อุปกรณ์แปลงข้อมูลภาพด้วยวิธี Discrete Cosine Transform DCT

DCT:IMAGE TRANSFORM

<VLSI USE VHDL DESIGN>

โดย

กิตินันท์ อ่อนศรี

ทวี ปรากฏ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

อุปกรณ์แปลงข้อมูลภาพด้วยวิธี Discrete Cosine Transform (DCT)

DCT:IMAGE TRANSFORM

<VLSI USE VHDL DESIGN>

โดย

กิตินันท์ อ่อนศรี 36014037

ทวี ปรากฏ 36014157

อาจารย์ที่ปรึกษา

ค.ร.สมศักดิ์ ชุมช่วย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่1

บทนำ

(INTRODUCTION)

ในด้านการประมวลผลต่างๆ ของ Computer หรือ อุปกรณ์ที่ใช้ในการสื่อสารต่างๆ ความสามารถถูกจำกัดด้วยความเร็วในการประมวลผล และเวลาที่ใช้ในการติดต่อแลกเปลี่ยนข้อมูลข่าวสารระหว่างเครื่องและแหล่งข้อมูล ข้อมูลข่าวสารต่างๆ ที่สำคัญนอกจากจะอยู่ในรูปของเสียง เอกสารและสัญลักษณ์ต่างๆ ข้อมูลอีกอย่างหนึ่งที่มีความสำคัญก็คือข้อมูลภาพ ซึ่งข้อมูลภาพที่นำมาใช้ในงานต่างๆจะอยู่ในรูปข้อมูล ทางดิจิตอล การที่จะให้ภาพมีรายละเอียดหรือความคมชัดเพียงพอต่อการใช้งานนั้นจะต้องใช้หน่วยความจำเป็นจำนวนมากสำหรับการเก็บข้อมูล และในกรณีที่มีการรับส่งข้อ มูลภาพด้วยแล้วจะทำให้สิ้นเปลืองเวลาที่ใช้ในการรับส่งข้อมูล เนื่องจากข้อมูลขนาดใหญ่ จากปัญหาดังกล่าวจึงต้องทำให้มีการลดข้อมูลภาพ

วิธีการลดข้อมูลภาพแบบ DCT (Discrete cosine transform) ถูกนำมาประยุกต์อย่างกว้างขว่างเพื่อใช้ในการประมวลผลภาพ(Image processing) จากอัลกอริทึมของมัน สามารถถูกนำไปใช้สร้าง Hardware ซึ่งเป็น VLSI โดยในวิทยานิพนธ์นี้จะใช้โปรแกมภาษา VHDL(Very fast Hardware Description Language)ในการออกแบบ หลังจากนั้น ก็จะทำการทดสอบ และ สร้างต่อไป

บทที่ 1. เป็นบทนำ กล่าวถึงวัตถุประสงค์และขอบเขตของรายงาน

บทที่ 2. เป็นหลักการประมวลผลภาพ(Image processing)บอกถึงลักษณะของข้อมูลภาพ ความหมาย และนิยามของภาพในระบบดิจิตอล การแทนภาพด้วยข้อมูลดิจิตอล การประมวลผลภาพและเทคนิคต่างๆ

บทที่ 3. เป็นวิธีการและหลักการเบื้องต้นของการลดข้อมูลภาพ (Principle of Image Data Compression) เทคนิคต่างๆ ที่นิยมใช้ในกระบวนการลดข้อมูลภาพ ซึ่งมีวิธีการของ Predictive coding, Transform coding โดย Discrete cosine transform (DCT), Hybrid coding

บทที่ 4. เป็นเนื้อหาเกี่ยวกับภาษา VHDL

บทที่ 5. เป็นการออกแบบและสร้างอุปกรณ์

บทที่ 6. เป็นบทสรุปและวิจารณ์

DCT : IMAGE TRANSFORM

Kitinun Onsri

Tawee Prakot

Dr.Somsak Chumchuay Advisor

1996

ABSTRACT

In this report ,we first demonstrate that the forward and inverse discrete cosine transform (DCT, IDCT) can be represented by Chebyshev polynomials of the third and second kind, respectively. Then, we derive recursive algorithms formu lae for the Chebyshev polynomials. the proposed algorithms are particulary suitable for VLSI implementation using array processing architectures. Then,we can accomplished by use VHDL (Very fast Hardware description Language) for design . And then,we implement simulate and verify.

อุปกรณ์แปลงข้อมูลภาพด้วยวิธี DCT

กิตินันท์ อ่อนศรี
ทวี ปรากฏ
ด.ร. สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา

บทคัดย่อ

ในรายงานเล่มนี้ ขั้นแรกจะเป็นการแสดงการแปลงข้อมูลภาพ และ การแปลงกลับ โดยวิธี DCT และ IDCT (the forward and inverse discrete cosine transform) ตามลำดับ ซึ่งแทนได้ด้วยสมการทางคณิตศาสตร์ (Chebyshev polinomials) แล้วต่อไปสร้างอัลกอริทึมของ DCT และ IDCT จาก Chebyshev polinomials โดยมีจุดประสงค์เพื่อให้มีความเหมาะสมในการสร้างอุปกรณ์ VLSI ซึ่งจะถูกทำการออกแบบโดยโปรแกมภาษาVHDL (Very fast Hardware Description Language) แล้วทำการสร้าง และ ทดสอบต่อไป

บทที่ 2

หลักการประมวลผลภาพ

(Image processing)

ในการประมวลผลสัญญาณภาพด้วยระบบคอมพิวเตอร์ จำเป็นต้องเปลี่ยนข้อมูลหรือสัญญาณภาพที่อยู่ในรูปอนาลอก ให้เป็นสัญญาณทางดิจิตอล เพื่อประโยชน์ในการคำนวณและประมวลผลได้ง่าย ในบทนี้ จะกล่าวถึง ความหมายของภาพระบบดิจิตอล และคณิตศาสตร์ที่เกี่ยวข้อง

2.0 ลักษณะข้อมูลภาพ

ซึ่งแบ่งตามการจัดเก็บข้อมูลได้เป็น

1. ภาพ 2 ระดับ คือ มีแต่จุดขาวกับดำเท่านั้น โดยแต่ละจุดเป็นข้อมูล 1 bit
2. ภาพ 16 ระดับ ซึ่ง ในแต่ละจุดภาพจะเป็นข้อมูล 4 bit ซึ่งทำให้สามารถแสดงภาพได้ 16 ระดับสี หรือ 16 ระดับ Graylevel ขึ้นอยู่กับว่าภาพนั้น เป็นภาพสี หรือขาว - ดำ
3. ภาพ 256 ระดับ ซึ่งในแต่ละจุดภาพจะเป็นข้อมูล 8 bit ซึ่งทำให้สามารถแสดงภาพได้ 256 ระดับสี หรือระดับ Graylevel ขึ้นอยู่กับว่า ภาพนั้นเป็นภาพสี หรือภาพขาว - ดำ
4. ภาพ TRUE COLOR ซึ่ง ในแต่ละจุดภาพจะเป็นข้อมูลขนาด 24 bit ทำให้สามารถแสดงผลภาพได้เหมือนภาพจริงที่สุด เพราะสามารถแสดงสีได้ถึง 16,777,216 สี ภาพ True color สามารถแสดงได้เฉพาะภาพสีเท่านั้น ไม่สามารถแสดงภาพขาว - ดำได้

การแสดงภาพนี้ ใช้วิธี ตั้งค่าของแม่สีในตารางสี โดยอาจเลือกสีเป็นแบบ 16 สี จาก 64 สี หรือ 16 สี จาก 262,144 สี หรือ 256 สี จาก 262,144 สี ขึ้นอยู่กับ mode การแสดงผล สำหรับ True Color ไม่มีการเลือกสี แสดงผลโดยการส่งค่าสี RGB ผ่าน D/A สีละ 8 bit ออกไปเลย ความแตกต่างของการแสดงภาพสี และภาพขาว - ดำ คือ ภาพขาว - ดำ จะต้องตั้งให้แม่สีทั้ง 3 สี มีค่าเท่ากัน เนื่องจาก VGA กำหนดให้แม่สะแต่ละสีใช้ register 6 bit ออกไปเลย ทำให้แต่ละแม่สีแสดงผลได้เพียง 34 ระดับเท่านั้น ยังผลให้เราแสดงภาพ 256 ระดับให้เห็นได้เพียง 64 ระดับเท่านั้น หากต้องการให้เห็นจริงทั้ง 256 ระดับ ต้องแสดงใน

mode True color แล้วให้ RGB มีค่า เท่ากัน ซึ่ง mode นี้ register 8 bit สำหรับแม่สีแต่ละสี

2.1 ความหมายและนิยามของภาพในระบบดิจิตอล

ภาพ (Image) ในเชิงคณิตศาสตร์จะหมายถึง ฟังก์ชัน 2 มิติ f(x,y) โดย x และ y เป็นพิกัดในระนาบ 2 มิติ ค่าฟังก์ชัน f(x,y) จะเป็นสัดส่วนกับความสว่าง หรือความเข้มของภาพ ที่ตำแหน่ง (x,y) ซึ่งเราเรียกว่า ระดับสีเทา (Gray level) ในรูปที่ 2.1 แสดงให้เห็นถึงระนาบและจุดพิกัดของภาพ ซึ่งปกติเราจะให้จุดกำเนิดของแกนพิกัด (Coordinate) อยู่ทางมุมบนซ้ายของภาพ Origin

รูปที่ 2.1 ระนาบและพิกัดที่ใช้ในระบบภาพ

ภาพ 2 มิติ ที่แทนด้วยฟังก์ชัน f(x,y) โดย x และ y เป็นแกนในระนาบของภาพ ค่าของฟังก์ชันที่จุด (x,y) คือความเข้มของแสงที่จุดนั้น เนื่องจากแสงเป็นพลังงานรูปหนึ่ง ดังนั้น f(x,y) ต้องไม่เป็นศูนย์ และมีค่า (finite) นั้นคือ

$$0 < f(x,y) < \alpha \qquad \text{..... (2.1.1)}$$

โดยธรรมชาติของแสง ซึ่งจะต้องมีแหล่งกำเนิดแสง และส่วนที่สะท้อนของแสง ดังนั้นเราสามารถแยกฟังก์ชัน f(x,y) ออกเป็น 2 ส่วนคือ อิลูมิเนชั่นคอมโพเนนท์ (illumination component) และรีเฟลีกแทนท์คอมโพเนนท์ (reflectance component) จะได้ว่า

$$f(x,y) = I(x,y) \qquad .....(2.1.2)$$

เมื่อ

$$0 < I(x,y) < \alpha \qquad .....(2.1.3)$$

และ

$$0 < r(x,y) < 1 \qquad .....(2.1.4)$$

สมการ (2.4) แสดงให้เห็นว่า ฟังก์ชั่นการสะท้อนถูกจำกัดขอบเขตระหว่าง 0 (ซึ่งหมายถึง การดูดซึมสมบูรณ์) และ 1 (ซึ่งหมายถึง การสะท้อนโดยสมบูรณ์) ธรรมชาติของ $i(x,y)$ ขึ้นอยู่กับแหล่งกำเนิดแสง ในขณะที่ $r(x,y)$ ขึ้นอยู่กับวัตถุที่สะท้อนแสงมาเข้าตา

ดังที่กล่าวมาแล้ว ความเข้มของภาพที่จุด (x,y) เราเรียกว่า ระดับสีเทา (Gray level) I จากสมการที่ (2.2) ถึง (2.4) จะเห็นว่า I ควรอยู่ในช่วง

$$I_{min.} \leq I \leq L_{max.} \qquad ...(2.1.5)$$

ในทางทฤษฎี L_{min} ต้องมีค่าบวก ในขณะที่ $L_{max.}$ ต้องมีค่าน้อยกว่าอนันต์ ในทางปฏิบัติ $L_{min} = L_{min}\, r_{min}$ และ $L_{max.} = L_{max.}\, r_{max.}$ ช่างของ $(L_{min}, L_{max.})$ เราเรียกว่าช่วงของระดับสีเทา ในทางปฏิบัติโดยใช้หลักคณิตศาสตร์ เรานิยมปรับช่วง $(L_{min}, L_{max.})$ ให้เป็นช่วง (O,L) โดย L = 0 หมายถึง ดำสนิท และ L = 1 หมายถึงขาว

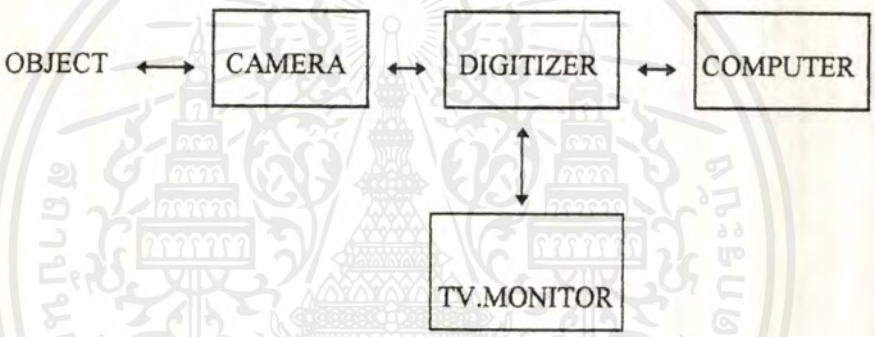
2.2 การแทนภาพด้วยข้อมูลแบบดิจิตอล

ภาพดิจิตอล (digital image) เป็นภาพที่ถูกแปลนมาจากภาพอนาลอก อยู่ในรูปตัวเลข โดยภาพอนาลอกถูกแบ่งเป็นพื้นที่สี่เหลี่ยมเล็ก ๆ ที่เรียกว่า Pixel ในแต่ละ Pixel จะถูกระบุตำแหน่งโดย (x,y) และค่าระดับสีเทาของ Pixel นั้นคือ ค่าของ (x,y) รูป 2.2 เป็นภาพดิจิตอลขนาด 64*64 Pixel

รูปที่ 2.2 แสดงภาพดิจิตอลขนาด 64*64 Pixel

2.3 ระบบการประมวลผลทางดิจิตอล

ระบบการประมวลผลภาพประกอบด้วย 3 ส่วนใหญ่ ๆ คือ ส่วยเปลี่ยนสัญญาณอนาลอก ให้เป็นสัญญาณทางด้านดิจิตอล ซึ่งเรียกว่า ดิจิไดเซอร์ (Digitizer) ส่วนประมวลผล (Processing) และส่วนแสดงผล (display) แสดงในรูป 2.3



รูปที่ 2.3 ระบบประมวลผลภาพดิจิตอล

จากรูปที่ 2.3 ส่วนแรกคือ ส่วนที่เปลี่ยนสัญญาณอนาลอก ให้เป็นสัญญาดิจิตอล กล้อง (CAMERA) เปรียบเสมือนดวงตาของมนุษย์ ทำหน้าที่เปลี่ยนภาพวัตถุมาเป็นสัญญาณทางไฟฟ้า และส่งให้ดิจิไดเซอร์ (Digitizer) ซึ่งทำหน้าที่เปลี่ยนสัญญาณไฟฟ้า ให้เป็นสัญญาณดิจิตอล อุปกรณ์ส่วนนี้ได้แก่ กล้องโทรทัศน์ดิจิไดเซอร์ ซึ่งภาพในประกอบด้วย หลอดวิดิคอน ทำหน้าที่เป็นสื่อนำไฟฟ้าทางแสง ภาพถูกโฟกัสลงบนผิวของหลอด และถูกเปลี่ยนให้เป็นสัญญาณไฟฟ้าที่สอดคล้องกับความสว่างของภาพในตำแหน่งนั้น ๆ จานนั้น ทำการควอนไตซิ่ง (quantizing) ข้อมูลภาพที่ได้เป็นสัญญาณดิจิตอล

ส่วนประมวลผลคือ คอมพิวเตอร์ ซึ่งเปรียบเสมือนสมอง ทำหน้าที่ประมวลผลและวิเคราะห์ข้อมูลภาพ

ส่วนแสดงผล ทำหน้าที่เปลี่ยนข้อมูลตัวเลข (ซึ่งเป็นระดับสีเทา) ที่เก็บเป็น array ในคอมพิวเตอร์ ให้อยู่ในรูปที่เหมาะสม และสื่อความหมายกับมนุษย์ได้ คือเป็นภาพที่ปกติทั่ว

ๆ ไป อุปกรณ์ในส่วนนี้ได้แก่ monitor ทีวี เครื่องพิมพ์ที่สามารถแสดงผล ในรูปกราฟฟิก ได้

ภาพ 1 ภาพ ที่ถูกเปลี่ยนจาก สัญญาณดิจิตอล สำหรับคอมพิวเตอร์นี้มีขนาดใหญ่ ขึ้นอยู่กับความละเอียดของภาพที่ต้องการ และจะมีผลทำให้ใช้เนื้อที่ในหน่วยความจำมากในการเก็บข้อมูลภาพ 1 ภาพ เช่น การเก็บ ภาพ 1 ภาพ ขนาด 256*265 จุด 2 ที่มีความแตกต่างของระดับความเข้มของแต่ละจุด เท่ากับ 256 ระดับ จะต้องใช้เนื้อที่ในหน่วยความจำถึง 64 Kbytes ในการเก็บภาพนี้ดังนั้นในปัจจุบันได้มีการค้นคว้าวิจัย หาวิธีการที่จะเก็บภาพด้วยคอมพิวเตอร์ โดยให้เนื้อที่ในหน่วยความจำให้น้อยที่สุด และยังรักษาความละเอียดของภาพตามการใช้งานได้อีกด้วย

2.4 การสุ่มแบบสม่ำเสมอ และควอนไตเซชั่น (Uniform sampling and Quantization)

เพื่อที่จะประมวลสัญญาณภาพด้วยระบบคอมพิวเตอร์ ฟังก์ชันของภาพ f(x,y) จะถูกทำให้เป็นสัญญาณไม่ต่อเนื่อง ทั้งระนาบของภาพ ซึ่งเรียบกว่า การสุ่มภาพ (Image sampling) ของฟังก์ชั่นที่ได้เรียกว่า การควอนไดเซชั่นระดับสีเทา (gray level quantization)

สมมุติว่าสัญญาณภาพต่อเนื่อง f(x,y) ถูกดิจิไดซ์ ในระนาบ X Y เป็นช่วงเท่ากัน เราสามารถจัด f(x,y) ให้อยู่ในรูปเมตริกซ์ นาด N * N ได้ดังสมการ (2.4.1)

