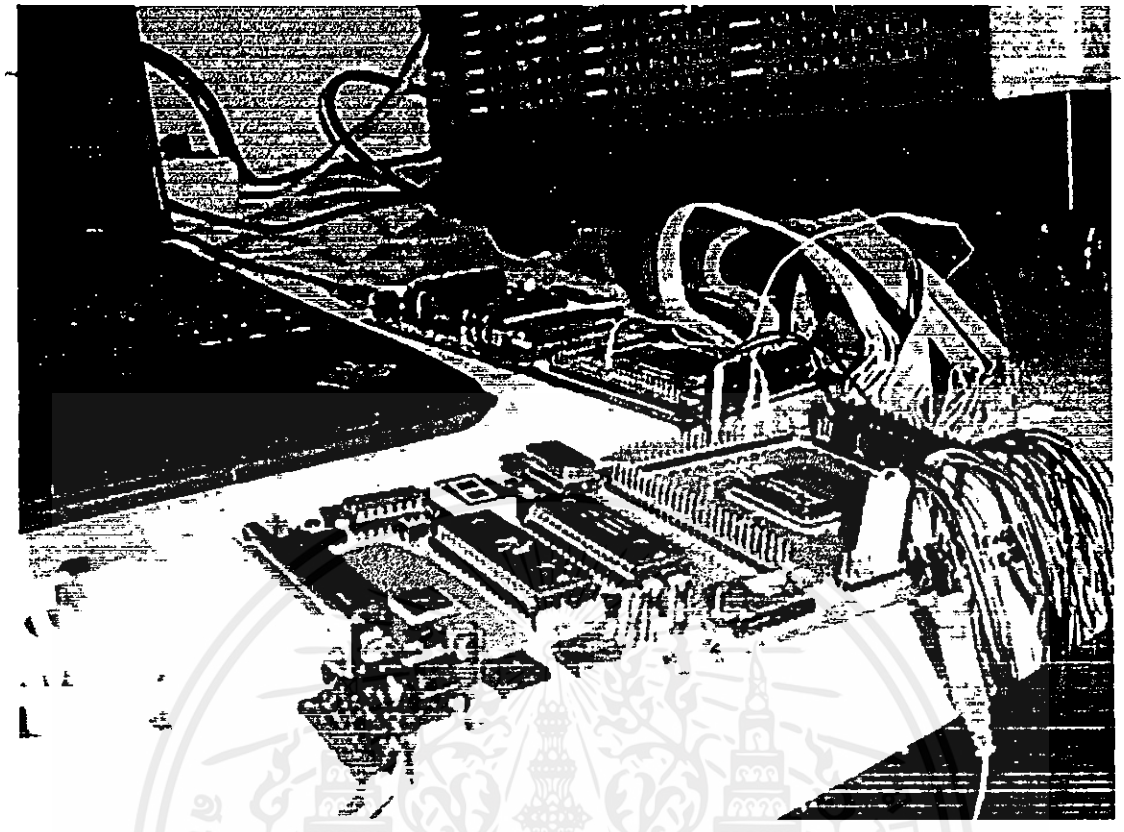

**รูปที่ 5.23** แสดงขั้นตอนการวัดผลการทำงานบนบอร์ด FPGA

จากรูปที่ 5.23 ขั้นตอนการวัดผลการทำงานคือ เริ่มต้นจากป้อนข้อมูล x(n) เข้าไปในบอร์ด

FPGA บอร์ดที่ 1 ที่โปรแกรมการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วเอาไว้เพื่อทดสอบการทำงาน จากนั้นใช้เครื่องวัดสัญญาณดิจิตอลวัดผล (การวัดครั้งที่ 1) ที่ได้จากการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว แสดงในรูปที่ 5.26 ซึ่งผลที่ได้จากการทำงานจะออกมาเป็นข้อมูล X(k) จากนั้นจะนำข้อมูล X(k) ที่ได้ป้อนเข้าไปในบอร์ด FPGA บอร์ดที่ 2 ที่โปรแกรมการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วเอาไว้เพื่อทดสอบการทำงาน จากนั้นใช้เครื่องวัดสัญญาณดิจิตอลวัดผล (การวัดครั้งที่ 2) ที่ได้จากการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว แสดงในรูปที่ 5.27 ซึ่งผลที่ได้จากการทำงานจะออกมาเป็นข้อมูล x(n) การวัดผลการทำงานบนบอร์ด FPGA ด้วยเครื่องวัดสัญญาณดิจิตอล แสดงในรูปที่ 5.24-5.25

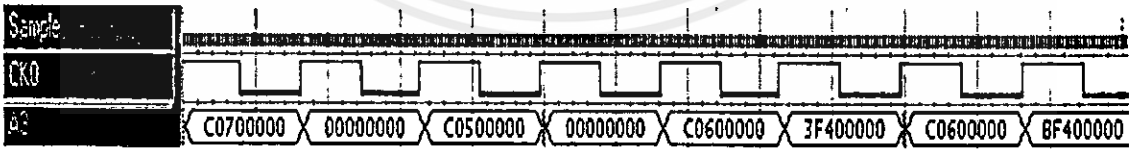


**รูปที่ 5.24** แสดงการวัดผลการทำงานบนบอร์ด FPGA ด้วยเครื่องวัดสัญญาณดิจิตอล (Logic Analyzer)



**รูปที่ 5.25** แสดงการวัดผลการทำงานบนบอร์ด FPGA ด้วยเครื่องวัดสัญญาณดิจิตอล (Logic Analyzer)

จากการทดสอบการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วบนบอร์ด FPGA สามารถวัดผลการทำงาน (การวัดครั้งที่ 1) โดยใช้เครื่องวัดสัญญาณดิจิตอล ได้ผลดังแสดงในรูปที่ 5.26



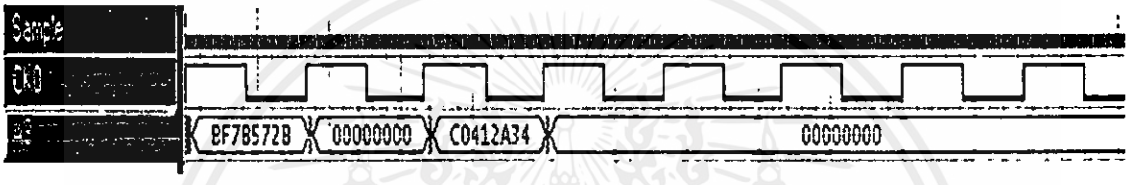

**รูปที่ 5.26** แสดงผลการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วบนบอร์ด FPGA ที่วัดด้วยเครื่องวัดสัญญาณดิจิตอล (Logic Analyzer)



จากรูปที่ 5.26 แสดงให้เห็นว่าผลจากการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็วบนบอร์ด FPGA ที่วัดด้วยเครื่องวัดสัญญาณดิจิตอล จะได้ผลการทำงานเหมือนกับผลที่

ได้จากการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ดังแสดงในรูปที่ 5.1

เมื่อทำการทดสอบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และตรวจสอบผลการทำงานที่ได้แล้ว จะนำผลลัพธ์ที่ได้ไปเป็นข้อมูลที่ส่งเข้าไปในส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว โดยผลลัพธ์ที่ได้จากส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วจะต้องมีค่าเท่ากับข้อมูลที่ส่งเข้าไปในส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว ซึ่งผลจากการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วบนบอร์ด FPGA ที่วัดได้ (การวัดครั้งที่ 2) จากเครื่องวัดสัญญาณดิจิตอล แสดงในรูปที่ 5.27




**รูปที่ 5.27** แสดงผลการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วบนบอร์ด FPGA ที่วัดด้วยเครื่องวัดสัญญาณดิจิตอล (Logic Analyzer)

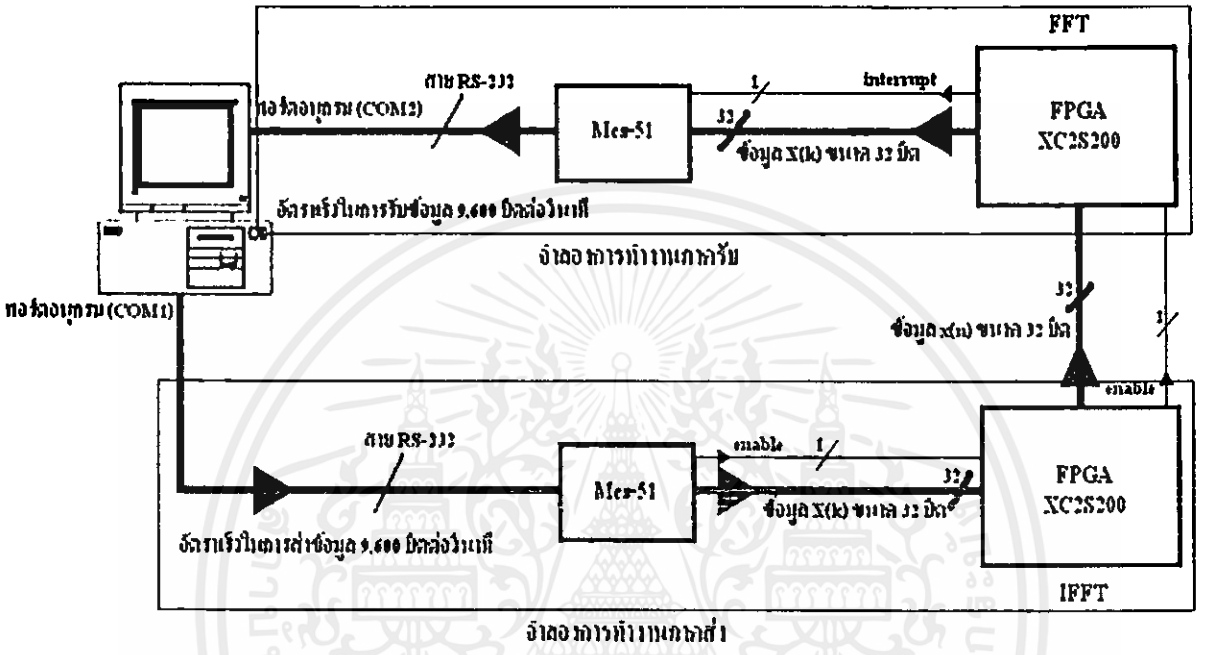
จากรูปที่ 5.27 แสดงให้เห็นว่าผลจากการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วบนบอร์ด FPGA ที่วัดด้วยเครื่องวัดสัญญาณดิจิตอล จะได้ผลการทำงานเหมือนกับผลที่ได้จากการจำลองการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ดังแสดงในรูปที่ 5.2

จากการทดสอบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วบนบอร์ด FPGA และวัดผลการทำงานด้วยเครื่องวัดสัญญาณดิจิตอล พบว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วสามารถทำงานได้ตามที่ออกแบบไว้

## 5.4 การทดสอบการประยุกต์ใช้งาน (จำลองการทำงานของภาครับและภาคส่งข้อมูล)

ส่วนประมวลผลการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วนั้น สามารถนำไปประยุกต์ใช้งานที่ภาคส่งสัญญาณ และภาครับสัญญาณได้ตามที่แสดงไว้ในรูปที่ 2.6 คือ ที่ภาคส่งสัญญาณส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วจะทำหน้าที่แปลงสัญญาณในเชิงความถี่ที่ได้รับมาจากการทำมอดูเลต (Modulate) มาเป็นสัญญาณในเชิงเวลา ก่อนที่จะนำไปแทรกช่วงเวลาป้องกันแล้วส่งสัญญาณออกไป ส่วนการทำงานที่ภาครับจะใช้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว มาทำการแปลงสัญญาณในเชิง

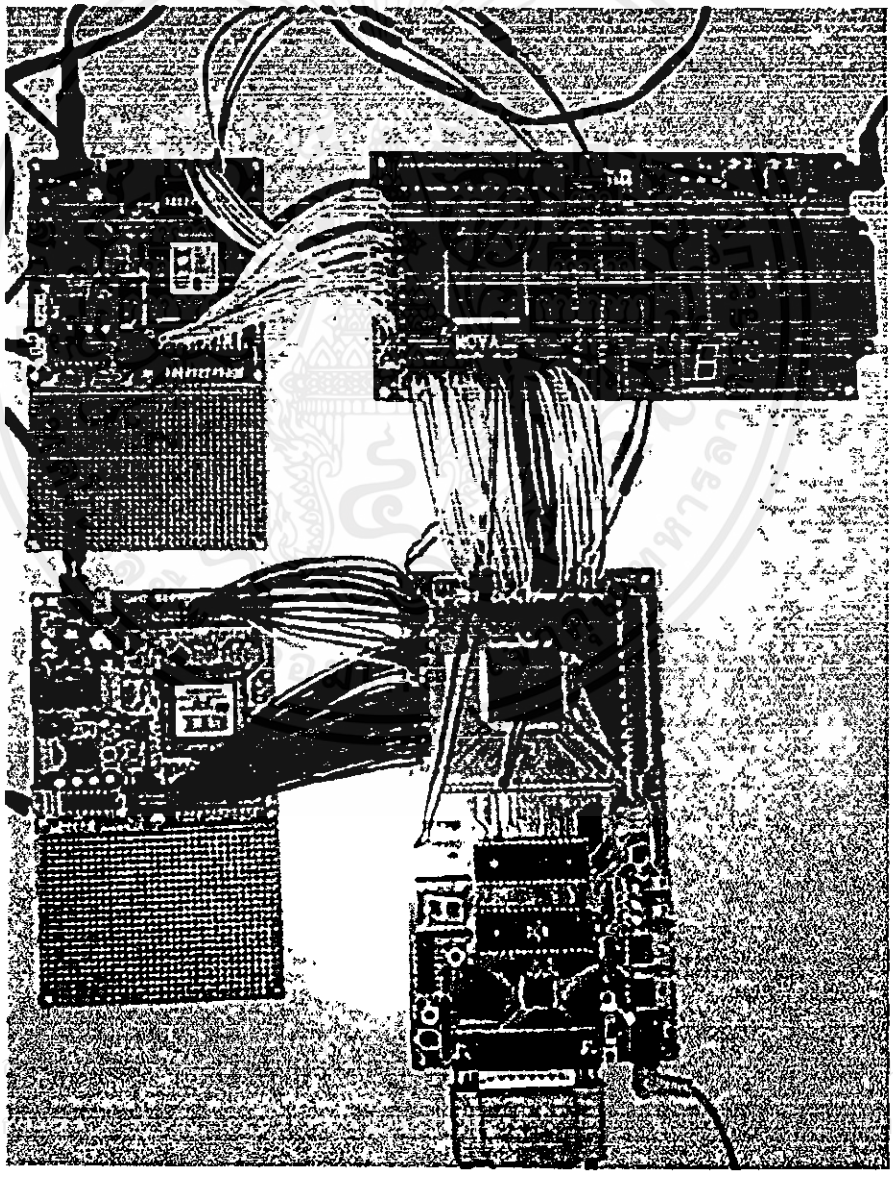
เวลาที่รับเข้ามาให้เป็นสัญญาณในเชิงความถี่ ก่อนที่จะส่งไปทำการดีมอดูเลต (Demodulate) ดังนั้น การทดสอบการประยุกต์ใช้งานของส่วนประมวลผลการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว จึงจำลองการทำงานมาจากภาคส่งและภาครับข้อมูล จะมีลักษณะดังรูปที่ 5.28




รูปที่ 5.28 แสดงแผนผังการทดสอบการประยุกต์ใช้งานที่ภาคส่งและภาครับ

จากรูปที่ 5.28 สามารถอธิบายการทดสอบการทำงานได้คือ เครื่องคอมพิวเตอร์จะส่งสัญญาณที่ได้จากการมอดูเลตแบบ QAM (Quadrature Amplitude Modulation) โดยใช้โปรแกรม Matlab ซึ่งทำการมอดูเลตข้อมูลเข้ากับคลื่นพาห์ที่มีค่าถี่เท่ากับ 10 Hz และส่งออกไปทางพอร์ตอนุกรมที่ 1 (COM1) โดยสามารถส่งข้อมูลออกไปได้ครั้งละ 8 บิต ซึ่งจะส่งข้อมูลไปที่บอร์ด Mcs-51 โดยที่บอร์ด Mcs-51 จะรับข้อมูลครั้งละ 8 บิต เก็บไว้จนข้อมูลครบ 32 บิต ก็จะส่งข้อมูลออกไป 1 ครั้ง (ได้ข้อมูลเป็นเลขอิงดรรชนีขนาด 32 บิต) ให้กับบอร์ด FPGA บอร์ดที่ 1 ที่โปรแกรมการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว (IFFT) ไว้ เพื่อทำการแปลงสัญญาณในเชิงความถี่มาเป็นสัญญาณในเชิงเวลา จากนั้นส่งสัญญาณในเชิงเวลาที่ได้จากการแปลงไปให้บอร์ด FPGA บอร์ดที่ 2 (เปรียบเหมือนการทำงานที่ภาคส่งสัญญาณ) ซึ่งบอร์ด FPGA บอร์ดที่ 2 ที่โปรแกรมการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว (FFT) ก็จะรับสัญญาณในเชิงเวลาเข้ามาเพื่อทำการแปลงสัญญาณในเชิงเวลามาเป็นสัญญาณในเชิงความถี่ ก่อนที่จะส่งให้บอร์ด Mcs-51 ครั้งละ 32 บิต และบอร์ด Mcs-51 เมื่อรับข้อมูลขนาด 32 บิตเข้ามาก็จะแบ่งข้อมูลออกเป็น 4 ส่วนส่วนละ 8 บิต เพื่อที่จะส่งกลับไปให้คอมพิวเตอร์ครั้งละ 8 บิต ในการรับ

ข้อมูลขนาด 8 บิต จากบอร์ด Mcs-51 จะใช้โปรแกรม HyperTerminal เพื่อรับข้อมูลที่ส่งกลับเข้ามา โดยบอร์ด Mcs-51 จะส่งข้อมูลมาทางพอร์ตอนุกรมที่ 2 (COM2) เปรียบเหมือนการทำงานที่ภาครับ โปรแกรม HyperTerminal จะแสดงผลที่รับกลับเข้ามาในลักษณะของรหัส ASCII เพื่อต้องการทราบผลลัพธ์ ก็จะนำรหัส ASCII ที่ได้ไปเปิดตารางของรหัส ASCII (ภาคผนวก ค.) เพื่อเปลี่ยนข้อมูลกลับมาเป็นเลขฐาน 2 โดยรหัส ASCII 1 ตัวจะแทนข้อมูลขนาด 8 บิต ดังนั้นข้อมูลแบบเลขอิงดรรชนีขนาด 32 บิต จึงต้องใช้รหัส ASCII 4 ตัวในการแทนค่ากลับมา การจำลองการทำงานทั้งหมดจะทำงานที่ความถี่ 300 Hz เนื่องสัญญาณนาฬิกาที่ใช้เป็นสัญญาณนาฬิกาที่สร้างจากบอร์ด MCS-51 ซึ่งมีข้อจำกัดในการสร้างสัญญาณนาฬิกา แสดงการต่อพ่วงอุปกรณ์ที่ใช้ในการทดสอบในรูปที่ 5.29



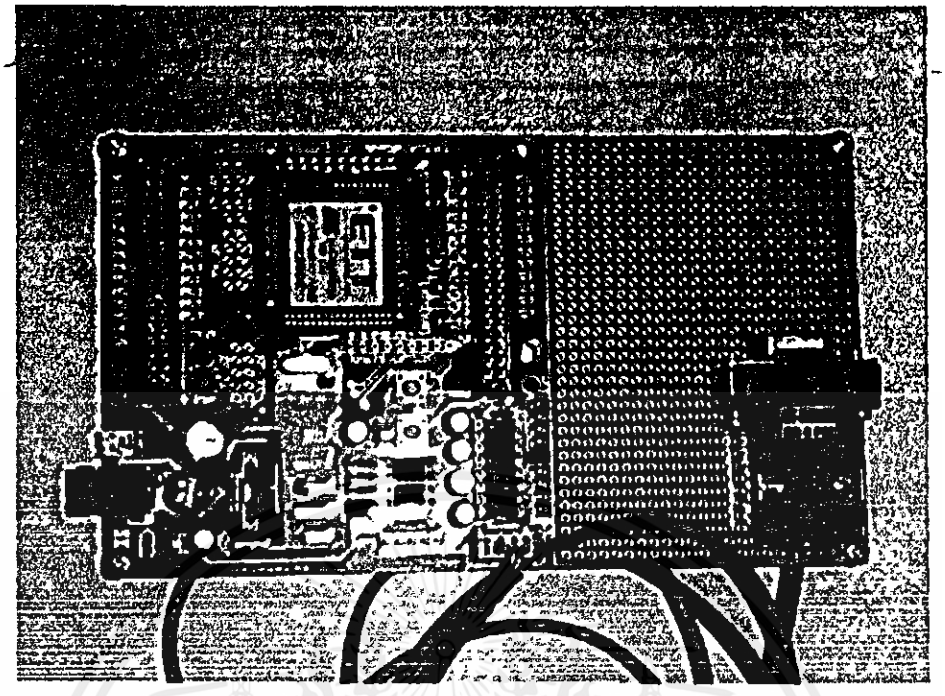
**รูปที่ 5.29** แสดงอุปกรณ์ที่ใช้ในการทดสอบการทำงาน

และแสดงการทดสอบการทำงาน ในรูปที่ 5.30

รูปที่ 5.30 แสดงการทดสอบการประยุกต์ใช้งานที่ภาคส่งและภาครับ

**5.4.1 ไมโครคอนโทรลเลอร์ MCS-51**

การนำไมโครคอนโทรลเลอร์มาใช้ในการทดสอบการทำงานของส่วนประมวลผลการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วนั้น จะนำไมโครคอนโทรลเลอร์มาใช้ในการพักข้อมูล และจัดเรียงข้อมูลระหว่างการทำงาน ซึ่งจะแบ่งการทำงานออกเป็น 2 ส่วนคือ การทำงานที่ภาคส่ง และการทำงานที่ภาครับ สามารถแสดงบอร์ดของไมโครคอนโทรลเลอร์ที่นำมาใช้งานได้ในรูปที่ 5.31



รูปที่ 5.31 แสดงบอร์ดของไมโครคอนโทรลเลอร์ MCS-51

ซึ่งไมโครคอนโทรลเลอร์ที่ภาครับและภาคส่ง ก็จะมีการออกแบบการทำงานที่ต่างกัน สามารถอธิบายการทำงานในแต่ละส่วนได้ดังนี้

ไมโครคอนโทรลเลอร์ที่ภาคส่งจะมีลักษณะการทำงาน คือ ไมโครคอนโทรลเลอร์จะรับข้อมูลที่ส่งมาจากเครื่องคอมพิวเตอร์ด้วยอัตราบอดเรต (Baud Rate) เท่ากับ 9,600 บิตต่อวินาทีผ่านทางพอร์ตอนุกรมมาพักไว้ก่อน เนื่องจากเครื่องคอมพิวเตอร์สามารถส่งข้อมูลผ่านพอร์ตอนุกรมได้ครั้งละ 8 บิต แต่ข้อมูลที่จะส่งให้บอร์ด FPGA จะมีขนาด 32 บิต ดังนั้นไมโครคอนโทลเลอร์ก็จะรับข้อมูลครั้งละ 8 บิต ทั้งหมด 4 ครั้งก่อนที่จะส่งข้อมูลแบบขนานขนาด 32 บิต ออกไปให้กับบอร์ด FPGA ซึ่งการออกแบบการทำงานของไมโครคอนโทรลเลอร์ที่ภาคส่ง (โปรแกรมแสดงในภาคผนวก จ.) จะมีลักษณะดังนี้

1. ตั้งค่าเริ่มต้นต่างๆ (Serial Port Interrupt, Serial Mode, Baud Rate, Timer)
2. รอการ Interrupt ของพอร์ตอนุกรมในการรับข้อมูล
3. เมื่อเกิด Interrupt จะนำค่าที่ได้ไปเก็บไว้ในบัฟเฟอร์ (Buffer) ข้อมูล
4. นับให้ครบ 4 ไบท์ แล้วส่งออกไปที่พอร์ต 0, 1, 2 และ 4 พร้อมทั้ง สร้างสัญญาณ ACK ที่เป็น Logic 0 ไปให้บอร์ด FPGA ด้วย แล้วใช้ Timer นับเวลาตามที่คำนวณแล้วเปลี่ยน ACK กลับเป็น 1
5. วนรับข้อมูลจากพอร์ตอนุกรมจนครบ 2,048 ไบท์ แล้วทำการส่ง ACK ออกไปให้ใช้เป็น Clock ของบอร์ด FPGA ตามการหน่วงเวลาของ Timer

ไมโครคอนโทรลเลอร์ที่ภาครับจะมีลักษณะการทำงานคือ ไมโครคอนโทรลเลอร์จะรับข้อมูลในแบบขนานขนาด 32 บิต จากบอร์ด FPGA มาพักไว้แล้วแบ่งข้อมูลขนาด 32 บิต ออกเป็น 4 ส่วน ส่วนละ 8 บิต เพื่อที่จะส่งข้อมูลขนาด 8 บิต กลับไปให้คอมพิวเตอร์ผ่านทางพอร์ตอนุกรมด้วยอัตราบอดเรต (Baud Rate) เท่ากับ 9,600 บิตต่อวินาทีทีละส่วนจนครบ ซึ่งการออกแบบการทำงานของไมโครคอนโทรลเลอร์ที่ภาครับ (โปรแกรมแสดงในภาคผนวก จ.) จะมีลักษณะดังนี้

