การสร้างวงจรการแปลงดิสครีตโคไซน์บนเอฟพีจีเอ

IMPLEMENTRATION OF DISCRETE COSINE TRANSFORM (DCT) ON FPGA

โดย

นายพิพัฒน์ เอื้อมธุรพจน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

ปีการศึกษา 2545

การสร้างวงจรการแปลงดิสครีตโคไซน์บนเอฟพีจีเอ

IMPLEMENTRATION OF DISCRETE COSINE TRANSFORM (DCT) ON FPGA

โดย

นายพิพัฒน์ เอื้อมธุรพจน์ รหัสนักศึกษา 42010235

อาจารย์ที่ปรึกษา

ศาสตราจารย์ ดร. วัลลภ สุระกำพลธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เรื่อง การสร้างวงจรการแปลงดิสครีตโคไซน์บนเอฟพีจีเอ

Implementation of Discrete Cosine Transform (DCT) on FPGA

วัตถุประสงค์

1. เพื่อศึกษาหลักการและกระบวนการแปลงสัญญาณ แบบ DCT
2. เพื่อศึกษาและประยุกต์ใช้งานอุปกรณ์โปรแกรมได้ เอฟพีจีเอ (FPGA)
3. เพื่อศึกษาและเขียนชุดคำสั่งภาษา VHDL
4. เพื่อนำความรู้ที่ได้ศึกษามาประยุกต์ใช้กับการทำงานจริง

ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถเข้าใจการแปลงสัญญาณ แบบ DCT
2. สามารถประยุกต์ใช้งานอุปกรณ์โปรแกรมได้ เอฟพีจีเอ(FPGA)
3. สามารถเขียนชุดคำสั่งภาษา VHDL ได้
4. สามารถนำความรู้มาประยุกต์ใช้งานจริงได้

ปริญญานิพนธ์ ปีการศึกษา 2545

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การสร้างวงจรการแปลงดิสครีตโคไซน์บนเอฟพีจีเอ

ผู้จัดทำ นายพิพัฒน์ เอื้อมธุรพจน์ รหัสนักศึกษา 42010235

..... อาจารย์ที่ปรึกษา

(ผ.ศ.วสันต์ สถาพล)

การสร้างวงจรการแปลงดิสครีตโคไซน์บนเอฟพีจีเอ

IMPLEMENTRATION OF DISCRETE COSINE TRANSFORM (DCT) ON FPGA

นายพิพัฒน์ เอื้อมธุรพจน์ รหัสนักศึกษา 42010235

โครงงานนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

..... อาจารย์ที่ปรึกษา

(ศาสตราจารย์ ดร. วัลลภ สุระกำพลธร)

การสร้างวงจรการแปลงดิสครีตโคไซน์ในเอฟพีจีเอ

นายพิพัฒน์ เอื้อมธุรพจน์ รหัส 42010235
อ.วัลลภ สุระกำพลธร (อาจารย์ที่ปรึกษา)
ปีการศึกษา 2545

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอการนำทฤษฎีการแปลงดิสครีตโคไซน์มาใช้ในการบีบอัดสัญญาณ โดยใช้อุปกรณ์โปรแกรมได้ เอฟพีจีเอ (FPGA) และภาษาบรรยายอุปกรณ์ วีเฮชดีแอล (VHDL) ในการสร้างวงจรการแปลงดิสครีตโคไซน์ โดยสามารถที่จะนำไปประยุกต์ใช้ได้กับการบีบอัดสัญญาณข้อมูลเชิงเลข หรือสื่อสัญญาณได้

Implementation of Discrete Cosine Transform (DCT) on FPGA

Mr. Pipat Eumthurapot I.D. 42010235
Prof. Wanlop Surakamthon (Advisor)
Academic 2002

ABSTRACT

This thesis presents an implementation of Discrete Cosine Transform (DCT) theorem. In this implementation used Programmable Logic Device FPGA and hardware description language VHDL to implement in signal compression. This project can be applied to use in compressing a digital data or in communication.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ ประสบความสำเร็จและเป็นไปด้วยดีนั้น ทางผู้จัดทำขอขอบคุณ พี่ๆ ทุกคนที่ทำงานอยู่ในห้อง Lab เดียวกัน ที่คอยสอนสิ่งต่าง ๆ ให้ยืมอุปกรณ์ และหนังสือ และให้คำปรึกษาในเรื่องงาน มาว่าจะเป็นทางด้าน Hardware หรือ Software ก็ตาม และที่สำคัญขอขอบคุณ อาจารย์วัลลภ สุระกำพลธร ที่ให้ปรึกษาเกี่ยวกับโครงงานนี้

แม้ว่าผู้จัดทำอาจมีปัญหาในการทำงานก็ตาม แต่ด้วยความช่วยเหลือของทุกท่านที่ได้กล่าวไว้ทำให้การจัดทำปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมา และความสำคัญของโครงงาน

ในปัจจุบัน ความก้าวหน้าทางเทคโนโลยี โดยเฉพาะในวงการอิเล็กทรอนิกส์นับว่ามีการพัฒนาไปอย่างรวดเร็ว ทั้งนี้เนื่องจากการพัฒนาทางด้านกายภาพของอุปกรณ์สารกึ่งตัวนำให้มีคุณสมบัติสำคัญ ที่มีคุณลักษณะการทำงานได้ตามความต้องการ รวมทั้งยังผนวกการเขียนโปรแกรมเพื่อควบคุมการทำงานของอุปกรณ์เหล่านั้นให้มีความสามารถทำงานในการทำงานที่ซับซ้อนได้ เช่นการประมวลผลสัญญาณซึ่งเหมาะเป็นอย่างมากที่จะใช้อุปกรณ์โปรแกรมได้จำพวก FPGA นำมาเป็นส่วนประมวลผลสัญญาณนั้น ๆ โดยในโครงงานนี้ได้ใช้การบีบอัดสัญญาณโดยกระบวนการ DCT ซึ่งในปัจจุบันถือว่าเป็นการประมวลผลสัญญาณทางดิจิตอลที่กำลังได้รับความนิยม ทั้งการใช้งานและการพัฒนาอย่างกว้างขวาง ซึ่งทางผู้จัดทำ ได้เล็งเห็นถึงความสำคัญในการเรียนรู้กระบวนการในจุดนี้ จึงได้นำเรื่องนี้มาทำการศึกษา ดังในปริญญานิพนธ์ฉบับนี้

1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังนี้

1. ใช้อุปกรณ์โปรแกรมได้ FPGA ทำหน้าที่ในส่วนของการประมวลผลสัญญาณ
2. ในส่วนของการประมวลผลสัญญาณจะมีส่วน DCT
3. ใช้ประมวลผลสัญญาณขนาด 8 บิต
4. มีส่วนการแปลงสัญญาณ แอนาลอก เป็นสัญญาณ ดิจิตอล

1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในรายงานฉบับนี้แบ่งออกเป็นบทต่าง ๆ เพื่อสะดวกต่อการศึกษา และทำความเข้าใจ ซึ่งเนื้อหาดังกล่าวประกอบด้วย

บทที่ 2 ทฤษฎีและหลักการ ประกอบไปด้วยเนื้อหาดังนี้ คือ ทฤษฎีการแปลงสัญญาณ , การแปลงดิสครีตโคไซน์ , ทฤษฎีและหลักการของ FPGA , ทฤษฎีและหลักการของภาษา VHDL , การแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล และทฤษฎีการสุ่มสัญญาณ

บทที่ 3 การออกแบบ การสร้าง และการทำงาน กล่าวถึงวิธีการสร้างในส่วนต่าง ๆ ของโครงงาน ได้แก่ การสร้างส่วนวงจรการแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล , วงจรภาคจ่ายไฟ , การคำนวณการหาค่าสัมประสิทธิ์ของการแปลง DCT , การเขียนวงจร Serial In Parallel Out , การเขียนวงจร Parallel In Serial Out , การเขียนวงจรการแปลง DCT

บทที่ 4 การทดลองการ และผลการทดลอง ประกอบไปด้วยการทดลองการทำงานของโครงงานนี้ ดังนี้ การเปรียบเทียบค่าเอาท์พุตของวงจร DCT ที่ได้กับค่าที่ได้จากโปรแกรม MATLAB , การจำลองการทำงานของโปรแกรมการส่วน Serial In Parallel Out , การจำลองการทำงานของโปรแกรมการส่วน Parallel In Serial Out , การจำลองการทำงานของโปรแกรมการส่วน DCT

บทที่ 5 บทสรุป แนวทางแก้ไข และการพัฒนา ได้รวบรวมปัญหาที่เกิดขึ้นในขั้นตอนต่าง ๆ ระหว่างจัดทำโครงงาน และได้เสนอแนวทางการแก้ไขปัญหา และการพัฒนาโครงงานให้มีประสิทธิภาพมากขึ้น

ภาคผนวก ก โปรแกรม MATLAB ที่ใช้ทดสอบ Algorithm DCT

ภาคผนวก ข โปรแกรมการทำงาน

บทที่ 2

ทฤษฎีและหลักการ

2.1 กล่าวนำ

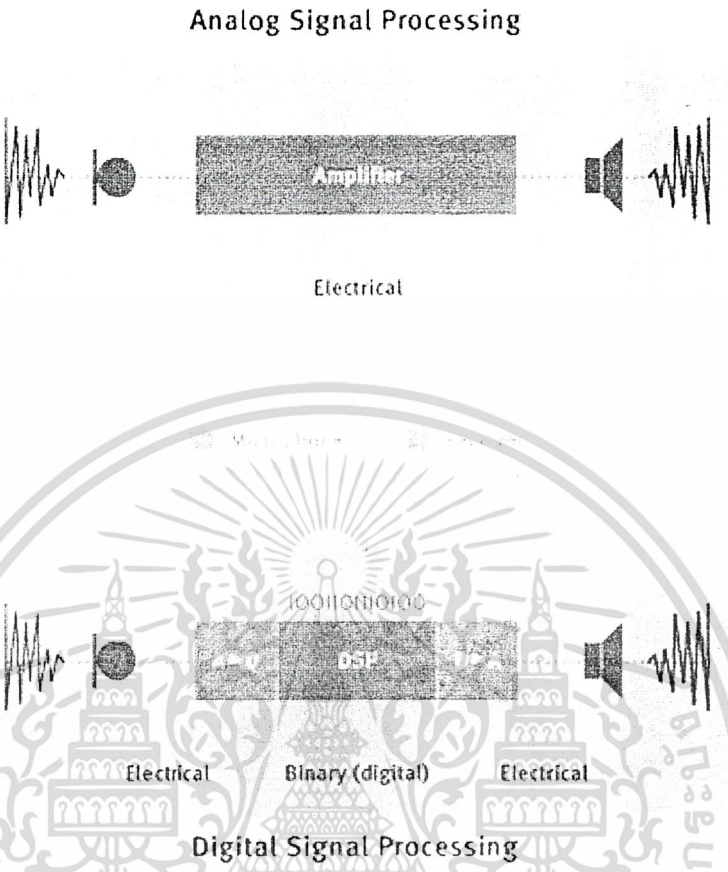
การแปลงดีสครีตโคไซน์ (Discrete Cosine Transform : DCT) เป็นหนึ่งในกระบวนการประมวลผลสัญญาณเชิงเลข (Digital Signal Processing : DSP) ซึ่งเป็นการแปลงสัญญาณเชิงเลขที่อยู่ในรูปแบบโดเมนของเวลา ให้อยู่ในโดเมนของโคไซน์ ซึ่งผลที่ได้จากการแปลงโดเมนจะทำให้ลดขนาดของข้อมูลได้ ซึ่งเป็นกระบวนการในการบีบอัดสัญญาณ (Signal Compressing) ซึ่งสามารถแปลงกลับค่าโดเมนของโคไซน์ โดยใช้การแปลงกลับดีสครีตโคไซน์ (Inverse Discrete Cosine Transform :IDCT) ทำให้ได้ข้อมูลกลับมาเช่นเดิม

2.2 ทฤษฎีการแปลงสัญญาณ

รูปแบบของสัญญาณไฟฟ้าโดยส่วนมากมักอยู่ในรูปสัญญาณแอนาลอก การนำเอาสัญญาณไฟฟ้ามาประมวลผล เพื่อให้เกิดรูปแบบที่ต้องการนั้น ต้องใช้อุปกรณ์ทางแอนาลอกแต่ปัจจุบันนี้เทคโนโลยีทางดิจิตอลก้าวหน้าไปมาก ทำให้การประมวลผลสัญญาณทางดิจิตอล สามารถทำได้อย่างรวดเร็ว และมีประสิทธิภาพ

ดังนั้นการแปลงรูปแบบของสัญญาณ (Conversion) จึงมีความจำเป็นในการแปลงสัญญาณแอนาลอกที่มีอยู่แล้วให้เป็นสัญญาณดิจิตอลโดยอุปกรณ์การแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลและจะถูกประมวลผลโดยตัวประมวลผลสัญญาณทางดิจิตอล เช่น คอมพิวเตอร์ เป็นต้น จากผลลัพธ์ที่ได้อาจถูกนำมาแสดงผลโดยตรงเลย หรืออาจถูกแปลงกลับมาเป็นสัญญาณแอนาลอกที่ใช้งานได้ การที่แปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลได้นั้นสามารถทำได้โดยใช้อุปกรณ์แปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก สำหรับระบบที่มีการประมวลผลข้อมูลทางดิจิตอลแสดงดังรูปที่ 2.1

จากรูปที่ 2.1 การเปลี่ยนแปลงทางกายภาพในลักษณะใด ๆ ก็ตาม เช่น อุณหภูมิ แรงดัน ความเร็ว จะถูกเปลี่ยนให้เป็นสัญญาณไฟฟ้าแบบแอนาลอก โดยทรานสดิวเซอร์ (Transducer) ทฤษฎีการสุ่มที่มีรูปแบบที่เหมาะสมกับลักษณะทางกายภาพนั้น ๆ จากสัญญาณทางไฟฟ้าก็จะถูกปรับให้อยู่ในรูปแบบ และขนาดที่เหมาะสมก่อน โดยวงจรต่าง ๆ เช่น วงจรขยาย หรือวงจรกรองสัญญาณ เป็นต้น



รูปที่ 2.1 รูปแบบการประมวลผลทางแอนาลอกและทางดิจิตอล

2.3 การแปลงดีสครีตโคไซน์ (Discrete Cosine Transform : DCT)

2.3.1 แนะนำ DCT

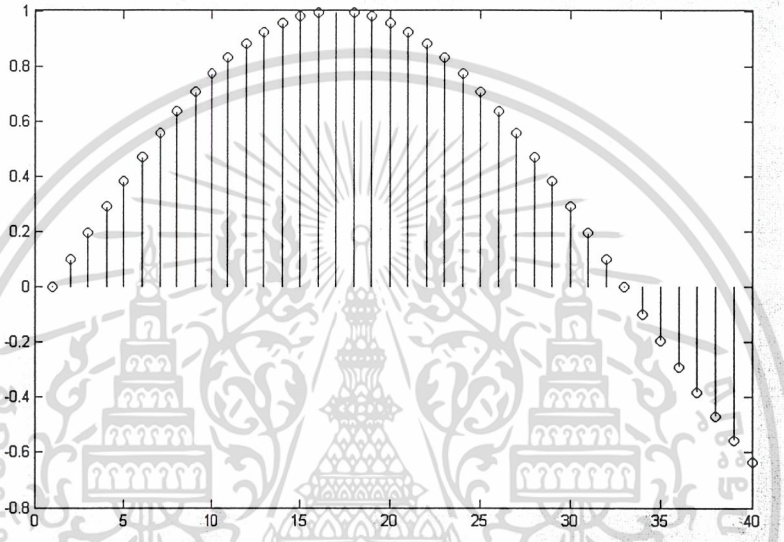
การแปลงฟูริเยอร์เต็มหน่วย (Discrete Fourier Transform : DFT) และการแปลงดีสครีตโคไซน์ (Discrete Cosine Transform : DCT) สามารถนำมาประยุกต์ใช้ในการประมวลผลสัญญาณเชิงเลข (Digital Signal Processing : DSP) เช่นในการประมวลผลภาพ การวิเคราะห์พลังงานของระยะคลื่น การกรองสัญญาณ เป็นต้น

ขั้นตอนวิธีของ DFT ที่เร็ว มีประสิทธิภาพและเป็นที่รู้จักคือ การแปลงฟูริเยอร์อย่างเร็ว (Fast Fourier Transform : FFT) ซึ่งถูกเสนอโดย Cooley และ Tukey ในปี ค.ศ. 1965 ใช้ในการคำนวณหาผลของการทำ DFT

DCT เริ่มแรกถูกค้นพบโดย Ahmed Natarajan และ Rao ในปี ค.ศ. 1974 ซึ่งถูกจัดทำขึ้นเพื่อใช้ในการบีบอัดสัญญาณในการประมวลผลเชิงเลข ขั้นตอนวิธี และการสร้าง DCT อยู่บน

พื้นฐานของ FFT ซึ่งเป็นการแปลงสัญญาณเชิงเลขที่อยู่ในโดเมนของเวลา ให้อยู่ในรูปโดเมนของความถี่ ซึ่งผลของการแปลงโดเมนจะทำให้ลดขนาดของข้อมูล ซึ่งเป็นกระบวนการในการบีบอัดสัญญาณ ซึ่งสามารถแปลงกลับค่าโดเมนของโคไซน์ (Inverse Discrete Cosine Transform : IDCT) ทำให้ได้ข้อมูลกลับเช่นเดิม

สัญญาณในโดเมนเวลา



รูปที่ 2.2 สัญญาณในโดเมนเวลา

DCT

สัญญาณในโดเมนโคไซน์

รูปที่ 2.3 สัญญาณในโดเมนโคไซน์

2.3.2 ขั้นตอนวิธีของ DCT

DCT N จุดสามารถอธิบายได้ตามสมาการที่ 2.1 :

$$y(k) = c(k)\sum_{n=0}^{N-1}\cos\frac{2\pi k(2n+1)}{4N}x(n) \qquad (2.1)$$

โดย y (k) เป็นค่าสัมประสิทธิ์ในการแปลง

x (n) เป็นข้อมูลเริ่มต้น

$c(0) = 1/\sqrt{N}$ และ $c(k) = \sqrt{(2/N)}$, $1 \le k \le N-1$

ขั้นตอนวิธีของ DCT ต้องการเพียงเลขจำนวนจริงเท่านั้นในการคำนวณ ซึ่งจะเร็วกว่าการคำนวณโดยใช้เลขเชิงซ้อน (complex arithmetic) ใน FFT

DCT N-จุด สามารถนิยามได้ดังนี้

$$y(k) = c(k)\sum_{n=0}^{N-1}\cos\frac{2\pi k(2n+1)}{4N}x(n)$$

เมื่อ $c(0) = 1/\sqrt{N}$, และ $c(k) = \sqrt{(2/N)}$, $1 \le k \le N-1$ส่วน inverse DCT สามารถเขียนได้ว่า :

$$x(n) = \sum_{k=0}^{N-1}\cos\frac{2\pi k(2n+1)}{4N}c(k)y(k)$$

สำหรับกรณี N = 8, DCT สามารถเขียน a matrix vector product ได้ว่า

$$\mathbf{y} = \mathbf{C}_8\,\mathbf{x}$$

matrixขนาด 8*8 ของ $\mathbf{C}_8$ สามารถแยกแฟกเตอร์เป็นเทอมของ matrix 3 ตัวคูณกันได้เป็น:

$$\mathbf{C}_8 = \mathbf{P}_8\,\mathbf{K}_8\,\mathbf{B}$$

เมื่อ P_8 คือ permutation matrix

$$P_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

K_8 คือ block diagonal matrix

$$K_8 = \frac{1}{2}\begin{bmatrix} G_1 & & & \\ & G_1 & & \\ & & G_2 & \\ & & & G_4 \end{bmatrix}$$

โดยที่ $G_1 = \cos(\pi/4)$, $G_2 = \begin{bmatrix} \cos(3\pi/8) & \cos(\pi/8) \\ -\cos(\pi/8) & \cos(3\pi/8) \end{bmatrix}$, และ

$$G_4 = \begin{bmatrix} \cos(5\pi/16) & \cos(9\pi/16) & \cos(3\pi/16) & \cos(\pi/16) \\ -\cos(\pi/16) & \cos(5\pi/16) & \cos(9\pi/16) & \cos(3\pi/16) \\ -\cos(3\pi/16) & -\cos(\pi/16) & \cos(5\pi/16) & \cos(9\pi/16) \\ -\cos(9\pi/16) & -\cos(3\pi/16) & -\cos(\pi/16) & \cos(5\pi/16) \end{bmatrix}$$

เป็น anti-circulant matrix.