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \ldots\ldots & f(0,N-1) \\ f(1,0) & f(1,1) & \ldots\ldots & f(1,N-1) \\ \ldots\ldots & \ldots\ldots & & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & & \ldots\ldots \\ f(N-1,0) & f(N-1,1) & \ldots\ldots & f(N-1,N-1) \end{bmatrix}$$

ทางขวาของสมการ จะเรียกว่า ภาพดิจิตอล และทุก ๆ สมาชิกของเมตริกซ์จะเรียกว่า พิกเซล จากขบวนการสร้างภาพดิจิตอลข้างต้น จะเห็นว่า เราต้องการทราบขนาดความละเอียดของภาพ N * N พิกเซล และจำนวนระดับของ Gray level ในทางปฏิบัติการทำควอนไดเซชั่น ในระบบภาพดิจิตอล จะเป็นค่าของ 2 ยกกำลังจำนวนเต็ม คือ

$$N = 2n \qquad \ldots\ldots(2.4.2)$$

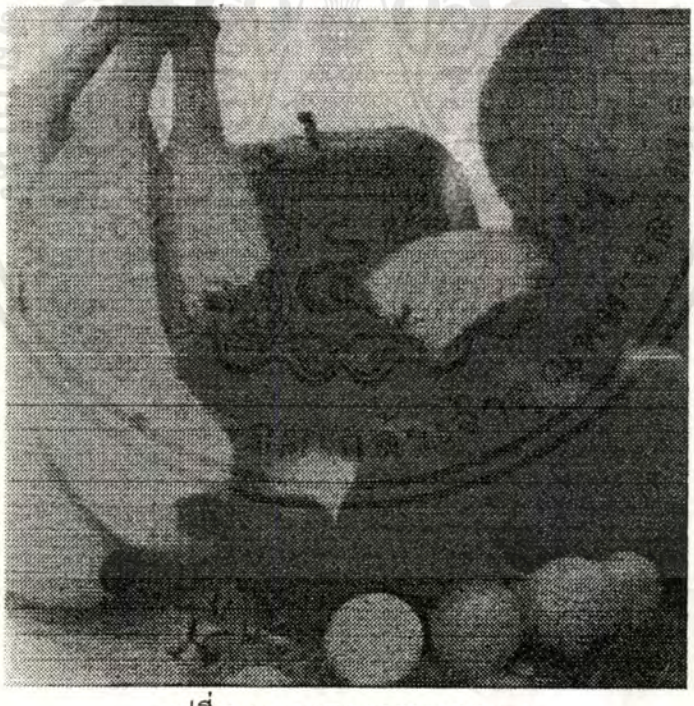
และ

$$G = 2m \qquad .....(2.4.3)$$

เมื่อค่า G คือ จำนวนระดับของ Gray level ดังนั้นจำนวนบิท (bit) ที่ใช้ในการเก็บภาพหนึ่งภาพที่ถูกดิจิไดซ์ คือ

$$B = N * N * m \quad \text{บิท} \qquad .....(2.4.4)$$

ดังตัวอย่างภาพขนาด 128 * 128 Pixel และระดับ Gray level จำนวน 256 ระดับ ต้องใช้หน่วยความจำขนาด 131,072 บิท ในรูปที่ 2.4 ได้แสดงการเปรียบเทียบภาพเมื่อลดความละเอียดของภาพลง และตาราง 2.1 แสดงจำนวนไบท์ที่ใช้ในการเก็บภาพ เมื่อ N และ M เปลี่ยนไป



รูปที่ 2.4.1 ภาพ 256 X 256 Pixel

รูปที่ 2.4.2 ภาพ 128 X 128 Pixel

รูปที่ 2.4.3 ภาพ 64 X 64 Pixel

รูปที่ 2.4 เปรียบเทียบภาพเมื่อลดความละเอียดของภาพลง

ตารางที่ 2.1 จำนวน BYTE ที่ใช้ในการเก็บภาพ เมื่อ N และ M เปลี่ยนไป

m N	1	2	3	4	5	6	7	8
32	128	256	512	1024	1024	1024	1024	1024
64	512	1024	2048	2048	4096	4096	4096	4096
128	2048	4096	8192	8192	16384	16384	16384	16384
256	8192	16384	32768	32768	65536	65536	65536	65536
512	32768	65536	131072	131072	262144	262144	262144	262144

2.5 เทคนิคต่าง ๆ สำหรับการประมวลผลภาพ

เทคนิคต่าง ๆ สำหรับการประมวลผลภาพ แบ่งเป็น 4 พวกใหญ่ ๆ คือ

1. อิเมจดิจิไตเซชั่น (Image digitization)
2. อิเมจเอนฮานเมนต์และรีสตอเรชั่น (Image enhancement and restoration)
3. อิเมจเอนโค้ดดิ้ง (Image Encoding)
4. อิเมจรีคอนสตรัคชั่น (image reconstruction)

2.5.1 อิเมจดิจิไตเซชั่น (Image digitization)

ดังได้กล่าวมาแล้วถึงความหมายของการ digitize ภาพ ซึ่งความละเอียดของภาพที่ได้ขึ้นอยู่กับการจัดระดับภาพ ในปัจจุบันเครื่องมือที่ใช้ทำขบวนการนี้ที่เรียกว่า ดิจิไตเซอร์ (digitizer) ดิจิไตเซอร์ สามารถเปลี่ยนสัญญาณอนาลอก เป็นสัญญาณดิจิตอลได้ ดังนั้นข้อมูลที่ได้จึงเป็นเลขฐานสอง โดยใช้ดิจิไตเซอร์เป็นตัวจัดการ

2.5.2 อิเมจเอนฮานเมนต์และรีสตอเรชั่น (Image enhancement and restoration)

อิเมจเอนฮานเมนต์ เป็นการทำภาพให้อยู่ในรูปที่เหมาะสมขึ้น มีความคมชัดมากยิ่งขึ้น สำหรับการนำไปใช้งานเฉพาะอย่าง กล่าวคือ วิธีที่ทำภาพ หรือปรับปรุงภาพ X-ray อาจจะไม่เป็นวิธีที่ดี เมื่อเรานำมาปรับปรุงภาพถ่ายดาวเคราะห์ ที่ได้ส่งมาจากการสำรวจทางอวกาศ

วิธีปรับปรุงคุณภาพของภาพ (enhancement) มีหลายวิธีดังนี้

1. คอนทราสท์เอนฮานเมนต์ (Contrast enhancement) เป็นวิธีที่ทำให้ภาพคมชัดขึ้น โดยอาศัยฮีสโตแกรม อาจใช้แบบลิเนียร์สเทรท (linear stretch) , พีชลิเนียร์สเทรท (picecwise lenear stretch) หรืออีควอไลเซชั่น (equalization)

2. เอดจ์เอนฮานเมนท์ (Edge enhancement) เป็นการแยกความแตกต่างของจุดภาพที่ใกล้เคียงกัน เพื่อหาขอบเขตของภาพ

3. การประมวลผลภาพสีเทียม (Pseudo - color image processing) เป็นการใช้เทคนิคของการทำ density slicing และการใส่สีเทียมให้กับภาพขาว - ดำ ที่มีระดับ Gray level ต่าง ๆ กัน

4. การกรองภาพ (Filtering) เพื่อให้ภาพเรียบ (smoothing) หรือคมชัด (sharpening) โดยใช้ตัวกรองความถี่ต่ำ (low pass filter) หรือ ตัวกรองความถี่สูง (high pass filter) ตามลำดับ

อิเมจรีสตอเรชั่น (Image restoration) เป็นขบวนการในการสร้างภาพกลับคืน โดยการหาค่า ชดเชย และแก้ความคลาดเคลื่อน เนื่องมาจากข้อมูลในภาพผิดพลาดไป หรือเป็นขบวนการสร้างภาพกลับคืน จากภาพที่ถูกทำให้เสียไป เนื่องจากปรากฏการณ์ต่าง ๆ โดยใช้หลักการของพีชคณิตเชิงเส้น (linear Algarve)

2.5.3 อิเมจเอนโค้ดดิ้ง (Image Encoding)

เป็นการใช้เทคนิคต่าง ๆ เพื่อเข้ารหัสข้อมูล เนื่องจากข้อมูลภาพที่ได้จะถูกเก็บในลักษณะเป็นจำนวนไบท์ ดังตาราง 2.1 ซึ่งถ้าภาพมีขนาดใหญ่ ก็ต้องใช้พื้นที่ในการเก็บมาก ด้วยข้อจำกัดของเครื่องไมโครคอมพิวเตอร์ ที่มีขนาดหน่วยความจำจำกัด

การเข้ารหัสข้อมูล จึงมีประโยชน์ในด้านการลดพื้นที่ในการเก็บข้อมูลภาพดังกล่าวมาก ผลของการเข้ารหัสข้อมูลนี้ เรียกว่าเป็นการลดข้อมูล (Data reduction หรือ Data compression) ซึ่งเป็นเนื้อหาที่ทำในโครงงานนี้ รายละเอียดของการลดข้อมูลภาพ ได้อธิบายในบทที่ 3 นอกจากนี้ การเข้ารหัสข้อมูลยังมีประโยชน์ในการช่วยลดปริมาณข้อมูลภาพ ที่ใช้ในระบบสื่อสาร เช่น การส่งภาพถ่ายจากอวกาศ มายังโลก การส่งข้อมูลผ่านโมเด็ม(modem) เป็นต้น

2.5.4 อิเมจรีคอนสตรัคชั่น (image reconstruction)

เป็นวิธีการสร้างภาพตัดขวางของวัตถุโดยไม่ต้องผ่า หรือทำลายวัสดุ เพื่อประมวลผลโดยใช้คอมพิวเตอร์ เราเรียกการสร้างภาพตัดขวางด้วยคอมพิวเตอร์ว่า คอมพิวเตอร์โทโมกราฟฟี่ (Computer tomography)

บทที่ 3

หลักการเบื้องต้นของการลดข้อมูลภาพ

(PRINCIPLE OF IMAGE DATA COMPRESSION)

สัญญาณภาพโดยธรรมชาติแล้ว จะเป็นสัญญาณต่อเนื่อง(Analog) ที่มีความกว้างของย่านความถี่สูงมาก ซึ่งเมื่อเทียบกับสัญญาณเสียงแล้วความกว้างของย่านความถี่สูงกว่าของสัญญาณเสียงเป็น 1000 เท่า หรือ มากกว่า การนำข้อมูลภาพมาใช้ในการประมวลผลภาพในด้านต่าง ๆ ที่มีเครื่องคอมพิวเตอร์เป็นตัวประมวลผลแล้วนั้น สัญญาณภาพจะต้องแปลงให้เป็นสัญญาณที่เครื่องสามารถเข้าใจได้เสียก่อน นั่นคือ ต้องแปลงให้อยู่ในรูปของสัญญาณดิจิตอล ซึ่งอยู่ในลักษณะที่ไม่ต่อเนื่อง (discrete signal) โดยผ่านการสุ่ม และจัดระดับ (sampled and quantized) เมื่อรายละเอียดของภาพ ๆ หนึ่ง ถูกแทนที่ด้วยข้อมูลแบบดิจิตอล ขนาดของข้อมูลภาพจึงมีจำนวนสูงมาก เพื่อที่จะสามารถเก็บรายละเอียดของภาพได้เพียงพอ ทำให้ต้องใช้ขนาดของหน่วยความจำในการเก็บข้อมูลเหล่านี้มีขนาดใหญ่ตามไปด้วย ในกรณีที่มีการรับส่งข้อมูลภาพเหล่านี้ด้วยแล้ว จะต้องเสียเวลาในการรับส่งเป็นอันมาก โดยเฉพาะอย่างยิ่งหากเป็นการสื่อสารข้อมูลผ่านดาวเทียมแล้ว เวลาที่มากนั้นก็หมายถึงค่าใช้จ่ายที่สูงมากตามไปด้วย ดังนั้นจึงจำเป็นอย่างยิ่งที่จะต้องลดเวลาในการส่งข้อมูลให้เหลือน้อยที่สุด เท่าที่จะเป็นไปได้

การลดขนาดของข้อมูลภาพ จึงถูกนำมาใช้ในการแก้ปัญหาเหล่านี้ เพื่อเป็นการประหยัดทางด้านเศรษฐกิจ ในส่วนของจำนวนหน่วยความจำที่ลดลง และในส่วนของเวลาที่ใช้ไปในการรับส่งข้อมูลภาพ สิ่งที่เป็นตัวกำหนดว่าข้อมูลของภาพ สามารถที่จะลดลงไปได้เท่าไหร่นั้น ขึ้นอยู่กับงานที่ใช้ว่าต้องการรายละเอียดมากน้อยเพียงใด เมื่อทำการลดข้อมูลแล้ว จึงจะสามารถนำข้อมูลเดิมกลับมา (reconstruct) ได้อย่างเหมาะสม ซึ่งงานแต่ละอย่างมีความต้องการรายละเอียดของภาพที่ไม่เท่ากัน อย่างเช่นถ้านำไปใช้ในการตรวจสอบและจดจำรูปแบบวัตถุต่าง ๆ โดยใช้คอมพิวเตอร์แล้ว ภาพที่ใช้จะเน้นเฉพาะส่วนของขอบวัตถุ ในภาพนั้น ๆ ก็เพียงพอ โดยไม่จำเป็นต้องมีรายละเอียดโครงสร้างภายในของภาพ ในขณะที่การลดข้อมูลภาพ เพื่อการลดเวลาที่ใช้ไปในการรับส่งภาพผ่านช่องสัญญาณความเร็วต่ำ อย่างเช่น ระบบโครงข่ายโทรศัพท์สาธารณะนั้น มีความจำเป็นอย่างยิ่งที่จะต้องเก็บรายละเอียดต่าง ๆ ของภาพให้ได้มากที่สุด หลักการพื้นฐานที่นิยมใช้ในการลดข้อมูลภาพ มีอยู่สามหลักการด้วยกัน คือ

1. Predictive Coding ซึ่งเป็นการเข้ารหัสใน data domain
2. Transform Coding เป็นวิธีที่ประมวลผล (process) ในโดเมนของความถี่ และ

3. Hybrid Coding หลักการสุดท้ายเป็นการรวมเอาข้อดีต่าง ๆ จากสองหลักการแรกเข้าด้วยกัน

3.1 Predictive Coding

การลดขนาดของข้อมูลภาพด้วยวิธี Predictive Coding นี้ เป็นการอาศัยคุณสมบัติของข้อมูลภาพที่มักจะมีค่าซ้ำ ๆ กัน และเมื่อข้อมูลอินพุทถูกกำหนดให้มีความเกี่ยวพันธ์กันนั้น ก็คือ จุดภาพที่มีตำแหน่งอยู่ใกล้ ๆ กัน มักจะมีค่าระดับแอมปลิจูด ใกล้เคียงกันหรือ เท่ากัน ดังนั้นจึงสามารถที่จะใช้ค่าของจุดภาพหนึ่งจุดหรือหลาย ๆ จุด ที่ผ่านมาใน Line นั้น Line ก่อนหน้านั้น หรือในเฟรมที่ผ่านมา เป็นตัวคาดคะเน หรือแทนค่าของจุดภาพปัจจุบัน ซึ่งโดยธรรมชาติทางสถิติของข้อมูลภาพเราสามารถที่จะคาดคะเนค่าของข้อมูลได้ไม่ผิดพลาดมากนัก จากค่าที่ได้จากการคาดคะเนนี้เอาไปลบกับค่าจริงของจุดภาพ จะได้เป็นค่าความแตกต่างระหว่างค่าจริง กับค่าที่เราคาดคะเนเอาไว้ ซึ่งค่านี้จะมีขนาดเล็ก และค่าผลต่างนี้จะถูกนำไปเข้ารหัสเพื่อจะเก็บไว้ใช้ในตอนถอดรหัส พร้อมกับค่าที่เราคาดคะเนเอาไว้ ดังนั้นในการที่จะเก็บในหน่วยความจำ หรือต้องการส่งก็ใช้ค่าสองค่านี้ เมื่อถึงตอนที่จะนำภาพเดิมกลับมา หรือตอนถอดรหัส จะนำเอาค่าที่เราคาดคะเนไว้ในตอนแรกบวกกับค่าของผลต่างของจุดภาพนั้น กับค่าคาดคะเน ก็จะได้เป็นค่าของจุดภาพนั้น ๆ ค่าผิดพลาดที่ได้ในตอนถอดรหัส เกิดขึ้นเพียงกรณีเดียวเท่านั้น คือ ตอนจัดระดับ (quantized) ค่าความแตกต่างของการเข้ารหัสเท่านั้น วิธีนี้เป็นวิธีที่ง่ายแก่การสร้างระบบ และสามารถลดขนาดข้อมูลภาพให้เหลือประมาณ 1 - 2 bit/element

3.2 Transform Coding

การลดข้อมูลภาพด้วยวิธีของ Transform Coding นี้เป็นวิธีที่ซับซ้อน และมีขั้นตอนมากกว่าวิธีการลดข้อมูลภาพ โดย Predictive Coding หลักการของวิธี Transform Coding จะทำการแปลงข้อมูลอินพุทที่อยู่ในรูปของ data domain ให้อยู่ในรูปของ spectral หรือ frequency domain โดยใช้วิธีการ Transform แบบต่าง ๆ เช่น Fourier Transform จะเป็นการแปลงที่อยู่ในรูปของ spatial domain ให้อยู่ในรูปสัมประสิทธิ์ของพลังงานความถี่ โดยค่าความถี่ต่ำ ๆ จะมีพลังงานสูง ที่ความถี่สูงพลังงานจะลดลงไป การเข้ารหัสจึงใช้จำนวนบิทสำหรับแต่ละช่วงความถี่ไม่เท่ากัน เมื่อต้องการอัตราบิทเรทสูง ๆ ค่าของพลังงานความถี่สูงจะถูกตัดทิ้งไปเป็นส่วนใหญ่ เป็นเหตุให้รายละเอียดส่วนที่เป็นขอบภาพขาดหายไป ทำให้ภาพที่ได้เบลอ ขาดความคมชัด

การ Transform ที่นิยมใช้ในการลดข้อมูลภาพมีอยู่ด้วยกันหลายวิธี เช่น Fast Fourier Transform, Fast Walsh - Hadamard Transform, Fast Slant Transform, Fast