1. ตั้งค่าเริ่มต้นต่างๆ (Serial Port Interrupt, External Interrupt, Serial Mode, Baud Rate, Timer)
2. รอสัญญาณ External Interrupt ที่มาจากบอร์ด FPGA
3. รับข้อมูลแบบขนานเข้าทางพอร์ต 0,1,2 และ4 แล้วนำไปเก็บไว้ในบัฟเฟอร์ (Buffer) ข้อมูล
4. นำข้อมูลที่เก็บไว้ส่งออกไปที่เครื่องคอมพิวเตอร์ผ่านพอร์ตอนุกรมทีละ 1 ไบท์
5. กลับมารอข้อมูลใหม่

### 5.4.2 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมในเครื่องคอมพิวเตอร์นั้นจะมีความเร็วในการสื่อสารช้ากว่าแบบขนาน ทั้งนี้ก็เพราะว่าการเคลื่อนย้ายข้อมูลแบบอนุกรมนั้นเป็นการส่งข้อมูลครั้งละ 1 บิต แต่พอร์ตขนานนั้นสามารถส่งข้อมูลได้ครั้งละหลายๆ บิตพร้อมกันส่งผลให้การสื่อสารข้อมูลแบบอนุกรมมีความเร็วต่ำกว่าแบบขนาน

แต่ว่าการส่งข้อมูลแบบอนุกรมนั้นมีข้อที่เหนือกว่าการส่งข้อมูลแบบขนานคือ สามารถส่งข้อมูลได้ในระยะทางที่ไกลกว่าแบบขนาน อีกทั้งสายสัญญาณที่ใช้ยังมีน้อยกว่าการส่งข้อมูลแบบขนานอีกด้วย สำหรับมาตรฐานของการส่งข้อมูลแบบอนุกรมที่ได้รับความนิยมอย่างสูงตั้งแต่อดีตถึงปัจจุบัน โดยใช้งานกันอย่างแพร่หลายทั้งการสื่อสาร และการควบคุมทางอุตสาหกรรมนั่นก็คือมาตรฐาน RS-232

### 5.4.3 มาตรฐาน RS-232

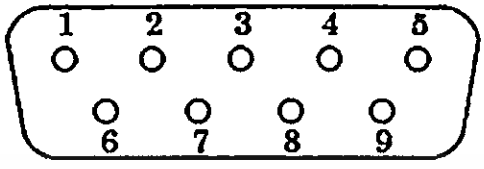
มาตรฐาน RS-232 เป็นมาตรฐานที่ได้รับการออกแบบมาเพื่อที่จะทำให้อุปกรณ์ต่อพ่วงจากผู้ผลิตต่างกันให้สามารถทำงานร่วมกันได้ มาตรฐาน RS-232 ได้ถูกประกาศใช้ในปี ค.ศ. 1969 โดยสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA)

มาตรฐาน RS-232 ได้แบ่งอุปกรณ์ออกเป็น 2 ประเภท ซึ่งอุปกรณ์ทั้งสองประเภทนี้ก็คือ

1. อุปกรณ์ DTE (Data Terminal Equipment) เป็นอุปกรณ์สำหรับส่งข้อมูล
2. อุปกรณ์ DCE (Data Communication Equipment) เป็นอุปกรณ์สำหรับรับข้อมูล

ตามมาตรฐาน RS-232 แล้ว คอนเน็กเตอร์ (connector) ที่นิยมใช้จะเป็นชนิด D-Type แบบ

9 ขา และแบบ 25 ขา ซึ่งบางครั้งจะเรียกว่า DB9 และ DB25 โดยที่หัวต่อทั้งสองชนิดจะมีลักษณะการทำงานของสัญญาณที่เหมือนกัน แต่จะมีการจัดเรียงไม่เหมือนกัน สามารถแสดงรายละเอียดของคอนเน็กเตอร์ (connector) ชนิด D-Type แบบ 9 ขา ซึ่งเป็นชนิดที่นำมาใช้งานได้ดังรูปที่ 5.32




**รูปที่ 5.32** แสดงแผนผังคอนเน็กเตอร์ (connector) ของ RS-232 ชนิด D-Type แบบ 9 ขา

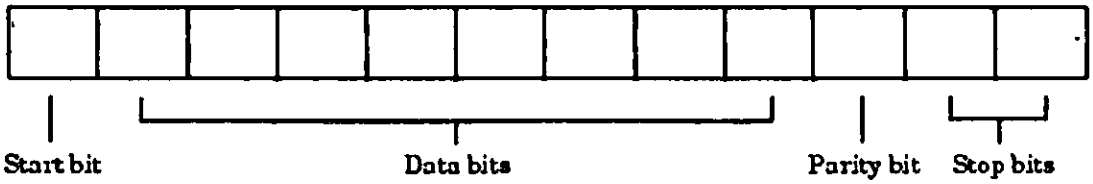
จากรูปที่ 5.32 สามารถแสดงรายละเอียดของสายสัญญาณแต่ละสายได้ตามตารางที่ 5.15

**ตารางที่ 5.15** แสดงรายละเอียดของสายสัญญาณ

| D-Type 9 ขา | สัญลักษณ์ | ชื่อสัญญาณ | รูปแบบสัญญาณ |
|---|---|---|---|
| ขา 1 | CD | Carrier Detect | Control |
| ขา 2 | RD | Receive Data | Data |
| ขา 3 | TD | Transmit Data | Data |
| ขา 4 | DTR | Data Terminal Ready | Control |
| ขา 5 | SG | Signal Ground | Ground |
| ขา 6 | DSR | Data Set Ready | Control |
| ขา 7 | RTS | Request To Send | Control |
| ขา 8 | CTS | Clear To Send | Control |
| ขา 9 | RI | Ring Indicator | Control |

**5.4.4 องค์ประกอบของการรับส่งข้อมูลแบบอนุกรม**

การสื่อสารแบบอนุกรมที่นิยมใช้กับคอมพิวเตอร์นั้น เป็นการสื่อสารแบบอะซิงโครนัส (Asynchronous) นั่นคือ ต้องใช้สายสัญญาณเส้นเดียวทำหน้าที่ทั้งส่งส่วนที่เป็นข้อมูล และส่วนที่ใช้ควบคุมการส่งข้อมูล ดังนั้นข้อมูลที่อ่านได้แต่ละบิตจากการส่งแบบอนุกรม จึงถูกแยกแยะว่าใช้สำหรับวัตถุประสงค์ใด โดยสามารถแบ่งได้เป็น 4 ส่วนดังแสดงในรูปที่ 5.33




**รูปที่ 5.33** แสดงการแบ่งส่วนของข้อมูลในการส่ง

จากรูปที่ 5.33 แต่ละข้อมูลที่ถูกส่งออกไปเป็นกลุ่ม ซึ่งจะประกอบไปด้วยบิตเริ่มต้น (Start bit) บิตข้อมูล (Data bit) บิตพาริตี้ (Parity bit) ซึ่งจะมีหรือไม่มีก็ได้ และบิตจบ (Stop bit) โดยสามารถสรุปหน้าที่ของแต่ละส่วนได้ดังนี้

1. บิตเริ่มต้น (มีขนาด 1 บิต) จะใส่ที่จุดเริ่มต้นเสมอ เพื่อเตือนอุปกรณ์ฝ่ายรับว่าข้อมูลกำลังจะมาถึง
2. บิตข้อมูล (มีขนาด 7 หรือ 8 บิต) การส่งบิตข้อมูลจะส่งเป็นกลุ่มๆ โดยทั่วไปจะส่งเป็น 7 หรือ 8 บิต ซึ่งเพียงพอสำหรับการส่งค่าแบบรหัส ASCII
3. บิตพาริตี้ (มีขนาด 1 บิต) ใช้ในการตรวจสอบความถูกต้องของข้อมูลที่ส่ง โดยจะใส่บิตพาริตี้เข้าไป ซึ่งทั้งตัวรับและตัวส่งจะต้องรู้กันว่าใช้พาริตี้แบบไหนในการส่งข้อมูล ซึ่งหลักการในการกำหนดบิตพาริตี้มีหลายแบบดังนี้

- พาริตี้คู่ (Even Parity) ค่าของบิตพาริตี้นี้เมื่อรวมกับทุกๆ บิตของข้อมูลแล้วจะต้องมีจำนวนบิตที่เป็นเลข 1 เป็นเลขคู่ ตัวอย่างเช่น ข้อมูล 1000101 มีเลข 1 ทั้งหมด 3 ตัว ดังนั้นบิตพาริตี้จะเป็น 0

- พาริตี้คี่ (Odd Parity) ค่าของบิตพาริตี้นี้เมื่อรวมกับทุกๆ บิตของข้อมูลแล้วจะต้องมีจำนวนบิตที่เป็นเลข 1 เป็นเลขคี่ ตัวอย่างเช่น ข้อมูล 1000101 มีเลข 1 ทั้งหมด 3 ตัว ดังนั้นบิตพาริตี้จะเป็น 1

- ไม่มีพาริตี้ (None) ถ้าตั้งบิตพาริตี้เป็น None ทั้งภาครับและภาคส่งจะไม่มีการตรวจสอบบิตพาริตี้

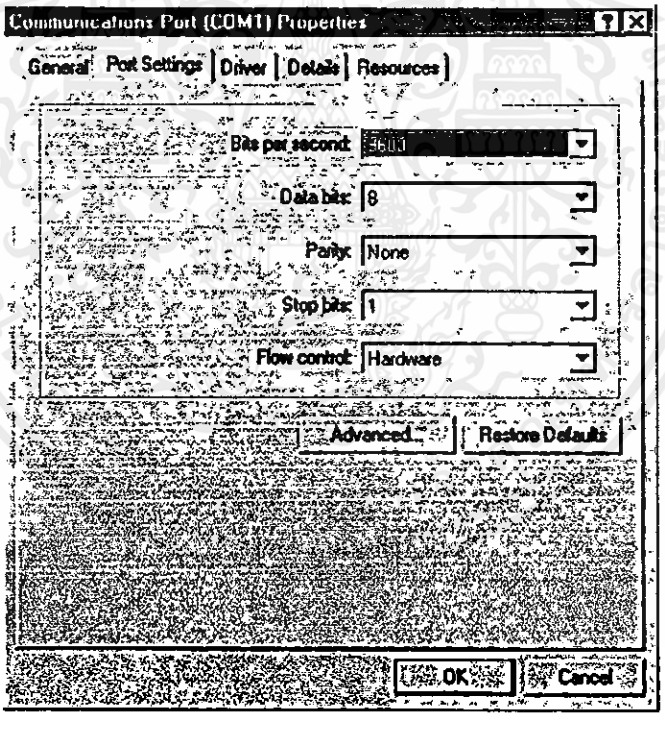
4. หรือบิตจบ (มีขนาด 1 หรือ 2 บิต) เป็นบิตที่ส่งมาปิดท้ายข้อมูล

### 5.4.5 อัตราเร็วในการรับส่งข้อมูลแบบอนุกรม (Baud Rate)

การที่อุปกรณ์ 2 อย่างจะติดต่อสื่อสารกันได้นั้น จะต้องทำงานด้วยอัตราเร็วเท่ากัน ซึ่งอัตราเร็วในการสื่อสารแบบอะซิงโครนัส (Asynchronous) คือ ค่าบอดเรต (Baud Rate) มีหน่วยเป็นบิตต่อวินาที ซึ่งค่าอัตราเร็วในการสื่อสารแบบอนุกรมสำหรับมาตรฐาน RS-232 นั้นมีใช้ดังนี้ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที

**5.4.6 การออกแบบการทำงานของโปรแกรม Matlab**

การออกแบบการทำงานของโปรแกรม Matlab คือ จะนำฟังก์ชั่นสำหรับการทำการมอดูเลตแบบ QAM (Quadrature Amplitude Modulation) มาใช้ในการทำการมอดูเลตข้อมูล ก่อนที่จะส่งผลที่ได้จากการมอดูเลตไปให้กับส่วนประมวลผลการแปลงฟูริเยร์แบบรวดเร็ว ซึ่งผลที่ได้จากการมอดูเลตแบบ QAM ในโปรแกรม Matlab จะได้ผลออกมาในรูปของเลขฐาน 10 จากนั้นจะนำผลที่ได้ในเลขฐาน 10 มาแปลงเป็นเลขอิงดรรชนีขนาด 32 บิต โดยใช้ฟังก์ชั่นการทำงานที่ออกแบบในลักษณะของ m-file จากนั้นข้อมูลขนาด 32 บิตจะถูกแบ่งออกเป็น 4 ส่วนส่วนละ 8 บิต (เนื่องจากว่าการส่งข้อมูลผ่านพอร์ตอนุกรมจะส่งข้อมูลได้ทีละ 8 บิต) จากนั้นจะนำข้อมูล 8 บิต มาแปลงกลับให้อยู่ในเลขฐาน 10 เพื่อให้สามารถที่จะใช้คำสั่งภายในโปรแกรม Matlab ส่งข้อมูลขนาด 8 บิตออกไปได้ (แสดงโปรแกรมในภาคผนวก ข.) ซึ่งก่อนที่จะทำการส่งข้อมูลผ่านพอร์ตอนุกรมโดยใช้โปรแกรม Matlab จำเป็นที่จะต้องกำหนดค่าต่างๆ ให้กับพอร์ตอนุกรมก่อน โดยจะกำหนดค่าให้กับพอร์ตอนุกรมดังแสดงในรูปที่ 5.34




**รูปที่ 5.34** แสดงการกำหนดค่าให้กับพอร์ตอนุกรม



จากรูปที่ 5.34 จะเป็นการกำหนดค่าต่างๆ ให้กับพอร์ตอนุกรมซึ่งจะกำหนดอัตราเร็วในการรับส่งข้อมูลไว้ที่ 9600 บิตต่อวินาที บิตข้อมูลมีจำนวน 8 บิต ไม่มีการกำหนดค่าของบิตพาริตี้ และกำหนดให้ส่งข้อมูลไปยังฮาร์ดแวร์ ซึ่งเมื่อกำหนดพอร์ตอนุกรมในลักษณะเช่นนี้ ทำให้สามารถใช้คำสั่งของโปรแกรม Matlab ในการส่งข้อมูลผ่านพอร์ตอนุกรมได้ โดยมีขั้นตอนดังต่อไปนี้

1. สร้างพอร์ตอนุกรมภายในโปรแกรม Matlab ตามคำสั่งต่อไปนี้

   S = serial ('COM1'); เป็นการกำหนดให้ S คือพอร์ตอนุกรมที่ 1 (COM1)

2. เขียนคำสั่งสำหรับติดต่อกับฮาร์ดแวร์

   fopen(S) เป็นการเปิดพอร์ตอนุกรม S ให้ติดต่อกับฮาร์ดแวร์ภายนอก

3. เขียนคำสั่งสำหรับส่งข้อมูลออกพอร์ตอนุกรม

   fwrite(S,[ข้อมูล]); เป็นคำสั่งในการส่งข้อมูล โดยข้อมูลจะถูกส่งออกไปในลักษณะของเลขฐาน 2 ขนาด 8 บิต

4. เมื่อส่งข้อมูลเสร็จก็จะสั่งยกเลิกการติดต่อและยกเลิกการใช้พอร์ตอนุกรมภายในโปรแกรม Matlab ด้วยคำสั่งต่อไปนี้

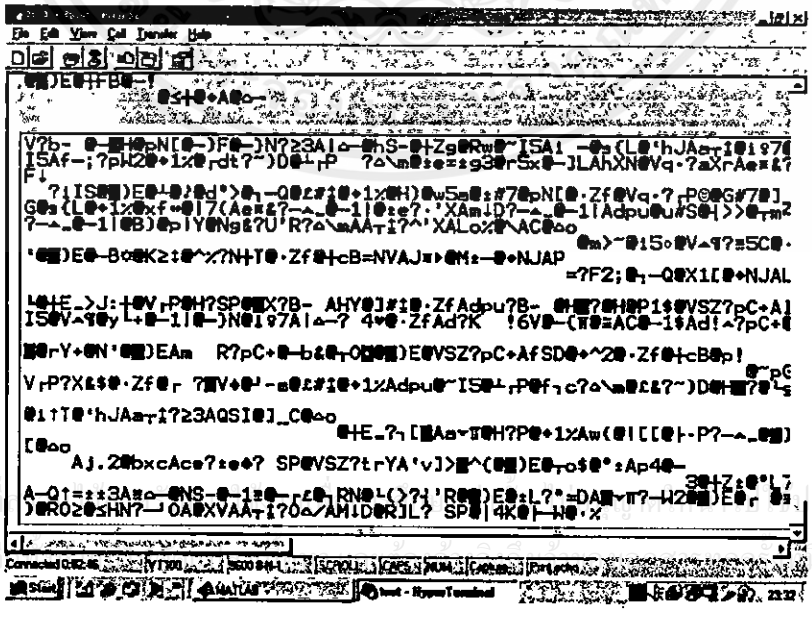
   fclose(S) เป็นการปิดพอร์ตอนุกรม S

   delete(S) เป็นการลบพอร์ตอนุกรม S

   clear S ; เป็นการยกเลิกตัวแปร S

### 5.4.7 การใช้โปรแกรม HyperTerminal

โปรแกรม HyperTerminal เป็นโปรแกรมสำเร็จรูปที่มาพร้อมกับระบบ Windows ใช้สำหรับการติดต่อสื่อสารผ่านพอร์ตอนุกรม ซึ่งการใช้โปรแกรม HyperTerminal เพื่อรับข้อมูลผ่านพอร์ตอนุกรมนั้น ข้อมูลต่างๆ ที่รับเข้ามาจะแสดงค่าออกมาในลักษณะของรหัส ASCII แสดงผลที่ได้จากการทำสอบการทำงานโดยการรับข้อมูลผ่านโปรแกรม HyperTerminal ในรูปที่ 5.35



**รูปที่ 5.35** แสดงข้อมูลที่ได้จากการทดสอบรับเข้ามาผ่านทางโปรแกรม HyperTerminal

จากรูปที่ 5.35 เมื่อ HyperTerminal รับข้อมูลเข้ามาแล้วแสดงออกมาในรหัส ASCII ทำให้ต้องนำข้อมูลที่ได้ในรหัส ASCII ไปเปิดตารางเทียบกลับมาเป็นค่าของเลขฐาน 2 โดยที่ค่าในรหัส ASCII 1 ตัวจะแทนข้อมูลในเลขฐาน 2 จำนวน 8 บิต ดังนั้นค่าของข้อมูลในลักษณะของเลขอิงดรรชนี 1 ข้อมูล จะแทนด้วยรหัส ASCII 4 ตัว โดยค่าในรหัส ASCII ทั้งหมดสามารถแปลงเป็นค่าในเลขฐาน 2 ได้ตาม ภาคผนวก ค.

## 5.5 แสดงผลจากการทดสอบเทียบกับผลจากโปรแกรม Matlab

จากการทดสอบการทำงานของส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วโดยการประยุกต์ใช้งานที่ภาคส่ง และภาครับสามารถแสดงตัวอย่างของผลที่ได้จากการทดสอบการทำงานเปรียบเทียบกับผลที่ได้จากการคำนวณโดยใช้โปรแกรม Matlab แสดงในตารางที่ 5.16 ผลทั้งหมดแสดงใน (ภาคผนวก ฉ.)

**ตารางที่ 5.16** แสดงผลจากการทดสอบการทำงานเทียบกับผลจากโปรแกรม Matlab

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 1 | 01000001011001100001010001111011 | 01000001010011001100110011001101 |
| | 11000010000001001110011101010010 | 11000010000001001100110011001101 |
| 2 | 01000001101000111101011100001010 | 01000001100110000000000000000000 |
| | 01000010010100100101000111101100 | 01000010010011010011001100110011 |
| 3 | 11000001111001001000110100010110 | 11000001111001001000111101011100 |
| | 01000000101001111110011011100000 | 01000000101001111010111000010100 |
| 4 | 00111111110100010000010110000000 | 00111111110100011110101110000101 |
| | 01000010000110010011000010111010 | 01000010000110010011001100110011 |
| 5 | 11000000100110001110001011101011 | 11000000100110001110011100000100 |
| | 11000000100110011001100110011010 | 11000000100110011011010010100010 |
| 6 | 01000001000000000011000100100111 | 01000001000000000000000000000000 |
| | 01000000001101001101110100101111 | 01000000001101010001000110011101 |
| 7 | 10111101100111111011111001110111 | 10111101100110110111000101110110 |
| | 11000000110111111001010110000001 | 11000000110111110111110011101110 |
| 8 | 11000000001001011100001010001111 | 11000000001001011110011010011011 |
| | 01000000010000000111010111110111 | 01000000010000000000000000000000 |

### 5.5.1 สรุปผลการทดสอบ

จากตารางที่ 5.16 พบว่าผลที่ได้จากการทดสอบการทำงานมีความผิดพลาดเกิดขึ้น และเมื่อนำผลที่ได้จากการทดสอบการทำงานมาเปรียบเทียบกับผลที่ได้จากโปรแกรม Matlab แล้วพบว่าผลที่ได้จากการทำสอบการทำงานจะมีความผิดพลาดเฉลี่ยที่ 15 บิต ของข้อมูลทางด้าน LSB

ซึ่งข้อผิดพลาดที่เกิดขึ้นนี้เป็นผลมาจากการทำงานภายในส่วนประมวลผลการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว โดยเป็นข้อผิดพลาดที่เกิดมาจากวงจรคูณ เนื่องจากว่าในการคูณเลขอิงดรรชนีขนาด 32 บิต เข้าด้วยกันการทำงานก็คือ จะนำค่า exponent มาบวกกันและนำค่า mantissa มาคูณกัน ในส่วนของค่า exponent จะไม่มีปัญหาอะไร แต่ในส่วนของค่า mantissa จะมีปัญหาคือ จากเลขอิงดรรชนีขนาด 32 บิต จะมีค่า mantissa ทั้งหมด 23 บิต ซึ่งเมื่อนำค่า mantissa ขนาด 23 บิต สองจำนวนมาคูณกันแล้วจะได้ผลลัพธ์ออกมาเป็นค่า mantissa ขนาด 46 บิต แต่เวลาส่งผลลัพธ์ที่ได้จากการคูณออกไปจำเป็นต้องตัดค่า mantissa ส่วนที่เกินจาก 23 บิต ทิ้งไปส่งผลให้มีความผิดพลาดจากการทำงานเกิดขึ้น และจากการทดสอบการทำงานของวงจรคูณเทียบกับการคูณโดยใช้โปรแกรม Matlab พบว่าผลที่ได้จากการทำงานมีความผิดพลาดที่ค่า mantissa มากที่สุด 8 บิต ของข้อมูลทางด้าน LSB ซึ่งจะเกิดขึ้นในกรณีที่ผลที่ได้จากการคูณเป็นค่าทศนิยมแบบไม่รู้จบ

## 5.6 สรุป

จากการจำลองการทำงานบนเครื่องคอมพิวเตอร์แสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว สามารถทำงานได้ตรงตามที่ออกแบบ และจากการทดสอบการทำงานแสดงให้เห็นว่าส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว สามารถทำงานบนบอร์ด FPGA ได้ และเมื่อวัดผลการทำงานด้วยเครื่องวัดสัญญาณดิจิตอล ก็พบว่าผลที่วัดได้จะมีผลการทำงานเหมือนกับผลที่ได้จากการจำลองการทำงาน จากนั้นเป็นการนำไปทดสอบการประยุกต์ใช้งานที่ภาคส่งและภาครับข้อมูล ซึ่งผลที่ได้เมื่อนำมาเปรียบเทียบกับผลที่ได้จากโปรแกรม Matlab แล้วพบว่ามีความผิดพลาดเกิดขึ้นทางด้าน LSB ของข้อมูล และเมื่อทำการตรวจสอบการทำงานแล้วพบว่าเป็นข้อผิดพลาดที่เกิดขึ้นมาจากวงจรคูณ

# บทที่ 6
# บทสรุป

## 6.1 สรุปผลการวิจัย

วิทยานิพนธ์นี้ได้นำเสนอการออกแบบส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วด้วยภาษา VHDL ให้มีขนาดเล็ก สำหรับการนำไปประยุกต์ใช้ในการสื่อสาร ซึ่งแสดงการทดสอบการทำงานไว้ในวิทยานิพนธ์นี้ โดยการออกแบบจะเลือกใช้วิธีการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว วิธีระบบฐาน 2 แบบแตกเป็นส่วนย่อยทางฝั่งเวลา เนื่องจากตัวแปรต่างๆ มีจำนวนน้อย ทำให้ง่ายต่อการนำมาออกแบบ และปรับปรุงให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วมีขนาดเล็ก เมื่อเทียบกับวิธีการคำนวณการแปลงฟูริเยร์แบบรวดเร็ววิธีอื่นๆ ที่มีการนำมาประยุกต์ใช้ในการสื่อสาร ดังรายละเอียดในบทที่ 2

บทที่ 3 เป็นการออกแบบวงจรคำนวณทางคณิตศาสตร์ที่ต้องนำมาใช้งาน เช่น วงจรบวก วงจรลบ วงจรคูณ และวงจรหาร ซึ่งวงจรทั้งหมดนี้เป็นการออกแบบมาเพื่อรองรับการทำงานของระบบเลขอิงดรรชนีแบบ 32 บิต เพื่อแก้ไขข้อจำกัดในการรับข้อมูลที่มีขนาดใหญ่หรือข้อมูลที่มีขนาดเล็กมากๆ เข้ามาคำนวณ

บทที่ 4 เป็นการออกแบบ และปรับปรุงส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วให้มีขนาดเล็ก โดยการลดโครงสร้างการทำงานภายในส่วนประมวลผลของแผนภาพผีเสื้อลง ซึ่งสามารถลดโครงสร้างการทำงานให้เหลือเพียง 2 วงจรคูณ และ 2 วงจรบวก ทำให้ส่วนประมวลผลของแผนภาพผีเสื้อมีขนาดเล็กลงเมื่อเทียบกับการออกแบบโดยใช้โครงสร้างการทำงานเดิม ส่งผลให้ส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วที่ออกแบบมีขนาดเล็กลงตามไปด้วย ซึ่งได้ถูกนำเสนอเป็นครั้งแรกในวิทยานิพนธ์นี้

สำหรับบทที่ 5 เป็นการนำเอาส่วนประมวลผลของการแปลงฟูริเยร์แบบรวดเร็ว และส่วนประมวลผลของการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว มาจำลองการทำงาน และทดสอบการทำงานบนบอร์ด FPGA ว่าสามารถทำงานได้ตามที่ออกแบบไว้หรือไม่ รวมทั้งการนำไปทดสอบการประยุกต์ใช้งานโดยจำลองการทำงานที่ภาคส่งและภาครับมาใช้ในการทดสอบ



## 6.2 ข้อเสนอแนะและแนวทางการทำวิจัยเพื่อพัฒนาต่อ

การแปลงฟูริเยร์แบบรวดเร็ว และการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ได้มีการนำไปประยุกต์ใช้งานที่หลากหลาย รวมทั้งการสื่อสาร โดยที่ความต้องการส่วนใหญ่คือ ต้องการส่วน

ประมวลผลของการแปลงฟูริเยร์ที่มีขนาดเล็ก เพื่อที่จะให้ระบบมีขนาดเล็ก และประหยัดพลังงาน [23-24] เมื่อนำไปใช้งาน

ดังนั้นแนวการทำงานวิจัยเพื่อทำการพัฒนาต่อไปคือ เนื่องจากวิธีการคำนวณการแปลงฟูริเยร์แบบรวดเร็ว และการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็วในวิธีระบบฐาน รูปแบบต่างๆ [25-27] นั้นมีลักษณะการทำงานใกล้เคียงกับวิธีระบบฐาน 2 ดังนั้นโครงสร้างของแผนภาพผีเสื้อที่นำเสนอในครั้งนี้ น่าจะสามารถนำไปประยุกต์ใช้กับการแปลงฟูริเยร์แบบรวดเร็ว และการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ในวิธีระบบฐานอื่นๆ เพื่อให้มีขนาดที่เล็กลงได้

# เอกสารอ้างอิง

[1] พรชัย ภววงษ์ศักดิ์. การประมวลผลสัญญาณดิจิตอลเบื้องต้น. พิมพ์ครั้งที่ 1. กรุงเทพมหานคร : มหาวิทยาลัยเทคโนโลยีมหานคร. 2542.

