การใช้ CORDIC อัลกอริธึมสำหรับการออกแบบการแปลง DCT ด้วยอุปกรณ์ FPGA

CORDIC Algorithm for DCT by FPGA

โดย

นายนริศร รณรงค์ฤทธิ์

นายประยุทธ ผาโคตร

นายสนธิ วิมลศิลปวิญญู

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

การใช้ CORDIC อัลกอริธึมสำหรับการออกแบบการแปลง DCT ด้วยอุปกรณ์ FPGA

CORDIC Algorithm for DCT by FPGA

โดย

นายนริศร รณรงค์ฤทธิ์ 44015012

นายประยุทธ ผาโคตร 44015014

นายสนธิ วิมลศิลปวิญญู 44015029

อาจารย์ที่ปรึกษา

อาจารย์ อัครพล ตรีรัตน์

อาจารย์ ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ปีการศึกษา 2546

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การใช้ CORDIC อัลกอริธึมสำหรับการออกแบบการแปลง DCT ด้วยอุปกรณ์ FPGA**

CORDIC Algorithm for DCT by FPGA

ผู้จัดทำ

1. นายนริศร รณรงค์ฤทธิ์ 44015012

2. นายประยุทธ ผาโคตร 44015014

3. นายสนธิ วิมลศิลปวิญญู 44015029

..... อาจารย์ที่ปรึกษา

(อาจารย์ อัครพล ตรีรัตน์)

..... อาจารย์ที่ปรึกษา

(อาจารย์ ศรวัฒน์ ชิวปรีชา)

การใช้ CORDIC อัลกอริธึมสำหรับการออกแบบการแปลง DCT ด้วยอุปกรณ์ FPGA

CORDIC Algorithm for DCT by FPGA

โดย นายนริศร รณรงค์ฤทธิ์ 44015012
นายประยุทธ ผาโคตร 44015014
นายสนธิ วิมลศิลปวิญญู 44015029

อาจารย์ที่ปรึกษา อาจารย์ อัครพล ตรีรัตน์
อาจารย์ที่ปรึกษา อาจารย์ ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

โครงงานนี้จะทำการศึกษาเกี่ยวกับการนำแนวความคิดของ CORDIC Algorithm (Coordinate Rotation Digital Computer) มาประยุกต์ใช้ในการแปลง ด้วยวิธีการของ DCT(Discrete Cosine Transform) ซึ่งวิธีการนี้จะมีค่าสัมประสิทธิ์ในโดเมนความถี่ เป็นค่าจำนวนจริงทั้งหมด โดยขั้นตอนการออกแบบจะใช้วิธีอธิบายพฤติกรรมการทำงานของวงจรด้วยภาษา VHDL (Very high speed integrated circuit Hardware Description Language) และใช้โปรแกรม Max+Plus 2. ในการพัฒนาวงจรทั้งหมด โดยวงจรที่ได้จะนำไปโปรแกรมลงในอุปกรณ์ FPGA (Field Programmable Gate Array) เพื่อเป็นต้นแบบในการใช้งาน

Abstract

This project proposes the CORDIC Algorithm (Coordinate Rotation Digital Computer) in DCT Algorithm (Discrete Cosine Transform). By using DCT, we can obtain real coefficients in frequency domain. In this project, we use VHDL (Very high speed integrated circuit Hardware Description Language) together with Max+Plus2 as a tool to design and implement in FPGA (Field Programmable Gate Array).

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาของหัวข้อปริญญานิพนธ์

ในปัจจุบันแนวโน้มเกี่ยวกับการสร้างฮาร์ดแวร์ (Hardware) เพื่อมาใช้ในการประมวลผลสัญญาณมีมากขึ้น แต่ว่าความสามารถในการประมวลผลไม่ได้ขึ้นอยู่กับโครงสร้างของฮาร์ดแวร์แต่เพียงอย่างเดียวกล่าวคือ จะขึ้นอยู่กับ Algorithm ที่นำมาใช้สร้างด้วยรวมถึงการที่เราจะทำการพัฒนาหรือการนำไปสร้างเป็นฮาร์ดแวร์ของทฤษฎี (Theory) หรือ วิธีการ (Method) ใดๆนั้นหนทางหนึ่งที่เป็นไปได้คือการนำเอา Algorithm ต่างๆที่มีนำมาประยุกต์เพื่อใช้ร่วมกับวิธีการที่มีอยู่ให้สามารถสร้างได้จริงหรือมีประสิทธิภาพที่ดีขึ้น ในปริญญานิพนธ์ฉบับนี้ก็เช่นกันได้มีการนำเอาแนวคิดของ CORDIC (Coordinate Rotation Digital Computer) Algorithm มาทำการศึกษาถึงวิธีการทำงาน แล้วได้ทดลองนำไปประยุกต์ใช้กับวิธีการของ DCT (Discrete Cosine Transform) ซึ่งเป็นรูปแบบของการแปลงข้อมูล พร้อมทั้งทำการออกแบบและสร้างเป็นฮาร์ดแวร์ซึ่งในที่นี้จะใช้การบรรยายพฤติกรรมการทำงานด้วยภาษา VHDL (Very high speed integrated circuit Hardware Description Language) ในการออกแบบ

แนวความคิดของ CORDIC Algorithm นับว่าได้ถูกคิดค้นขึ้นมานานแล้ว โดยใช้ชื่อเริ่มต้นว่า Volder's Coordinate Rotation Digital Computer ซึ่งถูกคิดค้นขึ้นมาในปี ค.ศ.1959 เพื่อการเดินเรือและควบคุมอุปกรณ์ต่างๆ หลังจากนั้นก็ถูกนำมาประยุกต์ใช้ในการแก้ปัญหาต่างๆมากมาย ทั้งยังมีการนำมาพัฒนาสร้างเป็นฮาร์ดแวร์ในการคำนวณทางด้านคณิตศาสตร์ (Arithmetic) ลักษณะพิเศษของ CORDIC Algorithm คือสามารถคำนวณหาค่าใน ฟังก์ชันของตรีโกณมิติต่างๆได้ โดยใช้วิธีการเลื่อนและบวก (Shift and Add) เป็นหลักและยังมีผลลัพธ์ของสมการอยู่ในรูปผลบวกของผลคูณ (Sum of Product) จึงเป็นทางออกสำหรับการสร้างฮาร์ดแวร์ที่ไม่ต้องการใช้การคูณ (Multiplication) โดยตรงเพื่อหวังว่าจะสามารถลดจำนวนอุปกรณ์ที่ต้องใช้ ก็สามารถนำ CORDIC Algorithm ไปประยุกต์ใช้แทนได้

ในการประมวลผลสัญญาณหรือการสื่อสารในปัจจุบัน มีการพัฒนาในด้านต่างๆให้มีประสิทธิภาพและตรงตามความต้องการของผู้ใช้อาทิเช่น ความถูกต้องและความเร็วในการประมวลผลหรือการส่งเป็นต้น ดังนั้นส่วนใหญ่สัญญาณเหล่านี้จึงทำให้อยู่ในรูปของข้อมูลดิจิตอล (Digital) ซึ่งจะทำให้มีคุณสมบัติดีกว่าอนาล็อก (Analog) ที่ใช้กันในอดีต และการแก้ปัญหาด้านความเร็วอีกวิธีหนึ่งก็คือการลดข้อมูลที่มีก็จะลดเวลาในการประมวลผลหรือการส่งได้ ซึ่งวิธีการของ DCT ก็เป็นอีกวิธีการหนึ่งที่ถูกนำมาใช้กันอย่างกว้างขวางในการลดขนาดข้อมูล ข้อดีอย่างหนึ่งของวิธีการแบบ DCT ก็คือผลที่ได้จากการแปลงซึ่งเป็นค่าสัมประสิทธิ์ในโดเมนความถี่ (Frequency Domain) จะเป็นค่าของจำนวนจริงเท่านั้นทำให้เป็นการง่ายต่อการพิจารณา อีกทั้งยังสามารถคงข้อมูลที่มีความสำคัญไว้ได้อย่างดีอีกด้วย

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1.2.1 เพื่อศึกษาทฤษฎีและหลักการทำงานของ CORDIC Algorithm

1.2.2 ทำการจำลองการทำงานของวิธีการแปลงแบบ DCT ด้วยภาษา วีเอชดีแอลและเก็บผลการทดลองจากโปรแกรม max+plus2

1.2.3 นำหลักการของ CORDIC Algorithm ไปประยุกต์ใช้กับวิธีการแปลงแบบ DCT

1.2.4 นำทฤษฎีที่ได้จากการประยุกต์มาใช้ในการออกแบบสร้างเป็นฮาร์ดแวร์บนชิพของอุปกรณ์เอฟพีจีเอด้วยภาษาวีเอชดีแอล

1.2.5 ทำการเปรียบเทียบผลที่ได้จากวิธีการแปลงแบบ DCT กับการประยุกต์ด้วยหลักการของ CORDIC Algorithm

1.3 ขอบเขตของปริญญานิพนธ์

จากจุดประสงค์ที่กล่าวมาข้างต้นจะเห็นได้ว่า ในโครงงานนี้จะเป็นการศึกษาและแสดงให้เห็นถึงการนำไปใช้งานของ CORDIC Algorithm โดยในที่นี้จะนำไปประยุกต์ใช้กับวิธีการแปลงแบบ DCT หนึ่งมิติ (One Dimension DCT) และทำการทดลองให้เห็นถึงการทำงานจริง ซึ่งกระบวนการทำงานทั้งหมดจะถูกบรรยายพฤติกรรมการทำงานเพื่อสร้างเป็นฮาร์ดแวร์ โดยใช้การเขียนด้วยภาษา วีเอชดีแอล ในการออกแบบ วงจรทางลอจิกภายในชิพอุปกรณ์ FPGA พร้อมทั้งจำลองผลที่ได้จากการทำงานจริงของการแปลงแบบ DCT มาเปรียบเทียบผลลัพธ์ที่ได้จากการจำลองผลเพื่อการพัฒนาต่อไป

1.4 เนื้อหาของปริญญานิพนธ์

ในบทที่ 2 จะกล่าวถึงทฤษฎีของ CORDIC Algorithm โดยจะแสดงแนวคิดในการเริ่มต้นของสมการพื้นฐานของ CORDIC Algorithm , การประยุกต์สมการเพื่อนำไปหาค่าในฟังก์ชันของตรีโกณมิติ (trigonometric) ต่างๆรวมถึงวิธีการคำนวณหาค่า, ข้อดีที่จะนำไปสร้างเป็นฮาร์ดแวร์ในทางดิจิตอลเบื้องต้น, วิธีการของการแปลงแบบ DCT ซึ่งเป็นการเปลี่ยนข้อมูลให้อยู่ในเมนความถี่ (Frequency Domain), ทฤษฎีของภาษาวีเอชดีแอล, ตัวอย่างการออกแบบในส่วนของการออกแบบโครงสร้างและการนำทฤษฎีของ DA(Distributed Arithmetic) มาใช้ร่วมเพื่อให้ง่ายต่อการประมวลผลและลดจำนวนอุปกรณ์ในการนำไปสร้างใช้งาน, ทฤษฎีและหลักการของชิพอุปกรณ์เอฟพีจีเอและหลักการออกแบบเพื่อโปรแกรมลงบนชิพอุปกรณ์เอฟพีจีเอ

ในบทที่ 3 กล่าวถึงวิธีการคำนวณและออกแบบการสร้างการแปลงแบบ DCT และวิธีการนำหลักการของ CORDIC Algorithm มาประยุกต์ใช้กับการแปลงแบบ DCT

ในบทที่ 4 กล่าวถึงการทดลองและผลการทดลองที่ได้จากการคำนวณและออกแบบการสร้างการแปลงแบบ DCT และกรณีที่นำหลักการของ CORDIC ประยุกต์ใช้กับการแปลงแบบ DCT

ในบทที่ 5 กล่าวถึง บทวิจารณ์และสรุปผล

หนังสืออ้างอิง

บทที่2
ทฤษฎีและหลักการ

2.1 CORDIC Algorithm

CORDIC (Coordinate Rotation Digital Computer) เป็นทฤษฎีของการหาค่าในฟังก์ชันในตรีโกณมิติเช่น sine,cosine,tangent,arcsine,arcosine และ arctangent เป็นต้น ซึ่งจะทำการคำนวณค่าโดยวิธีการหมุนเวคเตอร์ในระบบของพิกัดฉาก ซึ่งให้ค่าผลลัพธ์ของขนาดหรือมุมของของเวคเตอร์นั้นๆ ออกมา การคำนวณจำเป็นต้องมีการกำหนดค่าเริ่มต้นที่ค่าๆหนึ่งแล้วจึงค่อยๆทำการลดค่าหรือวนรอบในการเปลี่ยนแปลงค่าไปเรื่อยๆจนได้ค่าที่ต้องการ โดยจะพิจารณา Vector diagram ดังที่แสดงในรูปที่ 2.1 ประกอบ

รูปที่ 2.1 การหมุนของเวคเตอร์ในพิกัดฉาก

จากรูปที่ 2.1 กำหนดให้ R คือขนาดของเวคเตอร์ θ คือมุมของเวคเตอร์ x และ y คือตำแหน่งของเวคเตอร์ในพิกัดฉาก ซึ่งจะได้ความสัมพันธ์คือ

$$\tan\theta = \frac{y_i}{x_i}$$

และ

$$\tan(\theta+\Delta\theta_i) = \frac{y_i+1}{x_i+1} \qquad (2.1)$$

จากเอกลักษณ์ของตรีโกณมิติ $\tan(x \pm y) = \dfrac{\tan x \pm \tan y}{1 \mp \tan x \tan y}$

เขียนสมการที่ (2.1) ใหม่ได้เป็น

$$\frac{y_{i+1}}{x_{i+1}} = \frac{\dfrac{y_i}{x_i} + \tan \Delta\theta_i}{1 - \left(\dfrac{y_i}{x_i}\right) \tan \Delta\theta_i}$$

นำ $\dfrac{x_i}{y_i}$ คูณเศษและส่วนด้านขวาของสมการตลอดแล้วทำการจัดรูปใหม่

$$\frac{y_{i+1}}{x_{i+1}} = \frac{y_i + y_i \tan \Delta\theta_i}{x_i - y_i \tan \Delta\theta_i} \tag{2.2}$$

พิจารณาเฉพาะขนาดของเวคเตอร์

$$y_{i+1}^2 + x_{i+1}^2 = y_i^2 + x_i^2 \tag{2.3}$$

จาก สมการที่ (2.2) และ (2.3) $y_i^2 + x_i^2 = x_{i+1}^2 + x_{i+1}^2 \left[\dfrac{y_i + x_i \tan \Delta\theta_i}{x_i - y_i \tan \Delta\theta_i}\right]^2$

คูณตลอดด้วย $(x_i - y_i \tan \Delta\theta_i)^2$

$$\begin{aligned}(x_i - y_i \tan \Delta\theta_i)^2 (y_i^2 + x_i^2) &= (x_i - y_i \tan \Delta\theta_i)^2 x_{i+1}^2 + (y_i + x_i \tan \Delta\theta)^2 x_{i+1}^2 \\ &= x_{i+1}^2 \left(x_i^2 + x_i^2 \tan^2 \Delta\theta_i + y_i^2 \tan^2 \Delta\theta_i + y_i^2\right) \\ &= x_{i+1}^2 (x_i^2 + y_i^2)(1 + \tan^2 \Delta\theta_i)\end{aligned}$$

$$\therefore \quad x_{i+1} = \frac{x_i - y_i \tan \Delta\theta_i}{\sqrt{1 + \tan^2 \Delta\theta_i}} \tag{2.4}$$

จาก (2.2) $$y_{i+1} = x_{i+1}\left[\frac{y_i + x_i \tan \Delta\theta_i}{x_i - y_i \tan \Delta\theta_i}\right]$$

แทน x_{i+1} จาก (2.4) จะได้

$$y_{i+1} = \frac{x_i - y_i \tan\Delta\theta_i}{\sqrt{1+\tan^2\Delta\theta_i}}\left[\frac{y_i + x_i \tan\Delta\theta_i}{x_i - y_i \tan\Delta\theta_i}\right]$$

$$= \frac{y_i - x_i \tan\Delta\theta_i}{\sqrt{1+\tan^2\Delta\theta_i}} \tag{2.5}$$

นำเทอมของส่วนมาพิจารณา

$$= \sqrt{1+\tan^2\Delta\theta_i}$$

$$= \left(\frac{\cos^2\Delta\theta_i + \sin^2\Delta\theta_i}{\cos^2\Delta\theta_i}\right)^{1/2}$$

$$= \left(\frac{1}{\cos\Delta\theta_i}\right) \tag{2.6}$$

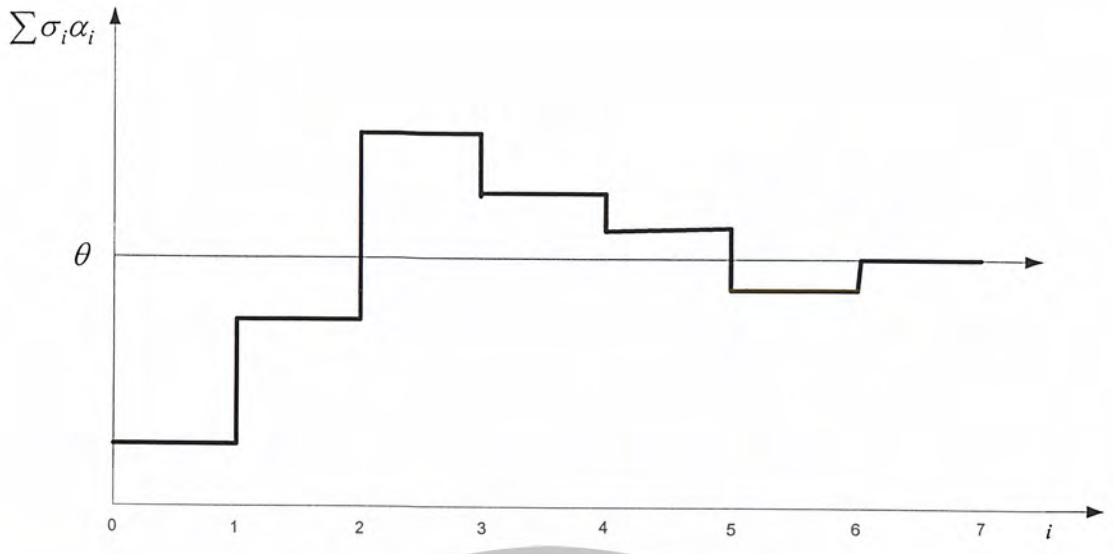
จาก (2.6) เขียนสมการ (2.4) และ (2.5) ใหม่เป็น

$$\left.\begin{aligned} x_{i+1} &= x_i \cos\Delta\theta_i - y_i \sin\Delta\theta_i \\ y_{i+1} &= y_i \cos\Delta\theta_i + x_i \sin\Delta\theta_i \end{aligned}\right\} \tag{2.7}$$

เห็นได้ว่าสามารถหาตำแหน่งของเวคเตอร์อีกตำแหน่งหนึ่งได้โดยผ่านความสัมพันธ์จากสมการนี้ ซึ่งวิธีการของ CORDIC จะถูกกำหนดค่าการหมุนโดย $\Delta\theta_i$ และถ้าหากให้ลำดับในการหมุนมีค่าเท่ากับ α_i โดยมีการหมุนทั้งหมด n ครั้งและแต่ละครั้งถูกกำหนดเครื่องหมายโดย σ_i ดังนั้นจะทำให้ได้ค่าของ $\Delta\theta_i$ ตามสมการข้างล่าง

$$\Delta\theta_i = \sum_{i=0}^{n-1} \sigma_i \alpha_i \quad ;\sigma_i \in \{-1,1\} \tag{2.8}$$

โดยที่ σ_i จะมีการเปลี่ยนแปลงตามค่ามุมของแต่ละครั้งที่มีการหมุนเป็นตำแหน่ง α_i ซึ่งจะขึ้นอยู่กับผลของค่าผลรวมก่อนหน้านั้นเพื่อพิจารณาค่าของ $\Delta\theta_i$ ไปในทิศทางเดียวโดยเปรียบเทียบกับค่าของ $\Delta\theta_i$ ดังรูปที่ 2.2



รูปที่ 2.2 ผลการกระทำของค่า σ_i ที่เกิดกับค่า θ

จากรูปที่ 2.2 จะเห็นว่าค่าผลรวมของ α_i ($\sum \sigma_i \alpha_i$) จะมีค่าที่เป็นไปได้ทั้งช่วงที่มากกว่าและน้อยกว่าค่าของมุม θ และเมื่อจำนวนค่า i สูงขึ้นเรื่อยๆนั่นคือเกิดการหมุนของตำแหน่งเวคเตอร์จำนวนหลายครั้งขึ้นก็จะได้ผลรวมของ α_i จนเท่ากับค่าของมุม θ ซึ่งจะเห็นได้ว่า σ_i แต่ละครั้งเป็นตัวแปรสำคัญที่จะกำหนดให้ค่าผลรวมมีทิศทางเข้าหาค่าของ θ และจากวิธีการนี้เองจะได้สมการที่มาช่วยในการพิจารณาเครื่องหมาย(ค่าของ σ_i) คือ

$$z_{i+1} = z_i - \sum_{i=0}^{n-1} \sigma_i \alpha_i \tag{2.9}$$

โดยที่

$$\sigma_i \begin{cases} +1 & z_i \geq 0 \\ -1 & z_i < 0 \end{cases} \tag{2.10}$$

พิจารณาจากสมการที่ (2.7) จะเห็นว่าค่าของ $\Delta\theta_i$ ก็คือ α_i ที่ตำแหน่ง i ค่าต่างๆนั่นเองซึ่งสามารถเขียนสมการใหม่ได้เป็น

$$\left.\begin{aligned} x_{i+1} &= x_i \cos\alpha_i - \sigma_i y_i \sin\alpha_i \\ y_{i+1} &= y_i \cos\alpha_i + \sigma_i x_i \sin\alpha_i \end{aligned}\right\} \tag{2.11}$$

เพื่อให้ง่ายต่อการคำนวณจะใช้การประมาณค่าของ $\tan\alpha_i$ เทียบเคียงกับอนุกรมกำลังของ 2 ดังสมการที่ (2.12)

$$\tan\alpha_i = 2^{-i} \quad ;i=0,1,2,...,n-1 \tag{2.12}$$

จากสมการที่ (2.11) และ (2.12) จะได้ว่า

$$x_{i+1} = \cos\alpha_i\left[x_i - \sigma_i y_i \tan\alpha_i\right]$$
$$y_{i+1} = \cos\alpha_i\left[y_i + \sigma_i x_i \tan\alpha_i\right]$$

ดังนั้น

$$x_{i+1} = k_i\left[x_i - \sigma_i 2^{-i} y_i\right]$$
$$y_{i+1} = k_i\left[y_i + \sigma_i 2^{-i} x_i\right] \tag{2.13}$$

และเมื่อ $k_i = \cos\alpha_i$ ทำการจัดอยู่ในรูปของเทอม 2^{-i} ได้ดังนี้

$$= \frac{1}{\sqrt{\frac{\cos^2\alpha_i + \sin^2\alpha_i}{\cos^2\alpha_i}}}$$
$$= \frac{1}{\sqrt{1+\tan^2\alpha_i}}$$
$$= \frac{1}{\sqrt{1+2^{-2i}}}$$

ซึ่งเมื่อกระทำการวนซ้ำหรือหมุนเวกเตอร์ไปแต่ละค่าเรื่อยๆนำค่า k_i มาแยกพิจารณาก็จะได้สัมประสิทธิ์ใหม่เป็น

$$k = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} \tag{2.14}$$
$$\cong 0.607253$$

ดังนั้น จะพิจารณาเฉพาะเทอมของตัวแปรที่จะนำมาใช้ในการวนซ้ำจากสมการที่ (2.9) และ (2.13) จะเขียนความสัมพันธ์ใหม่ได้เป็น

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i$$
$$y_{i+1} = y_i + \sigma_i 2^{-i} x_i \tag{2.15}$$
$$z_{i+1} = z_i - \sigma_i \tan^{-1}(2^{-i})$$

ซึ่งจากสมการที่ (2.15) นี้สามารถคำนวณหาค่าผลลัพธ์ที่ต้องการโดยการป้อน x_0, y_0 และ z_0 เป็นค่าเริ่มต้นให้ระบบโดยที่ค่าของ z_0 จะต้องอยู่ในช่วงของ $\pm\frac{\pi}{2}$ เท่านั้นซึ่งถ้าค่าของ z_0 นอกเหนือไปจากนี้อาจจะนำคุณสมบัติของตรีโกณมิติมาช่วยดังที่แสดงในสมการข้างล่างนี้

$$\cos(z \pm 2\pi) = \cos z\,,\ \sin(z \pm 2\pi) = \sin z$$
$$\cos(z - \pi) = -\cos z\,,\ \sin(z - \pi) = -\sin z \qquad (2.16)$$

เมื่อทำการกำหนดหรือป้อนค่าเริ่มต้นแล้วทำการคำนวณค่าออกมาก็จะได้ผลดังตัวอย่างในตารางที่ 2.1 ซึ่งกำหนดให้ $z_0 = 30^\circ$, $x_0 = k$ และ $y_0 = 0$

i	σ_i	$x_{i+1} \rightarrow \cos z_0$	$y_{i+1} \rightarrow \sin z_0$	$z_{i+1} \rightarrow 0$
		k =0.607253	0.000000	30.0000000
0	1	0.607253	0.607253	-15.00000
1	-1	0.910880	0.303627	11.565051
2	1	0.834973	0.531347	-2.471192
3	-1	0.901391	0.426975	4.653824
4	1	0.874705	0.483312	1.077490
5	1	0.859602	0.510647	-0.712420
6	-1	0.867581	0.497216	0.182754
7	1	0.863697	0.503994	-0.264860
8	-1	0.865666	0.500620	-0.041049

ตารางที่ 2.1 แสดงการคำนวณหาค่าของ sin และ cosine

โดยที่ค่าจริงของ $\cos 30^\circ = 0.866025$ และ $\sin 30^\circ = 0.500000$ นอกจากนี้ยังสามารถที่จะหาค่าอื่นๆได้จากการพิจารณาจากผลลัพธ์ของเวคเตอร์ที่ได้จากการหมุนโดยต้องมีตัวใดตัวหนึ่งที่จะถูกนำมาเปรียบเทียบในการพิจารณาเครื่องหมายตัวอย่างเช่น ถ้ากำหนดให้ผลลัพธ์ของ $y_n = 0$ จะทำให้ได้ผลลัพธ์จากการคำนวณเป็นดังนี้

$$x_n = \sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 - \tan^{-1}\left(\frac{y_0}{x_0}\right) \qquad (2.17)$$

เช่นเดียวกันกับวิธีการที่แสดงในตัวอย่างข้างต้นดังตารางที่ 2.1 คือทำการลดค่า z_n ลงสู่ค่าศูนย์ แต่ในตัวอย่างนี้จะทำการลดค่าของ y_n ลงแทนดังนั้นเทอมที่จะนำมาใช้ในการพิจารณาเครื่องหมายจะกลายเป็นเทอมของ y_i ที่ตำแหน่งต่างๆดังสมการ (2.18)

$$\sigma_i = \begin{cases} -1 & y_i \geq 0 \\ +1 & y_i < 0 \end{cases} \tag{2.18}$$

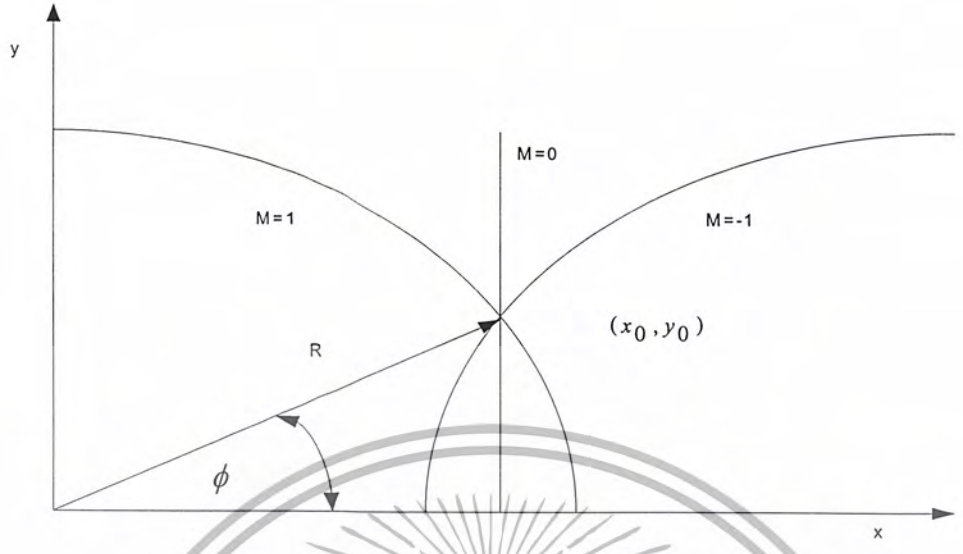
เมื่อทำการกำหนดค่าเริ่มต้นให้ตามสมการที่ (2.17) โดยให้ค่า $x_0 = 1, y_0 = 0.41421$ และก็จะได้ผลการคำนวณดังตารางที่ 2.2

i	σ_i	$x_{i+1} = \sqrt{x_0^2 + y_0^2}/k$	$y_{i+1} \to 0$	$z_{i+1} \to \tan^{-1}\left(y_0/x_0\right)$
		1.000000	0.414210	0.000000
0	-1	1.414210	-0.585790	45.000000
1	1	1.707105	0.121315	18.434949
2	-1	1.737434	-0.305461	32.471192
3	1	1.775617	-0.088282	25.346176
4	1	1.781135	0.022694	21.769842
5	-1	1.781844	-0.032966	23.559752
6	1	1.782359	-0.005125	22.664578
7	1	1.782399	0.008800	22.216964
8	-1	1.782433	0.000184	22.440775

ตารางที่ 2.2 แสดงการคำนวณหาค่า $\tan^{-1}(\frac{y_0}{x_0})$

ซึ่งค่าของ $\tan^{-1} 0.41421 = 22.49983$ จะเห็นได้ว่ายิ่งเพิ่มรอบในการคำนวณมากเท่าใด ค่าที่ได้ก็จะใกล้เคียงกับค่าจริงมากขึ้นและยังสามารถนำไปประยุกต์ใช้เพื่อหาค่าอื่นๆในฟังก์ชันตรีโกณมิติได้อีกมากมาย

2.1.1 การประยุกต์ใช้กับฟังก์ชันไฮเปอร์บอลิก (Hyperbolic Function)



รูปที่ 2.3 แสดงตำแหน่งของเวคเตอร์ R ในแต่ละพิกัด

พิจารณาจากรูปที่ 2.3 เป็นการแสดงตำแหน่งของเวคเตอร์ R และมุม (ϕ) ณ.จุดของเวคเตอร์ (x_0, y_0) โดยมีตัวแปร m เป็นตัวกำหนดระบบพิกัดซึ่งจะสามารถเขียนเป็นสมการได้ดังนี้

$$R = \sqrt{x_0^2 + m y_0^2} \tag{2.19}$$

$$\phi = \frac{1}{\sqrt{m}} \tan^{-1}\left(\sqrt{m}\,\frac{y_0}{x_0}\right) \tag{2.20}$$

หรือในทางตรงกันข้ามคือ

$$x_0 = R\cos\left(\sqrt{m}\phi\right)$$

$$y_0 = \frac{1}{\sqrt{m}} \tan^{-1}\left(\frac{\phi}{\sqrt{m}}\right) \quad \text{โดยที่ } m \in \{1,0,-1\} \tag{2.21}$$

ซึ่งจะสามารถแบ่งรูปแบบพิกัดออกเป็น 3 แบบคือฟังก์ชัน ไฮเปอร์บอลิก (Hyperbolic; m = -1), เชิงเส้น(Linear; m = 0) และวงกลม(Circular; m = 1) ดังที่แสดงในรูปที่ 2.3 ดังนั้นในสมการที่ (2.15) สามารถนำมาเขียนความสัมพันธ์ใหม่ได้เป็น

$$\begin{aligned} x_{i+1} &= x_i - m\sigma_i\delta_{m,i}y_i \\ y_{i+1} &= y_i + \sigma_i\delta_{m,i}x_i \\ z_{i+1} &= z_i - \sigma_i\alpha_{m,i} \end{aligned} \tag{2.22}$$

โดยที่พารามิเตอร์แต่ละตัวมีค่าดังต่อไปนี้

m เป็นตัวแปรที่กำหนดพิกัดของระบบที่มีค่าเป็น $\{1,0,-1\}$

σ_i ตัวกำหนดทิศทาง(เครื่องหมาย)ของการหมุนแต่ละตำแหน่ง $\{1,-1\}$

$\alpha_{m,i}$ มุมที่เปลี่ยนไปที่ค่า i แต่ละตำแหน่ง

$\delta_{m,i}$ ค่าของ 2^{-i} ที่จะถูกกำหนดค่า i ในแต่ละพิกัดต่างๆกัน $\{0<\delta_{m,i}\leq 1\}$

i ตำแหน่งของการหมุน $\{0,1,2,....,n-1\}$

ซึ่งในส่วนที่เป็นขนาด(สัมประสิทธิ์ตัวคูณ)และส่วนของมุมแต่ละลำดับในพิกัดต่างๆสามารถจะสรุปความสัมพันธ์ได้ดังนี้

m	1	0	-1
$\alpha_{m,i}$	$\tan^{-1}\delta_{1,i}$	$\delta_{0,i}$	$\tanh^{-1}\delta_{-1,i}$
$k_{m,i}^{-1}$	$\sqrt{1+\delta_{1,i}^2}$	1	$\sqrt{1-\delta_{-1,i}^2}$

ตารางที่ 2.3 แสดงค่าของสัมประสิทธิ์และมุมที่แทนในสมการต่างๆ

ดังนั้นในสมการที่ (2.14) และ (2.8) จะถูกนำมาเขียนได้ใหม่ในส่วนของสัมประสิทธิ์ตัวคูณกับผลรวมของมุมหลังการหมุนไปจำนวน n รอบแล้วได้เป็นดังนี้

$$k_m=\prod_{i=0}^{n-1}k_{m,i}=\prod_{i=0}^{n-1}\frac{1}{\sqrt{1+m\delta_{m,i}^2}} \tag{2.23}$$

$$\alpha_m=\sum_{i=0}^{n-1}\sigma_i\alpha_{m,i}=\begin{cases}\sum_{i=0}^{n-1}\sigma_i\tan^{-1}\delta_{1,i} & m=1\\ \sum_{i=0}^{n-1}\sigma_i\delta_{0,i} & m=0\\ \sum_{i=0}^{n-1}\sigma_i\tanh^{-1}\delta_{-1,i} & m=-1\end{cases} \tag{2.24}$$

จากที่ได้ทราบมาแล้วในตัวอย่างที่แสดงในตารางที่ 2.1 และ 2.2 ซึ่งจะเป็นการหาค่า $\sin z_0, \cos z_0$ และ $\tan^{-1} z_0$ ในโหมดของไฮเปอร์บอลิกก็เช่นกันสามารถที่จะกำหนดค่าเริ่มต้นและ

คำนวณหาผลลัพธ์ได้ด้วยวิธีการเดียวกันนี้นอกจากนี้ยังสามารถนำไปหาค่าของฟังก์ชันพิเศษอื่นๆ โดยผ่านความสัมพันธ์ตามสมการที่ (2.25)

$$
\begin{aligned}
e^{z} &= \cosh z + \sinh z \\
e^{-z} &= \cosh z - \sinh z \\
\ell nw &= 2\tanh^{-1}\left(\frac{w-1}{w+1}\right) \\
\sqrt{w} &= \sqrt{\left(w+\frac{1}{4}\right)^{2} - \left(w-\frac{1}{4}\right)^{2}} \\
\tan w &= \frac{\sin w}{\cos w} \\
\tanh w &= \frac{\sinh w}{\cosh w}
\end{aligned}
\tag{2.25}
$$

และเพื่อให้ง่ายต่อการสร้างของ CORDIC ขั้นตอนการเพิ่มของ $\delta_{m,i}$ ในแต่ละรอบการคำนวณจึงพิจารณาในเทอมกำลังของสองคือ

$$
\delta_{m,i} = 2^{-s(m,i)} \quad ; i \in \{0,1,2,\ldots,n-1\} \tag{2.26}
$$

ในส่วนของ $\delta(m,i)$ ก็จะมีความแตกต่างกันไปตามระบบของพิกัดนั้นๆซึ่งในกรณีของไฮเปอร์บอลิกจะทำการคำนวณซ้ำที่ $i = 4,13,40,\ldots k, 3k+1$ จึงจะให้ค่าที่ใกล้เคียงค่าจริงและ $\tanh^{-1}(2^{0})$ จะไม่มีในข้อกำหนด(หาค่าไม่ได้)ดังนั้นสำหรับที่ค่า $m=-1$ จึงเริ่มที่ $i=1$ ในตารางข้างล่างนี้จะแสดงตัวอย่างของค่า $\delta=(m,i)$ ในแต่ละพิกัดและค่า k_m ที่จะได้ดังนี้

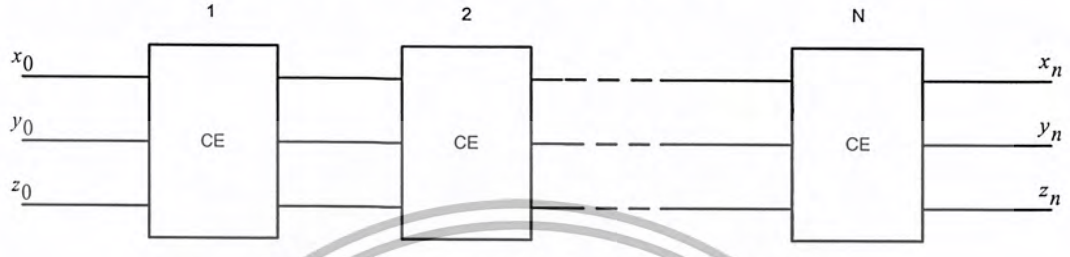
m	$s(m,i)$	k_m
1	0,1,2,3,4,5,...,i,...	0.607253
0	1,2,3,4,5,6,...,$i+1$,...	1.000000
-1	1,2,3,4,4,5,...,12,13,13,14,...	1.207497

ตารางที่ 2.4 แสดงค่าของตัวแปรที่ใช้ในแต่ละโหมด

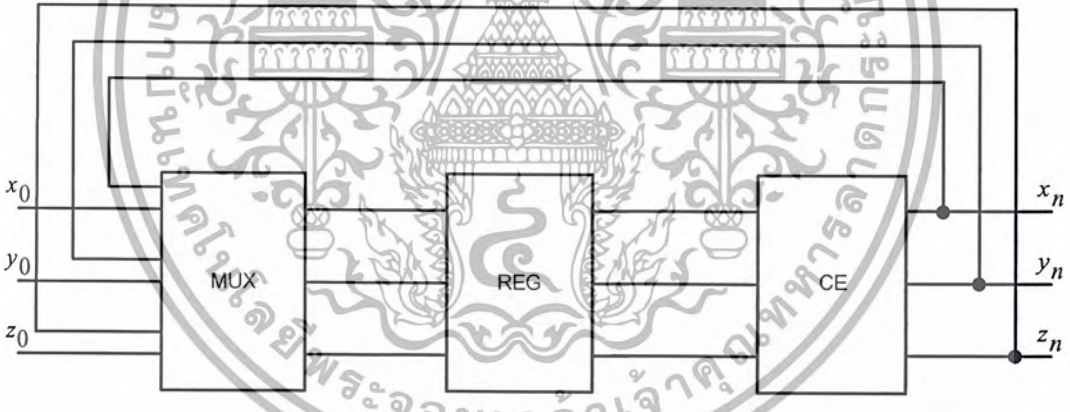
2.1.2โครงสร้างทางฮาร์ดแวร์ของ CORDIC algorithm

หลักการของ CORDIC algorithm จะสามารถนำไปสร้างเป็นชุดคำนวณของสมการ CORDIC แต่ละชุดต่อลำดับกันเป็นจำนวน n ลำดับซึ่งถ้าพิจารณาตามสมการที่ (2.22) จะเห็นว่าแต่ละชุดจะต้องมีการกำหนดค่าที่ต้องการหา,ค่าตัวแปรที่กำหนดพิกัดของระบบ(m)และระดับของตำแหน่งค่า i

ในรูปที่ 2.4 จะแสดงลำดับของชุดการคำนวณ CORDIC แต่ละชุดหรือฮาร์ดแวร์อาจจะมีโครงสร้างในลักษณะที่เป็นแบบวนค่าในการคำนวณดังเช่นในรูปที่ 2.5



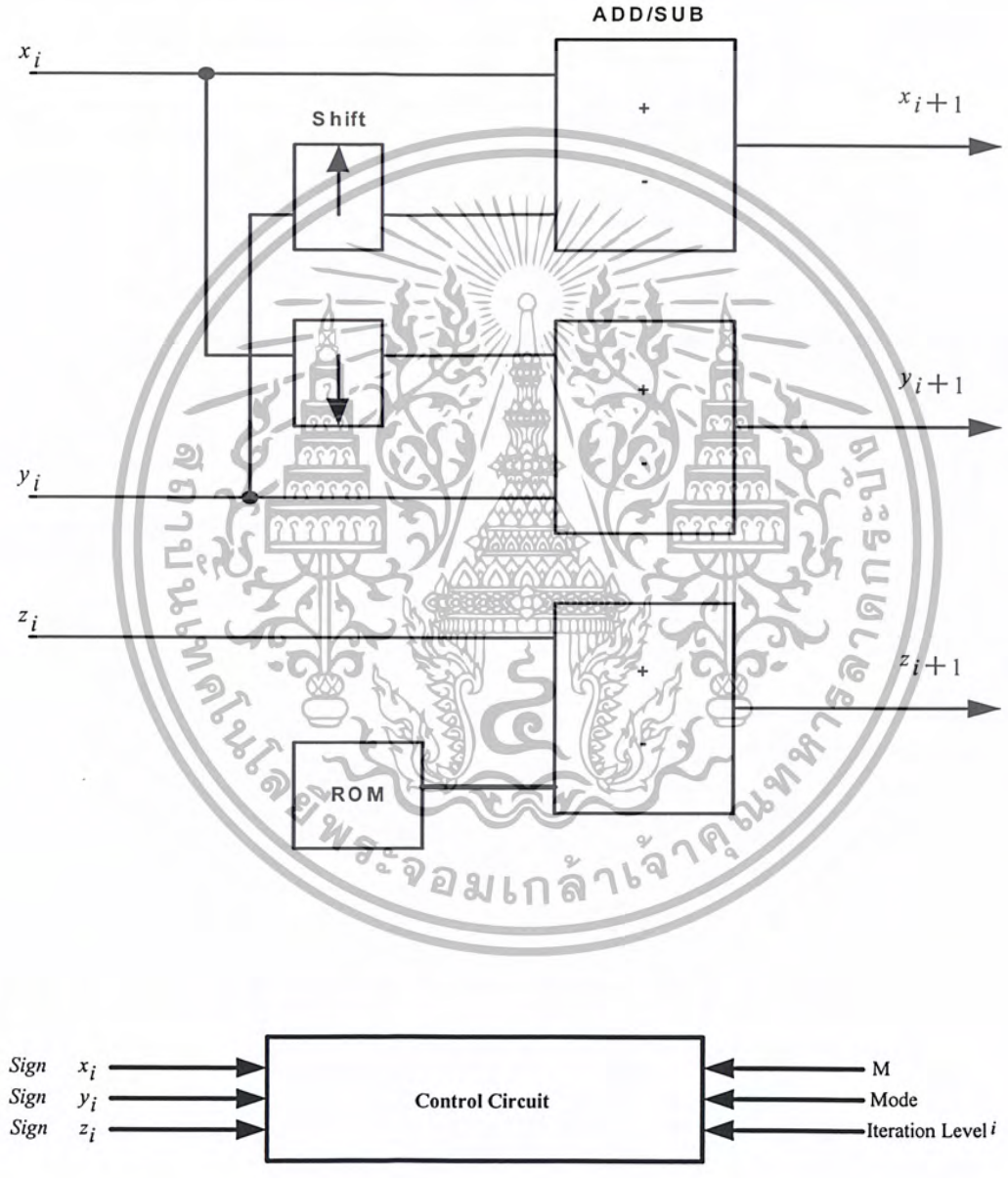
รูปที่ 2.4 โครงสร้างโดยการต่อชุด CORDIC element (CE) ลำดับกัน