Discrete Cosine Transform, Fast Discrete Sine Transform เป็นต้น ซึ่งแต่ละวิธีก็มีข้อดีข้อเสียที่แตกต่างกันไป การ Transform ที่นิยมใช้กันมากได้แก่ Discrete Cosine Transform เพราะเป็นวิธีที่สามารถคำนวณได้ง่าย เนื่องจากมีการคำนวณเฉพาะค่าจริงไม่ใช่ค่าจินตภาพ (Imaginary)

วิธี Transform Coding ถึงแม้ว่าจะเป็นวิธีที่ยุ่งยากแต่ก็สามารถสร้างระบบได้ด้วยอุปกรณ์ทางฮาร์ดแวร์ (Hardware) ดิจิตอลความเร็วสูงได้ง่าย และเป็นระบบที่ยืดหยุ่น (adaptive system) สามารถที่จะลดขนาดของข้อมูลให้อยู่ในอัตราบิทเรทประมาณ 0.5 - 1 บิทต่อจุดภาพ ซึ่งเป็นช่วงอัตราการลดที่สามารถสร้างภาพเดิมกลับมาได้สมบูรณ์เพียงพอต่อการนำไปใช้งานต่าง ๆ

3.3 Hybrid Coding

Hybrid Coding เป็นเทคนิคที่นำข้อดีของวิธี Predictive Coding และ Transform Coding มาใช้ร่วมกัน ทั้งนี้เพราะบางครั้งการลดขนาดข้อมูลภาพด้วย Transform Coding ไม่อาจจะให้รายละเอียดของภาพเพียงพอ ดังนั้นการเพิ่มรายละเอียดของภาพจึงต้องใช้ความสัมพันธ์ของจุดภาพที่มีอยู่ทั้งแนวนอน และทางแนวตั้ง โดยการใช้การ Transform แบบมิติเดียวในทิศทางแกนใดแกนหนึ่งของภาพ เช่น ใช้การ Transform กับข้อมูลภาพในทางแนวนอน ต่อจากนั้นจึงใช้ Predictive Coding กับสัมประสิทธิ์ของการ Transform ที่ได้เพื่อที่จะใช้ในการคาดคะเนค่าของกลุ่มสัมประสิทธิ์ที่เหมือน ๆ กัน ที่ตามมาในแนวนอน ในกรณีนี้ความสัมพันธ์ของข้อมูลในทางแนวตั้งจะเป็นตัวกำหนดผลที่ได้ว่าถูกต้องมากน้อยเพียงใด แต่โดยเฉลี่ยแล้วค่าสัมประสิทธิ์ที่ตามมักจะมีค่าที่เหมือน ๆ กัน ซึ่งเหมือนกับการทำ Predictive ใน data domain นั้นเอง และในกรณีของการ Transform ของภาพบริเวณเดียวกันแต่เป็นเฟรมที่แตกต่างกันตามมา การใช้ Hybrid Coding ในลักษณะนี้สามารถเป็นไปได้ อีกลักษณะหนึ่งคือ ใช้การ Predictive ในขั้นตอนแรกก่อนแล้วจึงทำการ Transform แต่ในกรณีนี้ ต้องการระบบที่ซับซ้อนมากกว่า

การลดข้อมูลภาพด้วยวิธีของ Hybrid Coding นี้ อาจจะพูดได้ว่ามีคุณสมบัติอยู่ระหว่างวิธีของ Predictive Coding และ Transform Coding ดังนั้นอัตราบิทเรทต่ำสุดของวิธีนี้จึงไม่สามารถทำให้ต่ำกว่าวิธีของ Transform Coding เพียงแต่สามารถสร้างได้ง่ายกว่า โดย อัตราบิทเรทของระบบ จะมีค่าประมาณ 1 บิท ต่อจุดภาพ ซึ่งเป็นอัตราบิทเรทที่สามารถสร้างภาพเดิมกลับมาใหม่ (reconstruction) ได้ภาพที่มีคุณภาพดีพอสมควร

3.4 การลดข้อมูลภาพด้วยวิธีรันเลงจ์ (Runlength Compression)

การลดข้อมูลด้วยวิธีนี้ อาศัยลักษณะทั่วไปของข้อมูลภาพที่จะต้องมีส่วนของฉากหลัง (Background) และ พื้นหน้า (foreground) ในส่วนของ ฉากหลังจะมีรายละเอียดของภาพไม่มากนัก ส่วนนี้เองจะมีการเปลี่ยนแปลงของข้อมูลน้อย เมื่อเทียบกับส่วนของ foreground ซึ่งมีรายละเอียด และการเปลี่ยนแปลงของข้อมูลมาก ในส่วนที่มีการเปลี่ยนแปลงข้อมูลน้อยนี่เอง ที่เราสามารถนำการเข้ารหัสแบบ Runlength มาประยุกต์ใช้ได้อย่างมีประสิทธิภาพ

การเข้ารหัสแบบนี้ จะจัดข้อมูลภาพเดิม ให้อยู่ในรูปของคู่ลำดับ (G_i, L_i) โดย G_i แทนระดับความเข้ม หรือ Graylevel L_i แทนความยาวของข้อมูลหรือ จำนวนจุดที่มีระดับ Graylevel G_i การเข้ารหัสแบบนี้มีด้วยกัน 2 วิธีใหญ่คือ

<u>วิธีที่</u> 1 จะทำการอ่านข้อมูลเข้ามา โดยการนับจำนวนข้อมูลที่ซ้ำกัน กับข้อมูลนั้นเข้ามาด้วย แล้วแปลงข้อมูลไปเป็น 2 Byte คือ ในไบท์แรก จะเก็บจำนวนตัวข้อมูลที่ซ้ำกัน โดยจะมีไบท์ที่สอง เก็บค่าข้อระดับสีที่ซ้ำกันนั้นเอาไว้ โดยใน 1 ชุดข้อมูลเอาท์พุท (2 Byte) จะนับจำนวน จุดที่ซ้ำกันได้ 256 จุดสี ตัวอย่างของการเข้ารหัสข้อมูล เช่น

ข้อมูลเป็น (ค่าที่แสดงเป็นเลขฐาน 16)

AC AC AC AC 15 15 15 15 15 15 45 45 78 20 10 10 10 10

เข้ารหัสได้เป็น

04 AC 06 15 02 45 01 78 01 20 04 10

จะเห็นได้ว่าเราสามารถ ลดข้อมูลจาก 19 Byte เหลือ 12 Byte ซึ่งหากข้อมูลซ้ำกันถึง 256 จุดแล้วจะทำให้ เราลดข้อมูลลงไปได้อย่างมหาศาล แต่ในขณะเดียวกันหากเกิดกรณีที่แย่ที่สุดของการใช้วิธีนี้ก็คือ กรณีที่ข้อมูลแต่ละตัวไม่ซ้ำกับจุดข้างเคียงเลย ซึ่งจะทำให้ผลของการเข้ารหัส จะได้รหัสที่ยาวเป็น 2 เท่าของข้อมูลอินพุท

<u>วิธีที่</u> 2 จากความบกพร่องของวิธีการทำ Run Length Encoding วิธีแรก ตรงที่โปรแกรมจะทำการเข้ารหัสข้อมูล เมื่อนับจำนวนข้อมูลได้ตั้งแต่ 1 - 255 จุด ดังนั้นถ้าเกิดข้อมูลของเรามี จุดสีที่อยู่โดด ๆ (ไม่ซ้ำกับจุดข้างเคียง) อยู่มากมายจะทำให้การเข้ารหัสไม่ทำให้ข้อมูลเล็กลงมากนัก หรืออาจจะใหญ่กว่า Origin ด้วยซ้ำไป และ Pixel ที่ซ้ำกันที่เดียวถึง 255 ตัว คงไม่เกิดขึ้นบ่อยนัก จึงได้มีแนวความคิดที่จะทำการแก้ไขปัญหาของการทำ Runlength วิธีแรก เกี่ยวกับการมีข้อมูลที่ไม่ซ้ำกันกับตัวข้างเคียง โดยมีหลักการอยู่ 2 ข้อ ดังนี้

1. จะไม่ทำการลดข้อมูลกับส่วนที่มีจำนวนซ้ำกันน้อยกว่า 3 ตัว
2. ให้มีการทำเครื่องหมายเพื่อที่จะแยกส่วนที่มีการลด กับส่วนที่ไม่มีกาลดข้อมูลออกจากกัน โดยใช้บิท ๆ หนึ่ง เป็นตัวบอกว่าถูกลดขนาดหรือไม่

จากหลักการดังกล่าวนี้ จะใช้บิทที่ 7 ของไบท์บอกขนาดเป็นตัวบอกว่าข้อมูลที่ต่อจากนี้ไปมีการลดขนาดข้อมูลหรือไม่ ส่วนบิทที่เหลือเราก็ยังคงใช้บอกขนาดต่อไป เมื่อเรานับ Pixel ได้ซ้ำกันตั้งแต่ 3 ตัวขึ้นไป บิทที่ 7 ของ ไบท์บอกขนาดจะถูกแทน ให้เป็น 1 หรือ และเราจะเริ่มนับ 1 ตั้งแต่ Pixel ที่ซ้ำกันตั้งแต่ตัวที่ 4 เป็นต้นไป ทำให้บิทบอกขนาดมีค่าได้ตั้งแต่ 3 - 130 หรือใน 2 Byte นี้เราอาจเก็บข้อมูลได้ถึง 130 Byte ส่วนในกรณีที่ไม่มีการลดขนาดบิทที่ 7 ของ Byte บอกขนาดก็จะถูก set ให้เป็น 0 และ 7 bit ที่เหลือจะบอกจำนวนของข้อมูลที่มีค่าไม่ซ้ำกันนั้น โดยค่าที่ตามมาจะเป็นข้อมูลที่ไม่ถูกลดขนาด เป็นจำนวน Byte เท่ากับจำนวนที่แสดงไว้ใน ไบท์บอกขนาด ซึ่งจะมีค่าระหว่าง 0 ถึง 127 โดยจำนวนจริงจะมีค่าเท่ากับ ไบท์บอกขนาดบวกหนึ่งตัวอย่างเช่น

ข้อมูลเป็น (ค่าที่แสดงเป็นเลขฐาน 16)

AC AC AC AC 15 15 15 15 15 15 45 45 78 20 10 10 10 10

เข้ารหัสได้เป็น

81 AC 83 15 03 45 45 78 20 81 10

ความหมาย : 81 (ถูกลดขนาดลง 4 ไบท์มีค่า) AC

: 83 (ถูกลดขนาดลง 6 ไบท์มีค่า) 15

: 03 (ไม่ถูกลดขนาดลง 4 ไบท์มีค่า) 45 45 78 20

: 81 (ถูกลดขนาดลง 4 ไบท์มีค่า) 10

โดยวิธีนี้ผลลัพธ์ที่ได้จะแย่ที่สุดในกรณีที่เกิดมีข้อมูลที่ไม่ซ้ำกันเลยมากกว่า 128 ไบท์ เมื่อทำการเข้า รหัสจะได้รหัสถึง 129 ไบท์ สำหรับทุก ๆ 128 ไบท์ ของข้อมูลอินพุท และในกรณีที่ดีที่สุดก็คือ เมื่อข้อมูลซ้ำกันถึง 128 ไบท์ เมื่อเข้ารหัสแล้วจะได้ข้อมูลที่ลดลงเหลือเพียง 2 ไบท์

ความสามารถในการลดข้อมูลภาพด้วย Runlength

ในกรณีที่ข้อมูลภาพ มีความแตกต่างของระดับเทามาก อัตราของข้อมูลต่อ 1 จุดภาพ (Data Bite Rate) จะมากตามไปด้วย เนื่องจากมีความเป็นไปได้ที่ข้อมูลจะมีการเปลี่ยนแปลงสูง

การลดข้อมูลด้วย Runlength นี้จะไม่มี ความคลาดเคลื่อนเกิดขึ้น หลังจากการถอดรหัส การใช้วิธีนี้ต้องพิจารณาถึง ลักษณะของภาพต้นแบบว่ามีความเหมาะสมหรือไม่

ควรมีรายละเอียดของภาพน้อย เช่น ภาพเอกสาร ภาพทางด้านกราฟจากตาราง เราสามารถลดข้อมูลภาพได้ดียิ่งขึ้นได้ โดยการปรับข้อมูลให้มีความแตกต่างของระดับเทาให้น้อยลงด้วยวิธีของ Gray scale Transformation

3.5 Discrete Cosine Transform Coding

Discrete cosine transform coding เป็นวิธีการที่ใช้ในการลดข้อมูลภาพที่ได้รับความนิยมอย่างแพร่หลาย ทั้งนี้เพราะค่าสัมประสิทธิ์ในโดเมนของความถี่ ที่ได้จะเป็นเทอมของค่าจริง (real time) เท่านั้น อีกทั้งยังสามารถคำนวณได้แบบรวดเร็ว (fast algorithms) และยังสามารถใช้งานจริงในลักษณะ real time โดยใช้ฮาร์ดแวร์ได้ไม่ยาก ในปัจจุบันหลักการของ Discrete cosine transform coding ยังคงมีการวิจัยกันอยู่ต่อไป เพื่อให้สามารถลดขนาดของข้อมูลให้ได้มากที่สุด พร้อมทั้ง หาวิธีที่จะเพิ่มความเร็วในการคำนวณให้ได้เร็วขึ้นไปอีก

ดังนั้นในที่นี้จึงยกตัวอย่าง ระบบการลดข้อมูลภาพโดยใช้เทคนิคของ Discrete cosine transform coding โดยมีโครงสร้างของระบบเป็นดังรูปที่ 3.1 จาก block diagram ของการ transform coder ภาพอินพุทที่เข้ามาจะถูกแยกออกเป็นบล็อกเล็ก ๆ โดยเราสามารถกำหนดขนาดของบล็อคได้ว่าให้เป็นเท่าไร ขนาดของบล็อคที่เหมาะสมจะเป็นตัวเพิ่มประสิทธิภาพในการรวมพลังงานของการ Transform ซึ่งจะทำให้ภาพที่ได้มีรายละเอียดที่ดี ความเหมาะสมของขนาดของบล็อคค่าหนึ่ง ๆ จะเหมาะสมที่ค่าบิทเรทค่าหนึ่ง ๆ แต่อย่างไรก็ตามขนาดของบล็อคที่เหมาะสม สามารถคำนวณได้ด้วยคณิตศาสตร์ที่ยุ่งยากพอ ๆ กับการ Transform เลยทีเดียว โดยส่วนมากแล้วขนาดของบล็อค จะมีค่าเป็นเลขยกกำลังของสองอย่าง เช่น 4 , 8, 16 ซึ่งได้มาจาก 2^2, 2^3, 2^4 เป็นต้น หลังจากการแยกข้อมูลออกเป็นบล็อคย่อย ๆ แล้ว จะทำการ Transform ไปโดยอิสระของแต่ละบล็อค โดยมีสมการของการ Transform มิติเดียว และสองมิติ ดังต่อไปนี้เป็น

$$F(k) = \sqrt{(2/N)}\ \alpha(k) \sum_{n=0}^{N-1} f(n)\ \cos\{(2n+1)\pi k/2n$$

เมื่อ $k = 0, 1, \ldots\ldots, N-1$ .....(3.7)

เมื่อ

$\alpha(0) = \sqrt{(1/2)}$ และ $\alpha(k) = 1$ เมื่อ k ไม่เท่ากับ 0 .....(3.8)

F(k) เป็นผลที่ได้จากการ Transform และ f(n) เป็นข้อมูลอินพุทตัวที่ n

ส่วนสมการของการ Transform แบบสองมิติสามารถเขียนได้ดังนี้ คือ

$$F(u,v) = \frac{2}{N}\,\alpha(u)\,\alpha(v)\sum_{m=0}^{N-1}\sum_{m=0}^{N-1} f(m,n)\,\cos\left\{\frac{(2m+1)u\,\pi}{2N}\right\}\cos\left\{\frac{(2n+1)v\,\pi}{2N}\right\}$$

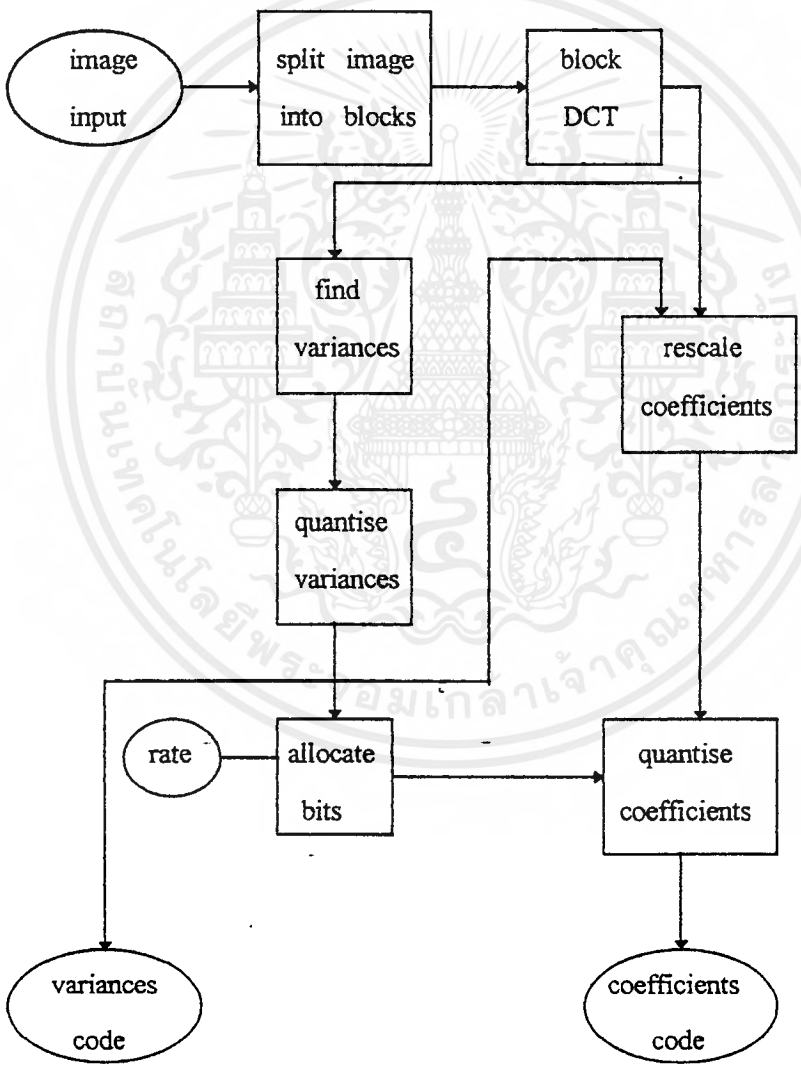
เมื่อ

u,v เป็นตัวแปรของ discrete frequency มีค่าเป็น 0, 1, 2,.....N - 1

f (m,n) เป็นตำแหน่งของจุดภาพภายในบล็อคขนาด N*N (0, 1, 2, ...N - 1)