[2] Cetin, E. Richard, C.S. M. and Kale I. "An Integrated 256-point Complex FFT Processor for Real-time Spectrum Analysis and Measurement.". IEEE Proceedings of Instrumentation and Measurement Technology Conference. Canada. vol. 1, 1997. pp. 96-101.

[3] Baas B.M. "A Low-Power, High-Performance, 1024-Point FFT Processor." IEEE Journal of Solid-State Circuits. vol. 34, no. 3, 1999. pp. 380-387.

[4] Khan, A.U. Marwan, M. Al-Akaidi. Khan, A.S. Khattak, S. and Mir, A.. "Performance Analysis of a 64-point FFT/IFFT Block designed for OFDM Technique Used in WLAN's". IEEE Proceedings of Multi Topic Conference. INMIC. 2003. 7th International 2003. pp. 65 – 71.

[5] Jiang, M. Yang, B. Fu, Y. Jiang A. and Wang X. "Design of FFT processor with Low Power Complex Multiplier for OFDM-based High-speed Wireless Applications". International Symposium on Communication and Information Technologies. Japan. 2004. pp. 639-641.

[6] Ganapathiraju A., Hamaker J., Picone J. and Skjellum A. "A Comparative Analysis of FFT Algorithms." Mississippi State University.

[7] Zhao Y., Erdogan A.T. and Arslan T. "A Novel Low-Power Reconfigurable FFT Processor." ISCAS 2005. IEEE International Symposium on Circuits and Systems. vol. 1, 2005. pp. 41-44.

[8] Nee R.V. , Prasad R. OFDM for Wireless Multimedia Communications. London : Artech House. 2000.

[9] ณพงศ์ สุวรรณนิพนธ์. "OFDM เทคโนโลยีสำหรับการสื่อสารไร้สายความเร็วสูง" วารสารคณะวิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์

[10] Edfors O., Sandell M., Beek J.J., Landstrom D. and Sjoberg F. "An introduction to orthogonal frequency-division multiplexing." Lulea University of Technology. 1996.

[11] Salz J., Weinstein S.B. "Fourier Transform Communication System." Proceedings of the first ACM symposium on Problems in the optimization of data communications systems. 1969. pp. 99-128.

[12] Han W., Erdogan A.T., Arslan T. and Hasan M. "The Development of High Performance FFT IP Core through Hybrid Low Power Algorithmic Methodology." Proceedings of the Design Automation Conference ASP-DAC 2005. Asia and South Pacific. vol 1, 2005. pp. 549 – 552.

[13] Weste, N. Bickerstaff, M. and Arivoli, T. "A 50 MHz 16-point FFT Processor for WLAN Applications". IEEE Custom Integrated Circuit Conference. 1997. pp. 457-460.

[14] Rudburg, M.K. Sandburg, M. and Ekholm K. "Design and Implementation of a FFT processor for VDSL". IEEE Asia-Pacific Conference on Circuits and Systems. 1998. pp. 611-614.

[15] Suk, J.H. Kim, D.W. Kwon, T.W. Hyung, S.K. and Choi, J.R. "A 8192 Complex Point FFT/IFFT for COFDM Modulation Scheme in DVB-T System". IEEE Proceedings of International Conference Systems-on-Chip. 2003. pp. 131-134.

[16] Son, B.S. Jo B.G. Sunwoo, H.M. and Kim, Y.S. "A High-Speed FFT Processor for OFDM System". IEEE International Symposium on Circuits and Systems. ISCAS 2002. vol 3, 2002. pp. 281-284.

[17] Heo, L.K. Baek, H.J. Sunwoo, H.M. Jo, G.B. and Son, S.B. "New In-Place Strategy for a Mixed-Radix FFT Processor". IEEE Proceedings of International Conference Systems-on-Chip. 2003. pp. 81-84.

[18] Brown B. "Floating Point Numbers." Computer Science Department, Southern Polytechnic State University.

[19] An American National Standard. "IEEE Standard for Radix-Independent Floating-point Arithmetic." Technical Committee on Microprocessors and Microcomputers of the IEEE Computer Society. 1994

[20] Shousheng He and Torkelson M. "Design and implementation of a 1024-point pipeline FFT processor." IEEE Proceedings of Custom Integrated Circuits Conference, 1998. pp. 131 – 134.

[21] Khan, A.U. Marwan, M. Al-Akaidi. Khan, A.S. Khattak, S. and Mir, A.. "Performance Analysis of a 64-point FFT/IFFT Block designed for OFDM Technique Used in WLAN's". IEEE Proceedings of Multi Topic Conference. INMIC. 2003. 7th International 2003. pp. 65 – 71.

[22] ชำนาญ ปัญญาใส และวัชรากร หนูทอง. ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล เบื้องต้น. กรุงเทพมหานคร : ซีเอ็ดยูเคชั่น. 2547.

[23] Hasan M., Arslan T. and Thompson J.S. "A Novel Coefficient Ordering based Low Power Pipelined Radix-4 FFT Processor for Wireless LAN Applications". IEEE Trans on Consumer Electronics. Vol. 49, no. 1, Feb. 2003. pp. 128-134.

[24] Jiang M., Yang B., Fu Y., Jiang A. and Wang X. "Design of FFT Processor with Low Power Complex Multiplier for OFDM-based High-speed Wireless Applications". International Symposium on Communication and Information Technologies. 2004. pp. 639-641.

[25] He S., Torkelson M. "Designing Pipeline FFT Processor for OFDM (de)Modulation". URSI International Symposium on Signals, Systems, and Electronics. 1998. pp. 257-262.

[26] Rudberg M.K., Sandberg M. and Ekholm K. "Design and Implementation of a FFT Processor for VDSL". IEEE Asia-Pacific Conference on Circuits and Systems. 1998. pp. 611-614.

[27] Jung Y., Yoon H., and Kim J. "New Efficient FFT Algorithm and Pipeline Implementation Results for OFDM/DMT Application". IEEE Trans on Consumer Electronics. Vol.49, no. 1, Feb. 2003. pp. 14-20.

# ภาคผนวก ก.

## แสดงค่าสัมประสิทธิ์ที่ใช้ในการคำนวณที่หาได้จากโปรแกรม matlab

ค่าสัมประสิทธิ์ ($W_N^k$) ที่ใช้ในการคำนวณการแปลงฟูริเยร์แบบเร็ว สามารถหาได้โดยใช้โปรแกรม matlab สามารถแสดงค่าสัมประสิทธิ์ที่หาได้ในตารางที่ ก.1

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{0}$ | 1.0000 | 00111111100000000000000000000000 |
| | + 0.0000i | 00000000000000000000000000000000 |
| $W_{256}^{1}$ | 0.9997 | 00111111011111111110110001010111 |
| | - 0.0245i | 10111100110010001011010000111001 |
| $W_{256}^{2}$ | 0.9988 | 00111111011111111011000101011011 |
| | - 0.0491i | 10111101010010010001110100010101 |
| $W_{256}^{3}$ | 0.9973 | 00111111011111110100111100001110 |
| | - 0.0736i | 10111101100101101011101110011001 |
| $W_{256}^{4}$ | 0.9952 | 00111111011111101100010101101101 |
| | - 0.0980i | 10111101110010001011010000111001 |
| $W_{256}^{5}$ | 0.9925 | 00111111011111100001010001111011 |
| | - 0.1224i | 10111101111110101010110011011010 |
| $W_{256}^{6}$ | 0.9892 | 00111111011111010011110000110110 |
| | - 0.1467i | 10111110000101100011100010000110 |
| $W_{256}^{7}$ | 0.9853 | 00111111011111000011110010011111 |
| | - 0.1710i | 10111110001011110001101010100000 |
| $W_{256}^{8}$ | 0.9808 | 00111111011110110001010110110101 |
| | - 0.1951i | 10111110010001111100100001001011 |
| $W_{256}^{9}$ | 0.9757 | 00111111011110011100011101111010 |
| | - 0.2191i | 10111110011000000101101111000000 |
| $W_{256}^{10}$ | 0.9700 | 00111111011110000101000111101100 |
| | - 0.2430i | 10111110011110001101010011111110 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{11}$ | 0.9638 | 00111111011101101011101110011001 |
| | - 0.2667i | 10111110100010001000110011100111 |
| $W_{256}^{12}$ | 0.9569 | 00111111011101001111011101100110 |
| | - 0.2903i | 10111110100101001010001000110100 |
| $W_{256}^{13}$ | 0.9495 | 00111111011100110001001001101111 |
| | - 0.3137i | 10111110101000001001110101001001 |
| $W_{256}^{14}$ | 0.9415 | 00111111011100010000011000100101 |
| | - 0.3369i | 10111110101011000111111000101000 |
| $W_{256}^{15}$ | 0.9330 | 00111111011011101101100100010111 |
| | - 0.3599i | 10111110101110000100010011010000 |
| $W_{256}^{16}$ | 0.9239 | 00111111011011001000010010110110 |
| | - 0.3827i | 10111110110000111111000101000001 |
| $W_{256}^{17}$ | 0.9142 | 00111111011010100000100100000011 |
| | - 0.4052i | 10111110110011110111011001100000 |
| $W_{256}^{18}$ | 0.9040 | 00111111011001110110110010001011 |
| | - 0.4276i | 10111110110110101110111001100011 |
| $W_{256}^{19}$ | 0.8932 | 00111111011001001010100011000001 |
| | - 0.4496i | 10111110111001100011000111111001 |
| $W_{256}^{20}$ | 0.8819 | 00111111011000011100010000110011 |
| | - 0.4714i | 10111110111100010101101101010111 |
| $W_{256}^{21}$ | 0.8701 | 00111111010111101011111011100000 |
| | - 0.4929i | 10111110111111000101110101100100 |
| $W_{256}^{22}$ | 0.8577 | 00111111010110111001001000111010 |
| | - 0.5141i | 10111111000000111001110000001111 |
| $W_{256}^{23}$ | 0.8449 | 00111111010110000100101101011110 |
| | - 0.5350i | 10111111000010001111010111000011 |
| $W_{256}^{24}$ | 0.8315 | 00111111010101001101110100101111 |
| | - 0.5556i | 10111111000011100011101111001101 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{25}$ | 0.8176 | 00111111010100010100111000111100 |
| | - 0.5758i | 10111111000100110110011110100001 |
| $W_{256}^{26}$ | 0.8032 | 00111111010011011001111010000100 |
| | - 0.5957i | 10111111000110000111111111001100 |
| $W_{256}^{27}$ | 0.7883 | 00111111010010011100111000000111 |
| | - 0.6152i | 10111111000111010111110110111111 |
| $W_{256}^{28}$ | 0.7730 | 00111111010001011110001101010100 |
| | - 0.6344i | 10111111001000100110100000001010 |
| $W_{256}^{29}$ | 0.7572 | 00111111010000011101011111011100 |
| | - 0.6532i | 10111111001001110011100000011101 |
| $W_{256}^{30}$ | 0.7410 | 00111111001111011011001000101101 |
| | - 0.6716i | 10111111001010111110110111111010 |
| $W_{256}^{31}$ | 0.7242 | 00111111001110010110010100101100 |
| | - 0.6895i | 10111111001100001000001100010010 |
| $W_{256}^{32}$ | 0.7071 | 00111111001101010000010010000001 |
| | - 0.7071i | 10111111001101010000010010000001 |
| $W_{256}^{33}$ | 0.6895 | 00111111001100001000001100010010 |
| | - 0.7242i | 10111111001110010110010100101100 |
| $W_{256}^{34}$ | 0.6716 | 00111111001010111110110111111010 |
| | - 0.7410i | 10111111001111011011001000101101 |
| $W_{256}^{35}$ | 0.6532 | 00111111001001110011100000011101 |
| | - 0.7572i | 10111111010000011101011111011100 |
| $W_{256}^{36}$ | 0.6344 | 00111111001000100110100000001010 |
| | - 0.7730 | 10111111010001011110001101010100 |
| $W_{256}^{37}$ | 0.6152 | 00111111000111010111110110111111 |
| | - 0.7883i | 10111111010010011100111000000111 |
| $W_{256}^{38}$ | 0.5957 | 00111111000110000111111111001100 |
| | - 0.8032i | 10111111010011011001111010000100 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{39}$ | 0.5758 | 00111111000100110110011110100001 |
| | - 0.8176i | 10111111010100010100111000111100 |
| $W_{256}^{40}$ | 0.5556 | 00111111000011100011101111001101 |
| | - 0.8315i | 10111111010101001101110100101111 |
| $W_{256}^{41}$ | 0.5350 | 00111111000010001111010111000011 |
| | - 0.8449i | 10111111010110000100101101011110 |
| $W_{256}^{42}$ | 0.5141 | 00111111000000111001110000001111 |
| | - 0.8577i | 10111111010110111001001000111010 |
| $W_{256}^{43}$ | 0.4929 | 00111110111111000101110101100100 |
| | - 0.8701i | 10111111010111101011111011100000 |
| $W_{256}^{44}$ | 0.4714 | 00111110111100010101101101010111 |
| | - 0.8819i | 10111111011000011100010000110011 |
| $W_{256}^{45}$ | 0.4496 | 00111110111001100011000111111001 |
| | - 0.8932i | 10111111011001001010100011000001 |
| $W_{256}^{46}$ | 0.4276 | 00111110110110101110111001100011 |
| | - 0.9040i | 10111111011001110110110010001011 |
| $W_{256}^{47}$ | 0.4052 | 00111110110011110111011001100000 |
| | - 0.9142i | 10111111011010100000100100000011 |
| $W_{256}^{48}$ | 0.3827 | 00111110110000111111000101000001 |
| | - 0.9239i | 10111111011011001000010010110110 |
| $W_{256}^{49}$ | 0.3599 | 00111110101110000100010011010000 |
| | - 0.9330i | 10111111011011101101100100010111 |
| $W_{256}^{50}$ | 0.3369 | 00111110101011000111111000101000 |
| | - 0.9415i | 10111111011100010000011000100101 |
| $W_{256}^{51}$ | 0.3137 | 00111110101000001001110101001001 |
| | - 0.9495i | 10111111011100110001001001101111 |
| $W_{256}^{52}$ | 0.2903 | 00111110100101001010001000110100 |
| | - 0.9569i | 10111111011101001111011101100110 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{53}$ | 0.2667 | 00111110100010001000110011100111 |
| | - 0.9638i | 10111111011101101011101110011001 |
| $W_{256}^{54}$ | 0.2430 | 00111110011110001101010011111110 |
| | - 0.9700i | 10111111011110000101000111101100 |
| $W_{256}^{55}$ | 0.2191 | 00111110011000000101101111000000 |
| | - 0.9757i | 10111111011110011100011101111010 |
| $W_{256}^{56}$ | 0.1951 | 00111110010001111100100001001011 |
| | - 0.9808i | 10111111011110110001010110110101 |
| $W_{256}^{57}$ | 0.1710 | 00111110001011110001101010100000 |
| | - 0.9853i | 10111111011111000011110010011111 |
| $W_{256}^{58}$ | 0.1467 | 00111110000101100011100010000110 |
| | - 0.9892i | 10111111011111010011110000110110 |
| $W_{256}^{59}$ | 0.1224 | 00111101111110101010110011011010 |
| | - 0.9925i | 10111111011111100001010001111011 |
| $W_{256}^{60}$ | 0.0980 | 00111101110010001011010000111001 |
| | - 0.9952i | 10111111011111101100010101101101 |
| $W_{256}^{61}$ | 0.0736 | 00111101100101101011101110011001 |
| | - 0.9973i | 10111111011111111010011110000111 0 |
| $W_{256}^{62}$ | 0.0491 | 00111101010010010001110100010101 |
| | - 0.9988i | 10111111011111111011000101011011 |
| $W_{256}^{63}$ | 0.0245 | 00111100110010001011010000111001 |
| | - 0.9997i | 10111111011111111110110001010111 |
| $W_{256}^{64}$ | 0.0000 | 00000000000000000000000000000000 |
| | - 1.0000i | 10111111100000000000000000000000 |
| $W_{256}^{65}$ | - 0.0245 | 10111100110010001011010000111001 |
| | - 0.9997i | 10111111011111111110110001010111 |
| $W_{256}^{66}$ | - 0.0491 | 10111101010010010001110100010101 |
| | - 0.9988i | 10111111011111111011000101011011 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{67}$ | - 0.0736 | 10111101100101101011101110011001 |
| | - 0.9973i | 10111111011111110100111100001110 |
| $W_{256}^{68}$ | - 0.0980 | 10111101110010001011010000111001 |
| | - 0.9952i | 10111111011111101100010101101101 |
| $W_{256}^{69}$ | - 0.1224 | 10111101111110101010110011011010 |
| | - 0.9925i | 10111111011111100001010001111011 |
| $W_{256}^{70}$ | - 0.1467 | 10111110000101100011100010000110 |
| | - 0.9892i | 10111111011111010011110000110110 |
| $W_{256}^{71}$ | - 0.1710 | 10111110001011110001101010100000 |
| | - 0.9853i | 10111111011111000011110010011111 |
| $W_{256}^{72}$ | - 0.1951 | 10111110010001111100100001001011 |
| | - 0.9808i | 10111111011110110001010110110101 |
| $W_{256}^{73}$ | - 0.2191 | 10111110011000000101101111000000 |
| | - 0.9757i | 10111111011110011100011101111010 |
| $W_{256}^{74}$ | - 0.2430 | 10111110011110001101010011111110 |
| | - 0.9700i | 10111111011110000101000111101100 |
| $W_{256}^{75}$ | - 0.2667 | 10111110100010001000110011100111 |
| | - 0.9638i | 10111111011101101011101110011001 |
| $W_{256}^{76}$ | - 0.2903 | 10111110100101001010001000110100 |
| | - 0.9569i | 10111111011101001111011101100110 |
| $W_{256}^{77}$ | - 0.3137 | 10111110101000001001110101001001 |
| | - 0.9495i | 10111111011100110001001001101111 |
| $W_{256}^{78}$ | - 0.3369 | 10111110101011000111111000101000 |
| | - 0.9415i | 10111111011100010000011000100101 |
| $W_{256}^{79}$ | - 0.3599 | 10111110101110000100010011010000 |
| | - 0.9330i | 10111111011011101101100100010111 |
| $W_{256}^{80}$ | - 0.3827 | 10111110110000111111000101000001 |
| | - 0.9239i | 10111111011011001000010010110110 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{81}$ | - 0.4052 | 10111110110011110111011001100000 |
| | - 0.9142i | 10111111011010100000100100000011 |
| $W_{256}^{82}$ | - 0.4276 | 10111110110110101110111001100011 |
| | - 0.9040i | 10111111011001110110110010001011 |
| $W_{256}^{83}$ | - 0.4496 | 10111110111001100011000111111001 |
| | - 0.8932i | 10111111011001001010100011000001 |
| $W_{256}^{84}$ | - 0.4714 | 10111110111100010101101101010111 |
| | - 0.8819i | 10111111011000011100010000110011 |
| $W_{256}^{85}$ | - 0.4929 | 10111110111111000101110101100100 |
| | - 0.8701i | 10111111010111101011111011100000 |
| $W_{256}^{86}$ | - 0.5141 | 10111111000000111001110000001111 |
| | - 0.8577i | 10111111010110111001001000111010 |
| $W_{256}^{87}$ | - 0.5350 | 10111111000010001111010111000011 |
| | - 0.8449i | 10111111010110000100101101011110 |
| $W_{256}^{88}$ | - 0.5556 | 10111111000011100011101111001101 |
| | - 0.8315i | 10111111010101001101110100101111 |
| $W_{256}^{89}$ | - 0.5758 | 10111111000100110110011110100001 |
| | - 0.8176i | 10111111010100010100111000111100 |
| $W_{256}^{90}$ | - 0.5957 | 10111111000110000111111111001100 |
| | - 0.8032i | 10111111010011011001111010000100 |
| $W_{256}^{91}$ | - 0.6152 | 10111111000111010111110110111111 |
| | - 0.7883i | 10111111010010011100111000000111 |
| $W_{256}^{92}$ | - 0.6344 | 10111111001000100110100000001010 |
| | - 0.7730 | 10111111010001011110001101010100 |
| $W_{256}^{93}$ | - 0.6532 | 10111111001001110011100000011101 |
| | - 0.7572i | 10111111010000011101011111011100 |
| $W_{256}^{94}$ | - 0.6716 | 10111111001010111110110111111010 |
| | - 0.7410i | 10111111001111011011001000101101 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{95}$ | - 0.6895 | 10111111001100001000001100010010 |
| | - 0.7242i | 10111111001110010110010100101100 |
| $W_{256}^{96}$ | - 0.7071 | 10111111001101010000010010000001 |
| | - 0.7071i | 10111111001101010000010010000001 |
| $W_{256}^{97}$ | - 0.7242 | 10111111001110010110010100101100 |
| | - 0.6895i | 10111111001100001000001100010010 |
| $W_{256}^{98}$ | - 0.7410 | 10111111001111011011001000101101 |
| | - 0.6716i | 10111111001010111110110111111010 |
| $W_{256}^{99}$ | - 0.7572 | 10111111010000011101011111011100 |
| | - 0.6532i | 10111111001001110011100000011101 |
| $W_{256}^{100}$ | - 0.7730 | 10111111010001011110001101010100 |
| | - 0.6344i | 10111111001000100110100000001010 |
| $W_{256}^{101}$ | - 0.7883 | 10111111010010011100111000000111 |
| | - 0.6152i | 10111111000111010111110110111111 |
| $W_{256}^{102}$ | - 0.8032 | 10111111010011011001111010000100 |
| | - 0.5957i | 10111111000110000111111111001100 |
| $W_{256}^{103}$ | - 0.8176 | 10111111010100010100111000111100 |
| | - 0.5758i | 10111111000100110110011110100001 |
| $W_{256}^{104}$ | - 0.8315 | 10111111010101001101110100101111 |
| | - 0.5556i | 10111111000011100011101111001101 |
| $W_{256}^{105}$ | - 0.8449 | 10111111010110000100101101011110 |
| | - 0.5350i | 10111111000010001111010111000011 |
| $W_{256}^{106}$ | - 0.8577 | 10111111010110111001001000111010 |
| | - 0.5141i | 10111111000000111001110000001111 |
| $W_{256}^{107}$ | - 0.8701 | 10111111010111101011111011100000 |
| | - 0.4929i | 10111110111111000101110101100100 |
| $W_{256}^{108}$ | - 0.8819 | 10111111011000011100010000110011 |
| | - 0.4714i | 10111110111100010101101101010111 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{109}$ | - 0.8932 | 10111111011001001010100011000001 |
| | - 0.4496i | 10111110111001100011000111111001 |
| $W_{256}^{110}$ | - 0.9040 | 10111111011001110110110010001011 |
| | - 0.4276i | 10111110110110101110111001100011 |
| $W_{256}^{111}$ | - 0.9142 | 10111111011010100000100100000011 |
| | - 0.4052i | 10111110110011110111011001100000 |
| $W_{256}^{112}$ | - 0.9239 | 10111111011011001000010010110110 |
| | - 0.3827i | 10111110110000011111100010100000 1 |
| $W_{256}^{113}$ | - 0.9330 | 10111111011011101101100100010111 |
| | - 0.3599i | 10111110101110000100010011010000 |
| $W_{256}^{114}$ | - 0.9415 | 10111111011100010000011000100101 |
| | - 0.3369i | 10111110101011000111111000101000 |
| $W_{256}^{115}$ | - 0.9495 | 10111111011100110001001001101111 |
| | - 0.3137i | 10111110101000001001110101001001 |
| $W_{256}^{116}$ | - 0.9569 | 10111111011101001111011101100110 |
| | - 0.2903i | 10111110100101001010001000110100 |
| $W_{256}^{117}$ | - 0.9638 | 10111111011101101011101110011001 |
| | - 0.2667i | 10111110100010001000110011100111 |
| $W_{256}^{118}$ | - 0.9700 | 10111111011110000101000111101100 |
| | - 0.2430i | 10111110011110001101010011111110 |
| $W_{256}^{119}$ | - 0.9757 | 10111111011110011100011101111010 |
| | - 0.2191i | 10111110011000000101101111000000 |
| $W_{256}^{120}$ | - 0.9808 | 10111111011110110001010110110101 |
| | - 0.1951i | 10111110010001111100100001001011 |
| $W_{256}^{121}$ | - 0.9853 | 10111111011111000011110010011111 |
| | - 0.1710i | 10111110001011110001101010100000 |
| $W_{256}^{122}$ | - 0.9892 | 10111111011111010011110000110110 |
| | - 0.1467i | 10111110000101100011100010000110 |