รูปที่ 2.5 โครงสร้างแบบวนค่ากลับ

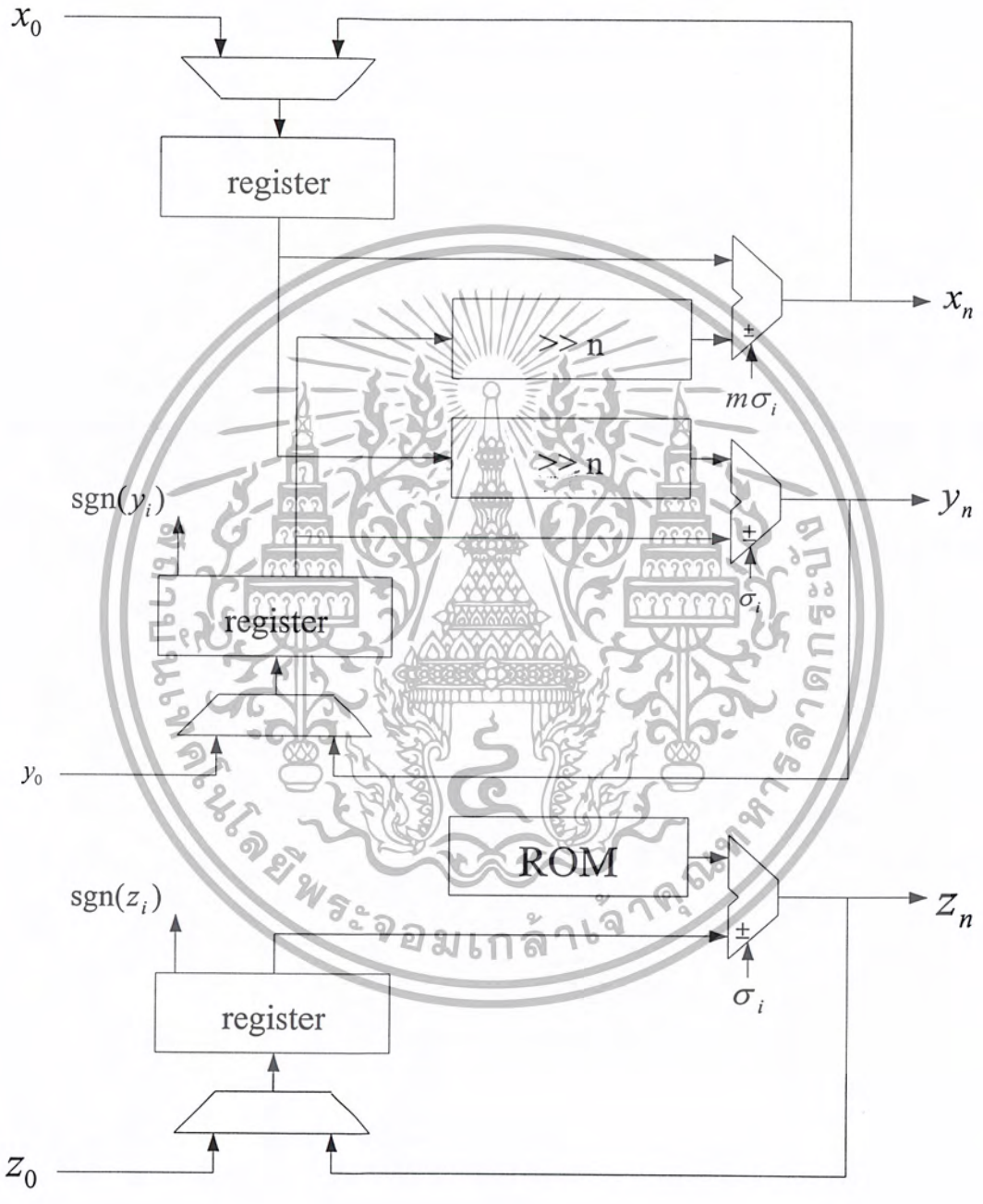
ส่วนของการสร้างชุดคำนวณของสมการ CORDIC แต่ละชุดนั้นจะพิจารณาการคำนวณในรูปของเลขฐานสอง(binary)ซึ่งจากสมการที่ (2.22) จะเห็นว่าแต่ละสมการจะมีเทอมของการบวกและลบ (adder and subtracter) อยู่สองเทอมใหญ่ๆและส่วนของเทอมย่อยจะมีการคูณของตัวแปรเกิดขึ้นซึ่งตัวแปร σ_i และ m จะเป็นตัวแปรที่กำหนดเครื่องหมายและในระบบเลขฐานสองการเปลี่ยนเครื่องหมายจะใช้วิธีการทำ two' complement ดังนั้นชุดนี้จึงเพียงแค่เปรียบเทียบเครื่องหมายและส่งไปควบคุมการทำ

two' complement ส่วนอีกตัวแปรคือ $\delta_{m,i}$ ซึ่งก็คือค่าของ 2^{-i} โดยที่ค่าของ i จะมีค่าแตกต่างกันไปตามระบบพิกัดที่ทำการคำนวณ แต่อย่างไรก็ตามเมื่อพิจารณาในระบบเลขฐานสองแล้วจะเห็นว่าค่าของ 2^{-i} ก็คือการเลื่อน(shift)ไปทางด้านขวามือเป็นจำนวน i ตำแหน่งนั่นเองซึ่งหมายความว่าค่าของ x_i และ y_i ในสมการที่ (2.22) ที่ถูกคูณอยู่กับตัวแปรเหล่านี้จะไม่ถูกนำเข้าวงจรคูณโดยตรงแต่จะทำการเลื่อนและทำ two'complement ค่านั้นๆแทนซึ่งนับเป็นข้อดีของ CORDIC algorithm ดังนั้นสามารถเขียนโครงสร้างในส่วนของการคำนวณ CORDIC แต่ละชุดได้ดังรูปที่ 2.6

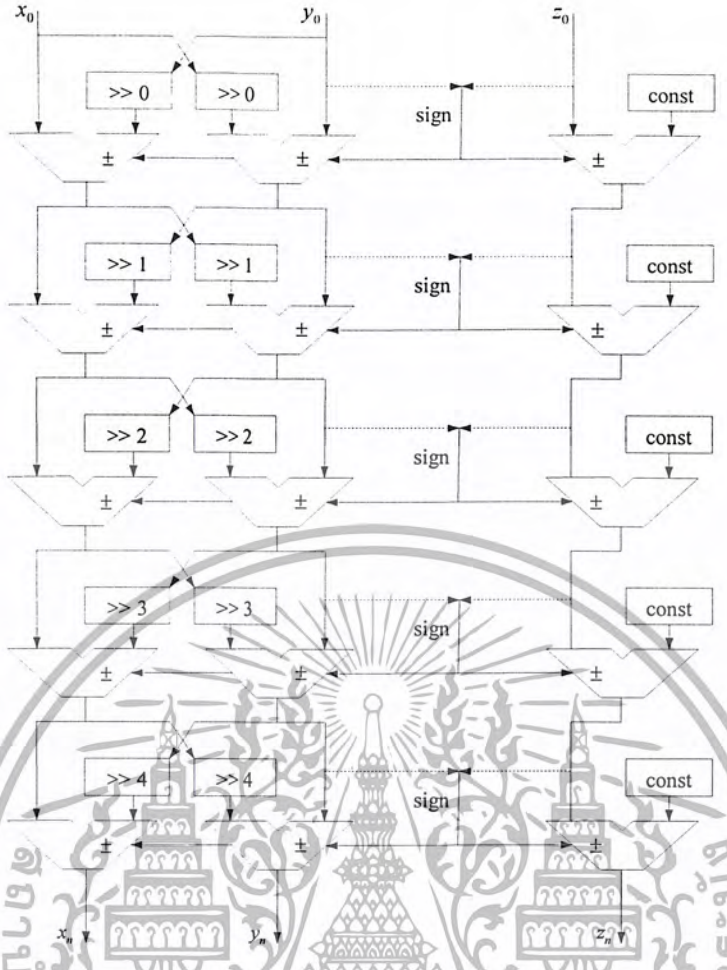


รูปที่ 2.6 โครงสร้างของชุดคำนวณ CORDIC หนึ่งชุด

โดยที่ ROM จะเป็นตัวเก็บค่าของ $\tan^{-1}(2^{-i})$ ที่ $i = 0,1,2,3,...,N$ ตามจำนวนชุดการคำนวณ และ Control Circuitry จะเป็นตัวกำหนดฟังก์ชันต่างๆรวมถึงพิจารณาเครื่องหมายเพื่อส่งไปควบคุมแต่ละชุดอีกด้วย



รูปที่ 2.7 โครงสร้างของ CORDIC algorithm แบบ loop



รูปที่ 2.8โครงสร้างของ CORDIC algorithm แบบ cascade

ในรูปที่ 2.7 และ 2.8 เป็นตัวอย่างแสดงโครงสร้างของ CORDIC algorithm ที่จะถูกสร้างในการนำไปใช้งานจริงซึ่งในรูปจะแสดงในส่วนที่เป็นโครงสร้างของการคำนวณเท่านั้นและสามารถที่จะเพิ่มหรือลดจำนวนรอบการทำงานได้ตามความเหมาะสมทั้งนี้ทั้งนั้นการเปลี่ยนแปลงดังกล่าวก็จะมีผลต่อจำนวนอุปกรณ์ที่นำมาใช้สร้างเป็นฮาร์ดแวร์และความแม่นยำในการคำนวณเป็นต้น ดังนั้นจึงควรพิจารณาเลือกออกแบบใช้งานให้เหมาะสม

2.2 Discrete Cosine Transform (DCT)

คือการแปลงข้อมูลให้อยู่ในรูปแบบของโดเมนความถี่ เพื่อทำให้ค่าที่ได้ออกมาเป็นอิสระต่อกันและลดขนาดของข้อมูลลงจากการที่ค่าพลังงานส่วนใหญ่ของข้อมูลจะอยู่ในช่วงความถี่ต่ำ โดย DCT เป็นการแปลงแบบเชิงเส้นที่มีค่าสัมประสิทธิ์ในการแปลงคงที่ และค่าที่ได้ในโดเมนความถี่จะมีเฉพาะจำนวนจริงเท่านั้น ซึ่งข้อมูลส่วนใหญ่ที่จะใช้การแปลงแบบ DCT จะเป็นได้ทั้งข้อมูลแบบหนึ่งมิติ (1-Dimension) และข้อมูลแบบสองมิติ (2-Dimensions) สมการที่ใช้ในการแปลงด้วย DCT สามารถแสดงได้ดังนี้

หนึ่งมิติ (1-D)

$$y(n)=e(n)\sum_{k=0}^{N-1}x(k)\cos\left[\frac{(2k+1)n\pi}{2N}\right] \qquad ;n=0,1,.....,N-1 \tag{2.27}$$

สองมิติ (2-D)

$$y(m,n)=\frac{4}{N^2}e(m)e(n)\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}x(k,l)\cos\left[\frac{(2k+1)m\pi}{2N}\right]\cos\left[\frac{(2l+1)n\pi}{2N}\right] \tag{2.28}$$

$m=0,1,.....,N-1$

$n=0,1,.....,N-1$

$$e(n)=\begin{cases}\frac{1}{\sqrt{2}} & ;n=0\\ 1 & ;ete\end{cases}$$

ส่วนสมการการแปลงกลับแบบ DCT สามารถแสดงได้ดังนี้

หนึ่งมิติ (1-D)

$$x(k)=\frac{2}{N}\sum_{n=0}^{N-1}e(n)y(n)\cos\left(\frac{(2k-1)n\pi}{2N}\right) \qquad ;k=0,1,.....,N-1 \tag{2.29}$$

สองมิติ (2-D)

$$x(k,l)=\frac{2}{N}\sum_{n=0}^{N-1}\sum_{n=0}^{N-1}e(m)e(n)y(m,n)\cos\left(\frac{(2k-1)m\pi}{2N}\right)\cos\left(\frac{(2l-1)n\pi}{2N}\right) \tag{2.30}$$

$k=0,1,.....,N-1$

$l=0,1,.....,N-1$

การจะทำเป็นตัวอุปกรณ์ของ DCT 1 มิติ ที่มีขนาด N จะต้องใช้ตัวคูณและตัวบวกรวมกันแล้วทั้งหมดมี N^2 ตัว ส่วนของ DCT 2 มิติ ที่มีขนาด N×N จะต้องใช้ตัวคูณและตัวบวกรวมกันแล้วทั้งหมดมี N^4 ตัว แต่สามารถทำการลดจำนวนตัวคูณและตัวบวกลงได้ด้วยการแยกจาก DCT 2 มิติ ให้มาเป็น DCT 1 มิติ 2 ตัว โดยอาศัยคุณสมบัติของการแปลง DCT ที่เป็น separable transform ซึ่งจะประกอบด้วย

การแปลงแนวนอนของเมตริก แล้วตามด้วยการแปลงแนวตั้ง การใช้วิธีนี้จะทำให้ได้ DCT 2 มิติ ที่มีขนาด N×N จะใช้ตัวคูณและตัวบวกรวมกันแล้วทั้งหมดเพียง $2N^3$ ตัว

2.2.1 สมการของ DCT

สมการที่ใช้ในการแปลงแบบ DCT และการแปลงกลับแบบ DCT ขนาด N สามารถหามาได้จากสมการของ Discrete Fourier Transform (DFT) ขนาด 2N ให้ $z(k)$ เป็นลำดับของสัญญาณที่มีขนาด 2N ของ $x(k)$ ได้ดังนี้

$$z(k)=x(k)+x(2N-k-1)=\begin{cases} x(k) & ; \quad 0\le k\le N-1 \\ x(2N-k-1) & ; \quad N\le k\le 2N-1 \end{cases} \tag{2.31}$$

จากสมการ (2.31) ทำการแปลงแบบ DFT ขนาด 2N

$$\begin{aligned} Z_D(n) &= \sum_{k=0}^{2N-1} z(k)\cdot e^{-j\frac{2\pi}{2N}kn} \\ &= \sum_{k=0}^{N-1} x(k)\cdot e^{-j\frac{2\pi}{2N}kn} + \sum_{k=N}^{2N-1} x(2N-k-1)\cdot e^{-j\frac{2\pi}{2N}kn} \end{aligned} \tag{2.32}$$

โดยที่ $0\le n\le 2N-1$ และทำการแทนค่า $k=2N-k'-1$ ในพจน์ที่สองของสมการ (2.32)

$$\begin{aligned} \sum_{k=N}^{2N-1} x(2N-k-1)\cdot e^{-j\frac{2\pi}{2N}kn} &= \sum_{k'=N-1}^{0} x(k')\cdot e^{-j\frac{2\pi}{2N}n(2N-k'-1)} \\ &= \sum_{k'=0}^{N-1} x(k')\cdot e^{j\frac{2\pi}{2N}nk'}\, e^{j\frac{2\pi}{2N}n} \end{aligned} \tag{2.33}$$

นำสมการ (2.33) ไปแทนในสมการ (2.32) ได้

$$Z_D(n)=\sum_{k=0}^{N-1} x(k)\cdot e^{-j\frac{2\pi}{2N}kn} + \sum_{k=0}^{N-1} x(k)\cdot e^{j\frac{2\pi}{2N}nk}\, e^{j\frac{2\pi}{2N}n}$$

$$= e^{j\frac{n\pi}{2N}} \cdot \left(\sum_{k=0}^{N-1} x(k) \cdot e^{-j\frac{(2k+1)n\pi}{2N}} + \sum_{k=0}^{N-1} x(k) \cdot e^{j\frac{(2k+1)n\pi}{2N}} \right)$$

$$= e^{j\frac{n\pi}{2N}} \cdot \sum_{k=0}^{N-1} 2x(k) \cos\left(\frac{(2k+1)n\pi}{2N} \right) \tag{2.34}$$

นิยามให้

$$\hat{y}(n) = \begin{cases} Z_D(n) \cdot e^{-j\frac{n\pi}{2N}} & ; \quad 0 \le n \le N-1 \\ 0 & ; \quad etc \end{cases} \tag{2.35}$$

ดังนั้นสมการของการแปลงแบบ DCT ขนาด N สามารถแสดงได้ดังนี้

$$y(n) = \frac{e(n)\hat{y}(n)}{2}$$

การแปลงกลับแบบ DCT นั้น ได้จากความสัมพันธ์ของ $Z_D(n)$ กับ $y(n)$ โดยที่หา $z(k)$ ได้มาจาก $Z_D(n)$ ที่ทำการแปลงกลับแบบ DFT และหา $x(k)$ ได้จาก $z(k)$ แม้ว่า $Z_D(n)$ เป็นสัญญาณขนาด 2N และ $y(n)$ เป็นสัญญาณขนาด N ก็สามารถหา $Z_D(n)$ ได้จาก $y(n)$
โดยที่

$$Z_D(n) = \begin{cases} e^{j\frac{n\pi}{2N}} \cdot \hat{y}(n) & ; \quad 0 \le n \le N-1 \\ 0 & ; \quad N+1 \le n \le 2N-1, 1 \le 2N-n \le N-1 \end{cases}$$

ดังนั้น

$$Z_D(2N-n) = e^{j\frac{(2N-n)\pi}{2N}} \cdot \hat{y}(2N-n) = -e^{-j\frac{n\pi}{2N}} \cdot \hat{y}(2N-n) \tag{2.36}$$

แต่จากสมการ (2.34)

$$Z_D(2N-n) = e^{j\frac{(2N-n)\pi}{2N}} \cdot \sum_{k=0}^{N-1} 2x(k)\cos\left(\frac{(2k+1)(2N-n)\pi}{2N}\right)$$

$$= -e^{j\frac{2N\pi}{2N}} \cdot e^{-j\frac{n\pi}{2N}} \cdot \sum_{k=0}^{N-1} 2x(k)\cos\left(\frac{(2k+1)n\pi}{2N}\right)$$

$$= e^{-j\frac{2n\pi}{2N}} \cdot e^{j\frac{n\pi}{2N}} \cdot \sum_{k=0}^{N-1} 2x(k)\cos\left(\frac{(2k+1)n\pi}{2N}\right)$$

$$= e^{-j\frac{2n\pi}{2N}} \cdot Z_D(n) \qquad (2.37)$$

ดังนั้นจะได้

$$Z_D(n) = e^{j\frac{2n\pi}{2N}} \cdot Z_D(2N-n)$$

$$= -e^{j\frac{2n\pi}{2N}} \cdot e^{-j\frac{n\pi}{2N}} \cdot \hat{y}(2N-n)$$

$$= -e^{j\frac{n\pi}{2N}} \cdot \hat{y}(2N-n) \qquad (2.38)$$

$$Z_D(n) = \begin{cases} e^{j\frac{n\pi}{2N}} \cdot \hat{y}(n) & ; \quad 0 \le n \le N-1 \\ 0 & ; \quad n = N \\ -e^{j\frac{n\pi}{2N}} \cdot \hat{y}(2N-n) & ; \quad N+1 \le n \le 2N-1 \end{cases} \qquad (2.39)$$

ทำการแปลงกลับแบบ DFT ของ $Z_D(n)$ จะได้

$$z(k) = \frac{1}{2N} \sum_{n=0}^{2N-1} Z_D(n) \cdot e^{j\frac{2\pi}{2N}nk}$$

$$=\frac{1}{2N}\left(\sum_{n=0}^{N-1}\hat{y}(n)\cdot e^{j\frac{(2k+1)n\pi}{2N}}+\sum_{n=N+1}^{2N-1}\left(-e^{j\frac{n\pi}{2N}}\cdot\hat{y}(2N-n)\right)\cdot e^{j\frac{2\pi}{2N}nk}\right) \quad (2.40)$$

จากสมการ (2.4จ) ทำการจัดรูปของค่าในผลบวกในเทอมที่ 2 จากนั้นใช้คุณสมบัติของ $\frac{1}{e(0)}=2e(0)$ และ $\frac{1}{e(n)}=e(n)$; $n\neq 0$ สามารถเขียนสมการได้ใหม่ดังนี้

$$z(k)=\frac{1}{2N}\left(\sum_{n=0}^{N-1}\hat{y}(n)\cdot e^{j\frac{(2k+1)n\pi}{2N}}+\sum_{n=1}^{N-1}\hat{y}(n)\cdot e^{-j\frac{(2k+1)n\pi}{2N}}\right)$$

$$=\frac{1}{2N}\left(\hat{y}(0)+2\sum_{n=1}^{N-1}\hat{y}(n)\cos\left(\frac{(2k+1)n\pi}{2N}\right)\right)$$

$$=\frac{1}{2N}\left(e(0)y(0)+\sum_{n=1}^{N-1}e(n)y(n)\cos\left(\frac{(2k+1)n\pi}{2N}\right)\right);0\leq k\leq 2N-1 \quad (2.41)$$

จะได้การแปลงกลับแบบ DCT ขนาด N จาก $z(k)$ ดังนี้

$$x(k)=z(k)=\frac{2}{N}\sum_{n=0}^{N-1}e(n)y(n)\cos\left(\frac{(2k+1)n\pi}{2N}\right);0\leq k\leq N-1 \quad (2.42)$$

สัญญาณของ $x(k)$ และ $y(n)$ ที่มีขนาดเท่ากับ N สามารถแสดงในรูปแบบของเมตริกได้ดังนี้

$$x=\begin{bmatrix}x(0)\\x(1)\\.....\\x(N-1)\end{bmatrix},\qquad y=\begin{bmatrix}y(0)\\y(1)\\.....\\y(N-1)\end{bmatrix} \quad (2.43)$$

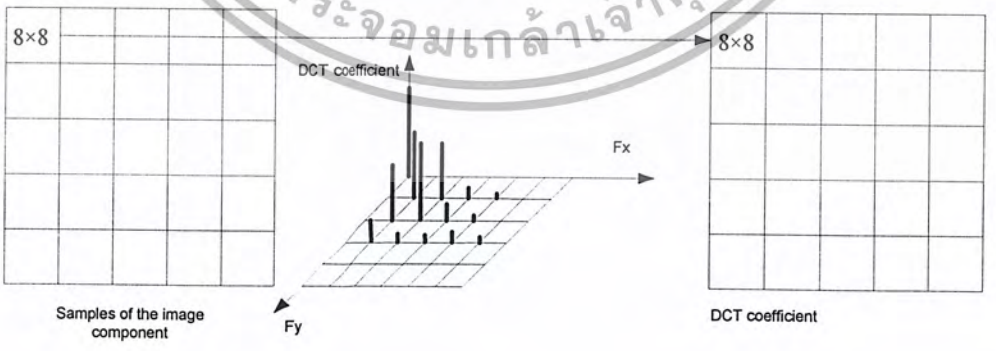
และค่าสัมประสิทธิ์ที่ใช้ในการแปลงแบบ DCT สามารถแสดงอยู่ในรูปแบบของเมตริกดังนี้

$$\Lambda=\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos\left(\frac{\pi}{2N}\right) & \cos\left(\frac{3\pi}{2N}\right) & \cdots & \cos\left(\frac{(2N-1)\pi}{2N}\right) \\ \cdots & \cdots & \cdots & \cdots \\ \cos\left(\frac{(N-1)\pi}{2N}\right) & \cos\left(\frac{3(N-1)\pi}{2N}\right) & \cdots & \cos\left(\frac{(2N-1)(N-1)\pi}{2N}\right) \end{bmatrix} \quad (2.44)$$

ดังนั้น สามารถทำการหาค่าของการแปลงแบบ DCT และการแปลงกลับแบบ DCT จากการคำนวณในรูปแบบของเมตริกได้ดังนี้

$$y=\Lambda\cdot x \quad , \quad x=\frac{2}{N}\cdot\Lambda^{T}\cdot y \quad (2.45)$$

ในการพิจารณาให้เห็นการทำงาน โดยยกตัวอย่างของข้อมูลภาพซึ่งจะถูกแบ่งออกเป็นบล็อกย่อยๆ ขนาด 8×8 จุดเพื่อที่จะทำการแปลงทีละชุดแบบสองมิติ (2-Dimensions) และข้อมูลที่มีความสำคัญสูง (เทอมความถี่ต่ำ) จะอยู่ในด้านมุมซ้ายบน โดยที่ตำแหน่ง (0, 0) จะเป็นองค์ประกอบของไฟ DC ส่วนที่เหลือจะกระจายออกไปเป็นองค์ประกอบที่มีความถี่สูงขึ้นตามแกน x และ y ดังที่แสดงในรูปที่ 2.9



รูปที่ 2.9 ลักษณะของการแปลง DCT ทีละ 8×8 จุดจากข้อมูลทั้งหมด โดยมีเทอมความถี่ต่ำอยู่มุมบนด้านซ้าย

140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

(ก) ข้อมูลภาพขนาด 8×8

12	16	19	12	12	27	51	47
16	24	12	19	12	20	39	51
24	27	8	39	35	34	24	44
40	17	28	32	24	27	8	32
34	20	28	20	12	8	19	34
19	39	12	27	27	12	8	34
8	28	-5	39	34	16	12	19
20	27	8	27	24	19	19	8

(ข) ข้อมูลภาพขนาด 8×8 ที่ทำการสเกลให้อยู่ในช่วง -128 ถึง 127

185	-17	14	-8	23	-9	13	-18
20	-34	26	-9	-10	10	13	6
-10	-23	-1	6	-18	3	-20	0
-8	-5	14	-14	-8	-2	-3	8
-3	9	7	1	-11	17	18	15
3	-2	-18	8	8	-3	0	-6
8	0	-2	3	-1	-7	-1	-1
0	-7	-2	1	1	4	-6	0

(ค) ข้อมูลภาพที่ผ่านการแปลงแบบ DCT

รูปที่ 2.10 แสดงลักษณะของข้อมูลก่อนและหลังการแปลงแบบ DCT

ซึ่งจากผลการแปลง DCT ของข้อมูลภาพที่แสดงในรูปที่ 2.10 จะเห็นว่าผลที่ได้จะมีข้อมูลส่วนใหญ่อยู่ในเทอมความถี่ต่ำ และทำการกระจายไปในเทอมความถี่สูง (มุมล่างด้านขวาของเมตริกซ์) ซึ่งจะมีค่าน้อยมากเมื่อเทียบกับเทอมความถี่ต่ำที่มีความสำคัญมากกว่า ซึ่งนอกจากนี้ยังสามารถนำข้อมูลที่ได้จากการแปลงแบบ DCT ไปผ่านการควอนไทซ์ เพื่อทำให้ข้อมูลบริเวณความถี่สูงมีค่าเข้าใกล้ศูนย์ ทำให้สามารถลดขนาดของข้อมูลลงได้

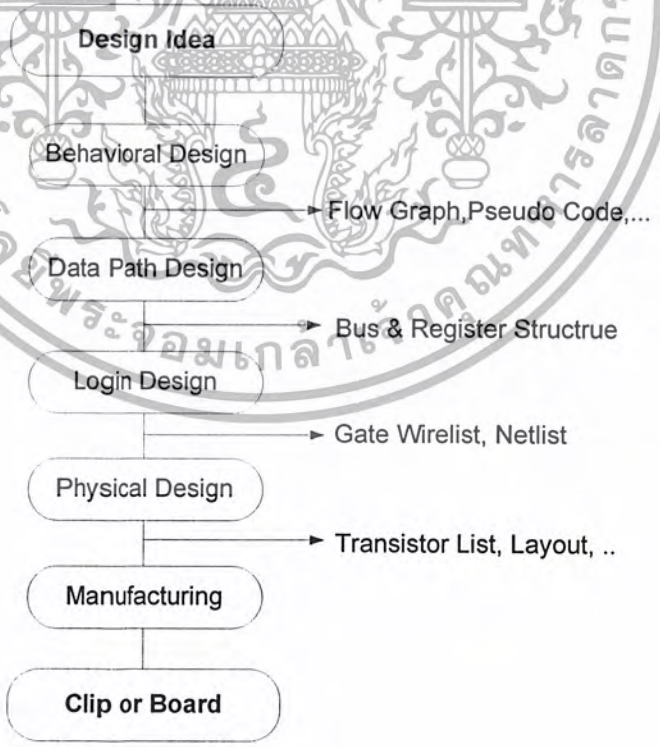
2.3 ภาษาวีเอชดีแอล

ความซับซ้อนและขนาดของระบบดิจิตอลในปัจจุบันได้เพิ่มมากขึ้นทุกขณะ ส่งผลให้มีการนำคอมพิวเตอร์เพื่อช่วยในการออกแบบ มาใช้ในกระบวนการออกแบบฮาร์ดแวร์เพิ่มขึ้นเช่นกัน อีกทั้งอุปกรณ์และวิธีการออกแบบใหม่ๆ ก็ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบมากขึ้นด้วย สำหรับภาษาบรรยายอุปกรณ์ฮาร์ดแวร์ เอชดีแอล (HDL: Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการออกแบบระบบดิจิตอลเป็นไปอย่างมีประสิทธิภาพ

2.3.1 การออกแบบระบบดิจิตอล

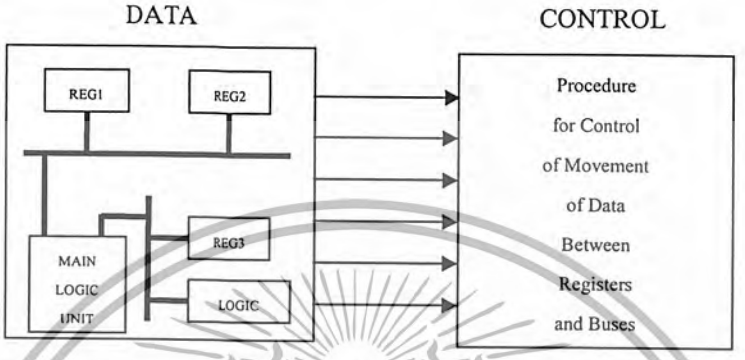
ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำเนิดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตนต่างๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้นก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป

รูปที่ 2.11 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบ แล้วทำการพัฒนาให้สามารถนำไปใช้ได้อย่างสมบูรณ์ ซึ่งภายในขั้นตอนนี้ผู้ออกแบบจำเป็นต้องสร้างรูปแบบระบบเชิงพฤติกรรมขึ้นมาตรวจสอบซึ่งอาจจะเป็นผังงานแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo code) ก็ได้



รูปที่ 2.11 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์และวงจรลอจิกที่จำเป็นทั้งหมด เพื่อนำมาประกอบเป็นระบบที่สมบูรณ์ โดยแต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรลอจิกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 2.12



รูปที่ 2.12 การออกแบบระบบเส้นทางข้อมูล

ขั้นตอนถัดมาเป็นการออกแบบวงจรลอจิกซึ่งจะเกี่ยวข้องกับการนำเกทดิจิตอลพื้นฐาน และฟลิปฟลอป (Flip – Flop) มาประกอบเป็นอุปกรณ์ย่อยต่าง ๆ เช่นรีจิสเตอร์เก็บข้อมูล บัสวงจรลอจิก และส่วนควบคุมฮาร์ดแวร์ ซึ่งผลลัพธ์ที่ได้ในขั้นตอนนี้จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง

การออกแบบในขั้นตอนถัดไปเป็นการเปลี่ยนเครือข่ายการโยงใยที่ได้จากขั้นตอนที่แล้ว ให้เป็นลำดับของทรานซิสเตอร์ (Transistor List) และ โครงงาน (Layout) ซึ่งขั้นตอนนี้จะเกี่ยวข้องกันโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อแทนเกทและฟลิปฟลอปต่าง ๆ

และในขั้นตอนสุดท้ายจะเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

2.3.2 ประวัติความเป็นมาของภาษาวีเอชดีแอล

วีเอชดีแอล ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบด้วยเหตุผลนี้จึงทำให้ภาษาวีเอชดีแอล เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกรได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชั่นการทำงานของวงจรอย่างสังเขป โดยไม่คำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนี้

วีเอชดีแอล ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มีความซับซ้อนได้ทั้งหมด ดังนั้น วีเอชดีแอล จึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง สำหรับมาตรฐานของภาษาที่ใช้บรรยายพฤติกรรมวงจร หรือฮาร์ดแวร์ของระบบสำหรับโครงการ ไอซีวีเอชเอส ที่ ดีโอดี ได้ให้ไว้สามารถสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่น เดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรมภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า ภาษาโปรแกรมระดับสูง

2.3.3 องค์ประกอบพื้นฐานต่างๆ ของภาษาวีเอชดีแอล

ในการเขียนรูปแบบภาษาบรรยายระบบดิจิตอล ในมุมมองของการออกแบบลักษณะบนลงล่าง จะต้องทำความเข้าใจในเรื่องของโครงสร้างและส่วนประกอบต่างๆ ของรูปแบบภาษาวีเอชดีแอลเสียก่อน ซึ่งส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียน คือ

1) หน่วยการออกแบบเอนทิตี้ (Entity Design Unit)

หน่วยการออกแบบนี้ เป็นส่วนที่ใช้สำหรับติดต่อระหว่างโลกภายนอกกับรูปแบบที่เขียนขึ้น ที่เรียกว่า หน่วยการออกแบบเอนทิตี้ ในส่วนนี้ใช้กำหนดจุดเชื่อมต่อ ของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆ ของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น รูปที่2.13 แสดงให้เห็นโครงสร้างอย่างง่ายๆ ของ หน่วยการออกแบบเอนทิตี้

```
ENTITY component_name IS
        Input and output ports
        Physical and other parameter
END [component_name];
```

รูปที่ 2.13 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี้

ส่วนนี้จะขึ้นต้นด้วยคำว่า ENTITY และ IS ระหว่างคำทั้งสองเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component_name) สำหรับการตั้งชื่อนั้นต้องเป็นไปตามกฎเกณฑ์ของภาษาหลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและออกข้อมูล (input – output) รวมทั้งพารามิเตอร์อื่นๆ ส่วนนี้เรียกว่าส่วนหัว (entity header) และที่สำคัญคือ หน่วยการออกแบบเอนทิตี้จะต้องปิดท้ายด้วยคำว่า END และเครื่องหมายอัฒภาคเสมอ (;)

2) หน่วยการออกแบบสถาปัตยกรรม (Architecture Design Unit)

คือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบ ในมุมมองของการจำลองการทำงานพฤติกรรมต่างๆ ที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออก ตรงช่องทางตลอดจนพารามิเตอร์ต่างๆ ที่กำหนดในหน่วยการออกแบบเอนทิตี้ รูปที่ 2.14 แสดงให้เห็นถึงโครงสร้างอย่างง่ายๆของหน่วยการออกแบบสถาปัตยกรรม

```
ARCHITECTURE identifier OF component_name IS
[declaration]
BEGIN
        specification of the functionality of the
        component in terms of its input lines and as
        influenced by physical and other parameters
END [identifier];
```

รูปที่ 2.14 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม

ส่วนของหน่วยการออกแบบสถาปัตยกรรม เริ่มต้นด้วยคำว่า ARCHITECTURE และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า ARCHITECTURE นั้นใช้บรรยายหน่วยการออกแบบเอนทิตี้ใดๆ (OF < entity design unit > IS) ส่วนที่อยู่ระหว่าง ARCHITECTURE และ BEGIN เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (architecture declarative area) ที่เป็นเพียงส่วนเผื่อเลือก (option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าต่างๆ ที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ อาทิเช่นประเภท (type) ต่างๆ (ตัวอย่างเช่น bit และ bit_vector) สัญญาณ (signal) ตัวคงที่ (constant) โปรแกรมย่อย (ได้แก่ function และ procedure) และอุปกรณ์(component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า และไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง PORT) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า BEGIN กับ END ของหน่วยการออกแบบสถาปัตยกรรม และนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (concurrent statement) เท่านั้น หน่วยการออกแบบสถาปัตยกรรมจะต้องปิดท้ายด้วยคำสั่ง END และชื่อของสถาปัตยกรรมนั้นๆ ที่เป็นส่วนเผื่อเลือกโดยทั่วไปการเขียนรูปแบบระบบดิจิตอลด้วยภาษาวีเอชดีแอลสามารถเขียนได้ในลักษณะต่างๆ ดังนี้

-ประเภทการไหลของข้อมูล (Dataflow description)

-ประเภทพฤติกรรม (Behavioral description)

-ประเภทโครงสร้าง (Structure description)

-ประเภทผสม (Mixed model description)

3) หน่วยการออกแบบแพ็กเก็ต (Package Design Unit)

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย (subprogram) ที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิตอล สามารถเก็บไว้ในส่วนที่เรียกว่า package ได้ และข้อมูลเหล่านี้สามารถถูกเรียกไปใช้ได้โดย entity design unit architecture design unit หรือจาก package design unit อื่นๆ ด้วยชุดคำสั่ง USE statement นอกจากนั้นสิ่งที่นิยมทำกันมากคือรูปแบบ (model) มาตรฐานต่างๆ อาทิเช่น standard components (model ของ IC ตระกูล 74xx) จะถูกเก็บไว้ในแพ็กเก็ต ที่ทุกคนสามารถเข้าถึงและนำไปใช้ได้

โดยปกติแล้ว แพ็กเก็ตจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็กเก็ต (Package declaration) และส่วนของบอดี้แพ็กเก็ต (Package body) เนื่องจาก แพ็กเก็ตถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้นการที่นำแพ็กเก็ตไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษาวีเอชดีแอลสามารถกระทำได้ด้วยชุดคำสั่ง USE

- การประกาศแพ็กเก็ต (Package Declaration)

ส่วนที่มีความสำคัญที่สุดของแพ็กเก็ต (มองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ การประกาศแพ็กเก็ต เพราะจะเป็นส่วนที่กำหนดชื่อ (identifier) ของสิ่งที่ประกาศอยู่ภายในแพ็กเก็ต สำหรับนำไปใช้ภายนอกตัวของแพ็กเก็ตเอง ถ้าสิ่งใดๆ ถูกประกาศในส่วนของบอดี้แพ็กเก็ต แต่ไม่ถูกประกาศในการประกาศแพ็กเก็ตจะไม่สามารถถูกนำค่า และพฤติกรรมไปใช้จากส่วนนอกได้ ซึ่งสามารถเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศเอนทิตี้ คือ จุดเชื่อมต่อ ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเก็ตสามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้และยังสามารถถูกนำไปใช้จากรูปแบบ (model) ภายนอกได้เช่นใช้สำหรับประกาศ ชนิด (TYPE) เช่นเดียวกันกับบอดี้แพ็กเก็ตที่ไม่จำเป็นต้องมีการประกาศแพ็กเก็ต แต่แพ็กเก็ตนั้นจะไม่สามารถนำไปใช้ ในรูปแบบ (model) อื่นได้ การเขียนการประกาศแพ็กเก็ต มีกฎเกณฑ์ตามที่แสดงในรูปที่ 2.15

```
PACKAGE package_name IS
        package_declarative_part
END package_name ;
```

รูปที่ 2.15 แสดงโครงสร้างโดยทั่วไปของส่วนการประกาศแพ็กเก็ต

- โครงสร้างของแพ็กเก็ต (Package Body)

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (Subprogram) ทั้งหลายที่ชื่อของโปรแกรมย่อยนั้นๆ ที่ถูกประกาศไปในส่วนของการประกาศแพ็กเก็ต แล้วจะถูกเก็บไว้ในส่วนบอดี้แพ็กเก็ต แต่ถูกกำหนดค่าคงที่ต่างๆ อันได้แก่ตัวค่าคงที่ที่

ถูกประกาศชื่อก่อนในส่วนของการประกาศแพ็กเก็ต แต่ถูกกำหนดค่าในส่วนของบอดี้แพ็กเก็ต ฉะนั้น ส่วนบอดี้แพ็กเก็ต จึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็กเก็ต ไม่มีการประกาศชื่อ ที่เป็นโปรแกรมย่อย หรือ ค่าคงที่ การเขียนบอดี้แพ็กเก็ตนั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 2.16

```
PACKAGE BODY package_name IS
    declarative part
END package_name ;
```

รูปที่ 2.16 โครงสร้างของบอดี้แพ็กเก็ต

4) หน่วยการออกแบบโครงแบบ (Configuration Design Unit)

รูปแบบหนึ่งของระบบดิจิตอลไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนทิตี้ได้ เพียงหนึ่งเดียวเท่านั้น แต่ในขณะที่หน่วยการออกแบบเอนทิตี้ หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรม ที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมี หน่วยการออกแบบโครงแบบมาเพื่อกำหนดการใช้โครงแบบ (Configuration) ประกอบเอนทิตี้กับหน่วยการออกแบบสถาปัตยกรรมหน่วยไหนเข้าด้วยกัน

```
CONFIGURATION identifier OF entity_name IS
    configuration_declarative_part
END;
```

รูปที่ 2.17 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

5) โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเจอร์ใน วีเอชดีแอล เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาขั้นสูงทั่วๆ ไปค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าใช้ฟังก์ชันแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรลอจิกจริงๆ ในขณะที่ถ้าใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลาแล้ว ก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

6) โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา วีเอชดีแอล มีตัวดำเนินการหรือโอเปอร์เรเตอร์ทางลอจิกและคณิตศาสตร์เช่นเดียว กับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 2.18

PREDEFINE OPERATORS

LOGICAL OPERATORS : NOT AND OR NAND NORXOR

OPERAND TYPE : BIT BOOLEAN

RESULT TYPE : BIT BOOLEAN

RELATIONAL OPERATORS : = / = < <= > >=

OPERAND TYPE : any type

RESULT TYPE : Boolen

ARITHMETIC OPERATORS : + - * / ** MOD REM ABS

OPERAND TYPE : INTEGER REAL Physical

RESULT TYPE : INTEGER REAL Physical

CONCANTENATION OPERATOR : &

OPERAND TYPE : ARRAY of any type

RESULT TYPE : array of any type

RESULT TYPE : array of any type

รูปที่ 2.18 ตัวดำเนินการใน วีเอชดีแอล

7) เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์อุปกรณ์ทุกๆ ตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลา เข้ามาเกี่ยวข้องในทุกๆ เหตุการณ์ที่เกิดขึ้นเสมอ วีเอชดีแอล เป็นภาษาที่ได้รับการออกแบบมาเพื่อให้สามารถบรรยายรูปแบบและการพ้องกันของเวลา สำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วน ของการบรรยายสถาปัตยกรรม จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โปรเซสซึ่งมีการทำงานภายในเป็น แบบลำดับคำสั่งก็ตาม ซึ่งหากมีหลายๆ โปรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆ โปรเซสก็จะทำงานไปพร้อมๆ กันด้วย

8) สัญญาณและตัวแปร

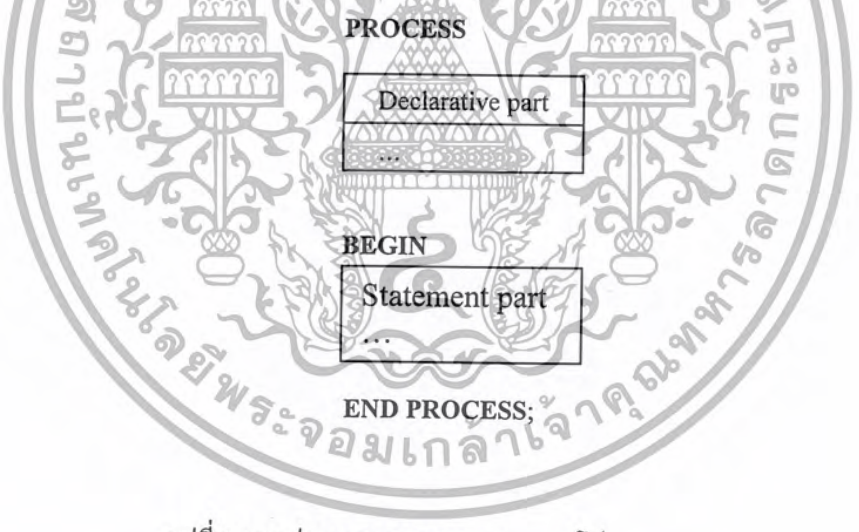
สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ ที่ใช้ในการผ่านข้อมูลและมีเรื่องของเวลามาเกี่ยวข้องด้วยการกำหนดค่าให้สัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น <= a AFTER 12 NS หมายถึงการกำหนดค่าสัญญาณ a ให้กับ w หลังจากเวลา ผ่านไป 12 นาโนวินาที ในทางตรงข้ามตัวแปรมีลักษณะเป็นเสมือนตัวกลางที่ใช้ในการส่งผ่านข้อมูลและไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย ซึ่งตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นในฟังก์ชั่น โพรซีเจอร์และโปรเซสสำหรับกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ : =

2.3.4 การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรม เป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึม สำหรับการคำนวณผลลัพธ์ที่เกิดขึ้น ซึ่งสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามา โดยไม่คำนึงถึงลักษณะโครงสร้าง หรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในว่าจะเป็นอย่างไร ในหัวข้อนี้จะแสดงถึงการบรรยายเชิงพฤติกรรมแทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่างๆ ที่ควรรู้

2.3.5 การโปรเซส

โปรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดให้กับสัญญาณ โปรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติคำสั่งพร้อมๆ กันกับโปรเซสอื่นๆ ที่อยู่ในสถาปัตยกรรมบรรยายเดียวกัน โดยโปรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์กำหนดค่าให้กับสัญญาณ (<=) การบรรยาย โปรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และ END PROCESS ในรูปที่ 2.19 เป็นการแสดงส่วนประกอบของการบรรยายแบบโปรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้และส่วนการปฏิบัติ คำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 2.19 รูปแบบของการบรรยายแบบโปรเซส

2.3.6 การกำหนดตัวดำเนินการภายในโปรเซส

ตัวดำเนินการภายในโปรเซสมี 3 ชนิดคือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวดำเนิน การทั้ง 3 ชนิดนี้หากมีการประกาศไว้ในโปรเซสใดก็จะใช้ได้เฉพาะภายในโปรเซสนั้นเท่านั้นสำหรับการติดต่อภายนอก หรือระหว่างโปรเซสสามารถทำได้โดยใช้สัญญาณ (Signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE

2.3.7 การกำหนดการกระทำภายในโปรเซส

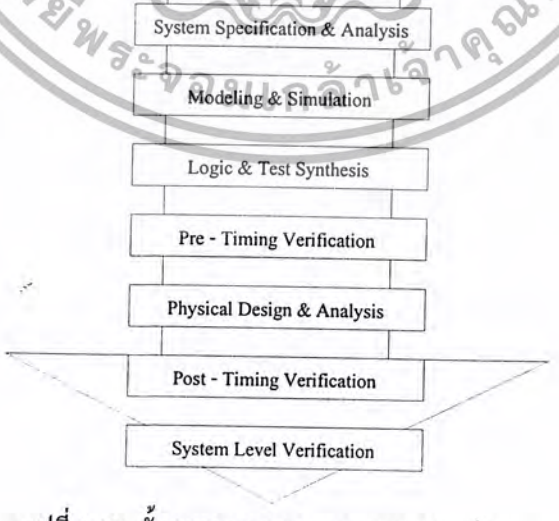
การกระทำใดๆ ภายในโปรเซสจะเป็นการปฏิบัติแบบลำดับ (Sequential) เสมอ ซึ่งภายในโปรเซสสามารถใช้ประโยคเงื่อนไขหรือการทำซ้ำได้เช่น IF-ELSE CASE-WHEN FOR LOOP และ WHILE LOOP

2.3.8 การกระตุ้นและยับยั้งการกระทำของโปรเซส

การกระทำภายในโปรเซสจะอยู่ในสภาพเตรียมพร้อม และมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์ เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโปรเซสได้ โดยการกำหนดรายการของสัญญาณที่ต้องการให้โปรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่กำหนดไว้เท่านั้น ส่วนเหตุการณ์ใดๆ ที่เกิดขึ้นกับสัญญาณ ที่ไม่ได้กำหนดไว้ในรายการก็จะไม่ส่งผลให้มีการกระทำภายในโปรเซส ซึ่งรายการของสัญญาณนี้เรียกว่า Sensitivity List และกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS

2.3.9 การออกแบบจากบนลงล่าง (Top-Down Design)

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของ บล็อกไดอะแกรมก่อนที่ทำวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอลนั้นอนุญาตให้อธิบายและวิเคราะห์การณ์ ทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการนอกจากนี้ยัง สามารถเพิ่มเติมในรายละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่างนั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวางวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ ก่อนแล้วจึงทำการจำลองการทำงาน และตรวจสอบความถูกต้อง วีเอชดีแอล กับหลักการออกแบบจากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถ ออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.20 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.20 แสดงถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้าง เล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