F (u,v) เป็นผลจากการทำ discrete cosine Transform

$\alpha(0) = \sqrt{(1/2)}$ และ $\alpha(j) = 1$ เมื่อ j ไม่เท่ากับ 0



รูปที่ 3.1 แผนภาพแสดงตอน Forward Transform ของ DCT

หลังจากการผ่านการ Transform โดยใช้ Discrete Cosine Transform เพื่อเปลี่ยนข้อมูลให้อยู่ในรูปของ frequency domain พลังงานรวมส่วนใหญ่ของข้อมูลจะถูกเก็บ Z(pack) อยู่ในสัมประสิทธิ์เพียงไม่กี่ตัวของการ Transform จากสัมประสิทธิ์ที่ได้จะถูกนำมาหาค่า variances ของแต่ละบล็อค เพื่อที่จะนำค่า Standard deviation ไปคำนวณหา bit allocate สำหรับการจัดให้กับสัมประสิทธิ์แต่ละตัวในบล็อค และนอกจากนั้นยังส่งค่า Standard deviation ไปยังทางด้านรับรับด้วย เพื่อใช้ในตอน decode ดังนั้น จึงต้องทำการจัดระดับ (quantise) ค่านี้ก่อนที่จะเก็บหรือส่งไปยังด้านรับ และค่าสัมประสิทธิ์ตัวอื่น ๆ ก็จะถูกปรับระดับ (normalize) ด้วยค่าของ variances ที่ได้หลังจากการกำหนด bit allocate เพื่อปรับค่าของสัมประสิทธิ์ให้อยู่ในช่วงของค่าบิทเรทที่กำหนด ก่อนที่จะทำการ quantise ส่วนการกำหนดค่า bit allocate ของสัมประสิทธิ์ที่ตำแหน่ง (u,v) หลังการ transform ในแต่ละบล็อค คือ

$$b_{uv} = R + 0.5 \log_2 \{ \sigma_{uv}^2 / (\prod_{k=0}^{n-1} \prod_{k=0}^{n-1} \sigma_{kl}^2) \ 1/N \}$$

$$= \log_2 (\sigma_{uv}) + R - (1/N) \log_2 \{ \prod \prod \sigma_{kl} \}$$

$$= \log_2 (\sigma_{uv}) + \beta \qquad \text{.....(3.10)}$$

เมื่อค่า σ_{uv} คือค่า standard deviation ของสัมประสิทธิ์ ของการ Transform ที่ความถี่ (u,v) และ σ_{kl} เป็นค่า standard deviation ของสัมประสิทธิ์ ของการ Transform ที่ตำแหน่ง kl โดย k และ l มีค่าเป็น 0 ถึง n - 1 เมื่อ n คือขนาดของ บล็อค และ N คือจำนวนของสัมประสิทธิ์ในแต่ละบล็อค โดยที่ค่าคงที่ β กำหนดได้จาก

$$\beta = R - (1/N) \log_2 [\prod_{k=0}^{n-1} \prod_{k=0}^{n-1} \sigma_{kl}]$$

และค่าสูงสุดของบิทที่จะกำหนดให้กับ สัมประสิทธิ์แต่ละตัวจะกำหนดให้อยู่ภาพใต้

ค่า $b_{max.}$ $0 < b_{uv} < b_{max.}$ และค่า b_{vu} เป็นจำนวนเต็ม

โดยที่ $\sum_{u}\sum_{v} b_{uv} = B$ และ $B = R$ เมื่อ R คือ ค่าบิทเรทที่ต้องการ

หลังจากการคำนวณ bit allocation ค่าสัมประสิทธิ์ของการ Transform แต่ละตัวในบล็อคจะถูกปรับค่า(normalize) ให้อยู่ในช่วงขนาดของค่าไม่เกินจำนวนบิทที่กำหนด แล้วจึงทำการ quantise ค่าของสัมประสิทธิ์ แต่ละตัว โดยอาศัยคุณสมบัติการกระจายพลังงานของสัมประสิทธิ์ ของการ Transform ที่กล่าวไว้ว่า พลังงานส่วนใหญ่จะอยู่ในช่วงที่ความถี่ต่ำ ๆ หรือ อาจนำรูปแบบการกระจายความหนาแน่นแบบ Garssian หรือ Laplacian มาใช้ก็ได้ ดังนั้น สัมประสิทธิ์ที่อยู่บนมุมซ้ายบน เป็นกลุ่มของความถี่ต่ำ ซึ่งถือว่าสำคัญจะถูกกำหนดด้วยจำนวนบิทสูงในขณะที่กลุ่มสัมประสิทธิ์ที่อยู่มุมขวาล่าง เป็นกลุ่มของความถี่สูง ซึ่งถือว่าไม่สำคัญจะถูกกำหนดด้วยจำนวนต่ำ ๆ

8	8	8	7	7	7	5	5	4	4	4	4	4	4	4	4
8	8	7	6	5	5	3	3	3	3	3	2	2	2	2	2
8	7	6	4	4	4	3	3	2	2	2	2	2	2	2	2
7	6	4	3	2	2	2	2	1	1	1	1	0	0	0	0
7	5	4	2	2	2	2	1	1	1	0	0	0	0	0	0
7	5	4	2	2	2	1	1	0	0	0	0	0	0	0	0
5	5	3	2	2	1	1	0	0	0	0	0	0	0	0	0
5	3	3	2	1	1	0	0	0	0	0	0	0	0	0	0
4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0
4	3	2	1	1	0	0	0	0	0	0	0	0	0	0	0
4	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0
4	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0

รูปที่ 3.2 ตัวอย่างการกำหนบิทให้กับสัมประสิทธิ์ในโดเมน ความถี่หลังการแปลงด้วย DCT กับบล็อกขนาด 16*16 ซึ่งสามารถลดข้อมูลลงได้ จาก 8 บิทต่อจุดภาพ ให้ เหลือ 1.5 บิทต่อจุดภาพ

สำหรับขั้นตอนที่จะนำภาพกลับคืนมาหลังจากที่ได้เข้ารหัสไว้แล้ว สิ่งที่ต้องนำมาใช้คือ ค่าของสัมประสิทธิ์ และค่าของ Variances ของการ Transform โดยค่าของ Variances ถูกมาคำนวณค่า bit allocation จากค่าบิทเรทที่กำหนด แล้วจึงนำค่าที่ได้นี้ไปใช้ในการคำนวณปรับค่าของสัมประสิทธิ์ ที่ได้จากการเข้ารหัสในตอนแรก เพื่อให้ได้ค่าสัมประสิทธิ์เดิมกลับมา (rescale coefficients) เพื่อจะได้ทำ Inverse Transform ต่อไป โดยมีสมการของการ Inverse Transform เป็นดังนี้

กรณี แปลงกลับแบบ 1 มิติ (One dimensional Inverse Transform)

$$f(n) = \sqrt{(2/N)} \sum_{k=0}^{N-1} \alpha(k)\, F(k)\, \cos\left[(2n+1)\,\pi k / 2n\right]$$

เมื่อ $k = 0, 1, \ldots, N-1$ .....(3.11)

เมื่อ f(n) เป็นผลที่ได้จากการทำ Inverse Transform กับ F(k) ซึ่งสัมประสิทธิ์จากการ Transform

กรณี แปลงกลับแบบ 2 มิติ (Two dimensional Inverse Transform)

$$f(m,n) = \frac{2}{N} \alpha(u)\, \alpha(v) \sum_{u=0}^{N-1} \sum_{u=0}^{N-1} F(u,v)\, \cos\left[\frac{(2m+1)\,u\,\pi}{2N}\right] \cos\left[\frac{(2n+1)\,v\,\pi}{2N}\right]$$

.....(3.12)

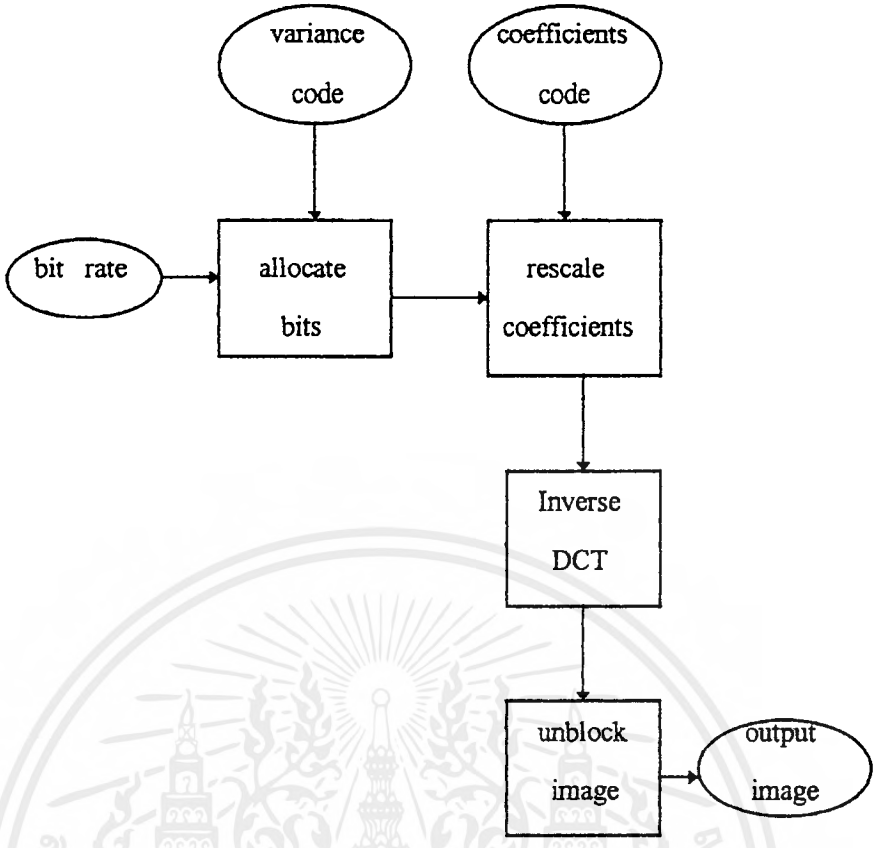
เมื่อ

$\alpha(0) = \sqrt{(1/2)}$ และ $\alpha(k) = 1$ เมื่อ k ไม่เท่ากับ 0

m,n เป็นตำแหน่งของจุดภาพ มีค่าตั้งแต่เป็น 0, 1, 2, ...N - 1

F(u,v) เป็นค่าสัมประสิทธิ์ที่ได้จากการ Transform ภาพขนาด N*N

f(m,n) เป็นผลลัพธ์ของการทำ Inverse Transform



รูปที่ 3.3 แผนภาพแสดงตอน DCT decoding

ถึงแม้ว่า การลดข้อมูลภาพโดยใช้หลักการของการ Transform จะเป็นที่ยอมรับและได้รับความนิยมแพร่หลายมาจะทุกวันนี้ เพราะสามารถสร้างในลักษณะของ real time ได้ด้วย แต่ค่าของบิทเรทที่ใช้จะอยู่ในช่วง 1 ถึง 0.5 เพราะต่ำกว่านี้ คุณภาพของภาพที่ได้จะแย่มาก คือเป็นภาพที่ขาดรายละเอียด ส่วนของขอบภาพจะเบลอไม่มีความคมชัด ทั้งนี้เนื่องจากการแปลง data domain ไปอยู่ในรูปของ frequency domain นั้น ที่ช่วงความถี่สูง ๆ ซึ่งเป็นส่วนบริเวณขอบต่าง ๆ ในภาพมักจะถูกตัดทิ้งไปเพื่อให้ขนาดของข้อมูลลดลง จึงทำให้ขอบต่าง ๆ ในภาพไม่คมชัด และภาพที่ได้ยังมีลักษณะเป็นบล็อค ๆ ตามขนาดของบล็อกที่ถูกแบ่งในตอน Transform

3.6 เกณฑ์การวัดความเหมือนจริงของภาพ (Image Fidelity)

ในการลดข้อมูลภาพนั้น จะมีส่วนหนึ่งที่เกิดผิดพลาดหรือ สูญเสียไป ข้อผิดพลาดที่เกิดขึ้นนี้จะมีผลตอนที่สร้างภาพกลับคืนมา (reconstruction) และค่าความผิดพลาดนี้จะอยู่ในช่วงหนึ่งที่สามารถยอมรับได้ ดังนั้นเกณฑ์การวัดความเหมือนจริงของภาพสามารถนำมาใช้ในการวัดประสิทธิภาพของระบบได้ ตัวอย่างเกณฑ์ที่นิยมใช้ในการวัดคุณภาพของภาพคือ ค่า root -

mean - square (rms.) ของ error ระหว่างข้อมูลภาพอินพุท และข้อมูลภาพเอาท์พุท (Signal - to - Noise Ratio) เมื่อกำหนดให้ข้อมูลภาพอินพุทประกอบด้วยอาเรย์ขนาด N * N ของจุดภาพ f (x,y,) โดยที่ x, y มีค่าเป็น 0, 1, ... N - 1 แต่ละจุดภาพมีค่าของระดับสีเทาที่เป็นไปได้ คือ 2^m เมื่อ m เป็นจำนวนบิทของเลขฐานสอง

สำหรับทุกค่าของ x และ y ในช่วง 0, 1, ... N - 1 ค่า error ระหว่างจุดภาพอินพุท และเอาท์พุท คือ

$$e(x,y) = g(x,y) - f(x,y) \qquad \text{.....(3.1)}$$

เมื่อ f(x,y) คือ input image ณ. จุด x,y ใด ๆ

เมื่อ g(x,y) คือ output image ณ. จุด x,y ใด ๆ

ค่าเฉลี่ยของ error กำลังสองของภาพ (mean square error) คือ

$$e2 = (1/N^2) \sum_{x=0}^{N-1} \sum_{x=0}^{N-1} (e(x,y))^2$$

$$= (1/N^2) \sum_{x=0}^{N-1} \sum_{x=0}^{N-1} \{g(x,y) - f(x,y)\}^2 \qquad \text{.....(3.2)}$$

ดังนั้น rms. ของ error จึงสามารถเขียนได้ดังนี้ คือ

$$e_{rms.} = (e^2)^{1/2} \qquad \text{.....(3.3)}$$

ค่า root mean square error เป็นค่าที่ใช้ในการวัดความแตกต่างของข้อมูลอินพุทกับข้อมูลเอาท์พุท แต่เมื่อพิจารณาขนาดของข้อมูลเอาท์พุท ต่อขนาดสัญญาณรบกวน (noise) ก็จะได้เป็นค่า Signal - to - Noise (SNR) เมื่อกำหนดได้สัญญาณภาพเอาท์พุทแต่ละจุดประกอบด้วยสัญญาณอินพุทบวกด้วยค่าสัญญาณรบกวน นั้นคือ

$$g(x,y) = f(x,y) + e(x,y) \qquad \text{.....(3.4)}$$

ดังนั้นค่า mean square Signal - to - Noise ของข้อมูลภาพเอาท์พุท สามารถหาได้โดยค่าเฉลี่ยของสัญญาณอินพุทกำลังสอง $f^2(x,y)$ หารด้วยค่าเฉลี่ยของสัญญาณรบกวนกำลังสอง $e^2(x,y)$ ของข้อมูลภาพทั้งหมด ซึ่งสามารถเขียนได้ดังนี้

$$(SNR) = \frac{(1/N^2)\sum_{x=0}^{N-1}\sum_{x=0}^{N-1} f^2(x,y)}{(1/N^2)\sum_{x=0}^{N-1}\sum_{x=0}^{N-1} e^2(x,y)} \qquad \text{.....(3.5)}$$

ค่า rms. ของ SNR จึงสามารถเขียนได้ดังต่อไปนี้

$$(SNR)_{rms.} = \{ (1/N^2)\sum_{x=0}^{N-1}\sum_{x=0}^{N-1} g^2(x,y) \; / \; (1/N^2)\sum_{x=0}^{N-1}\sum_{x=0}^{N-1} e^2(x,y) \}1/2$$

.....(3.6)

โดยที่เทอมส่วนของสมการข้างบน เป็นสมการของสัญญาณรบกวนที่อยู่ในรูปของผลต่าง ระหว่างข้อมูลอินพุทกับเอาท์พุท

หรือ เราอาจคิดเป็นค่า $(SNR)_{rms}$ dB ได้ โดย

$$(SNR)_{rms}\,dB = 20 \log (SNR)_{rms} \qquad \text{.....(3.7)}$$

จากวิธีการที่ใช้ในการวัดความเหมือนจริงของภาพที่กล่าวมาข้างบนนี้ ไม่สามารถที่จะบ่งบอกหรือใช้เป็นเกณฑ์ในการพิจารณาได้แต่เพียงอย่างเดียว ในกรณีของภาพเอาท์พุทที่ได้จากการประมวลผล หรือรับส่งสัญญาณ โดยมีสายตาของมนุษย์เป็นตัวรับภาพจากจอภาพอีกทีหนึ่ง อย่างเช่น broadcast TV, picture phone, หรืองานต่าง ๆ ที่มีการใช้ Image processing เป็นต้น ระบบการมองเห็นของมนุษย์จะมีลักษณะคุณสมบัติพิเศษกล่าวคือ ระบบการมองเห็นของตาจะไวต่อความเข้มของแสง ในลักษณะของ logarithmic ดังนั้น error ในบริเวณที่เป็นที่มืดของภาพ จะเป็นได้ชัดกว่า error ที่อยู่ในบริเวณที่สว่าง และระบบการมองเห็นยังไวต่อการเปลี่ยนแปลงอย่างทันทีทันใด ของระดับความเทาด้วย error ที่อยู่บนขอบหรือใกล้ ๆ ขอบของภาพจะเป็นได้ชัดมากกว่า error ที่อยู่ในโครงสร้างที่เป็นฉากหลังของภาพ ด้วยเหตุนี้ถึงแม้ว่าภาพสองภาพจะมีค่าของ rms. error เท่ากัน แต่อาจจะปรากฏความแตกต่างของคุณภาพของการมองเห็นที่แตกต่างกันได้