ตารางที่ ก.1 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{123}$ | - 0.9925 | 10111111011111100001010001111011 |
| | - 0.1224i | 10111101111110101010110011011010 |
| $W_{256}^{124}$ | - 0.9952 | 10111111011111101100010101101101 |
| | - 0.0980i | 10111101110010001011010000111001 |
| $W_{256}^{125}$ | - 0.9973 | 10111111011111110100111100001110 |
| | - 0.0736i | 10111101100101101011101110011001 |
| $W_{256}^{126}$ | - 0.9988 | 10111111011111111011000101011011 |
| | - 0.0491i | 10111101010010010001110100010101 |
| $W_{256}^{127}$ | - 0.9997 | 10111111011111111111011000101011 |
| | - 0.0245i | 10111100110010001011010000111001 |

ค่าสัมประสิทธิ์ ($W_N^k$) ที่ใช้ในการคำนวณการแปลงฟูริเยร์ผกผันแบบเร็ว สามารถหาได้โดยใช้โปรแกรม matlab สามารถแสดงค่าสัมประสิทธิ์ที่หาได้ในตารางที่ ก.2

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{0}$ | 1.0000 | 00111111100000000000000000000000 |
| | + 0.0000i | 00000000000000000000000000000000 |
| $W_{256}^{1}$ | 0.9997 | 00111111011111111111011000101011 |
| | + 0.0245i | 00111100110010001011010000111001 |
| $W_{256}^{2}$ | 0.9988 | 00111111011111111011000101011011 |
| | + 0.0491i | 00111101010010010001110100010101 |
| $W_{256}^{3}$ | 0.9973 | 00111111011111110100111100001110 |
| | + 0.0736i | 00111101100101101011101110011001 |
| $W_{256}^{4}$ | 0.9952 | 00111111011111101100010101101101 |
| | + 0.0980i | 00111101110010001011010000111001 |
| $W_{256}^{5}$ | 0.9925 | 00111111011111100001010001111011 |
| | + 0.1224i | 00111101111110101010110011011010 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{6}$ | 0.9892 | 00111111011111010011110000110110 |
| | + 0.1467i | 00111110000101100011100010000110 |
| $W_{256}^{7}$ | 0.9853 | 00111111011111000011110010011111 |
| | + 0.1710i | 00111110001011110001101010100000 |
| $W_{256}^{8}$ | 0.9808 | 00111111011110110001010110110101 |
| | + 0.1951i | 00111110010001111100100001001011 |
| $W_{256}^{9}$ | 0.9757 | 00111111011110011100011101111010 |
| | + 0.2191i | 00111110011000000101101111000000 |
| $W_{256}^{10}$ | 0.9700 | 00111111011110000101000111101100 |
| | + 0.2430i | 00111110011110001101010011111110 |
| $W_{256}^{11}$ | 0.9638 | 00111111011101101011101110011001 |
| | + 0.2667i | 00111110100010001000110011100111 |
| $W_{256}^{12}$ | 0.9569 | 00111111011101001111011101100110 |
| | + 0.2903i | 00111110100101001010001000110100 |
| $W_{256}^{13}$ | 0.9495 | 00111111011100110001001001101111 |
| | + 0.3137i | 00111110101000001001110101001001 |
| $W_{256}^{14}$ | 0.9415 | 00111111011100010000011000100101 |
| | + 0.3369i | 00111110101011000111111000101000 |
| $W_{256}^{15}$ | 0.9330 | 00111111011011101101100100010111 |
| | + 0.3599i | 00111110101110000100010011010000 |
| $W_{256}^{16}$ | 0.9239 | 00111111011011001000010010110110 |
| | + 0.3827i | 00111110110000011111100010100001 |
| $W_{256}^{17}$ | 0.9142 | 00111111011010100000100100000011 |
| | + 0.4052i | 00111110110011110111011001100000 |
| $W_{256}^{18}$ | 0.9040 | 00111111011001110110110010001011 |
| | + 0.4276i | 00111110110110101110111001100011 |
| $W_{256}^{19}$ | 0.8932 | 00111111011001001010100011000001 |
| | + 0.4496i | 00111110111001100011000111111001 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{20}$ | 0.8819 | 00111111011000011100010000110011 |
| | + 0.4714i | 00111110111100010101101101010111 |
| $W_{256}^{21}$ | 0.8701 | 00111111010111101011111011100000 |
| | + 0.4929i | 00111110111111000101110101100100 |
| $W_{256}^{22}$ | 0.8577 | 00111111010110111001001000111010 |
| | + 0.5141i | 00111111000000111001110000001111 |
| $W_{256}^{23}$ | 0.8449 | 00111111010110000100101101011110 |
| | + 0.5350i | 00111111000010001111010111000011 |
| $W_{256}^{24}$ | 0.8315 | 00111111010101001101110100101111 |
| | + 0.5556i | 00111111000011100011101111001101 |
| $W_{256}^{25}$ | 0.8176 | 00111111010100010100111000111100 |
| | + 0.5758i | 00111111000100110110011110100001 |
| $W_{256}^{26}$ | 0.8032 | 00111111010011011001111010000100 |
| | + 0.5957i | 00111111000110000111111111001100 |
| $W_{256}^{27}$ | 0.7883 | 00111111010010011100111000000111 |
| | + 0.6152i | 00111111000111010111110110111111 |
| $W_{256}^{28}$ | 0.7730 | 00111111010001011110001101010100 |
| | + 0.6344i | 00111111001000100110100000001010 |
| $W_{256}^{29}$ | 0.7572 | 00111111010000011101011111011100 |
| | + 0.6532i | 00111111001001110011100000011101 |
| $W_{256}^{30}$ | 0.7410 | 00111111001111011011001000101101 |
| | + 0.6716i | 00111111001010111110110111111010 |
| $W_{256}^{31}$ | 0.7242 | 00111111001110010110010100101100 |
| | + 0.6895i | 00111111001100001000001100010010 |
| $W_{256}^{32}$ | 0.7071 | 00111111001101010000010010000001 |
| | + 0.7071i | 00111111001101010000010010000001 |
| $W_{256}^{33}$ | 0.6895 | 00111111001100001000001100010010 |
| | + 0.7242i | 00111111001110010110010100101100 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{34}$ | 0.6716 | 00111111001010111110110111111010 |
| | + 0.7410i | 00111111001111011011001000101101 |
| $W_{256}^{35}$ | 0.6532 | 00111111001001110011100000011101 |
| | + 0.7572i | 00111111010000011101011111011100 |
| $W_{256}^{36}$ | 0.6344 | 00111111001000100110100000001010 |
| | + 0.7730 | 00111111010001011110001101010100 |
| $W_{256}^{37}$ | 0.6152 | 00111111000111010111110110111111 |
| | + 0.7883i | 00111111010010011100111000000111 |
| $W_{256}^{38}$ | 0.5957 | 00111111000110000111111111001100 |
| | + 0.8032i | 00111111010011011001111010000100 |
| $W_{256}^{39}$ | 0.5758 | 00111111000100110110011110100001 |
| | + 0.8176i | 00111111010100010100111000111100 |
| $W_{256}^{40}$ | 0.5556 | 00111111000011100011101111001101 |
| | + 0.8315i | 00111111010101001101110100101111 |
| $W_{256}^{41}$ | 0.5350 | 00111111000010001111010111000011 |
| | + 0.8449i | 00111111010110000100101101011110 |
| $W_{256}^{42}$ | 0.5141 | 00111111000000111001110000001111 |
| | + 0.8577i | 00111111010110111001001000111010 |
| $W_{256}^{43}$ | 0.4929 | 00111110111111000101110101100100 |
| | + 0.8701i | 00111111010111101011111011100000 |
| $W_{256}^{44}$ | 0.4714 | 00111110111100010101101101010111 |
| | + 0.8819i | 00111111011000011100010000110011 |
| $W_{256}^{45}$ | 0.4496 | 00111110111001100011000111111001 |
| | + 0.8932i | 00111111011001001010100011000001 |
| $W_{256}^{46}$ | 0.4276 | 00111110110110101110111001100011 |
| | + 0.9040i | 00111111011001110110110010001011 |
| $W_{256}^{47}$ | 0.4052 | 00111110110011110111011001100000 |
| | + 0.9142i | 00111111011010100000100100000011 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{48}$ | 0.3827 | 00111110110000111111000101000001 |
| | + 0.9239i | 00111111011011001000010010110110 |
| $W_{256}^{49}$ | 0.3599 | 00111110101110000100010011010000 |
| | + 0.9330i | 00111111011011101101100100010111 |
| $W_{256}^{50}$ | 0.3369 | 00111110101011000111111000101000 |
| | + 0.9415i | 00111111011100010000011000100101 |
| $W_{256}^{51}$ | 0.3137 | 00111110101000001001110101001001 |
| | + 0.9495i | 00111111011100110001001001101111 |
| $W_{256}^{52}$ | 0.2903 | 00111110100101001010001000110100 |
| | + 0.9569i | 00111111011101001111011101100110 |
| $W_{256}^{53}$ | 0.2667 | 00111110100010001000110011100111 |
| | + 0.9638i | 00111111011101101011101110011001 |
| $W_{256}^{54}$ | 0.2430 | 00111110011110001101010011111110 |
| | + 0.9700i | 00111111011110000101000111101100 |
| $W_{256}^{55}$ | 0.2191 | 00111110011000000101101111000000 |
| | + 0.9757i | 00111111011110011100011101111010 |
| $W_{256}^{56}$ | 0.1951 | 00111110010001111100100001001011 |
| | + 0.9808i | 00111111011110110001010110110101 |
| $W_{256}^{57}$ | 0.1710 | 00111110001011110001101010100000 |
| | + 0.9853i | 00111111011111000011110010011111 |
| $W_{256}^{58}$ | 0.1467 | 00111110000101100011100010000110 |
| | + 0.9892i | 00111111011111010011110000110110 |
| $W_{256}^{59}$ | 0.1224 | 00111101111110101010110011011010 |
| | + 0.9925i | 00111111011111100001010001111011 |
| $W_{256}^{60}$ | 0.0980 | 00111101110010001011010000111001 |
| | + 0.9952i | 00111111011111101100010101101101 |
| $W_{256}^{61}$ | 0.0736 | 00111101100101101011101110011001 |
| | + 0.9973i | 00111111011111110100111100001110 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{62}$ | 0.0491 | 00111101010010010001110100010101 |
| | + 0.9988i | 00111111011111111011000101011011 |
| $W_{256}^{63}$ | 0.0245 | 00111100110010001011010000111001 |
| | + 0.9997i | 00111111011111111110110001010111 |
| $W_{256}^{64}$ | 0.0000 | 00000000000000000000000000000000 |
| | + 1.0000i | 00111111100000000000000000000000 |
| $W_{256}^{65}$ | - 0.0245 | 10111100110010001011010000111001 |
| | + 0.9997i | 00111111011111111110110001010111 |
| $W_{256}^{66}$ | - 0.0491 | 10111101010010010001110100010101 |
| | + 0.9988i | 00111111011111111011000101011011 |
| $W_{256}^{67}$ | - 0.0736 | 10111101100101101011101110011001 |
| | + 0.9973i | 00111111011111110100111100001110 |
| $W_{256}^{68}$ | - 0.0980 | 10111101110010001011010000111001 |
| | + 0.9952i | 00111111011111101100010101101101 |
| $W_{256}^{69}$ | - 0.1224 | 10111101111110101010110011011010 |
| | + 0.9925i | 00111111011111100001010001111011 |
| $W_{256}^{70}$ | - 0.1467 | 10111110000101100011100010000110 |
| | + 0.9892i | 00111111011111010011110000110110 |
| $W_{256}^{71}$ | - 0.1710 | 10111110001011110001101010100000 |
| | + 0.9853i | 00111111011111000011110010011111 |
| $W_{256}^{72}$ | - 0.1951 | 10111110010001111100100001001011 |
| | + 0.9808i | 00111111011110110001010110110101 |
| $W_{256}^{73}$ | - 0.2191 | 10111110011000000101101111000000 |
| | + 0.9757i | 00111111011110011100011101111010 |
| $W_{256}^{74}$ | - 0.2430 | 10111110011110001101010011111110 |
| | + 0.9700i | 00111111011110000101000111101100 |
| $W_{256}^{75}$ | - 0.2667 | 10111110100010001000110011100111 |
| | + 0.9638i | 00111111011101101011101110011001 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขยิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{76}$ | - 0.2903 | 10111110100101001010001000110100 |
| | + 0.9569i | 00111111011101001111011101100110 |
| $W_{256}^{77}$ | - 0.3137 | 10111110101000001001110101001001 |
| | + 0.9495i | 00111111011100110001001001101111 |
| $W_{256}^{78}$ | - 0.3369 | 10111110101011000111111000101000 |
| | + 0.9415i | 00111111011100010000011000100101 |
| $W_{256}^{79}$ | - 0.3599 | 10111110101110000100010011010000 |
| | + 0.9330i | 00111111011011101101100100010111 |
| $W_{256}^{80}$ | - 0.3827 | 10111110110000111111000101000001 |
| | + 0.9239i | 00111111011011001000010010110110 |
| $W_{256}^{81}$ | - 0.4052 | 10111110110011110111011001100000 |
| | + 0.9142i | 00111111011010100000100100000011 |
| $W_{256}^{82}$ | - 0.4276 | 10111110110110101110111001100011 |
| | + 0.9040i | 00111111011001110110110010001011 |
| $W_{256}^{83}$ | - 0.4496 | 10111110111001100011000111111001 |
| | + 0.8932i | 00111111011001001010100011000001 |
| $W_{256}^{84}$ | - 0.4714 | 10111110111100010101101101010111 |
| | + 0.8819i | 00111111011000011100010000110011 |
| $W_{256}^{85}$ | - 0.4929 | 10111110111111000101110101100100 |
| | + 0.8701i | 00111111010111101011111011100000 |
| $W_{256}^{86}$ | - 0.5141 | 10111111000000111001110000001111 |
| | + 0.8577i | 00111111010110111001001000111010 |
| $W_{256}^{87}$ | - 0.5350 | 10111111000010001111010111000011 |
| | + 0.8449i | 00111111010110000100101101011110 |
| $W_{256}^{88}$ | - 0.5556 | 10111111000011100011101111001101 |
| | + 0.8315i | 00111111010101001101110100101111 |
| $W_{256}^{89}$ | - 0.5758 | 10111111000100110110011110100001 |
| | + 0.8176i | 00111111010100010100111000111100 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{90}$ | - 0.5957 | 10111111000110000111111111001100 |
| | + 0.8032i | 00111111010011011001111010000100 |
| $W_{256}^{91}$ | - 0.6152 | 10111111000111010111110110111111 |
| | + 0.7883i | 00111111010010011100111000000111 |
| $W_{256}^{92}$ | - 0.6344 | 10111111001000100110100000001010 |
| | + 0.7730 | 00111111010001011110001101010100 |
| $W_{256}^{93}$ | - 0.6532 | 10111111001001110011100000011101 |
| | + 0.7572i | 00111111010000011101011111011100 |
| $W_{256}^{94}$ | - 0.6716 | 10111111001010111110110111111010 |
| | + 0.7410i | 00111111001111011011001000101101 |
| $W_{256}^{95}$ | - 0.6895 | 10111111001100001000001100010010 |
| | + 0.7242i | 00111111001110010110010100101100 |
| $W_{256}^{96}$ | - 0.7071 | 10111111001101010000010010000001 |
| | + 0.7071i | 00111111001101010000010010000001 |
| $W_{256}^{97}$ | - 0.7242 | 10111111001110010110010100101100 |
| | + 0.6895i | 00111111001100001000001100010010 |
| $W_{256}^{98}$ | - 0.7410 | 10111111001111011011001000101101 |
| | + 0.6716i | 00111111001010111110110111111010 |
| $W_{256}^{99}$ | - 0.7572 | 10111111010000011101011111011100 |
| | + 0.6532i | 00111111001001110011100000011101 |
| $W_{256}^{100}$ | - 0.7730 | 10111111010001011110001101010100 |
| | + 0.6344i | 00111111001000100110100000001010 |
| $W_{256}^{101}$ | - 0.7883 | 10111111010010011100111000000111 |
| | + 0.6152i | 00111111000111010111110110111111 |
| $W_{256}^{102}$ | - 0.8032 | 10111111010011011001111010000100 |
| | + 0.5957i | 00111111000110000111111111001100 |
| $W_{256}^{103}$ | - 0.8176 | 10111111010100010100111000111100 |
| | + 0.5758i | 00111111000100110110011110100001 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{104}$ | - 0.8315 | 10111111010101001101110100101111 |
| | + 0.5556i | 00111111000011100011101111001101 |
| $W_{256}^{105}$ | - 0.8449 | 10111111010110000100101101011110 |
| | + 0.5350i | 00111111000010001111010111000011 |
| $W_{256}^{106}$ | - 0.8577 | 10111111010110111001001000111010 |
| | + 0.5141i | 00111111000000111001110000001111 |
| $W_{256}^{107}$ | - 0.8701 | 10111111010111101011111011100000 |
| | + 0.4929i | 00111110111111000101110101100100 |
| $W_{256}^{108}$ | - 0.8819 | 10111111011000011100010000110011 |
| | + 0.4714i | 00111110111100010101101101010111 |
| $W_{256}^{109}$ | - 0.8932 | 10111111011001001010100011000001 |
| | + 0.4496i | 00111110111001100011000111111001 |
| $W_{256}^{110}$ | - 0.9040 | 10111111011001110110110010001011 |
| | + 0.4276i | 00111110110110101110111001100011 |
| $W_{256}^{111}$ | - 0.9142 | 10111111011010100000100100000011 |
| | + 0.4052i | 00111110110011110111011001100000 |
| $W_{256}^{112}$ | - 0.9239 | 10111111011011001000010010110110 |
| | + 0.3827i | 00111110110000111111000101000001 |
| $W_{256}^{113}$ | - 0.9330 | 10111111011011101101100100010111 |
| | + 0.3599i | 00111110101110000100010011010000 |
| $W_{256}^{114}$ | - 0.9415 | 10111111011100010000011000100101 |
| | + 0.3369i | 00111110101011000111111000101000 |
| $W_{256}^{115}$ | - 0.9495 | 10111111011100110001001001101111 |
| | + 0.3137i | 00111110101000001001110101001001 |
| $W_{256}^{116}$ | - 0.9569 | 10111111011101001111011101100110 |
| | + 0.2903i | 00111110100101001010001000110100 |
| $W_{256}^{117}$ | - 0.9638 | 10111111011101101011101110011001 |
| | + 0.2667i | 00111110100010001000110011100111 |

ตารางที่ ก.2 แสดงค่าสัมประสิทธิ์ ($W_N^k$) ที่หาได้โดยใช้โปรแกรม matlab (ต่อ)

| $W_N^k$ | เลขฐาน 10 | เลขอิงค่าดรรชนีขนาด 32 บิต |
|---|---|---|
| $W_{256}^{118}$ | - 0.9700 | 10111111011110000101000111101100 |
| | + 0.2430i | 00111110011110001101010011111110 |
| $W_{256}^{119}$ | - 0.9757 | 10111111011110011100011101111010 |
| | + 0.2191i | 00111110011000000101101111000000 |
| $W_{256}^{120}$ | - 0.9808 | 10111111011110110001010110110101 |
| | + 0.1951i | 00111110010001111100100001001011 |
| $W_{256}^{121}$ | - 0.9853 | 10111111011111000011110010011111 |
| | + 0.1710i | 00111110001011110001101010100000 |
| $W_{256}^{122}$ | - 0.9892 | 10111111011111010011110000110110 |
| | + 0.1467i | 00111110000101100011100010000110 |
| $W_{256}^{123}$ | - 0.9925 | 10111111011111100001010001111011 |
| | + 0.1224i | 00111101111110101010110011011010 |
| $W_{256}^{124}$ | - 0.9952 | 10111111011111101100010101101101 |
| | + 0.0980i | 00111101110010001011010000111001 |
| $W_{256}^{125}$ | - 0.9973 | 10111111011111110100111100001110 |
| | + 0.0736i | 00111101100101101011101110011001 |
| $W_{256}^{126}$ | - 0.9988 | 10111111011111111101100010101011 |
| | + 0.0491i | 00111101010010010001110100010101 |
| $W_{256}^{127}$ | - 0.9997 | 10111111011111111111011000101011 |
| | + 0.0245i | 00111100110010001011010000111001 |

# ภาคผนวก ข.

## โปรแกรม Matlab

โปรแกรม Matlab สำหรับการคำนวณการแปลงฟูริเยร์แบบไม่ต่อเนื่อง ที่จำนวน N จุดใดๆ

```
N = length (y) ;
x = y ;
c = j*2*pi/N ;
%คำนวณตามสมการของการแปลง DFT
for k = 0 : N-1
    X (k+1) = sum (x.*exp(-c*k*[0 : N-1])) ;
end
```

โปรแกรม Matlab สำหรับการคำนวณการแปลงฟูริเยร์ย้อนกลับแบบไม่ต่อเนื่อง ที่จำนวน N จุดใดๆ

```
N = length (y) ;
x = y/N ;
c = j*2*pi/N ;
%คำนวณตามสมการของการแปลง IDFT
for k = 0 : N-1
    X (k+1) = sum (x.*exp(c*k*[0 : N-1])) ;
end
```

โปรแกรม matlab สำหรับคำนวณการแปลงฟูริเยร์แบบรวดเร็ว ที่จำนวน N จุดใดๆ

```
N = length (y) ;
x = y ;
b = log10 (N)/log10 (2)
%คำนวณค่าคงที่ W_N^k
W = exp (-j*2*pi*[0 : N/2 - 1]/N) ;
```

```
%สลับลำดับของสัญญาณขาเข้า เป็นลำดับที่สลับซ้ายขวา
        xold = x ;
        for i = 0 : N-1
                index_old = i ;
                index = 0 ;
                for j = 1 : b
                        index = index*2 ;
                        a = rem (index_old,2) ;
                        index_old = (index_old - a)/2 ;
                        index = index + a ;
                end
                x (index + 1) = xold (i + 1) ;
        end
%คำนวณตามแผนภาพผีเสื้อ
        M = 1 ;
        while M < N
                M2 = 2*M ;
                for i = 1 : M2 : N
                        xtemp1 = x (i : i + M - 1) ;
                        xtemp2 = x (i + M : i + M2 - 1) . *W(1 : N/M2 : N/2) ;
                        x (i : i + M - 1) = xtemp1 + xtemp2 ;
                        x (i + M : i + M2 - 1) = - xtemp2 + xtemp1 ;
                end
                M = M2 ;
        end
        X = x ;
```