1) ความต้องการของระบบและการวิเคราะห์ คือ การสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2) รูปแบบและการจำลองการทำงาน คือ การเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา วีเอชดีแอล หรือ ภาษา เอชดีแอล อื่นๆ สำหรับใช้ในการบรรยายพฤติกรรมการทำงาน พร้อมทั้งทำการจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3) ลอจิกและการทดสอบการสังเคราะห์ คือ หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมรายละเอียดลงมาเป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจรจริง หรือทำการสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่จะเขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของโครงข่ายการเชื่อมต่อ ที่สามารถนำไปผลิตอุปกรณ์อื่นได้

4) การตรวจสอบเวลาก่อนการออกแบบ คือ หลังจากได้ทำการสังเคราะห์วงจรให้อยู่ในระดับเกท หรือ โครงข่ายการเชื่อมต่อแล้ว ข้อมูลนี้จะถูกนำไปใช้สำหรับการจำลองการทำงานในเรื่องความถูกต้องของฟังก์ชันพร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาใช้ในการประกอบในการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ ทางอิเล็กทรอนิกส์ทุกชิ้นจะต้องมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็น เวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถที่จะทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

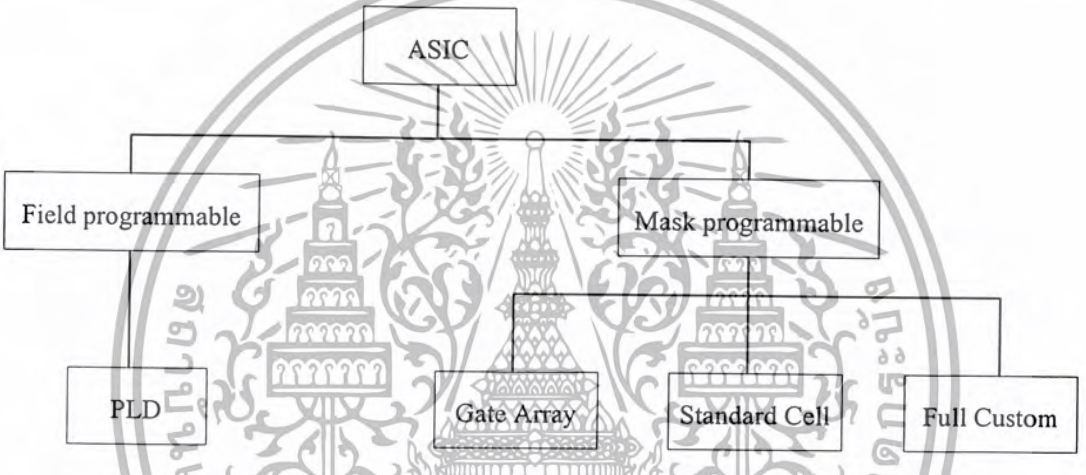
5) การออกแบบทางกายภาพและการวิเคราะห์ คือ ขั้นตอนในการผลิตเป็นวงจรจริง (Technology and device mapping) โดยจะนำข้อมูลที่ได้จากการสังเคราะห์ มาใช้ในการผลิตเป็นวงจรรวม ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวม เอซิก (ASIC)

6) การตรวจสอบเวลาหลังการออกแบบ คือ การทำการตรวจสอบการทำงานด้วยตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้ วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

7) การตรวจสอบระบบ คือ การนำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบรวมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

2.4 เอฟพีจีเอ

เทคโนโลยีความก้าวหน้าของอุตสาหกรรมอิเล็กทรอนิกส์ในปัจจุบัน ทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆ ซึ่งทำให้ลดค่าใช้จ่ายต่างๆได้มาก ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโปรเซสเซอร์ และหน่วยความจำปัจจุบัน ทุกๆครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างระหว่างวงจรรวมและไอซีมาตรฐานมากขึ้น นักออกแบบอุปกรณ์ทางด้านดิจิตอล ได้พิจารณาถึงการผลิตให้มีขนาดมากขึ้นและการผลิตวงจรรวมเอซิก (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวมจะแบ่งตามสร้างออกเป็น 2 กลุ่ม คือ ฟิลด์โปรแกรมเมเบิล (Field programmable) และ แมสโปรแกรมเมเบิล (Mask programmable) ดังแสดงในรูปที่ 2.21

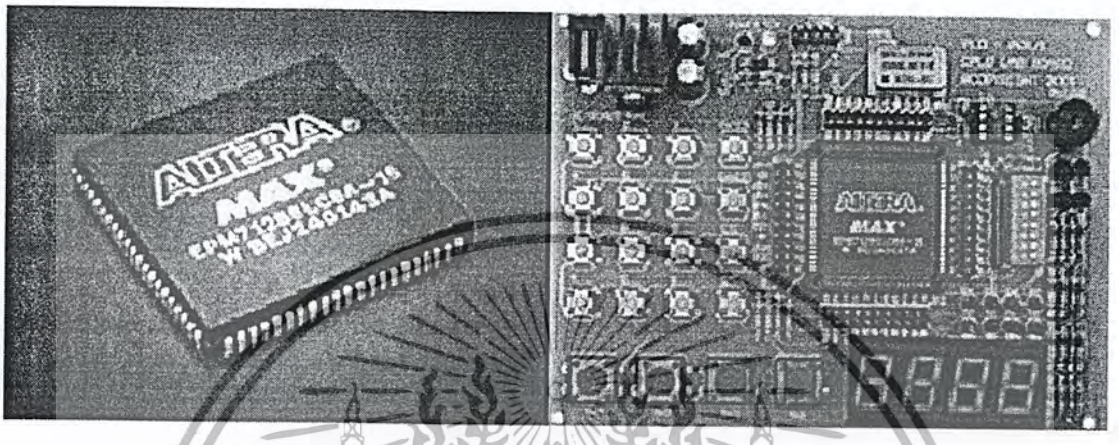


รูปที่ 2.21 ผังแสดงการแบ่งกลุ่มของวงจรรวมเอซิก

2.4.1 การออกแบบวงจรเชิงเลขด้วย ชิพอุปกรณ์เอฟพีจีเอ

ชิพอุปกรณ์เอฟพีจีเอ เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์เอฟพีจีเอ มีฟังก์ชั่นการทำงานตามที่ออกแบบไว้ ในการทำชิพอุปกรณ์ ซึ่งเป็นวิธีการออกแบบ ไอซี (IC : Integrated Circuit) แบบ เซมิคัสตัม (Semi custom) อีกวิธีหนึ่ง เมื่อเทียบกับการทำ เอซิก แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือ การทำชิพอุปกรณ์เอฟพีจีเอ จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในชิพอุปกรณ์เอฟพีจีเอ จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัด และการทำชิพอุปกรณ์เอฟพีจีเอ ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำชิพอุปกรณ์ก็คือ ระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำเอซิก มากและการตรวจสอบหรือแก้ไขการออกแบบที่ทำได้สะดวก

การทำชิพอุปกรณ์เอฟพีจีเอ ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตชิพอุปกรณ์เอฟพีจีเอ ได้เพิ่มความสามารถของชิพอุปกรณ์เอฟพีจีเอ โดยเพิ่มจำนวนองค์ประกอบภายใน หรือปรับปรุงโครงสร้างสถาปัตยกรรมภายในและยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ พีพีอาร์ (PPR: Partitioning Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย ลักษณะของตัว ชิพอุปกรณ์เอฟพีจีเอ และการนำไปใช้งานแสดงดังในรูปที่ 2.22



รูปที่ 2.22 แสดงลักษณะของตัว FPGA และการนำไปใช้งาน

สำหรับตัวชิพอุปกรณ์เอฟพีจีเอนั้นมีโครงสร้างพื้นฐาน เทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นชิพอุปกรณ์เอฟพีจีเอ ของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นชิพอุปกรณ์เอฟพีจีเอ สามารถนำไปประยุกต์ใช้งานได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP: Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์ เป็นต้น

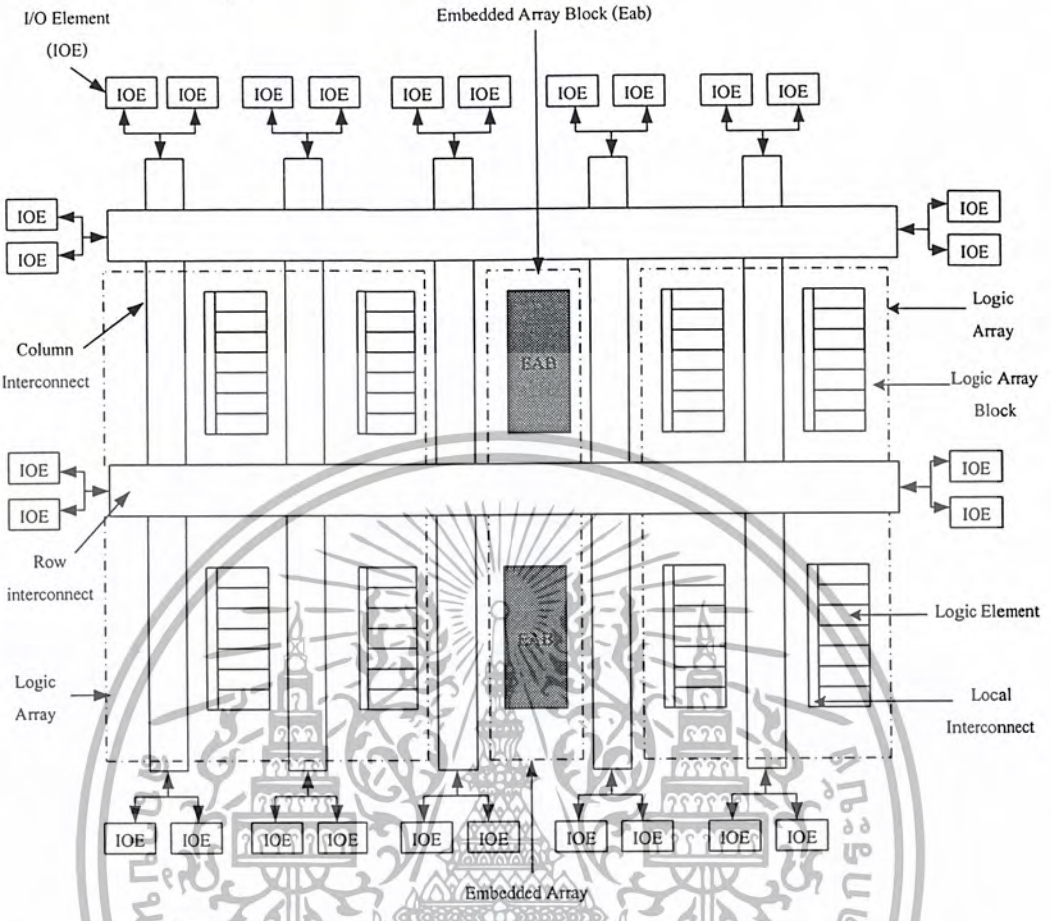
2.4.2 สถาปัตยกรรมภายในของชิพอุปกรณ์เอฟพีจีเอ

ชิพอุปกรณ์เอฟพีจีเอของบริษัท Altera ตระกูล FLEX 10 K เป็นอุปกรณ์ที่มีความหนาแน่นเกทประมาณตั้งแต่ 25,000 – 100,000 เกท โดยการจัดโครงสร้าง (Configuration) จะใช้วิธีโหลดโครงสร้างเข้าไปใน SRAM ภายใน ซึ่งหมายความว่าไม่ได้มีการจ่ายไฟเลี้ยงให้ โครงสร้างที่จัดเอาไว้ก็จะหายไป ชิพอุปกรณ์เอฟพีจีเอ ประเภทนี้จะสามารถโปรแกรมซ้ำได้ไม่จำกัดจำนวนครั้ง และการทำงานของลอจิกฟังก์ชั่นจะใช้วิธีการเปิดตารางความจริงดู (Look Up table : LUT) โดยโครงสร้างของชิพอุปกรณ์เอฟพีจีเอตระกูล FLEX 10 K แสดงดังรูปที่ 2.23 โดยในโครงสร้างของชิพอุปกรณ์เอฟพีจีเอตระกูล FLEX 10 K สามารถที่จะแบ่งเป็นส่วนต่างๆ ได้ดังนี้

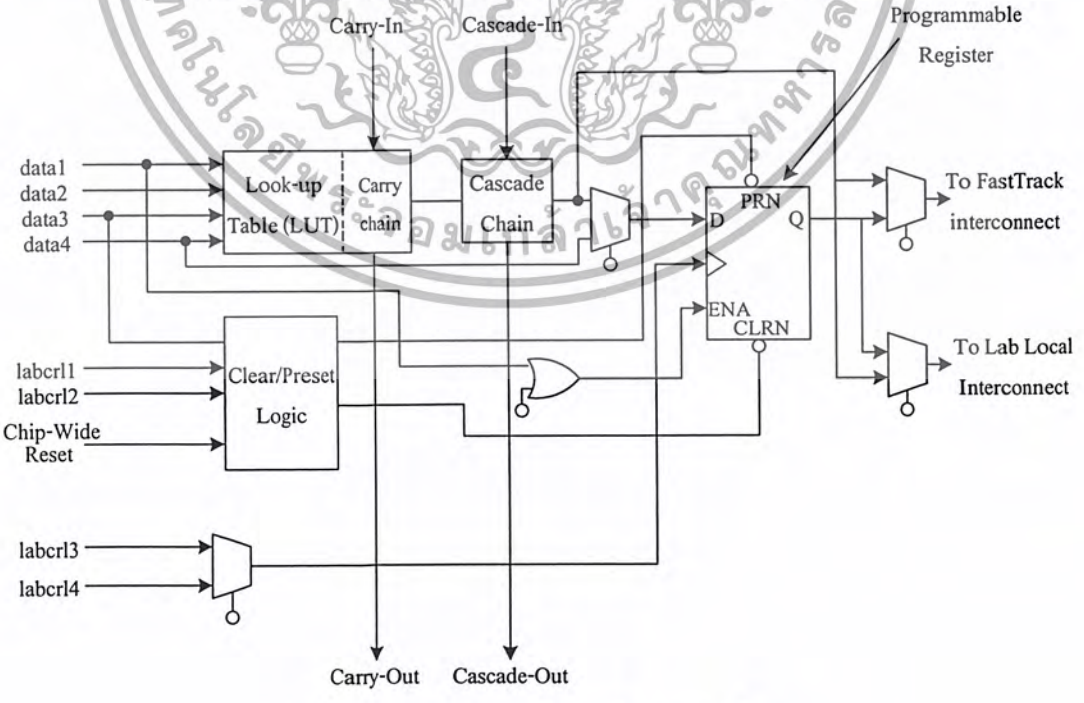
2.4.2.1 แอลอี (LE : Logic Element)

ในรูปที่ 2.24 แสดงโครงสร้างภายในของแอลอี โดยการกระทำทางบูลีนของลอจิกเกทจะสร้างด้วยวิธีการ LUT โดย LUT คือ 1x16 SRAM ซึ่ง LUT เพียงตัวเดียวสามารถนำมาทำโครงข่ายของ

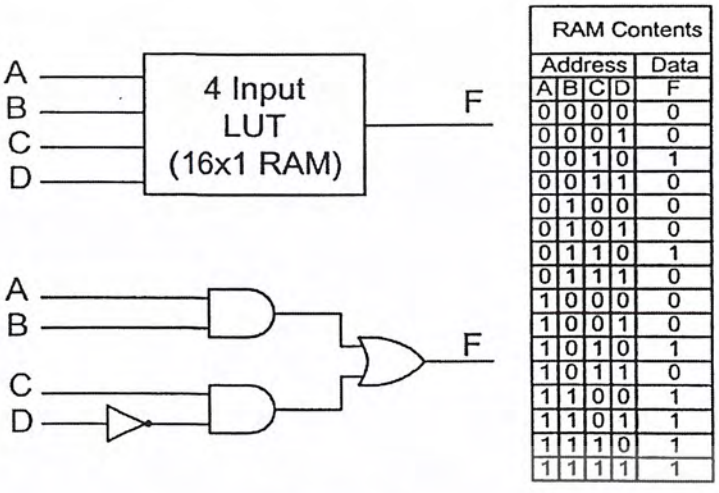
ลอจิกเกทที่มี 4 อินพุท และ 1 เอาท์พุท โดยโครงข่ายของลอจิกเกทจะถูกแปลงไปเป็นตารางค่าความจริง (Truth Table) ดังแสดงในรูปที่ 2.25



รูปที่ 2.23 แสดงโครงสร้างของชิพอุปกรณ์เอฟพีจีเอ ตระกูล FLEX 10 K



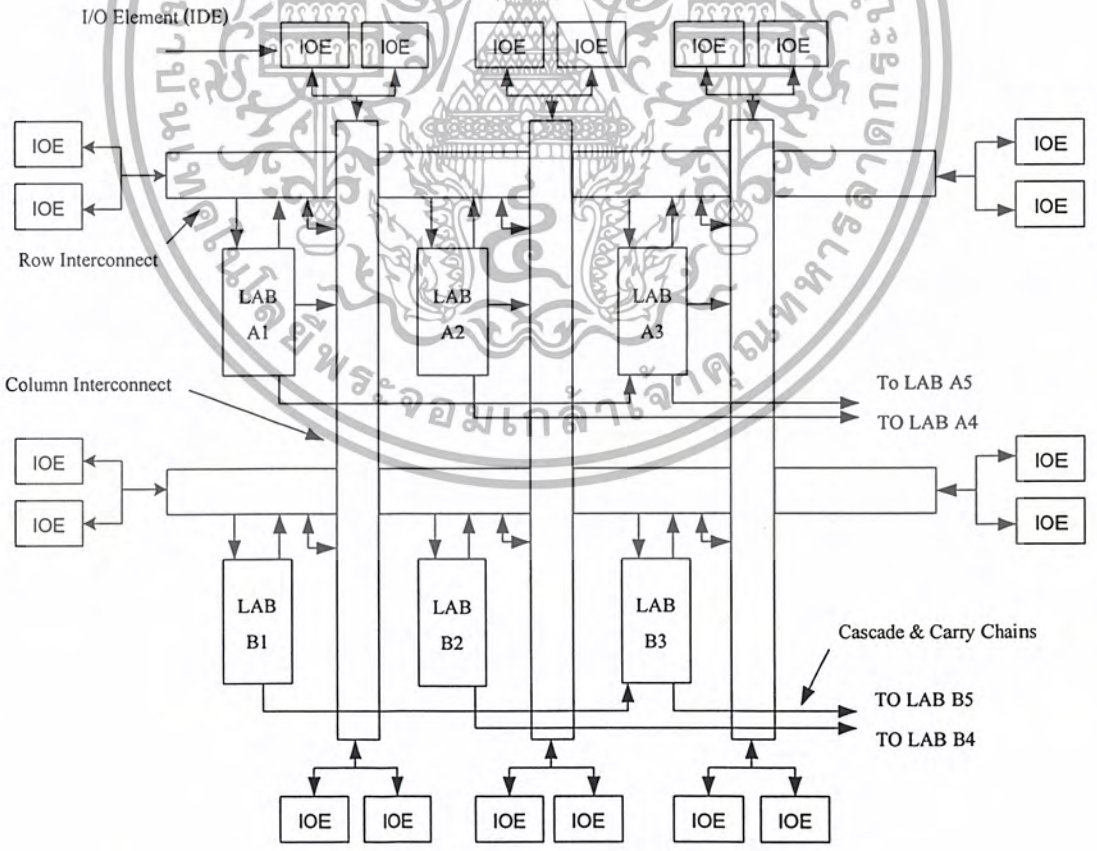
รูปที่ 2.24 แสดงโครงสร้างภายในของแอลอี



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

รูปที่ 2.25 แสดงการใช้งาน LUT เป็นโครงข่ายของลอจิก

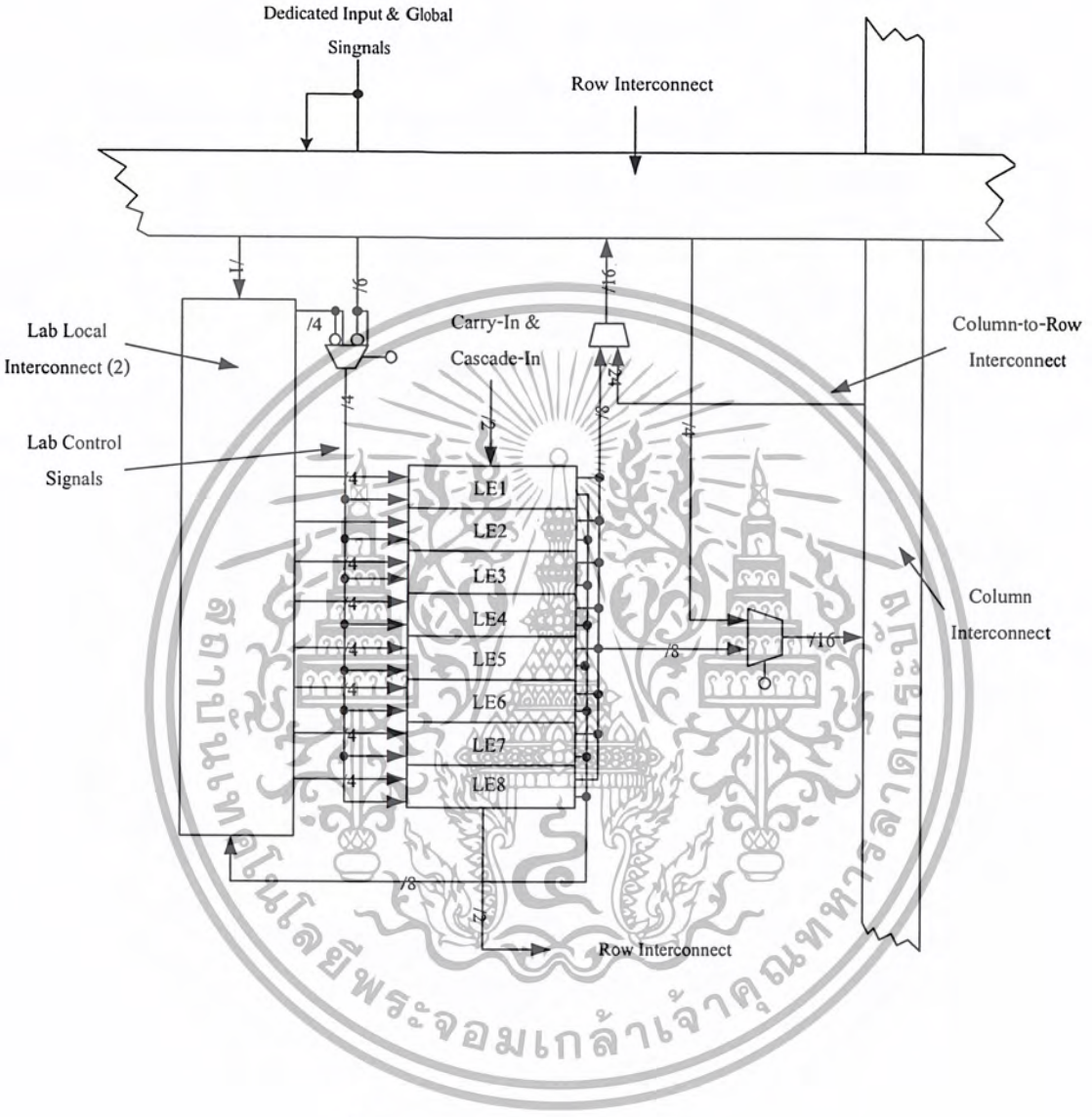
ถ้าโครงข่ายของลอจิกเกทมีความซับซ้อนขึ้นจะต้องใช้ LUT ของแต่ละ LE เป็นจำนวนหลายตัว โดยเอาท์พุทของ LUT จะส่งต่อไปยังฟลิปฟลอปและต่อไปยังโครงข่ายการเชื่อมต่อ (Interconnection Network) ดังแสดงในรูปที่ 2.26



รูปที่ 2.26 แสดงโครงข่ายของการเชื่อมต่อ

2.4.2.2 แอลเอบี (LAB: Logic Array Block)

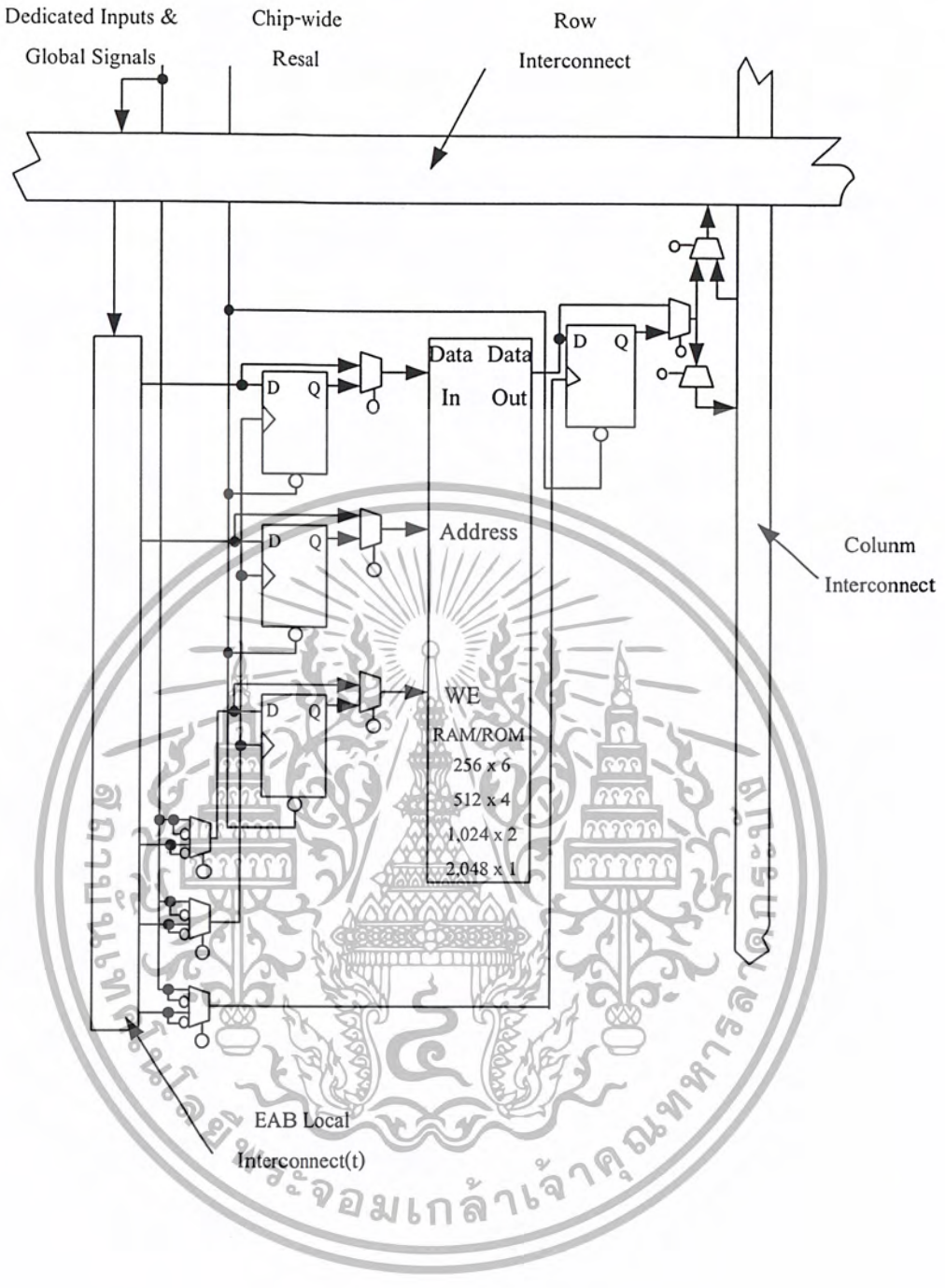
แอลเอบี 1 ตัว จะประกอบไปด้วย 8 แอลอี ดังแสดงในรูปที่ 2.27



รูปที่ 2.27 แสดงโครงสร้างภายในของ แอลเอบี

2.4.2.3 อีเอบี (EAB: Embedded Array Block)

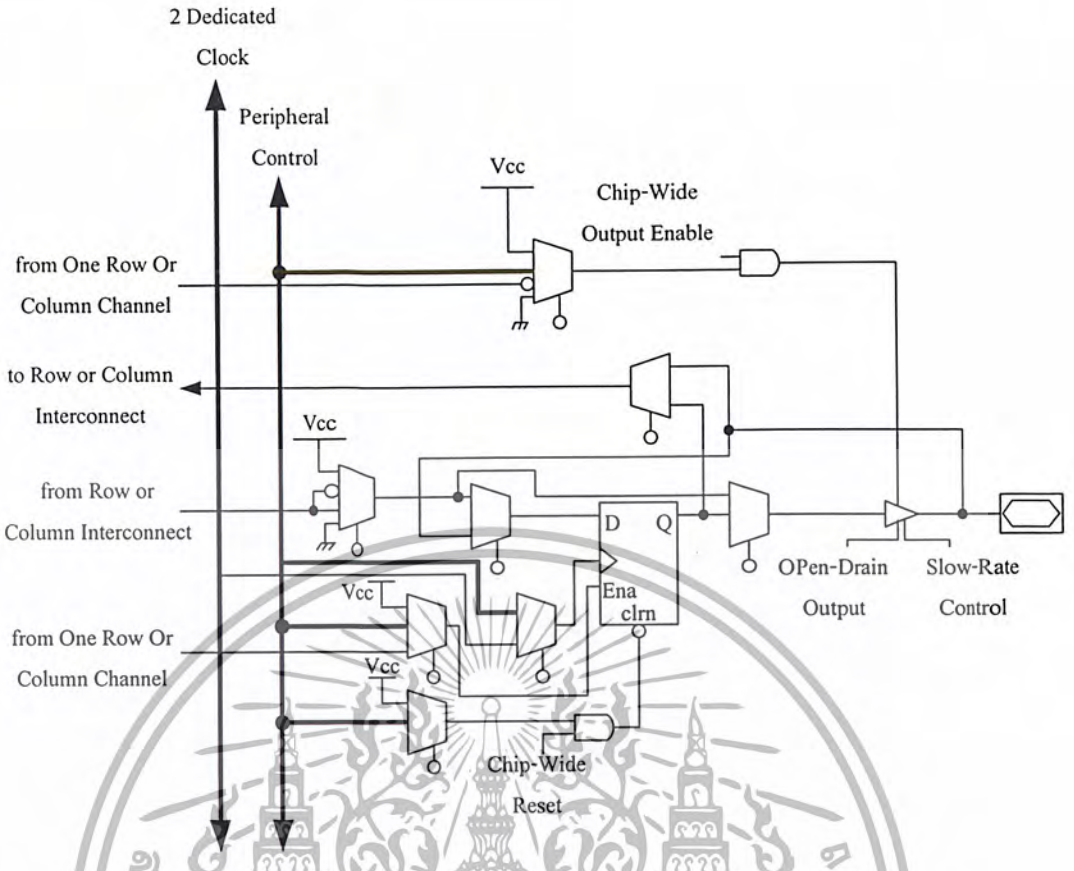
สถาปัตยกรรมโดยทั่วไปของ FLEX 10 K จะมีลักษณะของแอลเอบี ที่มีการจัดเรียงแบบเมตริกซ์ และ อีเอบี ซึ่งมีการเชื่อมต่อผ่านทางแถวและคอลัมน์ โดยในแต่ละแถวจะมี 1 อีเอบี จะมีขนาด 2048 บิต และสามารถกำหนดความกว้าง (Width) ความลึก (Depth) ของ อีเอบี ได้โดยไม่ส่งผลต่อความเร็ว



รูปที่ 2.28 แสดงโครงสร้างภายในอีเอบี

2.4.2.4 ไอโออี (IOE: Input Output Element)

ไอโออี จะถูกต่ออยู่กับขา I/O โดยจะประกอบด้วยส่วนของวงจรที่เป็น Tri State และส่วนที่เป็นฟลิปฟลอป ซึ่งเป็น option ดังแสดงในรูปที่ 2.29



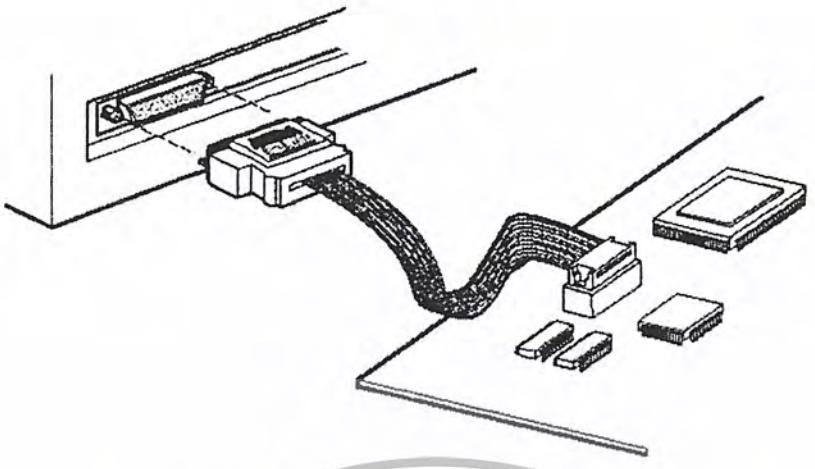
รูปที่ 2.29 แสดงโครงสร้างภายในของไอโออี

2.5 ปัจจัยที่ทำให้การออกแบบชิพอุปกรณ์เอฟพีจีเอ ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพเพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึงภาษาแอสแซมบี้ (Assembly) ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2. มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจรหรือเอชดีแอล เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มันจากนั้นตัวซอฟต์แวร์จะทำการสังเคราะห์ ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตรฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของคอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 2.30 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสียค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 2.30 การโปรแกรมลงในชิพอุปกรณ์เอฟพีจีเอ

2.6 หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจาย

โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) หรือเรียกอย่างย่อๆว่า "DA" เป็นการจัดรูปแบบทางคณิตศาสตร์ของสมการที่อยู่ในรูปของผลบวกของผลคูณ (Sum of Product) ของเลขฐานสิบให้กระจายออกเป็นบิทหรือในรูปแบบของเลขฐานสอง เพื่อให้การคำนวณทางคณิตศาสตร์สามารถแปลงเป็นวงจรดิจิตอลอิเล็กทรอนิกส์ได้ โดยจะปรากฏอยู่ในงานทางด้านการประมวลผลสัญญาณเชิงเลข หลักการเบื้องต้นของโครงสร้างเลขคณิตกระจายที่นำมาใช้งานด้านการประมวลผลสัญญาณเชิงเลข คือการแปลงฟังก์ชันถ่ายโอน (Transfer Function) ซึ่งเป็นสมการที่อยู่ในรูปผลบวกของผลคูณระหว่างสัญญาณอินพุทกับค่าสัมประสิทธิ์ของฟังก์ชั่นถ่ายโอนของระบบ โดยในการคูณกันระหว่างค่าสัมประสิทธิ์ของฟังก์ชั่นถ่ายโอนกับสัญญาณอินพุท จะใช้การคูณเลขฐานสองแบบส่วนเติมเต็มสอง (2's complement) และการคูณจะใช้แบบเปิดตาราง (Look-up table) โดยค่าผลบวกของผลคูณระหว่างสัมประสิทธิ์และสัญญาณอินพุทจะถูกเก็บไว้ในหน่วยความจำ EPROM และจะใช้สัญญาณอินพุทเป็นแอดเดรสของ EPROM โดยตรง ทั้งค่าสัมประสิทธิ์ของวงจรกรองและสัญญาณอินพุทจะถูกแทนด้วยเลขส่วนเติมเต็มสอง ดังนั้นโครงสร้างเลขคณิตกระจายจึงมีทฤษฎีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง (2's complement Multiplication)

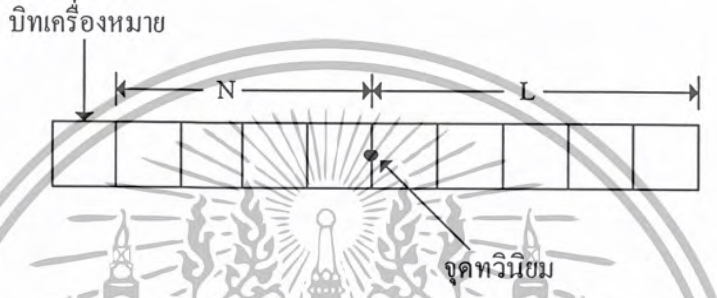
2.6.1 ระบบตัวเลข

สำหรับระบบเชิงเลข ตัวเลขต่างๆจะถูกแทนด้วยเลขฐานสอง ซึ่งโดยทั่วไปมีรูปแบบที่นิยมใช้กันอยู่ 2 รูปแบบ คือ รูปแบบจำนวนโดยตรง (Fixed point format) และ รูปแบบจำนวนอิงดรรชนี (Floating point format) ซึ่งรูปแบบจำนวนโดยตรงจะมีวงจรฮาร์ดแวร์ที่ใช้ในการคำนวณที่ง่ายกว่า แต่ให้ค่าจากการคูณค่อนข้างจำกัด ส่วนรูปแบบจำนวนอิงดรรชนีจะสามารถแทนค่าของสัญญาณ คือให้ย่านพลวัตร

(Dynamic range) ได้มากกว่า แต่ต้องใช้วงจรฮาร์ดแวร์ที่สลับซับซ้อน แพงกว่า และให้ความเร็วในการประมวลผลที่ลดลง

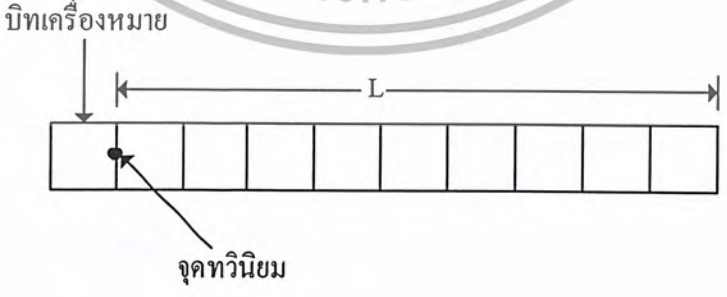
1. รูปแบบจำนวนโดยตรง

รูปแบบจำนวนโดยตรงปกติจะประกอบไปด้วย 3 ส่วน คือ บิทเครื่องหมาย (Sign bit) 1 บิท, บิทจำนวนเต็ม (Integer bit) N บิท และบิทเศษส่วน (Fractional bit) L บิท โดยจะมีจุดทวินิยม (Binary point) อยู่ระหว่างบิทจำนวนเต็มและบิทเศษส่วนดังแสดงในรูปที่ 2.31



รูปที่ 2.31 แสดงการจัดรูปแบบจำนวนโดยตรงที่ประกอบด้วยบิทจำนวนเต็มและบิทเศษส่วน

จำนวนบิท N เป็นตัวกำหนดย่านพลวัตรที่ต้องการ โดยถ้าเลือกให้มีจำนวนน้อยอาจทำให้เกิดการล้น (Overflow) จากการคำนวณได้ แต่ถ้าเลือกให้มีจำนวนมากความเที่ยงตรงก็จะน้อยลง ซึ่งในการสร้างวงจรกรองสัญญาณเชิงเลขโดยการแทนด้วยรูปแบบจำนวนโดยตรงนั้น นิยมที่จะทำมาตราส่วน (Scaling) เพื่อให้ขนาดของสัญญาณมีค่าอยู่ระหว่าง $-1 \le x < 1$ คือมีบิทเครื่องหมาย 1 บิท และบิทเศษส่วน L บิท ดังแสดงในรูปที่ 2.32



รูปที่ 2.32 แสดงการจัดรูปแบบจำนวนโดยตรงที่มีแต่บิทเศษส่วน

โดยทั่วไปเลขฐานสองแบบจำนวนโดยตรงแบ่งออกได้เป็น 3 รูปแบบด้วยกัน คือ (1) แบบขนาดและเครื่องหมาย (Sign magnitude), (2) แบบส่วนเติมเต็มหนึ่ง (1's complement) และ (3) แบบส่วนเติมเต็มสอง (2's complement) โดยคุณลักษณะที่สำคัญบางประการของการแทนตัวเลขด้วยเลขฐานสองแบบจำนวนโดยตรงทั้ง 3 รูปแบบสามารถสรุปได้ดังตารางที่ 2.5

Features	Sign and magnitude	2' complement	1'complement
Range	$-(1-2^{-L}) \le x \le (1-2^{-L})$	$-1 \le x \le (1-2^{-L})$	$-(1-2^{-L}) \le x \le (1-2^{-L})$
Representation of zero	0.000 and 1.000	0.000	0.000 and 1.111
Arithmetic rules	Simple must be kept track of, separately	Simple; negative numbers elegantly handled	Simple, but "end around carry" should be carefully handled
Suitability for serial arithmetic	**Not so good**	**Excellent**	**Good**

ตารางที่ 2.5 แสดงคุณสมบัติที่สำคัญของรูปแบบจำนวนโดยตรง

ใน 3 รูปแบบนี้ตัวเลขแบบส่วนเติมเต็มสองเป็นที่นิยมใช้กันมากในระบบการประมวลผลสัญญาณเชิงเลข ทั้งนี้เนื่องมาจาก

1. มีการแทนค่าเลขศูนย์ได้เพียงค่าเดียว
2. การสร้างวงจรฮาร์ดแวร์สำหรับการบวก ลบ และคูณ ของเลขส่วนเติมเต็มสองทำได้ง่ายโดยในการคูณสามารถใช้หลักการเลื่อนและบวก (Shift and add)
3. ในระหว่างผลการบวกย่อย (Partial sum) ของการบวกเลขส่วนเติมเต็มสอง สามหรือสี่จำนวนถึงแม้อาจจะเกิดการล้น (ตัวทดจากผลการบวกล้นข้ามไปทับบิทเครื่องหมาย) แต่ผลลัพธ์สุดท้ายมักให้ค่าถูกต้องเสมอ ถ้าผลบวกอยู่ในช่วง -1 ถึง $1-2^{-L}$ ดังตัวอย่าง

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

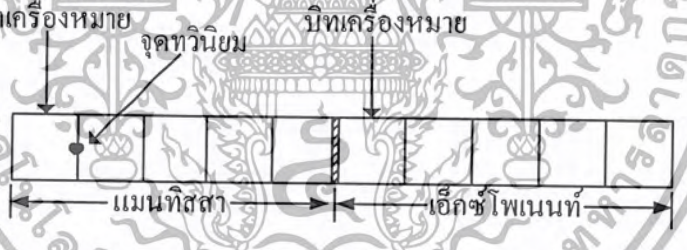
2 รูปแบบจำนวนอิงดรรชนี

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

$$X = M \times 2^{e} \tag{2.46}$$

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

M และ e เรียกว่า แมนทิสสา (Mantissa) และ เอ็กซ์โพเนนท์ (Exponent) ตามลำดับ ตัวอย่างเช่น จำนวน 0.00110101 และ 01001.11 สามารถแทนได้โดย 0.110101×2^{-2} และ 0.100111×2^{4} ตามลำดับ ส่วนจำนวนที่มีค่าเป็นลบก็ทำในลักษณะเดียวกัน รูปแบบจำนวนอิงดรรชนีสามารถแสดงได้ดังรูปที่ 2.33 โดยแบ่งเป็น 2 ส่วน คือส่วนหนึ่งสำหรับแมนทิสสา และอีกส่วนสำหรับเอ็กซ์โพเนนท์



รูปที่ 2.33 แสดงการจัดรูปแบบจำนวนอิงดรรชนี

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

2.6.2 ทฤษฎีเลขคณิตกระจาย

จากที่ได้กล่าวมาแล้วว่า โครงสร้างเลขคณิตกระจายมีพื้นฐานอยู่บนการคูณแบบเลขส่วนเติมเต็มสอง ดังนั้นในหัวข้อนี้จะได้อธิบายถึงหลักการของการคูณเลขส่วนเติมเต็มสอง

ให้เลขส่วนเติมเต็มสองของ X ซึ่งแทนด้วย $\overline{X}$ และนิยามโดย

$$\bar{x} = \begin{cases} x & \text{ถ้า } x \geq 0 \\ 2-|x| & \text{ถ้า } x < 0 \end{cases} \tag{2.47}$$

โดย X เป็นเลขที่เป็นเศษส่วน (Fractional number)

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

$$\overline{X} = X_0.X_1.X_2...X_L \tag{2.48}$$

ค่าของ $\overline{X}$ ในรูปของเลขฐานสิบสามารถหาได้ดังนี้

$$X = -X_0 + \sum_{i=1}^{L} X_i 2^{-i} \tag{2.49}$$

จากนั้นพิจารณาผลคูณต่อไปนี้

$$Y = Xm \tag{2.50}$$

ให้ $\overline{Y}$, $\overline{X}$ และ $\overline{m}$ เป็นเลขส่วนเติมเต็มสองของ Y, X และ m ตามลำดับ จากนั้นพิจารณาจากสมการที่ (2.49) และ สมการที่ (2.50) จะได้

$$\begin{aligned} Y &= -Y_0 + \sum_{i=1}^{L} Y_i 2^{-i} \\ &= -X_0 m + \sum_{i=1}^{L} X_i m 2^{-i} \end{aligned} \tag{2.51}$$

ดังนั้น

$$\begin{aligned} \overline{Y} &= \text{ส่วนเติมเต็มสองของ} \left(-X_0 m + 2^{-1}X_1 m + 2^{-2}X_2 m + 2^{-3}X_3 m + ... + 2^{-L}X_L m\right) \\ &= \text{ส่วนเติมเต็มสองของ} \left(-X_0 m + 2^{-1}(X_1 m + ... + 2^{-1}(X_{L-1} m + 2^{-1}(X_L m)))\right) \end{aligned} \tag{2.52}$$

ต่อไปพิจารณาส่วนเติมเต็มสองของ $2^{-1}U$ โดย

$\overline{U} = U_0.U_1U_2...U_M$; สำหรับ $U \geq 0$ (หรือ $U_0 = 0$)

ส่วนเติมเต็มสองของ $(2^{-1}U) = 2^{-1}\overline{U}$; และสำหรับ $U < 0$ (หรือ $U_0 = 1$)

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

ดังนั้นสรุปได้ว่า

$$\text{ส่วนเติมเต็มสองของ}(2^{-1}\,U)=\begin{cases} 2^{-1}\overline{U} & ;\, U_0=0 \\ 1+2^{-1}\overline{U} & ;\, U_0=1 \end{cases} \tag{2.53}$$

สมการที่ (2.53) นี้แสดงให้เห็นได้ว่า ส่วนเติมเต็มสองของ $(2^{-1}U)$ เป็นการเลื่อนข้อมูลของ $\overline{U}$ ไปทางขวา 1 บิท

$$\therefore \text{ส่วนเติมเต็มสองของ}\,(2^{-1}U)=2_2^{-1}\overline{U} \tag{2.54}$$