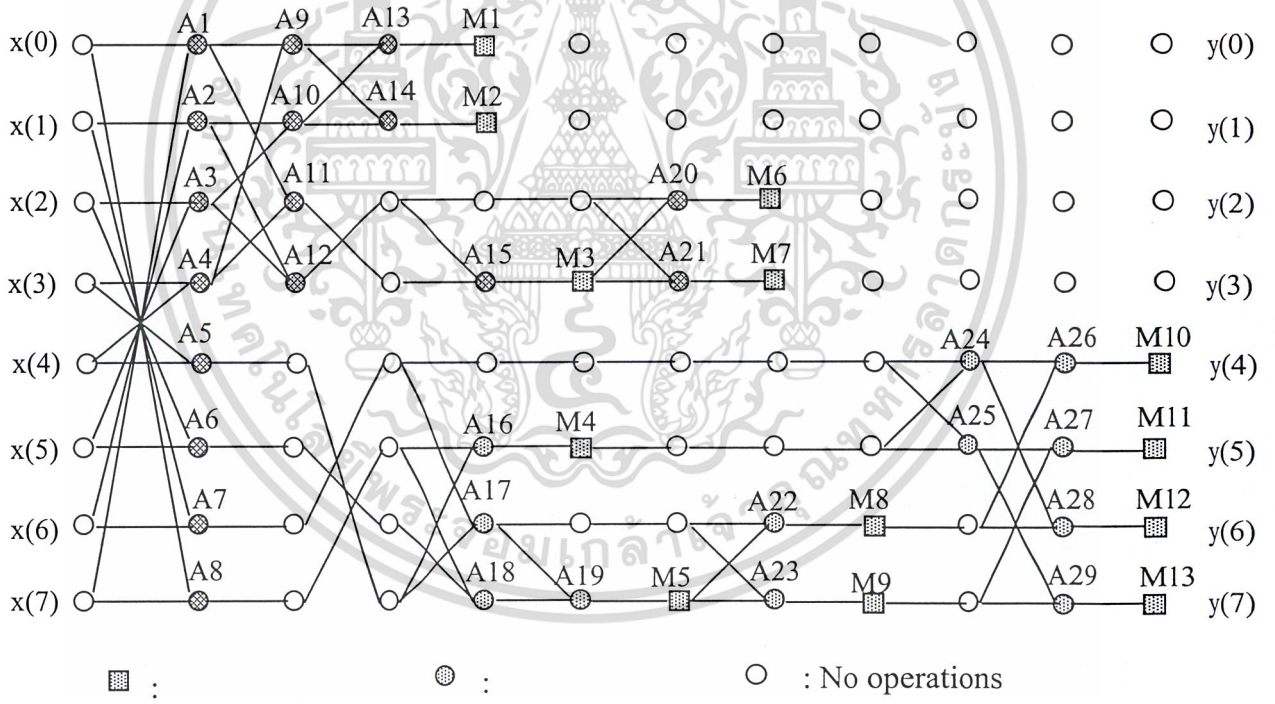
B สามารถแยกแฟกเตอร์เป็นเทอมผมคูณของ 3 matrices ได้เป็น : $B = B_1B_2B_3$ เมื่อ

$$B_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}$$

โดยหลักพื้นฐานของการแยกแฟกเตอร์, Feig and Winograd [Feig, 1992 #150] ได้เสนอวิธีการทำ DCT 8-point ซึ่งต้องการทำการคูณ 13 ครั้ง และการบวก 29 แสดงได้ดังนี้



รูปที่ 2.4 Dependence graph of fast DCT algorithm

การคูณของ matrix B จะต้องการ การบวกเท่ากับ 2+4+8 = 14: โดยเราต้องทำการคูณ [x(0) .. x(7)] ด้วย B_3 ก่อน:

$x(0,1) = x(0) + x(7)$ $\qquad$ $x(4,1) = x(0) - x(7)$

$x(1,1) = x(1) + x(6)$ $x(5,1) = x(1) - x(6)$

$x(2,1) = x(2) + x(5)$ $x(6,1) = x(2) - x(5)$

$x(3,1) = x(3) + x(4)$ $x(7,1) = x(3) - x(4)$

ถัดมา, ผลลัพธ์ที่ได้จะคูณกับ matrix B_2:

$x(0,2) = x(0,1) + x(3,1),$ $x(4,2) = x(4,1)$

$x(1,2) = x(1,1) + x(2,1),$ $x(5,2) = x(5,1)$

$x(2,2) = x(0,1) - x(3,1),$ $x(6,2) = x(6,1)$

$x(3,2) = x(1,1) - x(2,1),$ $x(7,2) = x(7,1)$

สุดท้าย, คูณกับ matrix B_1:

$x(0,3) = x(0,2) + x(1,2);$ $x(4,3) = -x(6,2)$

$x(1,3) = x(0,2) - x(1,2);$ $x(5,3) = x(7,2)$

$x(2,3) = x(3,2);$ $x(6,3) = -x(5,2)$

$x(3,3) = x(2,2);$ $x(7,3) = -x(4,2)$

ต่อมาเราจะศึกษาว่าการคูณอย่างไรของผลลัพธ์ [x(0:7,3)] กับ matrix G_1, G_2, and G_4 อันดับแรกก็คือ

$x(0,4) = (1/2)\cos(\pi/4)*x(0,3)$ $x(1,4) = (1/2)\cos(\pi/4)*x(1,3)$

ต่อมาคือ

$$G_2 = \frac{1}{2}\begin{bmatrix}\cos(3\pi/8) & \\ & \cos(\pi/8)\end{bmatrix}^{-1}\begin{bmatrix}1 & 1\\ -1 & 1\end{bmatrix}\begin{bmatrix}1 & \\ & \cos(\pi/4)\end{bmatrix}\begin{bmatrix}1 & 0\\ -1 & 1\end{bmatrix}$$

$$= \begin{bmatrix}\cos(3\pi/8) & \sin(3\pi/8)\\ -\sin(3\pi/8) & \cos(3\pi/8)\end{bmatrix}$$

ดังนั้นการคูณกับ G_2 สามารถทำได้ด้วยทำการคูณ 3 ครั้งและทำการบวก 3 ครั้ง แทนที่จะเป็นการทำการคูณ 4 ครั้ง และทำการบวก 2 ครั้ง นอกจากนี้ G_2 เป็นการหมุน matrix, ซึ่งพีชคณิต CORDIC สามารถนำมาใช้คำนวณผลลัพธ์ได้สะดวก

tmp = cos(π/4)*(—x(2,3)+x(3,3));

x(2,4) = (1/4)(1/cos(3π/8))*(x(2,3) + tmp);

x(3,4) = (1/4)(1/cos(π/8))*(—x(2,3) + tmp);

ตอนนี้ G_4 สามารถแยกแฟกเตอร์ได้ดังนี้:

$$G_4 = \frac{1}{2}\Gamma^{-1}\begin{bmatrix} 1 & 1 & -1 & 0 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 0 \\ 1 & -1 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & & & \\ & G_1 & & \\ & & G_2 & \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

เมื่อ Γ = diag[cos (5π/16), cos(π/16), cos(3π/16), cos(7π/16)]. ดังนั้นการคูณ G_4 จะต้องการการคูณอย่างน้อย 8 ครั้ง (1 สำหรับ G_1, 3 สำหรับ G_2, 4 สำหรับ Γ^{-1}), และทำการบวก 12 ครั้ง

tmp(4) = x(4,3);

tmp(5) = x(5,3) + x(7,3)

tmp(6) = x(4,3) — x(7,3);

tmp(7) = x(5,3) — x(6,3)

tmp(4,1) = tmp(4);

tmp(5,1) = cos(π/4)*tmp(5)

tmp = cos(π/4)*(—tmp(6) + tmp(7));

tmp(6,1) = (1/2)(1/cos(3π/8))*(tmp(6) + tmp);

tmp(7,1) = (1/2)(1/cos(π/8))*(—tmp(6) + tmp);

x(4,4) = (1/4) (1/cos(5π/16))*(tmp(4,1)+tmp(5,1)—tmp(6,1));

x(5,4) = (1/4) (1/cos(π/16))*(—tmp(4,1)+tmp(5,1)+tmp(7,1));

x(6,4) = (1/4) (1/cos(3π/16))*(—tmp(4,1)—tmp(5,1)—tmp(6,1));

x(7,4) = (1/4) (1/cos(7π/16))*(tmp(4,1)—tmp(5,1)+tmp(7,1));

สุดท้ายจะได้ผลลัพธ์เป็น

y(0) = x(0,4);

y(4) = x(1,4);

y(1) = —x(4,4);

y(5) = —x(7,4);

y(2) = x(2,4);

y(6) = x(3,4);

y(3) = —x(5,4);

y(7) = x(6,4);

นอกจากนี้แล้วการทำ DCT สามารถที่จะแยกเป็นประเภทย่อยๆได้อีก 4 ชนิด **(DCT type I-IV)** ซึ่งได้ปรับปรุงมา แต่รูปแบบก็จะคล้ายกับ DCT-II ซึ่งนิยมใช้กันกว้างขวาง รูปแบบของฟังก์ชั่นทั้ง 4 ชนิดสามารถนิยามได้ตามสมการต่อไปนี้

กำหนดให้

$$b(k)=\begin{cases}1 & ;k=1,2,...,N-1\\ \dfrac{1}{\sqrt{2}} & ;k=0,N\end{cases}$$

DCT-I :

$$X^{C(1)}(k)=\left(\frac{2}{N}\right)^{1/2} b(k)\sum_{n=0}^{N} b(n)x(n)\cos\left(kn\frac{\pi}{N}\right) \quad ; \text{k = 0, 1, 2, ...N}$$

DCT-II :

$$X^{C(2)}(k)=\left(\frac{2}{N}\right)^{1/2} b(k)\sum_{n=0}^{N-1} x(n)\cos\left((n+\frac{1}{2})k\frac{\pi}{N}\right) \quad ; \text{k = 0, 1, 2, ...N-1}$$

DCT-III :

$$X^{C(3)}(k)=\left(\frac{2}{N}\right)^{1/2} b(k)\sum_{n=0}^{N-1} b(n)x(n)\cos\left((k+\frac{1}{2})n\frac{\pi}{N}\right) \quad ; \text{k = 0, 1, 2, ...N-1}$$

DCT-IV :

$$X^{C(4)}(k)=\left(\frac{2}{N}\right)^{1/2} \sum_{n=0}^{N-1} x(n)\cos\left((n+\frac{1}{2})(k+\frac{1}{2})\frac{\pi}{N}\right) \quad ; \text{k = 0, 1, 2, ...N-1}$$

Inverse DCT-I :

$$x(n)=\left(\frac{2}{N}\right)^{1/2} b(n)\sum_{k=0}^{N} b(k)X^{C(1)}(k)\cos\left(kn\frac{\pi}{N}\right) \quad ; \text{n = 0, 1, 2, ...N}$$

Inverse DCT-II :

$$x(n)=\left(\frac{2}{N}\right)^{1/2} \sum_{k=0}^{N-1} X^{C(2)}(k)b(k)\cos\left((n+\frac{1}{2})k\frac{\pi}{N}\right) \quad ; \text{n = 0, 1, 2, ...N-1}$$

Inverse DCT-III :

$$x(n)=\left(\frac{2}{N}\right)^{1/2} b(n)\sum_{k=0}^{N-1} X^{C(3)}(k)\cos\left((k+\frac{1}{2})n\frac{\pi}{N}\right) \quad ; \text{n = 0, 1, 2, ...N-1}$$

Inverse DCT-IV :

$$x(n)=\left(\frac{2}{N}\right)^{1/2} \sum_{k=0}^{N-1} X^{C(4)}(k)\cos\left((n+\frac{1}{2})(k+\frac{1}{2})\frac{\pi}{N}\right) \quad ; \text{n = 0, 1, 2, ...N-1}$$

ความแตกต่างกันของรูปแบบฟังก์ชั่นภายใน เทอมของ summation และเทอมของเฟสชิฟที่แตกต่างกันจะเป็นตัวกำหนดชนิดของการแปลงDCT ว่าเป็น DCT ชนิดที่ I, II, III หรือ IVสามารถแสดงได้ดังรูปต่อไปนี้

ความสัมพันธ์ของ DCT แต่ละชนิดสามารถสรุปได้ดังต่อไปนี้

$$\left[X^{C(1)}(k)\right]^{-1} = \left[X^{C(1)}(k)\right] \text{ ; Forward = Inverse}$$

$$\left[X^{C(2)}(k)\right]^{-1} = \left[X^{C(3)}(k)\right] = \left[X^{C(2)}(k)\right]^{T}$$

$$\left[X^{C(3)}(k)\right]^{-1} = \left[X^{C(2)}(k)\right] = \left[X^{C(2)}(k)\right]^{T}$$

$$\left[X^{C(4)}(k)\right]^{-1} = \left[X^{C(4)}(k)\right] \text{ ; Forward = Inverse}$$

DCT ชนิด 1-4 สามารถ แยกแฟกเตอร์ย่อยได้โดยพื้นฐานของ decomposition ของ $\left[X^{C(4)}(k)\right]$(DCT ทั้งหมดนี้เป็น orthogonal ซึ่งกันและกัน สามารถแยกจากกันได้ และทั้งหมดมีวิธีการคำนวนแบบเร็วได้ fast- algorithm

ข้อมูลที่ได้หลังจากการหาผลรวมของผลคูณของค่าตั้งต้นกับค่าสัมประสิทธิ์โคไซน์จะมีขนาดของข้อมูลก่อนทำการแปลงค่าโดยที่การแปลงสัญญาณข้อมูลให้อยู่ในรูปของโดเมนโคไซน์จะมีลักษณะเด่นอยู่ 3 ประการ ดังนี้

1. ค่าความถี่ศูนย์จะเป็นค่าของความเข้มเฉลี่ยของข้อมูล
2. ค่าความถี่สูง เป็นค่าที่บอกถึงข้อมูลที่มีการเปลี่ยนแปลงสูง
3. ค่าความถี่ต่ำ เป็นค่าที่บอกรายละเอียดโดยรวมของข้อมูล

2.4 การแปลงกลับดีสครีตโคไซน์ (Inverse Discrete Cosine Transform : IDCT)

เป็นการแปลงสัญญาณที่อยู่ในรูปโดเมนของโคไซน์ให้กลับไปอยู่ในรูปโดเมนของเวลาเช่นเดิม โดยมีสมการการแปลงค่ากลับดังสมการที่ 2.2

$$x(n) = \frac{1}{\sqrt{2N}} \sum_{k=0}^{N-1} c(k)\, y(k) \cos \frac{2\pi k(2n+1)}{4N} \qquad (2.2)$$

โดย x (n) เป็นค่าสัมประสิทธิ์ในการแปลง

y (k) เป็นข้อมูลเริ่มต้น

$c(0) = 1/\sqrt{N}$ และ $c(k) = \sqrt{(2/N)}$, $1 \leq k \leq N-1$

N เป็นจำนวนจุดในการประมวลผล

ซึ่งทั้งในกระบวนการแปลงค่าไปสู่โดเมนของโคไซน์ก็ดี หรือการแปลงค่ากลับไปก็ดีจะมีส่วนของการประมาณค่าของข้อมูลที่ได้ ซึ่งในส่วนนี้เองที่เป็นการสูญเสีย ซึ่งอัตรการสูญเสียจะขึ้นอยู่กับการกระจายของค่าที่นำเข้ามาแปลง โดยจะส่งผลทำให้ค่าหลังจากแปลงกลับนั้นมีความเพี้ยนไปจากข้อมูลนำเข้าบ้าง แต่โดยในภาพรวมของกลุ่มข้อมูลยังถือว่า ข้อมูลยังอยู่ในรูปลักษณะที่เกาะกลุ่มกัน ซึ่งค่าของอัตราการสูญเสียนี้จะขึ้นอยู่กับสาเหตุต่าง ๆ ดังนี้

1. การจัดและการปรับแต่งค่าสัมประสิทธิ์ของโคไซน์

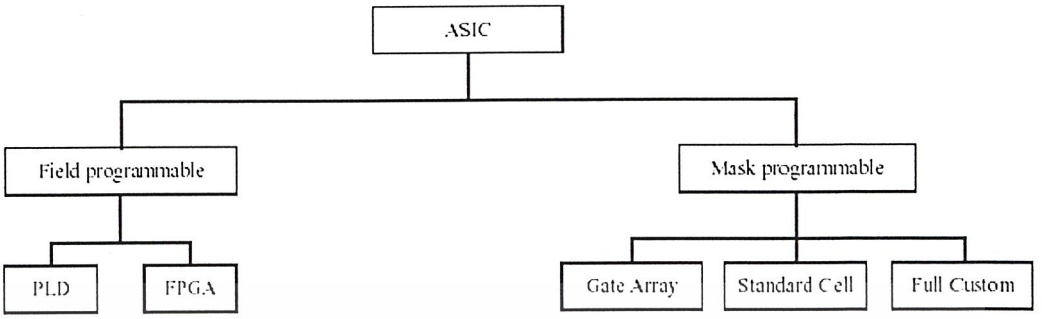
ในการนำค่าสัมประสิทธิ์ของโคไซน์ไปใช้งาน สามารถที่จะปรับแต่งค่าต่าง ๆ ในตารางสัมประสิทธิ์ให้เหมาะสมกับขนาดและกลุ่มของข้อมูล

2. การประมาณค่าผลลัพธ์จากการแปลงค่า

ผลลัพธ์ที่ได้จากการแปลงค่า ในบางครั้งต้องมีการตัดค่าน้อยทิ้ง ซึ่งทำให้ข้อมูลจากทำการแปลงค่ากลับมีความเพี้ยนเกิดขึ้น

2.5 เอฟพีจีเอ : FPGA (Field Programmable Gate Array)

ความก้าวหน้าของอุตสาหกรรมอิเลคทรอนิกส์ปัจจุบันทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่าง ๆ มากมาย ซึ่งทำให้เกิดการลดค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาดในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความ เชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโพรเซสเซอร์และหน่วยความจำปัจจุบัน ทุกๆ ครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวมและไอซีมาตรฐานมากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวนฟังก์ชันลอจิก ที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตให้ขนาดมากๆ และการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวม จะแบ่งตามการสร้างออกเป็น 2 กลุ่ม คือ Field programmable และ Mask programmable ดังแสดงในรูปที่ 2.4



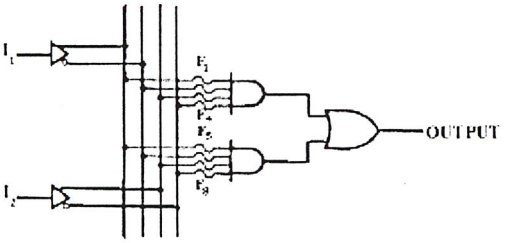
รูปที่ 2.4 ผังแสดงการแบ่งกลุ่มของวงจรรวม ASIC

2.5.1 Field Programmable

อุปกรณ์วงจรรวม ASIC แบบ field programmable มีอยู่มากมายหลายชนิด แต่มีลักษณะการสร้างหรือกำหนดการทำงานของวงจรที่เหมือนกัน กล่าวคือ ผู้ใช้งานสามารถออกแบบและสร้างวงจรที่ต้องการใช้ลงในตัวอุปกรณ์ได้เองโดยไม่ต้องไปโรงงานเพื่อผลิต โดยเฉพาะอย่างยิ่งในปัจจุบันนี้มีเครื่องมือที่ใช้ช่วยในการออกแบบ และสร้างวงจรร่วมกับไมโครคอมพิวเตอร์ที่มีความสามารถสูงในการพัฒนาตั้งแต่ขั้นการออกแบบ การจำลองการทำงาน จนถึงจัดสร้างวงจรลงในอุปกรณ์ รวมทั้งอุปกรณ์ field programmable เหล่านี้สามารถหาซื้อได้ง่าย ทำให้การสร้างวงจรอิเล็กทรอนิกส์จนถึงระบบไมโครโพรเซสเซอร์หันมาใช้อุปกรณ์จำพวกนี้ เป็นอุปกรณ์ประกอบในวงจรแทนอุปกรณ์ย่อยๆ แยกชิ้น (Discrete component) มากขึ้น

2.5.1.1 พีแอลดี (PLD: Programmable Logic Device)

ภายในอุปกรณ์พีแอลดีถูกเตรียมเป็นวงจรพื้นฐานทางด้านลอจิกต่อกันอยู่เป็นกลุ่มมีทั้งวงจรคอมบิเนชั่น (Combination) และแบบซีเควลเชียล (Sequential) ซึ่งมีส่วนประกอบเป็นวงจรภายในเทคโนโลยีของวงจรที่ใช้สร้างพีแอลดีมีทั้ง ทีทีแอล (TTL) อีซีแอล (ECL) และ ซีมอส (CMOS) ตามความเหมาะสมของแต่ละระบบ อุปกรณ์พีแอลดีทุกชนิดมีหลักการพื้นฐานของวงจรภายในที่เหมือนกันโดยมีวงจรหลักเป็นวงจรคอมบิเนชั่นที่ให้ผลเป็นผลคูณร่วมบวก (Sum of product) ประกอบไปด้วยชุดของแอนด์เกตที่ต่อร่วมกับออร์เกตการโปรแกรมคือ การเลือกว่าจะให้มีการต่ออินพุตภายในของแอนด์เกตกับสัญญาณอินพุตใดบ้างซึ่งมีทั้งจากภายนอกและสัญญาณป้อนกลับจากเอาต์พุตภายในเอง การติดต่ออินพุตของออเกตกับเอาต์พุตของ แอนด์เกต ตัวต่างๆ วิธีการเลือกหรือการโปรแกรมทางกายภาพ อินพุตต่าง ๆ ของอุปกรณ์ทุกตัวจะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใดจะตัดฟิวส์ทำให้สามารถโปรแกรมได้ครั้งเดียว อุปกรณ์พีแอลดีบางชนิดใช้มอสทรานซิสเตอร์แทนฟิวส์ทำให้สามารถโปรแกรมโดยใช้กระแสไฟฟ้า และสามารถลบและโปรแกรมใหม่เข้าไปได้อีก



รูปที่ 2.5 แสดงวงจรพื้นฐานของอุปกรณ์พีแอลดีซึ่งอยู่ในรูปผลคูณร่วมบวก

2.5.1.2 เอฟพีจีเอ (FPGA: Field Programmable Gate Array)