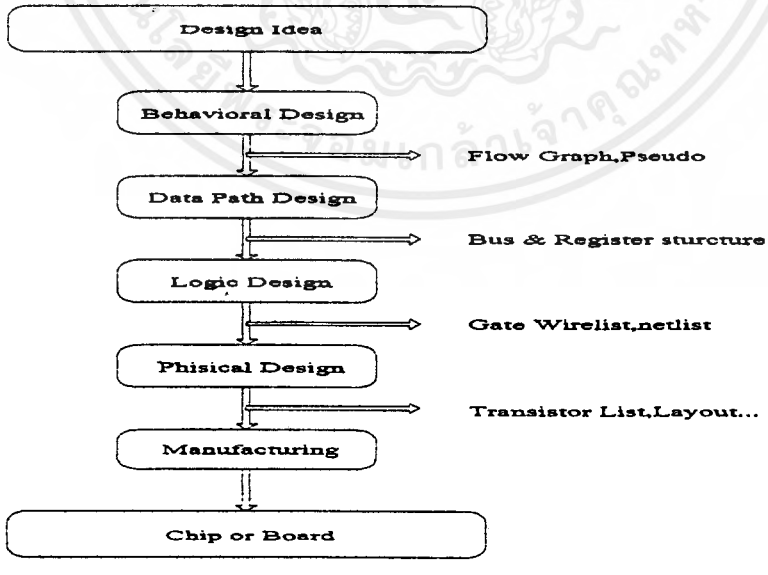
บทที่ 4

ภาษาVHDL

ในขณะที่ขนาดและความซับซ้อนของระบบดิจิตอลเพิ่มมากขึ้น คอมพิวเตอร์เพื่อช่วยในการออกแบบ (CAD: Computer Aided Design) ก็ได้ถูกนำเข้ามาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มมากขึ้นเช่นกัน อุปกรณ์และวิธีการออกแบบใหม่ๆได้ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบ และภาษาสำหรับบรรยายอุปกรณ์ฮาร์ดแวร์ (HDL: Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการในการออกแบบระบบดิจิตอล

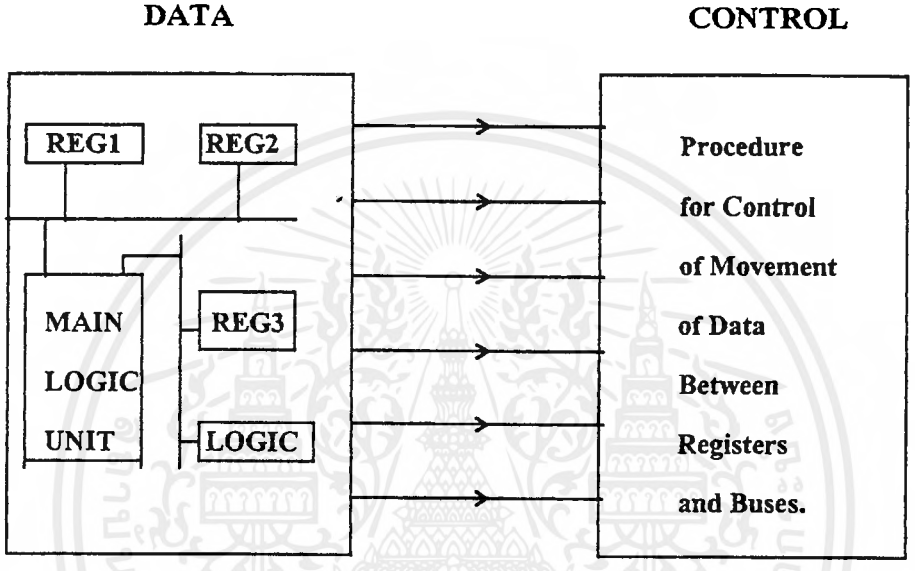
ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆมากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและก็เข้าสู่กระบวนการออกแบบในขั้นต่อไป

รูปที่ 4.1 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบเสียก่อนและทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นภายในขั้นตอนนี้จึงมีความจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจจะเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (pseudo code) ก็ได้



รูปที่ 4.1 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไป เป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์ (register) และวงจรตรรก (logic) ที่จำเป็นทั้งหมดเพื่อนำมาประกอบกันเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (unidirec - tional or bidirectional bus) กระบวนการในการควบคุมการเคลื่อนย้ายของข้อมูลระหว่างรีจิสเตอร์และวงจรตรรกก็จะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ ดังรูปที่ 4.2



รูปที่ 4.2 แสดงการออกแบบระบบเส้นทางของข้อมูล

การออกแบบวงจรตรรกจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้ เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิปฟลอป (flipflop) เป็นส่วนประกอบของอุปกรณ์ย่อยต่างๆ ได้แก่ รีจิสเตอร์ เก็บข้อมูล บัสวงจรตรรก และส่วนควบคุมฮาร์ดแวร์ ซึ่งในขั้นสุดท้ายจะได้ออกมาเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง

ต่อมา เป็นขั้นตอนของการเปลี่ยนเครือข่ายการโยงใยในขั้นตอนที่แล้ว ให้เป็นทรานซิสเตอร์และเลย์เอาท์ (transistor list and layout) ในขั้นตอนนี้จะเกี่ยวข้องกับการจัดวางเกทและฟลิปฟลอปแทนด้วยทรานซิสเตอร์ หรือไลบรารีเซล

ขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

4.1 แนะนำ VHDL (Introduction to VHDL)

ในช่วงฤดูร้อนของปี 1981 สถาบันเพื่อการป้องกัน (The institute for Defence Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่งเพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวข้องกับวงจรรวมที่มีความเร็วสูงมาก (Very High Speed Integrated Circuit) ต่อมาในปี 1985, IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านของการกำหนดพฤติกรรมการทำงานของวงจร ทำให้นักออกแบบสามารถกำหนด รูปแบบพฤติกรรมการทำงานได้ทั้งของ วงจรดิจิตอลทั่วไปและในระบบที่แตกต่างออกไป เช่น พฤติกรรมการทำงานของระบบเรดาร์ หรือพฤติกรรมการทำงาน ของระบบเครือข่ายใยประสาทในสมองมนุษย์ได้ ข้อดีหลักที่สำคัญของ VHDL ก็คือ ภาษานี้สามารถที่จะถูกใช้ได้ตลอดในทุกๆระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง (system level) จนถึงในระดับที่ต่ำกว่า (lower hardware levels) สามารถใช้ภาษาเดียวกันได้โดยตลอด ทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

4.1.1 ข้อกำหนด (VHDL Requirements)

ในเอกสารของ DoD (Department of Defense Requirements for Hardware Description Languages) ซึ่งออกมาในเดือนมกราคม ปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

4.1.1.1 ลักษณะทั่วไป (General Features)

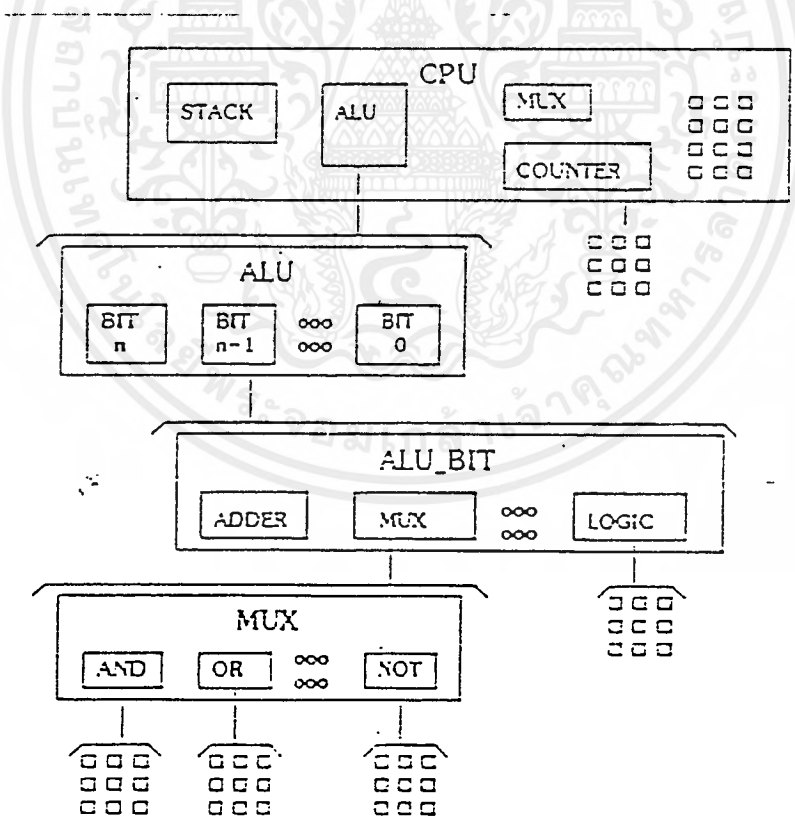
เอกสารของ DoD กำหนดไว้ว่า VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการเลียนแบบ (simulation) การสังเคราะห์ (synthesis) และการทดสอบ (testing)

นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับบนคือระบบจนถึงระดับเกทอีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลจริงๆ ทุกๆองค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมๆกัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือว่าเป็นข้อกำหนดที่สำคัญอย่างหนึ่งใน VHDL ด้วยเช่นกัน (สำหรับภาษาที่ใช้ในการบรรยาย

ฮาร์ดแวร์แล้ว ความพร้อมเพรียงจะหมายถึง ทุกๆคำสั่ง องค์ประกอบ เกท หรือวงจรตรรกจะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อมๆกัน)

4.1.1.2 สนับสนุนการออกแบบแบบลำดับขั้น (Support for Design Hierachy)

การออกแบบแบบลำดับขั้นก็เป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลายๆระดับ ในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตัวเอง หรือถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ ดังรูป 4.3



รูปที่ 4.3 แสดงตัวอย่างการออกแบบแบบลำดับขั้น

5.1.1.3 ไลบรารี (Library Support)

VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารีหรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆสามารถนำไปใช้ได้ด้วย

5.1.1.4 ลำดับคำสั่ง (Sequential Statement)

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if-then-else และ loop ทั่วๆไปได้

การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

4.1.1.5 การกำหนดคุณสมบัติ (Generic Design)

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่นๆก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้นๆ ภาษาสำหรับการออกแบบที่ดีควรจะสามารถให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วยเช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่นๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

4.1.1.6 ชนิดของข้อมูล (Type Declaration and Usage)

VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด BIT และ BOOLEAN เท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับ (enumerate type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเองก็ได้

4.1.1.7 โปรแกรมย่อย (Use of Subprograms)

ความสามารถในการใช้ฟังก์ชั่นและโพรซีเจอร์(procedure) ก็เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก การกำหนดตัวกระทำต่างๆทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

5.1.1.8 การควบคุมเวลา (Timing Control)

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ (event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

4.1.1.9 การกำหนดแบบโครงสร้าง (Structural Specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุกๆระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวมที่เกิดจากองค์ประกอบย่อยๆที่แตกต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

4.1.2 องค์ประกอบพื้นฐานใน VHDL (Basic Concepts in VHDL)

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (architecture) ดังแสดงในรูปที่ 4 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ ENTITY ตามด้วยชื่อขององค์ประกอบและคำ IS ภายในบรรยายถึงพอร์ตการติดต่อ อินพุท-เอาท์พุทพอร์ตขององค์ประกอบ ส่วนลักษณะภายนอกอื่นๆเช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท-เอาท์พุทและพารามิเตอร์อื่นๆที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูป การบรรยายหน้าที่ขององค์ประกอบเริ่มต้นหลังจากคำว่า BEGIN เป็นต้นไป

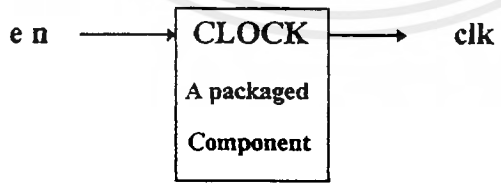
```
ENTITY component_name IS
        input and output ports,
    physical and other parameters,
END component_name;
```

```
ARCHITECTURE  identifier OF component_name  IS
  declarations.
BEGIN
    specification of the functionality of the component
    in term of its input lines and as influenced
    by physical and other parameters,
END  identifier;
```

รูปที่ 4.4 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม

4.1.2.1 การกำหนดการเชื่อมต่อ (Interface Description)

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 5 แสดงบล็อคไดอะแกรมและการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ clock_computer ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็ก IN และ OUT กำหนดโหมดของสัญญานเป็นอินพุทหรือเอาท์พุท BIT แสดงชนิดของข้อมูล




```
ENTITY clock, component  IS
    PORT (en: IN BIT: ck: OUT BIT):
END clock_component;
```

รูป 5.5 แสดงแผนภาพและการบรรยายการเชื่อมต่อของ clock_component

5.1.2.2 การกำหนดรูปแบบการบรรยาย (Architectural Description)

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในเทอมของอินพุทหรือในรูปขององค์ประกอบอื่นๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock-component ในรูปที่ 3.6 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี en เป็นอินพุท และ ck เป็นเอาท์พุท PROCESS เป็นคำเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม ภายในโพรเซสกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น ' 0' ถ้าสัญญาณ en มีค่าเป็น '1' ค่าของ periodic จะถูกคอมพลีเมนต์(complement) และส่งค่าให้กับ ck ซึ่งเป็นสัญญาณเอาท์พุท คำสั่ง WAIT กำหนดให้สัญญาณมีคาบเป็นเวลา 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock-component IS
BEGIN
  PROCESS
    VARIABLE periodic : BIT : = '0' ;
  BEGIN
   IF en = '1' THEN
     periodic : = NOT periodic;
   END IF ;
   ck <= periodic;
   WAIT FOR 1 US;
  END PROCESS;
END behavioral;
```

รูปที่ 4.6 แสดงการบรรยายเชิงพฤติกรรมของ clocl-component

5.1.2.3 โปรแกรมย่อย (Subprograms)

การใช้ฟังก์ชั่นและโพรซีเจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่น ถ้าเราใช้ฟังก์ชั่นแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกจริงๆ ในขณะที่ถ้าเราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลา แล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 4.7 แสดงการใช้โพรซีเจอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิตเป็นค่าจำนวนเต็ม รูปที่ 4.8 แสดงการใช้ฟังก์ชั่น โดยกำหนดให้ X เป็นตัวแปรชนิดบิดแทนการกระทำในสมการบูลีน

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF BIT;
...
PROCEDURE byte_to_integer(ib : IN byte; oi : OUT INTEGER) IS
 VARIABLE   result : INTEGER := 0;
BEGIN
   FOR  i IN 0 to 7 LOOP
     IF  ib(i) = '1' THEN
           result := result + 2**i;
     END IF;
   END LOOP;
   oi  := result;
END byte_to_integer;
```

รูปที่ 4.7 แสดงการใช้โพรซีเจอร์

```
FUNCTION f ( a, b, c : BIT ) RETURN BIT IS
 VARIABLE x : BIT;
BEGIN
  x := ((NOT a)AND(NOT b)AND c);
  RETURN x ;
END f;
```

รูปที่ 4.8 แสดงการใช้ฟังก์ชั่น

5.1.2.4 โอเปอร์เรเตอร์ (VHDL Operators)

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟท์แวร์ทั่วไปดังรูปที่4.9

PREDEFINED OPERATORS

LOGICAL OPERATORS : NOT AND OR NAND NOR

OPERAND TYPE : BIT BOOLEAN

RESULT : BIT BOOLEAN

ELATIONAL OPERATOR : = /= < <= > >=

OPERAND TYPE : anytype

RESULT TYPE : Boolean

RITHMETIC OPERATOR : + - * / ** 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

รูปที่ 4.9 แสดงตัวกระทำใน VHDL

5.1.2.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

ในวงจรอิเล็คทรอนิค อุปกรณ์ทุกๆตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (always active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องเสมอในทุกๆเหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของสถาปัตยกรรมบรรยาย (architecture) จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โพรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายๆโพรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆโพรเซสก็จะทำงานไปพร้อมๆกันด้วย

4.1.2.6 สัญญาณและตัวแปร (Signals and Variables)

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น w <= a AFTER 12 NS หมายถึง กำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 NS.