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

จากนั้นพิจารณาสมการที่ (2.52) และสมการที่ (2.53) จะได้ว่า

$$\begin{aligned} \overline{Y} &= -X_0\overline{m}+2^{-1}X_1\overline{m}+2^{-2}X_2\overline{m}+2^{-3}X_3\overline{m}+\ldots+2^{-L}X_L\overline{m} \\ &= -X_0\overline{m}+2^{-1}(X_1\overline{m}+\ldots+2^{-1}(X_{L-1}\overline{m}+2^{-1}(X_L\overline{m}))) \end{aligned} \tag{2.55}$$

ซึ่งจากสมการที่ (2.55) จะเห็นได้ว่าผลคูณจากสมการที่ (2.50) สามารถหาได้โดยการใช้หลักการเลื่อนและบวก (Shift and add) โดยผลลัพธ์ที่ได้จากการคูณแบบเลขส่วนเติมเต็มสอง สามารถหาได้ตามขั้นตอนดังนี้

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

ตัวอย่างการทำงานตามอัลกอริธึมนี้

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

$m = -0.390625$

$\overline{m} = 2 - |m|$ m เป็นเลขลบ

$= 2 - 0.390625$

$= 1.609375$

$\therefore \overline{m} = 1.100111$

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

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

โดยมีขั้นตอนการทำงาน ดังตารางต่อไปนี้

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

ตารางที่ 2.6 แสดงขั้นตอนการคูณเลขส่วนเติมเต็มสอง

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

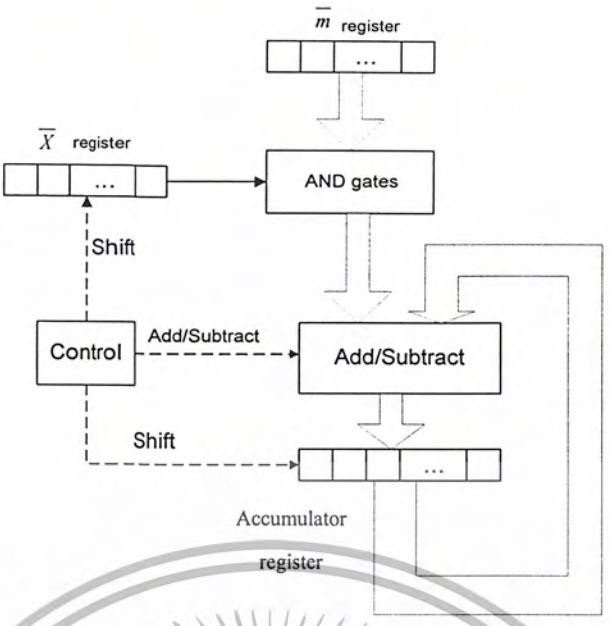
จะได้

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

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

$$= -0.3173828125$$

จากอัลกอริธึมดังกล่าวสามารถออกแบบการทำงานและสร้างวงจรแสดงได้ดังรูปที่ 2.41



รูปที่ 2.34 แสดงการคูณแบบเลขส่วนเติมเต็มสองโดยใช้เลขคณิตกระจาย

ที่ผ่านมาเป็นหลักการคูณแบบเลขส่วนเติมเต็มสอง ส่วนทฤษฎีเลขคณิตกระจายก็อาศัยหลักการดังกล่าวมาใช้ โดยทำการกระจายสมการที่อยู่ในรูปผลบวกของผลคูณให้แตกออกมาอยู่ในระดับบิท (Bit level) พิจารณาผลบวกของผลคูณต่อไปนี้

$$Y = \sum_{i=0}^{N} m_i X_i \tag{2.56}$$

โดย m_i เป็นค่าสัมประสิทธิ์ซึ่งที่ค่าคงที่

X_i เป็นข้อมูลอินพุท

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

$$X_i = -X_{i0} + \sum_{j=1}^{L} X_{ij} 2^{-j} \tag{2.57}$$

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

X_{i0} = บิทแสดงเครื่องหมาย

X_{iL} = บิทที่มีนัยสำคัญต่ำสุด (LSB)

$L+1$ = จำนวนบิทที่แทนข้อมูลอินพุท

แทนค่า X_i ในสมการที่ (2.57) ลงในสมการที่ (2.56) จะได้

$$Y = \sum_{i=0}^{N} m_i \left[-X_{i0} + \sum_{j=1}^{L} X_{ij} 2^{-j} \right] \tag{2.58}$$

เมื่อจัดเทอมของผลบวกใหม่จะได้

$$\begin{aligned} Y &= -X_{i0} \sum_{i=0}^{N} m_i + \sum_{j=1}^{L} X_{ij} 2^{-j} \sum_{i=0}^{N} m_i \\ &= -\sum_{i=0}^{N} X_{i0} m_i + \sum_{i=0}^{N} \sum_{j=1}^{L} X_{ij} m_i 2^{-j} \end{aligned} \tag{2.59}$$

จากนั้นทำการกระจายออกให้เป็นระดับบิท ได้ดังนี้

$$\begin{aligned} Y = &-(X_{00}m_0 + X_{10}m_1 + X_{20}m_2 + ... + X_{N0}m_N) \\ &+ 2^{-1}(X_{01}m_0 + X_{11}m_1 + X_{21}m_2 + ... + X_{N1}m_N) \\ &+ 2^{-2}(X_{02}m_0 + X_{12}m_1 + X_{22}m_2 + ... + X_{N2}m_N) \\ &+ ... + 2^{-L}(X_{0L}m_0 + X_{1L}m_1 + X_{2L}m_2 + ... + X_{NL}m_N) \end{aligned} \tag{2.60}$$

สมการที่ (2.60) นี้ถูกกระจายออกให้อยู่ในรูปผลบวกของผลคูณระหว่างสัมประสิทธิ์กับข้อมูลอินพุทในระดับบิท ซึ่งเป็นนิยามของการคำนวณแบบเลขคณิตกระจาย และเมื่อเปรียบเทียบสมการที่ (2.60) กับสมการที่ (2.55) จะเห็นว่าการคำนวณหาค่า Y ก็ใช้เลขคณิตกระจายนั่นเอง เพียงแต่นำค่าผลคูณย่อย (Partial product) ที่คำนวณไว้ล่วงหน้าแล้วสำหรับแต่ละค่าที่สอดคล้องกับแต่ละบิทของข้อมูลอินพุทไปเก็บไว้ในตารางเปิดดู ซึ่งเป็นหน่วยความจำ EPROM และใช้ข้อมูลอินพุทเป็นแอดเดรสของหน่วยความจำ เพื่อนำค่าในตารางเปิดดูมาผ่านขั้นตอนการคำนวณตามบูทอัลกอริธึม ซึ่งค่าในตารางเปิดดูสามารถแสดงได้ดังนี้

Bit pattern ของข้อมูลอินพุท				ผลคูณย่อยที่เก็บไว้ในตารางเปิดดู
X_{Nj}	.....	X_{2j}	X_{1j}	X_{0j}	
0	.....	0	0	0	0
0	.....	0	0	1	m_0
0	.....	0	1	0	m_1
0	.....	0	1	1	$m_1 + m_0$
0	.....	1	0	0	m_2

ตารางที่ 2.7 แสดงค่าผลคูณย่อยที่เก็บไว้ในตารางเปิดดูที่กำหนดโดยข้อมูลอินพุท

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

ตารางที่ 2.7 (ต่อ) แสดงค่าผลคูณย่อยที่เก็บไว้ในตารางเปิดดูที่กำหนดโดยข้อมูลอินพุท

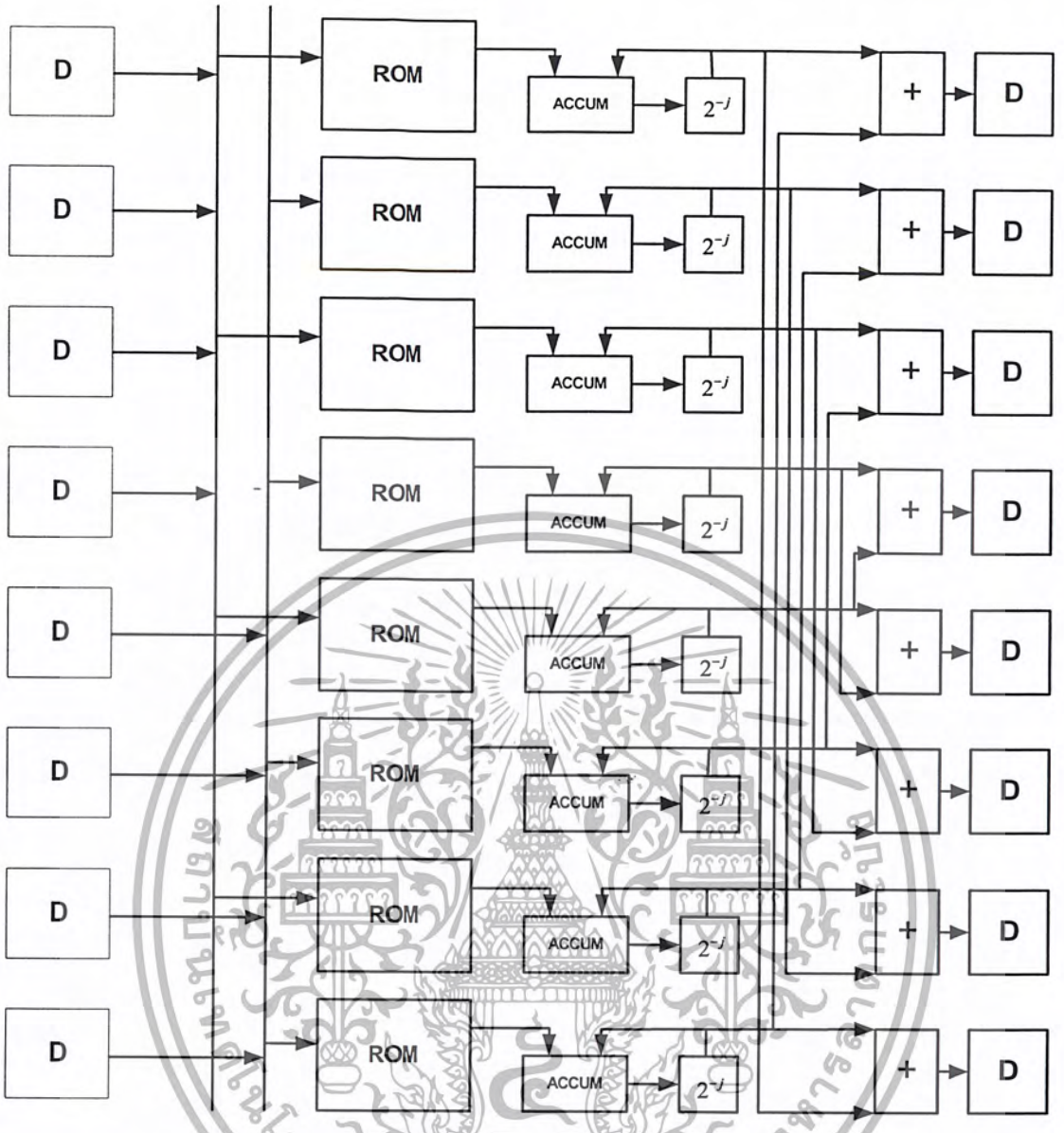
2.7 การประยุกต์ใช้งาน DA (Distributed Arithmetic) กับ CORDIC Algorithm

เนื่อง จากที่การแปลงแบบ DCT เป็นเทคนิคที่นิยมใช้กันมากในปัจจุบันทั้งยังถูกนำมาใช้เป็นมาตรฐานในการบีบอัดสัญญาณภาพและวีดีโอแบบดิจิตอลอย่างหลากหลาย ในโครงสร้างหลายๆแบบของ DCT โครงสร้างอีกแบบหนึ่งที่นิยมใช้กันก็คือการใช้งานบนโครงสร้างของ DA ซึ่งคุณสมบัติของ DA นี้จะทำงานอยู่ในรูปแบบของผลบวกของผลคูณ (Sum of Products) โดยไม่ใช้การคูณโดยตรงในการประมวลผลของข้อมูล ซึ่งแนวความคิดนี้เหมาะสมกับรูปแบบการคำนวณของ DCT เช่นดังตัวอย่างของการนำโครงสร้าง DA ไปใช้กับ IDCT แบบหนึ่งมิติดังรูปที่ 2.35

จากรูปที่แสดงในข้างต้นจะมี 3 องค์ประกอบที่จะเป็นตัวจำกัดความเร็วในการประมวลผลของ IDCT ในกรณีที่ใช้โครงสร้างของ DA คือเวลาของการเข้าถึงค่าใน ROM ในกรณีที่ ROM มีขนาดใหญ่, ความเร็วของ Accumulator และความเร็วของตัว Adder นอกจากนี้เนื่องจาก DA จำเป็นที่จะต้องคำนวณค่าในรูปแบบของบิตอนุกรมและต้องทำการตรวจสอบค่าของบิตเครื่องหมาย (Sign Bit) ด้วย ดังนั้น Accumulator จะได้รับค่าอินพุทไปคำนวณหลังจากที่ชุดของ DA ทำงานเสร็จซึ่งจะเป็นเวลา w ของรอบสัญญาณนาฬิกา เมื่อ w คือความกว้างของจำนวนบิตของค่าอินพุตในแต่ละ sample สำหรับส่งเข้าไปในอุปกรณ์แต่ละชุด

ในโครงงานนี้ได้นำเอาการทำงานของ DA มาประยุกต์ใช้กับการคำนวณโดยทฤษฎีของ CORDIC Algorithm ซึ่งจะถูกนำไปพิจารณาร่วมกับสมการพื้นฐานของ CORDIC และกำหนดค่าของ ROM ที่ใช้เป็นขนาด 4 word ซึ่งค่าที่กำหนดนี้จะไม่มีความเกี่ยวข้องกับขนาดความกว้างของค่าอินพุทที่ป้อนเข้ามา

ถึงแม้ว่าการนำไปสร้างโดยใช้ CORDIC Algorithm โดยตรงในการคำนวณค่าของ DCT จะดีกว่าเมื่อสร้างแบบใช้ตัวคูณแต่การนำไปสร้างจริงในรูปแบบโครงสร้างของการคำนวณแล้วก็ยังไม่เหนือไปกว่าการคำนวณโดยใช้ DA ดังนั้นหลักสำคัญก็คือการนำรูปแบบโครงสร้างของ DA มาใช้กับการคำนวณแบบหมุนของ CORDIC Algorithm ซึ่งจะทำให้ได้รูปแบบการคำนวณผลของ CORDIC Algorithm ในโครงสร้างแบบใหม่ขึ้นมาซึ่งสามารถที่จะพิจารณาได้ดังนี้



รูปที่ 2.35 IDCT แบบหนึ่งมิติที่นำ DA ไปใช้

จากสมการพื้นฐานของ CORDIC Algorithm คือ

$$x = x_i \cos\theta_i - y_i \sin\theta_i$$
$$y = y_i \cos\theta_i + x_i \sin\theta_i \tag{2.61}$$

โดยที่ x และ y เป็นตำแหน่งที่แทนค่าของแต่ละพิกเซลเมื่อเมื่อแต่ละพิกเซลถูกแทนด้วยจำนวน B บิต ค่าเวคเตอร์ของแต่ละตำแหน่งจะสามารถแสดงเป็นสมการได้ดังนี้

$$x = -x(0) + \sum_{j=1}^{B-1} x(j)2^{-j}$$
$$y = -y(0) + \sum_{j=1}^{B-1} y(j)2^{-j} \tag{2.62}$$

เมื่อ $x(j)$ และ $y(j)$ แทนค่าที่ตำแหน่งบิตต่างๆของ x และ y ทำการนำสมการที่ (2.62) ไปแทนในสมการที่ (2.61) และทำการจัดรูปใหม่ดังนี้

$$x = \left(-x(0) + \sum_{j=1}^{B-1} x(j)2^{-j} \right) \cos\theta - \left(-y(0) + \sum_{j=1}^{B-1} y(j)2^{-j} \right) \sin\theta$$

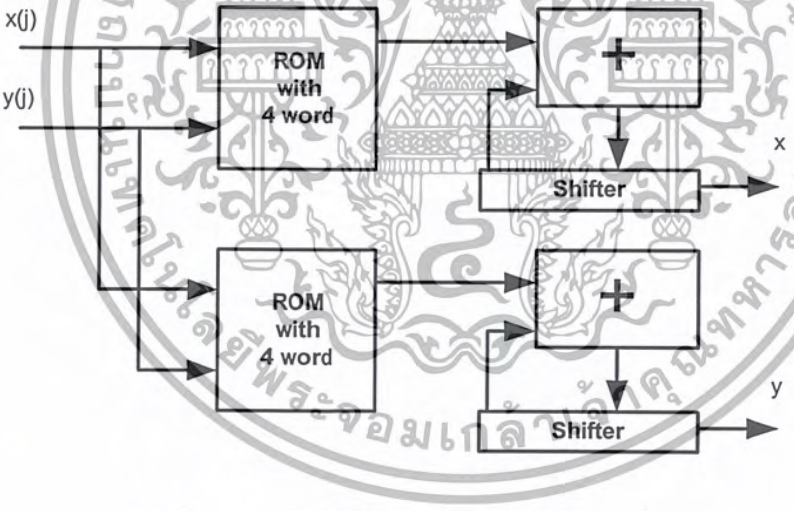
$$y = \left(-y(0) + \sum_{j=1}^{B-1} y(j)2^{-j} \right) \cos\theta + \left(-x(0) + \sum_{j=1}^{B-1} x(j)2^{j} \right) \sin\theta$$

จะได้เป็น

$$x = -(x(0)\cos\theta - y(0)\sin\theta) + \sum_{j=1}^{B-1} (x(j)\cos\theta - y(j)\sin\theta).2^{-j}$$

$$y = -(y(0)\cos\theta + x(0)\sin\theta) + \sum_{j=1}^{B-1} (y(j)\cos\theta + x(j)\sin\theta).2^{-j} \quad (2.63)$$

โดยที่ค่าของ $x(j)$ และ $y(j)$ จะเป็นค่าของ 0 หรือ 1 จากสมการที่ (2.63) จะทำให้มีผลของการหาค่าได้ 4 รูปแบบ ซึ่งค่าเหล่านี้เราสามารถที่จะทำการคำนวณและเก็บไว้ใน Lookup Table (หรือ ROM) และสมการที่ (2.63) ซึ่งจะแสดงให้เห็นว่าสามารถสร้างได้จากการคำนวณแบบผลบวกของผลคูณนั่นก็คือการนำเอา DA มาใช้ได้นั่นเองซึ่งโครงสร้างจะเป็นดังรูปที่ 2.36



รูปที่ 2.36 โครงสร้างที่นำ DA ไปใช้กับ CORDIC Algorithm

บทที่ 3

การคำนวณและการสร้าง

3.1 การออกแบบส่วนการแปลง Discrete Cosine Transform (DCT)

3.1.1 การคำนวณหาสมการและสัมประสิทธิ์ที่ใช้ในการสร้าง

การแปลงแบบ DCT ในที่นี้เป็นแบบ 1 มิติ ที่มีอินพุทและเอาท์พุทอย่างละ 8 ตัว ซึ่งสมการพื้นฐานที่ใช้ในการแปลงแบบ DCT นั้นนำมาจากบทที่ 2 สามารถแสดงได้ดังนี้

$$y(n)=e(n)\sum_{k=0}^{7}x(k)\cos\left[\frac{(2k+1)n\pi}{2\times 7}\right] \qquad ;n=0,1,...,7 \tag{3.1}$$

$$e(n)=\begin{cases}\frac{1}{\sqrt{2}} & ;n=0\\ 1 & ;etc\end{cases} \qquad c_k=e(n)\cos\left[\frac{(2k+1)n\pi}{2\times 7}\right]$$

จากสมการ (3.1) ทำการแทนค่าของ n ตั้งแต่ 0 ถึง 7 จะได้ออกมาดังสมการ (3.2)

$$\begin{aligned}
y(0)&=c_4x(0)+c_4x(1)+c_4x(2)+c_4x(3)+c_4x(4)+c_4x(5)+c_4x(6)+c_4x(7)\\
y(1)&=c_1x(0)+c_3x(1)+c_5x(2)+c_7x(3)-c_7x(4)-c_5x(5)-c_3x(6)-c_1x(7)\\
y(2)&=c_2x(0)+c_6x(1)-c_6x(2)-c_2x(3)-c_2x(4)-c_6x(5)+c_6x(6)+c_2x(7)\\
y(3)&=c_3x(0)-c_7x(1)-c_1x(2)-c_5x(3)+c_5x(4)+c_1x(5)+c_7x(6)-c_3x(7)\\
y(4)&=c_4x(0)-c_4x(1)-c_4x(2)+c_4x(3)+c_4x(4)-c_4x(5)-c_4x(6)+c_4x(7)\\
y(5)&=c_5x(0)-c_1x(1)+c_7x(2)+c_3x(3)-c_3x(4)-c_7x(5)+c_1x(6)-c_5x(7)\\
y(6)&=c_6x(0)-c_2x(1)+c_2x(2)-c_6x(3)-c_6x(4)+c_2x(5)-c_2x(6)+c_6x(7)\\
y(7)&=c_7x(0)-c_5x(1)+c_3x(2)-c_1x(3)+c_1x(4)-c_3x(5)+c_5x(6)-c_7x(7)
\end{aligned} \tag{3.2}$$

และจากสมการ (3.2) สามารถแสดงให้อยู่ในรูปของเมตริกได้ดังนี้

$$\begin{bmatrix}y(0)\\y(1)\\y(2)\\y(3)\\y(4)\\y(5)\\y(6)\\y(7)\end{bmatrix}=\begin{bmatrix}c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4\\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1\\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2\\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3\\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4\\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5\\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6\\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7\end{bmatrix}\begin{bmatrix}x(0)\\x(1)\\x(2)\\x(3)\\x(4)\\x(5)\\x(6)\\x(7)\end{bmatrix} \tag{3.3}$$

จากสมการ (3.2) ทำการจัดรูปใหม่ จะได้ดังสมการ (3.4)

$$
\begin{aligned}
y(0) &= c_4\left[\left(x(0)+x(7)\right)+\left(x(3)+x(4)\right)\right]+c_4\left[\left(x(1)+x(6)\right)+\left(x(2)+x(5)\right)\right]\\
y(4) &= c_4\left[\left(x(0)+x(7)\right)+\left(x(3)+x(4)\right)\right]-c_4\left[\left(x(1)+x(6)\right)+\left(x(2)+x(5)\right)\right]\\
y(2) &= c_2\left[\left(x(0)+x(7)\right)-\left(x(3)+x(4)\right)\right]+c_6\left[\left(x(1)+x(6)\right)-\left(x(2)+x(5)\right)\right]\\
y(6) &= c_6\left[\left(x(0)+x(7)\right)-\left(x(3)+x(4)\right)\right]-c_4\left[\left(x(1)+x(6)\right)-\left(x(2)+x(5)\right)\right]\\
y(1) &= \left[c_1\left(x(0)-x(7)\right)+c_7\left(x(3)-x(4)\right)\right]+\left[c_3\left(x(1)-x(6)\right)+c_5\left(x(2)-x(5)\right)\right]\\
y(7) &= \left[c_7\left(x(0)-x(7)\right)-c_1\left(x(3)-x(4)\right)\right]-\left[c_5\left(x(1)-x(6)\right)-c_3\left(x(2)-x(5)\right)\right]\\
y(3) &= \left[c_3\left(x(0)-x(7)\right)-c_5\left(x(3)-x(4)\right)\right]-\left[c_1\left(x(2)-x(5)\right)+c_7\left(x(1)-x(6)\right)\right]\\
y(5) &= \left[c_5\left(x(0)-x(7)\right)+c_3\left(x(3)-x(4)\right)\right]+\left[c_7\left(x(2)-x(5)\right)-c_1\left(x(1)-x(6)\right)\right]
\end{aligned}
\tag{3.4}
$$

กำหนดให้

$$
\begin{aligned}
P_0 &= x(0)+x(7) & M_0 &= x(0)-x(7)\\
P_1 &= x(1)+x(6) & M_1 &= x(1)-x(6)\\
P_2 &= x(2)+x(5) \quad ; & M_2 &= x(2)-x(5)\\
P_3 &= x(3)+x(4) & M_3 &= x(3)-x(4)
\end{aligned}
$$

ทำให้สามารถแสดงสมการ (3.4) ได้ใหม่ดังนี้

$$
\begin{aligned}
y(0) &= c_4\left[P_0+P_3\right]+c_4\left[P_1+P_2\right]\\
y(4) &= c_4\left[P_0+P_3\right]-c_4\left[P_1+P_2\right]\\
y(2) &= c_2\left[P_0-P_3\right]+c_6\left[P_1-P_2\right]\\
y(6) &= c_6\left[P_0-P_3\right]-c_4\left[P_1-P_2\right]\\
y(1) &= \left[c_1M_0+c_7M_3\right]+\left[c_3M_1+c_5M_2\right]\\
y(7) &= \left[c_7M_0-c_1M_3\right]-\left[c_5M_1-c_3M_2\right]\\
y(3) &= \left[c_3M_0-c_5M_3\right]-\left[c_1M_2+c_7M_1\right]\\
y(5) &= \left[c_5M_0+c_3M_3\right]+\left[c_7M_2-c_1M_1\right]
\end{aligned}
\tag{3.5}
$$

ส่วนค่าสัมประสิทธิ์ที่ใช้ในการแปลงแบบ DCT สามารถแสดงได้ดังตารางที่ 3.1

ค่าสัมประสิทธิ์	ค่าที่ได้จากสมการ	ค่าที่แปลงเป็นเลขฐานสอง	ค่าที่แปลงกลับมาเป็นเลขฐานสิบ	ค่าผิดพลาด
$c_1\left(\cos\left(\frac{\pi}{16}\right)\right)$	0.980	0.1111101	0.976565625	0.34343 %
$c_2\left(\cos\left(\frac{2\pi}{16}\right)\right)$	0.923	0.1110110	0.921875000	0.11250 %
$c_3\left(\cos\left(\frac{3\pi}{16}\right)\right)$	0.831	0.1101010	0.828125000	0.28750 %
$c_4\left(\cos\left(\frac{4\pi}{16}\right)\right)$	0.707	0.1011010	0.703125000	0.38750 %
$c_5\left(\cos\left(\frac{5\pi}{16}\right)\right)$	0.555	0.1000111	0.554687500	0.03125 %
$c_6\left(\cos\left(\frac{6\pi}{16}\right)\right)$	0.382	0.0110000	0.375000000	0.70000 %
$c_7\left(\cos\left(\frac{7\pi}{16}\right)\right)$	0.195	0.0011000	0.187500000	0.75000 %

ตารางที่ 3.1 แสดงค่าของสัมประสิทธิ์ที่ใช้ในการแปลง DCT

จากตารางที่ 3.1 จะเห็นได้ว่าค่าที่แปลงเป็นเลขฐานสองแล้ว จะมีค่าผิดพลาดเกิดขึ้น จึงได้ทำการเพิ่มค่าโดยการบวกเพิ่มเข้าไปที่บิทสุดท้ายหนึ่งบิท ซึ่งสามารถแสดงได้ดังตารางที่ 3.2

ค่าสัมประสิทธิ์	ค่าที่ได้จากสมการ	ค่าที่แปลงเป็นเลขฐานสอง	ค่าที่แปลงกลับมาเป็นเลขฐานสิบ	ค่าผิดพลาด
$c_1\left(\cos\left(\frac{\pi}{16}\right)\right)$	0.980	0.1111110	0.9843750	0.43750 %
$c_2\left(\cos\left(\frac{2\pi}{16}\right)\right)$	0.923	0.1110111	0.9296875	0.66875 %
$c_3\left(\cos\left(\frac{3\pi}{16}\right)\right)$	0.831	0.1101011	0.8359375	0.49375 %

ตารางที่ 3.2 แสดงค่าของสัมประสิทธิ์ที่ใช้ในการแปลง DCT ที่ทำการเพิ่มค่าที่บิทสุดท้าย

ค่าสัมประสิทธิ์	ค่าที่ได้จากสมการ	ค่าที่แปลงเป็นเลขฐานสอง	ค่าที่แปลงกลับมาเป็นเลขฐานสิบ	ค่าผิดพลาด
$c_4\left(\cos\left(\frac{4\pi}{16}\right)\right)$	0.707	0.1011011	0.7109375	0.39375 %
$c_5\left(\cos\left(\frac{5\pi}{16}\right)\right)$	0.555	0.1001000	0.5625000	0.75000 %
$c_6\left(\cos\left(\frac{6\pi}{16}\right)\right)$	0.382	0.0110001	0.3828125	0.08125 %
$c_7\left(\cos\left(\frac{7\pi}{16}\right)\right)$	0.195	0.0011001	0.1953125	0.03125 %

ตารางที่ 3.2 (ต่อ) แสดงค่าของสัมประสิทธิ์ที่ใช้ในการแปลง DCT ที่ทำการเพิ่มค่าที่บิทสุดท้าย

ทำการเปรียบเทียบค่าความผิดพลาดของสัมประสิทธิ์ที่ใช้ในการแปลงแบบ DCT ที่ไม่ได้ทำการเพิ่มค่าที่บิทสุดท้ายกับค่าที่ได้ทำการเพิ่มค่าที่บิทสุดท้าย สามารถทำการแสดงได้ดังตารางที่ 3.3

ค่าสัมประสิทธิ์	ค่าผิดพลาดที่ไม่ได้ทำการเพิ่มบิทสุดท้าย	ค่าผิดพลาดที่ทำการเพิ่มบิทสุดท้าย
$c_1\left(\cos\left(\frac{\pi}{16}\right)\right)$	0.34343 %	0.43750 %
$c_2\left(\cos\left(\frac{2\pi}{16}\right)\right)$	0.11250 %	0.66875 %
$c_3\left(\cos\left(\frac{3\pi}{16}\right)\right)$	0.28750 %	0.49375 %
$c_4\left(\cos\left(\frac{4\pi}{16}\right)\right)$	0.38750 %	0.39375 %
$c_5\left(\cos\left(\frac{5\pi}{16}\right)\right)$	0.03125 %	0.75000 %
$c_6\left(\cos\left(\frac{6\pi}{16}\right)\right)$	0.70000 %	0.08125 %
$c_7\left(\cos\left(\frac{7\pi}{16}\right)\right)$	0.75000 %	0.03125 %

ตารางที่ 3.3 แสดงการเปรียบเทียบของค่าความผิดพลาด

จากการพิจารณาในตารางที่ 3.3 จะเห็นได้ว่า ค่าผิดพลาดที่เกิดขึ้นมีความแตกต่างกันมาก ดังนั้นในการสร้างจึงทำการเลือกค่าของสัมประสิทธิ์ที่ใช้ในการแปลงแบบ DCT ที่มีค่าความผิดพลาดเกิดขึ้นน้อยที่สุด ซึ่งสามารถแสดงให้เห็นได้ดังตารางที่ 3.4

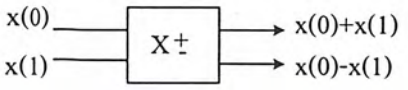
ค่าสัมประสิทธิ์	ค่าที่ได้จากสมการ	ค่าที่แปลงเป็นเลขฐานสอง	ค่าที่แปลงกลับมาเป็นเลขฐานสิบ	ค่าผิดพลาด
$c_1\left(\cos\left(\frac{\pi}{16}\right)\right)$	0.980	0.1111101	0.976565625	0.34343 %
$c_2\left(\cos\left(\frac{2\pi}{16}\right)\right)$	0.923	0.1110110	0.921875000	0.11250 %
$c_3\left(\cos\left(\frac{3\pi}{16}\right)\right)$	0.831	0.1101010	0.828125000	0.28750 %
$c_4\left(\cos\left(\frac{4\pi}{16}\right)\right)$	0.707	0.1011010	0.703125000	0.38750 %
$c_5\left(\cos\left(\frac{5\pi}{16}\right)\right)$	0.555	0.1000111	0.554687500	0.03125 %
$c_6\left(\cos\left(\frac{6\pi}{16}\right)\right)$	0.382	0.0110001	0.382812500	0.08125 %
$c_7\left(\cos\left(\frac{7\pi}{16}\right)\right)$	0.195	0.0011001	0.195312500	0.03125 %

ตารางที่ 3.4 แสดงค่าของสัมประสิทธิ์ที่ใช้ในการแปลง DCT
ที่ทำการแก้ไขให้มีค่าผิดพลาดน้อยที่สุด

3.1.2 โครงสร้างของ Discrete Cosine Transform

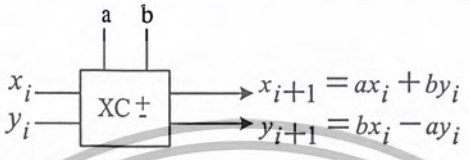
การสร้างการแปลงแบบ DCT จะใช้สมการ (3.5) ซึ่งได้ผ่านการจัดรูปแบบมาแล้ว ทำให้มีการใช้จำนวนของตัวคูณน้อยกว่าสมการ (3.2) ที่เป็นสมการพื้นฐานถึง 40 ตัว

3.1.2.1 ส่วนของวงจรบวกและลบ เป็นบล็อกไดอะแกรมที่แสดงให้เห็นถึงการนำสัญญาณอินพุทที่เข้ามา 2 ทาง มาทำการบวกและลบกันได้เป็นเอาท์พุทออกมา 2 ทาง ซึ่งทางหนึ่งจะเป็นการบวกกัน ส่วนอีกทางหนึ่งจะเป็นการลบกันของสัญญาณอินพุท โดยที่สัญญาณอินพุททั้ง 2 ทาง จะต้องมีขนาดของบิทข้อมูลที่เท่ากัน และได้เอาท์พุทที่มีขนาดของบิทข้อมูลมากกว่าอินพุทอยู่ 1 บิท มีลักษณะดังรูปที่ 3.1



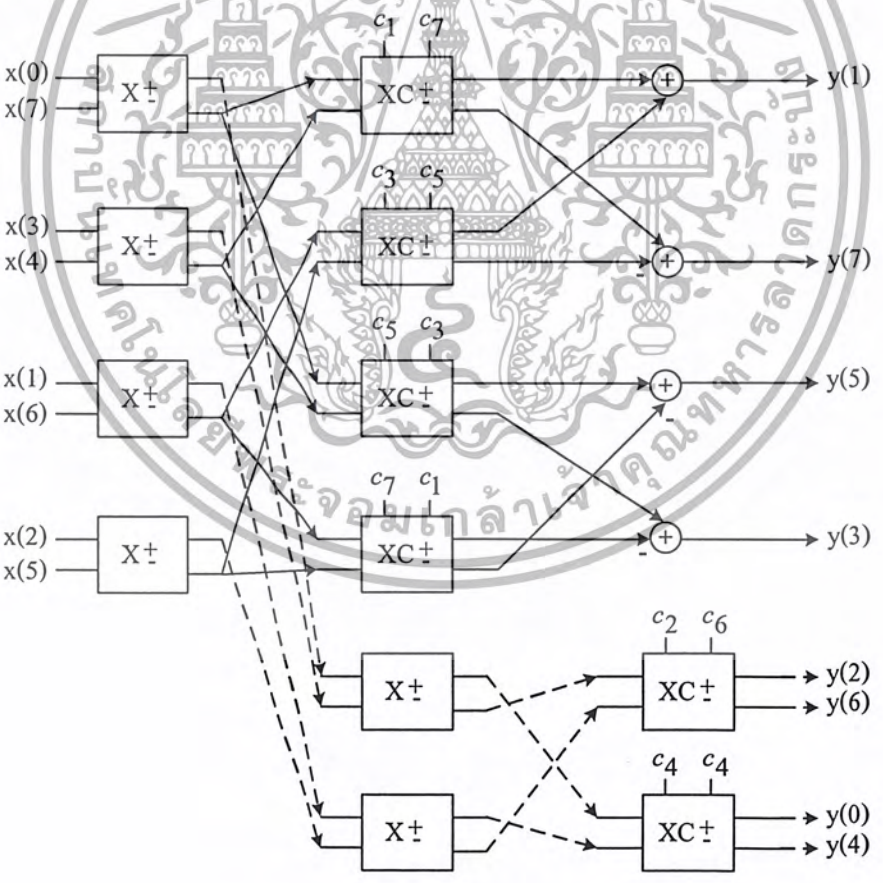
รูปที่ 3.1 แสดงบล็อกไดอะแกรมที่ใช้ในการบวกและลบ

3.1.2.2 ส่วนของวงจรคูณกับค่าสัมประสิทธิ์ในการแปลงแบบ DCT เป็นบล็อกไดอะแกรมที่แสดงถึงการนำสัญญาณอินพุทมาคูณกับค่าสัมประสิทธิ์ ที่ทำการลดจำนวนตัวคูณลง



รูปที่ 3.2 บล็อกไดอะแกรมที่ใช้คูณกับค่าสัมประสิทธิ์ในการแปลงแบบ DCT

3.1.2.3 โครงสร้างโดยรวมของการแปลงแบบ DCT



รูปที่ 3.3 โครงสร้างโดยรวมของการแปลงแบบ DCT แบบ 1 มิติ

3.2. การออกแบบที่นำหลักการของ CORDIC Algorithm มาประยุกต์ใช้กับการแปลงแบบ DCT

3.2.1 การคำนวณหาสมการและสัมประสิทธิ์ที่ใช้ในการสร้าง

จากหลักการของ CORDIC Algorithm ซึ่งจะแสดงรูปแบบของสมการพื้นฐานคือ

$$\begin{aligned} x_{i+1} &= x_i \cos\theta_i - y_i \sin\theta_i \\ y_{i+1} &= x_i \sin\theta_i + y_i \cos\theta_i \end{aligned} \tag{3.6}$$

สามารถแสดงอยู่ในรูปแบบของเมตริกได้ดังนี้

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{3.7}$$

ค่าสัมประสิทธิ์สามารถแสดงให้อยู่ในรูปแบบของฟังก์ชันตรีโกณมิติได้ดังตารางที่ 3.5

ค่าสัมประสิทธิ์	ฟังก์ชั่นโคไซน์ (cosine)	ฟังก์ชั่นไซน์ (sine)
c_1	$\cos\left(\frac{\pi}{16}\right)$	$\sin\left(\frac{7\pi}{16}\right)$
c_2	$\cos\left(\frac{2\pi}{16}\right)$	$\sin\left(\frac{6\pi}{16}\right)$
c_3	$\cos\left(\frac{3\pi}{16}\right)$	$\sin\left(\frac{5\pi}{16}\right)$
c_4	$\cos\left(\frac{4\pi}{16}\right)$	$\sin\left(\frac{4\pi}{16}\right)$
c_5	$\cos\left(\frac{5\pi}{16}\right)$	$\sin\left(\frac{3\pi}{16}\right)$
c_6	$\cos\left(\frac{6\pi}{16}\right)$	$\sin\left(\frac{2\pi}{16}\right)$
c_7	$\cos\left(\frac{7\pi}{16}\right)$	$\sin\left(\frac{\pi}{16}\right)$

ตารางที่ 3.5 แสดงค่าสัมประสิทธิ์ในรูปแบบของฟังก์ชันตรีโกณมิติ

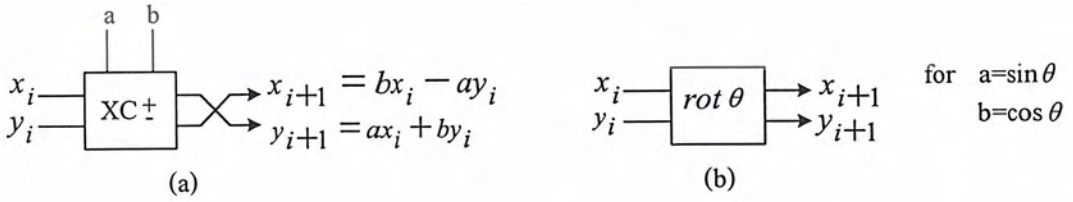
จากสมการ (3.5) ทำการค่าของสัมประสิทธิ์ด้วยค่าในตารางที่ 3.5 แล้วจัดให้อยู่ในรูปแบบที่เหมือนกับสมการ (3.6) ที่เป็นสมการพื้นฐานของ CORDIC Algorithm ได้ดังนี้

$$y(0)=(P_0+P_3)\sin\left(\frac{4\pi}{16}\right)+(P_1+P_2)\cos\left(\frac{4\pi}{16}\right)$$
$$y(4)=(P_0+P_3)\cos\left(\frac{4\pi}{16}\right)-(P_1+P_2)\sin\left(\frac{4\pi}{16}\right)$$
$$y(2)=(P_0-P_3)\sin\left(\frac{6\pi}{16}\right)+(P_1-P_2)\cos\left(\frac{6\pi}{16}\right)$$
$$y(6)=(P_0-P_3)\cos\left(\frac{6\pi}{16}\right)-(P_1-P_2)\sin\left(\frac{6\pi}{16}\right) \quad (3.8)$$

$$y(1)=\left[M_0\sin\left(\frac{7\pi}{16}\right)+M_3\cos\left(\frac{7\pi}{16}\right)\right]+\left[M_1\sin\left(\frac{5\pi}{16}\right)+M_2\cos\left(\frac{5\pi}{16}\right)\right]$$
$$y(7)=\left[M_0\cos\left(\frac{7\pi}{16}\right)-M_3\sin\left(\frac{7\pi}{16}\right)\right]-\left[M_1\cos\left(\frac{5\pi}{16}\right)-M_2\sin\left(\frac{5\pi}{16}\right)\right]$$
$$y(3)=\left[M_0\cos\left(\frac{3\pi}{16}\right)-M_3\sin\left(\frac{3\pi}{16}\right)\right]-\left[M_1\sin\left(\frac{\pi}{16}\right)+M_2\cos\left(\frac{\pi}{16}\right)\right]$$
$$y(5)=\left[M_0\sin\left(\frac{3\pi}{16}\right)+M_3\cos\left(\frac{3\pi}{16}\right)\right]-\left[M_1\cos\left(\frac{\pi}{16}\right)-M_2\sin\left(\frac{\pi}{16}\right)\right]$$

3.2.2 โครงสร้างของ CORDIC Algorithm ที่นำไปประยุกต์ใช้ในการแปลงแบบ DCT

3.2.2.1 ส่วนของวงจรที่ใช้หลักการของ CORDIC Algorithm



รูปที่ 3.4 (a) บล็อกไดอะแกรมที่ยังไม่ใช้หลักการของ CORDIC Algorithm
(b) บล็อกไดอะแกรมที่ใช้หลักการของ CORDIC Algorithm

จากรูปที่ 3.4 จะเห็นได้ว่าได้นำเอาหลักการของ CORDIC Algorithm มาใช้แทนวงจรคูณในการแปลงแบบ DCT โดยจะใช้หลักการหมุนจากค่าอินพุทที่เข้ามาด้วยมุม θ (rotθ) ซึ่งมุมθ นี้ได้มาจากค่ามุมของสัมประสิทธิ์ในการแปลง DCT (c_k) และในการสร้างจะใช้ทฤษฎีเลขคณิตกระจาย (DA Algorithm) จะได้ดังนี้

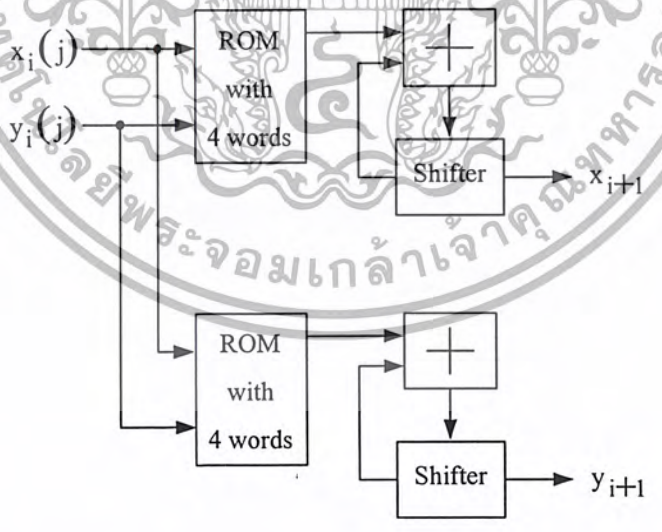
$$x = -x(0) + \sum_{j=1}^{B-1} x(j)2^{-j}$$
$$y = -y(0) + \sum_{j=1}^{B-1} y(j)2^{-j} \qquad (3.9)$$

โดยที่ B คือ จำนวนบิทของข้อมูล

นำสมการ (3.9) แทนลงไปในสมการ (3.6)

$$x_{i+1} = \sum_{j=1}^{B-1}\left(x_i(j)\cos\theta_i - y_i(j)\sin\theta_i\right)2^{-j} - \left(x_i(0)\cos\theta_i - y_i(0)\sin\theta_i\right)$$
$$y_{i+1} = \sum_{j=1}^{B-1}\left(y_i(j)\cos\theta_i + x_i(j)\sin\theta_i\right)2^{-j} - \left(y_i(0)\cos\theta_i + x_i(0)\sin\theta_i\right) \qquad (3.10)$$

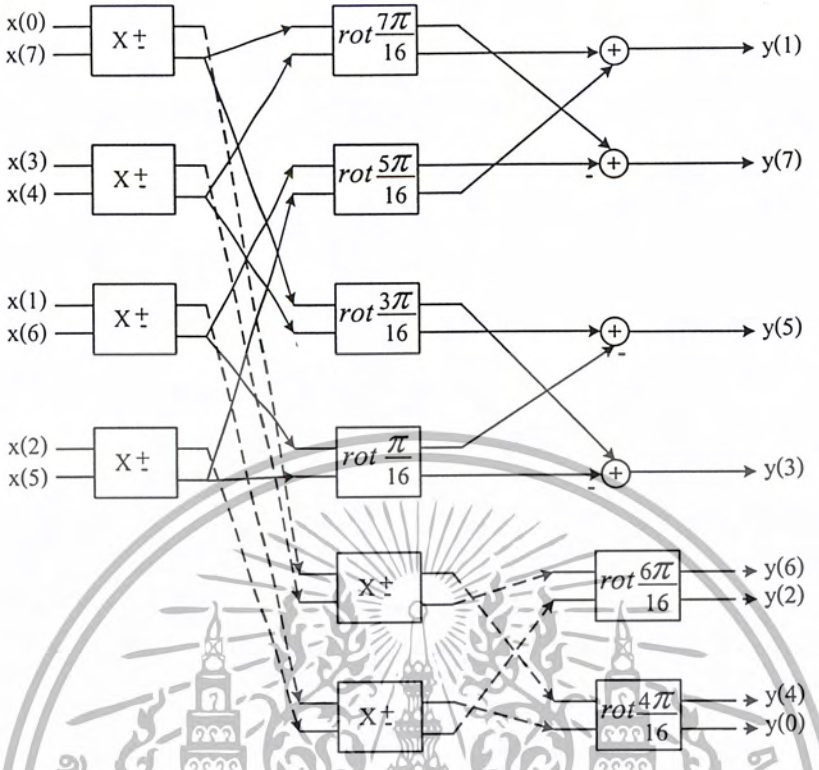
จากสมการ (3.10) สามารถเขียนเป็นโครงสร้างได้ดังรูป 3.5