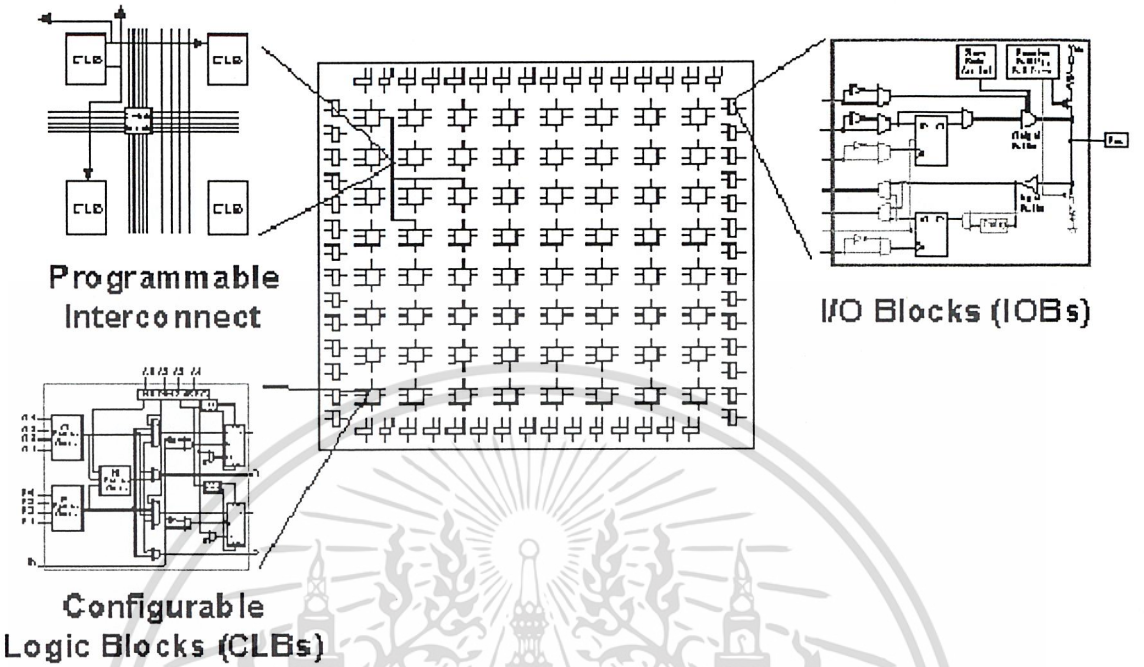
เป็นอุปกรณ์ที่ถูกพัฒนาต่อจากอุปกรณ์แอลซีเอของบริษัทไซลิงซ์ (XILINX Inc.) โดยมีประสิทธิภาพการทำงานและมีปริมาณความหนาแน่นของเกตสูง สามารถจะกำหนดฟังก์ชั่นการทำงานได้ความต้องการของผู้ใช้โดยผ่านการโปรแกรมเอฟพีจีเอได้รวบรวมข้อดีทั้งหมดของการทำคัสตัมวีแอลเอสไอ (Custom VLSI) มารวมไว้ทั้งหมดได้แก่ การออกแบบการผลิต, ลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรเป็นอย่างมาก นักออกแบบเพียงกำหนดฟังก์ชั่นการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้เอฟพีจีเอ สามารถออกแบบและทดสอบภายในเวลาเพียง 2 วัน เท่านั้น ตรงกันข้ามกับการออกแบบโดยใช้เกตอาเรย์ ซึ่งใช้เวลาหลายสัปดาห์ การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกัน

จากประโยชน์ของเอฟพีจีเอ ดังกล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์ ลดค่าเอ็นอาร์อี (NRE: Nonrecurring Engineering Cost) ลงไปด้วย

2.5.2 สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล XC4000

FPGA (Field Programmable Gate Array) เป็นชิพที่มีสนามวงจรขนาดใหญ่อยู่ภายใน ที่เราสามารถนำมาใช้และออกแบบวงจรต่าง ๆ ได้ตามที่เราต้องการ โดยมีโปรแกรมสำเร็จรูปที่ใช้ในการออกแบบอยู่แล้ว มีความเร็วสูง มีสถาปัตยกรรมการออกแบบคล้าย CPLD (Complex Programmable Logic Device) แต่มีส่วนประกอบที่ซับซ้อน และมีประสิทธิภาพมากกว่า ในการออกแบบวงจรสามารถทำได้ง่าย ทั้งในการเชื่อมต่อและการแก้ไข ดังนั้น FPGA จึงเหมาะสมสำหรับการออกแบบวงจรเป็นอย่างมาก สถาปัตยกรรมภายในของ FPGA แบ่งเป็น 3 ส่วน คือ

- CLB (Configuration Logic Block)
- IOB (Input Output Block)
- Interconnect

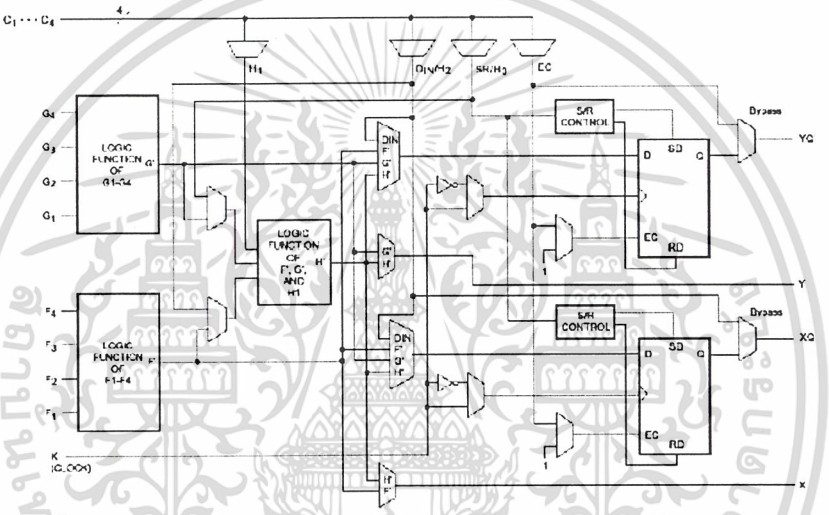


รูปที่ 2.6 แสดงวงจรพื้นฐานภายในไซลิ้งเอฟพีจีเอ

ซึ่งภายในมีสถาปัตยกรรมของ FPGA จะมีลักษณะเป็นตารางของลอจิกบล็อก (Logic block) และล้อมรอบไปด้วยบล็อกการเชื่อมต่อของไอโอ (I/O Interface block) การเชื่อมต่อระหว่างซีแอลบี (CLB: Configuration Logic Block) และไอโอบี (IOB: Input Output Block) ทำได้โดยผ่านช่องที่ว่างพาดผ่านระหว่างแถว (Row) และคอลัมน์ (Column) มีการทำงาน เหมือนกันไมโครโพรเซสเซอร์ ตัวแอลซีเอจะทำงานได้ต้องใช้ Program-driven logic device หน้าที่ของซีแอลบีและไอโอบีแต่ละตัว การเชื่อมต่อภายใน (Interconnection) ถูกกำหนดไว้ในโปรแกรมคอนฟิกคูเรชัน (Configuration program) หรือเก็บไว้ในอีพรอม(EPROM) ภายในแอลซีเอ(LCA : Logic Cell Array) โปรแกรมจะถูกโหลดเข้าสู่แอลซีเอเมื่อมีการจ่ายไฟ (Power-up) โดยทางคำสั่ง (Command) ซึ่งเป็นส่วนหนึ่งของการเริ่มต้นระบบ (System initialization) ประสิทธิภาพของแอลซีเอกำหนดโดยความเร็วของลอจิกส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่าง ๆ ความเร็วของอัตราของระบบสัญญาณนาฬิกา (System clock rate) ถูกกำหนดด้วย ทอกเกิลฟลิปฟลอป สำหรับการประยุกต์ใช้โดยทั่วไปจะอยู่ที่ประมาณ 1/3 ถึง 1/2 ค่าสูงสุดของทอกเกิลเกต (Maximum toggle gate)

2.5.2.1 CLB (Configuration Logic Block)

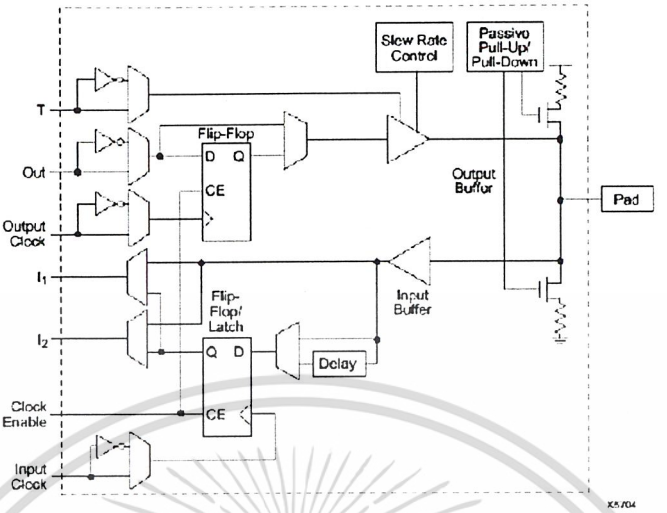
ภายใน LCA (Logic Cell Array) คือเมทริกซ์ของซีแอลบีแต่ละตัวประกอบด้วยหน่วยของคอมเบนิชั่นลอจิกที่สามารถโปรแกรมได้ (Programmable combination logic) และส่วนของเรจิสเตอร์เก็บข้อมูล (Storage register) ส่วนของวงจรคอมเบนิชั่นลอจิกสามารถใช้สร้างวงจรทางด้านฟังก์ชั่นบูลีนของอินพุต ส่วนเรจิสเตอร์รับค่าจากส่วนคอมบิเนชั่นหรือโดยตรงจากเอาต์พุตของซีแอลบี สามารถขับวงจรคอมบิเนชั่นลอจิกโดยตรงผ่านเส้นทางเดินย้อนกลับ (Feedback path)



รูปที่ 2.7 แสดงผังวงจรภายในของซีแอลบีของเอฟพีจีเอตระกูล XC4000

2.5.2.2 IOB (Input Output Block)

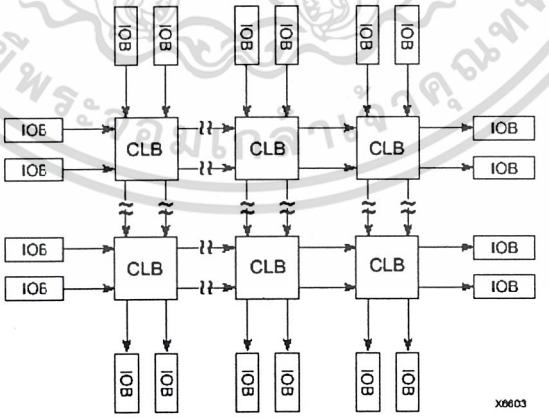
เป็นส่วนติดต่อกับวงจรภายนอกของแอลซีเอสร้างมาจากส่วนของอุปกรณ์อินพุต/เอาต์พุตที่สามารถโปรแกรมได้ (Programmable Input/Output device) แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้เป็นอินพุต/เอาต์พุตแบบ 3 สถานะหรือไอโอแบบสองทิศทางก็ได้ โดยอินพุตสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอลและซีมอสเทรดโฮลของไอโอบี แต่ละตัวมีฟลิปฟลอปสามารถใช้เป็นบัฟเฟอร์สำหรับอินพุตและเอาต์พุต



รูปที่ 2.8 แสดงผังวงจรของไอโอบีภายในเอฟพีจีเอตระกูล XC4000

2.5.2.3 Interconnect

ความยืดหยุ่นของการใช้แอลซีเอมาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรม ทรัพยากรต่างๆ ที่อยู่ภายในเข้าด้วยกันการที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปเหมือนกับเกตอาเรย์ทั่ว ๆ ไป การเชื่อมต่อภายในแอลซีเอประกอบด้วยเน็ตเวริ์ค 2 ทิศทางคือทาง แถวและคอลัมน์ซึ่งวางอยู่ระหว่าง CLB programmable switch จะทำการเชื่อมต่ออินพุตและเอาต์พุตของไอโอบีและซีแอลบีที่จุดต่อร่วมระหว่างแถวกับคอลัมน์สามารถสลับสัญญาณจาก เส้นทางไปยังส่วนต่าง ๆ



รูปที่ 2.9 แสดง Interconnect ระหว่างไอโอบีกับซีแอลบีของเอฟพีจีเอตระกูล XC4000

2.5.3 คุณสมบัติโดยทั่วไปของเอฟพีจีเอตระกูล XC4000

2.5.3.1 เป็นอุปกรณ์รุ่นที่สามของเอฟพีจีเอ

- มีฟลิปฟลอปเป็นจำนวนมาก
- ในการผลิตฟังก์ชั่นของการทำงานมีความยืดหยุ่นสูง
- มีจำนวนเกตภายในจำนวน 2,000 – 10,000 เกต
- เพิ่มความสามารถพิเศษของเรจิสเตอร์และอินพุต/เอาต์พุต
- มีค่าแฟนเอาต์ (fan-out) สูง
- มีบัสภายใน 3 สถานะ
- ทำงานกับสัญญาณทีทีแอลและซีมอส
- มีออสซิเลเตอร์ แอมปลิฟลายเออร์ภายใน
- มีแรมภายในความเร็วสูง (< 25 ns)
- ใช้กับงานที่ต้องการความเร็วสูง (ใช้งานได้ที่ความถี่ 70/100/125 MHz)
- มี Wide edge decoder
- เส้นทางการเชื่อมต่อ (Interconnect line) เป็นแบบลำดับชั้น
- มีการกระจายกำลังงานของสัญญาณต่ำ

2.5.3.2 มีสถาปัตยกรรมภายในที่ยืดหยุ่น

- มีลอจิกบล็อคและไอโอบล็อคที่สามารถโปรแกรมได้
- มีอินเตอร์คอนเน็คและ Wide decoder ที่โปรแกรมได้

2.5.3.3 ทำกระบวนการซับไมครอนชนิดซีมอสได้

- มีลอจิกและอินเตอร์คอนเน็คที่มีความเร็วสูง
- ใช้กำลังงานต่ำ

2.5.3.4 คุณลักษณะทาง System-Oriented

- รองรับมาตรฐาน IEEE 1149.1 ในการทำ boundary-scan logic
- สามารถโปรแกรมค่า output slew rate ได้
- สามารถโปรแกรมให้อินพุตมีลักษณะพูลอัพ (Pull-up) หรือ พูลดาวน์ (Pull-down) เรจิสเตอร์ได้

- ให้กระแสเอาต์พุตได้ตั้งแต่ 12-24 มิลลิแอมป์ (ขึ้นอยู่แต่ละรุ่น)

2.5.3.5 ทำการโหลดเอาแฟ้มข้อมูลประเภทไบนารี

- ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
- มีโหมดในการโปรแกรมให้เลือก 6 โหมด

2.5.3.6 มีโปรแกรมช่วยพัฒนาได้แก่ XACT Development System (ปัจจุบัน Foundation series) ที่ทำงานบนคอมพิวเตอร์รุ่นต่าง ๆ เช่น '486/Pentiums, NEC PC, Apollo, Sun-4, HP700

- สามารถติดต่อกับโปรแกรมอื่น ๆ ได้ เช่น Viewlogic, Mentor Graphic และ OrCAD เป็นต้น
- มีโปรแกรมการวางและเชื่อมโยงอุปกรณ์ภายในแบบอัตโนมัติ (automatic place and routing)
  ที่ครบสมบูรณ์
- มี Interactive Design Editor ที่ใช้สำหรับการทำ optimization
- มี 288 มาโคร 34 ฮาร์ดมาโคร และ แรม/รอมคอมพายเลอร์

2.6 ภาษา VHDL

ในขณะที่ขนาดและความซับซ้อนของระบบดิจิตอลเพิ่มมากขึ้น คอมพิวเตอร์เพื่อช่วยในการออกแบบ (CAD: Computer Aided Design) ก็ได้ถูกนำเข้ามาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มมากขึ้นเช่นกัน อุปกรณ์และวิธีการออกแบบใหม่ๆได้ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้แก่นักออกแบบ และภาษาสำหรับบรรยายอุปกรณ์ฮาร์ดแวร์ (HDL: Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการในการออกแบบระบบดิจิตอล

ภาษา VHDL เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) ซึ่งใช้อธิบายการทำงานของระบบเชิงเลข สามารถใช้อธิบายฟังก์ชันการทำงานได้หลายๆระดับ ตั้งแต่ระดับผังงานจนถึงระดับวงจรเกต ความซับซ้อนของระบบสามารถจะเขียนได้ตั้งแต่ระดับวงจรเกตประกอบกันจนเป็นระบบที่สมบูรณ์ รูปแบบของภาษา VHDL นั้นจะประกอบไปด้วย 2 ส่วนใหญ่ๆ ได้แก่ ส่วนของ Sequential Language และ Concurrent Language การโปรแกรมด้วยภาษา VHDL สามารถจะ

เขียนได้ทั้ง 2 รูปแบบรวมกัน เพราะในการทำงานของระบบใดๆ ย่อมจะมีการทำงานในแบบ Sequential และ Concurrent อยู่รวมกัน นอกจากนี้ตัวภาษา VHDL ยังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยๆ เข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้ ตัวภาษา VHDL นอกจากจะกำหนดรูปไวยากรณ์ (Syntax) ของตัวภาษาแล้วยังมีการตรวจสอบความหมายของตัวภาษาว่าจะทำการจำลองการทำงานได้หรือไม่ เพราะว่าโปรมแกรมที่เขียนโดย VHDL ต้องผ่านการจำลองการทำงานเพื่อตรวจสอบดูการทำงาน ฉะนั้นในการ Compile จะมีการตรวจสอบทั้ง Syntax และ Simulation Semantics อย่างไรก็ตามตัวภาษานั้นถึงจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษาแต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำมาใช้งานได้โดยไม่จำเป็นจะต้องศึกษาให้ละเอียด เนื่องจากตัวภาษา VHDL ออกแบบมาให้ใช้ออกแบบได้ตั้งแต่วงจรที่มีขนาดเล็กจนถึงวงจรขนาดใหญ่ที่ซับซ้อน

2.6.1 แนะนำ VHDL (Introduction to VHDL)

ในปี 1981 สถาบันเพื่อการป้องกัน (The Institute For Defense Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะหนึ่งเพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อของแผนกหนึ่งในสถาบันที่ทำงานเกี่ยวกับวงจรรวมที่มีความเร็วสูง (Vary High Speed Integrated Circuit) ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษามาตรฐานและเป็นที่ยอมรับกันอย่างกว้างขวางในวการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถกำหนด รูปแบบพฤติกรรมการทำงานของวงจรดิจิตอลทั่วไปและในระบบที่แตกต่างกัน ข้อดีที่สำคัญของ VHDL ก็คือ ภาษานี้สามารถใช้ได้ตลอดในทุกๆระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง (System Level) จนถึงในระดับที่ต่ำกว่า (Lower Hardware Level) สามารถใช้ภาษาเดียวกันได้ตลอด ทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

2.6.2 ความสามารถของภาษา VHDL (Capability)

หัวข้อดังต่อไปนี้คือความสามารถหลักๆ ของภาษา VHDL

- ภาษา VHDL สารถใช้เป็นสื่อกลางในการสื่อสารระหว่างผู้ผลิต Chip กับผู้ออกแบบ (CAD Tools)

- ใช้เป็นสื่อกลางในการสื่อสารระหว่าง CAE กับ CAD Tools เช่น Code VHDL สามารถใช้โปรแกรม Compile ได้หลายโปรแกรม
- ภาษา VHDL สนับสนุนการออกแบบทั้งแบบ Top- Down Design และ Bottom-Up Design
- ภาษา VHDL เป็น Generic คือไม่อ้างอิงเทคโนโลยีอันใดอันหนึ่ง (แต่สามารถอิงเทคโนโลยีใดก็ได้ และในขณะเดียวกันก็สามารถสนับสนุนหลายๆ เทคโนโลยี)
- สนับสนุนการออกแบบทั้งระบบ Synchronous และ Asynchronous
- สนับสนุนการออกแบบระบบเชิงเลขในหลายๆ เทคนิค เช่น Finite State Machine Algorithmic หรือ Boolean Equation
- สามารถอ่านและทำความเข้าใจได้โดยมนุษย์ (Human Readable)
- เป็นภาษามาตรฐานรับรองโดย IEEE และ ANSI ทำให้ Model ที่ออกแบบโดย VHDL สามารถเคลื่อนย้าย (Portable) ไปยังระบบใดๆ ก็สามารถนำกลับมาใช้ใหม่ได้ (Reuse)
- สนับสนุนรูปแบบการเขียนถึง 3 รูปแบบ ได้แก่ Behavioral Style, Strctural Style, Dataflow Style หรือสามารถเขียนรวมกันทั้ง 3 รูปแบบ
- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของ Component, Function, Procedure และ Pakage
- สามารถอธิบาย Parameter ที่เกี่ยวกับเวลาได้ เช่น Propagation Delay, Min-Max Delay, Setup, Holding Time, Spike Detection
- ภาษา VHDL เป็นมาตรฐานใช้โดยบริษัทและผู้ออกแบบหลายๆ แห่งฉะนั้นจึงเป็นการง่ายที่จะทำความเข้าใจ ถึงแม้ว่าจะนำมาจากแหล่งต่างๆ

2.6.3 หลักการสร้างโมเดลโดยใช้ภาษา VHDL (General VHDL Modeling Principles)

ภาษา VHDL เป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ (Human Readable) ซึ่งช่วยในการสร้างและออกแบบวงจรของระบบเชิงเลข (Digital Hardware System, Circuit Board) และอุปกรณ์ต่างๆ อาจใช้อธิบายระบบทั้งระบบ หรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปแผนผังอุปกรณ์ (Component Block) จากนั้นก็ทำการจำลองการทำงานโดยที่ทำการออกแบบนั้นยังไม่ได้สร้างขึ้นจริงหรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการสังเคราะห์เพื่อให้ได้วงจรเกตต่อไป

ประโยชน์จริงๆ ของการใช้ VHDL เป็น Tool แทนการสร้างระบบต้นแบบขึ้นมาจริงแบบสมัยก่อน ก็คือ เราสามารถอธิบาย แนวคิดของผลิตภัณฑ์ รายละเอียดของผลิตภัณฑ์ เป็นลักษณะใน

รูปของภาษา VHDL จากนั้นก็นำไป Compile เพื่อดูผังการทำงาน จากนั้นก็ทำการแก้ไขจนกว่าจะได้รายละเอียดของผลิตภัณฑ์ตามต้องการ เมื่อผลิตภัณฑ์ได้ผลตามต้องการแล้วจึงนำไปสู่การสังเคราะห์เพื่อให้ได้ผังวงจรเกต แล้วนำไปสร้างระบบต้นแบบจริงต่อไป ซึ่งเป็นการลดเวลาและค่าใช้จ่ายในการสร้างระบบต้นแบบได้มาก

เนื่องจากว่าภาษา VHDL เป็นภาษาที่มีประสิทธิภาพสูง เราจึงใช้ VHDL อธิบายฮาร์ดแวร์ เพราะว่ามีข้อดี 2 ประการคือ

- เข้าใจง่าย (Easy to Understand)
- สามารถแก้ไขได้ง่าย (Modifiable)