**โปรแกรม matlab สำหรับคำนวณการแปลงฟูริเยร์ย้อนกลับแบบรวดเร็ว ที่จำนวน N จุดใดๆ**



```
        N = length (y) ;
        x = y/N ;
        b = log10 (N)/log10 (2)
```

```
%คำนวณค่าคงที่ W_N^k
        W = exp (j*2*pi*[0 : N/2 - 1]/N) ;
%สลับลำดับของสัญญาณขาเข้า เป็นลำดับที่สลับซ้ายขวา
        xold = x ;
        for i = 0 : N-1
                index_old = i ;
                index = 0 ;
                for j = 1 : b
                        index = index*2 ;
                        a = rem (index_old,2) ;
                        index_old = (index_old - a)/2 ;
                        index = index + a ;
                end
                x (index + 1) = xold (i + 1) ;
        end
%คำนวณตามแผนภาพผีเสื้อ
        M = 1 ;
        while M < N
                M2 = 2*M ;
                for i = 1 : M2 : N
                        xtemp1 = x (i : i + M - 1) ;
                        xtemp2 = x (i + M : i + M2 - 1) . *W(1 : N/M2 : N/2) ;
                        x (i : i + M - 1) = xtemp1 + xtemp2 ;
                        x (i + M : i + M2 - 1) = - xtemp2 + xtemp1 ;
                end
                M = M2 ;
        end
        X = x ;
```



โปรแกรม Matlab สำหรับคำนวณ QAM แปลงค่าเป็นเลขอิงดรรชนีและส่งค่าที่ได้ออกพอร์ตอนุกรมครั้งละ 8 บิต

```
function[z] = mod(x)
 s=serial('COM3');
 fopen(s);
%คำนวณ QAM
 y=amod(x,10,50,'qam');
z=length(y);
for i =1:z
  u=y(i);
  o=qam_floating(u);

%แปลงค่าเป็นเลขฐาน 10 และส่งค่าออกพอร์ตอนุกรม
  out=[o(1) o(2) o(3) o(4) o(5) o(6) o(7) o(8)];
  if out(1)==1
    a=128;
  else
    a=0;
  end
  if out(2)==1
    b=64;
  else
    b=0;
  end
  if out(3)==1
    c=32;
  else
    c=0;
  end
  if out(4)==1
    d=16;
  else
    d=0;
  end
```

```
if out(5)==1
   e=8;
else
   e=0;
end
if out(6)==1
   f=4;
else
   f=0;
end
if out(7)==1
   g=2;
else
   g=0;
end
if out(8)==1
   h=1;
else
   h=0;
end
output = a+b+c+d+e+f+g+h
fwrite(s,[output])
out1=[o(9) o(10) o(11) o(12) o(13) o(14) o(15) o(16)];
if out1(1)==1
   aa=128;
else
   aa=0;
end
if out1(2)==1
   bb=64;
else
   bb=0;
```

```
end
if out1(3)==1
   cc=32;
else
   cc=0;
end
if out1(4)==1
   dd=16;
else
   dd=0;
end
if out1(5)==1
   cc=8;
else
   ee=0;
end
if out1(6)==1
   ff=4;
else
   ff=0;
end
if out1(7)==1
   gg=2;
else
   gg=0;
end
if out1(8)==1
   hh=1;
else
   hh=0;
end
output1 = aa+bb+cc+dd+ee+ff+gg+hh
```

```
fwrite(s,[output1])
out2=[o(17) o(18) o(19) o(20) o(21) o(22) o(23) o(24)];
if out2(1)==1
   aaa=128;
else
   aaa=0;
end
if out2(2)==1
   bbb=64;
else
   bbb=0;
end
if out2(3)==1
   ccc=32;
else
   ccc=0;
end
if out2(4)==1
   ddd=16;
else
   ddd=0;
end
if out2(5)==1
   eee=8;
else
   eee=0;
end
if out2(6)==1
   fff=4;
else
   fff=0;
end
```

```
if out2(7)==1
   ggg=2;
else
   ggg=0;
end
if out2(8)==1
   hhh=1;
else
   hhh=0;
end
output2 = aaa+bbb+ccc+ddd+eee+fff+ggg+hhh
fwrite(s,[output2])
out3=[o(25) o(26) o(27) o(28) o(29) o(30) o(31) o(32)];
if out3(1)==1
   aaaa=128;
else
   aaaa=0;
end
if out3(2)==1
   bbbb=64;
else
   bbbb=0;
end
if out3(3)==1
   cccc=32;
else
   cccc=0;
end
if out3(4)==1
   dddd=16;
else
   dddd=0;
```

```
        end
        if out3(5)==1
           eeee=8;
        else
           eeee=0;
        end
        if out3(6)==1
           ffff=4;
        else
           ffff=0;
        end
        if out3(7)==1
           gggg=2;
        else
           gggg=0;
        end
        if out3(8)==1
           hhhh=1;
        else
           hhhh=0;
        end
        output3 = aaaa+bbbb+cccc+dddd+eeee+ffff+gggg+hhhh
        fwrite(s,[output3])
      end
      fclose(s);
```

%แปลงค่าเป็นเลขอิงดรรชนี

```
      function [op] = qam_floating(m)
      H=length(m);
      for i=1:H
```



%check 0

```
      if m(i) ~= 0
```

```
% หาค่า sign
        if m(i) >= 0
          os = 0;
           else
            os = 1;
            m(i) = m(i)*(-1);
          end
          if m(i) < 2 & m(i) >= 1
            x = 127;
          end
%แยกส่วนการคิดค่ามากกว่า 2 ,น้อยกว่า 2 แต่มากกว่าเท่ากับ 1 , น้อยกว่า 1 แต่ไม่เท่ากับ 0
        n=0;
          while m(i) >= 2
            m(i)= m(i)/2;
            n = n+1;
            x = n+127;
          end
         b = 0;
          while m(i) < 1 & m(i) ~= 0
            m(i) = m(i)*2;
            b = b+1;
            x = 127-b;
          end
%หาค่า mantissa
          z = m(i)-1;
            o0m = 1 ;
          if z >= 1/2
            z = z-1/2;
            o1m = 1;
          else
            z = z;
            o1m = 0;
```

```
 end
 if z >= 1/4
   z = z-1/4;
   o2m = 1;
 else
   z = z;
   o2m = 0;
 end

 if z >= 1/8
    z = z-1/8;
    o3m = 1;
  else
    z = z;
    o3m = 0;
  end
 if z >= 1/16
    z = z-1/16;
    o4m = 1;
  else
    z = z;
    o4m = 0;
  end
 if z >= 1/32
    z = z-1/32;
    o5m = 1;
  else
    z = z;
    o5m = 0;
  end
 if z >= 1/64
    z = z-1/64;
```

```
    o6m = 1;
  else
    z = z;
    o6m = 0;
  end
if z >= 1/128
    z = z-1/128;
    o7m = 1;
  else
    z = z;
    o7m = 0;
  end
if z >= 1/256
    z = z-1/256;
    o8m = 1;
  else
    z = z;
    o8m = 0;
  end
if z >= 1/512
    z = z-1/512;
    o9m = 1;
  else
    z = z;
    o9m = 0;
  end
if z >= 1/1024
    z = z-1/1024;
    o10m = 1;
  else
    z = z;
    o10m = 0;
```

```
  end
if z >= 1/2048
    z = z-1/2048;
    o11m = 1;
  else
    z = z;
    o11m = 0;
  end
if z >= 1/4096
    z = z-1/4096;
    o12m = 1;
  else
    z = z;
    o12m = 0;
  end
if z >= 1/8192
    z= z-1/8192;
    o13m = 1;
  else
    z = z;
    o13m = 0;
  end
if z >= 1/16384
    z = z-1/16384;
    o14m = 1;
  else
    z= z;
    o14m = 0;
  end
if z >= 1/32768
    z = z-1/32768;
    o15m = 1;
```

```
  else
    z = z;
    o15m = 0;
  end
if z >= 1/65536
    z = z-1/65536;
    o16m = 1;
  else
    z = z;
    o16m = 0;
  end
if z>= 1/131072
    z = z-1/131072;
    o17m = 1;
  else
    z = z;
    o17m = 0;
  end
if z >= 1/262144
    z = z-1/262144;
    o18m = 1;
  else
    z = z;
    o18m = 0;
  end
if z >= 1/524288
    z = z-1/524288;
    o19m = 1;
  else
    z = z;
    o19m = 0;
  end
```

```
if z >= 1/1048576
    z = z-1/1048576;
    o20m = 1;
  else
    z = z;
    o20m = 0;
  end
if z>= 1/2097152
    z = z-1/2097152;
    o21m = 1;
  else
    z = z;
    o21m = 0;
  end
if z >= 1/4194304
    z = z-1/4194304;
    o22m = 1;
  else
    z = z;
    o22m = 0;
  end
%หาค่า exponent
if x >= 128
    x = x-128;
    o1e = 1;
  else
    x = x;
    o1e = 0;
  end
if x >= 64
    x = x-64;
    o2e = 1;
```

```
  else
    x = x;
    o2e = 0;
  end
if x >= 32
    x = x-32;
    o3e = 1;
  else
    x = x;
    o3e = 0;
  end
if x >= 16
    x = x-16;
    o4e = 1;
  else
    x = x;
    o4e = 0;
  end
if x >= 8
    x = x-8;
    o5e = 1;
  else
    x = x;
    o5e = 0;
  end
if x >= 4
    x = x-4;
    o6e = 1;
  else
    x = x;
    o6e = 0;
  end
```

```
        if x >= 2
          x = x-2;
          o7e = 1;
        else
          x = x;
          o7e = 0;
        end
        if x >= 1
          o8e = 1;
        else
          x = x;
          o8e = 0;
        end
%รวมเป็น 32 บิตของเลขอิงครรชนี
        op = [os o1e o2e o3e o4e o5e o6e o7e o8e o0m o1m o2m o3m o4m o5m o6m o7m o8m
o9m o10m o11m o12m o13m o14m o15m o16m o17m o18m o19m o20m o21m o22m];
        else
        op = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
         end
       end
```

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

**Source: www.LookupTables.com**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | Ç | 144 | É | 161 | í | 177 | ▒ | 193 | ┴ | 209 | ╤ | 225 | ß | 241 | ± |
| 129 | ü | 145 | æ | 162 | ó | 178 | ▓ | 194 | ┬ | 210 | ╥ | 226 | Γ | 242 | ≥ |
| 130 | é | 146 | Æ | 163 | ú | 179 | │ | 195 | ├ | 211 | ╙ | 227 | π | 243 | ≤ |
| 131 | â | 147 | ô | 164 | ñ | 180 | ┤ | 196 | ─ | 212 | ╘ | 228 | Σ | 244 | ⌠ |
| 132 | ä | 148 | ö | 165 | Ñ | 181 | ╡ | 197 | ┼ | 213 | ╒ | 229 | σ | 245 | ⌡ |
| 133 | à | 149 | ò | 166 | ª | 182 | ╢ | 198 | ╞ | 214 | ╓ | 230 | µ | 246 | ÷ |
| 134 | å | 150 | û | 167 | º | 183 | ╖ | 199 | ╟ | 215 | ╫ | 231 | τ | 247 | ≈ |
| 135 | ç | 151 | ù | 168 | ¿ | 184 | ╕ | 200 | ╚ | 216 | ╪ | 232 | Φ | 248 | ° |
| 136 | ê | 152 | ÿ | 169 | ⌐ | 185 | ╣ | 201 | ╔ | 217 | ┘ | 233 | Θ | 249 | ∙ |
| 137 | ë | 153 | Ö | 170 | ¬ | 186 | ║ | 202 | ╩ | 218 | ┌ | 234 | Ω | 250 | · |
| 138 | è | 154 | Ü | 171 | ½ | 187 | ╗ | 203 | ╦ | 219 | █ | 235 | δ | 251 | √ |
| 139 | ï | 156 | £ | 172 | ¼ | 188 | ╝ | 204 | ╠ | 220 | ▄ | 236 | ∞ | 252 | ⁻ |
| 140 | î | 157 | ¥ | 173 | ¡ | 189 | ╜ | 205 | ═ | 221 | ▌ | 237 | φ | 253 | ² |
| 141 | ì | 158 | ₧ | 174 | « | 190 | ╛ | 206 | ╬ | 222 | ▐ | 238 | ε | 254 | ■ |
| 142 | Ä | 159 | ƒ | 175 | » | 191 | ┐ | 207 | ╧ | 223 | ▀ | 239 | ∩ | 255 | |
| 143 | Å | 160 | á | 176 | ░ | 192 | └ | 208 | ╨ | 224 | α | 240 | ≡ | | |

**Source: www.LookupTables.com**

# ภาคผนวก ง.

## โปรแกรมภาษา VHDL

โปรแกรมภาษา VHDL สำหรับใช้ในการสร้างวงจรประมวลผลของการแปลงฟูริเยร์แบบเร็ว และวงจรประมวลผลของการแปลงฟูริเยร์ผกผันแบบเร็วที่จำนวน 256 จุด

โปรแกรม ง.1 โปรแกรม swap (comparator)

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity swap is
port (
    a : in std_logic_vector (31 downto 0) ;
    b : in std_logic_vector (31 downto 0) ;
    clock : in std_logic ;
    rst_swap , en_swap : in std_logic ;
    d : out std_logic_vector (31 downto 0) ;
    large_exp : out std_logic_vector (7 downto 0) ;
    cc  : out std_logic_vector (32 downto 0 ) ) ;
end swap;
architecture Behavioral of swap is
begin
process (a , b , clock , rst_swap , en_swap)
--กำหนดตัวแปรสำหรับเก็บค่า exponent และค่า mantissa
variable x , y : std_logic_vector (7 downto 0) ;
variable p , q : std_logic_vector (22 downto 0) ;
begin
-- reset เป็น 1 ไม่มีการทำงานส่งค่า 0 ออกไปทั้งหมด
if(rst_swap = '1')then
cc <= "000000000000000000000000000000000";
d <= "00000000000000000000000000000000";
```

```
large_exp <= "00000000";
elsif (clock = '1' and clock'event ) then
-- แยกค่า exponent และ mantissa ออกจากกัน
                if(en_swap = '1') then
                x := a (30 downto 23) ;
                y := b (30 downto 23) ;
                p := a (22 downto 0) ;
                q := b (22 downto 0) ;
-- ตรวจสอบหาข้อมูลค่ามากจากค่า exponent พร้อมส่ง mantissa ค่าน้อย
                if (x < y) then
                        cc <= '1' & a (22 downto 0) & "000000000" ;
                        d <= b (22 downto 0) & "000000000" ;
                        large_exp <= b (30 downto 23) ;
                        elsif (y < x) then
                        cc <= '1' & b (22 downto 0) & "000000000" ;
                        d <=  a (22 downto 0) & "000000000" ;
                        large_exp <= a (30 downto 23) ;
                        elsif ((x=y) and (p < q)) then
                        cc <= '1' & a (22 downto 0) & "000000000" ;
                        d <=  b (22 downto 0) & "000000000" ;
                        large_exp <= b (30 downto 23) ;
                        else
                        cc <= '1' & b (22 downto 0) & "000000000" ;
                        d <= a (22 downto 0) & "000000000" ;
                        large_exp <= a (30 downto 23) ;
                        end if ;
                end if ;
        end if ;
end process;
end Behavioral;
```

โปรแกรม ง.2โปรแกรม summer

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity summer is
port (
    num1 , num2 : in std_logic_vector (31 downto 0) ;
    addpulse_in , addsub , rst_sum , clock : in std_logic ;
    add_finish : out std_logic ;
    sumout : out std_logic_vector ( 32 downto 0) ) ;
end summer;
architecture Behavioral of summer is
begin
process (num1 , num2 , addpulse_in , rst_sum , clock )
-- กำหนดตัวแปรสำหรับพักข้อมูลที่เข้ามา
variable temp_num1 , temp_sum , temp_num2 : std_logic_vector (32 downto 0);
begin
if (clock = '1' and clock'event) then
if (rst_sum = '0') then
        if (addpulse_in = '1') then
        temp_num1 := '0' & num1 (31 downto 0) ;
        temp_num2 := '0' & num2 (31 downto 0) ;
– sign เหมือนกันเอาค่า mantissa มาบวกกัน
                if (addsub = '1') then
                temp_sum := temp_num1 + temp_num2 ;
                sumout <= temp_sum ;
                add_finish <= '1' ;
-- sign ต่างกันเอาค่า mantissa มาลบกัน
                else
                temp_sum := temp_num1 - temp_num2 ;
                sumout <= temp_sum ;
                add_finish <= '1' ;
```

```
                end if ;
        end if ;
else
add_finish <= '0';
end if ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.3 โปรแกรม shift

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity shift is
port (
    sub_control : in std_logic_vector (8 downto 0) ;
    c_in  : in std_logic_vector (32 downto 0) ;
    shift_out : out std_logic_vector (31 downto 0) ;
    clock , shift_en , rst_shift : in std_logic ;
    finish_out : out std_logic ) ;
end shift;
architecture Behavioral of shift is
begin
process(clock , sub_control , c_in , shift_en , rst_shift)
-- ตัวแปรสำหรับเก็บค่า exponent ที่ต่างกัน
variable sub_temp  : std_logic_vector(4 downto 0) ;
begin
if (clock = '1' and clock'event ) then
        if(rst_shift='0') then
                if(shift_en = '1') then
```

```
-- ค่า mantissa มี 23 บิตดังนั้นเลื่อนค่าได้ไม่เกิน 2 ยกกำลัง 5
                    sub_temp := sub_control (4 downto 0);
-- เลื่อนบิตตามจำนวนสถานะที่รับเข้ามา
-- แสดงการเลื่อนค่าเพียง 8 ระดับจากทั้งหมด 23 ระดับ
                    if (sub_temp="00000")then
                    shift_out <= c_in(31 downto 0);
                    finish_out <= '1';
                    elsif (sub_temp="00001")then
                    shift_out <= "0"& c_in(31 downto 1);
                    finish_out <= '1';
                    elsif (sub_temp="00010")then
                    shift_out <= "00"& c_in(31 downto 2);
                    finish_out <= '1';
                    elsif (sub_temp="00011")then
                    shift_out <= "000"& c_in(31 downto 3);
                    finish_out <= '1';
                    elsif (sub_temp="00100")then
                    shift_out <= "0000"& c_in(31 downto 4);
                    finish_out <= '1';
                    elsif (sub_temp="00101")then
                    shift_out <= "00000"& c_in(31 downto 5);
                    finish_out <= '1';
                    elsif (sub_temp="00110")then
                    shift_out <= "000000"& c_in(31 downto 6);
                    finish_out <= '1';
                    elsif (sub_temp="00111")then
                    shift_out <= "0000000"& c_in(31 downto 7);
                    finish_out <= '1';
                    end if ;
                end if ;
        end if;
end if;
```

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

โปรแกรม ง.4 โปรแกรม normal (registor)

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity normalize is
port (
    numb : in std_logic_vector (32 downto 0) ;
    ex : in std_logic_vector (7 downto 0) ;
    signbit , clock , rst_norm  : in std_logic  ;
    exit_n : out std_logic ;
    normal_sum : out std_logic_vector (31 downto 0) ) ;
end normalize;
architecture Behavioral of normalize is
begin
process (clock , rst_norm , numb , ex ,signbit)
variable temp_exp : std_logic_vector (7 downto 0) ;
begin
if (rst_norm = '1') then
 normal_sum <= "00000000000000000000000000000000";
 exit_n <= '0' ;
--- รับข้อมูลมาพัก ,ตรวจการเกิด overflow/underflow
-- จัดเรียงผลลัพธ์ที่ได้จากการคำนวณทั้ง 32 บิตและส่งผลลัพธ์ออกไป
elsif (clock='1'and clock'event) then
        temp_exp := ex (7 downto 0) ;
        if (numb = "00000000000000000000000000000000")then
        normal_sum <= "00000000000000000000000000000000";
        exit_n <= '1' ;
        elsif(numb(32) = '0' and numb(31) = '0' and numb(30) ='0' and numb(29)='0'and
numb(28)='0' and numb(27)='0' and numb(26)='0' and numb(25)='0' and numb(24)='1')then
```

```
temp_exp := temp_exp - "00000111" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(24 downto 2) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '0' and numb(30) ='0' and numb(29)='0'and
numb(28)='0' and numb(27)='0' and numb(26)='0' and numb(25)='1')then
temp_exp := temp_exp - "00000110" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(25 downto 3) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '0' and numb(30) ='0' and numb(29)='0'and
numb(28)='0' and numb(27)='0' and numb(26)='1')then
temp_exp := temp_exp - "00000101" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(26 downto 4) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '0' and numb(30) ='0' and numb(29)='0'and
numb(28)='0' and numb(27)='1')then
temp_exp := temp_exp - "00000100" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(27 downto 5) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '0' and numb(30) ='0' and numb(29)='0'and
numb(28)='1')then
temp_exp := temp_exp - "00000011" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(28 downto 6) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '0' and numb(30) ='0' and numb(29)='1')then
temp_exp := temp_exp - "00000010" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(29 downto 7) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '0' and numb(30)= '1')then
temp_exp := temp_exp - "00000001" ;
normal_sum <= signbit & temp_exp(7 downto 0) & numb(30 downto 8) ;
exit_n <= '1' ;
elsif(numb(32) = '0' and numb(31) = '1')then
```

```
        normal_sum <= signbit & temp_exp(7 downto 0) & numb(31 downto 9) ;
        exit_n <= '1' ;
        elsif(numb(32) = '1' )then
        temp_exp := temp_exp + "00000001" ;
        normal_sum <= signbit & temp_exp(7 downto 0) & numb(32 downto 10) ;--check!!
        exit_n <= '1' ;
        end if;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.5 โปรแกรม manexp

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity manexp is
 port(
    num_mux , num_rom : in std_logic_vector(31 downto 0) ;
    clock : in std_logic ;
    mant_temp : out std_logic_vector(45 downto 0) ;
    exp_mult : out std_logic_vector(7 downto 0) );
end manexp;
architecture Behavioral of manexp is
begin
process(num_mux , num_rom , clock)
variable temp1 , temp2 : std_logic_vector(22 downto 0) ;
variable exp_mux , exp_rom : std_logic_vector(6 downto 0) ;
variable mux_temp , rom_temp , diff_exp: std_logic_vector(7 downto 0) ;
begin
if (clock = '1' and clock'event )then
-- แยกค่า exponent และ mantissa ออกจากกัน
        temp1 := num_mux(22 downto 0) ;
```

```
        temp2 := num_rom(22 downto 0) ;
        exp_mux := num_mux (29 downto 23) ;
        exp_rom := num_rom (29 downto 23) ;
        mux_temp := '0' & exp_mux(6 downto 0) ;
        rom_temp := '0' & exp_rom(6 downto 0) ;
-- นำค่า mantissa มาคูณกัน และ นำค่า exponent มาบวกกันพร้อมกับส่งผลออกไป
                if(num_mux(30) ='0')then
                diff_exp := "01111111"- mux_temp;
                mant_temp <= temp1 * temp2 ;
                exp_mult <= rom_temp - diff_exp;
                elsif(num_rom(30) ='0')then
                diff_exp := "01111111"- rom_temp;
                mant_temp <= temp1 * temp2 ;
                exp_mult <= mux_temp - diff_exp;
                elsif(mux_temp = "00000000" and rom_temp = "00000000")then
                mant_temp <= temp1 * temp2 ;
                exp_mult <= "00000001";
                else
                mant_temp <= temp1 * temp2 ;
                exp_mult <= mux_temp + rom_temp ;
                end if;
 end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.6 โปรแกรม normal

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity normal is
port(
    mant : in std_logic_vector(45 downto 0) ;
```

```
    clock_main , zero , sign : in std_logic ;
    expo : in std_logic_vector(7 downto 0) ;
    mult_out : out std_logic_vector(31 downto 0) ) ;
end normal;
architecture Behavioral of normal is
begin
process(mant, expo , clock_main)
variable sign_mult : std_logic ;
variable exp_mult : std_logic_vector(7 downto 0) ;
variable res_temp : std_logic_vector(31 downto 0) ;
begin
if (clock_main = '1' and clock_main'event )then
sign_mult := sign ;
exp_mult := expo (7 downto 0) ;
-- ตรวจพบว่าข้อมูลที่เข้ามามีค่าเป็น 0 ส่งค่า 0 ออกไป
        if (zero = '1'or mant = "00000000000000000000000000000000000000000000000000") then
                mult_out <= "00000000000000000000000000000000" ;
        else
-- ตรวจสอบการเกิด overflow/underflow พร้อมทั้งตัดผลคูณจาก 46 บิตเหลือ 23 บิต
-- รวบรวมผลให้ครบ 32 บิตและส่งผลออกไป
                if(mant(45) = '1') then
                exp_mult := exp_mult + 1 ;
                if (exp_mult >="01000000" or exp_mult="11111111")then
                res_temp := sign_mult &'0'&exp_mult(6 downto 0) & mant(45 downto 23) ;
                mult_out <= res_temp(31 downto 0) ;
                else
                res_temp := sign_mult &'1'&exp_mult(6 downto 0) & mant(45 downto 23) ;
                mult_out <= res_temp(31 downto 0) ;
                end if ;
        elsif(mant(45) = '0') then
        if (exp_mult >="01000000" or exp_mult="11111111")then--(exp_mult <="01111111" or
                res_temp := sign_mult & '0'&exp_mult(6 downto 0) & mant(44 downto 22) ;
```

```
                mult_out <= res_temp(31 downto 0) ;
                else
                res_temp := sign_mult & '1'&exp_mult(6 downto 0) & mant(44 downto 22) ;
                mult_out <= res_temp(31 downto 0) ;
                end if;
        end if;
end if ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.7 โปรแกรม num_zero