รูปที่ 3.5 โครงสร้างของ CORDIC Algorithm โดยใช้ทฤษฎีเลขคณิตกระจาย

ซึ่งค่าใน ROM เป็นไปได้ 4 กรณี เนื่องจากมีค่าอินพุท 2 ตัว คือ $x_i(j)$ กับ $y_i(j)$ และค่าของอินพุทแต่ละตัวสามารถเป็นไปได้แค่ 0 กับ 1 เท่านั้น

3.2.2.2 โครงสร้างของ DCT ที่ใช้หลักการของ CORDIC Algorithm



รูปที่ 3.6 แสดงโครงสร้างการแปลงแบบ DCT แบบ 1 มิติ ที่ใช้หลักการของ CORDIC Algorithm

บทที่ 4

การทดลองและผลการทดลอง

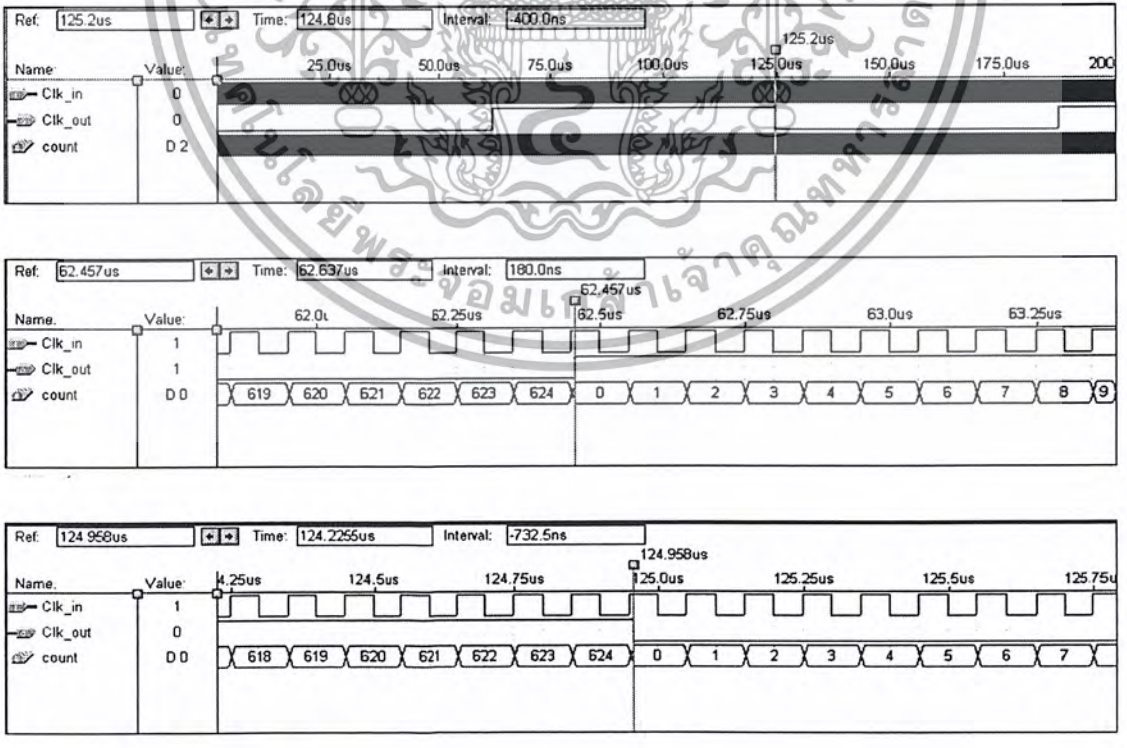
การออกแบบส่วนต่างๆ ของการแปลง DCT โดยการใช้ CORDIC (Coordinate Rotation Digital Computer) อัลกอริธึม ทำการเขียนโปรแกรมโดยให้แต่ละส่วนทำงานตามที่ได้ออกแบบโดยใช้ภาษา VHDL ทำการ Compile แล้วทำการจำลองการทำงานของโปรแกรมแต่ละส่วนที่ได้เขียนขึ้นให้ได้ผลตามที่ออกแบบไว้ และวัดผลการแปลง DCT ด้วยออสซิโลสโคป

4.1 ส่วนของการสร้างวงจรหารความถี่ (DIV)

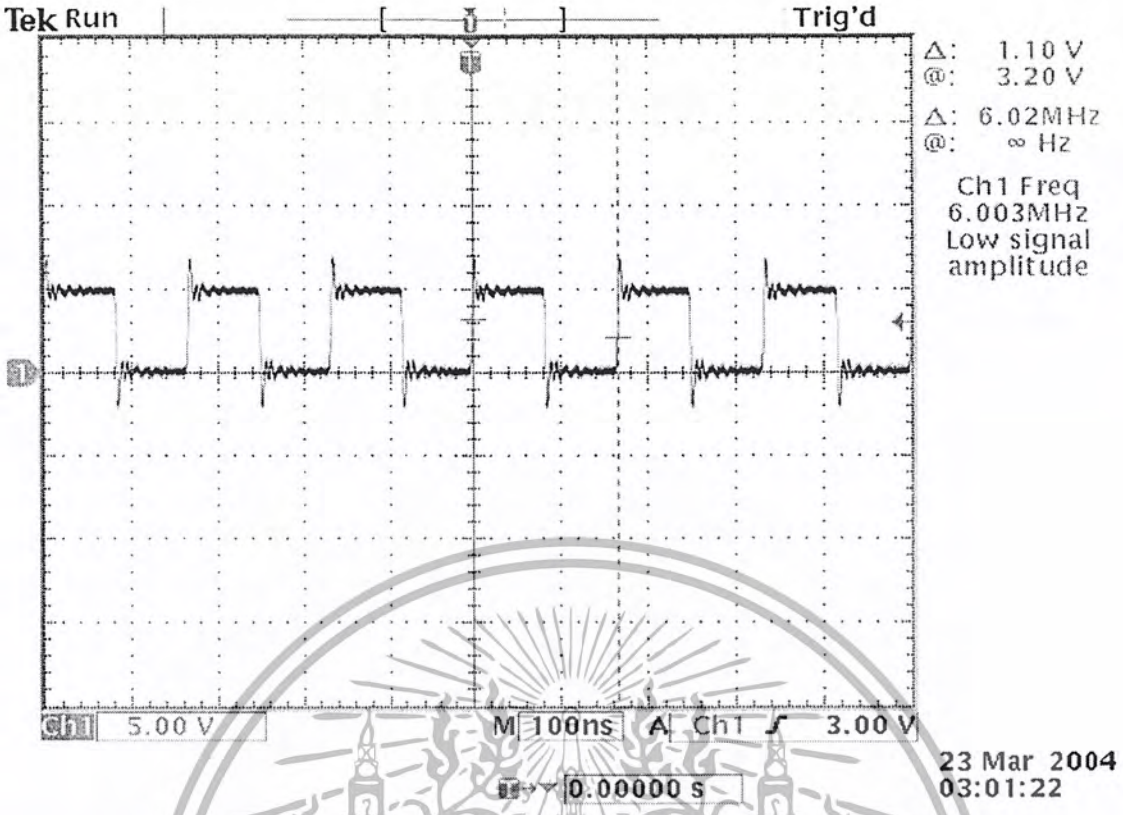
ส่วนของการสร้างวงจรหารความถี่ (DIV) ทำหน้าที่หารความถี่ที่ได้จาก โมดูลออสซิเลเตอร์ที่กำเนิดสัญญาณนาฬิกาให้ได้อัตราการส่งข้อมูล (Baud rate) ตามที่เรากำหนดไว้ สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.1

รูปที่ 4.1 สัญลักษณ์ของส่วนของวงจรหารความถี่

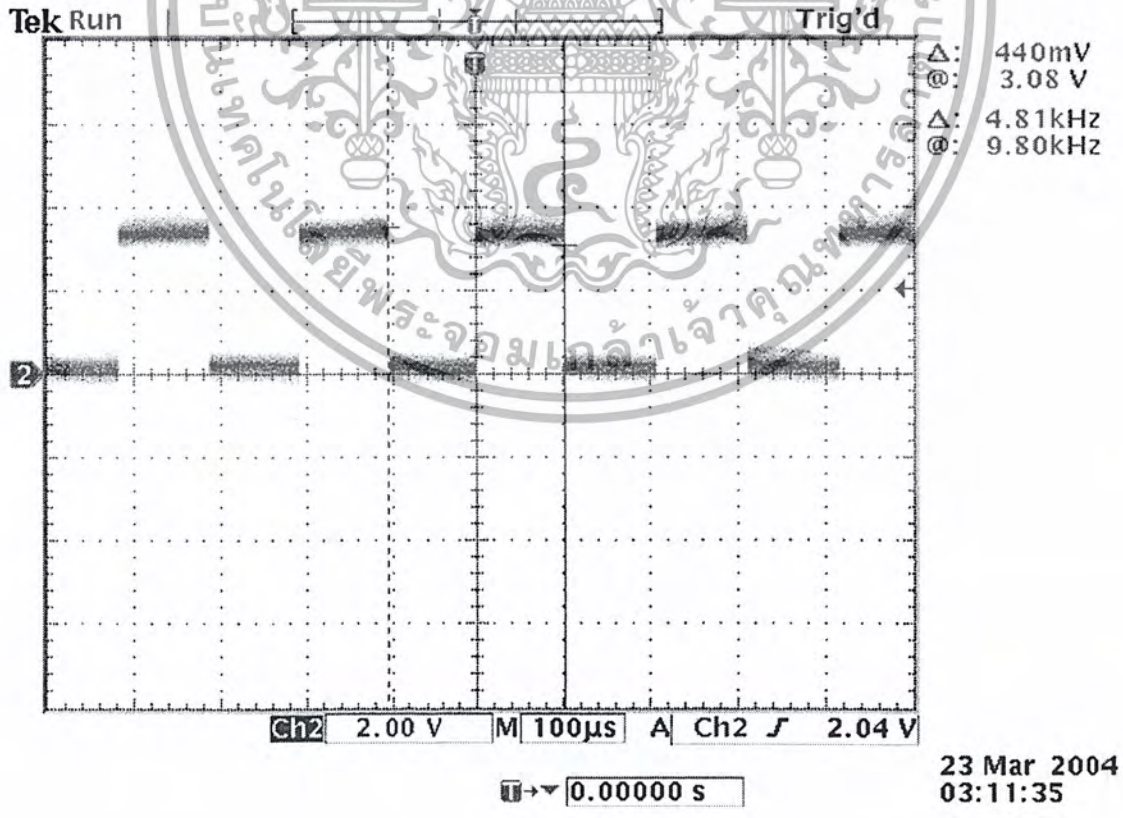
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



รูปที่ 4.2 ผลการจำลองการทำงานของส่วนวงจรหารความถี่



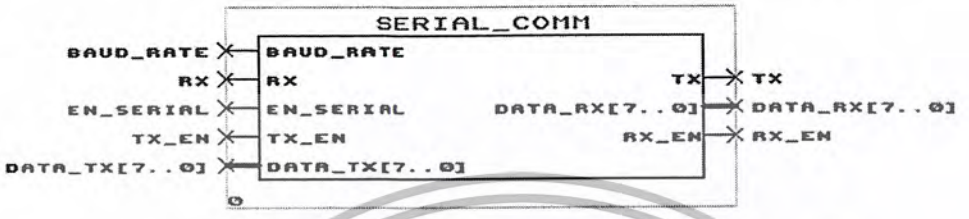
รูปที่ 4.3 ส่วนของอินพุตของวงจรลดค่าความถี่ (สัญญาณนาฬิกาบนบอร์ดเอฟพีจีเอความถี่ 6 เมกกะเฮิร์ต)



รูปที่ 4.4 สัญญาณอัตราการส่งข้อมูล 4800 บิทต่อวินาทีที่ได้จากวงจรหารความถี่ (DIV)

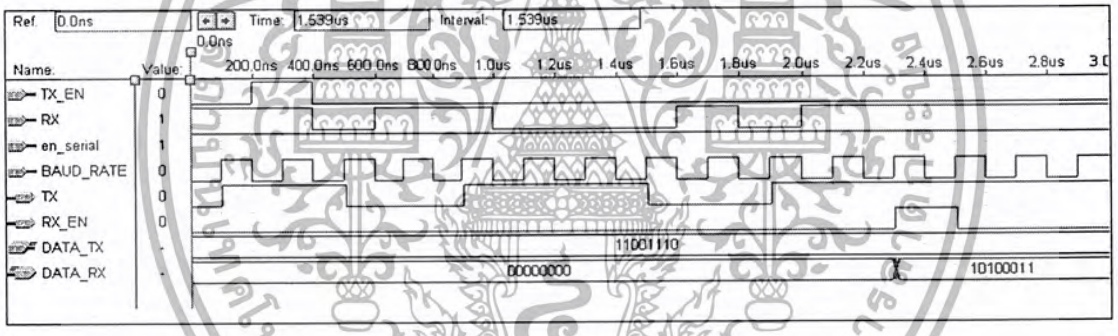
4.2 ส่วนของวงจรสื่อสารข้อมูลแบบอะซิงโครนัส (UART)

ส่วนของวงจรสื่อสารข้อมูลแบบอะซิงโครนัส (UART) หน้าที่หลักของ ส่วนของวงจรสื่อสารข้อมูลแบบอะซิงโครนัส คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานให้อยู่ในรูปแบบอนุกรมซิงโครนัสแล้วส่งออกไป และทำหน้าที่แปลงสัญญาณแบบอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง ส่วนของวงจรสื่อสารข้อมูลแบบอะซิงโครนัส ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่การทำงานลำดับต่อไป สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.5



รูปที่ 4.5 สัญลักษณ์ของวงจรสื่อสารข้อมูลแบบอะซิงโครนัส

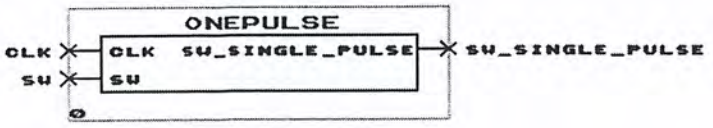
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



รูปที่ 4.6 ผลการจำลองการทำงานของส่วนวงจรสื่อสารข้อมูลแบบอะซิงโครนัส

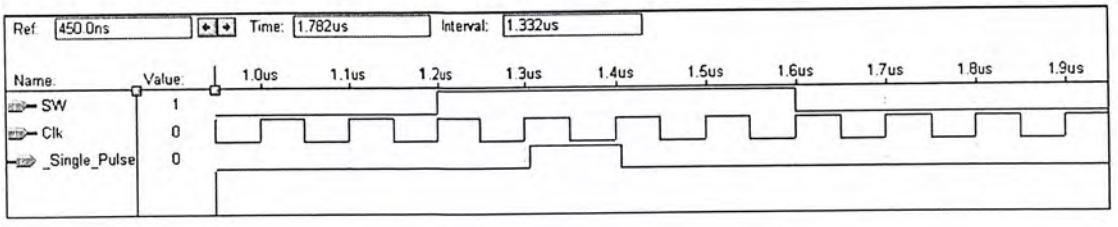
4.3 ส่วนของวงจรสร้างสัญญาณพัลส์ (One Pulse)

ส่วนของสร้างสัญญาณพัลส์ (One Pulse) ทำการสร้างสัญญาณพัลส์เพื่อสร้างสัญญาณทริก เพื่อเลื่อนการรับค่าอินพุทที่รับเข้ามาจากวงจรสื่อสารข้อมูลแบบอะซิงโครนัสเสร็จแล้วก็จะนำค่าเข้าไปเก็บในจงจรคงค่าสัญญาณชั่วคราว (Latch) เพื่อรอสัญญาณอินพุทจนครบ 8 ค่าแล้วจึงทำการหาค่าการแปลง DCT ที่ละชุดสามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.7



รูปที่ 4.7 สัญลักษณ์ของวงจรสร้างสัญญาณพัลส์(One Pulse)

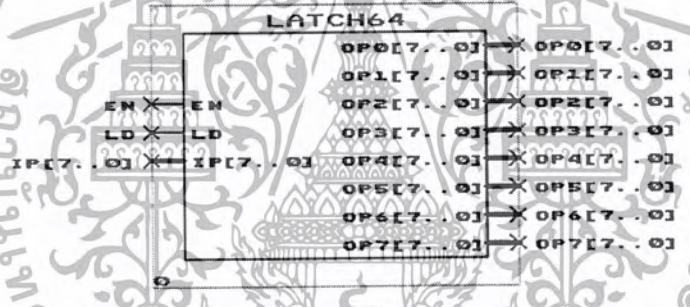
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



รูปที่ 4.8 ผลการจำลองการทำงานของวงจรสร้างสัญญาณพัลส์ (One Pulse)

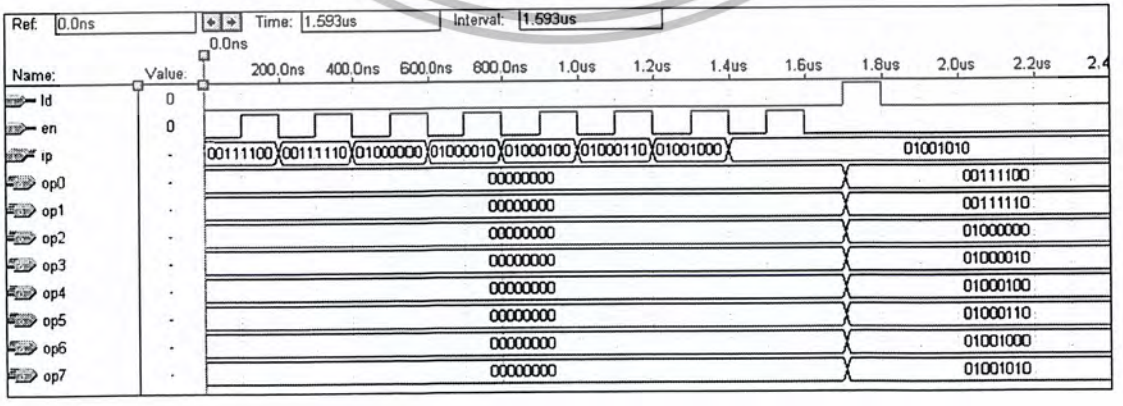
4.4 ส่วนของวงจรคงค่าสัญญาณชั่วคราว

ส่วนของวงจรคงค่าสัญญาณชั่วคราว ทำการเก็บข้อมูลอินพุทที่ป้อนเข้ามาชั่วคราวให้ครบ 8 ค่า แล้วจะทำการโอนค่าทั้ง 8 ค่าไปให้ส่วนของการแปลง DCT แล้วก็รับค่าข้อมูลชุดใหม่ให้ครบทั้ง 8 ค่าในรอบถัดไป จะทำงานซ้ำแบบนี้จนกระทั่งไม่มีการป้อนสัญญาณอินพุทเข้ามา สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.9



รูปที่ 4.9 สัญลักษณ์ของวงจรคงค่าสัญญาณชั่วคราว

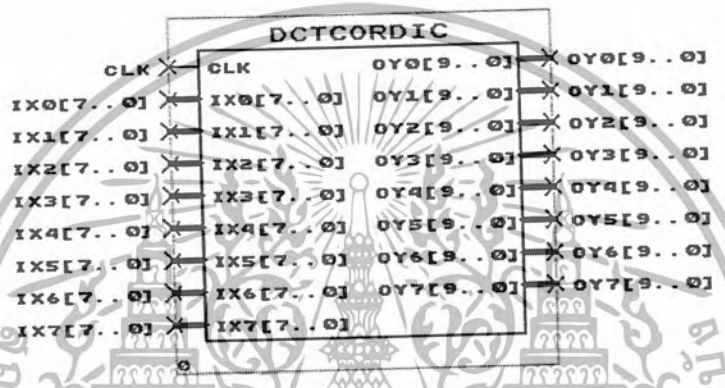
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



รูปที่ 4.10 ผลการจำลองการทำงานของวงจรคงค่าสัญญาณชั่วคราว

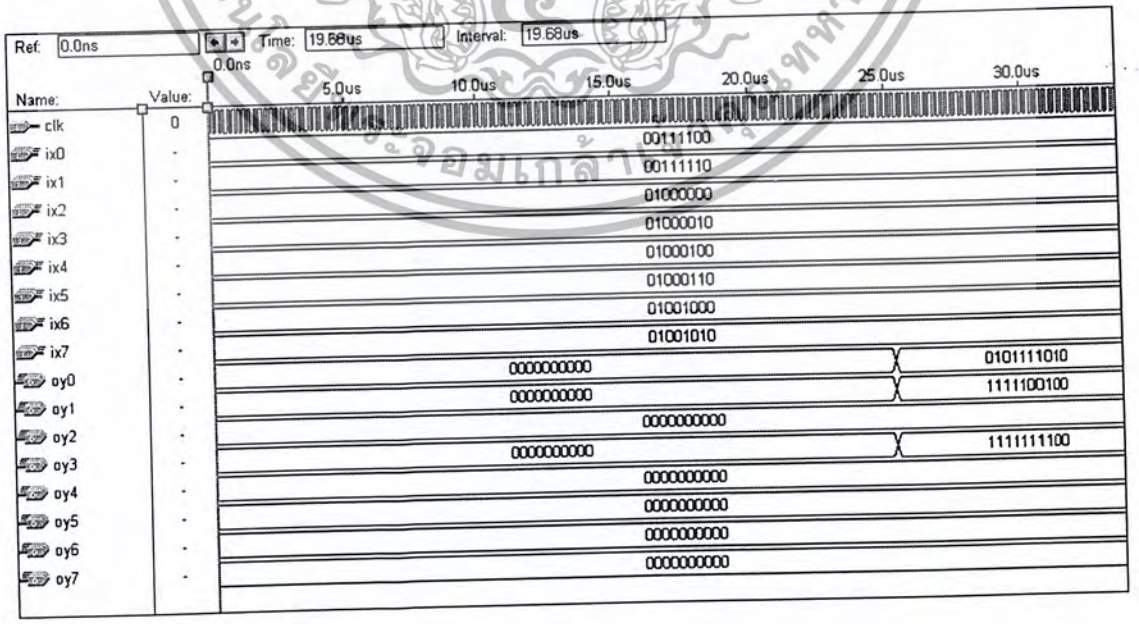
4.5 ส่วนของวงจรการแปลง DCT

ส่วนของวงจรการแปลง DCT ทำหน้าที่ในการคำนวณค่าการแปลง DCT ของข้อมูลที่ป้อนเข้ามาจากวงจรส่วนของวงจรคงค่าสัญญาณชั่วคราว ขนาด 8 บิท 8 ค่าซึ่งภายในส่วนของวงจรการแปลง DCT ประกอบไปด้วยส่วนของวงจรบวกและลบ 8 บิท, ส่วนของวงจรบวกและลบ 9 บิท, ส่วนของวงจรบวก 9 บิท,ส่วนของวงจรลบ 9 บิท,ส่วนของวงจร CORDIC ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ, ส่วนของวงจรรีจีสเตอร์ขนาด 10 บิท และ ส่วนของวงจรคอนโทรล (controller) ซึ่งส่วนของวงจรในส่วนนี้เป็นส่วนที่สำคัญมากในการทำโครงงานนี้ สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.11



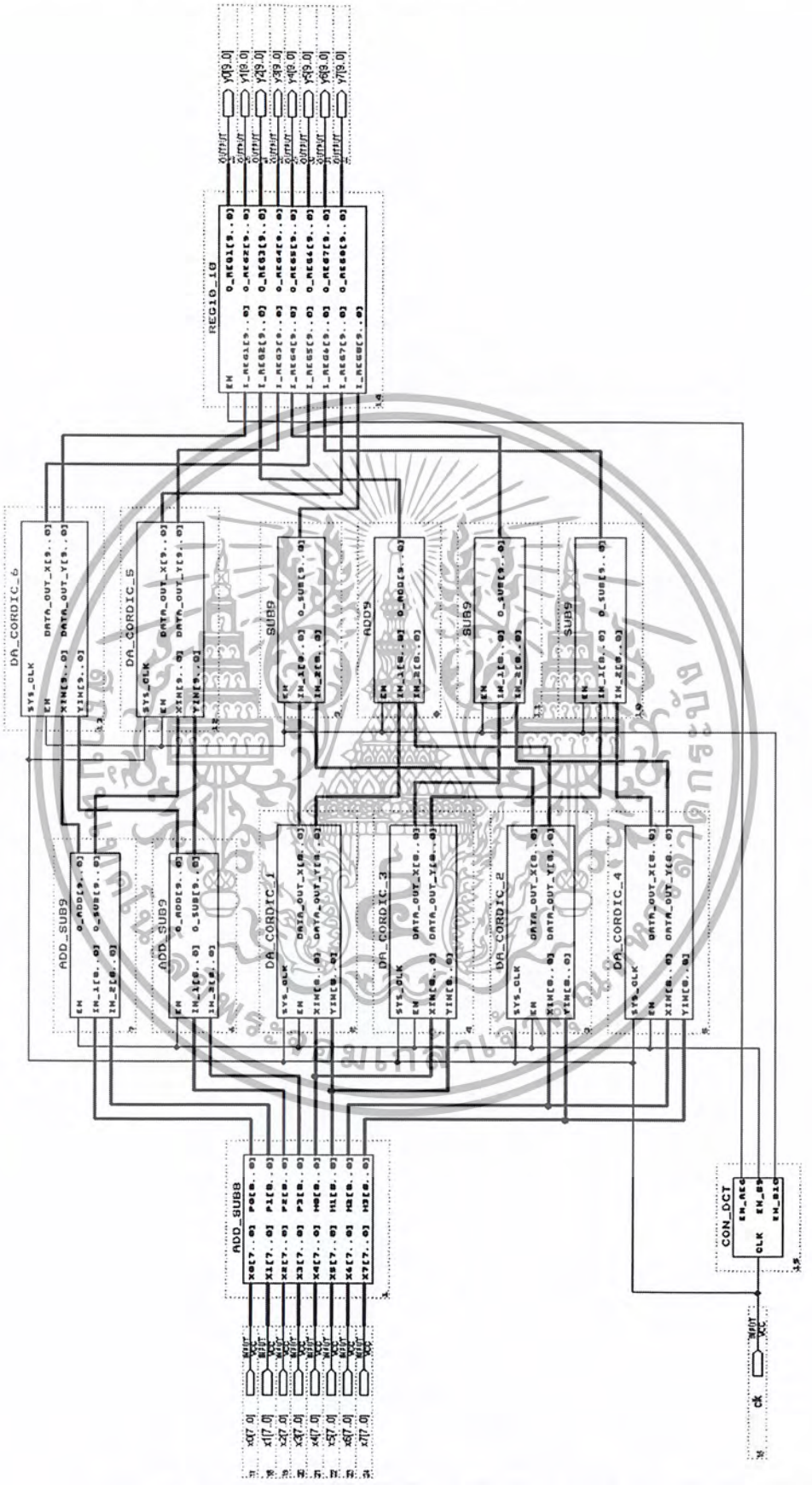
รูปที่ 4.11 สัญลักษณ์ของวงจรการแปลง DCT

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



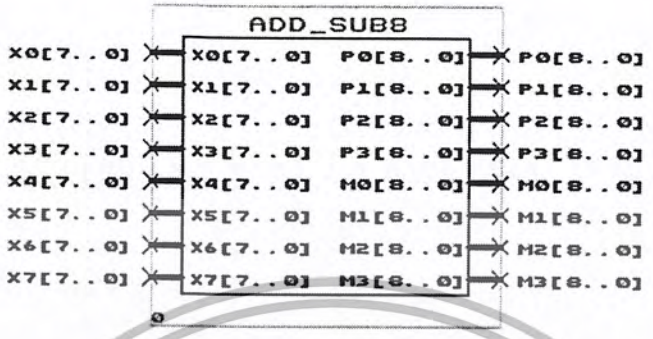
รูปที่ 4.12 ผลการจำลองการทำงานของวงจรการแปลง DCT

ส่วนประกอบภายในของวงจรการแปลง DCT มีดังนี้



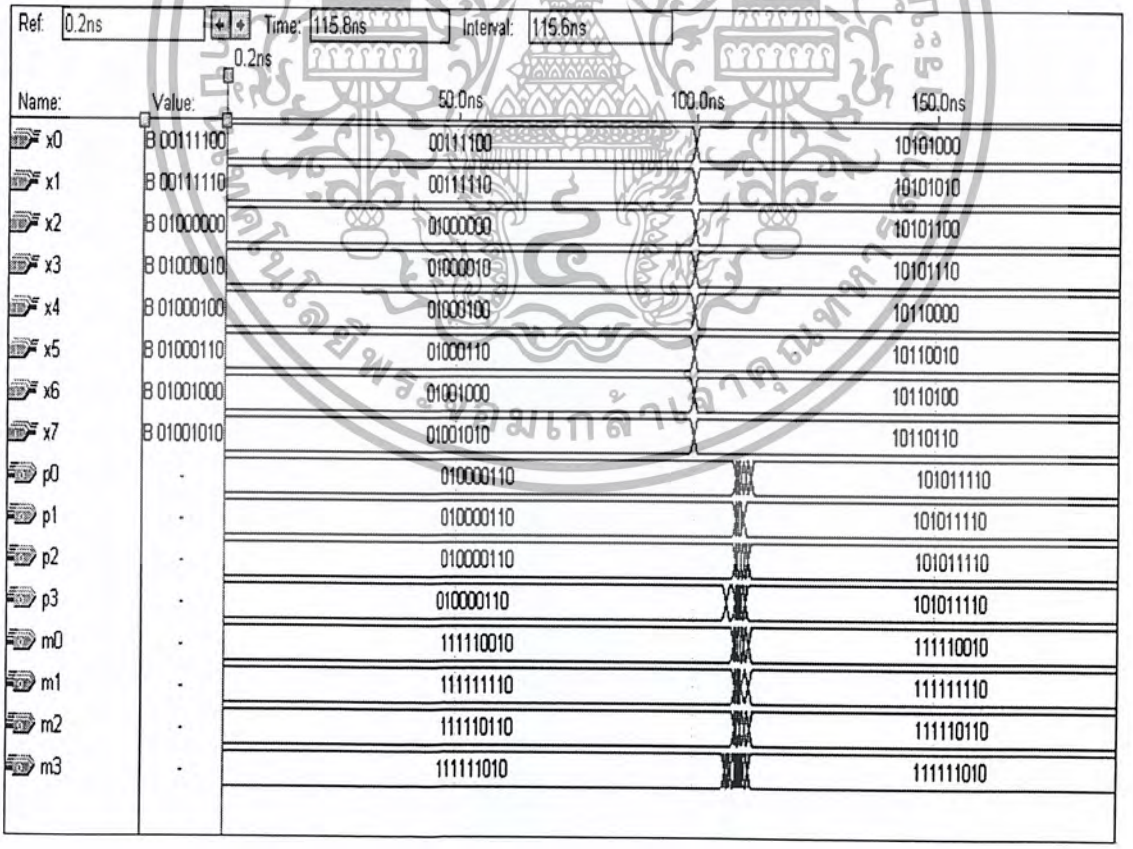
รูปที่ 4.13 ส่วนประกอบภายในของวงจรการแปลง DCT

4.5.1 ส่วนของวงจรบวกและลบ 8 บิท ทำหน้าที่ในการบวกและลบข้อมูลขนาด 8 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.14



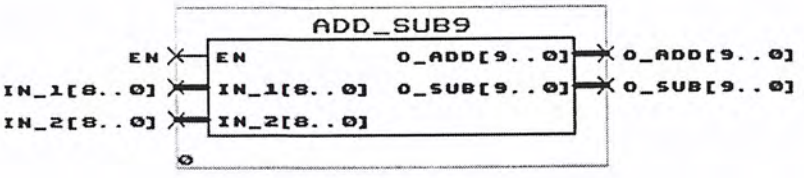
รูปที่ 4.14 สัญลักษณ์ส่วนส่วนของวงจรส่วนของวงจรบวกและลบ 8 บิท

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



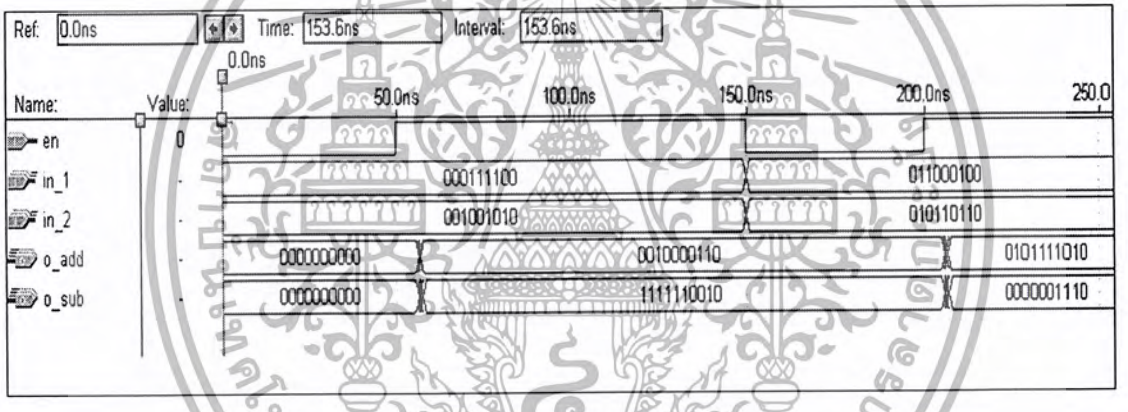
รูปที่ 4.15 ผลการจำลองการทำงานของวงจรบวกและลบ 8 บิท

4.5.2 ส่วนของวงจรบวกและลบ 9 บิท ทำหน้าที่ในการบวกและลบข้อมูลขนาด 9 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.16



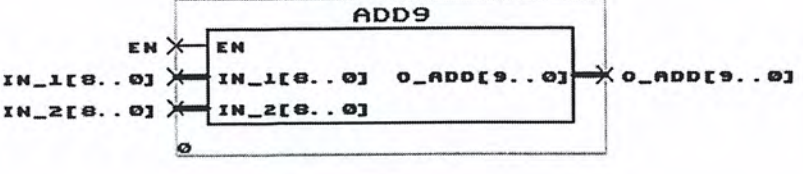
รูปที่ 4.16 สัญลักษณ์ส่วนของวงจรส่วนของวงจรบวกและลบ 9 บิท

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



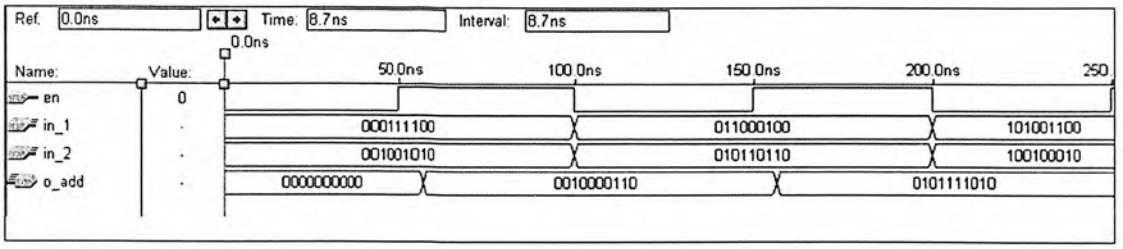
รูปที่ 4.17 ผลการจำลองการทำงานของวงจรส่วนของวงจรบวกและลบ 9 บิท

4.5.3 ส่วนของวงจรบวก 9 บิท ทำหน้าที่ในการบวกข้อมูลขนาด 9 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.18



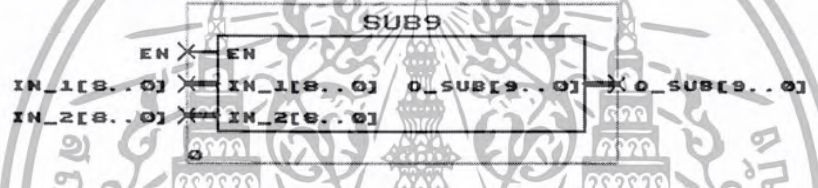
รูปที่ 4.18 สัญลักษณ์ส่วนของวงจรบวก 9 บิท

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



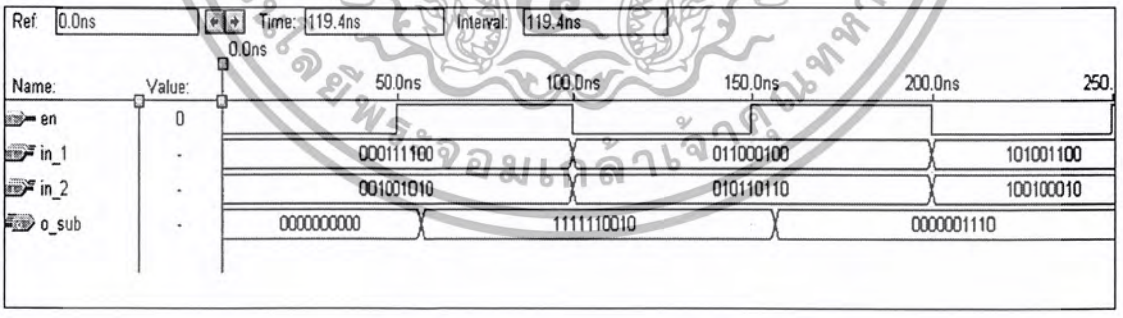
รูปที่ 4.19 ผลการจำลองการทำงานของวงจรบวก 9 บิท

4.5.4 ส่วนของวงจรลบ 9 บิท ทำหน้าที่ในการลบข้อมูลขนาด 9 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.20



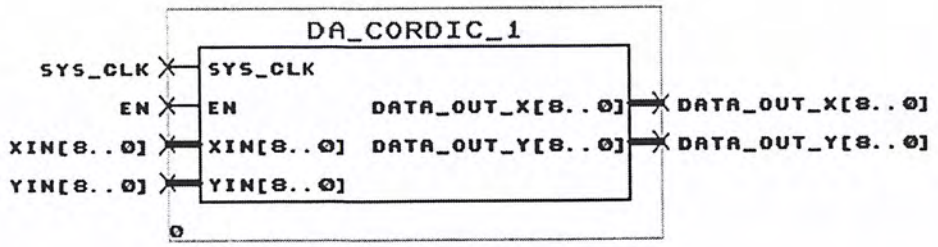
รูปที่ 4.20 สัญลักษณ์ส่วนของวงจรลบ 9 บิท

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



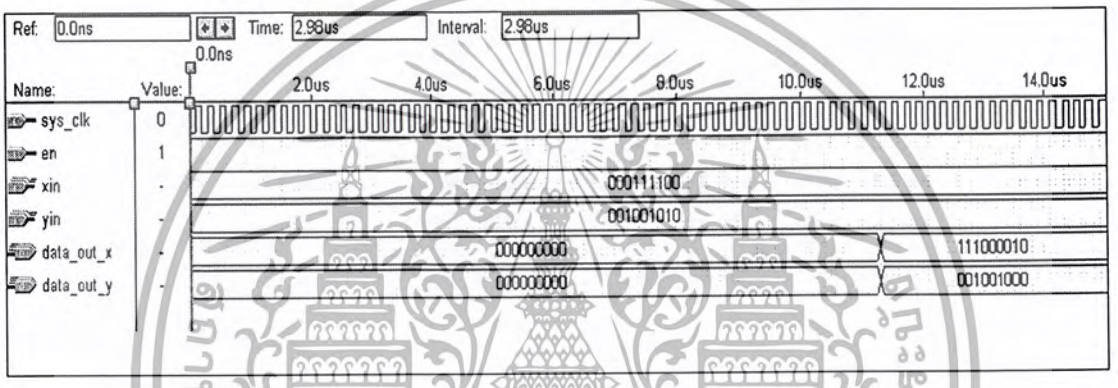
รูปที่ 4.21 ผลการจำลองการทำงานของวงจรลบ 9 บิท

4.5.5 ส่วนของวงจร DA_CORDIC1 ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ ทำหน้าที่ในการคูณข้อมูลขนาด 8 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.22



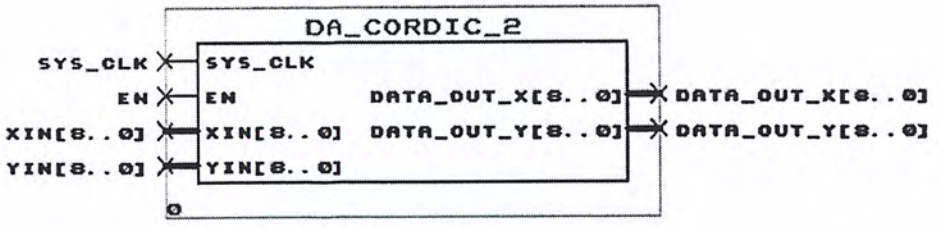
รูปที่ 4.22 สัญลักษณ์ของส่วนของวงจร DA_CORDIC1

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



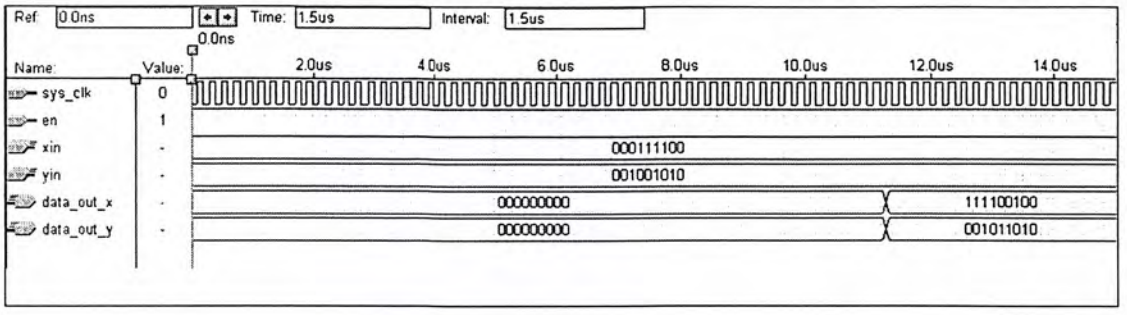
รูปที่ 4.23 ผลการจำลองการทำงานของวงจร DA_CORDIC1

4.5.6 ส่วนของวงจร DA_CORDIC2 ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ ทำหน้าที่ในการคูณข้อมูลขนาด 8 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.24



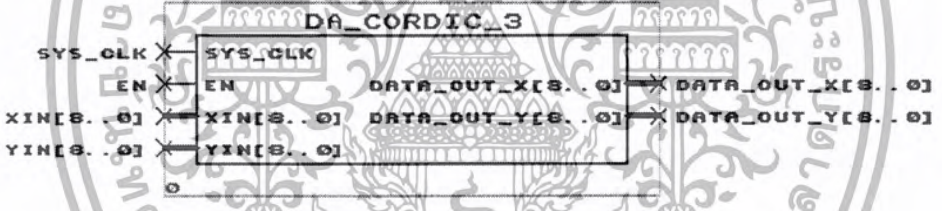
รูปที่ 4.24 สัญลักษณ์ของส่วนของวงจร DA_CORDIC2

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



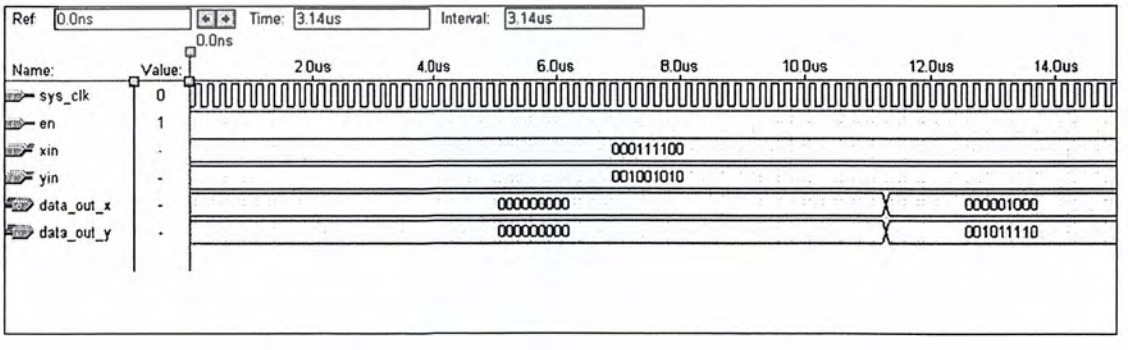
รูปที่ 4.25 ผลการจำลองการทำงานของวงจร DA_CORDIC2

4.5.7 ส่วนของวงจร **DA_CORDIC3** ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ ทำหน้าที่ในการคูณข้อมูลขนาด 8 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.26



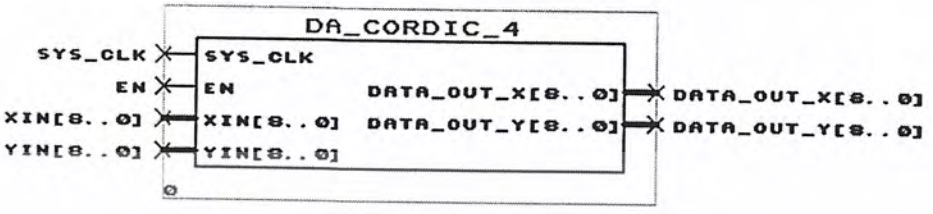
รูปที่ 4.26 สัญลักษณ์ของส่วนของวงจร DA_CORDIC3

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



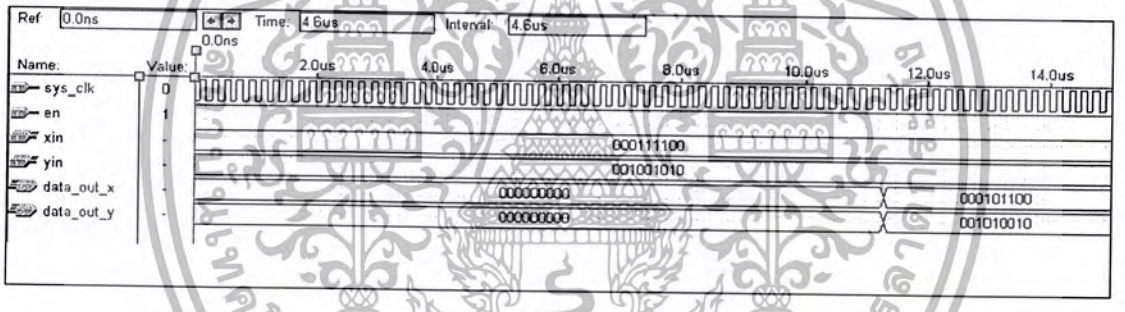
รูปที่ 4.27 ผลการจำลองการทำงานของวงจร DA_CORDIC3