การเข้าใจได้โดยง่ายมีประโยชน์ที่ใครก็ได้ที่จำเป็นต้องอ่านโค้ด (Code) ที่ได้ออกแบบมาแล้วโดยไม่จำเป็นต้องให้ผู้ออกแบบอธิบายให้ฟัง ตัวภาษา VHDL อธิบายการทำงานในตัวอยู่แล้ว ส่วนอีกประการหนึ่งก็คือ ความต้องการในการเปลี่ยนแปลง Hardware ที่ได้ออกแบบแล้วก็คือว่าหลังการทดสอบแล้วพบข้อผิดพลาดซึ่งต้องแก้ไข หรือว่าในระหว่างการพัฒนาต้องการเปลี่ยนแปลงการตอบสนองของระบบ

กรณีอื่นๆ ที่ VHDL มีประโยชน์ก็คือ ตัวภาษานั้นสนับสนุนหลักการต่างๆ ให้เขียน แก้ไข และปรับปรุงระบบเชิงเลขที่มีความซับซ้อนเป็นไปได้อย่างรวดเร็ว และมีประสิทธิภาพ

หลักการออกแบบในภาษา VHDL มีดังนี้

1. Top Down Design

ในการพัฒนาระบบเชิงเลขขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักจะมองระบบที่จะออกแบบให้อยู่ในรูปของแผนผังงานเสียก่อน ก่อนที่จะแบ่งย่อยระบบให้ลึกถึงรายละเอียดต่อไป

- อธิบายการทำงานของแต่ละส่วน
- วิเคราะห์การทำงาน
- จัดการแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ

2. Modularity

Modularity คือหลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ออกเป็นส่วนย่อยเล็กๆ ลงไปซึ่งปกติการทำงานของฮาร์ดแวร์ส่วนใหญ่ต้องประกอบด้วยฮาร์ดแวร์ส่วนย่อยๆ ลงไป ซึ่งวงจรระดับเกตทั้งหมดจะอยู่ในรูปๆ เดียว (Flatten Design) หลังจากนั้น จะตัดออกเป็นส่วนย่อยๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ VHDL หน้าที่การทำงานของแต่ละส่วนสามารถอธิบายได้โดย Module ของรหัส (คล้าย Procedure หรือ Function) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน ซึ่งการ

แยกระบบที่จะออกแบบใหญ่ๆ ออกเป็นส่วนย่อยๆ นี้ทำให้ง่ายต่อการจัดการและง่ายต่อการทำความเข้าใจ

3. Abstraction

คำนิยามของระบบที่จะออกแบบ จะอธิบายการทำงานของระบบมากกว่าที่อธิบายถึงว่าจะพัฒนาตัวระบบนั้นอย่างไร หลักการนี้จะมีความสัมพันธ์อย่างใกล้ชิดกับหลักการ Modularity

อีกวิธีการหนึ่งซึ่งแสดงถึงการอธิบายการทำงานของระบบโดยใช้ VHDL ในหลาย ๆ ระดับของการนิยาม ROM (Read Only Memory) อธิบายโดยใช้ภาษาระดับสูงถึง Address ต่าง ๆ ซึ่งเก็บ DATA ไว้ใน Address นั้น ๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Line, Data Line หรือ Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของ DATA โดยไม่ต้องคิดถึงสัญญาณควบคุมต่าง ๆ มากมายภายใน เพราะว่าส่วนนั้นจะถูกจัดการเองในระดับต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการจัดการสัญญาณภายในทุกเส้นภายในการที่จะอ่านข้อมูลหรือโปรแกรมข้อมูลใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM เราควรขึ้นมาแก้ไขในระดับที่สูงขึ้นมา จะทำให้ง่ายกว่าในการที่จะควบคุมสัญญาณภายใน ซึ่งเราจะเห็นว่าในแต่ละระดับมีความเหมาะสม แตกต่างกันไป และตรงจุดนี้เองทำให้ระบบที่เราออกแบบง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstraction

4. Information Hiding

เมื่อเราทำการเขียนรหัสต้นแบบของภาษา VHDL ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งเราอาจต้องการที่ซ่อนรายละเอียดการพัฒนา Module นั้น ๆ โดยไม่ต้องการให้ส่วน Module อื่น ๆ รู้การทำงานภายใน Information Hiding มีประโยชน์ก็คือ ทำให้ระบบที่ออกแบบนั้นสามารถจัดการได้ง่ายและสามารถจัดการได้ง่ายและสามารถอ่านและทำความเข้าใจได้ง่ายกว่า หลักการนี้จะใช้สนับสนุนการ Abstraction คือ จะสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่าระบบนั้นจะถูกสร้างขึ้นมาอย่างไรบ้างเป็นต้น การซ่อนรายละเอียดภายใน Module ทำให้ความสนใจของผู้ออกแบบสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ้อนไว้และเข้าถึงไม่ได้ เช่น คำอธิบายของ Nand Gate นั้นจะถูกปิดบังเอาไว้จากคนที่เขียนอธิบาย Flip-Flop คนที่เขียนอธิบายการทำงานของ Flip-Flop ไม่ต้องสนใจเลยว่า Nand Gate จะทำงานอย่างไร จะต่อกันภายในอย่างไร โดย Nand Gate สามารถเขียนขึ้นมาแล้ว Compile เก็บไว้ใน Library ผู้ที่ออกแบบ Flip-flop ระดับสูงขึ้นมาเพียงแต่ต้องรู้ว่าจะเชื่อมต่อ input/output ของ Nand Gate มาใช้งานได้อย่างไร และประโยชน์อีกอย่างของ Information Hiding ก็คือป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL Module ไปยังที่อื่น ๆ เช่นส่งไปให้อีกบริษัทใช้พัฒนาร่วมกับ VHDL อื่น ๆ โดยเป็นการ

แจกจ่าย อาจส่งไปแค่ VHDL ที่ Compile แล้ว ไม่ต้องส่งตัวรหัสต้นแบบไป ทำให้เราป้องกันทรัพย์สินทางปัญญาได้ในระดับหนึ่ง

5. Uniformity

Uniformity เป็นหลักการอีกอย่างที่ช่วยในการอธิบายระบบด้วยภาษา VHDL นั้นอ่านได้ง่ายขึ้น Uniformity หมายถึงการสร้าง Module ของรหัสในลักษณะคล้ายกัน โดยใช้ตัวภาษา VHDL Building Block ทำให้เกิดการเขียนรหัสต้นแบบที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้ Comment อธิบายเป็นต้น ทำให้การพัฒนา Module อ่านและทำการเข้าใจง่าย

2.6.4 ระดับของการอธิบายระบบ (Level of Abstraction)

การออกแบบโดยใช้ภาษา VHDL Description ก็มีหลายระดับ แต่ละระดับก็เหมาะสมและมีข้อเสียต่างกันไปซึ่งรูปแบบของภาษา VHDL มี 3 ระดับดังนี้

1. Behavioral Description
2. Dataflow Description
3. Structural Description

1. Behavioral Description

เป็นระดับที่เป็นนามธรรมที่สุด โดยภาษา VHDL นั้น มีรูปแบบนี้เป็นการอธิบายการทำงานของระดับคล้าย ๆ กับโปรแกรมคอมพิวเตอร์ คือ มี Procedural Form และ Function Form ซึ่งไม่ต้องกล่าวถึงรายละเอียดและการสร้างตัวระบบเลย การใช้ภาษา VHDL รูปแบบนี้เหมาะสำหรับอธิบายระบบที่มีความซับซ้อนมาก ๆ โดยนักออกแบบไม่ต้องทราบเลยว่าระบบจะประกอบด้วยอุปกรณ์บ้าง ทำให้ VHDL รูปแบบนี้เหมาะกับผู้ใช้งานทั่ว ๆ ไป และเป็นการอธิบายตัวระบบที่ดีไปในตัว (Good Documentation)

2. Dataflow Description

เป็นระดับที่ต่ำกว่า Behavioral คือแสดงการไหลของข้อมูลภายในหน่วยย่อย ๆ ของระบบ โดยตัวภาษาจะอธิบายการสื่อสารข้อมูลระหว่างย่อย ๆ นั้น ผู้ออกแบบจะต้องรู้การทำงานของหน่วยย่อย ๆ แต่ละหน่วยและรูปแบบของข้อมูลที่จะส่งไปมา การอธิบายในระดับนี้ไม่เหมาะกับผู้ใช้ทั่วไป แต่จะเหมาะกับระดับช่างเทคนิค หรือวิศวกรเท่านั้น การเขียนอธิบายแบบนี้จะเสียเวลามากกว่าแบบ Behavioral ก็จริงแต่ข้อดีก็คือ สามารถที่จะสังเคราะห์ออกมาได้ดีกว่าเพราะแสดงรายละเอียดของระบบออกมาในระดับผังงานซึ่งมากกว่าแบบ Behavioral ซึ่งไม่แสดงอะไรออกมาเลย แต่อย่างไรก็ตามเราจะใช้รูปแบบ Dataflow ในการอธิบายการทำงานของระบบโดยการไหลข้อมูลระหว่าง Register และ Bus

3. Structural Description

เป็นการอธิบายฮาร์ดแวร์ในระดับต่ำที่สุด โดยอธิบายถึงการเชื่อมต่ออุปกรณ์แต่ละตัวเพื่อที่จะให้ได้เป็นระบบขึ้นมา ระดับนี้เป็นระดับที่สังเคราะห์ได้ง่ายที่สุดเพราะแสดงรายละเอียดของระบบได้มากที่สุด อย่างไรก็ตามภาษา VHDL ในระดับนี้อธิบายฮาร์ดแวร์ได้ชัดเจนที่สุด บางครั้งเรียกว่า Hardware Description at Gate Level ระดับนี้เข้าใจได้ยากที่สุด แต่เหมาะสำหรับการจำลองการทำงานที่ต้องการผังเวลาที่ละเอียดที่สุด

2.6.5 สรุป

- ภาษา VHDL คือภาที่ออกแบบมาเฉพาะเพื่อที่ใช้อธิบายการทำงานของฮาร์ดแวร์ให้อยู่ในรูปแบบที่สามารถอ่านทำความเข้าใจได้ (Human Readable) สามารถอธิบายได้ถึงการจัดระบบและการทำงานของวงจรระดับเกต, วงจรระดับ Board และอุปกรณ์ต่างๆ
- เหตุผลที่ทำให้ภาษา VHDL ใช้ในการออกแบบและจำลองการทำงานของผลิตภัณฑ์ตัวหนึ่งซึ่งยังไม่ได้สร้างจริง ๆ เพื่อดูการทำงานก่อนลงมือสร้าง หรืออาจใช้เป็นตัวแทนแนวคิดในผลิตภัณฑ์นั้น ๆ มีดังนี้

  1. ภาษา VHDL อนุญาตให้เราออกแบบจำลองการทำงาน และทดสอบระบบ โดยใช้รูปแบบของภาษาระดับสูงจนถึงระดับวงจรเกต
  2. ภาษา VHDL ถ้าเราเขียนตามรูปแบบของ VHDL นั้นไปทำการสร้างวงจรได้โดยใช้ VHDL Synthesis Guide จะทำให้เราสามารถใช้รหัสภาษา VHDL นั้นไปทำการสร้างวงจรได้โดยใช้ VHDL Synthesis Tools
  3. เพราะว่าภาษา VHDL เป็นภาษาที่กำหนดเป็นมาตรฐาน IEEE 1076-1987 ผู้ออกแบบ สามารถใช้ภาษานี้ในการพัฒนาได้เหมือนกัน ลดปัญหาความเข้ากันไม่ได้ลงไป

- VHDL มีคุณสมบัติที่ดีทำให้เราสามารถเขียนและแก้ไขระบบที่มีขนาดใหญ่และซับซ้อนได้อย่างสะดวกรวดเร็วและมีประสิทธิภาพ ดังนี้คือ

  1. Top Down Design วิธีการนี้ ทำให้เราสามารถอธิบายการทำงานของระบบได้ในลักษณะของ Block ใหญ่ ๆ จากนั้นทำการวิเคราะห์จำลองการทำงานและแก้ไขให้ได้คุณสมบัติตามที่เราต้องการ ณ ระดับ Block ก่อนที่จะลงลึกในระดับต่ำต่อไป
  2. Modularity วิธีการที่แยกส่วน (หรือการประกอบส่วนย่อย ๆ ขึ้นมา) ระบบที่เราออกแบบเป็นส่วนย่อย ๆ เล็ก ๆ ออกมา

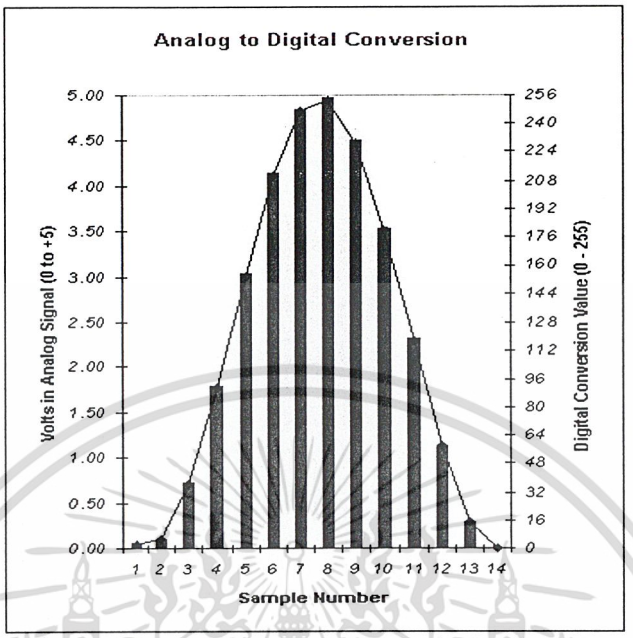
3. Abstraction เป็นรายละเอียดใน Module ซึ่งอธิบายการทำงานของ Modole มากกว่าที่จะอธิบายถึงการพัฒนาและการสร้าง Module นั้น

2.7 การแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล และการแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก

การติดต่อระหว่างมนุษย์ส่วนมากจะใช้สัญญาณต่อเนื่อง (Analog) เป็นสัญญาณติดต่อกัน แต่ว่าการทำงานของระบบคอมพิวเตอร์ จะใช้สัญญาณเป็นช่วง (Digital) เป็นสัญญาณในการทำงาน ดังนั้นถ้าเราต้องการให้คอมพิวเตอร์ช่วยเราทำงาน เราต้องเปลี่ยนสัญญาณแอนาลอกที่เราใช้งานอยู่เป็นสัญญาณดิจิตอลเพื่อให้คอมพิวเตอร์หรือเครื่องประมวลผลสัญญาณทางดิจิตอลรับรู้ เมื่อเครื่องประมวลผลสัญญาณเสร็จจะส่งสัญญาณออกมาเป็นสัญญาณดิจิตอล ซึ่งเป็นเรื่องยากที่มนุษย์จะทำความเข้าใจกับข้อมูลนั้น ดังนั้นพอสรุปได้ว่าการเปลี่ยนสัญญาณแอนาลอกซึ่งเป็นสัญญาณดิจิตอลและการเปลี่ยนสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก เป็นการประสานโลกของคอมพิวเตอร์ให้เข้ากับโลกของมนุษย์ได้ การเปลี่ยนสัญญาณแอนาลอกซึ่งเป็นสัญญาณดิจิตอล เรียกว่า Analog to Digital Converter (ADC) และการเปลี่ยนสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก เรียกว่า Digital to Analog Converter (DAC)

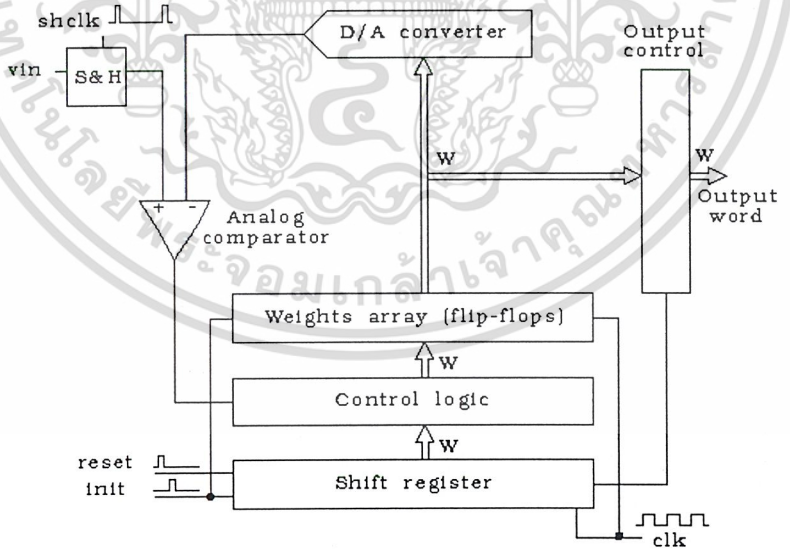
2.7.1 การแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล (Analog to Digital Conversion: ADC)

การแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลสัญญาณจะถูกแปลงเป็นจำนวนทางดิจิตอล โดยการสุ่ม ถ้าสมมติว่ามีเอาท์พุท 8 เส้น โดยเอาท์พุทแต่ละเส้นแสดงสถานะทางลอจิกเป็น 0 หรือ 1 จะมีความแตกต่างทางรหัสไบนารีทั้งหมด 256 รหัส ดังรูปที่ 2.10



รูปที่ 2.10 สัญญาณแอนาลอกจะถูกสุ่มที่ช่วงคลื่นเป็นระยะ

วงจรการแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลที่นิยมกันมากคือแบบ Successive Approximate เพราะใช้เวลาในการแปลงน้อยที่สุด



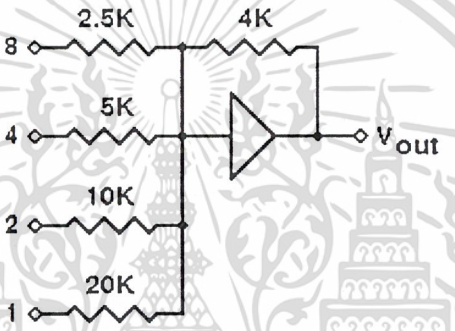
รูปที่ 2.11 บล็อคไดอะแกรมของ Successive Approximate ADC

2.7.2 การแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอก

(Digital to Analog Conversion: DAC)

ทุกวันนี้คอมพิวเตอร์เข้ามามีบทบาทกับมนุษย์มากขึ้น อุปกรณ์ประเภทวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอกก็มีบทบาทเพิ่มขึ้นด้วยจะพบว่าเสียงดนตรีเสียงพูดที่เป็นลักษณะดิจิตอลจะสามารถจำกัดเสียงรบกวนและส่งในกระแสไฟฟ้าได้ง่าย และเมื่อต้องการใช้งานจริงทางเครื่องรับจะถูกเปลี่ยนกลับเป็นสัญญาณแอนาลอกอีกครั้ง

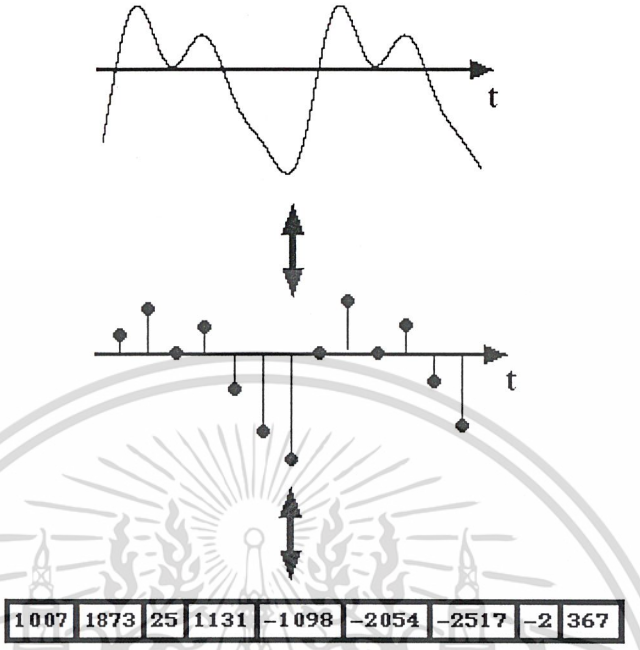
การแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอกง่ายกว่าการแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอลวงจรที่นิยมใช้เป็นแบบ Voltage Summing Amplifier ดังรูปที่ 2.12



รูปที่ 2.12 วงจร Voltage summing amplifier DAC

2.7.3 ทฤษฎีการสุ่มสัญญาณ (Sampling)

การสุ่มสัญญาณเป็นขั้นตอนแรกในการแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล โดยวงจรการสุ่มสัญญาณจะตรวจจับขนาดสัญญาณแอนาลอกที่ถูกส่งตามช่วงเวลาที่ถูกกำหนด โดยทั่วไปมักวัดขนาดในรูปของแรงดันไฟฟ้า ซึ่งแท้จริงแล้วกระบวนการสุ่มสัญญาณ เป็นกระบวนการตรวจวัดค่าแรงดันของสัญญาณที่ช่วงเวลาต่าง ๆ ซึ่งมีคาบการตรวจจับคงที่ อัตราหรือความถี่ของการสุ่มสัญญาณเป็นคาบนี้จะกำหนดให้อยู่ในหน่วยของจำนวนจุดสุ่มต่อหนึ่งหน่วยเวลา ยกตัวอย่าง เช่น ในระบบโทรศัพท์อัตราการสุ่มดังกล่าวมีค่าเป็น 8 KHz อาจกล่าวได้ว่าคาบเวลาของการสุ่มสัญญาณ หรือช่วงเวลาระหว่างการสุ่มแต่ละครั้ง มีค่าเป็น 1 / 8000 หรือ 125 us สำหรับวิธีในการคำนวณหาค่าอัตราการสุ่มของระบบโทรศัพท์โดยเป็นไปตามข้อกำหนดในทฤษฎีของการสุ่ม ซึ่งถูกกำหนดโดยแชนนอน นักคณิตศาสตร์ชาวอเมริกา กล่าวไว้ว่าอัตราการสุ่มจะต้องมีความถี่ไม่น้อยไปกว่า 2 เท่า ของความถี่สูงสุดของสัญญาณแอนาลอกที่จะนำมาสุ่มนั้น จึงจะสามารถสร้างสัญญาณต้นฉบับกลับคืนจากสัญญาณสุ่มได้