ในทางตรงข้าม ตัวแปรมีลักษณะเป็นเสมือนตัวกลางซอฟท์แวร์ที่ใช้ในการส่งผ่านข้อมูล และไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ := ตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่ง เช่นใน ฟังก์ชั่น โพรซีเจอร์ และโพรเซส

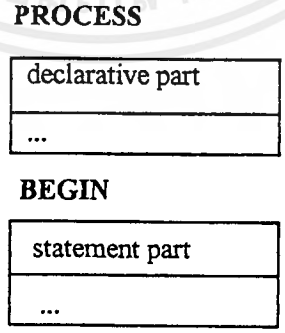
4.2 การบรรยายเชิงพฤติกรรม (Behavioral Description of Hardware)

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรมเป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามาโดยไม่คำนึงถึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในจะเป็นอย่างไร ในหัวข้อนี้จะแสดงให้เห็นถึงประโยชน์ในการใช้โปรแกรมย่อยที่จำลองรูปแบบอินพุทและเอาท์พุทในระดับพฤติกรรมแทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่างๆที่ควรรู้

4.2.1 โพรเซส (Process Statement)

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญญาณ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติคำสั่งพร้อมๆกันกับโพรเซสอื่นๆที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์การกำหนดค่าให้กับสัญญาณ (<=)

การบรรยายโพรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS รูปที่ 10 แสดงส่วนประกอบของการบรรยายแบบโพรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 4.10 แสดงรูปแบบของการบรรยายแบบโพรเซส

4.2.1.1 การกำหนดตัวกระทำภายในโพรเซส (Declarative Part of a Process)

ตัวกระทำภายในโพรเซสมี 3 ชนิด คือ ตัวแปร(variable) ไฟล์(file) และตัวคงที่(constant) ซึ่งตัวกระทำทั้งสามชนิดนี้หากมีการประกาศไว้ในโพรเซสใดก็จะใช้ได้เฉพาะในโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซสสามารถทำได้โดยสัญญาน (signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE

```
PROCESS
  FILE  flush : TEXT IS IN "filename.dat"
  VARIABLE  var : BIT;
  CONSTANT n : INTEGER := 0;
BEGIN
  ....
END PROCESS
```

รูปที่ 4.11 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส

รูปที่ 4.11 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN ค่าเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียวเท่านั้น แต่ค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้โปรแกรมย่อยนั้นๆ

4.2.1.2 การกำหนดการกระทำภายในโพรเซส (Statement Part of a process)

การกระทำใดๆภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับ (sequencial) เสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการทำซ้ำได้เช่น (F-THEN-ELSE, CASE-WHEN, FOR LOOP และ WHILE LOOP ดังตัวอย่างในรูปที่ 4.12 และ 4.13

```
ARCHITECTURE  demo OF partial_process IS
 ....
BEGIN
   PROCESS
      ....
   BEGIN
      ...
          x <= '1';
     IF   x = '1'  THEN
          perform action_1
     ELSE
          perform action_2
     END IF ;
      ...
   END PROCESS ;
END demo
```

รูปที่ 4.12 แสดงเงื่อนไขการกระทำในโพรเซส

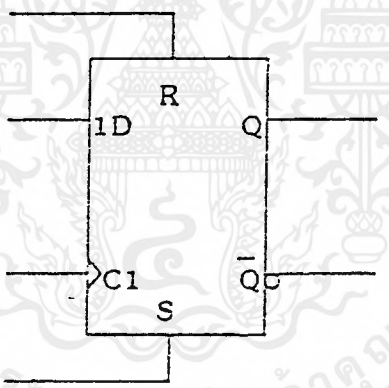
```
ARCHITECTURE  demo OF partial_process IS
 ....
BEGIN
   PROCESS
   BEGIN
      ...
          x <= a  AFTER 10 ns;
          y <= b  AFTER 6 ns;
      ...
   END PROCESS;
END  demo;
```

รูปที่ 4.13 แสดงการกระทำในโพรเซส

4.2.1.3 การกระตุ้นและยับยั้งการกระทำของโพรเซส (Sensitivity List)

การกระทำภายในโพรเซสจะเตรียมพร้อมและมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโพรเซสได้ โดยการกำหนดรายการของสัญญาณที่ต้องการให้โพรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้น เหตุการณ์ใดๆ ที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโพรเซส รายการของสัญญาณนี้เรียกว่า Sensitivity List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS

รูปที่ 3.14 แสดง (a) ตัวอย่างโมเดลและ (b) ตัวอย่างการบรรยายการเชื่อมต่อของD-Flip Flop รูปที่ 3.15 แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop (a) การใช้ตัวกระทำภายนอกโพรเซส และ (b) การใช้ตัวกระทำภายในโพรเซส โดยมีรายการของสัญญาณ (rst, set, clik) เป็นตัวกระตุ้นการปฏิบัติงานภายในโพรเซส



(a)

```
ENTITY d_sr_flipflop IS
    GENERIC ( sq_delay,rq_delay,cq_delay : TIME := 6 NS);
    PORT ( d, set , rst , clk : IN BIT ;  q , qb : OUT BIT );
END d_sr_flipflop;
```

(b)

รูปที่ 5.14 (a) ตัวอย่างโมเดล D-FlipFlop
(b) การบรรยายการเชื่อมต่อของ D-FLIPFLOP

```
ARCHITECTURE behavioral OF d_sr_flipflop IS
    SIGNAL  state : BIT := '0' ;
BEGIN
  dff : PROCESS ( rst , set , clk );
      BEGIN
       IF  set = '1' THEN
            state <= '1' AFTER  sq_delay;
       ELSIF  rst = '1' THEN
            state <= '0'  AFTER  rq_delay;
       ELSIF  clk = '1'  AND clk' EVENT THEN
          state <= d  AFTER  cq_delay;
       END  IF;
      END  PROCESS  dff;
       q <= state;
       qb <=  NOT state;
END  behavioral ;
```

(a)

```
ARCHITECTURE average_delay_behavioral OF d_sr_flipflop IS
 BEGIN
  dff : PROCESS ( rst , set , clk );
       VARIABLE  state : BIT  := '0';
      BEGIN
       IF  set = '1' THEN
            state <= '1' ;
       ELSIF  rst = '1' THEN
            state <= '0' ;
       ELSIF  clk = '1'  AND clk' EVENT THEN
          state <= d  ;
       END  IF;
       q <=  state  AFTER ( sq_delay +  rq_delay + cq_delay)/3;
       qb <=  NOT  state  AFTER ( sq_delay +  rq_delay + cq_delay)/3;
     END  PROCESS  dff;
END behavioral ;
```

(b)

รูปที่ 4.15 แสดงการบรรยายเชิงพฤติกรรมของ D-FLIPFLOP

(a) การใช้ตัวกระทำภายนอกโพรเซส

(b) การใช้ตัวกระทำภายในโพรเซส

4.2.2 การตรวจสอบการกระทำ (Assertion Statement)

VDHL ได้จัดเตรียมรูปแบบคำสั่งเพื่อใช้ในการตรวจดูการกระทำต่างๆที่เกิดขึ้นภายในอุปกรณ์ไว้ดังนี้

```
ASSERT assertion_condition REPORT "reporting_message" SEVERITY severity_level:
```

คำสั่งนี้จะถูกปฏิบัติเมื่อเงื่อนไข assertion_condition มีค่าเป็น FALSE และจะแสดงข้อความ "reporting_message" ส่วนพารามิเตอร์ severity_level จะเป็นตัวกำหนดระดับความรุนแรงที่เกิดขึ้น ซึ่งแบ่งเป็น 4 ระดับ คือ NOTE, WARNING, ERROR และ FAILURE ระดับความรุนแรง ERROR หรือ FAILURE จะทำให้การเลียนแบบการทำงานหยุดทันทีหลังจากที่ได้แสดงข้อความ reporting_message แล้ว ส่วนระดับ NOTE หรือ WARNING จะแสดงข้อความ reporting_message

```
ARCHITECTURE behavioral OF d_sr_flipflop IS
...
BEGIN
 ASSERT
    ( NOT ( set = '1' AND  rst = '1'))
 REPORT
     " set and rst are both 1"
 SAVERITY NOTE;
...
END  behavioral;
```

รูปที่ 4.16 แสดงการใช้ ASSERT

แต่ยังคงเลียนแบบการทำงานต่อไป สมมติว่าเราต้องการตรวจสอบการทำงานของ D-FlipFlop ในตัวอย่างข้างต้นว่าในขณะที่กำลังทำงานได้เกิดสัญญาณในกรณีที่ไม่พึงปรารถนาขึ้นหรือไม่เช่น สัญญาณ set และ rst มีค่าเป็น '1' พร้อมกันก็สามารถตรวจสอบได้โดยแทรกคำสั่ง ASSERT เพื่อตรวจสอบสัญญาณในโพรเซสได้ดังรูปที่ 4.16

4.2.3 การหยุดรอ (Sequential Wait Statements)

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณมีอยู่ 4 รูปแบบดังนี้

```
WAIT FOR waiting_time;
WAIT ON waiting_sinsitivity_list;
WAIT UNTIL waiting_condition;
WAIT;
```

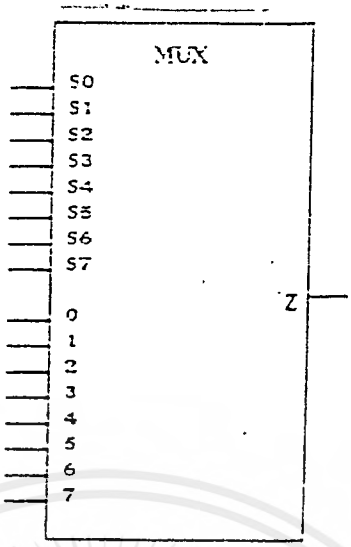
คำสั่งหยุดรอสามารถใช้ได้ภายในโพรเซสและโพรซีเจอร์ที่ไม่ถูกกำหนด Sensitivity List ไว้เท่านั้น Wait for จะหยุดรอเป็นเวลาเท่ากับ watiing time WAIT ON จะหยุดรอจนกว่าจะมีเหตุการณ์เกิดขึ้นกับสัญญาณ waiting sinsitvity list WAIT UNTIL จะหยุดรอจนกว่าเงื่อนไข waiting condition เปลี่ยนจาก FALSE เป็น TRUE WAIT จะหยุดรอตลอดไป

4. 3 การบรรยายเชิงกระแสข้อมูล (Dataflow Description of Hardware)

การบรรยายเชิงกระแสข้อมูลเป็นการบรรยายถึงการเลื่อนไหลของข้อมูลผ่านรีจิสเตอร์และบัสของระบบ เป็นระดับขั้นของการบรรยายที่อยู่ตรงกลางระหว่างการบรรยายเชิงพฤติกรรมและการบรรยายเชิงโครงสร้าง เครื่องมือที่ใช้ในการควบคุมการเคลื่อนไหวของข้อมูลได้แก่ conditional selectedและ guarded ในหัวข้อนี้จะแสดงให้เห็นถึงลักษณะและรูปแบบของการบรรยายเชิงกระแสข้อมูลรวมถึงข้อกำหนดและเงื่อนไขต่างพร้อมตัวอย่างประกอบ

4.3.1 การกำหนดเลือกข้อมูล (Multiplexing and Data Selection)

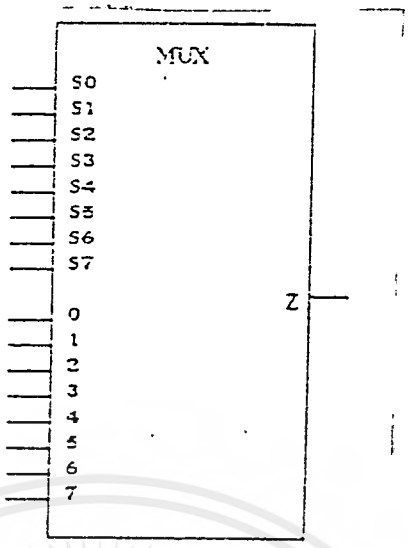
ในระบบดิจิตอล โครงสร้างของอุปกรณ์ฮาร์ดแวร์ส่วนใหญ่จะถูกใช้สำหรับการคัดเลือกและการนำข้อมูลเข้าสู่บัสและรีจิสเตอร์ รูปที่ 4.17 แสดงตัวอย่างโมเดลและการบรรยายเชิงกระแสข้อมูลของ 8- to-1 มัลติเพล็กเซอร์



(a)

```
ENTITY mux_8_to_1 IS
   PORT ( i7 , i6 , i5 , i4 , i3 , i2 , i1 , i0 : IN BIT ;
          s7 , s6 , s5 , s4 , s3 , s2 , s1 , s0  : IN BIT ;
          z : OUT BIT );
END mux_8_to_1;

ARCHITECTURE dataflow OF mux_8_to_1 IS
    SIGNAL sel_lines : BIT_VECTOR ( 7 DOWNTO 0 );
BEGIN
   sel_lines <= s7 & s6 & s5 & s4 & s3 & s2 & s1 & s0 ;
   WITH sel_lines SELECT
 z <= '0' AFTER 3 NS WHEN " 00000000";
        i7 AFTER 3 NS WHEN " 10000000";
        i6 AFTER 3 NS WHEN " 01000000";
        i5 AFTER 3 NS WHEN " 00100000";
        i4 AFTER 3 NS WHEN " 00010000";
        i3 AFTER 3 NS WHEN " 00001000";
        i2 AFTER 3 NS WHEN " 00000100";
        i1 AFTER 3 NS WHEN " 00000010";
        i0 AFTER 3 NS WHEN OTHER;
END dataflow;
```

(b)



(a)

```
ENTITY mux_8_to_1 IS
   PORT ( i7 , i6 , i5 , i4 , i3 , i2 , i1 , i0 : IN BIT ;
          s7 , s6 , s5 , s4 , s3 , s2 , s1 , s0  : IN BIT ;
          z :  OUT BIT );
END  mux_8_to_1;

ARCHITECTURE  dataflow  OF  mux_8_to_1  IS
    SIGNAL  sel_lines : BIT_VECTOR ( 7 DOWNTO 0 );
BEGIN
   sel_lines  <=  s7 & s6 & s5 & s4 & s3 & s2 & s1 & s0 ;
   WITH  sel_lines SELECT
 z <= '0'  AFTER  3  NS WHEN " 00000000";
    --- i7  AFTER  3  NS WHEN " 10000000";
        i6  AFTER  3  NS WHEN " 01000000";
        i5  AFTER  3  NS WHEN " 00100000";
        i4  AFTER  3  NS WHEN " 00010000";
        i3  AFTER  3  NS WHEN " 00001000";
        i2  AFTER  3  NS WHEN " 00000100";
        i1  AFTER  3  NS WHEN " 00000010";
        i0  AFTER  3  NS WHEN OTHER;
END dataflow;
```

(b)

รูปที่ 4.17 แสดง 18-to-1 มัลติเพล็กเซอร์

(a) โมเดล

(b) การบรรยายเชิงกระแสข้อมูล

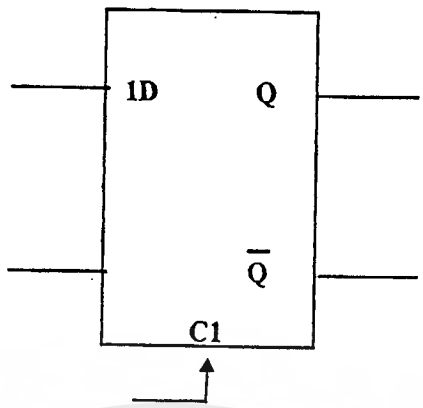
sei_lines เป็นสัญญาณที่กำหนดขึ้นมาเพื่อรวบรวมสัญญาณเลือกทั้ง 8 อินพุท(s7 - s0) เข้าด้วยกันและนำไปใช้เป็นสัญญาณในการกำหนดเลือกอินพุทตัวใดตัวหนึ่งให้เข้ากับเอาท์พุท z ถ้าไม่มีอินพุทใดมีค่าเป็น '1-' เลย ค่า '0' จะถูกส่งให้กับเอาท์พุท z หลังจากเวลาผ่านไป 3 NS ส่วนอินพุทอื่นๆ (i7, i6, i5, i4, i3,i2, i1, i0) จะถูกส่งให้ z ขึ้นอยู่กับสัญญาณเลือกใด (s7, s6, s5, s4, s3, s2, s1, s0) มีค่าเป็น '1'

4.3.2 การกำหนดการ์ด (Guarded Signal Assignments)

ในการบรรยายเชิงกระแสข้อมูล เราสามารถตั้งเงื่อนไขสำหรับการกำหนดค่าสัญญาณใดๆ โดยใช้คำสั่ง GUARDED ซึ่งมีรูปแบบในการเขียนดังนี้

```
target <= GUARDED waveforms or conditional waveforms or selected waveforms:
```

รูปที่ 4.18 แสดงตัวอย่างการใช้ GUARDED ในการบรรยายการทำงานของ d_flipflop การกำหนด GUARDED ทำได้โดยใช้รูปแบบของคำสั่ง BLOCK ตามด้วยเงื่อนไขภายในวงเล็บ ซึ่งผลลัพธ์ที่ได้จากวงเล็บนี้ก็คือ GUARDED นั่นเอง ในรูปเป็นการบรรยายการทำงานของ d_flipflop ที่มีการทำงานเมื่อสัญญาณนาฬิกามีการเปลี่ยนแปลงจาก '0' เป็น '1' หรือสัญญาณขาขึ้น จะเห็นว่า GUARDED ถูกใช้เป็นเงื่อนไขในการกำหนดค่าให้กับกับ q และ qb นั่นคือ ถ้าเงื่อนไขภายในวงเล็บมีค่าเป็น TRUE (GUARDED เป็น TRUE) ค่า d และ NOT d จะถูกส่งค่าให้กับ q และ qb หากเงื่อนไขภายในวงเล็บมีค่าเป็น FALSE (GUARDED เป็น FALSE) ค่า d และ NOT d ก็จะไม่ถูกส่งค่าให้กับ q และ qb



(a)

```
ENTITY  d_flipflop IS
 GENERIC  (delay1 : TIME := 4 NS ; delay2 : TIME := 5 NS ; )
 PORT ( d,c : IN BIT ; q ,qb : OUT BIT );
END  d_flipflop;

 ARCHITECTURE  guarding OF d_flipflop IS
 BEGIN
  ff : BLOCK ( c = '1'  AND  NOT c'STABLE )
     BEGIN
        q <= GUARDING d AFTER delay 1;
        qb <= GUARDING NOT d AFTER delay2;
      END BLOCK ff;
END guarding ;
```

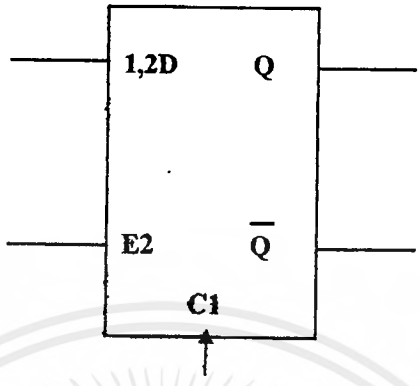
(b)

รูป 5.18 แสดงตัวอย่างการใช้ GUARDED

5.3.3 การกำหนดโครงข่ายของการ์ด (Nesting Guarded Blocks)

เราสามารถกำหนดการ์ดเป็นโครงข่ายซ้อนๆกันได้โดยการกำหนด BLOCK ซ้อนๆกัน ซึ่งจะทำให้เราสามารถออกแบบอุปกรณ์ที่มีการทำงานที่ซับซ้อนมากกว่าได้ พิจารณา d_flipflop ในรูปที่ 19 แสดงตัวอย่างการบรรยายการทำงานที่ซับซ้อนมากกว่าของ d_flipflop ในตัวอย่างข้างต้น ซึ่งนอกจากสัญญาณอินพุท d จะถูกส่งผ่านไปยัง q โดยมีเงื่อนไขว่า