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity num_zero is
port(
    num_mux , num_rom : in std_logic_vector(31 downto 0) ;
    clock_main : in std_logic ;
    sign_mult , t : out std_logic ) ;
end num_zero;
architecture Behavioral of num_zero is
begin
process(num_mux , num_rom , clock_main)
begin
-- หาค่า sign ของผลลัพธ์
if (clock_main = '1' and clock_main'event )then
-- sign เหมือนกัน signout =0
        if (num_mux(31) = '1' and num_rom(31) = '1') then
        sign_mult <= '0' ;
        elsif (num_mux(31) = '0' and num_rom(31) = '0') then
        sign_mult <= '0' ;
```

```
-- sign เหมือนต่างกัน signout =1
        else
        sign_mult <= '1' ;
        end if ;
if (num_mux = 0) then
        t <= '1' ;
        elsif (num_rom = 0) then
        t <= '1' ;
        else
        t <= '0' ;
        end if ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.8 โปรแกรม sign_div

```
library ieee ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all;
entity sign_div is
port (d_input : in std_logic_vector(31 downto 0);
      clock,div: in std_logic ;
      sign : out std_logic ;
      man_out : out std_logic_vector(22 downto 0) ) ;
end sign_div ;
architecture rtl of sign_div is
begin
process (clock,d_input,div)
begin
if(clock='0'and clock'event) then
if(div='1')then
```

```
—ส่งค่า sign และค่า mantissa ของข้อมูลที่รับเข้ามาออกไป
--หารด้วยค่าคงที่ไม่ต้องใช้ค่า sign กับ mantissa ในการคำนวณ
        if(d_input(31) ='1')then
        sign <= '1';
        man_out <= d_input(22 downto 0) ;
        else
        sign <= '0';
        man_out <= d_input(22 downto 0) ;
        end if ;
else
sign <= d_input(31);
man_out <= d_input(22 downto 0) ;
end if ;
end if;
end process ;
end rtl ;
```

โปรแกรม ง.9 โปรแกรม sub_exp

```
library ieee ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all;
entity sub_exp is
port (data_input : in std_logic_vector(31 downto 0);
      clock,div : in std_logic ;
      exp_out : out std_logic_vector(7 downto 0) ) ;
end sub_exp ;
architecture rtl of sub_exp is
begin
process (clock,data_input,div)
variable y : std_logic_vector(7 downto 0);
variable x : std_logic_vector(6 downto 0);
```

```
begin
if(clock='0'and clock'event) then
y := data_input(30 downto 23);
if(div='1')then
-- การหารข้อมูลด้วยค่า 256 = 2 ยกกำลัง 8
-- ลบค่า exponent ของข้อมูลที่เข้ามาด้วย 8
        if(y > "10001000")then
                x := data_input(29 downto 23) - "0001000";
                exp_out <= '1'& x;
        elsif(y < "10001000" and y > "00000000")then
                x := data_input(29 downto 23) - "0001000";
                exp_out <= '0'& x;
        elsif(y = "00000000")then
                exp_out <= "00000000";
        else
                exp_out <= "01111111";
        end if;
else
        exp_out <= y;
end if;
end if;
end process ;
end rtl ;
```

โปรแกรม ง.10 โปรแกรม div

```
library ieee ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all;
entity normal_div is
port (man : in std_logic_vector(22 downto 0);
      exp : in std_logic_vector(7 downto 0);
```

```
    clock,s : in std_logic ;
    out_data : out std_logic_vector(31 downto 0) ) ;
end normal_div ;
architecture rtl of normal_div is
begin
process (clock,s,exp,man)
begin
-- เป็นการรับข้อมูลทั้งหมดเข้ามาพักไว้และส่งออกไป
if(clock='1'and clock'event) then
        out_data <= s & exp & man;
end if;
end process ;
end rtl ;
```

โปรแกรม ง.11 โปรแกรม stage_sc

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity stage_se is
port (
   add_incr , add_clear : in std_logic ;
   st_rom : out std_logic_vector(7 downto 0) ) ;
end stage_se;
architecture Behavioral of stage_se is
begin
process(add_clear , add_incr)
variable count : std_logic_vector(7 downto 0) ;
begin
if(add_clear = '0')then
count := "00000000" ;
st_rom <= "00000000" ;
-- วงจรนับขนาด 8 บิตสำหรับสร้างที่อยู่ของข้อมูล
```

```
elsif (add_incr'event and add_incr = '1') then
st_rom <= (count + 1) ;
count := count + 1 ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.12 โปรแกรม shift_register

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
    entity  rotate_new is
    port( st :in std_logic_vector(2 downto 0);
          clock : in std_logic ;
          a :in std_logic_vector(7 downto 0);
          c :out std_logic_vector(7 downto 0));
     end rotate_new;
architecture  rtl of rotate_new is
begin
process(a,st)
begin
-- วงจรเลื่อนบิตจากบิตด้านขวามาทางด้านซ้ายตามจำนวนค่าสถานะที่รับเข้ามา
-- บิตทางด้านซ้ายที่ถูกเลื่อนจะนำมาต่อท้ายบิตทางด้านขวา
        if (clock='0' and clock'event) then
                        if (st="000") then
                        c <= a(7 downto 0) ;
                        elsif (st="001") then
                        c <= a(6)&a(5)&a(4)&a(3)&a(2)&a(1)&a(0)&a(7) ;
                        elsif (st="010") then
                        c <= a(5)&a(4)&a(3)&a(2)&a(1)&a(0)&a(7)&a(6) ;
                        elsif (st="011") then
                        c <= a(4)&a(3)&a(2)&a(1)&a(0)&a(7)&a(6)&a(5) ;
```

```
                elsif (st="100") then
                c <= a(3)&a(2)&a(1)&a(0)&a(7)&a(6)&a(5)&a(4) ;
                elsif (st="101") then
                c <= a(2)&a(1)&a(0)&a(7)&a(6)&a(5)&a(4)&a(3) ;
                elsif (st="110") then
                c <= a(1)&a(0)&a(7)&a(6)&a(5)&a(4)&a(3)&a(2) ;
                elsif (st="111") then
                c <= a(0)&a(7)&a(6)&a(5)&a(4)&a(3)&a(2)&a(1) ;
            end if ;
            end if ;
    end process;
  end Behavioral;
```

โปรแกรม ง.13 โปรแกรม reverse

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reverse is
 port (
    addr_in : in std_logic_vector(7 downto 0) ;
    clock,io_mode : in std_logic ;
    addr_out : out std_logic_vector(7 downto 0) ) ;
end reverse;
architecture Behavioral of reverse is
begin
process(addr_in , clock,io_mode)
begin
--เรียงลำดับของสัญญาณใหม่จากหลังไปหน้า
if (clock='1' and clock'event) then
if io_mode='1' then
```

```
addr_out <= addr_in(0)& addr_in(1)&addr_in(2)& addr_in(3)& addr_in(4)&addr_in(5)&
addr_in(6)&addr_in(7);
else
addr_out <= addr_in(7 downto 0);
end if ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.14 โปรแกรม add_sign

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity add_sign is
 port (
     a : in std_logic_vector (7 downto 0) ;
     half : in std_logic ;
     address : out std_logic_vector (8 downto 0)) ;
end ;
architecture Behavioral of add_sign is
begin
process (a , half)
begin
-- วงจรใส่ 0/1 ลงไปในบิตหน้าสุด
if (half = '1')then
--สำหรับข้อมูลค่าจริง
          address <= '0' & a (7 downto 0) ;
else
-- สำหรับข้อมูลค่าจินตภาพ
          address <= '1' & a (7 downto 0) ;
end if ;
```

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

โปรแกรม ง.15 โปรแกรม gen_clock

```
library ieee ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_1164.all ;
entity clock_gen is
port (
    clock , clear : in std_logic ;
    clock_out : out std_logic ) ;
end clock_gen ;
architecture rtl of clock_gen is
begin
process (clock , clear)
begin
-- ส่งสัญญาณนาฬิกาออกไปเมื่อ clear เป็น 1 clear เป็น 0 ไม่มีการส่งสัญญาณนาฬิกาออกไป
if clear = '0'then
        clock_out <= '0';
else
        clock_out <= clock;
end if;
end process ;
end rtl ;
```

โปรแกรม ง.16 โปรแกรม frequency divider

```
library ieee ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_1164.all ;
entity gen_choose is
port (
```

```
    clock,clear : in std_logic ;
    choose : out std_logic ) ;
end gen_choose ;
architecture rtl of gen_choose is
begin
process (clock ,clear)--,c0_en,preset )
variable cnt : unsigned (4 downto 0) ;
variable buff : std_logic;
begin
if clear = '0'then
        cnt:="00000";
        buff := '1';
        choose <= '0';
elsif (clock='0'and clock'event) then
-- วงจรหารความถี่ลงครึ่งหนึ่ง โดยการนับ 0 ถึง 1 แล้วส่งสัญญาณออกไป และเมื่อนับถึง 1 ให้ clear
ค่าเป็น 0
if cnt = 1 then
        cnt:="00001";
        buff := not(buff);
else
        cnt := cnt +1;
end if;
choose <= buff;
end if;
end process ;
end rtl ;
```

โปรแกรม ง.17 โปรแกรม load 1



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

```
entity buff1 is
 port (
     data_in : in std_logic_vector(31 downto 0) ;
     clock : in std_logic ;
     data_out : out std_logic_vector(31 downto 0) ) ;
end buff1;
architecture Behavioral of buff1 is
begin
process(data_in , clock)
begin
--ปล่อยข้อมูลที่เข้ามาออกไปเมื่อ clock เป็น 1
if (clock='1' and clock'event) then
data_out <= data_in ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.18 โปรแกรม load 0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity buff1 is
 port (
     data_in : in std_logic_vector(31 downto 0) ;
     clock : in std_logic ;
     data_out : out std_logic_vector(31 downto 0) ) ;
end buff1;
architecture Behavioral of buff1 is
begin
process(data_in , clock)
begin
```

```
--ปล่อยข้อมูลที่เข้ามาออกไปเมื่อ clock เป็น 0
if (clock='0' and clock'event) then
data_out <= data_in ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.19 โปรแกรม control

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity con_fft is
port (
    reset_fft,clock_main : in std_logic ;
    end_w,end_r,start_w : in std_logic ;
    w,r,reset,enbl,clear,io_mode : out
    std_logic;
    stage_out : out std_logic_vector(2
    downto 0));
end con_fft;
architecture Behavioral of con_fft is
type state is
(rst1,rst2,rst3,rst4,rst5,rst6,rst7,rst8,rst9,
rst10,rst11,rst12,rst13,rst14,rst15,rst16,
rst17,rst18,rst19,rst20,rst21,rst22,rst23,
rst24,rst25,rst26,rst27) ;
signal current_state , next_state : state ;
begin
process
(current_state,end_w,end_r,start_w,clock_m
ain)
begin
if(clock_main='1'and clock_main'event)then
case current_state is
 when rst1 =>
-- io_mode = 1 เป็นการรับข้อมูลเข้ามา
        io_mode <= '1';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
--  ค่า stage_out  ใช้สำหรับกำหนดสถานะ
สำหรับการคำนวณ
        stage_out <= "000";
        next_state <= rst2 ;
 when rst2 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "111";
```

```
        next_state <=rst3 ;
        else
  next_state <= rst2 ;
  end if ;
when rst3 =>
        reset <= '0';
        clear <= '1';
 if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "111";
  next_state <= rst4 ;
  else
  next_state <= rst3 ;
  end if ;
when rst4 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "111";
  next_state <= rst5 ;
  else
  next_state <= rst4 ;
  end if ;
when rst5 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "110";
  next_state <= rst6 ;
  else
  next_state <= rst5 ;
  end if ;
when rst6 =>
reset <= '0';
        clear <= '1';
 if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "110";
 next_state <= rst7 ;
 else
 next_state <= rst6 ;
 end if ;
```

```
when rst7 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "110";
  next_state <= rst8 ;
  else
  next_state <= rst7 ;
  end if ;
        when rst8 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "101";
  next_state <= rst9 ;
  else
  next_state <= rst8 ;
  end if ;
 when rst9 =>
reset <= '0';
        clear <= '1';
if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "101";
  next_state <= rst10 ;
  else
  next_state <= rst9 ;
  end if ;
when rst10 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "101";
  next_state <= rst11 ;
  else
  next_state <= rst10 ;
  end if ;
        when rst11 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
```

```
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "100";
  next_state <= rst12 ;
  else
  next_state <= rst11 ;
  end if ;
 when rst12 =>
reset <= '0';
        clear <= '1';
 if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "100";
  next_state <= rst13 ;
  else
  next_state <= rst12 ;
  end if ;
 when rst13 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "100";
  next_state <= rst14 ;
  else
  next_state <= rst13 ;
  end if ;
        when rst14 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "011";
  next_state <= rst15 ;
  else
  next_state <= rst14 ;
  end if ;
 when rst15 =>
reset <= '0';
        clear <= '1';
 if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
```

```
        enbl <= '1';
        clear <= '1';
        stage_out <= "011";
  next_state <= rst16 ;
  else
  next_state <= rst15 ;
  end if ;
when rst16 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "011";
  next_state <= rst17 ;
  else
  next_state <= rst16 ;
  end if ;
         when rst17 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "010";
  next_state <= rst18 ;
  else
  next_state <= rst17 ;
  end if ;
 when rst18 =>
reset <= '0';
        clear <= '1';
 if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "010";
  next_state <= rst19 ;
  else
  next_state <= rst18 ;
  end if ;
 when rst19 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "010";
  next_state <= rst20 ;
```

```
else
next_state <= rst19 ;
end if ;
        when rst20 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "001";
next_state <= rst21 ;
else
next_state <= rst20 ;
end if ;
when rst21 =>
reset <= '0';
        clear <= '1';
if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "001";
next_state <= rst22 ;
else
next_state <= rst21 ;
end if ;
when rst22 =>
        reset <= '0';
        clear <= '1';
if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "001";
next_state <= rst23 ;
else
next_state <= rst22 ;
end if ;
        when rst23 =>
        reset <= '0';
        clear <= '1';
        if(end_w = '1')then
        io_mode <= '0';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "000";
next_state <= rst24 ;
else
next_state <= rst23 ;
end if ;
when rst24 =>
```

```
reset <= '0';
        clear <= '1';
 if(start_w = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '1';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "000";
  next_state <= rst25 ;
  else
  next_state <= rst24 ;
  end if ;
when rst25 =>
        reset <= '0';
        clear <= '1';
 if(end_r = '1') then
        io_mode <= '0';
        w <= '1';
        r <= '0';
        reset <= '0';
        enbl <= '1';
        clear <= '1';
        stage_out <= "000";
  next_state <= rst26 ;
  else
  next_state <= rst25 ;
  end if ;
when rst26 =>
reset <= '0';
clear <= '1';
if(end_w = '1')then
        io_mode <= '1';
        w <= '0';
        r <= '1';
        reset <= '1';
        enbl <= '1';
        clear <= '0';--0
        stage_out <= "000";
        next_state <= rst27 ;
  else
  next_state <= rst26 ;
  end if ;
when rst27 =>
reset <= '0';
clear <= '1';
        stage_out <= "000";
 when others =>
  next_state <= rst1 ;
end case ;
end if;
end process ;
process(clock_main , reset_fft)
begin
if(reset_fft = '1') then
current_state <= rst1 ;
elsif (clock_main'event and clock_main =
'0') then
current_state <= next_state ;
end if ;
end process ;
end Behavioral;
```

โปรแกรม ง.20 โปรแกรม connect

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity connect is
port (clock,we,re : in std_logic;
    end_w ,end_r : out std_logic;
    address_w,address_r: in std_logic_vector (8 downto 0));
end connect;
architecture Behavioral of connect is
begin
-- Write Functional Section
--ส่งสัญญาณ end_w = 1 เมื่อเขียนข้อมูล address สุดท้าย
process (clock,address_w,we)
begin
if (clock='0'and clock'event) then
if (we = '1') then
        if(address_w ="011111111")then
                end_w <= '1';
        else
                end_w <= '0';
        end if ;
end if ;
end if ;
end process ;
-- Read Functional Section
--ส่งสัญญาณ end_r = 1 เมื่ออ่านข้อมูล address สุดท้าย
process (clock,address_r,re)
begin
if (clock='0'and clock'event) then
if (re = '1') then
```

```
        if(address_r ="011111111")then
                end_r <= '1';
        else
                end_r <= '0';
        end if ;
end if;
end if;
end process;
end Behavioral;
```

# ภาคผนวก จ.

## โปรแกรมไมโครคอนโทลเลอร์

**ใช้แปลงรูปแบบสัญญาณที่ภาคส่ง**

```
#pragma code
#include <t89c51rd2.h>
void receive_byte();
void send();
void delay(unsigned char x);
void inc_byte();
void Timer0_delay();
void clk_continue();
unsigned char byte,Temp0,Temp1,Temp2,Temp3,TEMP4;
unsigned int cycle;
// ส่งสัญญาณเมื่อส่งข้อมูลออกไปครั้งละ 4 byte ใช้เป็นสัญญาณนาฬิกา
sbit ACK = P3^7;              // Signal show complete 4 byte
// ส่งสัญญาณเมื่อส่งข้อมูลครบ
sbit End_Package = P3^6; // Signal show complete package
// ส่งสัญญาณ enable ให้บอร์ด FPGA เริ่มทำงาน
sbit Enable = P3^5;
main()
{
        Enable=0;
        End_Package=1;
        ACK=1;
// นับข้อมูลทั้งหมด
        cycle=0;          // Package counter
// นับข้อมูลทีละ byte
        byte=0;           // 4 Byte counter
// ให้การทำงานอยู่ใน mode ธรรมดา
        CKCON=CKCON&0x0FE;        // Set to standard mode
```

```
        SCON=0x50;
// รอสัญญาณ interrupt จากพอร์ตอนุกรม
        ES=1;           // Enable serial interrupt
        TCON=0x00;
        TMOD=0x21;      // Timer 1 mode 2 and Timer 0 mode 1
        PS=1;           // Serial highest priority
// ใช้อัตราบอดเรต = 9,600 bit/sec
        TH1=0x0FB;      // 9600 b/s
        TL1=0x0FB;
        TR1=1;
        EA=1;           // Enable global interrupt
        while(1);       // Wait for serial interrupt
}
void receive_byte() interrupt 4
{
        cycle=cycle+1;
        switch(byte)
        {
                case 0:  {
                                Temp0=SBUF;
                                RI=0;
                                inc_byte();
                                }
                                break;
                case 1:  {
                                Temp1=SBUF;
                                RI=0;
                                byte=byte+1;
                                }
                                break;
                case 2: {
                                Temp2=SBUF;
```

```
                        byte++;
                        RI=0;
                        }
                        break;
                case 3:  {
                        Temp3=SBUF;
                        byte=0x00;
                        send();
                        RI=0;}
                        break;
        }
        if(cycle==2048)
        {
                End_Package=0;
                cycle=0;
        delay(10);
                End_Package=1;
                clk_continue();
        }
}
void send()             // Move 4 byte to port0-3
{
        ACK=1;
        P0=Temp0;
        P1=Temp1;
        P2=Temp2;
        P4=Temp3;
        ACK=0;
        Enable=1;
        Timer0_delay();
        ACK=1;          // HAVE CHANGED
}
```

```
void delay(unsigned char x)
{
        unsigned char j;
        while(x--)
        for(j=0;j<20;j++);
}
void inc_byte()
{
        TEMP4=0;
        byte=byte+1;
}
void Timer0_delay()
{
        TH0=0x0F5;
        TL0=0x0FA;
        TR0=1;
        while(TF0==0);
        TF0=0;
}
void clk_continue()
{
        while(1)
        {
                Timer0_delay();
                ACK=~ACK;
        }
}
```



**ใช้แปลงรูปแบบสัญญาณที่ภาครับ**

```
#pragma code
#include <t89c51rd2.h>
unsigned char Temp0,Temp1,Temp2,Temp3;
```

```
void receive_data();
void send();
//use P3.2 as External interrupt 0
main()
{
        CKCON=CKCON&0x0FE;        // Set to standard mode
// รอ interrupt จากบอร์ด FPGA
        IE=0x85;          // Enable external interrupt 0
        SCON=0x40;
        TCON=0x01;
        TMOD=0x21;    // Timer 1 mode 2 and Timer 0 mode 1
// ส่งข้อมูลให้คอมพิวเตอร์ที่อัตราบอดเรต = 9,600 bit/sec
        TH1=0x0FB;    // 9600 b/s
        TL1=0x0FB;
        TR1=1;
        while(1);
}
void receive_data() interrupt 0
{
        Temp0=P0; Temp1=P1; Temp2=P2; Temp3=P4;
        send();
}
void send()
{
        SBUF=Temp0;
        while(TI==0);
        TI=0;
        SBUF=Temp1;
        while(TI==0);
        TI=0;
        SBUF=Temp2;
        while(TI==0);
```