รูปที่ 2.13 การสุ่มสัญญาณ

บทที่ 3

การออกแบบ การสร้าง และการทำงาน

3.1 กล่าวนำ

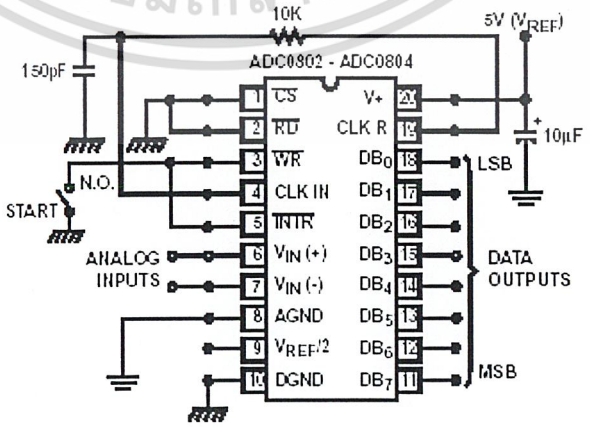
ในส่วนของการออกแบบ และการสร้างนั้นได้แบ่งออกเป็นส่วนการทำงานทั้งหมด 4 ส่วนใหญ่ ๆ คือ วงจรแปลงสัญญาณแอนาลอกเป็นดิจิตอล , การคำนวณหาค่าสัมประสิทธิ์ของการแปลง DCT , การเขียนคำสั่งในส่วนประมวลผล , การโปรแกรมบนอุปกรณ์ FPGA

3.2 วงจรการแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล

ในส่วนวงจรการทำงานในส่วนนี้จะเป็นส่วนในการแปลงสัญญาณให้อยู่ในรูปของสัญญาณดิจิตอล ในการออกแบบวงจรส่วนนี้ได้เลือกใช้ไอซี แปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล เบอร์ ADC 0804 ซึ่งเป็นไอซีแปลงสัญญาณ ขนาด 8 บิต ซึ่งมีคุณสมบัติพิเศษ คือ สามารถสร้างสัญญาณซิเลเตอร์ ได้โดยการต่อ ตัวต้านทาน และตัวเก็บประจุ เข้าไปได้ และยังสามารถกำหนดค่ากึ่งกลางของสัญญาณอ้างอิงในส่วนของ Vref/2 ได้

การออกแบบวงจร ได้กำหนดในส่วนรับสัญญาณอินพุตให้รับสัญญาณอินพุตอยู่ในช่วง 0 V ถึง

5 V โดยวงจรการทำงานแสดงดังรูปที่ 3.1



รูปที่ 3.1 วงจรแปลงสัญญาณแอนาลอกเป็นสัญญาณดิจิตอล

การปรับแต่งวงจร

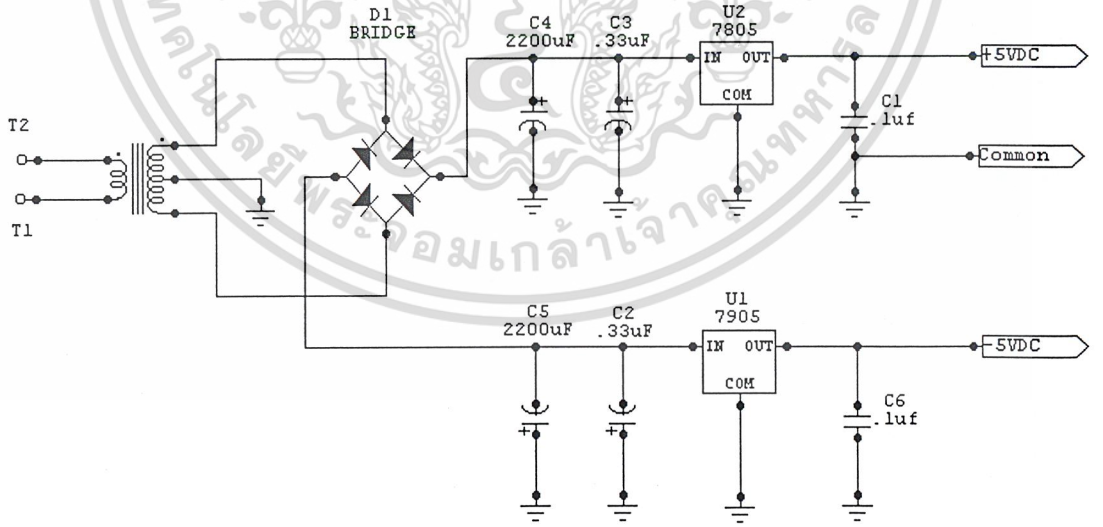
1) ทำการจ่ายไฟให้กับวงจร

2) ทำการปรับแต่งค่าแรงดันอ้างอิงโดยการวัดค่าแรงดันที่ตกคร่อม VR1 และทำการปรับ VR1 ให้มีแรงดันตกคร่อม 2.5 V

3) ทำการปรับระดับแรงดันของสัญญาณอินพุต โดยใช้ออสซิลโลสโคปวัดรูปคลื่นสัญญาณอินพุต ให้อยู่ในช่วง 0 V ถึง 5 V

วงจรภาคจ่ายไฟ

เนื่องจากวงจรทั้งหมดที่ออกแบบมานั้นต้องการใช้ไฟเลี้ยงวงจร $\pm$ 5 V ดังนั้นเราจึงต้องใช้วงจรแหล่งจ่ายแรงดันเป็น $\pm$ 5 V ซึ่งวงจรแหล่งจ่ายไฟแสดงดังรูปที่ 3.2 ทำงานโดยการใช้ไอซีเรกกูเลเตอร์ เบอร์ 7805 และ เบอร์ 7905 เป็นตัวให้แรงดันที่เอาท์พุตของวงจรเป็น $\pm$ 5 V



รูปที่ 3.2 วงจรภาคจ่ายไฟ

3.3 การสร้างวงจรการแปลง DCT บน FPGA

ในการสร้างวงจรการแปลง DCT นั้น มีลำดับขั้นตอนดังนี้

3.3.1 คำนวณหาค่าสัมประสิทธิ์ของ DCT

ในการหาค่าสัมประสิทธิ์ของการแปลงสัญญาณนั้น จะทำการคำนวณค่าจากสมการคณิตศาสตร์ของการแปลงสัญญาณนั้น จากสมการที่ 3.1 คือสมการแปลงสัญญาณ DCT ซึ่งในโครงงานนี้ได้ใช้ DCT 8 จุด (N = 8)

$$y(k) = c(k)\sum_{n=0}^{N-1}\cos\frac{2\pi k(2n+1)}{4N}x(n) \qquad (3.1)$$

โดย y (k) เป็นค่าสัมประสิทธิ์ในการแปลง

x (n) เป็นข้อมูลเริ่มต้น

$c(0) = 1/\sqrt{N}$ และ $c(k) = \sqrt{(2/N)}$, $1 \le k \le N-1$

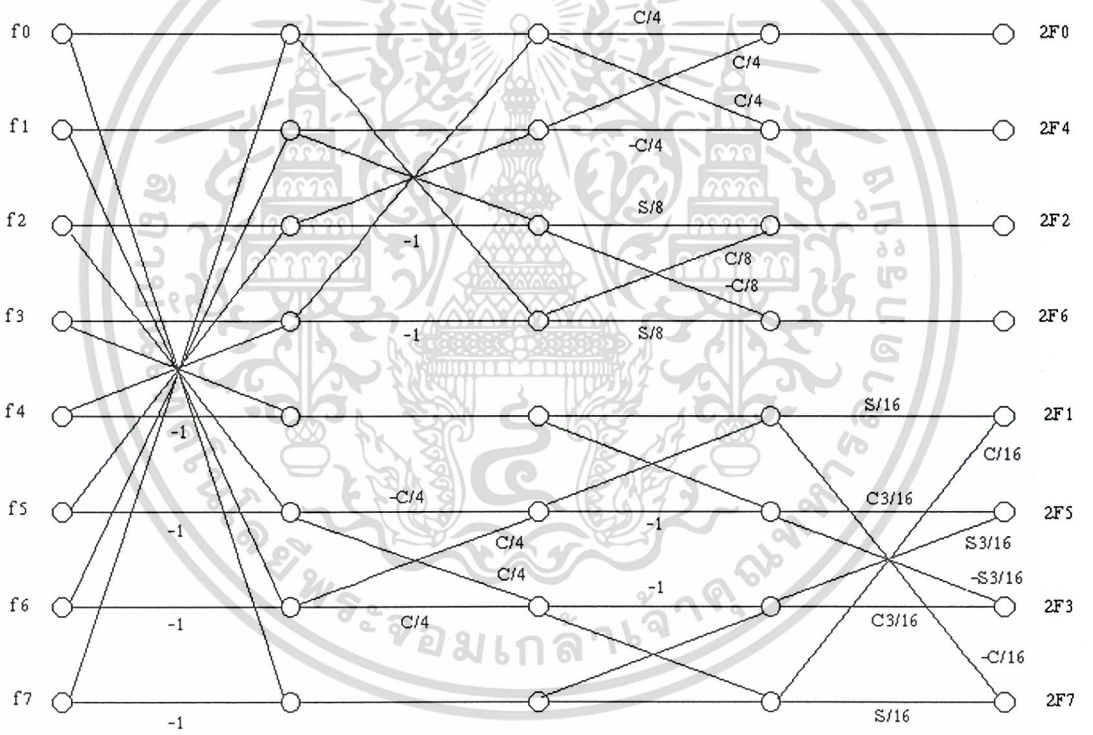
ซึ่งการหาค่าสัมประสิทธิ์ได้ใช้โปรแกรม MATLAB ในการหาค่า และได้แสดงค่าสัมประสิทธิ์ไว้ในตารางที่ 3.1

	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5	n = 6	n = 7
k = 0	0.70710	0.98078	0.92387	0.83146	0.70710	0.55557	0.38268	0.19509
k = 1	0.70710	0.83146	0.38268	-0.19509	-0.70710	-0.98078	-0.92387	-0.55557
k = 2	0.70710	0.55557	-0.38268	-0.98078	-0.70710	0.19509	0.92387	0.83146
k = 3	0.70710	0.19509	-0.92387	-0.55557	0.70710	0.83146	-0.38268	-0.98078
k = 4	0.70710	-0.19509	-0.92387	0.55557	0.70710	-0.83146	-0.38268	0.98078
k = 5	0.70710	-0.55557	-0.38268	0.98078	-0.70710	-0.19509	0.92387	-0.83146
k = 6	0.70710	-0.83146	0.38268	0.19509	-0.70710	0.98078	-0.92387	0.55557
k = 7	0.70710	-0.98078	0.92387	-0.83146	0.70710	-0.55557	0.38268	-0.19509

ตารางที่ 3.1 ค่าสัมประสิทธิ์ของ DCT 8 จุด

ซึ่งในการในการสร้าง DCT ถ้าทำตามการหาสัมประสิทธิ์ข้างต้น การคำนวณจะต้องคูณตัวเลข N x N ครั้ง และบวกตัวเลขอีก N x (N-1) ครั้ง ซึ่งในกระบวนการคูณนั้นใช้เวลาและทรัพยากรใน FPGAs มากกว่าการบวกมากจึงมีการพัฒนา ขั้นตอนวิธี การแปลงดิสครีตโคไซน์ที่ลดจำนวนการคูณให้น้อยลงซึ่งจะทำให้การแปลงสัญญาณมีความเร็วสูงขึ้น

ซึ่งขั้นตอนวิธีการแปลงดิสครีตโคไซน์มีอยู่หลายวิธี ในโครงงานนี้ได้ใช้ขั้นตอนวิธีดังกราฟการไหลของสัญญาณของ DCT 8 จุดได้แสดงในรูปที่ 3.3 โดยมีการบวนการคูณ 20 ครั้ง การคำนวณของ DCT จะใช้เลขจำนวนจริงทั้งด้านสัญญาณเข้าและออก โดยสัญญาณด้านเข้าคือ f0 ถึง f7 และสัญญาณออกที่จะได้คือ F0 ถึง F7



รูปที่ 3.3 กราฟสัญญาณไหลแสดง การแปลงดีสครีตโคไซน์ 8 จุด

(Signal Flow Diagram For 8-point DCT Computation)

โดย C/4 = $\cos(\pi/4)$ = 0.7071 , C3/16 = $\cos(3\pi/16)$ = 0.8314

C/8 = $\cos(\pi/8)$ = 0.9238 , S3/16 = $\sin(3\pi/16)$ = 0.5555

S/8 = $\sin(\pi/8)$ = 0.3826 , S/16 = $\sin(\pi/16)$ = 0.1950

C/16 = $\cos(\pi/16)$ = 0.9807

3.3.2 คำนวณหาค่าสัมประสิทธิ์ของ IDCT

เช่นเดียวกับการหาค่าสัมประสิทธิ์ของการแปลงDCT จะทำการคำนวณค่าจากสมการคณิตศาสตร์ของการแปลงสัญญาณนั้น จากสมการที่ 3.2 คือสมการแปลงสัญญาณ IDCT

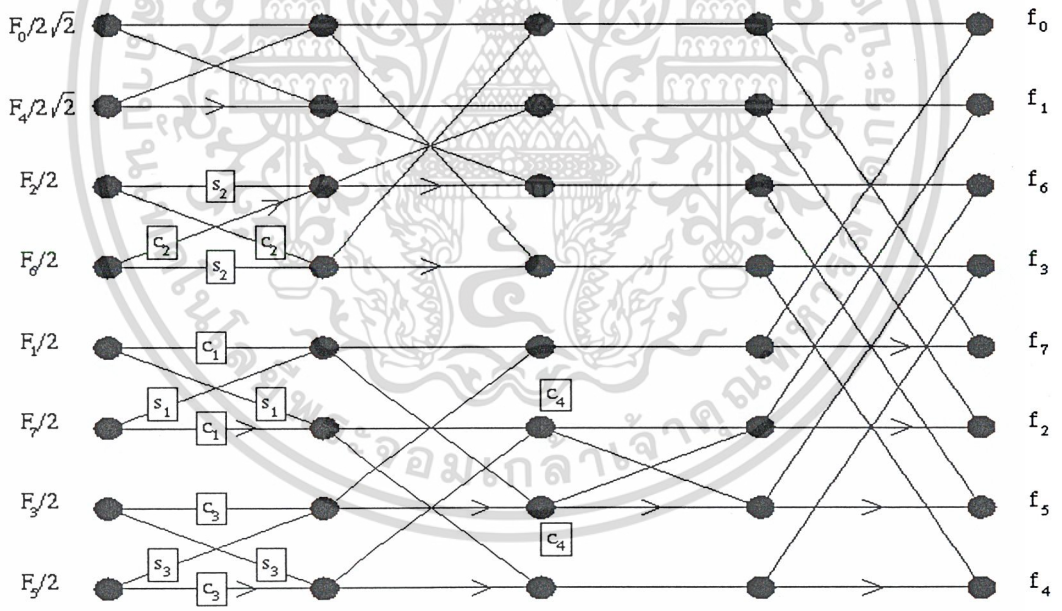
$$x(n) = \frac{1}{\sqrt{2N}} \sum_{k=0}^{N-1} c(k)y(k)\cos\frac{2\pi k(2n+1)}{4N} \qquad (3.2)$$

โดย x (n) เป็นค่าสัมประสิทธิ์ในการแปลง

y (k) เป็นข้อมูลเริ่มต้น

$c(0) = 1/\sqrt{N}$ และ $c(k) = \sqrt{(2/N)}$, $1 \leq k \leq N-1$

ขั้นตอนวิธี การแปลงกลับดิสครีตโคไซน์ที่ลดจำนวนการคูณให้น้อยลงซึ่งจะทำให้การแปลงสัญญาณมีความเร็วสูงขึ้น โดยสัญญาณด้านเข้าคือ F0 ถึง F7 และสัญญาณออกที่จะได้คือ f0 ถึง f7ดังรูปที่3.4



รูปที่ 3.4 กราฟสัญญาณไหลแสดง การแปลงกลับดีสครีตโคไซน์ 8 จุด

(Signal Flow Diagram For 8-point IDCT Computation)

โดย C1 = cos(π/16) = 0.9807 , S1 = sin(π/16) = 0.1950

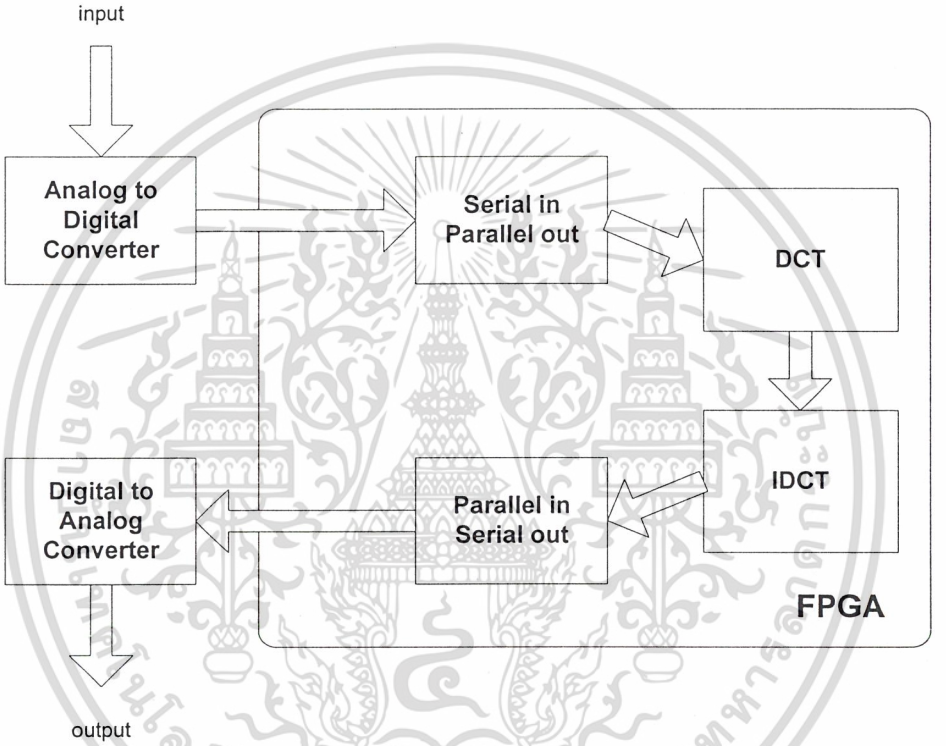
C2 = cos(2π/16) = 0.9238 , S2 = sin(2π/16) = 0.3826

C3 = cos(3π/16) = 0.8314 , S3 = sin(3π/16) = 0.5555

C4 = cos(4π/16) = 0.7071

3.4 การเขียนคำสั่งในการประมวลผล

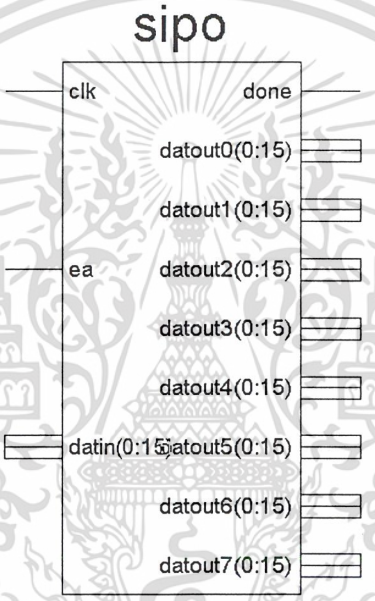
ในการเขียนโปรแกรมในส่วนของการทำงาน เพื่อนำไปโปรแกรมลงในส่วนของอุปกรณ์ FPGA นั้นได้ใช้ภาษา VHDL ในการเขียนโปรแกรม โดยที่จะแบ่งส่วนการเขียนโปรแกรมออกเป็นส่วน ๆ ตามการทำงาน ดังรูปที่ 3.4



รูปที่ 3.5 ไดอะแกรมการทำงานของโปรแกรม

3.4.1 การเขียนโปรแกรมส่วน Serial In Parallel Out

ในส่วนของ Serial In Parallel Out นั้นจะเป็นส่วนที่ทำการเตรียมข้อมูลนำเข้า ซึ่งมีลักษณะสัญญาณดิจิตอลขนาดความยาวข้อมูล 8 บิต มีลักษณะการส่งสัญญาณมาเป็นลักษณะสัญญาณอนุกรม ซึ่งโปรแกรม Serial In Parallel Out นี้จะเป็นส่วนที่จัดข้อมูล ให้มีลักษณะเป็นเมตริกซ์ขนาด 8 x 1 โดยจะมีการทำงานคือทำการนับข้อมูลที่เข้ามาทางด้านอินพุตให้ครบจำนวน 8 (0 - 7) จำนวน แล้วทำการนำข้อมูลออกไปในลักษณะของข้อมูลขนาน ขนาด 8 ช่องสัญญาณ ดังรูปที่ 3.5

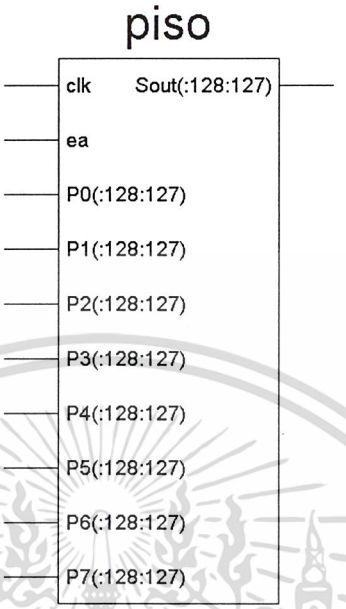


รูปที่ 3.6 ไดอะแกรมการทำงานโปรแกรมส่วน Serial In Parallel Out

3.4.2 การเขียนโปรแกรมส่วน Parallel In Serial Out

การทำงานในส่วนนี้จะเป็นส่วนการทำงานหลังจากส่วน DCT แล้วซึ่งจะเป็นส่วนที่จะจัดข้อมูลขนานที่อยู่ในลักษณะเมตริกซ์ ขนาด 1 x 8 ให้เป็นข้อมูลอนุกรมเพื่อต่อเข้าไปยังส่วนวงจรการแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนาลอกต่อไป