4.5.8 ส่วนของวงจร DA_CORDIC4 ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ ทำหน้าที่ในการคูณข้อมูลขนาด 8 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.28



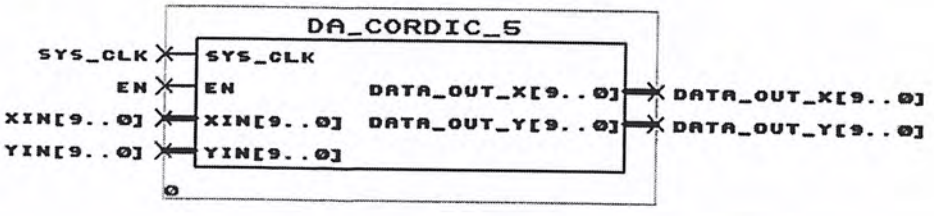
รูปที่ 4.28 สัญลักษณ์ของส่วนของวงจร DA_CORDIC4

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



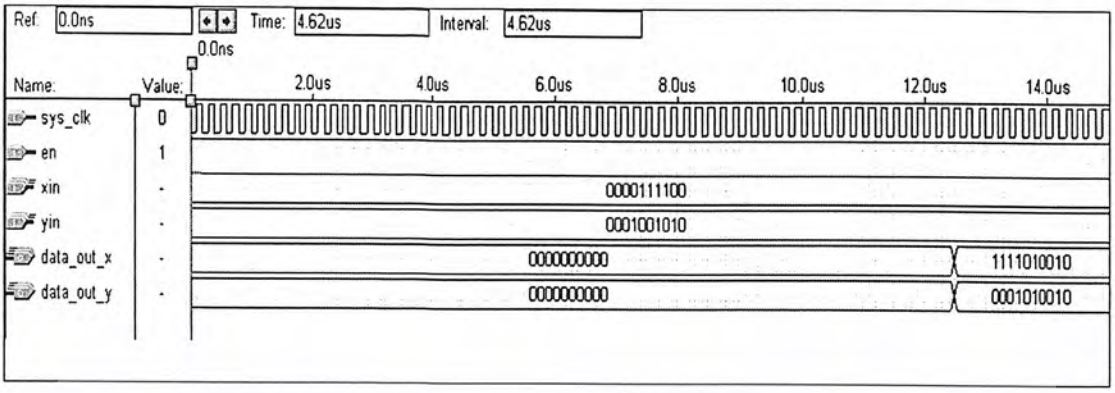
รูปที่ 4.29 ผลการจำลองการทำงานของวงจร DA_CORDIC4

4.5.9 ส่วนของวงจร DA_CORDIC5 ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ ทำหน้าที่ในการคูณข้อมูลขนาด 9 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.30



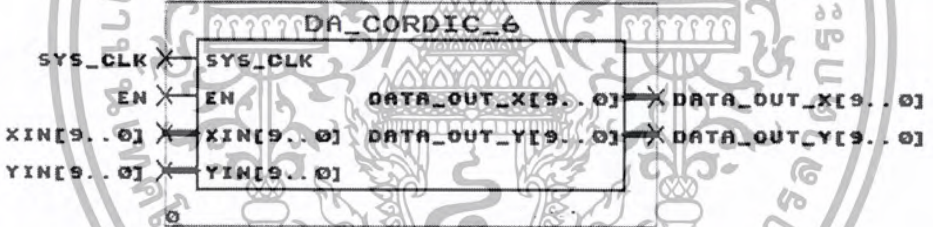
รูปที่ 4.30 สัญลักษณ์ของส่วนของวงจร DA_CORDIC5

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



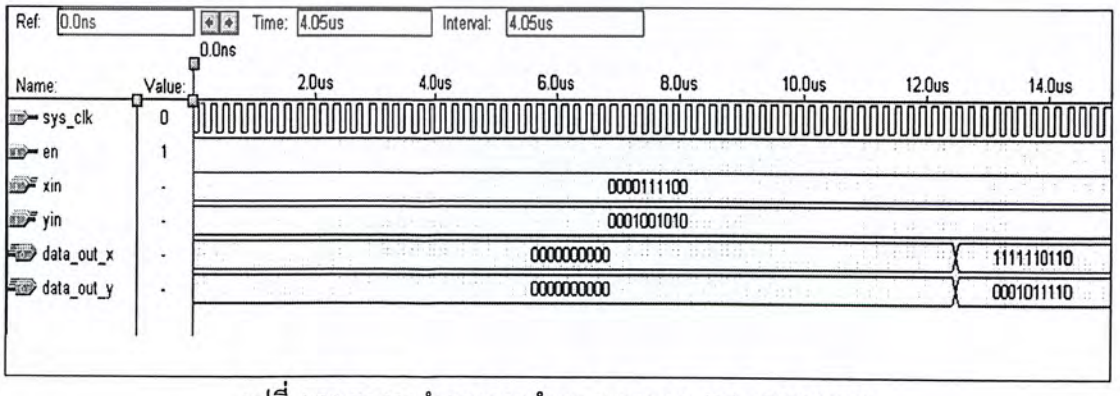
รูปที่ 4.31 ผลการจำลองการทำงานของวงจร DA_CORDIC5

4.5.10 ส่วนของวงจร DA_CORDIC6 ซึ่งใช้โครงสร้างเลขคณิตกระจาย (Distributed Arithmetic) เข้ามาช่วยในการคำนวณ ทำหน้าที่ในการคูณข้อมูลขนาด 9 บิท ตามโครงสร้างการแปลง DCT ที่ได้กล่าวไปแล้วในบทที่ 3 สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.32



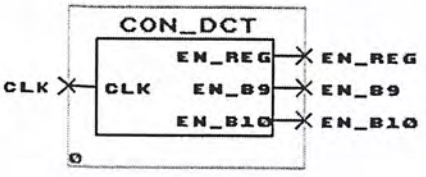
รูปที่ 4.32 สัญลักษณ์ของส่วนของวงจร DA_CORDIC6

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



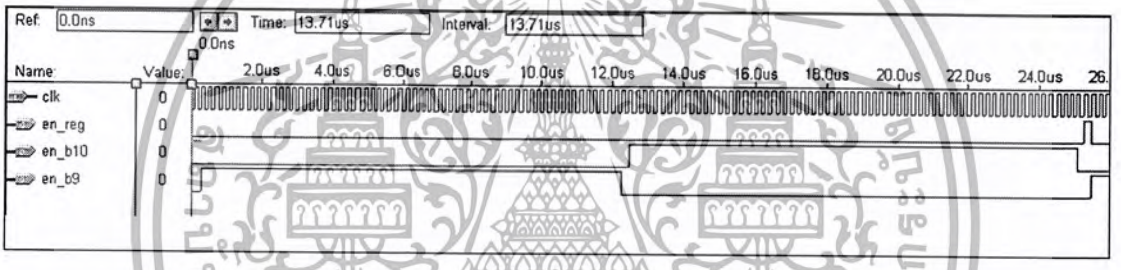
รูปที่ 4.33 ผลการจำลองการทำงานของวงจร DA_CORDIC6

4.5.11 ส่วนของวงจรควบคุม (control) ทำหน้าที่ในการจัดลำดับการทำงานของ แต่ละวงจร เพื่อให้สัญญาณเอาท์พุทที่ได้จากการแปลง DCT มีความถูกต้อง สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.34



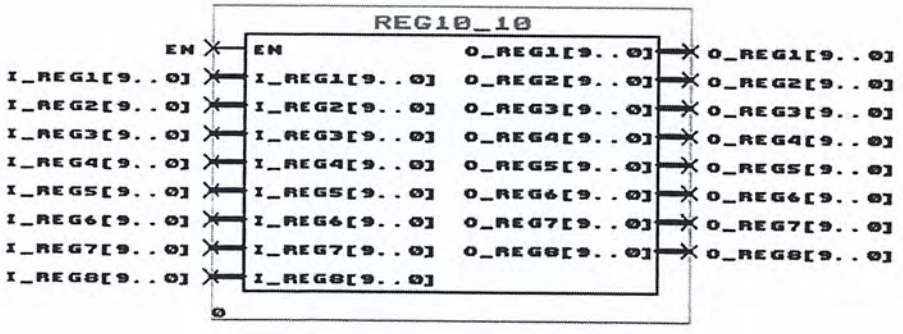
รูปที่ 4.34 สัญลักษณ์ของส่วนของวงจรควบคุม

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



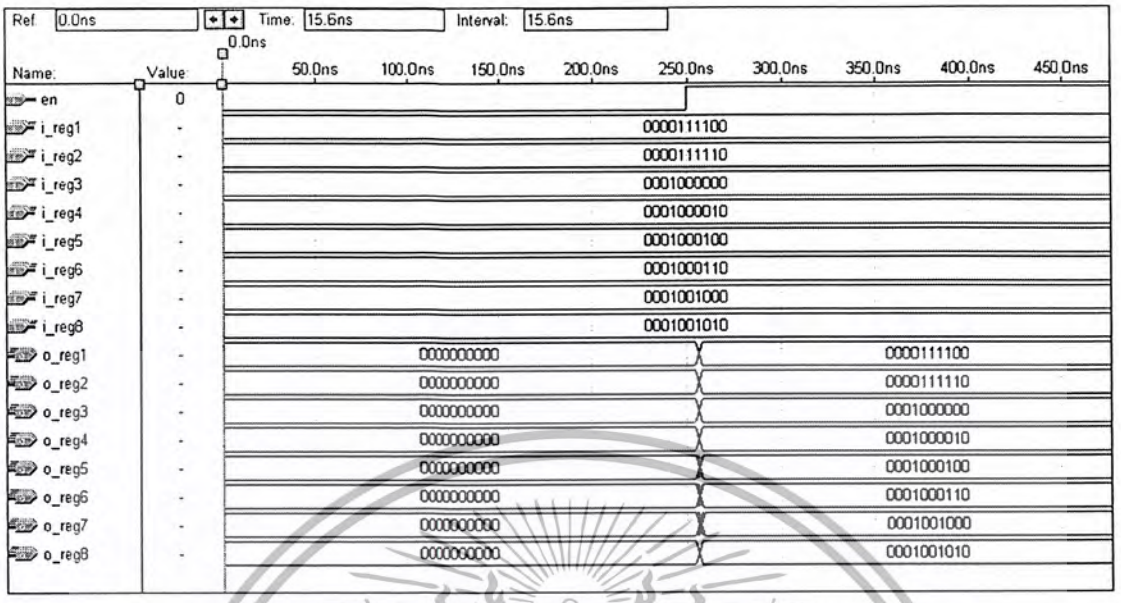
รูปที่ 4.35 ผลการจำลองการทำงานของส่วนของวงจรควบคุม

4.5.12 ส่วนของวงจรรีจีสเตอร์ขนาด 10 บิท (Register) ทำหน้าที่เก็บข้อมูลที่ได้จากการแปลง DCT เนื่องจากในบางกรณีค่าจากการแปลงจะส่งเอาท์พุทออกมาไม่พร้อมกัน จะเป็นรีจีสเตอร์ขนาด 10 บิท10 ชุด เมื่อเก็บข้อมูลครบ 10 ชุด จะส่งต่อให้วงจรมัลติเพล็ก จะทำงานซ้ำแบบนี้จนกระทั่งไม่มีการแปลงเกิดขึ้น สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.36



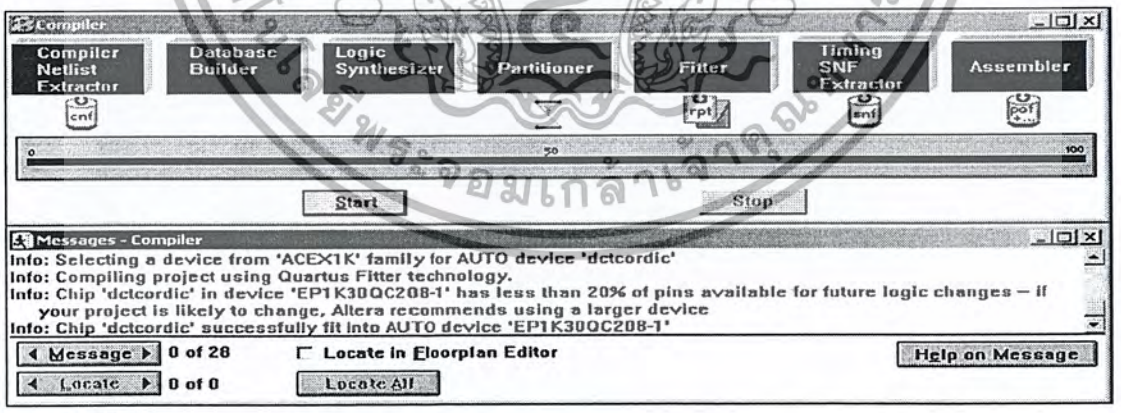
รูปที่ 4.36 สัญลักษณ์ของส่วนของวงจรรีจีสเตอร์ขนาด 10 บิท

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



รูปที่ 4.37 ผลการจำลองการทำงานของส่วนของวงจรรีจิสเตอร์ขนาด 10 บิท

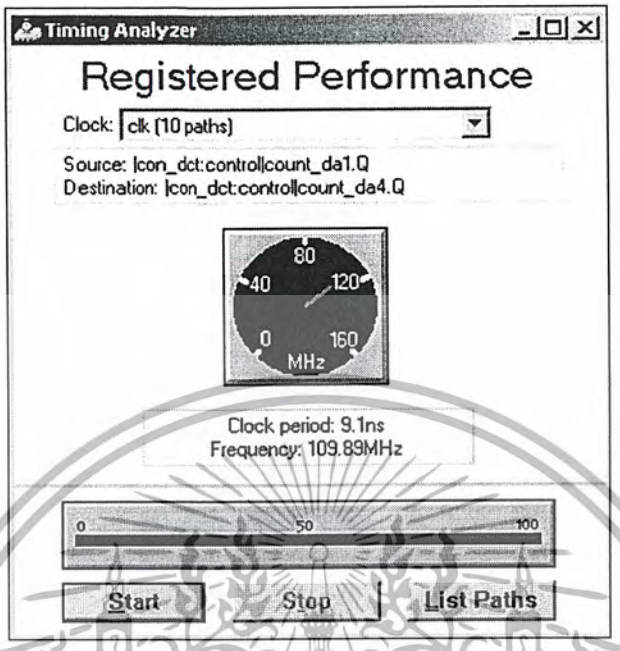
จากส่วนของการแปลง DCT ที่อธิบายมาข้างต้นสามารถแสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราะห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน ดังรูปที่ 4.38




```
DCTCORDIC

** DEVICE SUMMARY **

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

dctcordic
      EP1K30QC208-1                 65     80      0      0          0 %    1258      72 %

User Pins:                          65     80      0
```

รูปที่ 4.38 แสดงการคอมไพล์และแสดงรายละเอียดของการใช้อุปกรณ์

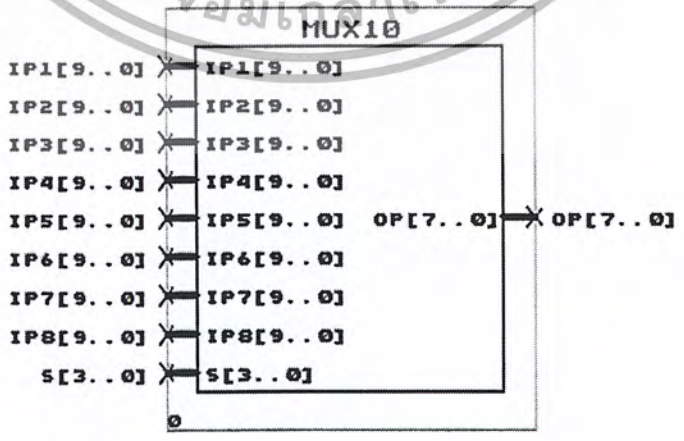
การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปที่ 4.39



รูปที่ 4.39 แสดงค่าเวลาและความถี่ที่เป็นข้อกำหนดของวงจร

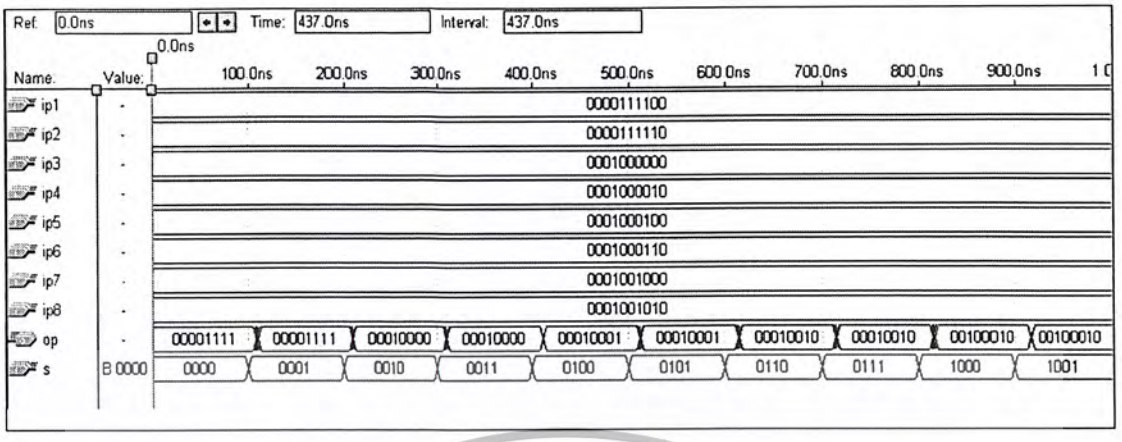
4.6 ส่วนของวงจรมัลติเพล็ก (MUX10)

ส่วนของวงจรมัลติเพล็ก (MUX10) ทำหน้าที่ในการตัดบิทให้เป็น 8 บิทจำนวน 10 ชุดเพื่อเหตุผลในการส่งข้อมูลไปยังคอมพิวเตอร์ทำได้ครั้งละ 8 บิท และ ใช้ในการส่งออกไปยังคอมพิวเตอร์ให้เป็นไปตามลำดับ สามารถเขียนโปรแกรมที่สามารถสังเคราะห์เป็นอุปกรณ์ที่มีสัญลักษณ์ (Symbol) ได้มีลักษณะดังรูปที่ 4.40



รูปที่ 4.40 สัญลักษณ์ของส่วนของวงจรมัลติเพล็ก

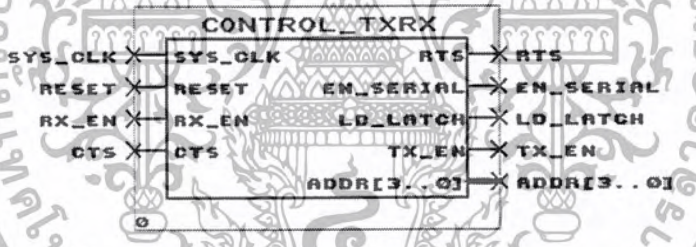
จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



รูปที่ 4.41 ผลการจำลองการทำงานของส่วนของวงจรมัลติเพล็ก

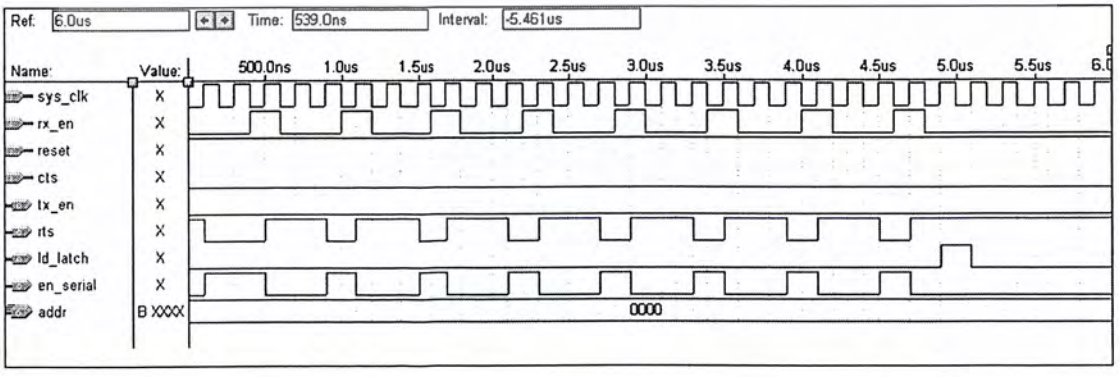
4.7 ส่วนของวงจรควบคุม (control)

ส่วนของวงจรควบคุม (control) ทำหน้าที่ในการควบคุมและจัดลำดับการทำงานของ แต่ละวงจรให้ถูกต้องตามกระบวนการที่ได้กำหนดไว้ มีลักษณะดังรูปที่ 4.42



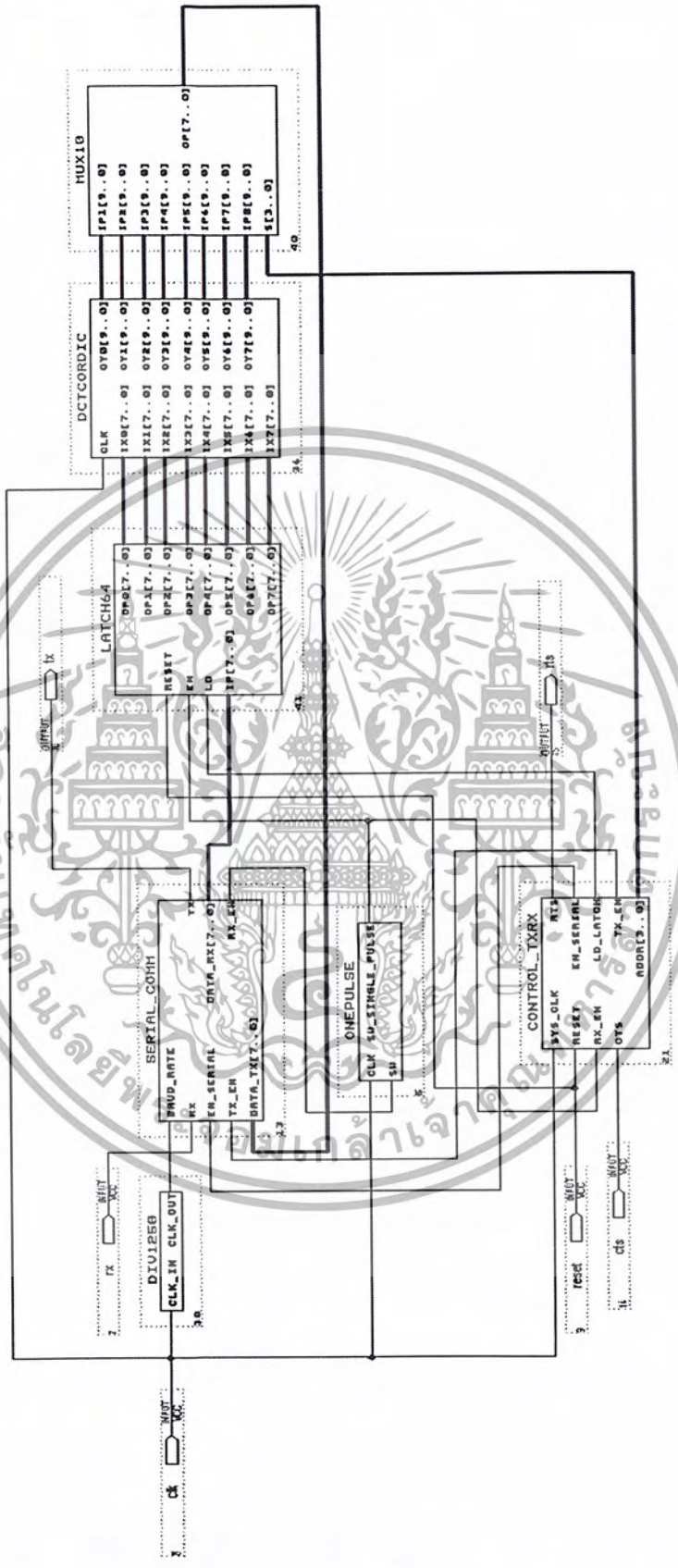
รูปที่ 4.42 สัญลักษณ์ของส่วนของวงจรควบคุม

จากโปรแกรมที่เขียนขึ้นสามารถทำการจำลองการทำงาน (Simulation) ได้ดังนี้



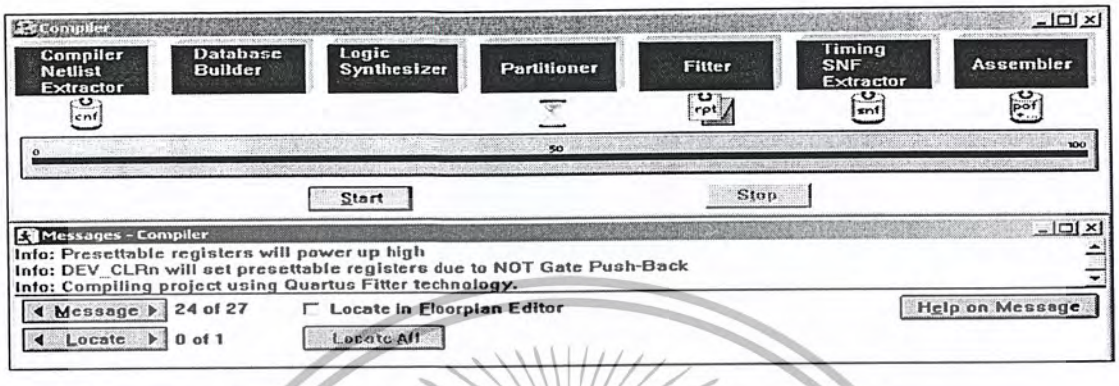
รูปที่ 4.43 ผลการจำลองการทำงานของส่วนของวงจรควบคุม

4.8 ส่วนประกอบภายในและการเชื่อมต่อของวงจรทั้งหมด มีดังนี้



รูปที่ 4.44 ส่วนประกอบภายในและการเชื่อมต่อของวงจรทั้งหมด

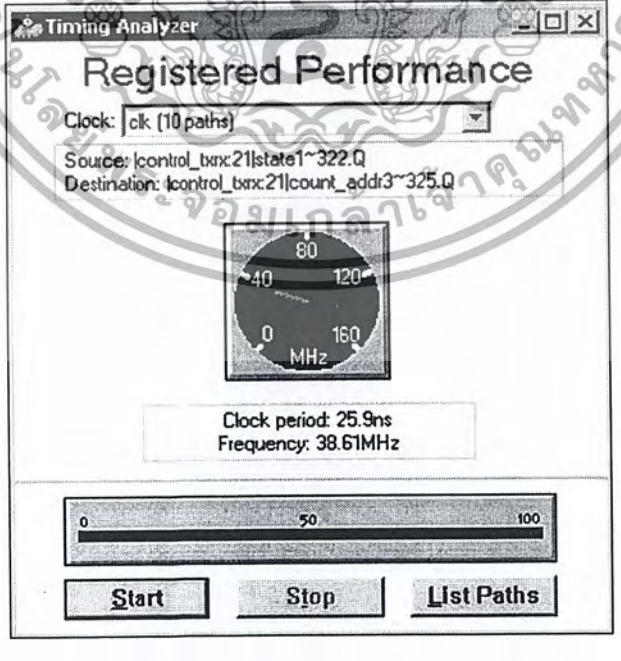
จากส่วนส่วนประกอบภายในและการเชื่อมต่อของวงจรทั้งหมด ข้างต้นสามารถแสดงการคอมไพล์วงจรที่ออกแบบขึ้นมาเพื่อให้ซอร์ฟแวร์ที่ใช้ในการสังเคราะห์แสดงค่าอุปกรณ์ที่สามารถรองรับการทำงานและรายละเอียดต่างๆ ของสิ่งที่เราออกแบบขนาดของเกทที่เราใช้งานโดยแสดงออกมาเป็นเปอร์เซ็นต์การใช้งาน ดังรูปที่ 4.45




```
** DEVICE SUMMARY **

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

dctcordic_check
       EP1K50TC144-3           4      2       0       0        0  %      1715      59 %

User Pins:                     4      2       0
```

รูปที่ 4.45 การคอมไพล์และแสดงรายละเอียดการใช้อุปกรณ์

การสังเคราะห์วงจรที่เราออกแบบมานั้นจะมีข้อกำหนดทางด้านความถี่ที่นำมาใช้งานและระดับความเร็วของสัญญาณนาฬิกาซึ่งซอร์ฟแวร์ที่นำมาสังเคราะห์นั้นจะบอกรายระเอียดแสดงได้ดังรูปที่ 4.46

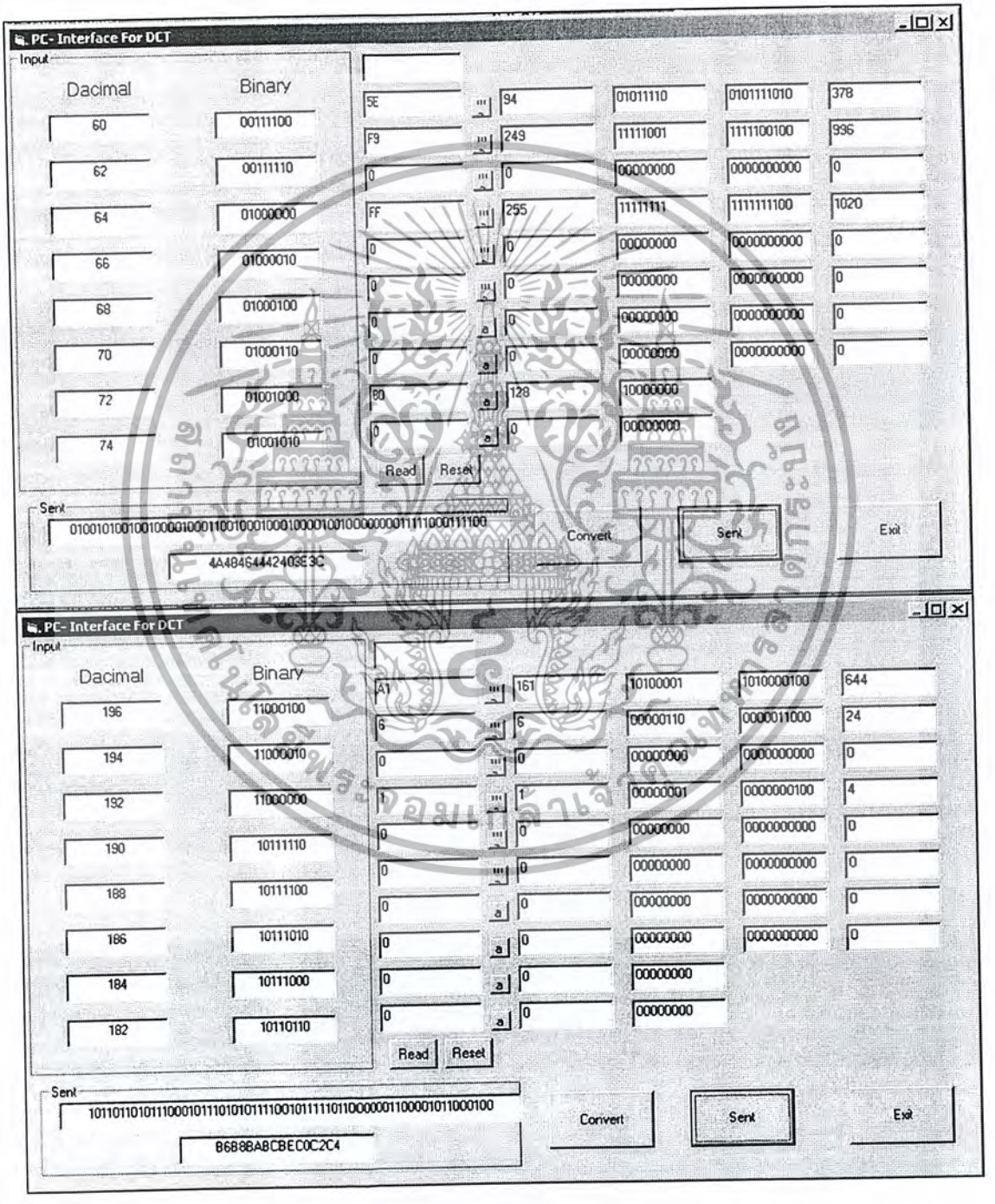


รูปที่ 4.46 ค่าเวลาและความถี่ที่เป็นข้อกำหนดของวงจร

4.9 ขั้นตอนการเก็บผลการทดสอบ Hard ware ด้วยชุดข้อมูลที่มีความสัมพันธ์กัน

1. ทำการกำหนดค่าสัญญาณอินพุทที่มีระยะห่างเท่ากันเพื่อให้สัญญาณมีความสัมพันธ์กันในการทดสอบนี้ใช้ค่า (60, 62, 64, 66, 68, 70, 72, 74) และ (-60,-62,-64,-66,-68,-70,-72,-74)

2. ทำการป้อนค่าสัญญาณ โดยผ่าน โปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของHard ware ที่เขียนด้วย Visual Basic ซึ่งถ้าค่าสัญญาณอินพุทเป็นค่าลบจะต้องทำการถ่วงน้ำหนักค่าให้เป็นค่าบวกที่อยู่ในช่วง 0 ถึง 255 เช่น ค่า -60 จะเท่ากับ 196 (หรือ 256 – 60) แสดงดังรูปที่ 4.47



รูปที่ 4.47 ป้อนค่าสัญญาณ โดยผ่านโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware

3. ซึ่งจากการทำงานของ Hard ware จะส่งค่ากลับมายังโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware จำนวน 10 ไบท์ เนื่องจากค่าที่ผ่านการแปลง DCT แล้วจะมีค่าๆละ 10 บิทจำนวน 8 ค่า (8 X 10 = 80 บิท) และค่าที่ส่งกลับมาจาก Hard ware จะเป็นเลขฐาน 16 ชุดละ 1 ไบท์จำนวน 10 ชุด ดังนั้นจึงต้องแปลงเป็นเลขฐาน 10 แล้วทำการแปลงเป็นเลขฐาน 2 จากนั้นจะทำการต่อบิทให้ได้ 10 บิท จำนวน 8 ชุดจะต่อบิทดังนี้

ข้อมูลที่ต่อบิท ชุดที่	ข้อมูลชุดที่+ข้อมูลชุดที่(บิทที่)	ข้อมูลที่ต่อบิท ชุดที่	ข้อมูลชุดที่+ข้อมูลชุดที่(บิทที่)
1	1 + 9(1,2)	5	1 + 10(1,2)
2	2 + 9(3,4)	6	2 + 10(3,4)
3	3 + 9(5,6)	7	3 + 10(5,6)
4	4 + 9(7,9)	8	4 + 10(7,8)

ตารางที่ 4.1 ผลของการการต่อบิทให้ได้ 10 บิทจำนวน 8 ชุด

ตามที่ Hard ware ประมวลผลแล้วจึงแปลงเป็นเลขฐาน 10 ซึ่งการทำงานทั้งหมดจะทำงานจากโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware

4. จะนำข้อมูลที่ได้มาถ่วงน้ำหนักค่ากลับคืนให้อยู่ในช่วงค่าบวกและลบ (-512 ถึง 511) เนื่องจากค่าที่ได้จาก Hard ware จะได้เฉพาะค่าที่อยู่ในช่วงบวก

5. แล้วนำข้อมูลที่ถ่วงน้ำหนักกลับคืนแล้วมาแปลงอินเวิร์ด DCT กลับด้วยโปรแกรม MATLAB เพื่อตรวจสอบความถูกต้องของการแปลง DCT ที่ได้จาก Hard ware

ผลที่ได้จากการทำงานของ Hardware การแปลง DCT โดยใช้ค่า (60, 62, 64, 66, 68, 70, 72, 74)

อินพุท	เอาท์พุท ฐาน 16	เอาท์พุท ฐาน 10	เอาท์พุท ฐาน 2	ต่อบิท	เอาท์พุท ฐาน 10	เอาท์พุทการ ถ่วงน.น.	IDCTจาก MATLAB
60	5E	94	01011110	0101111010	378	378	59.1205
62	F9	249	11111001	1111100100	996	-28	61.1895
64	0	0	00000000	0000000000	0	0	63.9065
66	FF	255	11111111	1111111100	1020	-4	66.0015
68	0	0	00000000	0000000000	0	0	67.6215

ตารางที่ 4.2 ผลที่ได้จากการทำงานของ Hard ware การแปลง DCT ใช้ค่า (60, 62, 64, 66, 68)

อินพุท	เอาท์พุท ฐาน 16	เอาท์พุท ฐาน 10	เอาท์พุท ฐาน 2	ต่อบิท	เอาท์พุท ฐาน 10	เอาท์พุทการถ่วงน.น.	IDCTจาก MATLAB
70	0	0	00000000	0000000000	0	0	69.7165
72	0	0	00000000	0000000000	0	0	72.4335
74	0	0	00000000	0000000000	0	0	74.5025
-	80	128	10000000	-	-	-	-
-	0	0	00000000	-	-	-	-

ตารางที่ 4.2 (ต่อ)ผลที่ได้จากการทำงานของ Hard ware การแปลง DCT ใช้ค่า (70, 72, 74)

ผลที่ได้จากการทำงานของ Hard ware การแปลง DCT โดยใช้ค่า (-60,-62,-64,-66,-68,-70,-72,-74)

อินพุท	เอาท์พุท ฐาน 16	เอาท์พุท ฐาน 10	เอาท์พุท ฐาน 2	ต่อบิท	เอาท์พุท ฐาน 10	เอาท์พุทการถ่วงน.น.	IDCTจาก MATLAB
196	A1	161	10100001	1010000100	644	-380	-60.454
194	6	6	00000110	0000011000	24	24	-62.374
192	0	0	00000000	0000000000	0	0	-64.815
190	1	1	00000001	0000000100	4	4	-66.550
188	0	0	00000000	0000000000	0	0	-67.780
186	0	0	00000000	0000000000	0	0	-69.515
184	0	0	00000000	0000000000	0	0	-71.956
182	0	0	00000000	0000000000	0	0	-73.876
-	0	0	00000000	-	-	-	-
-	0	0	00000000	-	-	-	-

ตารางที่ 4.3 ผลที่ได้จากการทำงานของ Hard ware การแปลง DCT ใช้ค่า (-60,-62,-64,-66,-68,-70,-72,-74)

เปรียบเทียบค่าการแปลง DCT ได้จาก Hard ware กับ ผลที่ได้จากโปรแกรม MATLAB

การเปรียบเทียบในที่นี้จะพิจารณาในกรณีของจำนวนเต็มเนื่องจากในการออกแบบการแปลง DCT ในโปรแกรม MAX+plus2 นั้นถ้าใช้จำนวนบิทน้อยเท่าไรนั้นจะทำให้ใช้ปริมาณเนื้อที่น้องลงเท่านั้น ในที่นี้จะสมมุติค่าที่ใช้ในการเปรียบเทียบผล 2 ชุด โดยชุดแรกมีค่าเป็นบวกและชุดที่สองมีค่าเป็นลบในการทดสอบจะกำหนดค่าที่มีความสัมพันธ์กันโดยให้มีช่วงลำดับในการเพิ่มหรือลดเท่ากันในที่นี้ค่าชุดแรกจะสมมุติให้เป็น 60, 62, 64, 66, 68, 70, 72, 74

ค่าอินพุท	ค่าการแปลง DCT จาก Hardware	ค่าการแปลง DCT จากโปรแกรม MATLAB
IX0 = 60	OY0 = 376	OY0 = 378.9520
IX1 = 62	OY1 = -26	OY1 = -25.7500
IX2 = 64	OY2 = 0	OY2 = 0
IX3 = 66	OY3 = -3	OY3 = -2.6940
IX4 = 68	OY4 = 0	OY4 = 0
IX5 = 70	OY5 = -1	OY5 = -0.8020
IX6 = 72	OY6 = 0	OY6 = 0
IX7 = 74	OY7 = -1	OY7 = -0.2060

ตารางที่ 4.4 ผลการเปรียบเทียบค่าการแปลง DCT ด้วยค่าชุดที่ 1

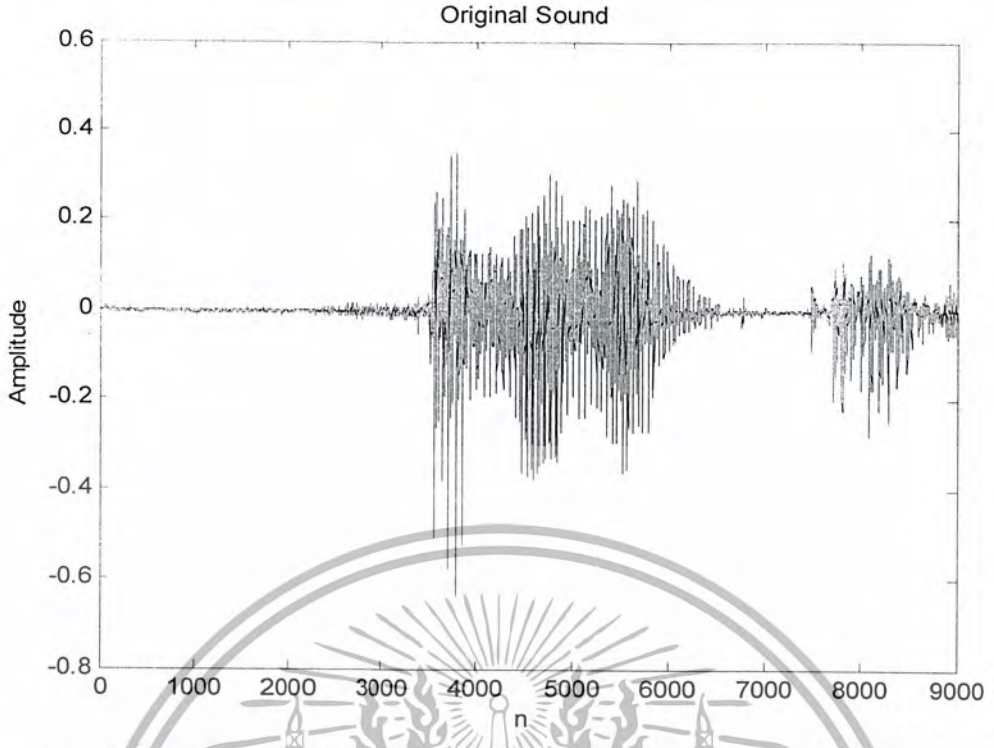
และชุดที่สองสมมุติให้เป็น -60,-62,-64,-66,-68,-70,-72,-74 ดังจะได้ผลการเปรียบเทียบในตาราง ที่ 4.4 และ 4.5

ค่าอินพุท	ค่าการแปลง DCT จาก Hardware	ค่าการแปลง DCT จากโปรแกรม MATLAB
IX0 = -60	OY0 = -377	OY0 = -378.9520
IX1 = -62	OY1 = 25	OY1 = 25.7500
IX2 = -64	OY2 = 0	OY2 = 0
IX3 = -66	OY3 = 2	OY3 = 2.6940
IX4 = -68	OY4 = 0	OY4 = 0
IX5 = -70	OY5 = 0	OY5 = 0.8020
IX6 = -72	OY6 = 0	OY6 = 0
IX7 = -74	OY7 = 0	OY7 = 0.2060

ตารางที่ 4.5 ผลการเปรียบเทียบค่าการแปลง DCT ด้วยค่าชุดที่ 2

4.10 ขั้นตอนการเก็บผลการทดสอบ Hard ware ด้วยความถี่เสียง

1. สัญญาณเสียงที่นำมาทดสอบมีชื่อว่า Bluetooth input.wav ซึ่งมีลักษณะสัญญาณดังรูปที่ 4.48



รูปที่ 4.48 สัญญาณเสียงที่นำมาทดสอบมีความถี่แซมปลิ้ง 8000 เฮิร์ตจำนวน9000 จุด

2. ทำการป้อนค่าสัญญาณเสียงที่กำเนิดค่าของสัญญาณมาจากโปรแกรม MATLAB ผ่านโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware ที่เขียนด้วยโปรแกรม Visual Basic ซึ่งค่าสัญญาณอินพุทจะต้องทำการจัดค่าให้เป็นค่าบวกที่อยู่ในช่วง 0 ถึง 255 ซึ่งการจัดค่าจะทำการประมวลผลด้วยโปรแกรม MATLAB และข้อมูลที่ได้กลับมาจาก Hard ware ซึ่งจะถูกเก็บเป็นไฟล์ตัวอักษร (output.txt) ซึ่งจะเป็นค่าข้อมูลที่ได้ทำการแปลง DCT แล้วจะอยู่ในรูปของข้อมูลที่เป็นไบนารี่ ดังนั้นเราจึงต้องนำข้อมูลไบนารี่ที่ได้จาก Hard ware มาทำการประมวลผลและทำการแปลงกลับแบบ DCT ด้วยโปรแกรม MATLAB แล้วจึงนำค่าที่ได้มาเปรียบเทียบกับข้อมูลอินพุทที่เป็นสัญญาณเสียงต้นฉบับมีขั้นตอนคือ

2.1 เปลี่ยนสัญญาณเสียงที่ได้จากการจัดค่าให้อยู่ในช่วง 0-255 ได้โดยมีขั้นตอนคือ

2.1.1 ถ้ามีค่าไหนที่ติดลบให้นำมาบวกกับ 2

2.1.2 นำค่าทั้งหมดมาคูณกับ 128

2.2 ค่าจากการแปลง DCT ที่ใช้หลักการของ CORDIC Algorithm ที่ทำการจัดค่าโดยมีขั้นตอนคือ

2.2.1 นำค่าทั้งหมดมาหารด้วย 512

2.2.2 ถ้ามีค่าไหนที่เกิน 1 ให้ทำมาลบออกด้วย 2

2.2.3 แล้วนำค่าทั้งหมดมาคูณด้วย 4 เพื่อทำการให้ระดับของสัญญาณสูงขึ้น

3. ซึ่งจากการทำงานของ Hard ware จะส่งค่ากลับมายังโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware จำนวน 10 ไบท์ เนื่องจากค่าที่ผ่านการแปลง DCT แล้วจะมีค่าๆละ 10 บิทจำนวน 8