```
        TI=0;
        SBUF=Temp3;
        while(TI==0);
        TI=0;
}
```

# ภาคผนวก ฉ.

## ผลจากการทดสอบการทำงานเทียบกับผลจากโปรแกรม Matlab

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 9 | 1100000100100101110010111111111011 | 1100000100100101110110001010111 0 |
| | 1100000001011111110110001010111 0 | 1100000001011111101100010101101 1 |
| 10 | 0100000001000000110001001001110 0 | 0100000001000000000000000000000 0 |
| | 0100000100011100010011110000111 0 | 0100000100011100010110000011110 01 |
| 11 | 1011111110111100101101011101110 1 | 1011111110111100100000010110111 1 |
| | 1100000000001100110011001100 1101 | 1100000000001101001010111101010 0 |
| 12 | 1100000010011001100110011001101 0 | 1100000010011001101101001010001 0 |
| | 0100000100000000001010001111011 0 | 0100000100000000000000000000000 0 |
| 13 | 0100000100001000001101101001100 0 | 0100000100001000001011110001101 1 |
| | 1011111111100001111010111000010 1 | 1011111111100010100011000001010 1 |
| 14 | 1100000011111001011100001010010 0 | 1100000011111001011000000100001 0 |
| | 1100000001100001010110110101011 1 | 1100000001100001001110101001001 1 |
| 15 | 0100000010111111111011011111101 0 | 0100000011000000000000000000000 0 |
| | 0100000000110100111000000111011 0 | 0100000000110101000100011001110 1 |
| 16 | 1011110110010101100000010000011 0 | 1011110110011011011100010111011 0 |
| | 1100000011110010010101101010001 | 1100000011110010010011000011000 0 |
| 17 | 1100000010101110010011000011000 0 | 1100000010101110001111111110011 0 |
| | 0100000100010000001100011111100 1 | 0100000100010000000000000000000 0 |
| 18 | 0100000000110010010000001011100 0 | 0100000000110001111111110010111 0 |
| | 1011111111110101100111101000010 0 | 1011111111110101001001010100011 0 |
| 19 | 1100000011000101011100001010010 0 | 1100000011000101100110011001101 0 |
| | 1100000010101110000111110010000 1 | 1100000010101110001111111110011 0 |
| 20 | 0100000100010000011000100100111 0 | 0100000100010000000000000000000 0 |
| | 0100000011010011010001101101110 0 | 0100000011010011100000000011010 0 |
| 21 | 1011111110111100110001100011111 1 | 1011111110111100100000010110111 1 |
| | 1100000100011000110111110011101 1 | 1100000100011000111001110000010 0 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 22 | 11000000101001000010100011110110 | 11000000101001001010000101100010 |
| | 11000000101000011000101110101100 | 11000000101000010110100110101011 |
| 23 | 00111111100000101011100111110101 | 00111111100000000000000000000000 |
| | 01000000111111001011000111000100 | 01000000111111001001011100011101 |
| 24 | 00111101100101000100011001110100 | 00111101100011100010000110010110 |
| | 11000000100111111010100100101010 | 11000000100111111111101100010110 |
| 25 | 11000000000011101101101010111010 | 11000000000011110000101111100001 |
| | 01000000101000010000011000100101 | 01000000101000000000000000000000 |
| 26 | 01000000111110111111001110110110 | 01000000111111001001011110001101 |
| | 00111111101110011100000011101100 | 00111111101110111010110001110001 |
| 27 | 11000000011001101110001011101011 | 11000000011001101001000100000000 |
| | 11000000101000110011001100110011 | 11000000101000111001011111110110 |
| 28 | 01000000100000000111101011100001 | 01000000100000000000000000000000 |
| | 01000000100001000100001100101101 | 01000000100000111001111101010110 |
| 29 | 00111111000001011001101100111101 | 00111111000000110000101111100001 |
| | 11000000101111100110011001100110 | 11000000101111101000010110001000 |
| 30 | 11000000100110010000111111111001 | 11000000100110001111000000000111 |
| | 01000000000000010101100000010000 | 01000000000000000000000000000000 |
| 31 | 01000001000000110101110000101001 | 01000001000000110011110101110001 |
| | 10111111101011111111110010111010 | 10111111011000101000100011001110 |
| 32 | 11000000001100100010011010000001 | 11000000001100101100101001011000 |
| | 11000000101001000010100011110110 | 11000000101001000101110010010010 |
| 33 | 01000000110111111110110111111010 | 01000000111000000000000000000000 |
| | 01000001000000110011001100110011 | 01000001000000110011110101110001 |
| 34 | 10111110100101001111110111110100 | 10111110100110000011101011111110 |
| | 11000000111001000011110010011111 | 11000000111001000010001100111010 |
| 35 | 11000000101000111110011101101101 | 11000000101000111001011111110110 |
| | 01000000100000000110101001111111 | 01000000100000000000000000000000 |
| 36 | 01000000101101011001110000001111 | 01000000101101011101010101100111 |
| | 10111111011101010111100011101010 | 10111111011101011111011011111101 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 37 | 11000000101110100000010000011001 | 11000000101110011101111001101010 |
| | 11000000101101111101001000100000 | 11000000101110000010010000001011 |
| 38 | 01000000101000001111010111000011 | 01000000101000000000000000000000 |
| | 01000001000000110101110000101001 | 01000001000000110011110101110001 |
| 39 | 10111111100011100101011000000100 | 10111111100011011001011111110110 |
| | 11000000001100101000111101011100 | 11000000001100101100101001011000 |
| 40 | 11000000110000100011110101110001 | 11000000110000100000011101011111 |
| | 01000000010000001101010011111110 | 01000000010000000000000000000000 |
| 41 | 01000000111010001010001111010111 | 01000000111010001101000000010100 |
| | 00111111100010111000010100011111 | 00111111100011001100001011111000 |
| 42 | 11000000111010111000010100011111 | 11000000111010110011100000011101 |
| | 11000000100110011110101110000101 | 11000000100110011011010010100010 |
| 43 | 01000000100000000010010000001011 | 01000000100000000000000000000000 |
| | 01000000001101000111101011100001 | 01000000001101010001000110011101 |
| 44 | 11000000001101110000101000111101 | 11000000001101111010010111100011 |
| | 11000000110110101010010101111010 | 11000000110110101101011010100001 |
| 45 | 10111111111111000110110000010001 01 | 10111111111111001100111001110000 |
| | 01000000110000000101101000011101 | 01000000110000000000000000000000 |
| 46 | 01000000110100101000011111111101 | 01000000110100101011101110011001 |
| | 11000000100101100110011001100110 | 11000000100101101010111001111101 |
| 47 | 11000001000000000000000100111011 | 11000001000000000011101010010011 |
| | 11000000001110100011110101110001 | 11000000001110011010110101000011 |
| 48 | 01000000010000001010001000110100 | 01000000010000000000000000000000 |
| | 01000001000101110100010110100010 | 01000001000101110110011011001111 |
| 49 | 10111111101110110111000101110110 | 10111111101111001000000101101111 |
| | 11000001000000100110011001100110 | 11000001000000101000110110111001 |
| 50 | 10111111011101111100000110111110 | 10111111011110011010000000100111 |
| | 01000000101000000111101011100001 | 01000000101000000000000000000000 |
| 51 | 00111111111001111010111000010100 | 00111111111001000110100001000101 |
| | 00111110100110100010100111000111 | 00111110100101001100100110000110 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 52 | 11000000011001100100001001011011 | 11000000011001101001000100000000 |
| | 11000000100001010110101011101000 | 11000000100001010010100101011111 |
| 53 | 01000000010000001000000101101111 | 01000000010000000000000000000000 |
| | 01000001000010000101000110000011 | 01000001000010000010111100011011 |
| 54 | 10111111100111111010111000010100 | 10111111101000000011000100100111 |
| | 11000001000110010000010110111100 | 11000001000110001110011100000100 |
| 55 | 10111111110011001111000011011000 | 10111111110011101111110100100010 |
| | 01000000000000001010100011000001 | 01000000000000000000000000000000 |
| 56 | 01000001000010000101101010000110 | 01000001000010000010111100011011 |
| | 10111111110101100000011101011111 | 10111111110110001101000110110111 |
| 57 | 11000000111101110001101101110001 | 11000000111101101111001001111100 |
| | 11000000111101001000001111100100 | 11000000111101000011110101110001 |
| 58 | 01000000010000001010001110101101 | 01000000010000000000000000000000 |
| | 01000000100011000111010001010100 | 01000000100011001011111011100000 |
| 59 | 00111111011100100110100000001010 | 00111111011110100010011010000001 |
| | 11000000101010110101011100111111 | 11000000101010111011011001000110 |
| 60 | 11000000100110011111010101011010 | 11000000100110011011010010100010 |
| | 01000000100000000101001010111101 | 01000000100000000000000000000000 |
| 61 | 01000001000000110111001010110000 | 01000001000000110011110101110001 |
| | 10111111100111111011010010100010 | 10111111101000000011000100100111 |
| 62 | 11000001000011111001000100000000 | 11000001000011110111111101100011 |
| | 00111111111010010101010011001010 | 00111111111010100100000010111000 |
| 63 | 01000000111000000101101011101110 | 01000000111000000000000000000000 |
| | 01000000101010110111000101110110 | 01000000101010110010110101110111 |
| 64 | 11000000010010100010100111000111 | 11000000010010101010101100110110 |
| | 11000000101000000001001000000110 | 11000000100111111111101100010110 |
| 65 | 11000000101000110100010001100111 | 11000000101000111001011111110110 |
| | 00111111100000010100101011111010 | 00111111100000000000000000000000 |
| 66 | 01000000010111101011100111110101 | 01000000010111100010100000100100 |
| | 10111101101001100001011111000010 | 10111101100110110111000101110110 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 67 | 1100000011000101001110110110010 | 1100000011000101100110011001010 |
| | 0100000011000000010100101011101 | 0100000011000000000000000000000 |
| 68 | 0100000100011100011000111111001 | 0100000100011100010110000111100 1 |
| | 1011111110010110100010100111001 0 | 1011111110010111010011110000111 0 |
| 69 | 1100000011101000111000111011101 | 1100000011101000111111110010111 |
| | 1100000010011010001101101110001 1 | 1100000010011010011110010011111 0 |
| 70 | 0100000100000000001101011010100 0 | 0100000100000000000000000000000 |
| | 0100000100001000000001001000000 1 | 0100000100001000001011110001101 1 |
| 71 | 1011111111011000000001101000111 0 | 1011111111011000110100011011011 1 |
| | 1100000010011111100110110011110 1 | 1100000010011111111110110001011 0 |
| 72 | 1011111101111100100100011101000 1 | 1011111101111001101000000010011 1 |
| | 0100000000000000101101011101110 1 | 0100000000000000000000000000000 |
| 73 | 0100000000001110000101111100001 0 | 0100000000001101100000101010101 0 |
| | 1011111110000011100011000101010 0 | 1011111110000011110111011001100 0 |
| 74 | 1100000011011000001101100001000 1 | 1100000011011000011010001101110 0 |
| | 1100000010101101111110100100010 0 | 1100000010101110001111111111001 1 0 |
| 75 | 0100000010011111111100000110111 1 | 0100000010100000000000000000000 |
| | 0100000001110001101111000000001 0 | 0100000001110001111011011001100 |
| 76 | 0011111100101010001101101110001 1 | 0011111100101000001111100100001 0 |
| | 1100000001110000001011111000001 1 | 1100000001101111101110110011000 |
| 77 | 1100000100000011111010010111100 1 | 1100000100000100000000100000110 0 |
| | 0100000011100000010100010001101 0 | 0100000011100000000000000000000 |
| 78 | 0100000010000010100000001001110 1 | 0100000010000010110110101011101 0 |
| | 1011111010111011011100010111011 0 | 1011111010111111000001101111011 1 |
| 79 | 1100000011000101010110011011010 0 | 1100000011000101100110011001101 0 |
| | 1100000001110100011110010011111 0 | 1100000001110101000000100111011 |
| 80 | 0100000011100000011000010111110 0 | 0100000011100000000000000000000 |
| | 0100000010101011110011100000011 1 | 0100000010101011111100100001001 1 |
| 81 | 1100000000101110111011001100000 0 | 1100000000101110010110010101001 1 |
| | 1100000010011001000001111100100 0 | 1100000010011000111001110000010 0 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 82 | 11000000100110011111110110001011 | 11000000100110011011010010100010 |
| | 01000000001111111110100100010000 | 01000000010000000000000000000000 |
| 83 | 01000001000011010000110001001010 | 01000001000011010010000011000101 |
| | 10111110100100111000000111011000 | 10111110100110000001110101111110 |
| 84 | 11000000011001110001110111100111 | 11000000011001101001000100000000 |
| | 11000000001001100111100001101100 | 11000000001001011110011010011011 |
| 85 | 01000000010000000101000111110110 | 01000000010000000000000000000000 |
| | 01000000001101011000010001001101 | 01000000001101010001000110011101 |
| 86 | 00111111011101100101100101001011 | 00111111011101000100110100000001 |
| | 11000000111001101100101001011000 | 11000000111001101001000100000000 |
| 87 | 10111101010100011011011100010111 | 10111101010001001001101110100110 |
| | 01000000111000000111111011111010 | 01000000111000000000000000000000 |
| 88 | 01000000101101010101001001010100 | 01000000101101010001000011001011 |
| | 10111110110111100100001001011011 | 10111110111000101000111101011100 |
| 89 | 11000001000010111101011010100001 | 11000001000010111111010101011010 |
| | 11000000100011101100101001011000 | 11000000100011110000110010110011 |
| 90 | 01000000010000001010100011000001 | 01000000010000000000000000000000 |
| | 01000000111100001110001011101011 | 01000000111100010010101000110000 |
| 91 | 10111111111111111111110010111001 | 10111111111111101101110001011101 |
| | 11000001000100001000010100011111 | 11000001000100001011011001000110 |
| 92 | 10111111111111011000010101010011 | 10111111111111001100111001110000 |
| | 01000001000100000011000111111001 | 01000001000100000000000000000000 |
| 93 | 01000000101011000010100000100100 | 01000000101010111111001000010011 |
| | 00111111000001010010010101000110 | 00111111000000110000101111100001 |
| 94 | 11000000101101111011011100010111 | 11000000101101110111000010100100 |
| | 11000000110101011011000010001010 | 11000000110101011100111000000111 |
| 95 | 01000000100000001010110000001000 | 01000000100000000000000000000000 |
| | 01000000000001100010011010000001 | 01000000000001011111110010111010 |
| 96 | 00111101100100111010100100101010 | 00111101100011100010000110010110 |
| | 11000001001101100011010000000101 | 11000001001101100101010001100001 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 97 | 11000000100011110101101101010111 | 11000000100011110000110010110011 |
| | 01000000110000001010010010101001 | 01000000110000000000000000000000 |
| 98 | 01000000101101011011011100010111 | 01000000101101011010101011001111 |
| | 10111110001010101100100001100000 | 10111110000110110111000101110110 |
| 99 | 11000000101100110010101111010100 | 11000000101100101100101001011000 |
| | 11000000001001011011001111010000 | 11000000001001011110011010011011 |
| 100 | 00111111111111111101101111110101 | 01000000000000000000000000000000 |
| | 01000000101111110011010000000101 | 01000000101111101111010000011111 |
| 101 | 10111111011110001001101000000010 | 10111111011101011111011011111101 |
| | 11000000010110010010001110100011 | 11000000010110000110100011011100 |
| 102 | 11000000001110100110000101111100 | 11000000001110011010110101000011 |
| | 11000001000010111101011010100001 | 11000001000010111111010101011010 |
| 103 | 01000000111001111100100111101111 | 01000000111010000000101101111000 |
| | 10111111101111011100111000001111 | 10111111101111001000000101101111 |
| 104 | 11000001000100110001011111000010 | 11000001000100110000100101101100 |
| | 10111111101001111111011000101011 | 10111111101001110110111111010010 |
| 105 | 01000001000000000101000110000011 | 01000001000000000000000000000000 |
| | 01000000110010100010001001101000 | 01000000110010100110000101111100 |
| 106 | 10111111111010010100010001101O0 | 10111111100101110100111100001110 |
| | 11000000110011000101010100110010 | 11000000110011001010110110101100 |
| 107 | 10111111001100000000110100011011 | 10111111001100001010101001100101 |
| | 01000001000000000101000110000011 | 01000001000000000000000000000000 |
| 108 | 01000001000000101010000111001011 | 01000001000000101101101100100011 |
| | 11000000101100001011010100001011 | 11000000101100001001000111010001 |
| 109 | 11000001000010011100011100010001 | 11000001000010011010000111001011 |
| | 10111111011101111000000000110100 | 10111111011110011010000000100111 |
| 110 | 01000000101000001010010010101001 | 01000000101000000000000000000000 |
| | 01000000100011000011101000101010 | 01000000100011001011111011100000 |
| 111 | 01000000000111110011011001111010 | 01000000000111111100011010101000 |
| | 11000001000110100100000111110010 | 11000001000110100001110111100111 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 112 | 1100000010101101101010101100111 | 1100000010101101011110110100 1010 |
| | 0100000010000000011010110101 0001 | 0100000010000000000000000000000 |
| 113 | 0100000010000010101101100100 0110 | 0100000010000010110110110010 0011 |
| | 1100000010000100001100011111 1001 | 1100000010000011110111111001 11011 |
| 114 | 1100000100000111000101011011 0101 | 1100000100000111010011101010 0101 |
| | 0011111110011111011011100010 11101 | 0011111110011111000111000100 00110 |
| 115 | 0100000001000000000001111011 1100 | 0100000001000000000000000000000 |
| | 0100000010001101011001011111 1110 | 0100000010001101100000110111 1011 |
| 116 | 0011111100000101101010000101 1000 | 0011111100000011000010111 1100 001 |
| | 1100000011000101010111010011 1110 | 1100000011000101100110011 0011010 |
| 117 | 1100000010100010111100100111 1100 | 1100000010100010110100110101 1011 |
| | 0100000010100000000000010100 1011 0 | 0100000010100000000000000000000 |
| 118 | 0100000100000011011010100111 1111 | 0100000100000011001111010111 0001 |
| | 1011111110011111100010100000 1001 | 1011111110100000001100010010 0111 |
| 119 | 1100000100000010101001100100 1100 | 1100000100000010100011011 0111001 |
| | 1100000010011010110000101011 0110 1 | 1100000010011010011110010011 1110 |
| 120 | 0100000011000000000000011111 0111 | 0100000011000000000000000000000 |
| | 0100000010110101100101001010 1111 | 0100000010110101110101010110 0111 |
| 121 | 1100000000101111000000000110 1001 | 1100000000101110010110010100 1011 |
| | 1100000001100111000111000100 0011 | 1100000001100110100100010000 0000 |
| 122 | 1100000000111010010000100101 1011 | 1100000000111001101011010100 0011 |
| | 0100000001000000000000011011 0100 | 0100000001000000000000000000000 |
| 123 | 0100000011011100110100110101 1011 | 0100000011011101011000111000 1000 |
| | 1011111100101100000101010100 1101 | 1011111100101001111010000011 1110 |
| 124 | 1100000011110100110101101010 0001 | 1100000011110100101110011111 0101 |
| | 1011111111111000100110100000 0010 | 1011111111111001100111001110 0000 |
| 125 | 0100000010100000000000100111 0001 | 0100000010100000000000000000000 |
| | 0100000011111011010011100011 1100 | 0100000011111011110100101111 0010 |
| 126 | 0011111010010001010110110101 0111 | 0011111010010100110010011 0000110 |
| | 1100000100011000101100010101 1011 | 1100000100011000111001110 0000100 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 127 | 01000000101010110100011110101110 | 01000000101010110010110101110111 |
| | 10111111110001000100100110111010 | 10111111110001100011100010000110 |
| 128 | 11000010100101101100001010001111 | 11000010100101100011001100110011 |
| | 11000010001000010001111010111100 | 11000010000111101100110011001101 |
| 129 | 01000000100001010111000010100100 | 01000000001001100110011001100110 |
| | 01000010000101011100110001000110 | 01000010000101011001100110011010 |
| 130 | 01000001010000110011001100110011 | 01000001001100000000000000000000 |
| | 01000000001011001100110011001101 | 00111111101001100110011001100110 |
| 131 | 01000000101111010010111110101100 | 01000000101111001100110011001101 |
| | 01000001101100110001110110101011 | 01000001101100110011001100110011 |
| 132 | 11000010101010010000001011101101 | 01000010101010010000000000000000 |
| | 01000010100000111101011110001110 | 01000010100000011110011001100 1101 |
| 133 | 11000000100001000000010011101010 | 11000000100001000110001111110001 |
| | 01000000101000000000011001100110 | 01000000101000000000000000000000 |
| 134 | 01000000100000100000001001110101 | 01000000100000101101101010111010 |
| | 11000000011000010110111100000000 | 11000000011000100001111111110011 |
| 135 | 11000000110100001011011110000000 | 11000000110100010101001111111000 |
| | 11000000101011010011001100110011 | 11000000101011010111101101001010 |
| 136 | 01000000101000000000111001100111 | 01000000101000000000000000000000 |
| | 01000000110111101001000000101110 | 01000000110111101110110011000000 |
| 137 | 11000000010001010110101110111010 | 11000000010001011100111000000111 |
| | 11000000111001101100010010011100 | 11000000111001101001000100000000 |
| 138 | 11000000101001000000001101000111 | 11000000101001000101110010010010 |
| | 01000000111000000110001111110001 | 01000000111000000000000000000000 |
| 139 | 11000000110110000010110000111101 | 11000000110110000110100011011100 |
| | 11000000101110000110111010011000 | 11000000101110000010010000001011 |
| 140 | 01000000000000000000011111100011 | 01000000000000000000000000000000 |
| | 01000001000100011100110011001101 | 01000001000100100001001001101111 |
| 141 | 11000000000100011000111111000101 | 11000000000100100000011101011111 |
| | 11000001000000000101110000101001 | 11000001000000000011101010010011 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 142 | 11000000101001000010100011110110 | 11000000101001001010000101100010 |
| | 11000000101000011000101110101100 | 11000000101000101101001101011011 |
| 143 | 00111111100000101011100111110101 | 00111111100000000000000000000000 |
| | 01000000111111001011000111000100 | 01000000111111001001011110001101 |
| 144 | 11000000011000010111101001111000 | 11000000011000101100001111001010 |
| | 01000000101000000110011001100110 | 01000000101000000000000000000000 |
| 145 | 01000000100000110110111000101111 | 01000000100000111001111101010110 |
| | 01000000010101000000000000000000 | 01000000010100111000110101010000 |
| 146 | 11000000101011011101010101100111 | 11000000101011010111101101001010 |
| | 01000000100000000110101101010001 | 01000000100000000000000000000000 |
| 147 | 01000001000000101011011001000110 | 01000001000000101101101100100011 |
| | 11000000100001000011000111111001 | 11000000100000111101111100111011 |
| 148 | 11000000100110011111110110001011 | 11000000100110011011010010100010 |
| | 01000000001111111110100100010000 | 01000000010000000000000000000000 |
| 149 | 01000001000011010000110001001010 | 01000001000011010010000011000101 |
| | 10111110100100111000000111011000 | 10111110100110000001110101111110 |
| 150 | 01000000101000000000001111110001 | 01000000101000000000000000000000 |
| | 01000000111110110100111000111100 | 01000000111110111101001011110010 |
| 151 | 00111110100100010101101101010111 | 00111110100101001100100110000110 |
| | 11000001000110001011000101011011 | 11000001000110001110011100000100 |
| 152 | 01000000101000000000010010101001 | 01000000101000000000000000000000 |
| | 01000000100011000011101000101010 | 01000000100011001011111011100000 |
| 153 | 01000000000111110011011001111010 | 01000000000111111100011010101000 |
| | 11000001000110100100000111110010 | 11000001000110100001110111100111 |
| 154 | 11000000011001110001110111100111 | 11000000011001101001000100000000 |
| | 11000000001001100111100001101100 | 11000000001001011110011010011011 |
| 155 | 01000000010000000010100011110110 | 01000000010000000000000000000000 |
| | 01000000001101011000010001001101 | 01000000001101010001000110011101 |
| 156 | 01000000111000000110000101111100 | 01000000111000000000000000000000 |
| | 01000000101010111100111000000111 | 01000000101010111111001000010011 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 157 | 1100000010110011001010111110101 00 | 1100000010110010110010100101100 0 |
| | 1100000000100101101100111101000 0 | 1100000000100101111001101001101 1 |
| 158 | 1100000010101101101010101100111 | 1100000010101101011110110100101 0 |
| | 0100000010111111001101000000101 | 0100000010111110111101000001111 1 |
| 159 | 1011111110111111010111000010100 | 1011111110100000001100010010011 1 |
| | 1100000100011001000010110111100 | 1100000100011000111001110000010 0 |
| 160 | 1011111111001100111100001101100 0 | 1011111111001110111111010010001 0 |
| | 0100000000000000101010001100000 1 | 0100000000000000000000000000000 0 |