หลักการเขียนโปรแกรมในส่วนนี้ จะใช้หลักการที่นำข้อมูลขนานซึ่งไม่มีสัญญาณนาฬิกาเข้าจังหวะสัญญาณนำมา ซิงโครไนซ์ รวมกับสัญญาณนาฬิกาที่กำหนดขึ้นจากในส่วนของ FPGA เองและแยกข้อมูลออกเป็นห้วง ๆ จึงได้ข้อมูลกลับมาเป็นลักษณะของข้อมูลอนุกรมนั่นเอง ดังรูปที่ 3.6



รูปที่ 3.7 ไดอะแกรมการทำงานโปรแกรมส่วน Parallel In Serial Out

3.4.3 การเขียนโปรแกรมส่วน DCT

โปรแกรมส่วนนี้เป็นส่วนประมวลผลซึ่งจำเป็นต้องทำความเข้าใจ และทราบขั้นตอนวิธีการคำนวณหาค่า DCT จากกราฟสัญญาณไหลแสดง การแปลงดีสครีตโคไซน์ 8 จุด (รูปที่ 3.3) อย่างละเอียด เพื่อที่จะสามารถนำมาเขียนเป็นโปรแกรมการทำงานซึ่งจะทำการแบ่งส่วนการคำนวณออกเป็นลำดับขั้นตอนซึ่งมีทั้งหมด 4 ลำดับขั้นตอน ดังนี้

ลำดับขั้นตอนที่1

```
x00 = f0 + f7;      x01 = f1 + f6;      x02 = f2 + f5;
x03 = f3 + f4;      x04 = f3 – f4;      x05 = f2 – f5;
x06 = f1 – f6;      x07 = f0 – f7;
```

ลำดับขั้นตอนที่2

```
x10 = x00 + x03;    x11 = x01 + x02;    x12 = x01 - x02;
x13 = x00 - x03;    x14 = x04;          x15 = (-A*x05)+(A*x06);
x16 = (A*x05)+(A*x06);x17 = x07;
```

ลำดับขั้นตอนที่3

x20 = (A*x10)+(A*x11); x21 = (A*x10)-(A*x11);

x22 = (C*x12)+(B*x13); x23 = (-B*x12)+(C*x13);

x24 = x14 + x15; x25 = x14 - x15;

x26 = -x16 + x17; x27 = x16 + x17;

ลำดับขั้นตอนที่4

F0 = x20/2; F1 = ((G*x24)+(D*x27))/2;

F2 = x22/2; F3 = ((-F*x25)+(E*x26))/2;

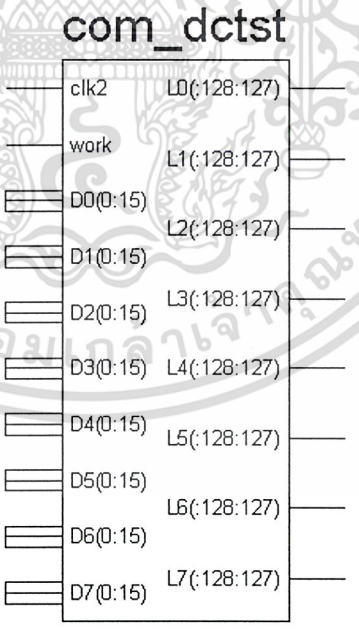
F4 = x21/2; F5 = ((E*x25)+(F*x26))/2;

F6 = x23/2; F7 = ((-D*x24)+(G*x27))/2;

โดย f0 – f7 เป็นสัญญาณขาเข้า F0 – F7 เป็นสัญญาณที่ขาออกที่ถูกแปลงแล้ว

A = 0.7071 B = 0.9238 C = 0.3826 D = 0.9807

E = 0.8314 F = 0.5555 G = 0.1950



รูปที่ 3.8 ไดอะแกรมการทำงานโปรแกรมส่วน DCT

3.4.4 การเขียนโปรแกรมส่วน IDCT

เช่นเดียวกับ DCT การหาค่าการคำนวณของ IDCT ต้องทำความเข้าใจ และทราบขั้นตอนวิธีการคำนวณIDCT จากกราฟการไหลของสัญญาณการแปลงกลับดีสครีตโคไซน์ 8 จุด (รูปที่ 3.4) อย่างละเอียด เพื่อที่จะสามารถนำมาเขียนเป็นโปรแกรมการทำงานซึ่งจะทำการแบ่งส่วนการคำนวณออกเป็นลำดับขั้นตอนซึ่งมีทั้งหมด 4 ลำดับขั้นตอน ดังนี้

ลำดับขั้นตอนที่1

```
x00 = ((F0*c4) + (F4*c4))/2;      x01 = ((F0*c4) - (F4*c4))/2;
x02 = ((F2*s2) - (F6*c2))/2;      x03 = ((F2*c2) + (F6*s2))/2;
x04 = ((F1*c1) + (F7*s1))/2;      x05 = ((F1*s1) - (F7*c1))/2;
x06 = ((F3*c3) + (F5*s3))/2;      x07 = ((F3*s3) - (F5*c3))/2;
```

ลำดับขั้นตอนที่2

```
x10 = (x00 + x03);    x11 = (x01 + x02);    x12 = (x01 – x02);
x13 = (x00 – x03);    x14 = x04 + x06;      x15 = (x05 + x07) * c4;
x16 = (x04 – x06) * c4;   x17 = x05 – x07;
```

ลำดับขั้นตอนที่3

```
x20 = x10;    x21 = x11;    x22 = x12;    x23 = x13;
x24 = x14;    x25 = x15 + x16;        x26 = x15 – x16;
x27 = x17;
```

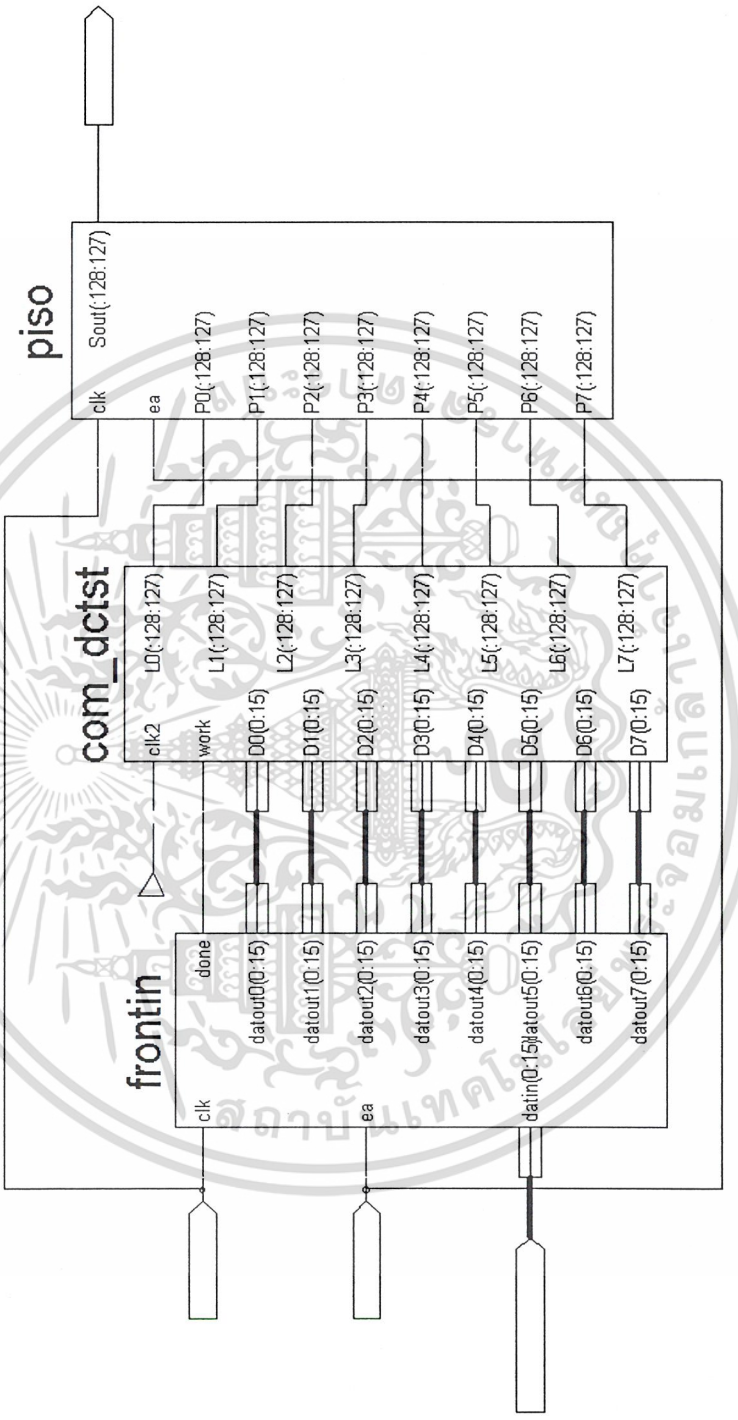
ลำดับขั้นตอนที่4

```
f0 = (x20 + x24)*2;    f1 = (x21 + x25)*2;    f5 = (x22 + x26)*2;
f3 = (x23 + x27)*2;    f7 = (x20 – x24)*2;    f6 = (x21 – x25)*2;
f2 = (x22 – x26)*2;    f4 = (x23 – x27)*2;
```

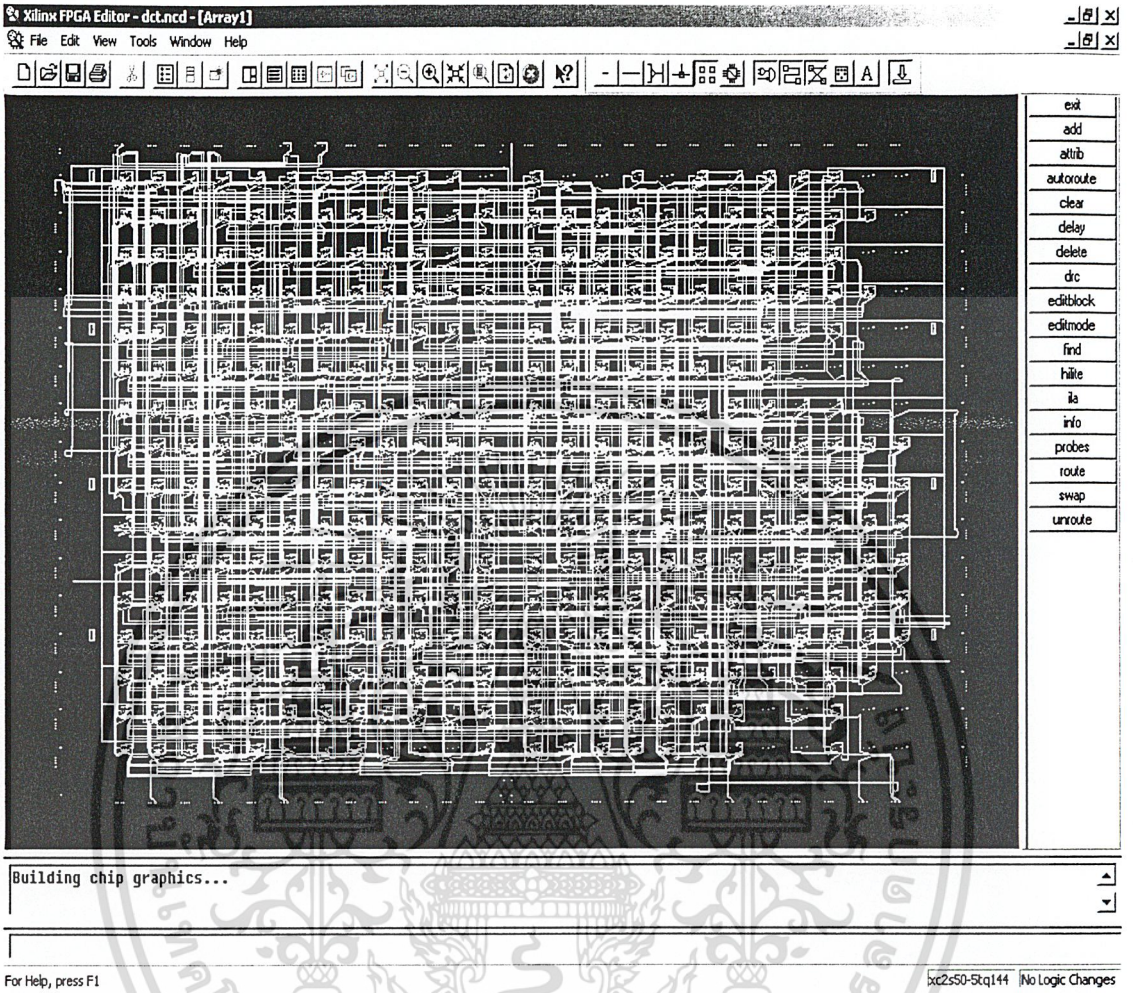
โดย F0 – F7 เป็นสัญญาณขาเข้า f0 – f7 เป็นสัญญาณที่ขาออกที่ถูกแปลงกลับ

C1 = 0.9807 S1 = 0.1950 C2 = 0.9238 S2 = 0.3826

C3 = 0.8314 S3 = 0.5555 C4 = 0.7071



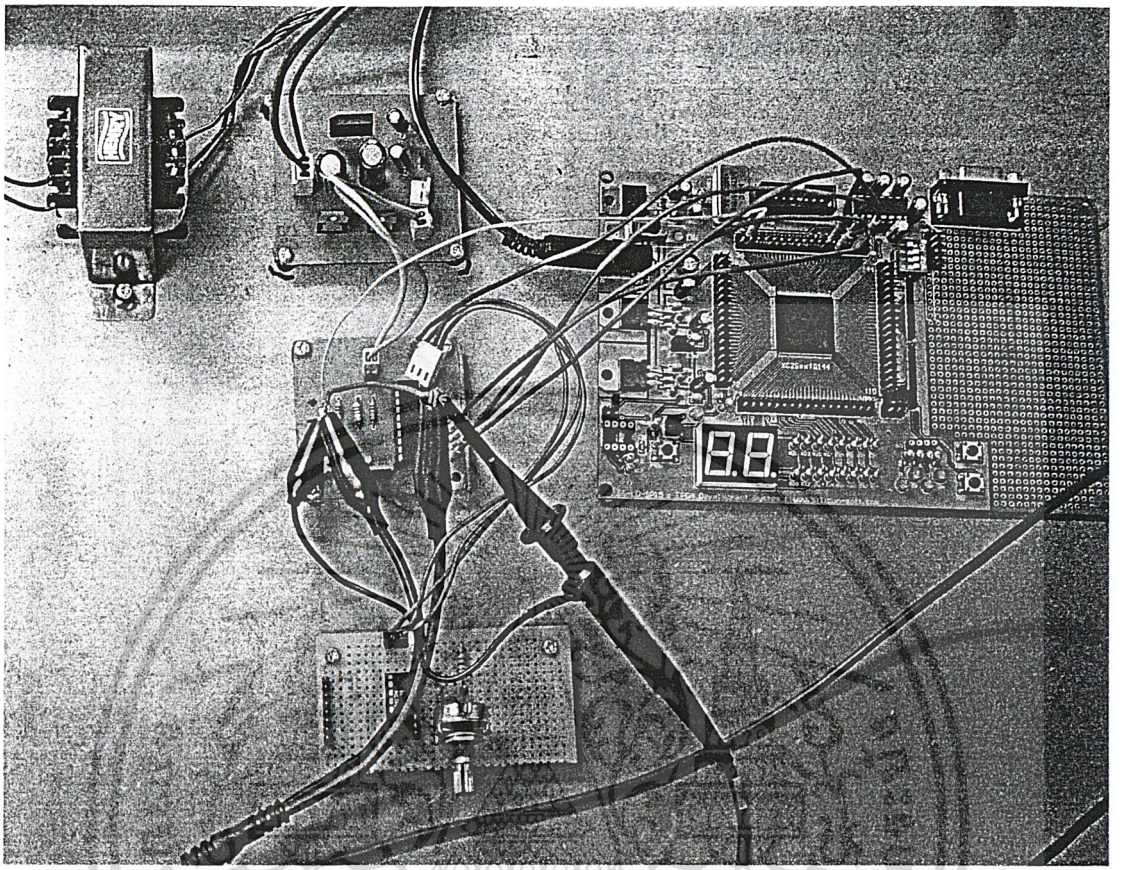
รูปที่ 3.9 ไดอะแกรมการทำงานโปรแกรมส่วนประมวลผล DCT



Device utilization summary:

Number of External GCLKIOBs	1 out of 4	25%
Number of External IOBs	14 out of 92	15%
Number of LOCed External IOBs	14 out of 14	100%
Number of SLICEs	658 out of 768	85%
Number of GCLKs	1 out of 4	25%

รูปที่ 3.10 วงจรรวม (จากรูปที่ 3.9) เมื่อ สร้างบน FPGA



รูปที่ 3.11 วงจรต่าง ๆ ที่ใช้ในโครงงาน

บทที่ 4
การทดลอง และผลการทดลอง

การทดลองการจำลองการทำงานของโปรแกรมนั้นได้ใช้โปรแกรมจำลองการทำงาน Modelsim Simulator จากโปรแกรม Modelsim SE 5.6 และ Xilinx Project Navigator ในการแสดงผลการจำลองการทำงานซึ่งการจำลองการทำงานแบ่งออกเป็น 4 ส่วนคือ ส่วนโปรแกรม Serial in Parallel out , ส่วนโปรแกรม Parallel in Serial Out , ส่วนโปรแกรม DCT ซึ่งเป็นการตรวจสอบความถูกต้องในการทำงานของโปรแกรม , ส่วนวงจรการแปลงสัญญาณ DCT และการแปลงสัญญาณกลับ IDCT

4.1 การจำลองการทำงานส่วนโปรแกรม Serial in Parallel out

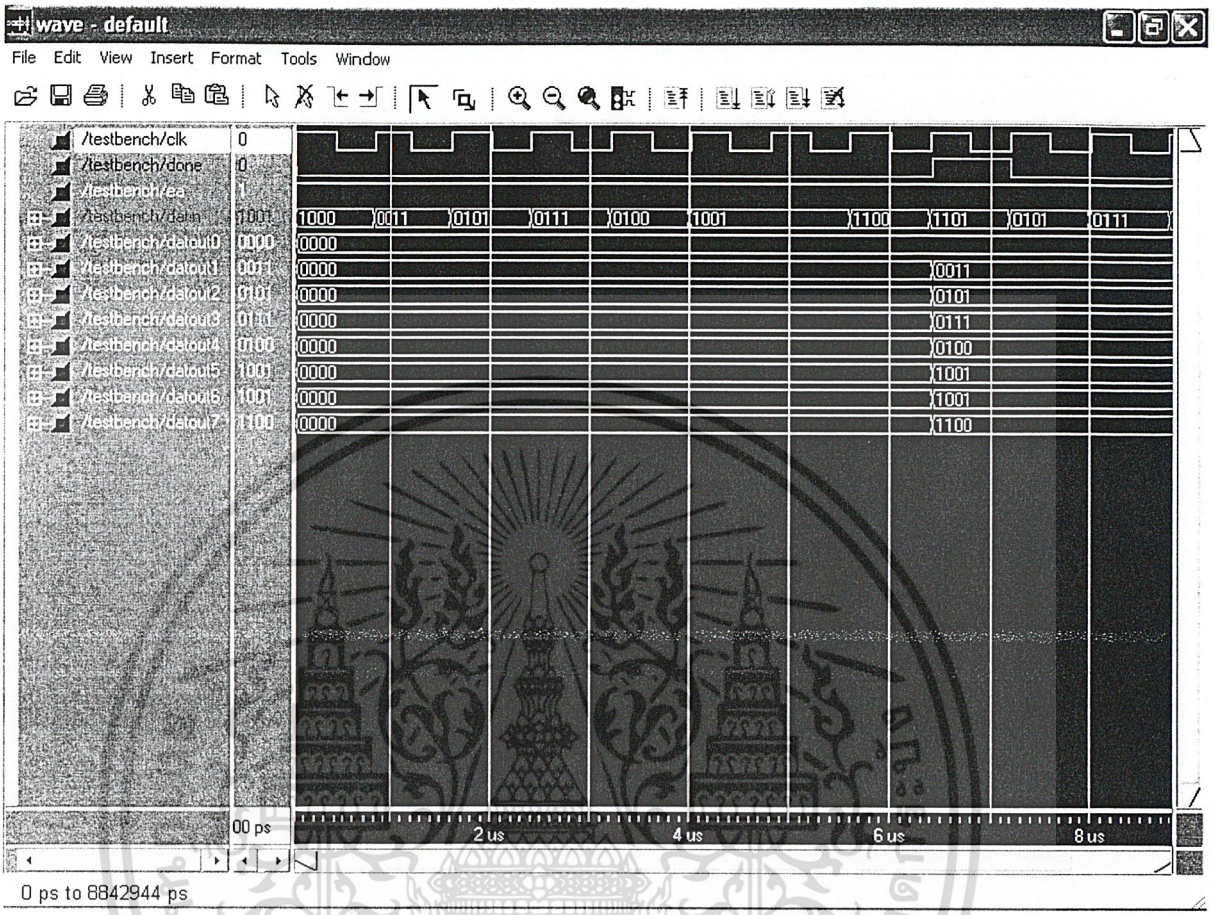
ในส่วนของการจำลองการทำงานในส่วนนี้จะทำการป้อนสัญญาณลอจิก 8 บิต เป็นการนับขึ้น และผลที่ได้จากการทำงานของโปรแกรม คือเมื่อส่วนของโปรแกรม Serial in Parallel out ทำการเรียงข้อมูลอินพุตได้ครบ 8 ชุด จะทำการส่งข้อมูลทั้ง 8 ชุด ออกเป็นลักษณะข้อมูลขนาน และจะส่งสัญญาณลอจิก "1" ที่ขา DONE

ลำดับขั้นตอนการทดลอง

1. สร้างโปรแกรมสำหรับการทดสอบชื่อ testsipo.tbw
2. ป้อนค่าสัญญาณอินพุตใส่ไปในโปรแกรมที่ใช้ทดสอบนั้น
3. ใช้โปแกรม Modelsim Simulator ทำการ Simulate
4. สังเกตและบันทึกผลการทดลอง