ชุด (8 X 10 = 80 บิท) และค่าที่ส่งกลับมาจาก Hard ware จะเป็นเลขฐาน 16 ชุดละ 1 ไบท์จำนวน 10 ชุด ดังนั้นจึงต้องแปลงเป็นเลขฐาน 10 แล้วทำการแปลงเป็นเลขฐาน 2 จากนั้นจะทำการต่อบิทให้ได้ 10 บิท จำนวน 8 ชุดจะต่อ ซึ่งจากสัญญาณเสียงที่ทำมาทดสอบ (bluetooth_ input.wav) มีจำนวน 9000 จุด ซึ่งการทำงานของโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบจะต้องทำการส่งค่าทั้งหมดจำนวน 1125 ครั้ง ซึ่งเมื่อทำการประมวลผลเสร็จเรียบร้อยแล้วเราจะสามารถหาค่าความผิดพลาดที่เกิดจากการแปลงแบบ DCT และ ความผิดพลาดที่เกิดจากการแปลงกลับแบบ DCT

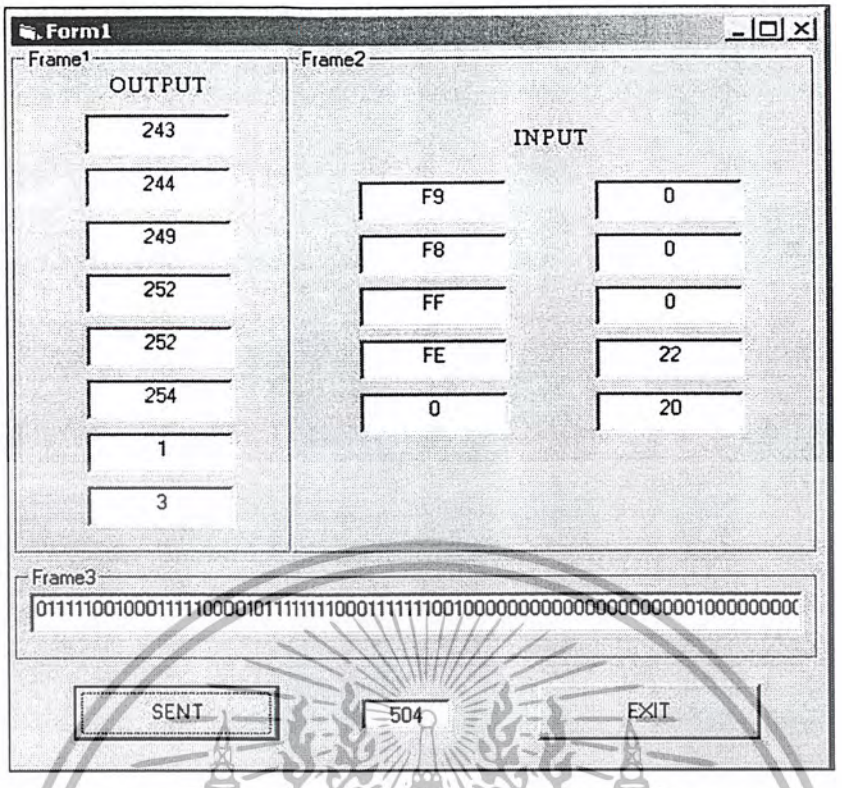
ข้อมูลที่ต่อบิท ชุดที่	ข้อมูลชุดที่+ข้อมูลชุดที่(บิทที่)	ข้อมูลที่ต่อบิท ชุดที่	ข้อมูลชุดที่+ข้อมูลชุดที่(บิทที่)
1	1 + 9(1,2)	5	1 + 10(1,2)
2	2 + 9(3,4)	6	2 + 10(3,4)
3	3 + 9(5,6)	7	3 + 10(5,6)
4	4 + 9(7,9)	8	4 + 10(7,8)

ตารางที่ 4.6 ผลของการการต่อบิทให้ได้ 10 บิทจำนวน 8 ชุด

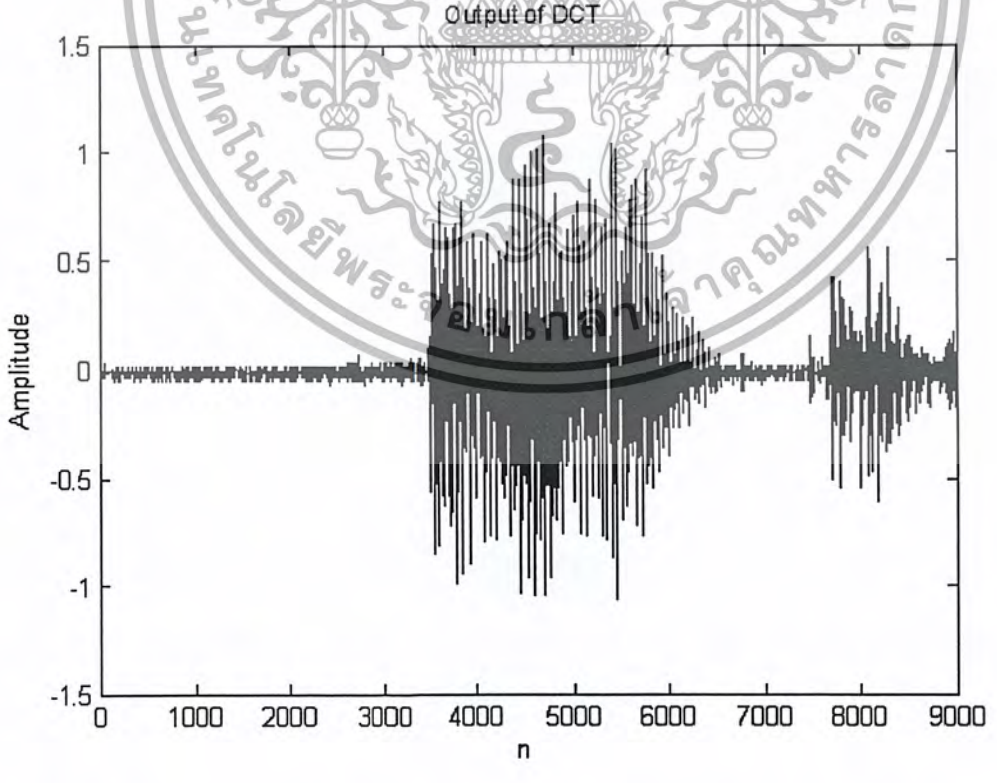
ตัวอย่างการถ่วงจัดค่าสัญญาณ

สัญญาณเสียง	สัญญาณเสียงที่ทำการจัดค่าให้อยู่ในช่วง 0-255	ค่าที่ได้จากการแปลงด้วย DCT ที่ใช้หลักการของ CORDIC Algorithm	ค่าที่ได้จากการแปลงด้วย DCT ที่ใช้หลักการของ CORDIC Algorithm ที่ทำการจัดค่ากับมา	สัญญาณเสียงที่ได้จากการแปลงกลับแบบ DCT (MATLAB)
-0.1016	243	996	-0.2188	-0.1109
-0.0938	244	994	-0.2344	-0.0919
-0.0547	249	1020	-0.0313	-0.0560
-0.0313	252	1018	-0.0469	-0.0331
-0.0313	252	0	0	-0.0298
-0.0156	254	2	0.0156	-0.0154
0.0078	1	0	0	0.0086
0.0234	3	0	0	0.0191

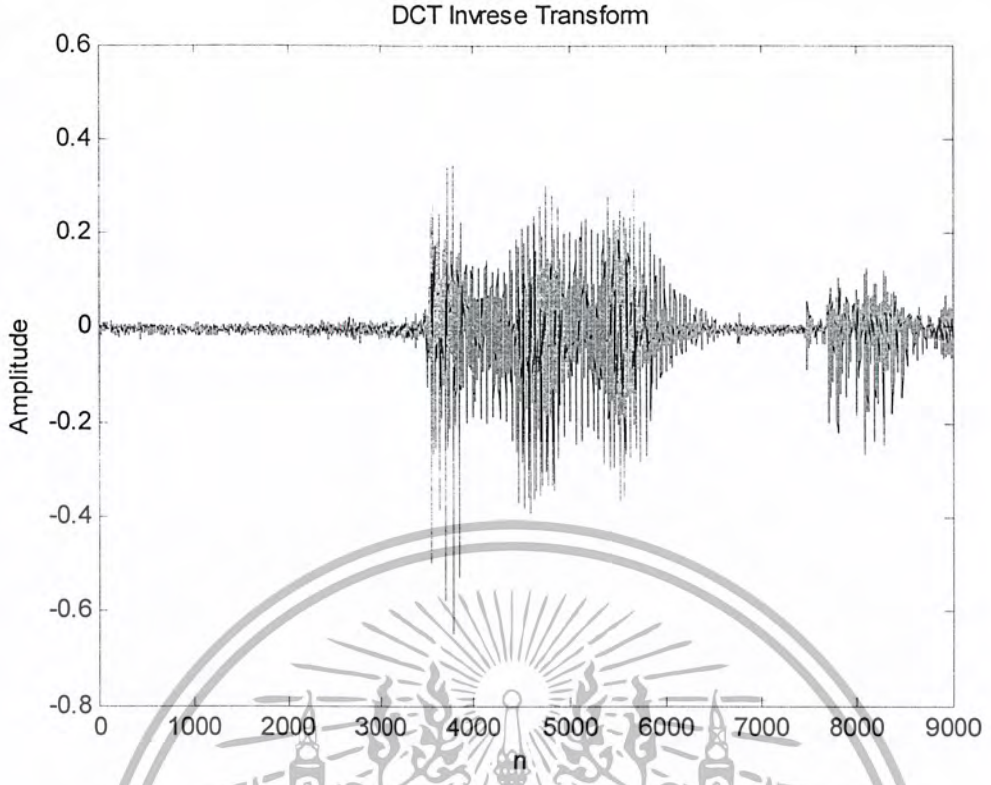
ตารางที่ 4.7 ตัวอย่างการการถ่วงน้ำหนักค่า



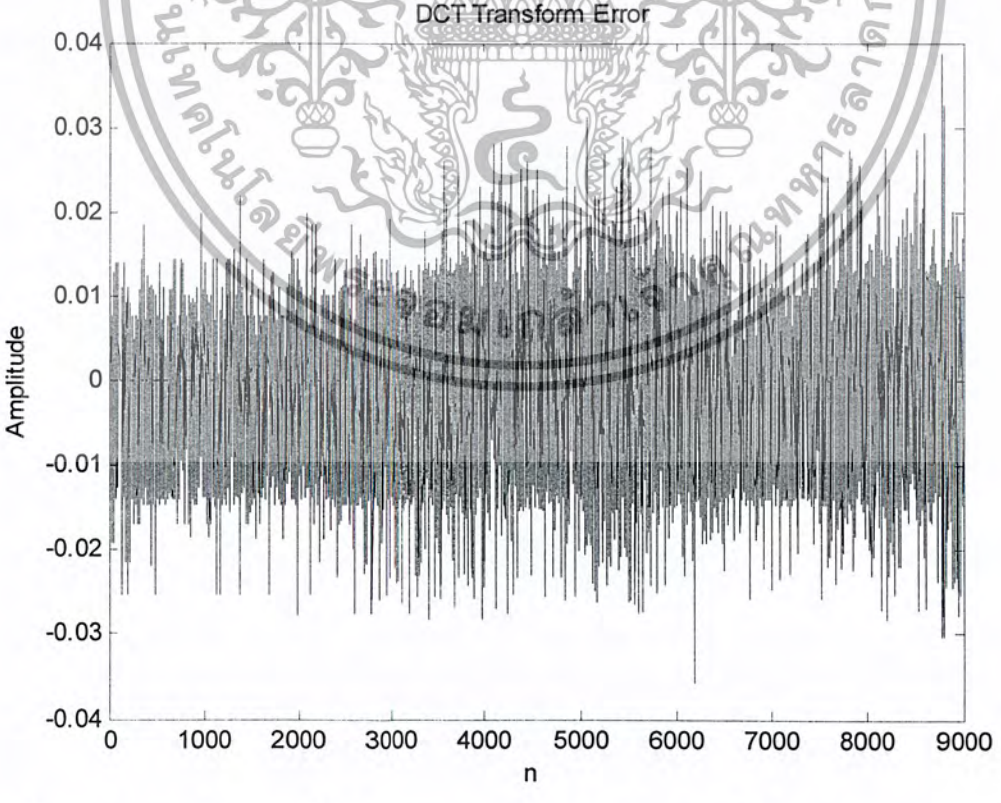
รูปที่ 4.49 ป้อนค่าสัญญาณโดยผ่านโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware



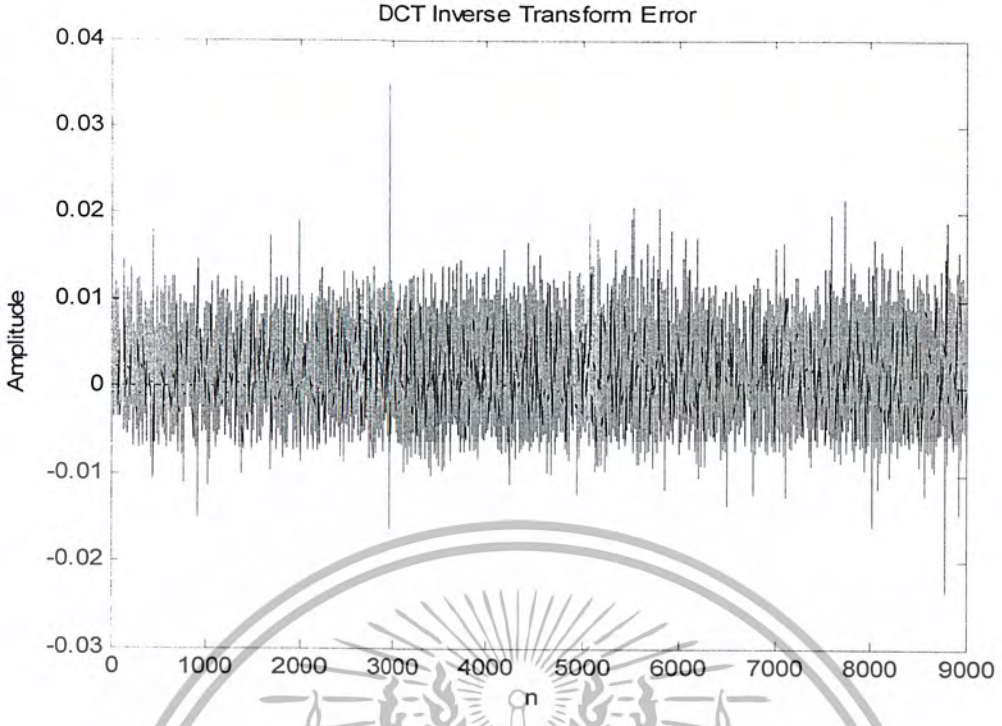
รูปที่ 4.50 สัญญาณจากการแปลงด้วย DCT ที่ใช้หลักการของ CORDIC Algorithm ที่ทำการสเกลกับมา



รูปที่ 4.51 สัญญาณที่ได้จากการแปลงกลับแบบ DCT ด้วยโปรแกรม MATLAB



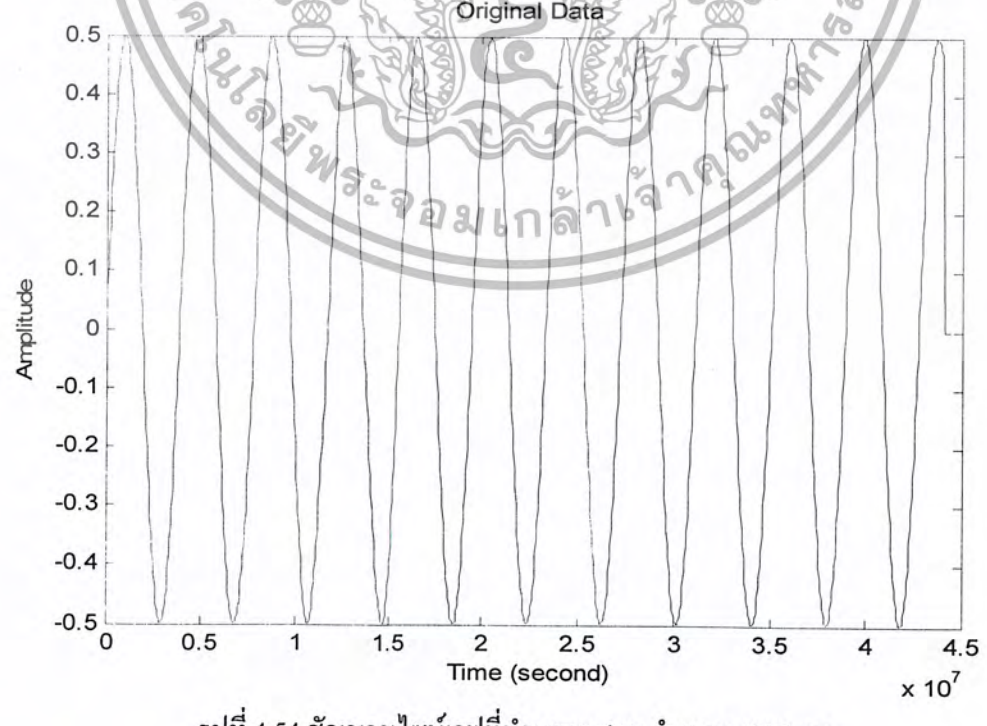
รูปที่ 4.52 สัญญาณค่าผิดพลาดที่ได้จากการแปลงแบบ DCT เทียบกับค่าที่ได้จากโปรแกรม MATLAB



รูปที่ 4.53 สัญญาณค่าผิดพลาดจากการแปลงกลับแบบ DCT เทียบกับค่าที่ได้จากโปรแกรม MATLAB

4.11 ขั้นตอนการเก็บผลการทดสอบ Hard ware ด้วยสัญญาณไซน์

1. สัญญาณเสียงที่นำมาทดสอบคือสัญญาณไซน์เวฟที่มีความถี่ 500 เฮิร์ต ขนาด 0.5 โวลต์ ความถี่แซมปลิ้ง 44100 เฮิร์ต โดยการกำเนิดสัญญาณมาจากโปรแกรม MATLAB ซึ่งมีลักษณะสัญญาณดังรูปที่ 4.54



รูปที่ 4.54 สัญญาณไซน์เวฟที่นำมาทดสอบ จำนวน 1008 จุด

2. ทำการป้อนค่าสัญญาณโดยผ่าน โปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware ที่เขียนด้วยโปรแกรม Visual Basic ซึ่งค่าสัญญาณอินพุทจะต้องทำการจัดค่าให้เป็นค่าบวกที่อยู่ในช่วง 0 ถึง 255 ซึ่งการจัดค่าจะทำการประมวลผลด้วยโปรแกรม MATLAB และข้อมูลที่ได้กลับมาจาก Hard ware ซึ่งจะถูกเก็บเป็นไฟล์ตัวอักษร (output.txt) ซึ่งจะเป็นค่าข้อมูลที่ได้ทำการแปลงแบบ DCT แล้วจะอยู่ในรูปของข้อมูลที่เป็นไบนารี่ ดังนั้นเราจึงต้องนำข้อมูลไบนารี่ที่ได้จาก Hardware มาทำการประมวลผลและทำการแปลงกลับแบบ DCT ด้วยโปรแกรม MATLAB แล้วจึงนำค่าที่ได้มาเปรียบเทียบกับข้อมูลอินพุทที่เป็นสัญญาณเสียงต้นฉบับ

2.1 เปลี่ยนสัญญาณเสียงที่ได้จากการจัดค่าให้อยู่ในช่วง 0-255 ได้โดยมีขั้นตอนคือ

2.1.1 ถ้ามีค่าไหนที่ติดลบให้นำมาบวกกับ 2

2.1.2 นำค่าทั้งหมดมาคูณกับ 128

2.2 ค่าจากการแปลง DCT ที่ใช้หลักการของ CORDIC Algorithm ที่ทำการจัดค่าโดยมีขั้นตอนคือ

2.2.1 นำค่าทั้งหมดมาหารด้วย 512

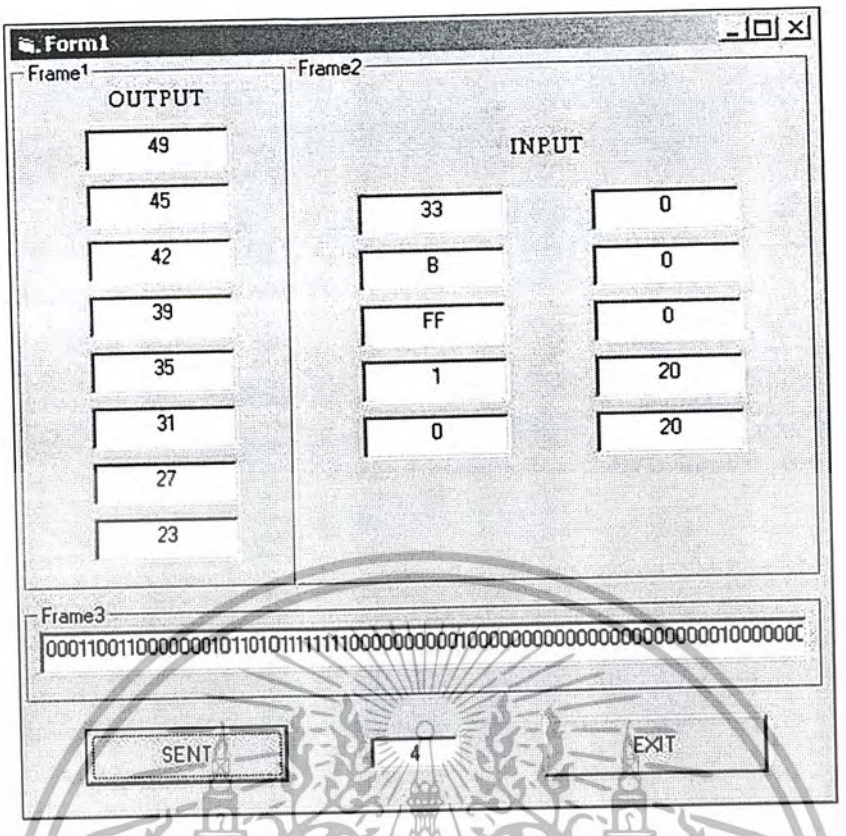
2.2.2 ถ้ามีค่าไหนที่เกิน 1 ให้ทำมาลบออกด้วย 2

2.2.3 แล้วนำค่าทั้งหมดมาคูณด้วย 4 เพื่อทำการให้ระดับของสัญญาณสูงขึ้น

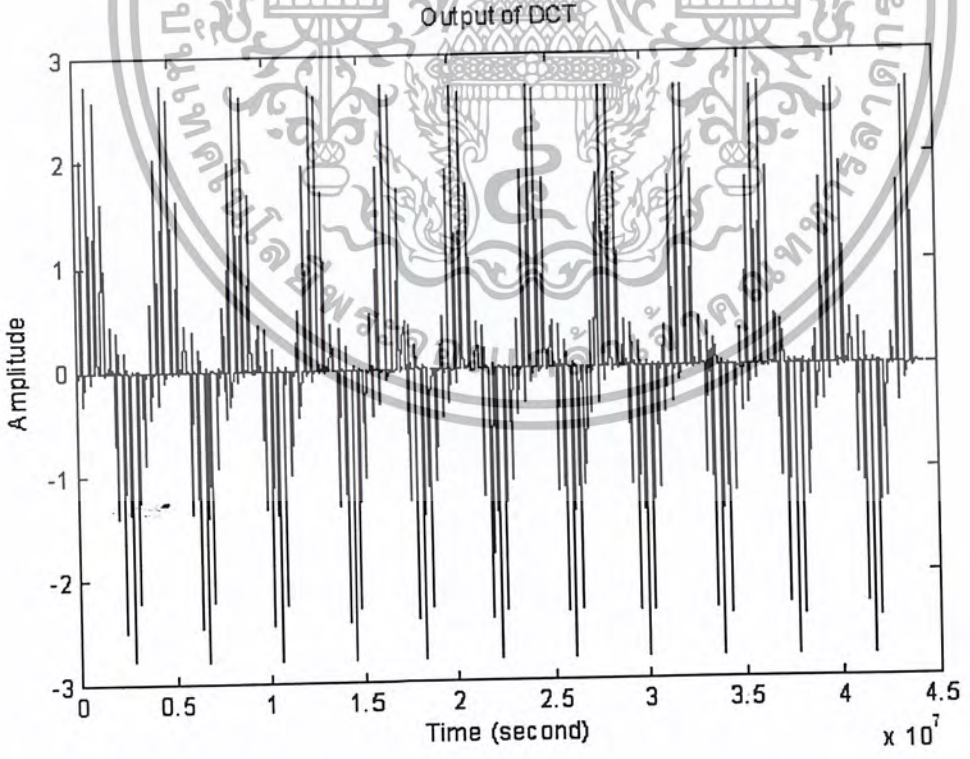
3. ซึ่งจากการทำงานของ Hard ware จะส่งค่ากลับมายังโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware จำนวน 10 ไบท์ เนื่องจากค่าที่ผ่านการแปลงDCTแล้วจะมีค่าๆละ 10 บิทจำนวน 8 ค่า (8 X 10 = 80 บิท) และค่าที่ส่งกลับมาจาก Hardware จะเป็นเลขฐาน 16 ชุดละ 1 ไบท์จำนวน 10 ชุด ดังนั้นจึงต้องแปลงเป็นเลขฐาน 10 แล้วทำการแปลงเป็นเลขฐาน 2 จากนั้นจะทำการต่อบิทให้ได้ 10 บิท จำนวน 8 ชุดจะต่อ ซึ่งจากสัญญาณไซน์เวฟที่นำมาทดสอบมีจำนวน 1008 จุด ซึ่งการทำงานของโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบจะต้องทำการส่งค่าทั้งหมดจำนวน 126 ครั้ง ซึ่งเมื่อทำการประมวลผลเสร็จเรียบร้อยแล้วเราจะสามารถหาค่าความผิดพลาดที่เกิดจากการแปลงแบบ DCT และ ความผิดพลาดที่เกิดจากการแปลงกลับแบบ DCT

ข้อมูลที่ต่อบิทชุดที่	ข้อมูลชุดที่+ข้อมูลชุดที่(บิทที่)	ข้อมูลที่ต่อบิทชุดที่	ข้อมูลชุดที่+ข้อมูลชุดที่(บิทที่)
1	1 + 9(1,2)	5	1 + 10(1,2)
2	2 + 9(3,4)	6	2 + 10(3,4)
3	3 + 9(5,6)	7	3 + 10(5,6)
4	4 + 9(7,9)	8	4 + 10(7,8)

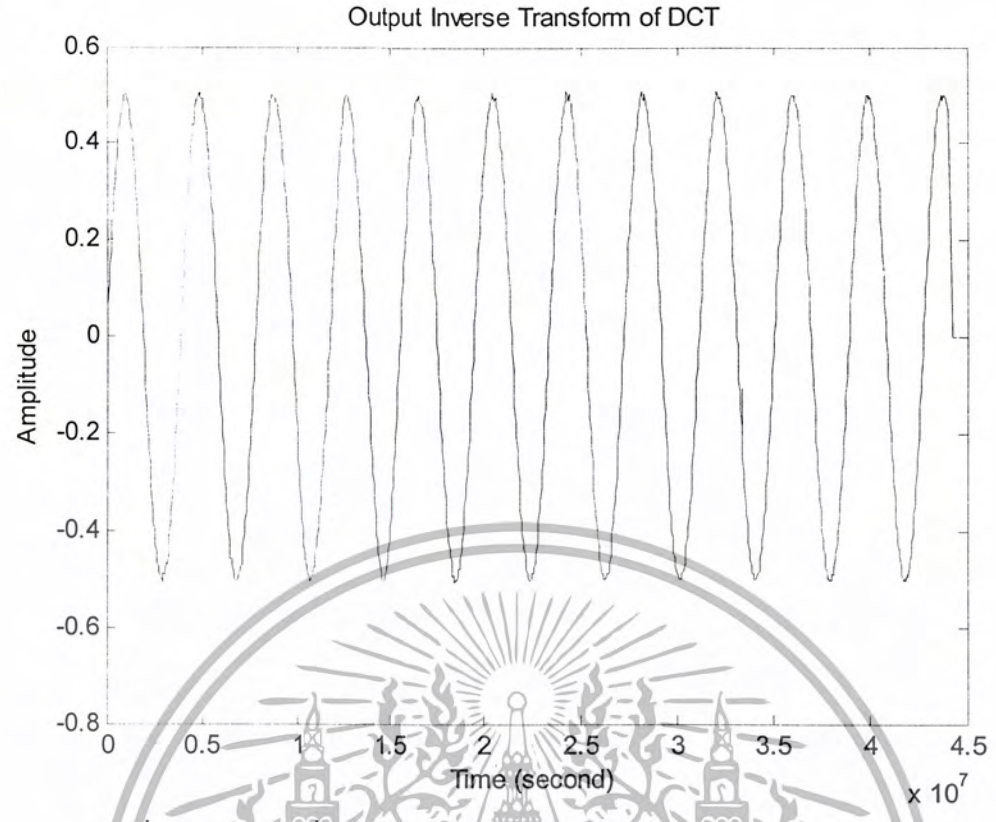
ตารางที่ 4.8 ผลของการการต่อบิทให้ได้ 10 บิทจำนวน 8 ชุด



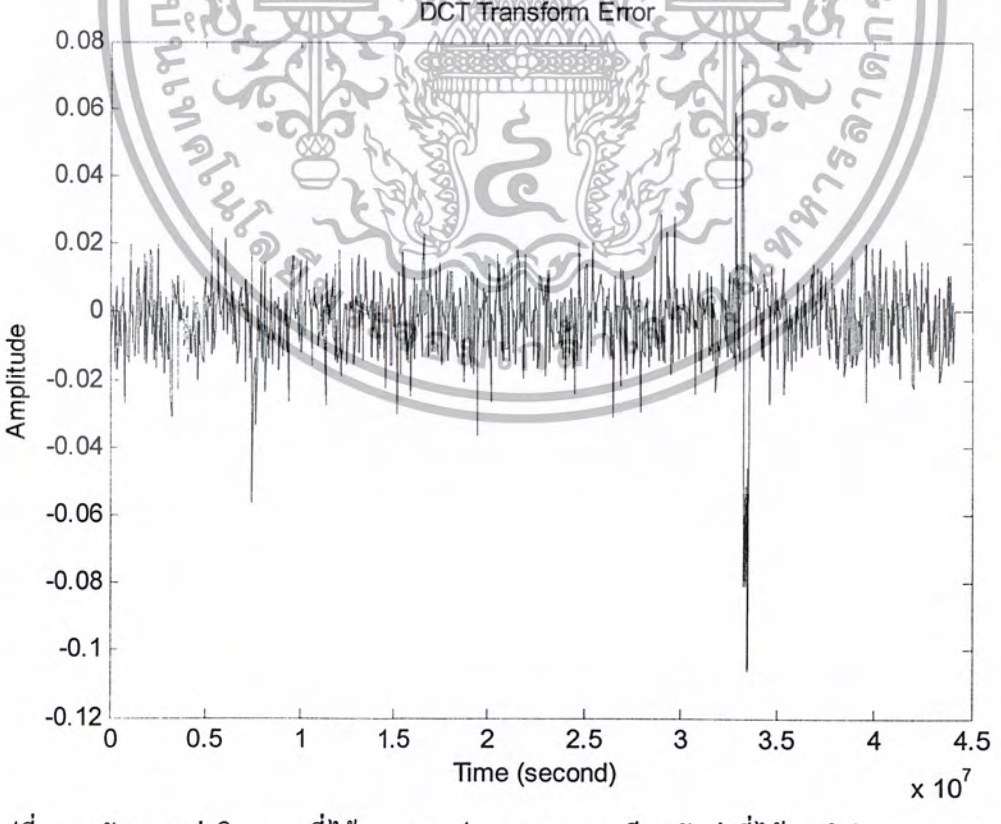
รูปที่ 4.55 ป้อนค่าสัญญาณโดยผ่านโปรแกรมที่เขียนขึ้นมาเพื่อทดสอบการทำงานของ Hard ware



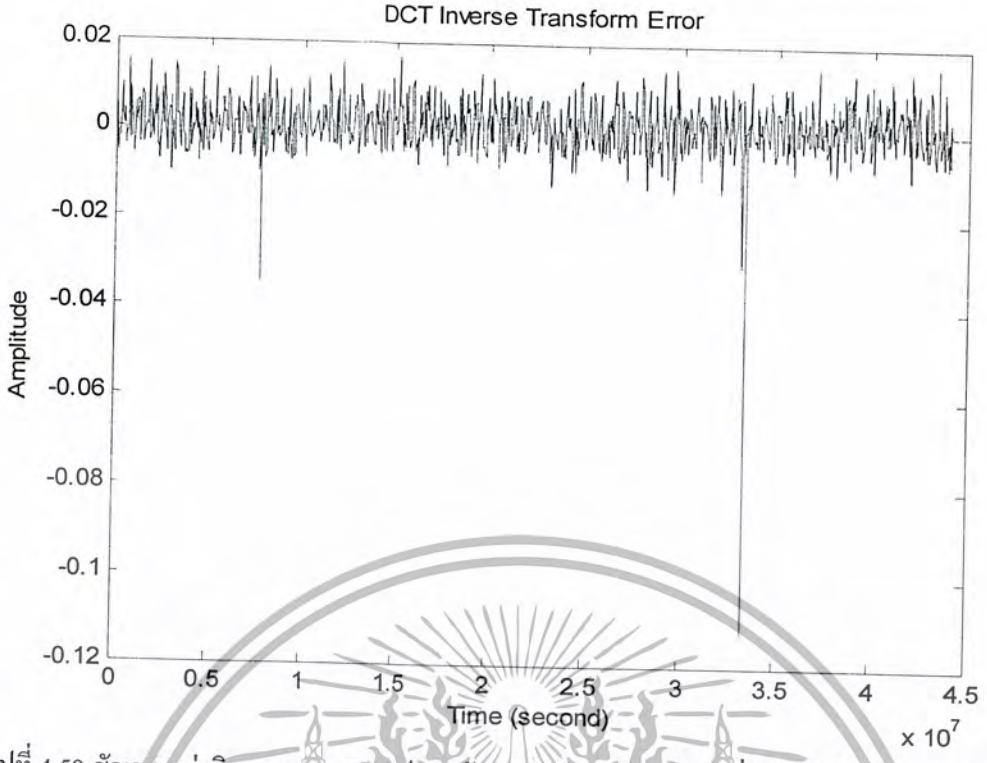
รูปที่ 4.56 สัญญาณที่ได้จากการแปลงด้วย DCT ที่ใช้หลักการของ CORDIC Algorithm ที่จัดค่ากลับมา



รูปที่ 4.57 สัญญาณที่ได้จากการแปลงกลับแบบ DCT ด้วยโปรแกรม MATLAB

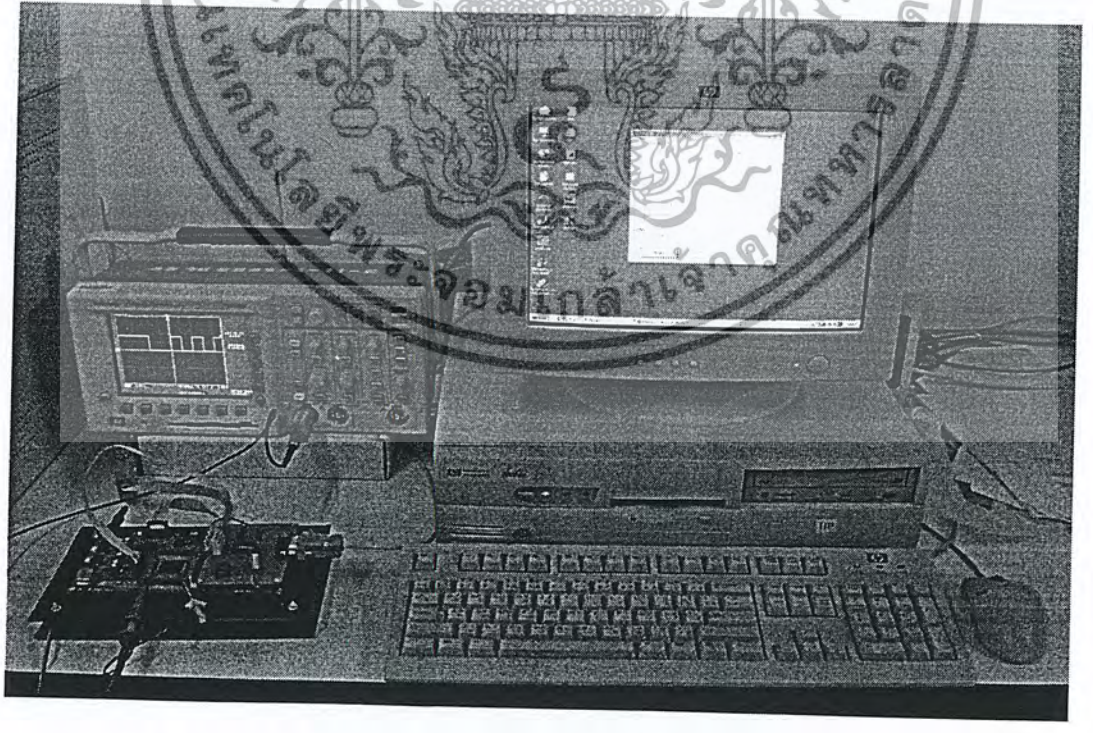


รูปที่ 4.58 สัญญาณค่าผิดพลาดที่ได้จากการแปลงแบบ DCT เทียบกับค่าที่ได้จากโปรแกรม MATLAB

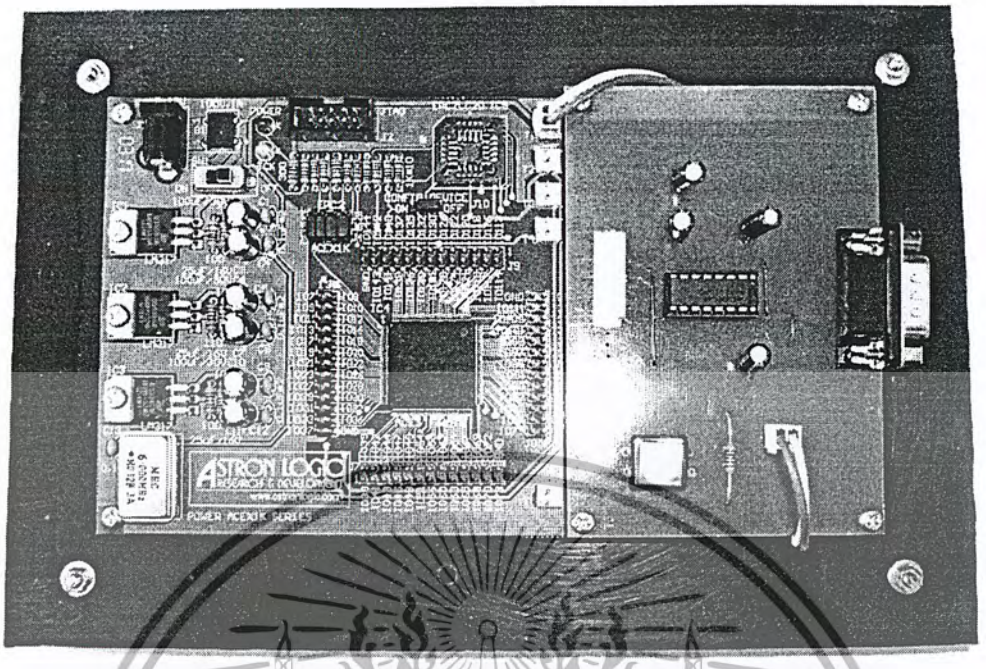


รูปที่ 4.59 สัญญาณค่าผิดพลาดจากการแปลงกลับแบบ DCT เทียบกับค่าที่ได้จากโปรแกรม MATLAB

4.12 ภาพอุปกรณ์ที่ใช้ในทดสอบการแปลง DCT ที่ใช้หลักการของ CORDIC Algorithm



รูปที่ 4.60 อุปกรณ์ที่ใช้ในทดสอบการแปลง DCT ที่ใช้หลักการของ CORDIC Algorithm



รูปที่ 4.61 แสดงบอร์ดเอฟพีจีเอที่ทำการเชื่อมต่อกับพอร์ตอนุกรม

บทที่ 5
บทวิจารณ์และบทสรุป

จากการศึกษาในทางทฤษฎีและทำการทดลองตามขอบเขตของโครงงานนี้สามารถที่จะสรุปความสำคัญของเนื้อหาแต่ละส่วนได้เป็นดังนี้

ทฤษฎีของ CORDIC Algorithm หลักการสำคัญคือการอาศัยความสัมพันธ์ของการเปลี่ยนแปลงตำแหน่งเวคเตอร์ที่มีขนาดเท่ากันหรือการเปลี่ยนตำแหน่งพิจารณาเส้นรัศมีของวงกลมใดๆ ซึ่งจากการเปลี่ยนตำแหน่งนี้จะทำให้ได้มุมเกิดขึ้น และเมื่อทำการโยงความสัมพันธ์ของมุมกับตำแหน่งของเวคเตอร์เข้าด้วยกันแล้วก็จะทำให้ได้ความสัมพันธ์ที่เป็นสมการพื้นฐานของ CORDIC Algorithm ทั้งยังสามารถเพิ่มพารามิเตอร์เพื่อเปลี่ยนระบบในการพิจารณาหรือจัดรูปสมการเพื่อหาค่าที่ต้องการได้อีกเช่น ฟังก์ชันไฮเปอร์บอลิก (Hyperbolic) และฟังก์ชันเอ็กโปเนนเชียล (exponential) เป็นต้น ส่วนการนำไปสร้างเป็นฮาร์ดแวร์ถูกทำให้ง่ายโดยการเลือกพจน์ที่นำมาคูณ ซึ่งในที่นี้คือ $\tan\theta_i$ เป็น 2^{-i} ผลก็คือสามารถนำไปใช้ในฮาร์ดแวร์ที่เป็นระบบดิจิตอลหรือการคำนวณในระบบคอมพิวเตอร์ได้ง่ายและลดอุปกรณ์ในการสร้างลง อีกทั้งยังสามารถเลือกความละเอียดของการคำนวณจากจำนวนการวนรอบของการทำงานได้อีกด้วย

วิธีการแปลงแบบ DCT พื้นฐานสำคัญก็คือการทำ DFT ของสัญญาณนั่นเอง ซึ่งจะทำการหาสมการจากการแปลง DCT ได้จากการแปลง DFT ของลำดับสัญญาณ แล้วทำการจัดรูปให้อยู่ในเทอมของ cosine คูณกับค่าคงที่และอินพุท โดยที่เทอมของ cosine สามารถคำนวณตามจำนวนลำดับสัญญาณอินพุทให้อยู่ในรูปของค่าคงที่ได้ดังนั้นภาพรวมของแปลงแบบ DCT จึงเป็นการนำลำดับสัญญาณเข้ามาคูณกับค่าคงที่ชุดหนึ่ง แต่เนื่องจากว่าสัมประสิทธิ์ค่าคงที่เหล่านี้เกิดจากการแปลง DFT ของลำดับสัญญาณผลลัพธ์ที่ได้จึงกลายเป็นข้อมูลที่อยู่ในโดเมนของความถี่ ซึ่งข้อมูลเหล่านี้จะมีความเป็นอิสระต่อกันทำให้สามารถแยกข้อมูลที่มีความสำคัญมากและน้อยเพื่อนำมาพิจารณาในการลดข้อมูลได้อย่างมีประสิทธิภาพ ทั้งนี้ผลลัพธ์ที่ได้ยังเป็นค่าจำนวนจริงอีกด้วยจึงเป็นที่นิยมนำไปใช้งานอย่างแพร่หลาย

การนำมาประยุกต์ใช้งานร่วมกันโดยการจัดรูปสมการของ DCT ให้อยู่ในรูปสมการพื้นฐานของ CORDIC Algorithm เป็นสำคัญ สิ่งที่จะถูกนำมาพิจารณาร่วมก็คือค่าผลลัพธ์ของ sine และ cosine ที่จะนำมาจัดเข้ารูป ซึ่งใช้ความสัมพันธ์ที่ว่าค่า sine และ cosine จะอยู่ในช่วง –1 ถึง 1 เหมือนกัน ดังนั้นถ้าเราทำการจัดมุมให้ในตำแหน่งที่ลงตัวก็สามารถที่จะนำเทอม cosine ทั้งหมดของ DCT ไปแทนในสมการพื้นฐานของ CORDIC Algorithm ได้เมื่อทำการจัดและค่ามุมได้ลงตัวแล้วทำการคำนวณตามวิธีการของ CORDIC Algorithm ในชุดนั้นๆแทนการคำนวณจากสมการโดยตรงก็จะได้ผลลัพธ์ตามสมการที่ถูกจัดไว้ กระบวนการทั้งหมดจะถูกจัดเป็นหนึ่งบล็อกย่อยๆ ซึ่งก็คือการนำเอาวิธีการของ CORDIC Algorithm เข้าไปช่วยคำนวณค่าตามสมการที่ถูกจัดไว้แต่ละชุดของวิธีการแปลงแบบ DCT นั่นเอง

การทดลองและการนำไปสร้างเป็นฮาร์ดแวร์ ในส่วนของ CORDIC Algorithm ได้ทำการจำลองการทำงาน (simulation) ด้วยโปรแกรม Matlab ซึ่งผลที่ได้เป็นไปตามทฤษฎีโดยที่ความถูกต้องแม่นยำจะขึ้นอยู่กับการเก็บค่าความละเอียดจากการคำนวณแต่ละรอบและจำนวนรอบที่ใช้ในการคำนวณ(การหมุน) ส่วนของ DCT ทดลองสร้างโดยใช้ภาษาวีเอชดีแอล และเมื่อทำการจำลองการทำงานดูผลที่ได้ก็มีความ

ใกล้เคียงกับค่าที่คำนวณได้ในทางทฤษฎี ทั้งนี้ความผิดพลาด (error) ที่เกิดขึ้นอันเนื่องมาจากว่าเมื่อทำการนำมาสร้างจริงค่าอินพุท,ค่าคงที่และค่าผลลัพธ์จากการคำนวณแต่ละชุดซึ่งเป็นค่าจำนวนจริงจะถูกแปลงให้อยู่ในรูปของเลขไบนารี่ (binary) ที่มีจำนวนบิทจำกัด ทำให้เกิดค่าผิดพลาดขึ้น และในการสร้างโดยการบรรยายพฤติกรรมด้วยภาษาวีเอชดีแอลนี้ จำเป็นที่จะต้องคำนึงถึงการทำงานในอุปกรณ์จริงที่อุปกรณ์แต่ละตัวจะทำงานพร้อมกัน แต่อาจใช้เวลาในการทำงานไม่เท่ากันจึงต้องมีการควบคุมเวลาเช่น การส่งและรับค่าให้ดีมิฉะนั้นแล้วอาจจะทำให้ผลลัพธ์มีความผิดพลาดเกิดขึ้นได้เช่นกัน