| 161 | 0011111110000010111100011010101 0 | 0011111110000000000000000000000 0 |
| | 0100000010010111011000111000100 0 | 0100000010010111011001101100111 1 |
| 162 | 1011111111110101011000001000010 | 1011111111110101001001010100011 0 |
| | 1100000100001111100000011101100 0 | 1100000100001111011111110110001 1 |
| 163 | 1100000010101110010101111010100 0 | 1100000010101110001111111110011 0 |
| | 0100000000110100110111010001011 1 | 0100000000110101000100011001110 1 |
| 164 | 1011110110011111101111100111011 1 | 1011110110011011011100010111011 0 |
| | 1100000011011111100101011000000 1 | 1100000011011111011110011101110 |
| 165 | 1100000000100101110000101000111 1 | 1100000000100101111001101001101 1 |
| | 0100000001000000011101011111011 1 | 0100000001000000000000000000000 0 |
| 166 | 0100000101111101110001110111101 | 0100000101111101111010000011111 |
| | 0011111111110100110001100011111 1 | 0011111111110100010100000010010 0 |
| 167 | 0100000011111100101100011100010 0 | 0100000011111100100101111000110 1 |
| | 1100000001100001011110100111100 0 | 1100000001100010110000111100101 0 |
| 168 | 0100000010100000011001100110011 0 | 0100000010100000000000000000000 0 |
| | 0100000010000011011011100010111 1 | 0100000010000011100111110101011 0 |
| 169 | 0100000001010011100000001001110 | 0100000001010011100011010101000 0 |
| | 0100000010100000011001100110011 0 | 0100000010100000000000000000000 0 |
| 170 | 0100000010000011011011100010111 1 | 0100000010000011100111110101011 0 |
| | 0100000001010100000000000000000 0 | 0100000001010011100011010101000 0 |
| 171 | 0011111010010001010110110101011 1 | 0011111010010100110010011000011 0 |
| | 1100000100011000101100010101101 1 | 1100000100011000111001110000010 0 |

ตารางที่ 5.16 แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 172 | 10111111001100000000110100011011 | 10111111001100001010101001100101 |
| | 01000001000000000101000110000011 | 01000001000000000000000000000000 |
| 173 | 01000001000000101010000111001011 | 01000001000000101101101100100011 |
| | 11000000101100001011010100001011 | 11000000101100001001000111010001 |
| 174 | 11000001000010011100011100010001 | 11000001000010011010000111001011 |
| | 10111111011101111000000000110100 | 10111111011110011010000000100111 |
| 175 | 01000000101000001010010010101001 | 01000000101000000000000000000000 |
| | 01000000100011000011101000101010 | 01000000100011001011111011100000 |
| 176 | 01000000000111110011011001111010 | 01000000000111111100011010101000 |
| | 11000001000110100100000111110010 | 11000001000110100001110111100111 |
| 177 | 11000000101011011101010101100111 | 11000000101011010111101101001010 |
| | 01000000100000000110101101010001 | 01000000100000000000000000000000 |
| 178 | 01000001000000101011011001000110 | 01000001000000101101101100100011 |
| | 11000000100001000011000111111001 | 11000000100000111101111100111011 |
| 179 | 11000001000001110001010110110101 | 11000001000001110100111010100101 |
| | 00111111100111101101110001011101 | 00111111100111100011100010000110 |
| 180 | 01000000010000011101011111011100 | 01000000010000000000000000000000 |
| | 01000000100011010110010111111110 | 01000000100011011000001101111011 |
| 181 | 00111111000001011010100001011000 | 00111111000000110000101111100001 |
| | 11000000110001010101111010011110 | 11000000110001011001100110011010 |
| 182 | 11000000100110011111110110001011 | 11000000100110011011010010100010 |
| | 01000000001111111110100100010000 | 01000000010000000000000000000000 |
| 183 | 01000001000011010000110001001010 | 01000001000011010010000011000101 |
| | 10111110100100111000000111011000 | 10111110100111000000111010111110 |
| 184 | 11000000011001110001110111100111 | 11000000011001101001000100000000 |
| | 11000000001001100111100001101100 | 11000000001001011110011010011011 |
| 185 | 01000000010000000010100011110110 | 01000000010000000000000000000000 |
| | 01000000000110101100001000100110 | 01000000000110101000100011001101 |
| 186 | 00111111011101100101100101001011 | 00111111011101000100110100000001 |
| | 11000000111001101100101001011000 | 11000000111001101001000100000000 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 187 | 11000001001001011100101111111011 | 11000001001001011101100010101110 |
| | 11000000010111111101100010101110 | 11000000010111111011000101011011 |
| 188 | 01000000010000001100010010011100 | 01000000010000000000000000000000 |
| | 01000001000111000100111100001110 | 01000001000111000101100001111001 |
| 189 | 10111111101111001011010111011101 | 10111111101111001000000101101111 |
| | 11000000000011001100110011001101 | 11000000000011010010101111010100 |
| 190 | 11000000100110011001100110011010 | 11000000100110011011010010100010 |
| | 01000001000000000010100011110110 | 01000001000000000000000000000000 |
| 191 | 01000001000010000011011101001100 | 01000001000010000010111100011011 |
| | 10111111111000011110101110000101 | 10111111111000101000110000010101 |
| 192 | 11000000101001000010100011110110 | 11000000101001001010000101100010 |
| | 11000000101000011000101110101100 | 11000000101000010110100110101011 |
| 193 | 00111111100000101011100111110101 | 00111111100000000000000000000000 |
| | 01000000111111001011000111000100 | 01000000111111001001011110001101 |
| 194 | 00111101100101000100011001110100 | 00111101100011100010000110010110 |
| | 11000000100111111010100100101010 | 11000000100111111111101100010110 |
| 195 | 11000000000001110110110101011010 | 11000000000001111000010111100001 |
| | 01000000101000000000011000100101 | 01000000101000000000000000000000 |
| 196 | 01000000111110111111001110110110 | 01000000111111001001011110001101 |
| | 00111111101110011100000011101100 | 00111111101110111010110001110001 |
| 197 | 11000000011001100100001001011011 | 11000000011001101001000100000000 |
| | 11000000100001010110101011101000 | 11000000100001010010100101011111 |
| 198 | 01000000010000000000000101101111 | 01000000010000000000000000000000 |
| | 01000001000010000101000110000011 | 01000001000010000010111100011011 |
| 199 | 10111111100111111010111000010100 | 10111111101000000011000100100111 |
| | 11000001000110010000010110111100 | 11000001000110001110011100000100 |
| 200 | 10111111110011001111000011011000 | 10111111110011101111110100100010 |
| | 01000000000000000101010001100000 1 | 01000000000000000000000000000000 |
| 201 | 01000001000010000101101010000110 | 01000001000010000010111100011011 |
| | 10111111110101100000011101011111 | 10111111110110001101000110110111 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 202 | 1100000010100100010100011110110 | 1100000010100100101000010110001 0 |
| | 1100000010100001100010111010110 0 | 1100000010100010110100110101101 1 |
| 203 | 0011111110000010101110011111010 1 | 0011111110000000000000000000000 0 |
| | 0100000011111100101100011100010 0 | 0100000011111100100101111000110 1 |
| 204 | 1100000001100001011110100111000 | 1100000001100010110000111100101 0 |
| | 0100000010100000011001100110 | 0100000010100000000000000000000 |
| 205 | 0100000010000011011011100010111 1 | 0100000010000011100111110101011 0 |
| | 0100000001010100000000000000000 | 0100000001010011100011010101000 0 |
| 206 | 1100000010101101110101010110011 1 | 1100000010101101011110110100101 0 |
| | 0100000010000000000101101010001 | 0100000010000000000000000000000 |
| 207 | 1100000001100001011110100111000 | 1100000001100010110000111100101 0 |
| | 0100000010100000000001100110 | 0100000010100000000000000000000 |
| 208 | 0100000010000011011011100010111 1 | 0100000010000011100111110101011 0 |
| | 0100000001010011100010100001000 0 | 0100000001010011100011010101000 0 |
| 209 | 1100000010111010001010001111011 0 | 1100000010111001110111100110101 0 |
| | 1100000010101101110111111001110 11 | 1100000010101110001111111111001 10 |
| 210 | 0100000010000000001110000101001 0 | 0100000010000000000000000000000 |
| | 0100000011100110110101001111111 0 | 0100000011100111010001101101110 0 |
| 211 | 0011111111010110000101000111011 | 0011111111010111111111001011100 1 |
| | 1100000100001101000100000110001 | 1100000100001101001011000011110 1 |
| 212 | 1100000010100100000000110100011 1 | 1100000010100100010111001001001 0 |
| | 0100000011100000011000111111000 01 | 0100000011100000000000000000000 |
| 213 | 0100000010101011010001111010111 0 | 0100000010101011001011010111011 1 |
| | 1011111111000100010010011011101 0 | 1011111111000110001110001000011 0 |
| 214 | 1100000011011000001011000011110 1 | 1100000011011000011010001101110 0 |
| | 1100000010111000011011101001100 0 | 1100000010111000001001000000101 1 |
| 215 | 0100000000000000011000111111000 11 | 0100000000000000000000000000000 |
| | 0100000100010001110011001100 1101 | 0100000100010010000100100110111 1 |
| 216 | 1100000000010001100011111110001 01 | 1100000000010010000001110101111 1 |
| | 1100000100000000010111000010100 1 | 1100000100000000001110101001001 1 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 217 | 1100000010100010111100100111110 0 | 1100000010100010110100110101101 1 |
| | 0100000010100000001100101001011 0 | 0100000010100000000000000000000 0 |
| 218 | 0100000100000011011010100111111 1 | 0100000100000011001111010111000 1 |
| | 1011111110011111100010100000100 1 | 1011111110100000001100010010011 1 |
| 219 | 1100000100000010101001100100110 0 | 1100000100000010100011011011100 1 |
| | 1100000010011010110001010110110 1 | 1100000010011010011110010011111 0 |
| 220 | 0100000011000000011101011111011 1 | 0100000011000000000000000000000 0 |
| | 0100000010110101100101001010111 1 | 0100000010110101111010101011001 1 |
| 221 | 1100000000101111000000000110100 1 | 1100000000101110010110010100101 1 |
| | 1100000001100111000111000100001 1 | 1100000001100110100100010000000 0 |
| 222 | 1100000001100110111000101110101 1 | 1100000001100110100100010000000 0 |
| | 1100000010100011001100110011001 1 | 1100000010100011100101111111010 0 |
| 223 | 0100000010000000011110101110000 1 | 0100000010000000000000000000000 0 |
| | 0100000010000100010000110010110 1 | 0100000010000011100111110101011 0 |
| 224 | 0011111100000101100110110011110 1 | 0011111100000011000010111110000 1 |
| | 1100000010111110011001100110011 0 | 1100000010111110100001011000100 0 |
| 225 | 1100000010011001000011111111100 1 | 1100000010011000111100000000011 1 |
| | 0100000000000001010110000001000 0 | 0100000000000000000000000000000 0 |
| 226 | 0100000100000011010111000010100 1 | 0100000100000011001111010111000 1 |
| | 1011111101011111111111001011100 1 0 | 1011111101100010100010001100111 0 |
| 227 | 1100000000001100110011001100110 1 | 1100000000001101001010111101010 0 |
| | 1100000010011001100110011001101 0 | 1100000010011001101100101010001 0 |
| 228 | 0100000100000000001010001111011 0 | 0100000100000000000000000000000 0 |
| | 0100000100001000001101110100110 0 | 0100000100001000001011110001101 1 |
| 229 | 1011111111100001111010111000010 1 | 1011111111100010100011000001010 1 |
| | 1100000010100100001010001111011 0 | 1100000010100100101000010110001 0 |
| 230 | 1100000010100001100010111010110 0 | 1100000010100010110100110101101 1 |
| | 0011111110000010101110011111010 1 | 0011111110000000000000000000000 0 |
| 231 | 0100000011111100101100011100010 0 | 0100000011111100100101111000110 1 |
| | 0011110110010100010001100111010 0 | 0011110110001110001000011001011 0 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 232 | 1100000100011001000001011011110 0 | 1100000100011000111001110000010 0 |
| | 1011111111001100111100001101100 0 | 1011111111001110111111010010001 0 |
| 233 | 0100000000000000101010001100000 1 | 0100000000000000000000000000000 0 |
| | 0011111110000010111100011010101 0 | 0011111110000000000000000000000 0 |
| 234 | 0100000010010111011001110001000 | 0100000010010111011001101100111 1 |
| | 1011111111110101011000000100001 0 | 1011111111110101001001010100011 0 |
| 235 | 1100000100001111100000011011000 | 1100000100001111011111110110001 1 |
| | 1100000010101110010101111010100 0 | 1100000010101110001111111110011 0 |
| 236 | 0100000000110100110111010010111 1 | 0100000000110101000100011001110 1 |
| | 1011110110011111101111100111011 1 | 1011110110011011011100010111011 0 |
| 237 | 1011111101110111100000000110100 | 1011111101111001101000000010011 1 |
| | 0100000010100000001001010101001 | 0100000010100000000000000000000 0 |
| 238 | 0100000010001100001110100010101 0 | 0100000010001100101111101110000 0 |
| | 0100000000011111001101100111101 0 | 0100000000011111110001101010100 0 |
| 239 | 1100000100011010010000111110010 | 1100000100011010000111011110011 1 |
| | 1100000010101101110101010110011 1 | 1100000010101101011110110100101 0 |
| 240 | 0100000010000000001010110101000 1 | 0100000010000000000000000000000 0 |
| | 0100000100000010101101100100011 0 | 0100000100000010110110110010001 1 |
| 241 | 1100000010000100001100011111100 1 | 1100000010000011110111110011101 1 |
| | 1100000100000111000101011011010 1 | 1100000100000111010011101010010 1 |
| 242 | 1100000010101101110111111001110 1 | 1100000010101110001111111111001 1 0 |
| | 0100000010000000101110000101001 0 | 0100000010000000000000000000000 0 |
| 243 | 0100000011100110110101001111111 0 | 0100000011100111010001101101110 0 |
| | 0011111111010110000101000111101 1 | 0011111111010111111111001011100 1 |
| 244 | 1100000100001101000010000011000 1 | 1100000100001101001011000011110 1 |
| | 1100000010000100000001001110101 0 | 1100000010000100011000111111000 1 |
| 245 | 0100000010100000000000011110011 0 | 0100000010100000000000000000000 0 |
| | 0100000010000010000000100111010 1 | 0100000010000010110110101011101 0 |
| 246 | 1100000001100001011011111000000 0 | 1100000001100010000111111111001 1 |
| | 1100000011010000101101111000000 0 | 1100000011010001010100111111100 0 |

**ตารางที่ 5.16** แสดงผลที่ได้จากการทดสอบการทำงานเทียบกับผลที่ได้จากโปรแกรม Matlab (ต่อ)

| ข้อมูลชุดที่ | ผลที่ได้จากการทดสอบการทำงาน | ผลที่ได้จากโปรแกรม Matlab |
|---|---|---|
| 247 | 0011110110010100010001100111010 | 0011110110001110001000011001011 |
| | 1100000010011111101010010010101 | 1100000010011111111110110001011 |
| 248 | 1100000000000111011011010101110 | 1100000000000111000010111110000 |
| | 0100000010100001000001100010010 | 0100000010100000000000000000000 |
| 249 | 0100000011111101111110011101101 | 0100000011111100100101111100011 |
| | 0011111110111001110000001101100 | 0011111110111011101011000111000 |
| 250 | 1100000001100110111000101101011 | 1100000001100110100100010000000 |
| | 0100000000110100110111010010111 | 0100000000110101000100011001110 |
| 251 | 1011110110011011101111100111011 | 1011110110011011011100010111011 |
| | 1100000011011111100101011000001 | 1100000011011111011111001110111 |
| 252 | 1100000000100101110000101000111 | 1100000000100101111001101001101 |
| | 0100000001000000011101011111011 | 0100000001000000000000000000000 |
| 253 | 1011111110011111110001010000010 | 1011111110100000001100010010011 |
| | 1100000100000010101001100100110 | 1100000100000010100011011011100 |
| 254 | 0011111111110100111000110001111 | 0011111111110100010100000010010 |
| | 0100000010111110111000111011101 | 0100000010111110111101000001111 |
| 255 | 1100000011010111100010000110011 | 1100000011010110100100101010011 |
| | 0100000000000000110110111000110 | 0100000000000000000000000000000 |
| 256 | 1100000110010011110101110000101 | 1100000110010001100110011001010 |
| | 1100000101010000101000111101011 | 1100000101001001100110011001010 |

# ภาคผนวก ช.
## ผลงานวิจัยที่ได้รับการตีพิมพ์

บทความวิจัยที่ได้รับการตีพิมพ์ในวารสารต่างประเทศในวิทยานิพนธ์นี้ประกอบด้วย บทความ ดังต่อไปนี้

- Ratapracha Charnchonsamutra , Apinunt Thanachayanont and Varakorn Kasemsuwan "Design and Implementation of A Compact 256-point complex FFT Processor" *International Technical Conference on Circuit/Systems ComputersandCommunications (ITC-CSCC)* , Volume 4, pp. 1315-1316, 2005.

ITC-CSCC 2005

# ITC-CSCC 2005

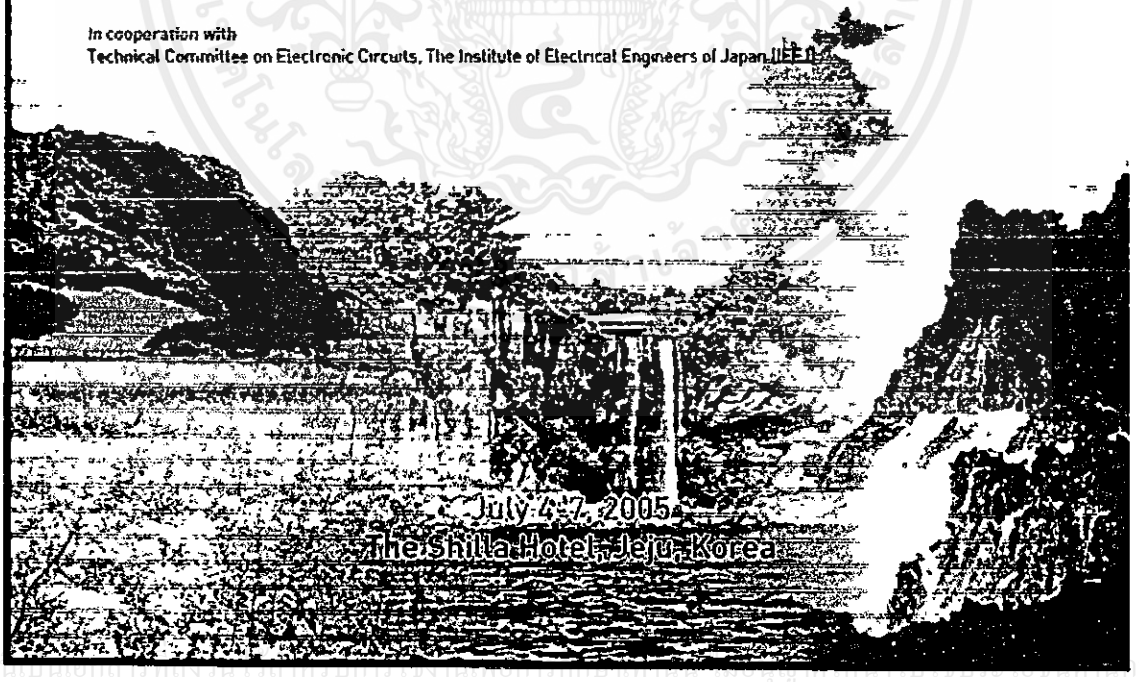
The 20th Commemorative International Technical Conference on Circuits/Systems, Computers and Communications

## Proceedings Volume 4

Sponsored by
The Institute of Electronics Engineers of Korea (IEEK)
The Institute of Electronics, Information and Communication Engineers (IEICE), The Engineering Sciences Society, Japan
The Electrical Engineering/Electronics, Computer, Telecommunications and Information Association, Thailand

Co-Sponsored by
Ministry of Information and Communication
Samsung Electronics
KTF
SK Telecom
LG Electronics
Institute of Information Technology Assessment
The Korean Federation of Science and Technology Societies
Korea Research Foundation
Korea National Tourism Organization
Jeju Provincial Government

In cooperation with
Technical Committee on Electronic Circuits, The Institute of Electrical Engineers of Japan (IEE J)



July 4-7, 2005
The Shilla Hotel, Jeju, Korea

# Design and implementation of a compact 256-point complex FFT Processor

Ratapracha Charnchonsamutra[1], Apinunt Thanachayanont[2] and Varakorn Kasemsuwan[3]
Microelectronics Research Laboratory, Research Center of Communications and Information Technology
Department of Electronic Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang
Chalongkrung Road, Ladkrabang, Bangkok, 10520, THAILAND, Tel: (02) 737-3000 ext 3309
E-mail: engin_nan@hotmail.com[1], ktapinun@kmitl.ac.th[2], kkvarako@kmitl.ac.th[3]

**Abstract:** This paper presents the design and implementation of a compact 256-point complex FFT processor. The proposed architecture uses a single-memory architecture. To achieve a compact hardware realization, input data words are sequentially fetched into the main 2-point radix-2 butterfly processor and FFT calculation is performed. The FFT processor has been designed and simulated by VHDL, and implemented by using XILINX SPARTAN-II FPGA. The realized FFT processor occupies 37,281 gates, excluding RAM. The processor can operate at 53 MHz and calculate a 256-point complex FFT in 77 μsec.

## 1. Introduction

Fast Fourier Transform (FFT) plays an important role in digital signal processing. It has been applied in a wide range of applications. Calculation speed and gate area are two major concerns of the FFT processor. The radix-2 FFT algorithm is chosen in this work to reduce the hardware size. The proposed FFT processor has been design and implemented by using XILINX SPARTAN-II FPGA.

## 2. Algorithm

The Decimation in Time (DIT) radix-2 algorithm was chosen because of its regularity and simplicity which reduces the design time, debugging effort, clock speed and other key parameters [1]. Figure 1 shows the signal flow graph of 8-point radix-2 FFT which can be explained by equations (1)-(5).

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad \text{where } W_N = e^{-j2\pi/N} \text{ and } k = 0,1,\ldots N-1 \quad (1)$$

If N is even, X(k) can be written as,

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

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{2nk} W_N^{k} \quad (3)$$

Consider the term $W_N^{ab}$ where a and b are not zero,

$$W_N^{ab} = e^{-j\frac{2\pi}{N}ab} = e^{-j\frac{2\pi}{N/b}a} = W_{N/b}^{a} \quad (4)$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{kn} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_{N/2}^{nk} \times W_N^{k} \quad (5)$$

In equation (5), X(k) consists of two terms each computes N/2 point FFT.

## 3. Structure of 256-point complex FFT processor

The structure of the proposed 256-point complex FFT processor is shown in figure 2. It comprises a single DIT-FFT radix-2 butterfly processor, control unit, address generator, coefficient ROM, clock generator and memory. FFT is computed in 8 stages (= $\log_2(256)$) and each stage requires reading and writing 256-point data word to the memory. All datapaths use the IEEE 32-bit floating point format.

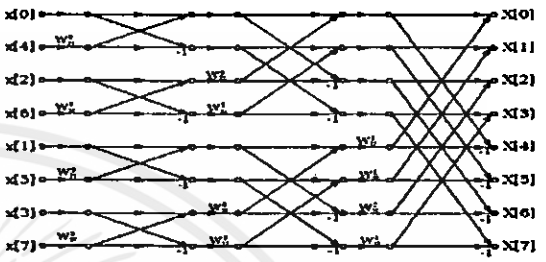



Figure 1. Radix-2 FFT signal flow graph for 8 point

The complete FFT operation can be explained in detail as follows. First, the 256-point data word are fetched and stored into the 512x32-bit memory. The control unit sends the value of the FFT stage to the address generator and the coefficient ROM, The address generator generates the address of the memory for reading of 256-point input data word. The coefficient ROM send the coefficient value, W (real, imaginary) to the butterfly processor. The computational result of each stage is written back to the memory and calculation for the next stage follows. The FFT is completed after 8 stages of butterfly computation.

The butterfly processor uses the DIT-FFT radix-2 butterfly algorithm. The signal flow graph of 256-point complex FFT processor can be reduced to the 2-point FFT processor, as shown in Figure 3. It comprises two real-multiplications and two real-additions, as shown in figure 4. The butterfly processor reads a 32-bit data (FFT-in) from the memory to compute a two-point FFT every four clock cycles. The two input data words are separated into real and imaginary parts: x (real), X (imaginary), y (real) and Y (imaginary). Two multiplexers are used to fetch the data into the multipliers, which multiply the data with the corresponding real and imaginary coefficient values obtained from the ROM. The output of the multipliers are summed by Adder 1. The output of adder 1 is inversed and added to the input data x,X by adder 2. The output of adder 2 is the calculation result of each FFT stage, which is written back to the memory.

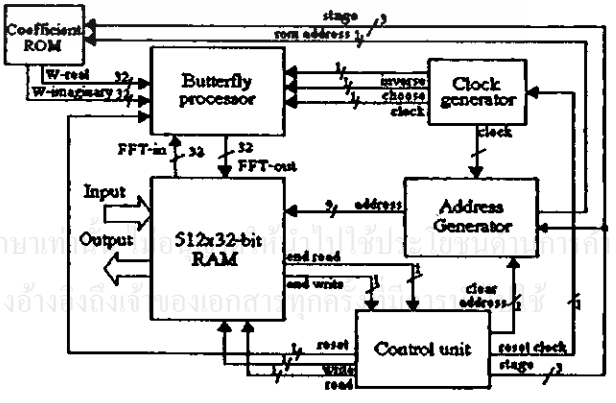



Figure 2. Block diagram of FFT processor

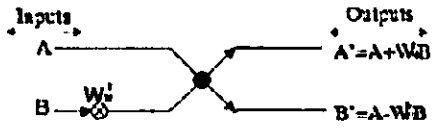



Figure 3. Signal flow graph of the 2-point FFT processor

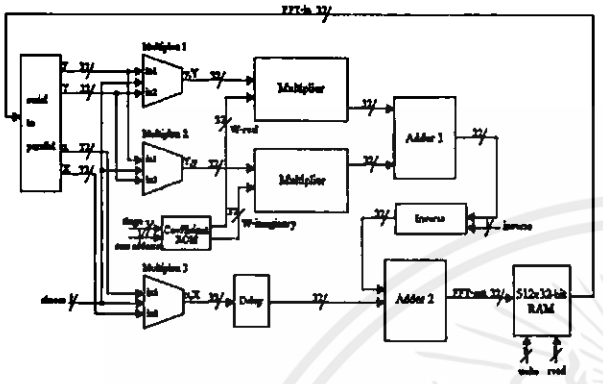


Figure 4. Block diagram of butterfly processor

## 4. FPGA implementation

The proposed FFT processor is designed and simulated by using VHDL and implemented on XILINX SPARTAN-II FPGA. Each of internal word lengths for real and imaginary parts is 32 bits. The memory has 512x32- bits word length. The FFT processor uses 37,281 gates excluding memory and it can operate at the maximum clock frequency of 53 MHz. The calculation time for 256-point complex FFT is 77 μsec. Table 1 compares the performance of proposed FFT processor with those reported in [2] and [3]. Figure 5 and Figure 6 show the simulation result and measurement result, respectively.

## 5. Conclusion

The proposed FFT processor architecture optimized for small gate area has been designed. The algorithm used was the decimation in Time (DIT) radix-2. The architecture consists of a radix-2 butterfly processor, dual port RAM, address generator, clock generator and control unit. Memory are used for stored the coefficients and the data. The proposed FFT processor consists of 37,281 gates excluding memory and can calculate a 256-point complex FFT in 77 μsec at 53 MHz clock frequency.

Table 1. Measured performance of the proposed FFT processor

| Parameters | [2] | [3] | This work |
|---|---|---|---|
| Datapath (bits) | 20 | 16 | 32 |
| Gate count | 98,326 | 37,000 | 37,281 |
| Time (μsec) | 6 | 6.6 | 77 |
| Clock Frequency (MHz) | 42 | 97 | 53 |
| Point Complex | 256 | 256 | 256 |

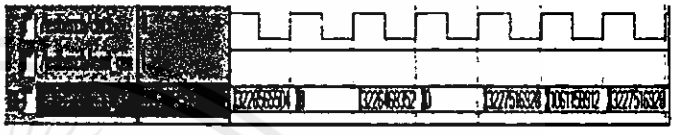


Figure 5. Simulated result (from VHDL code)

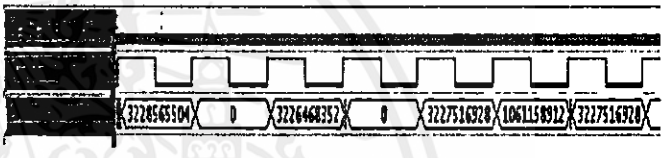


Figure 6. Measured result (from a logic analyzer)

## References

[1] Bevan M. Baas "A Low – Power , High Performance,1024 – point FFT Processor," IEEE Journal of Solid – State Circuit, vol. 34,No. 3,pp.380-387, 1999.

[2] Heo, K.L., Baek, J.H., Sunwoo, M.H., Jo, B.G.andSon, B.S. "New in-place strategy for a mixed-radix FFT processor" SOC Conference, 2003. Proceeding in IEEE International [Systems-on-Chip]. pp:81 – 84. 2003.

[3] Byung S. Son and Byung G. Jo "A High-Speed FFT Processor for OFDM system," IEEE, Circuits and Systems,.ISCAS 2002. IEEE International Symposium on, vol.3pp:281 -284, 2002.

# ประวัติผู้เขียน

ชื่อ-นามสกุล นาย รัฐประชา ชาญชลสมุทร

วัน เดือน ปีเกิด 3 มีนาคม 2522 จังหวัดชลบุรี

ประวัติการศึกษา :

ปี 2543 วศ.บ. สาขาวิศวกรรมอิเล็กทรอนิกส์ มหาวิทยาลัยเทคโนโลยีมหานคร

ประสบการณ์การทำงาน : -

ทุนการศึกษา : -