ผลการทดลอง

จากผลการจำลองการทำงานที่ได้ โปรแกรมสามารถทำงานได้ถูกต้องตามลำดับที่การทำงานที่ได้เขียนไว้ในโปรแกรม โดยผลการจำลองการทำงานแสดงไว้ในรูปที่ 4.1



รูปที่ 4.1 ผลการจำลองการทำงานของโปรแกรม Serial in Parallel out

4.2 การจำลองการทำงานส่วนโปรแกรม Parallel in Serial out

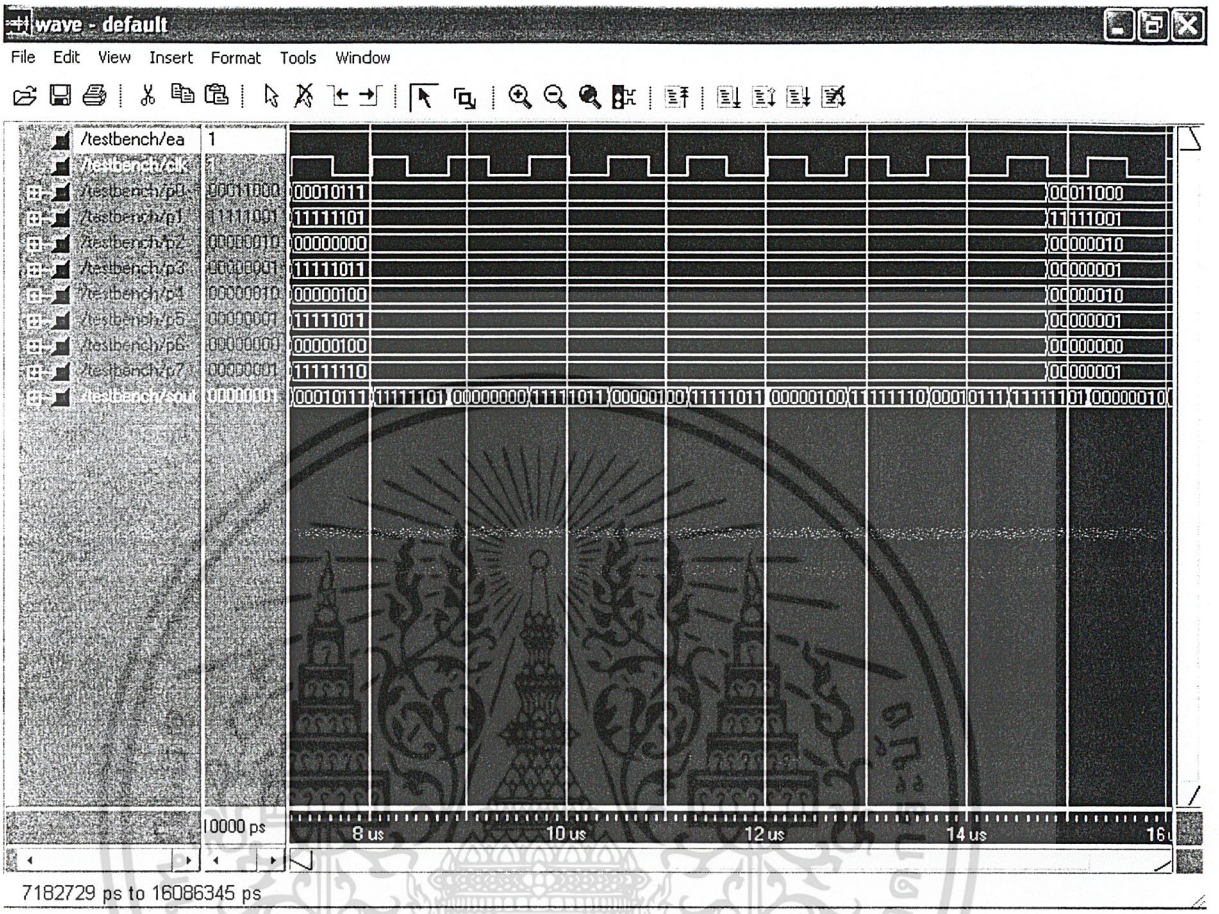
ในการจำลองการทำงานในส่วนนี้เป็นการทำงานของโปรแกรม Parallel in Serial out ซึ่งจะทำการเรียงข้อมูลที่เป็นลักษณะข้อมูลขนานให้เป็นข้อมูลในลักษณะอนุกรม

ลำดับขั้นตอนการทำงาน

1. สร้างโปรแกรมสำหรับการทดสอบชื่อ testpiso.tbw
2. ป้อนค่าสัญญาณอินพุตใส่ไปในโปรแกรมที่ใช้ทดสอบนั้น
3. ใช้โปแกรม Modelsim Simulator ทำการ Simulate
4. สังเกตและบันทึกผลการทดลอง

ผลการทดลอง

จากผลการจำลองการทำงานที่ได้ โปรแกรมสามารถทำงานได้ถูกต้องตามลำดับที่การทำงานที่ได้เขียนไว้ในโปรแกรม โดยผลการจำลองการทำงานแสดงไว้ในรูปที่ 4.2



รูปที่ 4.2 ผลการจำลองการทำงานของโปรแกรม Parallel in Serial out

4.3 การจำลองการทำงานส่วนโปรแกรม DCT

การทดลองการแปลงดิสครีสโคไซน์ เป็นการทดสอบความถูกต้องของโปรแกรมDCT ที่สร้างโดยโปรแกรม Modelsim Simulator กับฟังชันก์ DCT ของโปรแกรม MATLAB เพื่อตรวจสอบความถูกต้องของวงจรที่สร้างขึ้น

วงจร DCT ที่สร้างขึ้น ค่าอินพุต และเอาท์พุต เป็นตัวเลขฐานสอง 8 บิต ซึ่ง

สัญญาณอินพุต 8 บิต คือค่าทศนิยมเช่น

ตัวเลข	1	0	0	0	1	1	1	0
ค่าตำแหน่งฐานสอง	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
ค่าในเลขฐานสิบ	0.5547							

สัญญาณเอาท์พุต 8 บิต เป็นแบบ 2's Complement แบ่งออกเป็น บิตเครื่องหมาย 1 บิต , บิตจำนวนเต็ม 3 บิต , บิตทศนิยม 4 บิต

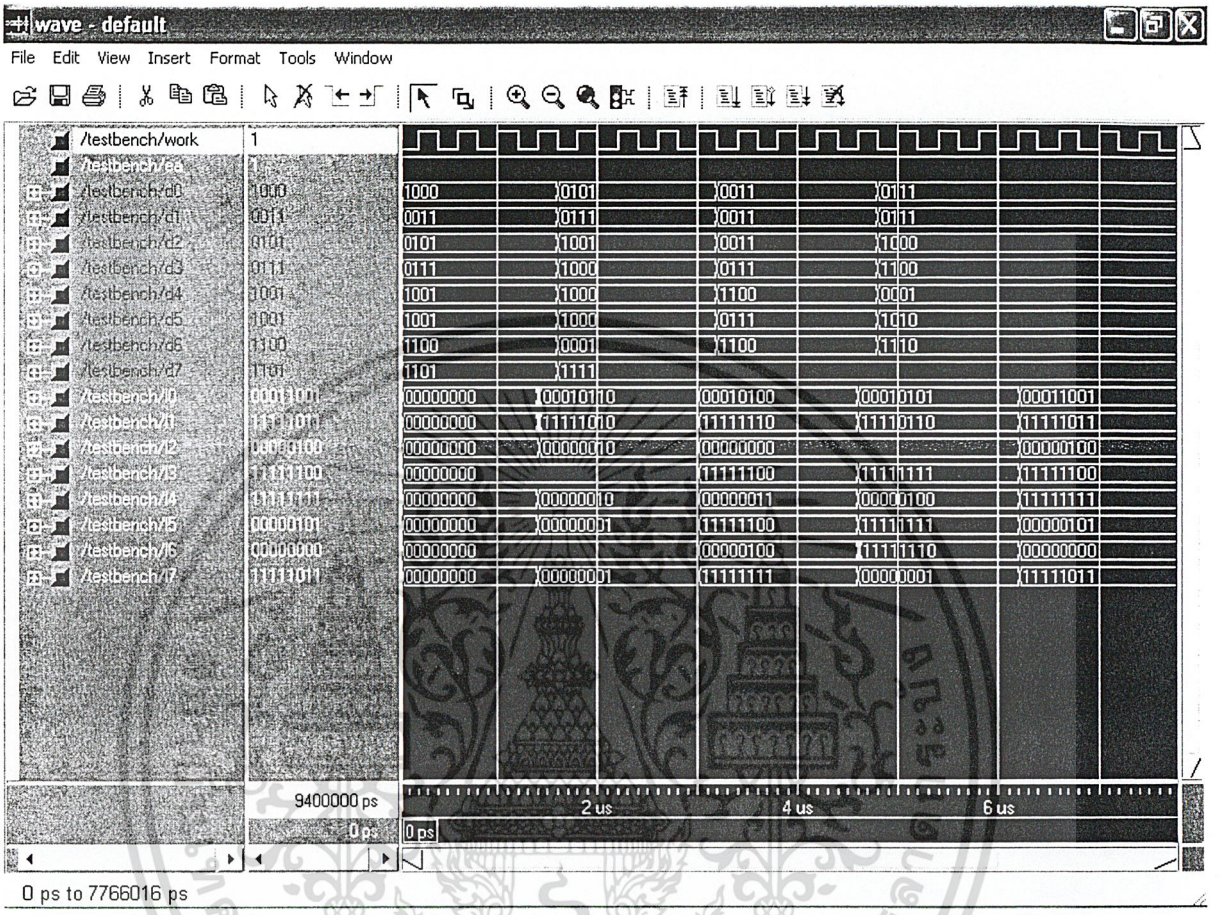
ตัวเลข	0	0	0	1	1	0	1	1
ค่าตำแหน่งฐานสอง	sign	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
ค่าในเลขฐานสิบ	1.6875							

ลำดับขั้นการทดลอง

1. สร้างโปรแกรมสำหรับการทดสอบชื่อ testdct.tbw
2. ป้อนค่าสัญญาณอินพุตใส่ไปในโปรแกรมที่ใช้ทดสอบนั้น
3. ใช้โปแกรม Modelsim Simulator ทำการ Simulate
4. สังเกตและบันทึกผลการทดลอง
5. นำผลการทดลองที่ได้เทียบกับค่าเอาท์พุตที่ได้จากฟังก์ชัน DCT ของโปรแกรม Matlab

ผลการทดลอง

จากผลการจำลองการทำงานที่ได้ โปรแกรมสามารถทำงานได้ถูกต้องตามลำดับที่การทำงานที่ได้เขียนไว้ในโปรแกรม โดยผลการจำลองการทำงานแสดงไว้ในรูปที่ 4.3 และค่าที่ได้จากเอาท์พุตโปรแกรมเมื่อนำมาเปรียบเทียบกับค่าเอาท์พุตที่ได้จากฟังก์ชัน DCT ของโปรแกรม Matlab แสดงให้เห็นในตารางที่ 4.1 – 4.6



รูปที่ 4.3 ผลการจำลองการทำงานของโปรแกรม DCT

ตารางเปรียบเทียบค่าที่ได้จากเอาท์พุตโปรแกรมกับค่าที่ได้จากฟังก์ชัน DCT ของโปรแกรม Matlab

ตารางที่ 4.1

Input		Output	ของโปรแกรม MATLAB	ของวงจร DCT ที่สร้างขึ้น	
				เลขฐานสอง	เลขฐานสิบ
f0	0	F0	0	0 0 0 0 0 0 0 0	0
f1	0	F1	0	0 0 0 0 0 0 0 0	0
f2	0	F2	0	0 0 0 0 0 0 0 0	0
f3	0	F3	0	0 0 0 0 0 0 0 0	0
f4	0	F4	0	0 0 0 0 0 0 0 0	0
f5	0	F5	0	0 0 0 0 0 0 0 0	0
f6	0	F6	0	0 0 0 0 0 0 0 0	0
f7	0	F7	0	0 0 0 0 0 0 0 0	0

ตารางที่ 4.2

Input		Output	ของโปรแกรม MATLAB	ของวงจร DCT ที่สร้างขึ้น	
				เลขฐานสอง	เลขฐานสิบ
f0	0.99609375	F0	2.81737858	0 0 0 0 0 0 0 0	2.8125
f1	0.99609375	F1	0	0 0 0 0 0 0 0 0	0
f2	0.99609375	F2	0	0 0 0 0 0 0 0 0	0
f3	0.99609375	F3	0	0 0 0 0 0 0 0 0	0
f4	0.99609375	F4	0	0 0 0 0 0 0 0 0	0
f5	0.99609375	F5	0	0 0 0 0 0 0 0 0	0
f6	0.99609375	F6	0	0 0 0 0 0 0 0 0	0
f7	0.99609375	F7	0	0 0 0 0 0 0 0 0	0

ตารางที่ 4.3

Input		Output	ของโปรแกรม MATLAB	ของวงจร DCT ที่สร้างขึ้น	
				เลขฐานสอง	เลขฐานสิบ
f0	0.5546875	F0	1.55370142	0 0 0 1 1 0 0 0	1.5
f1	0.234375	F1	-0.45719373	1 0 0 0 0 1 1 1	-0.4375
f2	0.34375	F2	0.18848557	0 0 0 0 0 0 1 0	0.125
f3	0.453125	F3	0.09566212	0 0 0 0 0 0 0 1	0.0625
f4	0.56640625	F4	0.15606067	0 0 0 0 0 0 1 0	0.125
f5	0.609375	F5	0.11871201	0 0 0 0 0 0 0 1	0.0625
f6	0.7890625	F6	0.04002043	0 0 0 0 0 0 0 0	0
f7	0.84375	F7	0.0710100	0 0 0 0 0 0 0 1	0.0625

ตารางที่ 4.4

Input		Output	ของโปรแกรม MATLAB	ของวงจร DCT ที่สร้างขึ้น	
				เลขฐานสอง	เลขฐานสิบ
f0	0.3203125	F0	1.45288346	0 0 0 1 0 1 1 1	1.4375
f1	0.453125	F1	-0.19341921	1 0 0 0 0 0 1 1	-0.1875
f2	0.5625	F2	-0.00359067	1 0 0 0 0 0 0 0	0
f3	0.54296875	F3	-0.31199413	1 0 0 0 0 1 0 1	-0.3125
f4	0.5546875	F4	0.25411649	0 0 0 0 0 1 0 0	0.25
f5	0.55859375	F5	-0.35503851	1 0 0 0 0 1 0 1	-0.3125
f6	0.12109375	F6	0.29447931	0 0 0 0 0 1 0 0	0.25
f7	0.99609375	F7	-0.15078177	1 0 0 0 0 0 1 0	-0.125

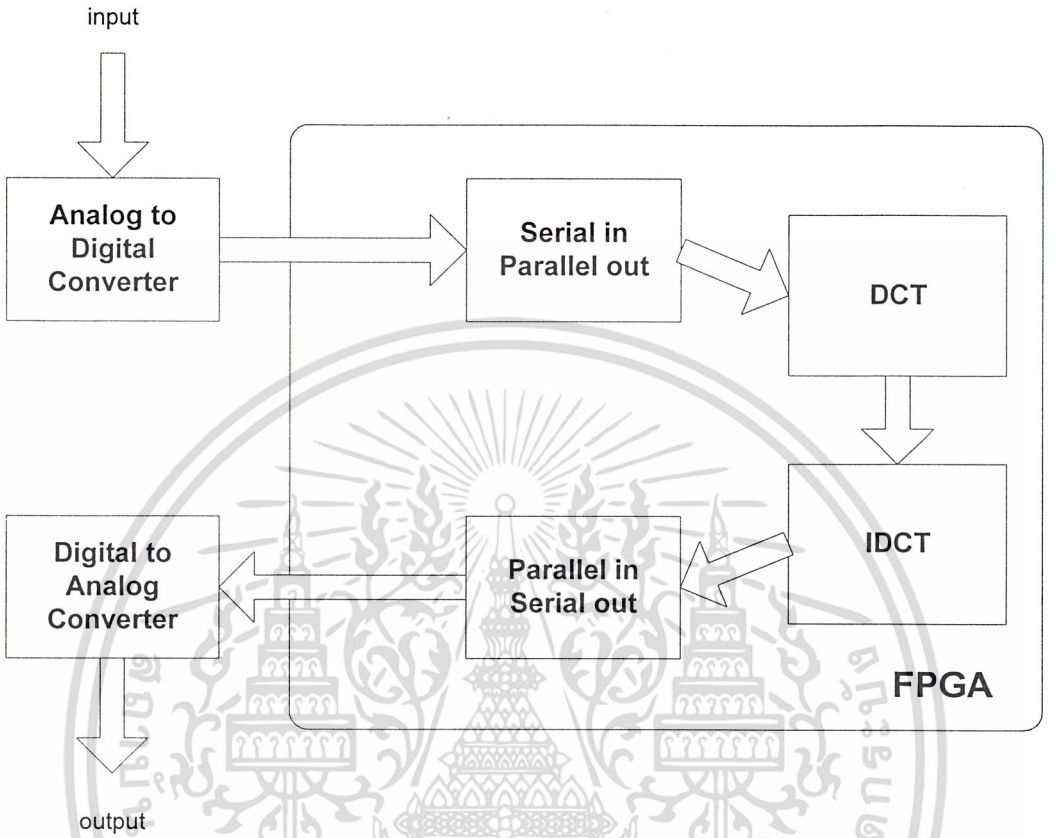
ตารางที่ 4.5

Input		Output	ของโปรแกรม MATLAB	ของวงจร DCT ที่สร้างขึ้น	
				เลขฐานสอง	เลขฐานสิบ
f0	0.234375	F0	1.47774268	0 0 0 1 0 1 1 1	1.4375
f1	0.23046875	F1	-0.69263601	1 0 0 0 1 0 1 1	-0.6875
f2	0.23046875	F2	0.04368245	0 0 0 0 0 0 0 0	0
f3	0.46484375	F3	-0.05627169	1 0 0 0 0 0 0 0	0
f4	0.796875	F4	0.27621358	0 0 0 0 0 1 0 0	0.25
f5	0.453125	F5	-0.09608015	1 0 0 0 0 0 0 1	-0.0625
f6	0.78515625	F6	-0.16160015	1 0 0 0 0 0 1 0	-0.125
f7	0.984375	F7	0.15118478	0 0 0 0 0 0 1 0	0.125

ตารางที่ 4.6

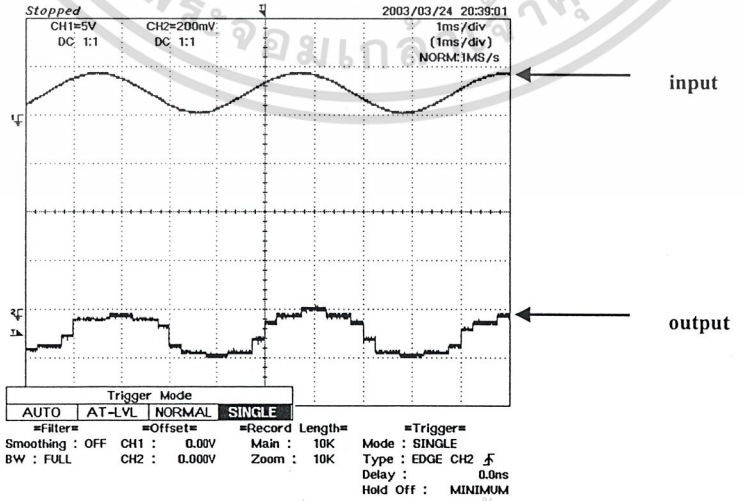
Input		Output	ของโปรแกรม MATLAB	ของวงจร DCT ที่สร้างขึ้น	
				เลขฐานสอง	เลขฐานสิบ
f0	0.453125	F0	1.71942957	0 0 0 1 1 0 1 1	1.6875
f1	0.453125	F1	-0.41881770	1 0 0 0 0 1 1 0	-0.375
f2	0.51171875	F2	0.27816411	0 0 0 0 0 1 0 0	0.25
f3	0.7890625	F3	-0.29542266	1 0 0 0 0 1 0 0	-0.25
f4	0.12109375	F4	-0.05109951	1 0 0 0 0 0 0 0	0
f5	0.6640625	F5	0.31889322	0 0 0 0 0 1 0 1	0.3125
f6	0.875	F6	0.03277150	0 0 0 0 0 0 0 0	0
f7	0.99609375	F7	-0.32667493	1 0 0 0 0 1 0 1	-0.3125

4.4 ส่วนวงจรการแปลงสัญญาณ DCT และการแปลงสัญญาณกลับ IDCT

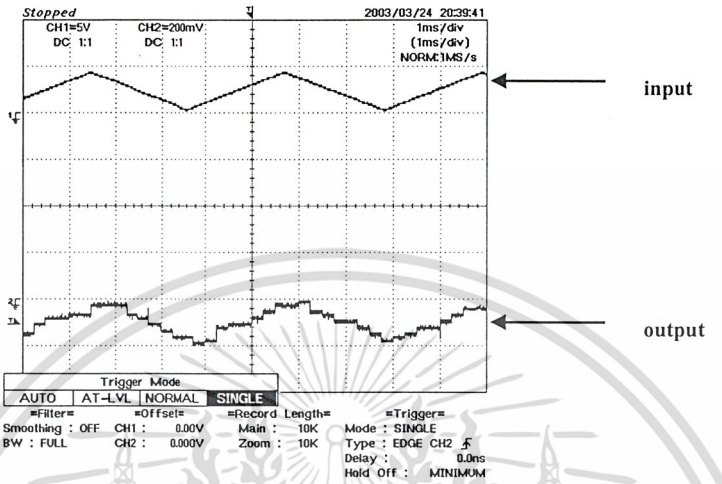


รูปที่ 4.4 วงจรการแปลงสัญญาณ DCT และการแปลงกลับ IDCT

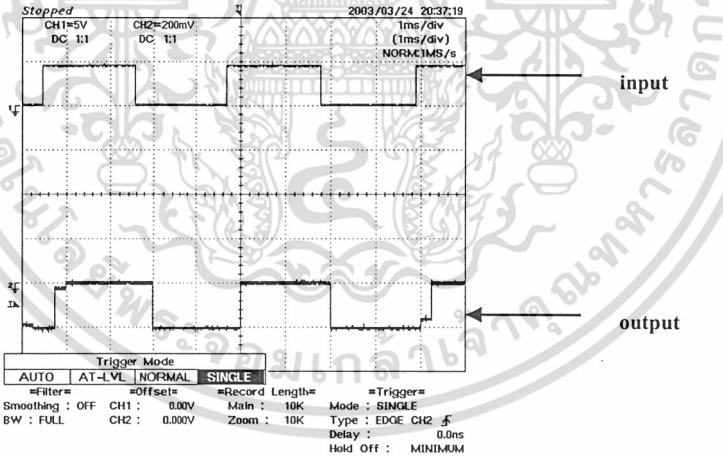
ผลการทดลอง



รูปที่ 4.5 ผลการทดลองการแปลงสัญญาณ โดยรูปคลื่นไซน์



รูปที่ 4.6 ผลการทดลองการแปลงสัญญาณโดยรูปคลื่นสามเหลี่ยม



รูปที่ 4.7 ผลการทดลองการแปลงสัญญาณโดยรูปคลื่นสี่เหลี่ยม

บทที่ 5
บทสรุป แนวทางการแก้ไข และการพัฒนา

5.1 บทสรุป

ในการจัดทำโครงงานนี้ ทำให้ได้เรียนรู้ ศึกษา ค้นคว้า ในสิ่งใหม่ คือทฤษฎีการแปลงสัญญาณเชิงเลข ภาษา VHDL อุปกรณ์โปรแกรมได้ FPGA การใช้โปรแกรม MATLAB , Modelsim SE 5.6 และ Xilinx Project Navigator