จากที่กล่าวมาข้างต้นเป็นส่วนของการศึกษาในทางทฤษฎีและการจำลองผลการทำงานเบื้องต้นของอุปกรณ์ต่างๆ ซึ่งในการนำไปสร้างจริงนั้น จะทำการพิจารณาถึงข้อจำกัดของตัวอุปกรณ์ (Hard Ware) ที่มีในท้องตลาด ดังนั้นจึงได้นำอาทฤษฎีของเลขคณิตกระจายมาใช้ในโครงสร้างของ CORDIC Algorithm ซึ่งทำให้จำนวนของอุปกรณ์ที่ใช้งานจริงลดลง อีกทั้งยังง่ายต่อการสร้างเนื่องจากมีโครงสร้างที่ไม่ซับซ้อนเท่ากับโครงสร้างของ CORDIC Algorithm พื้นฐาน ทั้งนี้เพราะว่าวิธีการแปลงแบบ DCT มีสัมประสิทธิ์การคูณที่เป็นค่าตายตัวดังนั้นค่ามุมที่ใช้จึงมีค่าตายตัวเช่นกัน ค่าสัมประสิทธิ์เหล่านี้จึงสามารถคำนวณหาค่าตามสมการได้เลยและนำไปเก็บไว้ใน ROM เพื่อใช้กับการคำนวณตามทฤษฎีเลขคณิตกระจายได้อย่างลงตัว

ส่วนของการทดลองการทำงานจริงกับตัวอุปกรณ์ เนื่องจากขอบเขตของโครงงานนี้ได้ทำการสร้างขึ้นมาเฉพาะตัวแปลง DCT ดังนั้นการแปลงกลับจึงได้อาศัยการทำงานของโปรแกรมคอมพิวเตอร์ซึ่งก็คือโปรแกรม Matlab ทำให้ต้องมีการเชื่อมต่อ (Interface) ระหว่างคอมพิวเตอร์กับตัวอุปกรณ์ ซึ่งในที่นี้จะใช้โปรแกรม VB (Visual Basic) ในการควบคุมการส่งผ่านข้อมูลผ่านพอร์ทอนุกรม (Serial Port) โดยจะทำหน้าที่ทั้งรับและส่งข้อมูลเพื่อนำไปประมวลผลต่อไป และในส่วนนี้เราจำเป็นที่จะต้องจัดรูปแบบการส่งข้อมูลให้เข้ากับมาตรฐานของคอมพิวเตอร์และตัวโปรแกรมของตัว VB เอง และรายละเอียดต่างๆ ของการทำงานในแต่ละวงจรก็ได้กล่าวไว้ในส่วนของบทที่ 4 ที่ว่าด้วยการทดลองและผลการทดลอง

จะเห็นได้ว่าโครงงานนี้ก็ได้แสดงให้เห็นถึงวิธีการโดยรวมของการสร้างอุปกรณ์จริง นับตั้งแต่การศึกษาถึงทฤษฎีและการนำมาประยุกต์ใช้งานร่วมกัน ตลอดจนการนำเอาทฤษฎีเหล่านั้นมาสร้างเป็นอุปกรณ์ด้วยภาษาที่มีความยืดหยุ่นสูง จึงนับว่าวิธีการเหล่านี้เหมาะที่จะทำการศึกษาและค้นคว้าเพื่อการพัฒนาหรือการนำไปใช้งานจริงของวิศวกรเป็นอย่างยิ่ง

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;
entity      add_sub8 is
port(x0,x1,x2,x3,x4,x5,x6,x7 : in std_logic_vector (7 downto 0);
     p0,p1,p2,p3,m0,m1,m2,m3 : out std_logic_vector (8 downto 0));
end;
architecture behave of add_sub8 is
begin
                  p0 <= (x0(7)&x0)+(x7(7)&x7);
                  p1 <= (x3(7)&x3)+(x4(7)&x4);
                  p2 <= (x1(7)&x1)+(x6(7)&x6);
                  p3 <= (x2(7)&x2)+(x5(7)&x5);
                  m0 <= (x0(7)&x0)-(x7(7)&x7);
                  m1 <= (x3(7)&x3)-(x4(7)&x4);
                  m2 <= (x1(7)&x1)-(x6(7)&x6);
                  m3 <= (x2(7)&x2)-(x5(7)&x5);
end behave;


Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;
entity      add_sub9 is
port(en : in std_logic;
     in_1,in_2 : in std_logic_vector (8 downto 0);
     o_add,o_sub : out std_logic_vector (9 downto 0));
end;
architecture behave of add_sub9 is
begin
process(en)
begin
            if en='1' then
                  o_add <= (in_1(8)&in_1)+(in_2(8)&in_2);
                  o_sub <= (in_1(8)&in_1)-(in_2(8)&in_2);
             end if;
end process;
end;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity add_acc9 is
Port( add_acc_in          : in  std_logic_vector(8 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(8 downto 0));
End;
Architecture add_acc_a of add_acc9 is
Signal add_out,acc_out : std_logic_vector(8 downto 0);
Begin
addp : Process (s_a,add_acc_in,acc_out)
         Variable s_sub,sum : std_logic_vector(8 downto 0);
         Variable s         : std_logic_vector(9 downto 0);
       Begin
          if   s_a = '0' then
               s:=("0"&add_acc_in)+acc_out;
               sum(8):=s(9) xor add_acc_in(8) xor acc_out(8) ;
               sum(7 downto 0):=s(8 downto 1);
```

```
          else s_sub:=(add_acc_in xor "111111111");
               s_sub:=s_sub+1;
               sum(8 downto 0):=acc_out(8 downto 0)+s_sub(8 downto 0);
          end if;
          add_out<=sum;
        End Process addp ;
accp :  Process (clacc,lacc)
        Begin
          if  clacc='0' then
              acc_out<=(others=>'0');
          elsif lacc'event and lacc='1' then
              acc_out<=add_out;
         end if;
        End Process accp ;
Buff :  Process (ld)
        begin
          if ld='1' then
             add_acc_out(8 downto 1)<=acc_out(7 downto 0);
             add_acc_out(0)<='0';
          end if;
        End process Buff ;
End;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity add_acc10 is
Port( add_acc_in       : in  std_logic_vector(9 downto 0);
      s_a,clacc,lacc,ld : in  std_logic;
      add_acc_out       : out std_logic_vector(9 downto 0));
End;
Architecture add_acc_a of add_acc10 is
Signal add_out,acc_out : std_logic_vector(9 downto 0);
Begin
addp :  Process (s_a,add_acc_in,acc_out)
          Variable s_sub,sum : std_logic_vector(9 downto 0);
          Variable s         : std_logic_vector(10 downto 0);
        Begin
          if    s_a = '0' then
                s:=("0"&add_acc_in)+acc_out;
                sum(9):=s(10) xor add_acc_in(9) xor acc_out(9) ;
                sum(8 downto 0):=s(9 downto 1);
          else s_sub:=(add_acc_in xor "1111111111");
                s_sub:=s_sub+1;
                sum(9 downto 0):=acc_out(9 downto 0)+s_sub(9 downto 0);
          end if;
          add_out<=sum;
        End Process addp ;
accp :  Process (clacc,lacc)
        Begin
         if  clacc='0' then
             acc_out<=(others=>'0');
         elsif lacc'event and lacc='1' then
             acc_out<=add_out;
         end if;
        End Process accp ;
Buff :  Process (ld)
        begin
          if ld='1' then
              add_acc_out(9 downto 1)<=acc_out(8 downto 0);
```

```
            add_acc_out(0)<='0';
        end if;
      End process Buff ;
End;


library ieee;
use ieee.std_logic_1164.all;
entity romxy11_25 is
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end;
architecture rtl of romxy11_25 is
begin
process(rom_add_x,rom_add_y)
begin
      if rom_add_x='0' then
            if rom_add_y='0' then
                  rom_out_x<="000000000";
                  rom_out_y<="000000000";
            else
                  rom_out_x<="111101000";
                  rom_out_y<="001111101";
            end if;
      else
            if rom_add_y='0' then
                  rom_out_x<="001111101";
                  rom_out_y<="000011000";
            else
                  rom_out_x<="001100100";
                  rom_out_y<="010010110";
            end if;
      end if;
end process;
end;


library ieee;
use ieee.std_logic_1164.all;
entity romxy33_75 is
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end;
architecture rtl of romxy33_75 is
begin
process(rom_add_x,rom_add_y)
begin
      if rom_add_x='0' then
            if rom_add_y='0' then
                  rom_out_x<="000000000";
                  rom_out_y<="000000000";
            else
                  rom_out_x<="110111001";
                  rom_out_y<="001101010";
            end if;
      else
            if rom_add_y='0' then
                  rom_out_x<="001101010";
                  rom_out_y<="001000111";
            else
                  rom_out_x<="000100011";
```

```
                  rom_out_y<="010110001";
            end if;
      end if;
end process;
end;


library ieee;
use ieee.std_logic_1164.all;
entity romxy45 is
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(9 downto 0));
end;
architecture rtl of romxy45 is
begin
process(rom_add_x,rom_add_y)
begin
      if rom_add_x='0' then
            if rom_add_y='0' then
                  rom_out_x<="0000000000";
                  rom_out_y<="0000000000";
            else
                  rom_out_x<="1101001011";
                  rom_out_y<="0010110101";
            end if;
      else
            if rom_add_y='0' then
                  rom_out_x<="0010110101";
                  rom_out_y<="0010110101";
            else
                  rom_out_x<="0000000000";
                  rom_out_y<="0101101010";
            end if;
      end if;
end process;
end;


library ieee;
use ieee.std_logic_1164.all;
entity romxy56_25 is
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end;
architecture rtl of romxy56_25 is
begin
process(rom_add_x,rom_add_y)
begin
      if rom_add_x='0' then
            if rom_add_y='0' then
                  rom_out_x<="000000000";
                  rom_out_y<="000000000";
            else
                  rom_out_x<="110010110";
                  rom_out_y<="001000111";
            end if;
      else
            if rom_add_y='0' then
                  rom_out_x<="001000111";
                  rom_out_y<="001101010";
            else
```

```
                rom_out_x<="111011101";
                rom_out_y<="010110001";
            end if;
      end if;
end process;
end;


library ieee;
use ieee.std_logic_1164.all;
entity romxy67_5 is
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(9 downto 0));
end;
architecture rtl of romxy67_5 is
begin
process(rom_add_x,rom_add_y)
begin
      if rom_add_x='0' then
            if rom_add_y='0' then
                  rom_out_x<="0000000000";
                  rom_out_y<="0000000000";
            else
                  rom_out_x<="1100010100";
                  rom_out_y<="0001100001";
            end if;
      else
            if rom_add_y='0' then
                  rom_out_x<="0001100001";
                  rom_out_y<="0011101100";
            else
                  rom_out_x<="1101110110";
                  rom_out_y<="0101001110";
            end if;
      end if;
end process;
end;


library ieee;
use ieee.std_logic_1164.all;
entity romxy78_75 is
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end;
architecture rtl of romxy78_75 is
begin
process(rom_add_x,rom_add_y)
begin
      if rom_add_x='0' then
            if rom_add_y='0' then
                  rom_out_x<="000000000";
                  rom_out_y<="000000000";
            else
                  rom_out_x<="110000011";
                  rom_out_y<="000011000";
            end if;
      else
            if rom_add_y='0' then
                  rom_out_x<="000011000";
                  rom_out_y<="001111101";
```

```
            else
                  rom_out_x<="110011100";
                  rom_out_y<="010010110";
            end if;
      end if;
end process;
end;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity shif_ps9 is
Port( shif_x,shif_y   : in  std_logic_vector(8 downto 0);
      clk,ld    : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
End;
Architecture rtl of shif_ps9 is
       signal shif_buf_x,shif_buf_y : std_logic_vector(8 downto 0);
Begin
Process (ld,clk)
Begin
      if   ld = '1' then
            shif_buf_x<=shif_x;
            shif_buf_y<=shif_y;
      elsif clk'event and clk='1' then
            shif_out_x<=shif_buf_x(0);
            shif_out_y<=shif_buf_y(0);
            shif_buf_x(7 downto 0)<=shif_buf_x(8 downto 1);
            shif_buf_y(7 downto 0)<=shif_buf_y(8 downto 1);
      end if;
End Process  ;
End;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity shif_ps10 is
Port( shif_x,shif_y   : in  std_logic_vector(9 downto 0);
      clk,ld    : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
End;
Architecture rtl of shif_ps10 is
       signal shif_buf_x,shif_buf_y : std_logic_vector(9 downto 0);
Begin
Process (ld,clk)
Begin
      if   ld = '1' then
            shif_buf_x<=shif_x;
            shif_buf_y<=shif_y;
      elsif clk'event and clk='1' then
            shif_out_x<=shif_buf_x(0);
            shif_out_y<=shif_buf_y(0);
            shif_buf_x(8 downto 0)<=shif_buf_x(9 downto 1);
            shif_buf_y(8 downto 0)<=shif_buf_y(9 downto 1);
      end if;
End Process  ;
End;
```

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
entity      control9 is
port(  sys_clk,en : in std_logic;
       s_a,clacc,lacc : out std_logic;
       ld,clk : out std_logic);
end;
architecture behave of control9 is
begin
process(sys_clk,en)
      VARIABLE state : std_logic_vector(2 downto 0):="000";
      VARIABLE count : integer range 0 to 8;
begin
      if en='0' then
            state:="000";
            count:=0;
            ld<='0';
            clacc<='0';
            clk<='0';
      elsif (sys_clk'event and sys_clk ='1') then
            case state is
                  when "000" =>
                        ld<='1';
                        clacc<='0';
                        state := "001";
                  when "001" =>
                        ld<='0';
                        state := "010";
                  when "010" =>
                        clk<='1';
                        state := "011";
                  when "011" =>
                        if count = 8 then
                              s_a<='1';
                        else
                              s_a<='0';
                        end if;
                        state := "100";
                  when "100" =>
                        clk<='0';
                        clacc<='1';
                        state := "101";
                  when "101" =>
                        lacc<='1';
                        state := "110";
                  when "110" =>
                        lacc<='0';
                        state := "111";
                  when "111" =>
                        if count < 8 then
                              count:= count+1;
                              state := "010";
                        else
                              count := 0;
                              state := "000";
                        end if;
                  when others =>
                        state := "111";
                  end case;
      end if;
```

```
end process ;
end behave;


Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
entity      control10 is
port(  sys_clk,en : in std_logic;
       s_a,clacc,lacc : out std_logic;
       ld,clk : out std_logic);
end;
architecture behave of control10 is
begin
process(sys_clk,en)
      VARIABLE state : std_logic_vector(2 downto 0):="000";
      VARIABLE count : integer range 0 to 9;
begin
      if en='0' then
            state:="000";
            count:=0;
            ld<='0';
            clacc<='0';
            clk<='0';
      elsif (sys_clk'event and sys_clk ='1') then
            case state is
                  when "000" =>
                        ld<='1';
                        clacc<='0';
                        state := "001";
                  when "001" =>
                        ld<='0';
                        state := "010";
                  when "010" =>
                        clk<='1';
                        state := "011";
                  when "011" =>
                        if count = 9 then
                              s_a<='1';
                        else
                              s_a<='0';
                        end if;
                        state := "100";
                  when "100" =>
                        clk<='0';
                        clacc<='1';
                        state := "101";
                  when "101" =>
                        lacc<='1';
                        state := "110";
                  when "110" =>
                        lacc<='0';
                        state := "111";
                  when "111" =>
                        if count < 9 then
                              count:= count+1;
                              state := "010";
                        else
                              count := 0;
                              state := "000";
                        end if;
```

```
                    when others =>
                          state := "111";
                    end case;
      end if;
end process ;
end behave;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity da_cordic_1 is
Port( sys_clk,en   : in  std_logic;
      xin,yin      : in std_logic_vector(8 downto 0);
      data_out_x,data_out_y : out std_logic_vector(8 downto 0));
End;
Architecture rtl of da_cordic_1 is
component shif_ps9
Port( shif_x,shif_y   : in  std_logic_vector(8 downto 0);
      clk,ld    : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
end component;
component romxy78_75
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end component;
component add_acc9
Port( add_acc_in          : in  std_logic_vector(8 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(8 downto 0));
end component;
component control9
port( sys_clk,en : in std_logic;
      s_a,clacc,lacc  : out std_logic;
      ld,clk : out std_logic);
end component;
signal s_clk,s_s_a,s_ld,s_clacc,s_lacc : std_logic;
signal add_x,add_y :std_logic;
signal add_romx,add_romy  : std_logic_vector(8 downto 0);
Begin
control_xy :control9
            port map(sys_clk,en,s_s_a,s_clacc,s_lacc,s_ld,s_clk );
shif_ps_xy : shif_ps9 port map(xin,yin,s_clk,s_ld,add_x,add_y );
rom    : romxy78_75       port map(add_x,add_y,add_romx,add_romy);
add_acc_x: add_acc9
           port map(add_romx,s_s_a,s_clacc,s_lacc,s_ld,data_out_x);
add_acc_y: add_acc9
           port map(add_romy,s_s_a,s_clacc,s_lacc,s_ld,data_out_y);
End;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity da_cordic_2 is
Port( sys_clk,en   : in  std_logic;
      xin,yin      : in std_logic_vector(8 downto 0);
      data_out_x,data_out_y : out std_logic_vector(8 downto 0));
End;
Architecture rtl of da_cordic_2 is
component shif_ps9
```

```
Port( shif_x,shif_y   : in  std_logic_vector(8 downto 0);
      clk,ld   : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
end component;
component romxy56_25
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end component;
component add_acc9
Port( add_acc_in          : in  std_logic_vector(8 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(8 downto 0));
end component;
component control9
port( sys_clk,en : in std_logic;
      s_a,clacc,lacc : out std_logic;
      ld,clk : out std_logic);
end component;
signal s_clk,s_s_a,s_ld,s_clacc,s_lacc : std_logic;
signal add_x,add_y :std_logic;
signal add_romx,add_romy  : std_logic_vector(8 downto 0);
Begin
control_xy :control9
            port map(sys_clk,en,s_s_a,s_clacc,s_lacc,s_ld,s_clk );
shif_ps_xy : shif_ps9 port map(xin,yin,s_clk,s_ld,add_x,add_y );
rom   : romxy56_25      port map(add_x,add_y,add_romx,add_romy);
add_acc_x: add_acc9
           port map(add_romx,s_s_a,s_clacc,s_lacc,s_ld,data_out_x);
add_acc_y: add_acc9
           port map(add_romy,s_s_a,s_clacc,s_lacc,s_ld,data_out_y);
End;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity da_cordic_3 is
Port( sys_clk,en   : in  std_logic;
      xin,yin      : in std_logic_vector(8 downto 0);
      data_out_x,data_out_y : out std_logic_vector(8 downto 0));
End;
Architecture rtl of da_cordic_3 is
component shif_ps9
Port( shif_x,shif_y   : in  std_logic_vector(8 downto 0);
      clk,ld   : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
end component;
component romxy33_75
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end component;
component add_acc9
Port( add_acc_in          : in  std_logic_vector(8 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(8 downto 0));
end component;
component control9
port( sys_clk,en : in std_logic;
      s_a,clacc,lacc : out std_logic;
      ld,clk : out std_logic);
end component;
```

```
signal s_clk,s_s_a,s_ld,s_clacc,s_lacc : std_logic;
signal add_x,add_y :std_logic;
signal add_romx,add_romy  : std_logic_vector(8 downto 0);
Begin
control_xy :control9
            port map(sys_clk,en,s_s_a,s_clacc,s_lacc,s_ld,s_clk );
shif_ps_xy : shif_ps9 port map(xin,yin,s_clk,s_ld,add_x,add_y );
rom   : romxy33_75      port map(add_x,add_y,add_romx,add_romy);
add_acc_x: add_acc9
           port map(add_romx,s_s_a,s_clacc,s_lacc,s_ld,data_out_x);
add_acc_y: add_acc9
           port map(add_romy,s_s_a,s_clacc,s_lacc,s_ld,data_out_y);
End;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity da_cordic_4 is
Port( sys_clk,en   : in  std_logic;
      xin,yin      : in std_logic_vector(8 downto 0);
      data_out_x,data_out_y : out std_logic_vector(8 downto 0));
End;

Architecture rtl of da_cordic_4 is
component shif_ps9
Port( shif_x,shif_y   : in  std_logic_vector(8 downto 0);
      clk,ld    : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
end component;
component romxy11_25
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(8 downto 0));
end component;
component add_acc9
Port( add_acc_in        : in  std_logic_vector(8 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(8 downto 0));
end component;
component control9
port( sys_clk,en : in std_logic;
      s_a,clacc,lacc : out std_logic;
      ld,clk : out std_logic);
end component;
signal s_clk,s_s_a,s_ld,s_clacc,s_lacc : std_logic;
signal add_x,add_y :std_logic;
signal add_romx,add_romy  : std_logic_vector(8 downto 0);
Begin
control_xy :control9
            port map(sys_clk,en,s_s_a,s_clacc,s_lacc,s_ld,s_clk );
shif_ps_xy : shif_ps9 port map(xin,yin,s_clk,s_ld,add_x,add_y );
rom   : romxy11_25      port map(add_x,add_y,add_romx,add_romy);
add_acc_x: add_acc9
           port map(add_romx,s_s_a,s_clacc,s_lacc,s_ld,data_out_x);
add_acc_y: add_acc9
           port map(add_romy,s_s_a,s_clacc,s_lacc,s_ld,data_out_y);
End;


library ieee;
Use ieee.std_logic_1164.ALL;
```

```
Use ieee.std_logic_Unsigned.ALL;
Entity da_cordic_5 is
Port( sys_clk,en       : in  std_logic;
      xin,yin     : in std_logic_vector(9 downto 0);
      data_out_x,data_out_y : out std_logic_vector(9 downto 0));
End;

Architecture rtl of da_cordic_5  is
component shif_ps10
Port( shif_x,shif_y   : in  std_logic_vector(9 downto 0);
      clk,ld   : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
end component;
component romxy67_5
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(9 downto 0));
end component;
component add_acc10
Port( add_acc_in          : in  std_logic_vector(9 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(9 downto 0));
end component;
component control10
port( sys_clk,en : in std_logic;
      s_a,clacc,lacc : out std_logic;
      ld,clk : out std_logic);
end component;
signal s_clk,s_s_a,s_ld,s_clacc,s_lacc : std_logic;
signal add_x,add_y :std_logic;
signal add_romx,add_romy  : std_logic_vector(9 downto 0);
Begin
control_xy :control10
           port map(sys_clk,en,s_s_a,s_clacc,s_lacc,s_ld,s_clk );
shif_ps_xy : shif_ps10 port map(xin,yin,s_clk,s_ld,add_x,add_y );
rom   : romxy67_5 port map(add_x,add_y,add_romx,add_romy);
add_acc_x: add_acc10
           port map(add_romx,s_s_a,s_clacc,s_lacc,s_ld,data_out_x);
add_acc_y: add_acc10
           port map(add_romy,s_s_a,s_clacc,s_lacc,s_ld,data_out_y);
End;


library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity da_cordic_6 is
Port( sys_clk,en       : in  std_logic;
      xin,yin     : in std_logic_vector(9 downto 0);
      data_out_x,data_out_y : out std_logic_vector(9 downto 0));
End;
Architecture rtl of da_cordic_6  is
component shif_ps10
Port( shif_x,shif_y   : in  std_logic_vector(9 downto 0);
      clk,ld   : in  std_logic;
      shif_out_x,shif_out_y   : out std_logic);
end component;
component romxy45
port( rom_add_x,rom_add_y : in std_logic;
      rom_out_x,rom_out_y : out std_logic_vector(9 downto 0));
end component;
component add_acc10
```

```
Port( add_acc_in          : in  std_logic_vector(9 downto 0);
      s_a,clacc,lacc,ld   : in  std_logic;
      add_acc_out         : out std_logic_vector(9 downto 0));
end component;
component control10
port( sys_clk,en : in std_logic;
      s_a,clacc,lacc : out std_logic;
      ld,clk : out std_logic);
end component;
signal s_clk,s_s_a,s_ld,s_clacc,s_lacc : std_logic;
signal add_x,add_y :std_logic;
signal add_romx,add_romy  : std_logic_vector(9 downto 0);
Begin
control_xy :control10
            port map(sys_clk,en,s_s_a,s_clacc,s_lacc,s_ld,s_clk );
shif_ps_xy : shif_ps10 port map(xin,yin,s_clk,s_ld,add_x,add_y );
rom   : romxy45          port map(add_x,add_y,add_romx,add_romy);
add_acc_x: add_acc10
           port map(add_romx,s_s_a,s_clacc,s_lacc,s_ld,data_out_x);
add_acc_y: add_acc10
           port map(add_romy,s_s_a,s_clacc,s_lacc,s_ld,data_out_y);
End;


Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;
entity      add9 is
port( en : in std_logic;
      in_1,in_2 : in std_logic_vector (8 downto 0);
      o_add : out std_logic_vector (9 downto 0));
end;
architecture behave of add9 is
begin
process(en)
begin
      if en='1' then
            o_add <= (in_1(8)&in_1)+(in_2(8)&in_2);
      end if;
end process;
end behave;

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;
entity      sub9 is
port( en : in std_logic;
      in_1,in_2 : in std_logic_vector (8 downto 0);
      o_sub : out std_logic_vector (9 downto 0));
end;
architecture behave of sub9 is
begin
process(en)
begin
      if en='1' then
            o_sub <= (in_1(8)&in_1)-(in_2(8)&in_2);
      end if;
end process;
end behave;
```

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
entity      reg10_10 is
port( en : in std_logic;
      i_reg1,i_reg2,i_reg3,i_reg4:in std_logic_vector (9 downto 0);
      i_reg5,i_reg6,i_reg7,i_reg8:in std_logic_vector (9 downto 0);
      o_reg1,o_reg2,o_reg3,o_reg4:out std_logic_vector (9 downto 0);
      o_reg5,o_reg6,o_reg7,o_reg8:out std_logic_vector (9 downto 0));
end;
architecture behave of reg10_10 is
begin
process(en)
begin
      if en='1' then
            o_reg1 <= i_reg1;
            o_reg2 <= i_reg2;
            o_reg3 <= i_reg3;
            o_reg4 <= i_reg4;
            o_reg5 <= i_reg5;
            o_reg6 <= i_reg6;
            o_reg7 <= i_reg7;
            o_reg8 <= i_reg8;
      end if;
end process;
end behave;


Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
entity      con_dct is
port(  clk : in std_logic;
       en_reg : out std_logic;
       en_b9,en_b10 : out std_logic);
end;
architecture behave of con_dct is
begin
process(clk)
      VARIABLE state : std_logic_vector(2 downto 0);
      VARIABLE count_da : std_logic_vector(5 downto 0);
begin
      if (clk'event and clk ='1') then
            case state is
                  when "001" =>
                        en_reg<='0';
                        en_b9<='1';
                        count_da:="000000";
                        state := "010";
                  when "010" =>
                        state := "011";
                  when "011" =>
                        if count_da < "011101" then
                              count_da:=count_da+"000001";
                              state := "010";
                        else
                              en_b9<='0';
                              count_da:="000000";
                              state := "100";
                        end if;
```

```
                when "100" =>
                      en_b10<='1';
                      state := "101";
                when "101" =>
                      if count_da < "100000" then
                            count_da:=count_da+"000001";
                            state := "100";
                      else
                            en_b10<='0';
                            count_da:="000000";
                            state := "110";
                      end if;
                when "110" =>
                      en_reg<='1';
                      state := "001";
                when others =>
                      state :="001";
          end case;
     end if;
end process ;
end behave;


Library ieee;
Use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
ENTITY dctcordic IS
port(clk : in std_logic;
   ix0,ix1,ix2,ix3,ix4,ix5,ix6,ix7:in std_logic_vector(7 downto 0);
   oy0,oy1,oy2,oy3,oy4,oy5,oy6,oy7:out std_logic_vector(9 downto 0));
END ;
ARCHITECTURE structural OF dctcordic IS
COMPONENT   add_sub8
port(  x0,x1,x2,x3,x4,x5,x6,x7 : in std_logic_vector (7 downto 0);
       p0,p1,p2,p3,m0,m1,m2,m3 : out std_logic_vector (8 downto 0));
END COMPONENT;
COMPONENT add_sub9
port(  en : in std_logic;
       in_1,in_2 : in std_logic_vector (8 downto 0);
       o_add,o_sub : out std_logic_vector (9 downto 0));
END COMPONENT;
COMPONENT add9
port(  en : in std_logic;
       in_1,in_2 : in std_logic_vector (8 downto 0);
       o_add : out std_logic_vector (9 downto 0));
END COMPONENT;
COMPONENT sub9
port(  en : in std_logic;
       in_1,in_2 : in std_logic_vector (8 downto 0);
       o_sub : out std_logic_vector (9 downto 0));
END COMPONENT;
COMPONENT da_cordic_1
Port(  sys_clk,en   : in  std_logic;
       xin,yin    : in std_logic_vector(8 downto 0);
       data_out_x,data_out_y : out std_logic_vector(8 downto 0));
END COMPONENT;
COMPONENT da_cordic_2
Port(  sys_clk,en   : in  std_logic;
       xin,yin    : in std_logic_vector(8 downto 0);
       data_out_x,data_out_y : out std_logic_vector(8 downto 0));
END COMPONENT;
```

```
COMPONENT da_cordic_3
Port(  sys_clk,en   : in  std_logic;
       xin,yin      : in std_logic_vector(8 downto 0);
       data_out_x,data_out_y : out std_logic_vector(8 downto 0));
END COMPONENT;
COMPONENT da_cordic_4
Port(  sys_clk,en   : in  std_logic;
       xin,yin      : in std_logic_vector(8 downto 0);
       data_out_x,data_out_y : out std_logic_vector(8 downto 0));
END COMPONENT;
COMPONENT da_cordic_5
Port(  sys_clk,en       : in  std_logic;
       xin,yin      : in std_logic_vector(9 downto 0);
       data_out_x,data_out_y : out std_logic_vector(9 downto 0));
END COMPONENT;
COMPONENT da_cordic_6
Port(  sys_clk,en       : in  std_logic;
       xin,yin      : in std_logic_vector(9 downto 0);
       data_out_x,data_out_y : out std_logic_vector(9 downto 0));
END COMPONENT;
COMPONENT con_dct
port(  clk : in std_logic;
       en_reg : out std_logic;
       en_b9,en_b10 : out std_logic);
END COMPONENT;
COMPONENT reg10_10
port(en : in std_logic;
     i_reg1,i_reg2,i_reg3,i_reg4:in std_logic_vector (9 downto 0);
     i_reg5,i_reg6,i_reg7,i_reg8:in std_logic_vector (9 downto 0);
     o_reg1,o_reg2,o_reg3,o_reg4:out std_logic_vector (9 downto 0);
     o_reg5,o_reg6,o_reg7,o_reg8:out std_logic_vector (9 downto 0));
END COMPONENT;
SIGNAL en_reg,en_b9,en_b10 : std_logic;
SIGNAL p0,p1,p2,p3,m0,m1,m2,m3 : std_logic_vector(8 downto 0);
SIGNAL pp0,pp1,pp2,pp3,mm0,mm1,mm2,mm3: std_logic_vector(8 downto 0);
SIGNAL pp4,pp5,mm4,mm5 : std_logic_vector(9 downto 0);
SIGNAL dy0,dy1,dy2,dy3,dy4,dy5,dy6,dy7: std_logic_vector(9 downto 0);
begin
      control:con_dct port map(clk,en_reg,en_b9,en_b10);
      add_sub8_1:add_sub8 port map(ix0,ix1,ix2,ix3,ix4,ix5,ix6,ix7,
                          p0,p1,p2,p3,m0,m1,m2,m3);
      add_sub9_1:add_sub9 port map(en_b9,p0,p1,pp4,mm4);
      add_sub9_2:add_sub9 port map(en_b9,p2,p3,pp5,mm5);
      cordic1:da_cordic_1 port map(clk,en_b9,m0,m1,pp0,mm0);
      cordic2:da_cordic_2 port map(clk,en_b9,m2,m3,pp1,mm1);
      cordic3:da_cordic_3 port map(clk,en_b9,m0,m1,pp2,mm2);
      cordic4:da_cordic_4 port map(clk,en_b9,m2,m3,pp3,mm3);
      cordic5:da_cordic_5 port map(clk,en_b10,mm4,mm5,dy6,dy2);
      cordic6:da_cordic_6 port map(clk,en_b10,pp4,pp5,dy4,dy0);
      add1:add9 port map(en_b10,mm0,mm1,dy1);
      sub1:sub9 port map(en_b10,pp0,pp1,dy7);
      sub2:sub9 port map(en_b10,mm2,pp3,dy5);
      sub3:sub9 port map(en_b10,pp2,mm3,dy3);
      reg:reg10_10 port map(en_reg,dy0,dy1,dy2,dy3,dy4,dy5,dy6,dy7,
                  oy0,oy1,oy2,oy3,oy4,oy5,oy6,oy7);
END structural ;


library ieee;
use ieee.std_logic_1164.all;
entity SERIAL_COMM is
```

```
port( BAUD_RATE:in std_logic;
      RX,en_serial:in std_logic;
      TX_EN: in std_logic;
      DATA_TX:in std_logic_vector(7 downto 0);
      TX:out std_logic;
      DATA_RX:out std_logic_vector(7 downto 0);
      RX_EN:out std_logic);
end SERIAL_COMM;
architecture rtl of SERIAL_COMM is
type State_type_RX is(idel,ReceiveData,Stop);
type State_type_TX is (Idel,Start,TransData,Stop);
signal State_RX:State_type_RX:=Idel;
signal State_TX:State_type_TX;
begin
process(BAUD_RAtE,TX_EN,DATA_TX,en_serial)
      variable TX_Data_Count:integer range 0 to 7;
begin
      if BAUD_RATE'Event and BAUD_RATE='1'then
            if en_serial='1' then
                  case State_TX is
                        when Idel=>
                              TX<='1';
                              if TX_EN='1'then
                                    TX_Data_Count:=0;
                                    State_TX<=Start;
                              Else
                                    TX_Data_Count:=0;
                                    State_TX<=Idel;
                              end if;
                        when Start=>
                              TX<='0';
                              TX_Data_Count:=0;
                              State_TX<=TransData;
                        when TransData=>
                              if TX_Data_Count=7 then
                                    Tx<= Data_TX(TX_Data_Count);
                                    State_TX<=Stop;
                              Else
                                    TX<=DATA_TX(TX_Data_Count);
                                    TX_Data_Count:=TX_Data_Count+1;
                                    State_TX<=TransData;
                              end if;
                        when Stop=>
                              TX<='1';
                              TX_Data_Count:=0;
                              State_TX<=Idel;
                        when others=>
                              TX_Data_Count:=0;
                              TX<='1';
                              State_TX<=Idel;
                  end case;
            else
                  TX<='1';
            end if;
      end if;
end process ;
process(BAUD_RATE,RX,en_serial)
variable RX_Data_Count:integer range 0 to 7;
variable Buffer_RX:std_logic_vector(7 downto 0);
begin
      if BAUD_RATE'Event and BAUD_RATE='1' then
```

```
            if en_serial='1' then
                  case State_RX is
                        when Idel=>
                              RX_EN<='0';
                              if RX='0' then
                                    RX_Data_Count:=0;
                                    State_RX<=ReceiveData;
                              Else
                                    RX_Data_Count:=0;
                                    State_RX<=Idel;
                              end if;
                        when ReceiveData=>
                              RX_EN<='0';
                              if RX_Data_Count=7 then
                                    Buffer_RX(RX_Data_Count):=RX;
                                    State_RX<=Stop;
                              Else
                                    Buffer_RX(RX_Data_Count):=RX;
                                    RX_Data_Count:=RX_Data_Count+1;
                                    State_RX<=ReceiveData;
                              end if;
                        when Stop=>
                              RX_Data_Count:=0;
                              data_rx<=buffer_rx;
                              RX_En<='1';
                              State_RX<=Idel;
                        when others =>
                              RX_Data_Count:=0;
                              RX_En<='0';
                              State_RX<=Idel;
                  end case;
            end if;
      end if;
end process ;
end rtl;


library ieee;
use ieee.std_logic_1164.all;
entity Div1250 is
port( Clk_in:in std_logic;
      Clk_out:out std_logic);
end Div1250;
architecture rtl of Div1250 is
begin
process(Clk_in)
      variable Clk_temp:std_logic:='0';
      variable count:integer range 0 to 624;
begin
      if Clk_in'Event and Clk_in='1'then
            if count<624 then
                  count:=count+1;
                  Clk_temp:=Clk_temp;
            Else
                  count:=0;
                  Clk_temp:=not(Clk_temp);
            end if;
                  Clk_out<=Clk_temp;
      end if;
end process;
end rtl;
```

```
LIBRARY IEEE;
use ieee.std_logic_1164.all;
entity Onepulse is
port( Clk:in std_logic;
      SW:in std_logic;
      SW_Single_Pulse:out std_logic);
end Onepulse;
architecture rtl of Onepulse is
signal sw_debounce_delay:std_logic;
begin
process(Clk)
begin
      if(Clk'Event)and(Clk='1')then
            if(sw='1')and(sw_debounce_delay='0')then
                  sw_single_pulse<='1';
            else
                  sw_single_pulse<='0';
            end if;
            sw_debounce_delay<=sw;
      end if;
end process;
end rtl;

library ieee;
use ieee.std_logic_1164.all;
entity latch64 is
port( en,ld:in std_logic;
   ip:in std_logic_vector(7 downto 0);
   op0,op1,op2,op3,op4,op5,op6,op7:out std_logic_vector(7 downto 0));
end latch64;
architecture rtl of latch64 is
      signal buf0,buf1,buf2,buf3,buf4,buf5,buf6,buf7:
std_logic_vector(7 downto 0);
begin
process(en)
begin
      if en'event and en='1' then
            buf0<=ip;
            buf1<=buf0;
            buf2<=buf1;
            buf3<=buf2;
            buf4<=buf3;
            buf5<=buf4;
            buf6<=buf5;
            buf7<=buf6;
      end if;
end process;
process(ld)
begin
      if ld='1' then
            op0<=buf7;
            op1<=buf6;
            op2<=buf5;
            op3<=buf4;
            op4<=buf3;
            op5<=buf2;
            op6<=buf1;
            op7<=buf0;
      end if;
end process;
end rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity mux10 is
port(ip1,ip2,ip3,ip4,ip5,ip6,ip7,ip8:in std_logic_vector(9 downto 0);
     op:out std_logic_vector(7 downto 0);
     s:in std_logic_vector(3 downto 0));
end;
architecture rtl of mux10 is
      SIGNAL buf1,buf2,buf3,buf4,buf5,buf6,buf7,buf8,buf9,buf10 :
std_logic_vector(7 downto 0);
begin
      buf1(7 downto 0) <= ip1(9 downto 2);
      buf2(7 downto 0) <= ip2(9 downto 2);
      buf3(7 downto 0) <= ip3(9 downto 2);
      buf4(7 downto 0) <= ip4(9 downto 2);
      buf5(7 downto 0) <= ip5(9 downto 2);
      buf6(7 downto 0) <= ip6(9 downto 2);
      buf7(7 downto 0) <= ip7(9 downto 2);
      buf8(7 downto 0) <= ip8(9 downto 2);
      buf9(7 downto 0) <= ip1(1 downto 0)&ip2(1 downto 0)
            &ip3(1      downto 0)&ip4(1 downto 0);
      buf10(7 downto 0) <= ip5(1 downto 0)&ip6(1 downto 0)
            &ip7(1 downto 0)&ip8(1 downto 0);
process(s)
begin
      case s is
            when "0000" =>
                  op <= buf1 ;
            when "0001" =>
                  op <= buf2 ;
            when "0010" =>
                  op <= buf3 ;
            when "0011" =>
                  op <= buf4 ;
            when "0100" =>
                  op <= buf5 ;
            when "0101" =>
                  op <= buf6 ;
            when "0110" =>
                  op <= buf7 ;
            when "0111" =>
                  op <= buf8 ;
            when "1000" =>
                  op <= buf9 ;
            when "1001" =>
                  op <= buf10 ;
            when others => op(7 downto 0) <= "00000000" ;
      end case;
end process;
end;


Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
entity      control_txrx is
port( sys_clk : in std_logic;
      reset,rx_en,cts : in std_logic;
      rts,en_serial : out std_logic;
      ld_latch : out std_logic;
```

```
      tx_en : out std_logic;
      addr : out std_logic_vector(3 downto 0));
end;
architecture behave of control_txrx is
begin
process(sys_clk)
      VARIABLE state : std_logic_vector(3 downto 0);
      VARIABLE count : integer range 0 to 24960;
      VARIABLE delay : integer range 0 to 64;
      VARIABLE count_latch : std_logic_vector(3 downto 0);
      VARIABLE count_addr      : std_logic_vector(3 downto 0);
      VARIABLE buf_addr  : std_logic_vector(3 downto 0);
begin
      if reset = '0' then
            ld_latch<='0';
            tx_en<='0';
            count:=0;
            rts<='1';
            addr<="0000";
            buf_addr:="0000";
            state:="0000";
      else
           if (sys_clk'event and sys_clk ='1') then
               case state is
               when "0000" =>
                     rts<='0';
                     en_serial<='1';
                     state:="0001";
               when "0001" =>
                     if rx_en='1' then
                           rts<='1';
                           en_serial<='0';
                           count_latch:=count_latch+"0001";
                           state:="0010";
                     else
                           state := "0001";
                     end if;
               when "0010" =>
                     if count_latch="1000" then
                           count_latch:="0000";
                           ld_latch<='1';
                           state:="0011";
                     ELSE
                           state := "0000";
                     end if;
               when "0011" =>
                     ld_latch<='0';
                     state:="0100";

               when "0100" =>
                     if delay<=64 then
                           delay:=delay+1;
                           state:="0100";
                     else
                           delay:=0;
                           state := "0101";
                     end if;
               when "0101" =>
                     if count_addr<"1010" then
                           addr<=buf_addr;
                           state:="0110";
```

```
                        else
                              count_addr:="0000";
                              buf_addr:="0000";
                              state := "1001";
                        end if;
                  when "0110" =>
                        if cts='0' then
                              en_serial<='1';
                              state:="0111";
                              else
                              en_serial<='0';
                              state:="0110";
                              end if;
                  when "0111" =>
                        if count<1872 then --29997 then
                           count:=count+1;
                              tx_en<='1';
                        state:="0111";
                        Else
                           tx_en<='0';
                     count:=0;
                     state:="1000";
                        end if;
                  when "1000" =>
                        if count<24960 then --399960 then
                           count:=count+1;
                           state:="1000";
                        Else
                           en_serial<='0';
                           count:=0;
                           count_addr:=count_addr+1;
                           buf_addr:=buf_addr+"0001";
                           state:="0101";
                        end if;
                   when "1001" =>
                        ld_latch<='0';
                        tx_en<='0';
                        count:=0;
                        rts<='1';
                        addr<="0000";
                        buf_addr:="0000";
                        state:="0000";
                  when others =>
                        state :="0000";
            end case;
      end if;
   end if;
end process ;
end behave;
```

กิตติกรรมประกาศ

รายงานฉบับนี้สำเร็จลุล่วงได้ดีด้วยได้รับความช่วยเหลือ และชี้แนะจากหลายท่าน ผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษา อาจารย์อัครพล ตรีรัตน์ ที่ให้คำปรึกษาและความช่วยเหลือด้านข้อมูลและอุปกรณ์ในการทำโครงงาน ขอขอบคุณ อาจารย์ศรวัฒน์ ชิวปรีชา ที่ให้คำปรึกษาและให้ความช่วยเหลือในด้านการใช้งานเอฟพีจีเอ และให้ความช่วยเหลือด้านข้อมูลและตำราต่างๆ มาโดยตลอด ผู้เขียนพึงระลึกอยู่เสมอว่ารายงานฉบับนี้จะไม่สามารถสำเร็จลงได้เลย หากขาดความช่วยเหลือจากทุกท่านจึงขอขอบพระคุณมาอย่างสูง

นายนริศร รณรงค์ฤทธิ์
นายประยุทธ ผาโคตร
นายสนธิ วิมลศิลปวิญญู

หนังสืออ้างอิง

[1] นอ.ชาติชาย ดิษฐกุล, "เอกสารประกอบการเรียนภาษา VHDL"

[2] บริษัท แอสทรอน ลอจิก รีเสิร์ชแอนด์ดิวิลอปเมนต์ จำกัด, "เปิดโลก FPGA กับ WIZARD PLD – AO1," 2544

[3] มนัส สังวรศิลป์, วรรัตน์ ภัทรอมรกุล, "คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์", ศูนย์การพิมพ์พลชัย, อินโฟเพรส, กรุงเทพฯ 2543

[4] D.L. Perry, "VHDL," New York: McGraw - Hill, 1995

[5] G.L.Haviland, A.A.tuszynski,"A CORDIC Arithmetic Processor Chip",IEEE Journal of Solid-State Circuits,Vol. SC-15,NO. 1, February 19800

[6] K.K.Parhi,"VLSI Digital Signal Processing Systems",A Wiley-Interscience Publication,John Wiley & Sons, INC., 1999

[7] M.J.Irwin, "CSE 575 Computer Arithmetic",MJIrwin,PSU,2002

[8] P.Pirsch, "Architectures for Digital Signal Processing",John Wiley & Sons,1998

[9] R.Andraka, "A survey of CORDIC algorithms for FPGA based computers",In Proceedings of the 1998 ACMISIGDA 6th international symposium on FPGA,pages 191-200,Monterey,CA

[10] S.Sjoholm, L.Lindh, "VHDL for Designer", Prentic Hall, 1997

[11] S.Brown, Z.Vranesic, "Fundamentals of DIGITAL LOGIC with VHDL design", McGraw-Hill Book 2000

[12] W.Phasamak, P.Thitimajshima, Y.Rangsanseri, "Architectures Design of Two Dimensional Discrete Cosine Transform"

[13] Y.Yang, C.Wang, M.O.Ahmad, M.N.S.Swamy, "An On-Line CORDIC Based 2-D IDCT Implementation using Distributed Arithmetic", IEEE, 2001