(c = '1' AND NOT c' STABLE) ต้องเป็น TRUE แล้ว ในกรณีนี้ยังมีเงื่อนไขเพิ่มเติมอีกว่าสัญญาณ enable (e) ต้องเป็น TRUE อีกด้วย



(a)

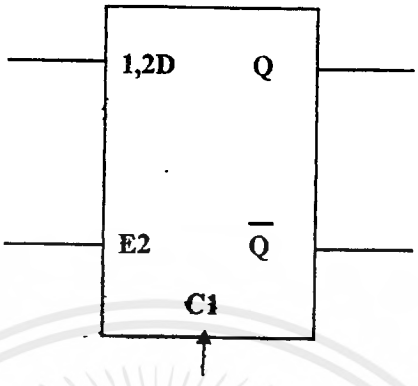
```
ENTITY de_flipflop IS
    GENERIC (delay1 : TIME := 4 NS ; delay2 : TIME := 5 NS ; )
    PORT ( d , e , c : IN BIT ; q , qb : OUT BIT );
END d_flipflop;

ARCHITECTURE guarding OF de_flipflop IS
BEGIN
 edge : BLOCK ( c = '1' AND NOT c'STABLE )
      BEGIN
       gate : BLOCK ( e = '1' AND GUARD )
            BEGIN
                q <= GUARDING d AFTER delay 1;
                qb <= GUARDING NOT d AFTER delay2;
            END BLOCK gate;
      END BLOCK edge;
END guarding ;
```

(b)

รูปที่ 4.19 แสดงตัวอย่างการใช้ NESTING GUARDED

GUARD ใน gate : BLOCK แทนเงื่อนไขใดๆที่อยู่ภายนอก ในกรณีนี้หมายถึงเงื่อนไขภายในวงเล็บ (c = '1' AND NOT c STABLE) ในขณะที่ GUARDED จะแทนเงื่อนไขทั้งหมด ในกรณีนี้คือ (e = '1') AND (c = '1' AND NOT c STABLE)

(c = '1' AND NOT c' STABLE) ต้องเป็น TRUE แล้ว ในกรณีนี้ยังมีเงื่อนไขเพิ่มเติมอีกว่าสัญญาณ enable (e) ต้องเป็น TRUE อีกด้วย



(a)

```
ENTITY de_flipflop IS
    GENERIC (delay1 : TIME := 4 NS ; delay2 : TIME := 5 NS ; )
    PORT ( d , e , c : IN BIT ; q , qb : OUT BIT );
END d_flipflop;

ARCHITECTURE guarding OF de_flipflop IS
BEGIN
 edge : BLOCK ( c = '1' AND NOT c'STABLE )
      BEGIN
       gate : BLOCK ( e = '1' AND GUARD )
            BEGIN
                q <= GUARDING d AFTER delay 1;
                qb <= GUARDING NOT d AFTER delay2;
            END BLOCK gate;
      END BLOCK edge;
END guarding ;
```

(b)

รูปที่ 4.19 แสดงตัวอย่างการใช้ NESTING GUARDED

GUARD ใน gate : BLOCK แทนเงื่อนไขใดๆที่อยู่ภายนอก ในกรณีนี้หมายถึงเงื่อนไขภายในวงเล็บ (c = '1' AND NOT c STABLE) ในขณะที่ GUARDED จะแทนเงื่อนไขทั้งหมด ในกรณีนี้คือ (e = '1') AND (c = '1' AND NOT c STABLE)

4.3.4 การเลือกสรรข้อมูล (Resolving Between Several Driving Values)

ในทางฮาร์ดแวร์มีหลายๆกรณีที่เรามีความจำเป็นต้องเชื่อมต่อหลายๆเอาท์พุทไปยังอินพุทใดอินพุทหนึ่ง ซึ่งในกรณีเช่นนี้อาจจะทำให้เกิดความสับสนของสัญญาณที่ปะปนกันทำให้ไม่สามารถรู้ค่าที่แน่นอนได้ ในทาง VHDL หมายถึงการกำหนดค่าจากหลายๆสัญญาณให้กับสัญญาณเดียว ซึ่งก็จะให้ผลลัพธ์ที่สับสนได้เหมือนกัน ดังตัวอย่างในรูปที่ 20 ซึ่งจะไม่สามารถบอกได้เลยว่าค่าของ circuit_noce มีค่าเป็นอะไร ถ้าสมมติว่าค่าอินพุท a เป็น '1' และค่าอินพุทอื่นๆ เป็น '0'

```
ENTITY y_circuit IS
    PORT ( a , b , c , d : IN BIT ; z : OUT BIT );
END y_circuit ;

ARCHITECTURE smoke_generator OF y_circuit IS
    SIGNAL circuit_note : BIT ;
BEGIN
      circuit_node <= a ;
      circuit_node <= b;
      circuit_node <= c ;
      circuit_node <= d ;
      z <= circuit_node;
END smoke_generator;
```

รูปที่ 4.20 แสดงการกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน

กรณีเช่นนี้เราสามารถแก้ไขได้โดยการกำหนดรูปแบบฟังก์ชั่นขึ้นมาเพื่อใช้เลือกสรรข้อมูล

รูปที่ 4.21 แสดงตัวอย่างฟังก์ชั่นเลือกสรรข้อมูลแบบANDซึ่งกำหนดให้สัญญาณที่เข้ามาที่โหนดเดียวกันจะถูกนำมารวมกันในลักษณะของการ AND เสียก่อน

```
FUNCTION anding (driver : BIT_VECTOR) RETURN BIT IS
    VARIABLE accumulator : BIT := '1';
    BEGIN
                                                (มีต่อ..)
```

```
( ต่อ..) FOR i IN drivers'RANGE LOOP
          accumulator := accumulator AND drivers( i );
        END LOOP
     RETURN    accumulator;
 END anding ;
```

รูปที่ 4.21 แสดงฟังก์ชั่นเลือกสรรข้อมูลแบบ anding

นำฟังก์ชั่นในรูปที่ 4.21 มาใช้ร่วมกับการบรรยายในรูปที่ 20 ก็สามารถแก้ปัญหาเรื่องความสับสนของข้อมูลได้ ดังรูปที่ 4.22

```
ARCHITECTURE wired_and OF y_circuit
FUNCTION anding (driver : BIT_VECTOR) RETURN BIT IS
   VARIABLE accumulator : BIT := '1';
   BEGIN
    FOR i IN drivers'RANGE LOOP
       accumulator := accumulator AND drivers( i );
    END LOOP
    RETURN    accumulator;
END anding ;
SIGNAL circuit_note : BIT ;
   BEGIN
          circuit_node <= a ;
          circuit_node <= b;
          circuit_node <= c ;
          circuit_node <= d ;
         z <= circuit_node;
    END wired_and;
```

รูปที่ 4.22 แสดงการใช้ฟังก์ชั่น anding กับการเลือกสรรข้อมูล

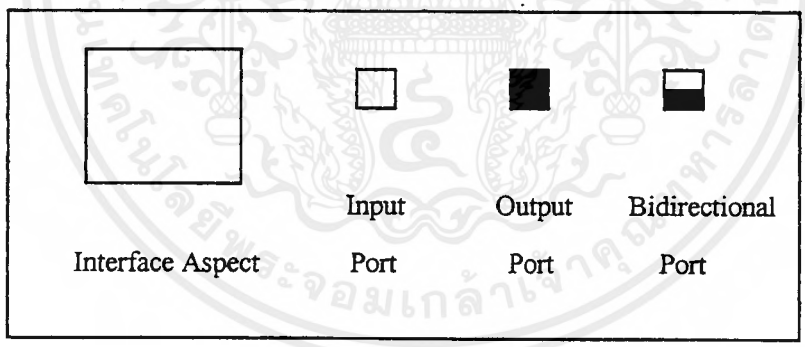
4.4 การบรรยายเชิงโครงสร้าง (Structural Specification of Hardware)

การบรรยายการทำงานของระบบในเชิงโครงสร้างจะต้องแสดงรายการของอุปกรณ์ทั้งหมดที่ใช้ในระบบนั้นและต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่างๆด้วย เพราะว่าการบรรยายในระดับนี้เป็นการบรรยายที่ใกล้เคียงลักษณะของฮาร์ดแวร์จริงที่สุด VHDL ได้จัดเตรียมเครื่องมือและลักษณะโครงสร้างของการบรรยายในระดับนี้ไว้ที่สำคัญ 4 ลักษณะคือ 1) ความสามารถในการเลือกหรือกำหนดอุปกรณ์ที่ต้องการได้จากไลบรารี 2) การสร้างไลบรารีเพื่อเก็บอุปกรณ์ที่ผู้ใช้ออกแบบไว้เองได้ 3) กลไกในการเชื่อมต่อระหว่างอุปกรณ์ 4) โครงสร้างการกำหนดอุปกรณ์ชนิดเดียวกันซ้ำๆกัน ในหัวข้อนี้จะได้แสดงรูปแบบและการบรรยายเชิงโครงสร้างโดยใช้ตัวอย่างการออกแบบวงจรเปรียบเทียบบิตตั้งแต่ระดับเกทจนถึงระดับวงจร

4.4.1 การกำหนดไลบรารี (Parts Library)

ในหัวข้อนี้จะแสดงการออกแบบเกทพื้นฐาน 3 ชนิด คือ INVERTER, NAND เกทชนิด 2 อินพุท และ 3 อินพุท ซึ่งเกทพื้นฐานเหล่านี้จะถูกเก็บไว้ในไลบรารีและนำมาใช้เป็นส่วนประกอบเพื่อออกแบบเป็นวงจรที่ใหญ่ขึ้นต่อไป

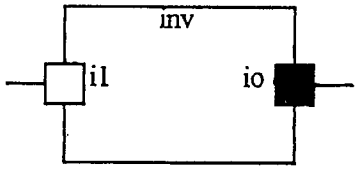
ในการออกแบบได้กำหนดสัญลักษณ์ที่เกี่ยวข้องไว้ดังนี้



รูปที่ 4.23 แสดงสัญลักษณ์แทนตัวอุปกรณ์

4.4.1.1 อินเวิร์เตอร์โมเดล

รูปที่ 4.24 (a) แสดงสัญลักษณ์ของ Inverter รูป (b) แสดงการบรรยายการเชื่อมต่อโดยกำหนดให้อุปกรณ์มีชื่อว่า inv และมีพอร์ตการติดต่อ i1 เป็นอินพุทและ o1 เป็นเอาท์พุทมีชนิดของข้อมูลที่ผ่านเข้าออกเป็น BIT เราสามารถกำหนดพอร์ตเป็นแบบสองทิศทาง (bidirectional) ได้โดยการกำหนดโหมดเป็น INOUT รูป (c) แสดงการบรรยายการทำงานของอุปกรณ์



(a)

```
ENTITY  inv  IS
  PORT ( i1 : IN BIT ; o1  : OUT  BIT );
END  inv ;
```

(b)

```
ARCHITECTURE  single_delay  OF  inv  ID
BEGIN
        o1 <= NOT   i1   AFFTER  4 NS ;
END  single_delay;
```

(c)

รูปที่ 4.24 แสดง Inverter Model

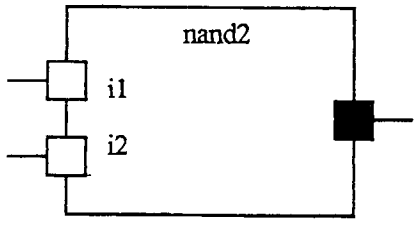
(a) แสดงสัญลักษณ์ของ Inverter

(b) แสดงการบรรยายการเชื่อม

(c) แสดงการบรรยายการทำงาน

4.4.1.2 NAND เกทโมเดล (NAND Gate Models)

รูปที่ 4.25 แสดงสัญลักษณ์และการบรรยายของ NAND เกทชนิด 2 อินพุท



(a)

```
ENTITY nand2 IS
   PORT ( i1 , i2 : IN BIT ; o1 : OUT BIT );
END nand2;
```

(b)

```
ARCHITECTURE single_delay OF nand2 IS
BEGIN
  o1 <= i1 NAND i2 AFTER 5 NS ;
END single_delay ;
```

(c)

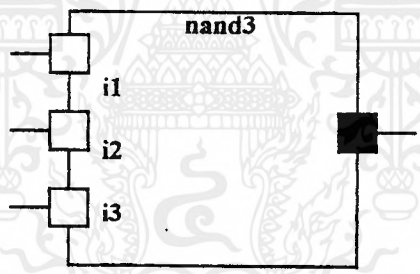
รูปที่ 4.25 แสดง NAND2 Model

(a) แสดงสัญลักษณ์ของ NAND2

(b) แสดงการบรรยายการเชื่อม

(c) แสดงการบรรยายการทำงาน

รูปที่ 4.26 แสดงสัญลักษณ์และการบรรยายของ NAND เกทชนิด 3 อินพุท



(a)

```
ENTITY nand3 IS
PORT ( i1 ,i2 ,i3 : IN BIT ; o1 : OUT BIT ) ;
END nand3 ;
```

(b)

```
ARCHITECTURE single_delay OF nand3 IS
BEGIN
   o1 <= NOT ( i1 AND i2 AND i3 ) AFTER 6 NS;
END single_delay;
```

(c)

รูปที่ 4.26 แสดง NAND3 Model

(a) แสดงสัญลักษณ์ของ NAND3

(b) แสดงการบรรยายการเชื่อม

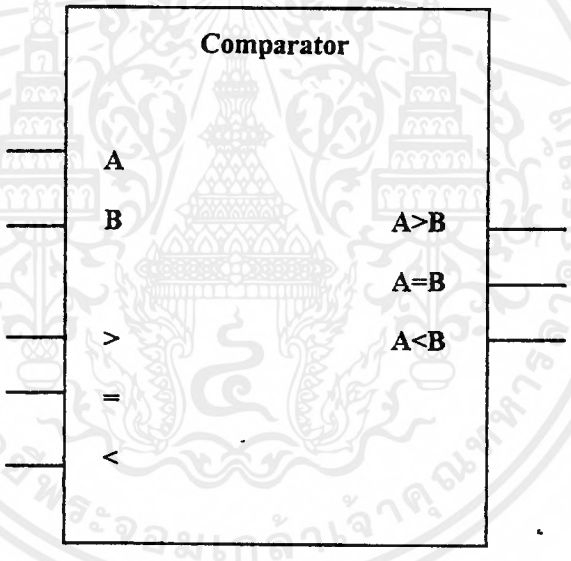
(c) แสดงการบรรยายการทำงาน

4.4.2 การเชื่อมต่ออุปกรณ์พื้นฐาน (Wiring of Primitive)

เมื่อเราได้ออกแบบเกทพื้นฐานเรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานเหล่านี้มาเชื่อมต่อกันเป็นวงจร ในหัวข้อนี้จะแสดงการออกแบบและการบรรยายเชิงโครงสร้างของวงจรเปรียบเทียบโดยใช้ Inverter และ NAND เกท

4.4.2.1 ตัวอย่างการออกแบบวงจรเปรียบเทียบ (Logic Design of Comparator)

วงจรเปรียบเทียบทีละบิด (bit-comparator) ประกอบด้วยสัญญาณข้อมูล 2 อินพุท, สัญญาณควบคุม 3 อินพุท และสัญญาณผลเปรียบเทียบ 3 เอาท์พุท ดังรูปที่ 4.27



รูปที่ 4.27 แสดงวงจรเปรียบเทียบทีละบิด

เอาท์พุท A>B มีค่าเป็น '1' เมื่ออินพุท A มีค่ามากกว่า B (AB = 10) หรือถ้า A เท่ากับ B และอินพุท > มีค่าเป็น '1' เอาท์พุท A=B มีค่าเป็น '1' เมื่ออินพุท A เท่ากับอินพุท B = มีค่าเป็น '1' ส่วนเอาท์พุท A<B จะตรงข้ามกับเอาท์พุท A>B นั่นคือมีค่าเป็น '1' เมื่ออินพุท A มีค่าน้อยกว่า B (AB = 01) หรือถ้า A เท่ากับ B และอินพุท <.มีค่าเป็น '1' ซึ่งสามารถเขียนเป็นสมการบูลีนในรูปของ NAND เกทได้ดังรูปที่ 4.28

(b) แสดงการบรรยายการเชื่อม

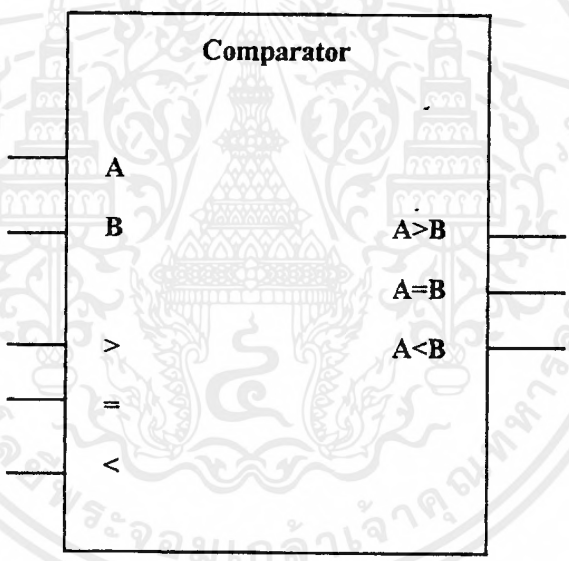
(c) แสดงการบรรยายการทำงาน

4.4.2 การเชื่อมต่ออุปกรณ์พื้นฐาน (Wiring of Primitive)

เมื่อเราได้ออกแบบเกทพื้นฐานเรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานเหล่านี้มาเชื่อมต่อกันเป็นวงจร ในหัวข้อนี้จะแสดงการออกแบบและการบรรยายเชิงโครงสร้างของวงจรเปรียบเทียบ โดยใช้ Inverter และ NAND เกท

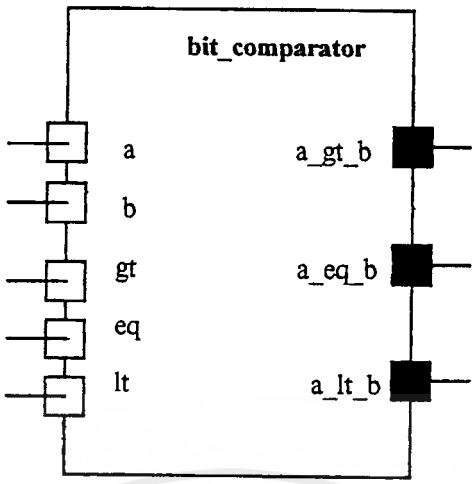
4.4.2.1 ตัวอย่างการออกแบบวงจรเปรียบเทียบ (Logic Design of Comparator)