จากผลการทดลองค่าที่ได้จากการจำลองการทำงานของโปรแกรม DCT ที่สร้างขึ้นในโปรแกรม Xilinx Project Navigator เมื่อเปรียบเทียบกับค่าที่ได้จากฟังชันก์ DCT ในโปรแกรม MATLAB มีค่าใกล้เคียงกัน ที่ต่างกันเพราะจำนวนบิตทางด้านเอาท์พุตมีจำนวนจำกัด คือมีจำนวนบิตทศนิยม 4 บิต ดังนั้นจะมีค่าความละเอียดเท่ากับ 0.0625 ถ้าต้องการให้ได้ค่าที่ใกล้เคียงมากขึ้นต้องเพิ่มจำนวนบิตให้มากขึ้น

5.2 ปัญหาที่เกิดขึ้นในการทำโครงงาน

1. ผู้จัดทำมีความรู้ความเข้าใจในเรื่องการประมวลผลเชิงเลข ไม่มากเท่าใด ทำให้ต้องใช้เวลาในการหาข้อมูล และทำความเข้าใจมากพอสมควร
2. ข้อจำกัดของภาษา VHDL ที่ไม่สามารถที่จะรองรับการทำงานได้ทุกจุดโดยเฉพาะอย่างยิ่งในส่วนของการคำนวณที่ซับซ้อน ซึ่งในส่วนนี้ทำให้ชุดคำสั่งมีขนาดที่ใหญ่และจำนวนของแต่ละภาคมีจำนวนมาก ทำให้ใช้จำนวนเกตในตัว FPGA มาก

5.3 แนวทางการแก้ไขปัญหา และการพัฒนา

1. จากค่าเอาท์พุตที่ได้ ถ้าต้องการให้มีค่าใกล้เคียงกับค่าที่แท้จริงมากขึ้น ทำได้โดยการเพิ่มจำนวนบิตให้มากขึ้น ซึ่งอาจจะเพิ่มจาก 8 บิตเป็น 16 บิต จะทำให้ค่าเอาท์พุตที่ได้จาก วงจร DCT มีค่าละเอียดมากขึ้น ใกล้ค่าที่ถูกต้องมากขึ้น
2. ในส่วนของการออกแบบชุดทำสั่ง หากต้องการที่จะให้ชุดคำสั่งมีขนาดที่เล็กและใช้จำนวนเกตภายใน FPGA น้อย ควรมีการออกแบบชุดคำสั่งในลักษณะของการทำงานแบบ RTL

บรรณานุกรม

1. ชาติชาย ดิษฐกุล, Introduction to VHDL. กรุงเทพฯ : ภาควิชาวิศวกรรมคอมพิวเตอร์

2. มนัส สังวงศิลป์ และวรรัตน์ ภัทรอมรกุล, คู่มือโปรแกรม MATLAB ฉบับสมบูรณ์. กรุงเทพฯ : อินโฟเพรส. 2543

3. วัชรากร หนูทอง, FPGA Design Workshop. กรุงเทพฯ : ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

4. วัลลภ สุระกำพล, การประมวลผลสัญญาณเชิงเลข. กรุงเทพฯ : สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2533

5. Charles H. Roth, Jr , Digital Systems Design Using VHDL. Boston : PWS Publishing , 1998

6. G. Panneerselvem, P. J .W. Graumann and L.E.Turner, Implementation of Fast Fourier Transforms and Discrete Cosine Transforms in FPGAs. Calgary, Canada : University of Calgary

7. Milos D. Ercegovac, Tomas Lang and Jaime H.Moreno, Introduction to Digital Systems. Singapore : John Wiley & Sons , 1999

ภาคผนวก ก

โปรแกรม Matlab เพื่อใช้ทดสอบ Algorithm DCT

```
%Function Y = testdct(f)
A = cos(pi/4);
B = cos(pi/8);
C = sin(pi/8);
D = cos(pi/16);
E = cos(3*pi/16);
F = sin(3*pi/16);
G = sin(pi/16);
%Input frequency
f = input(Input frequency);
%State1
x00 = f(1)+f(8);
x01 = f(2)+f(7);
x02 = f(3)+f(6);
x03 = f(4)+f(5);
x04 = f(4)-f(5);
x05 = f(3)-f(6);
x06 = f(2)-f(7);
x07 = f(1)-f(8);
%State2
x10 = x00+x03;
x11 = x01+x02;
x12 = x01-x02;
x13 = x00-x03;
x14 = x04;
x15 = (-A*x05)+(A*x06);
x16 = (A*x05)+(A*x06);
x17 = x07;
%State3
```

```
x20 = (A*x10)+(A*x11);
x21 = (A*x10)-(A*x11);
x22 = (C*x12)+(B*x13);
x23 = (-B*x12)+(C*x13);
x24 = x14+x15;
x25 = x14-x15;
x26 = -x16+x17;
x27 = x16+x17;
%State4
Y = zeros(1,8);
Y(1) = x20/2;
Y(2) = ((G*x24)+(D*x27))/2;
Y(3) = x22/2;
Y(4) = ((-F*x25)+(E*x26))/2;
Y(5) = x21/2;
Y(6) = ((E*x25)+(F*x26))/2;
Y(7) = x23/2;
Y(8) = ((-D*x24)+(G*x27))/2;
```

ภาคผนวก ข

1. โปรแกรมรับค่าส่วนหน้า (Serial in Parallel out)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity frontin is
  port (
        datin : in integer range 0 to 15;
        clk :in std_logic;
        ea : in std_logic;
        done : out std_logic;
        datout0 : out integer range 0 to 15;
        datout1 : out integer range 0 to 15;
        datout2 : out integer range 0 to 15;
        datout3 : out integer range 0 to 15;
        datout4 : out integer range 0 to 15;
        datout5 : out integer range 0 to 15;
        datout6 : out integer range 0 to 15;
        datout7 : out integer range 0 to 15
        );
end frontin;
architecture ARCH_front of frontin is
        signal    a0,a1,a2,a3,a4,a5,a6,a7 : integer range 0 to 15;
        signal    ai : integer range 0 to 7;
begin

        process (clk,ea,ai)
        begin
        if clk'event and clk = '1' then
                if ea = '1'then
```

```
                if ai = 7 then
                        done <= '1';
                        datout0 <= a7;
                        datout1 <= a0;
                        datout2 <= a1;
                        datout3 <= a2;
                        datout4 <= a3;
                        datout5 <= a4;
                        datout6 <= a5;
                        datout7 <= a6;
                else done <= '0';
                end if;
                if      (ai = 1) then a1 <= datin;
                elsif   (ai = 2) then a2 <= datin;
                elsif   (ai = 3) then a3 <= datin;
                elsif   (ai = 4) then a4 <= datin;
                elsif   (ai = 5) then a5 <= datin;
                elsif   (ai = 6) then a6 <= datin;
                elsif   (ai = 7) then a7 <= datin;
                else    a0 <= datin;
                end if;
                ai <= (ai+1);
        end if;
    end if;
    end process;
end ARCH_front;
```

2. โปรแกรมแปลงสัญญาณดีสครีตโคไซน์ (DCT)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity com_dctst is
    Port (      clk2 : in std_logic;
                work : in std_logic;
                D0 : in integer range 0 to 15;
                D1 : in integer range 0 to 15;
                D2 : in integer range 0 to 15;
                D3 : in integer range 0 to 15;
                D4 : in integer range 0 to 15;
                D5 : in integer range 0 to 15;
                D6 : in integer range 0 to 15;
                D7 : in integer range 0 to 15;
                L0 : out integer range -128 to 127;
                L1 : out integer range -128 to 127;
                L2 : out integer range -128 to 127;
                L3 : out integer range -128 to 127;
                L4 : out integer range -128 to 127;
                L5 : out integer range -128 to 127;
                L6 : out integer range -128 to 127;
                L7 : out integer range -128 to 127 );
end com_dctst;
architecture Behavioral of com_dctst is
                component dctst1 is
                    Port (clk2 : in std_logic;
                          work : in std_logic;
                          D0 : in integer range 0 to 15;
```

```
        D1 : in integer range 0 to 15;
        D2 : in integer range 0 to 15;
        D3 : in integer range 0 to 15;
        D4 : in integer range 0 to 15;
        D5 : in integer range 0 to 15;
        D6 : in integer range 0 to 15;
        D7 : in integer range 0 to 15;
        I0 : out integer range -128 to 127;
        I1 : out integer range -128 to 127;
        I2 : out integer range -128 to 127;
        I3 : out integer range -128 to 127;
        I4 : out integer range -128 to 127;
        I5 : out integer range -128 to 127;
        I6 : out integer range -128 to 127;
        I7 : out integer range -128 to 127 );
end component;
component dctst2 is
  Port ( clk2 : in std_logic;
        work : in std_logic;
        I0 : in integer range -128 to 127;
        I1 : in integer range -128 to 127;
        I2 : in integer range -128 to 127;
        I3 : in integer range -128 to 127;
        I4 : in integer range -128 to 127;
        I5 : in integer range -128 to 127;
        I6 : in integer range -128 to 127;
        I7 : in integer range -128 to 127;
        J0 : out integer range -128 to 127;
        J1 : out integer range -128 to 127;
        J2 : out integer range -128 to 127;
```

```
        J3 : out integer range -128 to 127;
        J4 : out integer range -128 to 127;
        J5 : out integer range -128 to 127;
        J6 : out integer range -128 to 127;
        J7 : out integer range -128 to 127 );
end component;
component dctst3 is
Port (  clk2 : in std_logic;
        work : in std_logic;
        J0 : in integer range -128 to 127;
        J1 : in integer range -128 to 127;
        J2 : in integer range -128 to 127;
        J3 : in integer range -128 to 127;
        J4 : in integer range -128 to 127;
        J5 : in integer range -128 to 127;
        J6 : in integer range -128 to 127;
        J7 : in integer range -128 to 127;
        K0 : out integer range -128 to 127;
        K1 : out integer range -128 to 127;
        K2 : out integer range -128 to 127;
        K3 : out integer range -128 to 127;
        K4 : out integer range -128 to 127;
        K5 : out integer range -128 to 127;
        K6 : out integer range -128 to 127;
        K7 : out integer range -128 to 127 );
end component;
component dctst4 is
Port (
        clk2 : in std_logic;
        work : in std_logic;
```

```
        K0 : in integer range -128 to 127;
        K1 : in integer range -128 to 127;
        K2 : in integer range -128 to 127;
        K3 : in integer range -128 to 127;
        K4 : in integer range -128 to 127;
        K5 : in integer range -128 to 127;
        K6 : in integer range -128 to 127;
        K7 : in integer range -128 to 127;
        L0 : out integer range -128 to 127;
        L1 : out integer range -128 to 127;
        L2 : out integer range -128 to 127;
        L3 : out integer range -128 to 127;
        L4 : out integer range -128 to 127;
        L5 : out integer range -128 to 127;
        L6 : out integer range -128 to 127;
        L7 : out integer range -128 to 127);
    end component;
        signal a0,a1,a2,a3,a4,a5,a6,a7 : integer range -128 to 127;
        signal b0,b1,b2,b3,b4,b5,b6,b7 : integer range -128 to 127;
        signal c0,c1,c2,c3,c4,c5,c6,c7 : integer range -128 to 127;
begin
            U1 : dctst1 port map
        (       clk2 => clk2,
                work => work,
                D0 => D0,
                D1 => D1,
                D2 => D2,
                D3 => D3,
                D4 => D4,
                D5 => D5,
```

```
D6 => D6,
D7 => D7,
I0 => a0,
I1 => a1,
I2 => a2,
I3 => a3,
I4 => a4,
I5 => a5,
I6 => a6,
I7 => a7);
U2 : dctst2 port map
(       clk2 => clk2,
        work => work,
        I0 => a0,
        I1 => a1,
        I2 => a2,
        I3 => a3,
        I4 => a4,
        I5 => a5,
        I6 => a6,
        I7 => a7,
        J0 => b0,
J1 => b1,
J2 => b2,
J3 => b3,
J4 => b4,
J5 => b5,
J6 => b6,
J7 => b7);
```

```
U3 : dctst3 port map
(       clk2 => clk2,
        work => work,
        J0 => b0,
        J1 => b1,
        J2 => b2,
        J3 => b3,
        J4 => b4,
        J5 => b5,
        J6 => b6,
        J7 => b7,
        K0 => c0,
        K1 => c1,
        K2 => c2,
        K3 => c3,
        K4 => c4,
        K5 => c5,
        K6 => c6,
        K7 => c7);
U4 : dctst4 port map
(       clk2 => clk2,
        work => work,
        K0 => c0,
        K1 => c1,
        K2 => c2,
        K3 => c3,
        K4 => c4,
        K5 => c5,
        K6 => c6,
        K7 => c7,
```

```
                                    L0 => L0,
                                    L1 => L1,
                                    L2 => L2,
                                    L3 => L3,
                                    L4 => L4,
                                    L5 => L5,
                                    L6 => L6,
                                    L7 => L7);
end Behavioral;
```

2.1 โปรแกรมแปลงสัญญาณดิสครีตโคไซน์ ลำดับขั้นตอนที่ 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity dctst1 is
   Port ( clk2 : in std_logic;
          work : in std_logic;
          D0 : in integer range 0 to 15;
          D1 : in integer range 0 to 15;
          D2 : in integer range 0 to 15;
          D3 : in integer range 0 to 15;
          D4 : in integer range 0 to 15;
          D5 : in integer range 0 to 15;
          D6 : in integer range 0 to 15;
          D7 : in integer range 0 to 15;
          I0 : out integer range -128 to 127;
          I1 : out integer range -128 to 127;
          I2 : out integer range -128 to 127;
          I3 : out integer range -128 to 127;
```

```
        I4 : out integer range -128 to 127;
        I5 : out integer range -128 to 127;
        I6 : out integer range -128 to 127;
        I7 : out integer range -128 to 127 );
end dctst1;
architecture Behavioral of dctst1 is
begin
Process(clk2,work)
        variable P0,P1,P2,P3,P4,P5,P6,P7 : integer range -128 to 127;
        begin
                if work = '1' then
                        if ((clk2 = '1') and (clk2'event)) then
                                P0 := D0;
                                P1 := D1;
                                P2 := D2;
                                P3 := D3;
                                P4 := D4;
                                P5 := D5;
                                P6 := D6;
                                P7 := D7;
--state1
                                I6 <= (P1 - P6);
                                I7 <= (P0 - P7);
                                I0 <= (P0 + P7);
                                I1 <= (P1 + P6);
                                I2 <= (P2 + P5);
                                I3 <= (P3 + P4);
                                I4 <= (P3 - P4);
                                I5 <= (P2 - P5);
```

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

2.2 โปรแกรมแปลงสัญญาณดิสครีตโคไซน์ ลำดับขั้นตอนที่ 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity dctst2 is
   Port ( clk2 : in std_logic;
          work : in std_logic;
          I0 : in integer range -128 to 127;
          I1 : in integer range -128 to 127;
          I2 : in integer range -128 to 127;
          I3 : in integer range -128 to 127;
          I4 : in integer range -128 to 127;
          I5 : in integer range -128 to 127;
          I6 : in integer range -128 to 127;
          I7 : in integer range -128 to 127;
          J0 : out integer range -128 to 127;
          J1 : out integer range -128 to 127;
          J2 : out integer range -128 to 127;
          J3 : out integer range -128 to 127;
          J4 : out integer range -128 to 127;
          J5 : out integer range -128 to 127;
          J6 : out integer range -128 to 127;
          J7 : out integer range -128 to 127 );
```

```
end dctst2;
architecture Behavioral of dctst2 is
begin
Process(clk2,work)
        begin
                if work = '1' then
                        if ((clk2 = '1') and (clk2'event)) then
--state2
                                J0  <= (I0 + I3);
                                J1  <= (I1 + I2);
                                J2  <= (I1 - I2);
                                J3  <= (I0 - I3);
                                J4  <= I4;
                                J5  <= ( (I6 * 11) - (I5 * 11) ) / 16;
                                J6  <= ( (I5 * 11) + (I6 * 11) ) / 16;
                                J7  <= I7;
                        end if;
                end if;
        end process;
end Behavioral;
```

2.3 โปรแกรมแปลงสัญญาณดิสครีตโคไซน์ ลำดับขั้นตอนที่ 3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity dctst3 is
   Port (       clk2 : in std_logic;
                work : in std_logic;
                J0 : in integer range -128 to 127;
```

```
        J1 : in integer range -128 to 127;
        J2 : in integer range -128 to 127;
        J3 : in integer range -128 to 127;
        J4 : in integer range -128 to 127;
        J5 : in integer range -128 to 127;
        J6 : in integer range -128 to 127;
        J7 : in integer range -128 to 127;
        K0 : out integer range -128 to 127;
        K1 : out integer range -128 to 127;
        K2 : out integer range -128 to 127;
        K3 : out integer range -128 to 127;
        K4 : out integer range -128 to 127;
        K5 : out integer range -128 to 127;
        K6 : out integer range -128 to 127;
        K7 : out integer range -128 to 127 );
end dctst3;
architecture Behavioral of dctst3 is
begin
Process(clk2,work)
    begin
        if work = '1' then
            if ((clk2 = '1') and (clk2'event)) then
--state3
                K0 <= ( (J0 * 11) + (J1 * 11) ) / 16;
                K1 <= ( (J0 * 11) - (J1 * 11) ) / 16;
                K2 <= ( (J2 * 6 ) + (J3 * 14) ) / 16;
                K3 <= ( (J3 * 6 ) - (J2 * 14) ) / 16;
                K4 <= (J4 + J5);
                K5 <= (J4 - J5);
                K6 <= (J7 - J6);
```

```
                K7 <= (J6 + J7);
            end if;
        end if;
    end process;
end Behavioral;
```

2.4 โปรแกรมแปลงสัญญาณดิสครีตโคไซน์ ลำดับขั้นตอนที่ 4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity dctst4 is
    Port (      clk2 : in std_logic;
                work : in std_logic;
                K0 : in integer range -128 to 127;
                K1 : in integer range -128 to 127;
                K2 : in integer range -128 to 127;
                K3 : in integer range -128 to 127;
                K4 : in integer range -128 to 127;
                K5 : in integer range -128 to 127;
                K6 : in integer range -128 to 127;
                K7 : in integer range -128 to 127;
                L0 : out integer range -128 to 127;
                L1 : out integer range -128 to 127;
                L2 : out integer range -128 to 127;
                L3 : out integer range -128 to 127;
                L4 : out integer range -128 to 127;
                L5 : out integer range -128 to 127;
                L6 : out integer range -128 to 127;
                L7 : out integer range -128 to 127);
```

```
end dctst4;
architecture Behavioral of dctst4 is
begin
Process(clk2,work)
        begin
                if work = '1' then
                        if ((clk2 = '1') and (clk2'event)) then
--state4
                                L0 <= (K0) / 2;
                                L1 <= ( (K4 * 3 ) + (K7 * 15) ) / 32;
                                L2 <= (K2) / 2;
                                L3 <= ( (K6 * 13) - (K5 * 8 ) ) / 32;
                                L4 <= (K1) / 2;
                                L5 <= ( (K5 * 13) + (K6 * 8 ) ) / 32;
                                L6 <= (K3) / 2;
                                L7 <= ( (K7 * 3 ) - (K4 * 15) ) / 32;
                        end if;
                end if;
        end process;
end Behavioral;
```

3. โปรแกรมส่งค่าส่วนหลัง (Parallel in Serial out)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity PISO is
  port ( P0  : in integer range -128 to 127;
         P1  : in integer range -128 to 127;
         P2  : in integer range -128 to 127;
         P3  : in integer range -128 to 127;
         P4  : in integer range -128 to 127;
         P5  : in integer range -128 to 127;
         P6  : in integer range -128 to 127;
         P7  : in integer range -128 to 127;
         clk : in std_logic;
         ea  : in std_logic;
         Sout : out integer range -128 to 127);
end PISO;
architecture ARCH_PISO of PISO is
                signal si : integer range 0 to 7;
begin
         Process (ea,clk,si)
         begin
                  if ea ='1' then
                           if clk ='1' and clk'event then
                                    if      (si=1) then Sout <= P1;
                                    elsif   (si=2) then Sout <= P2;
                                    elsif   (si=3) then Sout <= P3;
                                    elsif   (si=4) then Sout <= P4;
                                    elsif   (si=5) then Sout <= P5;
                                    elsif   (si=6) then Sout <= P6;
```

```
                    elsif    (si=7) then Sout <= P7;
                    else     Sout <= P0;
                    end if;
                si <= (si + 1);
                end if;
            end if;
end process;
end ARCH_PISO;
```