วงจรเปรียบเทียบทีละบิด (bit-comparator) ประกอบด้วยสัญญาณข้อมูล 2 อินพุท, สัญญาณควบคุม 3 อินพุท และสัญญาณผลเปรียบเทียบ 3 เอาท์พุท ดังรูปที่ 4.27



รูปที่ 4.27 แสดงวงจรเปรียบเทียบทีละบิด

เอาท์พุท A>B มีค่าเป็น '1' เมื่ออินพุท A มีค่ามากกว่า B (AB = 10) หรือถ้า A เท่ากับ B และอินพุท > มีค่าเป็น '1' เอาท์พุท A=B มีค่าเป็น '1' เมื่ออินพุท A เท่ากับอินพุท B = มีค่าเป็น '1' ส่วนเอาท์พุท A<B จะตรงข้ามกับเอาท์พุท A>B นั่นคือมีค่าเป็น '1' เมื่ออินพุท A มีค่าน้อยกว่า B (AB = 01) หรือถ้า A เท่ากับ B และอินพุท < มีค่าเป็น '1' ซึ่งสามารถเขียนเป็นสมการบูลีนในรูปของ NAND เกทได้ดังรูปที่ 4.28




```
a_gt_b  = (( a . gt )'.( b'. gt )'.( a . b')')'
a_eq_b = (( a . b . eq )'.( a'.b'.eq )')'
a_lt_b = (( a'.lt )'.(b.lt)'.(a'.b)')'
```

รูปที่ 4.28 แสดงสัญลักษณ์และสมการบูลีนของวงจรเปรียบเทียบทีละบิท

4.4.2.2 ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบทีละบิต (VHDL Description of Bit-Comparator)

รูปที่ 4.29 แสดงการบรรยายการเชื่อมต่อของวงจรเปรียบเทียบทีละบิตตามสัญลักษณ์ใน รูปที่ 4.28 เครื่องหมาย "--" ใช้แทนข้อความอธิบาย (comment)

```
ENTITY bit_comparator  IS
  PORT(a,b,                    -- data  inputs
       gt,                     -- previous  greater than
       eq,                     -- previous equal
       It  : IN BIT;           -- previous less than
      a_gt_b,                  -- greater
      a_eq_b,                  -- equal
     a_It_b : OUT BIT);        --less than
END  bit_comparator;
```

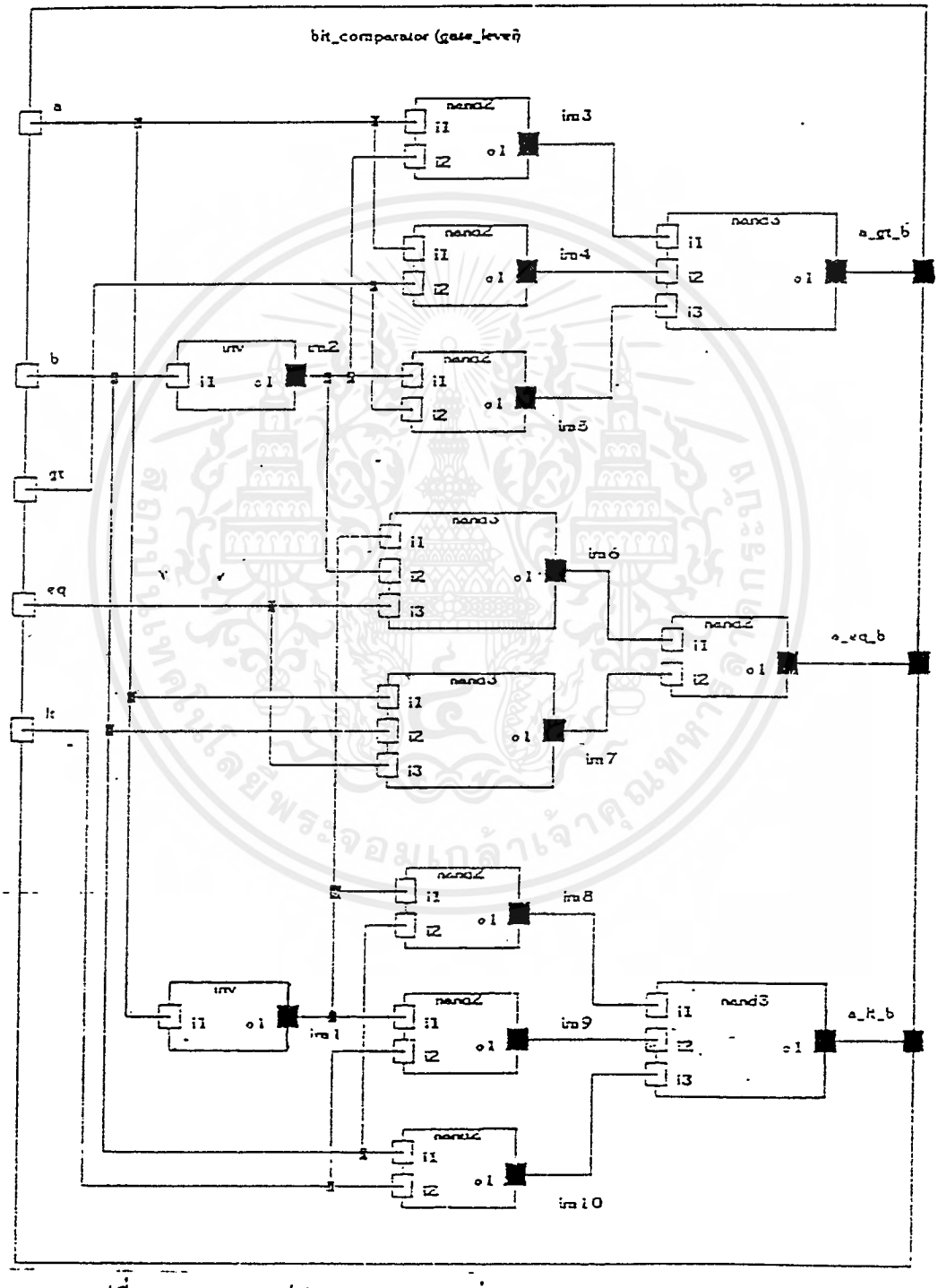
รูปที่ 4.29 แสดงการบรรยายการเชื่อม

```
ARCHITECTURE   gate_level OF  bit_comparator IS
  COMPONENT  n1 PORT( i1 : IN BIT; o1 : OUT BIT);END COMPONENT;
  COMPONENT  n1 PORT( i1,i2 : IN BIT; o1 : OUT BIT);END COMPONENT;
  COMPONENT  n1 PORT( i1,i2,i3 : IN BIT; o1 : OUT BIT);END COMPONENT;

  FOR ALL : n1 USE  ENTITY WORK.inv ( single_delay);
  FOR ALL : n2 USE  ENTITY WORK.nand2 ( single_delay);
  FOR ALL : n3 USE  ENTITY WORK.nand3 ( single_delay);
  SIGNAL  im1,im2,im3,im4,im5,im6,im7,im8,im9,im10 : BIT;
BEGIN
   -  a_gt_b output
         g0 : n1 PORT MAP( a , im1);
         g1 : n1 PORT MAP( b , im2);
         g2 : n2 PORT MAP( a , im2 , im3);
         g3 : n2 PORT MAP( a_gt , im4);
         g4 : n2 PORT MAP( im2 , gt , im5);
         g5 : n3 PORT MAP( im3 , im4 , im5 , a_gt_b);
          -  a_eq_b  output
         g6: n3 PORT MAP( im1 , im2 , eq , im6);
         g7: n3 PORT MAP( a , b , eq , im7);
         g8: n3 PORT MAP( im6 , im7 ,  a_eq_b);
         -  a_lt_b output
         g9: n2 PORT MAP( im1 , b , im8);
         g10: n2 PORT MAP( im1 , lt , im9);
         g11: n2 PORT MAP( b , lt , im10);
         g12: n3 PORT MAP( im8 , im9 , im10 , a_lt_b);
    END gate_level ;
```

รูปที่ 4.30 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต

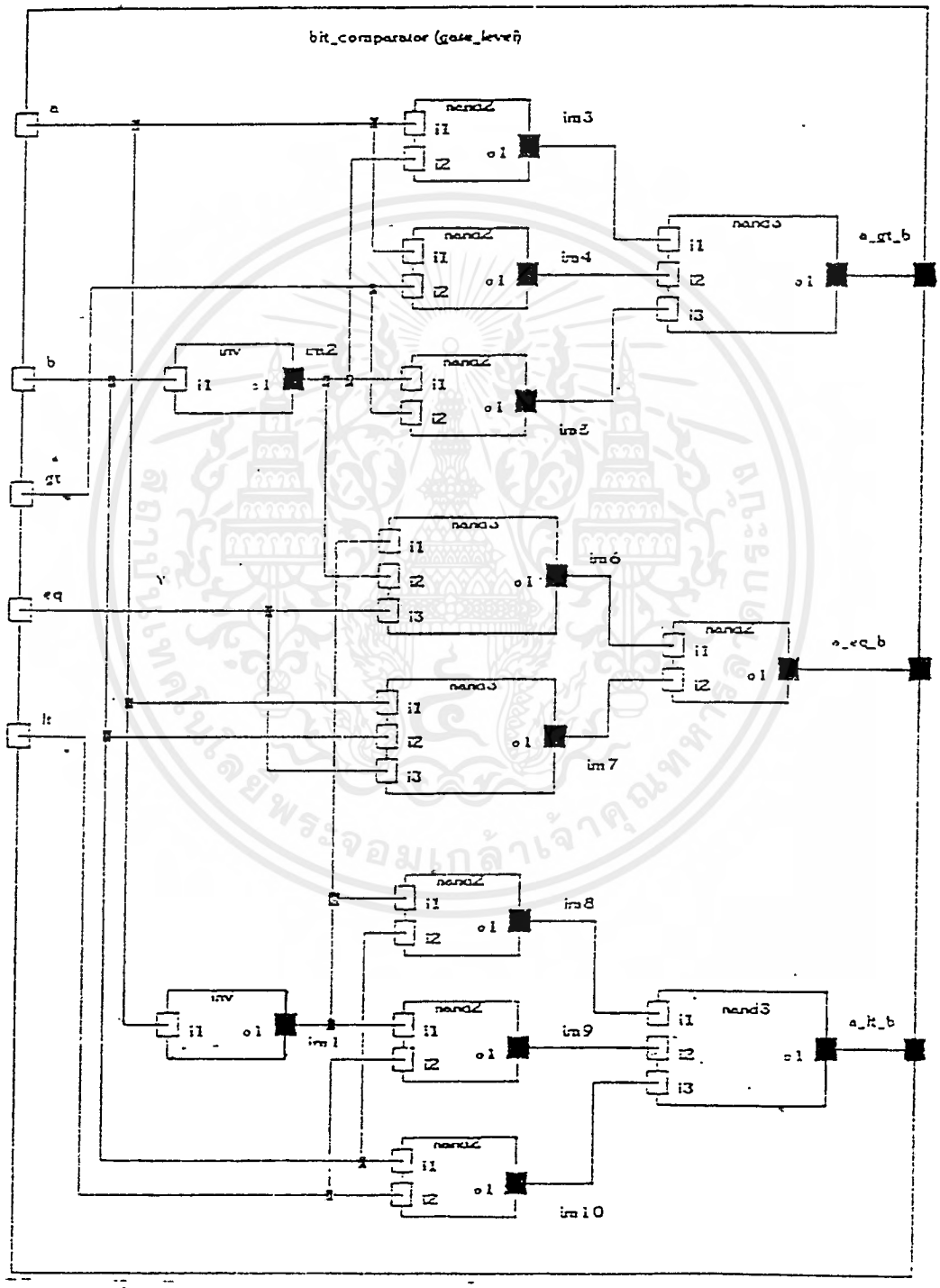
รูปที่ 4.30 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบ โดยกำหนดชื่อการบรรยายเป็น gate_level ภายในมีการกำหนดรายการพื้นฐานที่ต้องใช้เอาไว้ 3 ชนิด และกำหนดชื่อเป็น n1, n2, n3 โดยกำหนดให้ n1 อ้างอิงถึง Inverter เกท n2 อ้างอิงถึง nand2 เกท และ n3 อ้างอิงถึง nand3 เกท คำสั่ง FOR ALL : n1 เป็นการกำหนดให้อุปกรณ์ทุกตัวที่ขึ้นชื่อต้นด้วย n1 ให้อ้างอิง inverter

เกท ในกรณีของ n2 และ n3 ก็เช่นเดียวกัน WORK เป็นการกำหนดตำแหน่งที่อยู่หรือไลบรารีที่เก็บอุปกรณ์ที่อ้างอิงถึง ซึ่งอาจจะเป็นชื่อไดเร็กทอรีใดๆ (กรณีที่ใช้ค่า WORK จะหมายถึงไดเร็กทอรีปัจจุบัน) รูปที่ 4.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิตตามสมการบูลีนโดยใช้เกทพื้นฐานที่ได้ออกแบบไว้แล้ว



รูปที่ 4.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิต

เกท ในกรณีของ n2 และ n3 ก็เช่นเดียวกัน WORK เป็นการกำหนดตำแหน่งที่อยู่หรือไลบรารีที่เก็บอุปกรณ์ที่อ้างอิงถึง ซึ่งอาจจะเป็นชื่อไดเร็กทอรีใดๆ (กรณีที่ใช้ค่า WORK จะหมายถึงไดเร็กทอรีปัจจุบัน) รูปที่ 4.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิตตามสมการบูลีนโดยใช้เกทพื้นฐานที่ได้ออกแบบไว้แล้ว



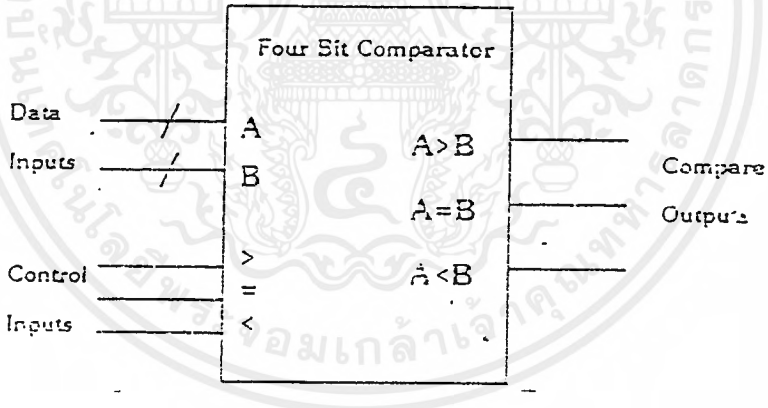
รูปที่ 4.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิต

4.4.3 การเชื่อมต่ออุปกรณ์ชนิดเดียวกัน (Writing Iterative Networks)

ในการเชื่อมต่ออุปกรณ์ต่างๆเข้าด้วยกันจะต้องอ้างอิงถึงอุปกรณ์ทีละตัว แต่หากเป็นการเชื่อมต่ออุปกรณ์ชนิดเดียวกันจำนวนหลายๆตัวเข้าด้วยกันแล้ว VHDL ได้จัดเตรียมเครื่องมือซึ่งเปรียบเสมือนเส้นทางลัดในการบรรยายโดยไม่จำเป็นต้องอ้างถึงอุปกรณ์ทุกๆตัว ในหัวข้อต่อไปจะได้ยกตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิตที่เรียกว่า nibble_comparator ซึ่งประกอบด้วยวงจรเปรียบเทียบทีละบิตที่ได้ออกแบบไว้ในหัวข้อก่อนหน้านี้จำนวน 4 ตัวต่อเข้าด้วยกัน

4.4.3.1 ตัวอย่างการออกแบบ VHDL ของวงจรเปรียบเทียบขนาด 4 บิต (Design of a 4-Bit Comparator)

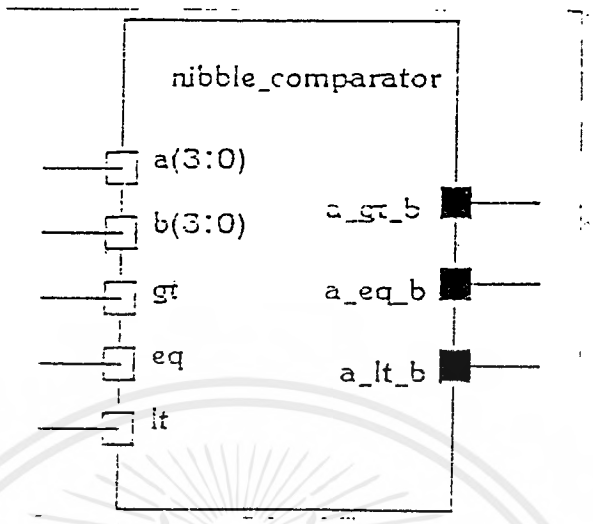
วงจรเปรียบเทียบขนาด 4 บิตมีสัญญาณข้อมูลขนาด 4 บิต จำนวน 2 อินพุท, สัญญาณควบคุม 3 อินพุท และสัญญาณผลเปรียบเทียบ 3 เอาท์พุท ดังรูปที่ 4.32



รูปที่ 4.32 แสดงวงจรเปรียบเทียบขนาด 4 บิต

4.4.3.2 ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบขนาด 4 บิต (VHDL Description of a 4-Bit Comparator)

รูปที่ 33 แสดงสัญลักษณ์ของวงจร nibble_comparator และการบรรยายการเชื่อมต่อของวงจร อินพุท a และ b เป็นอินพุทสำหรับข้อมูลขนาด 4 บิต เราสามารถกำหนดชนิดของอินพุทเป็น BIT_VECTOR ซึ่งเป็นอาเรย์ของ BIT ได้ด้วย



(a)

```
ENTITY nibble_comprator IS
  PORT(a,b : IN BIT_VECTOR(3 DOWNTO 0);  -- data inputs
       gt,                               -- previous greater tha
       eq,                               -- previous equal
       lt  : IN BIT;                     -- previos less than
       a_gt_b,                           -- greater
       a_eq_b,                           -- equal
       a_lt_b : OUT BIT);                -- less than
END  nibble_comparator;
```

(b)

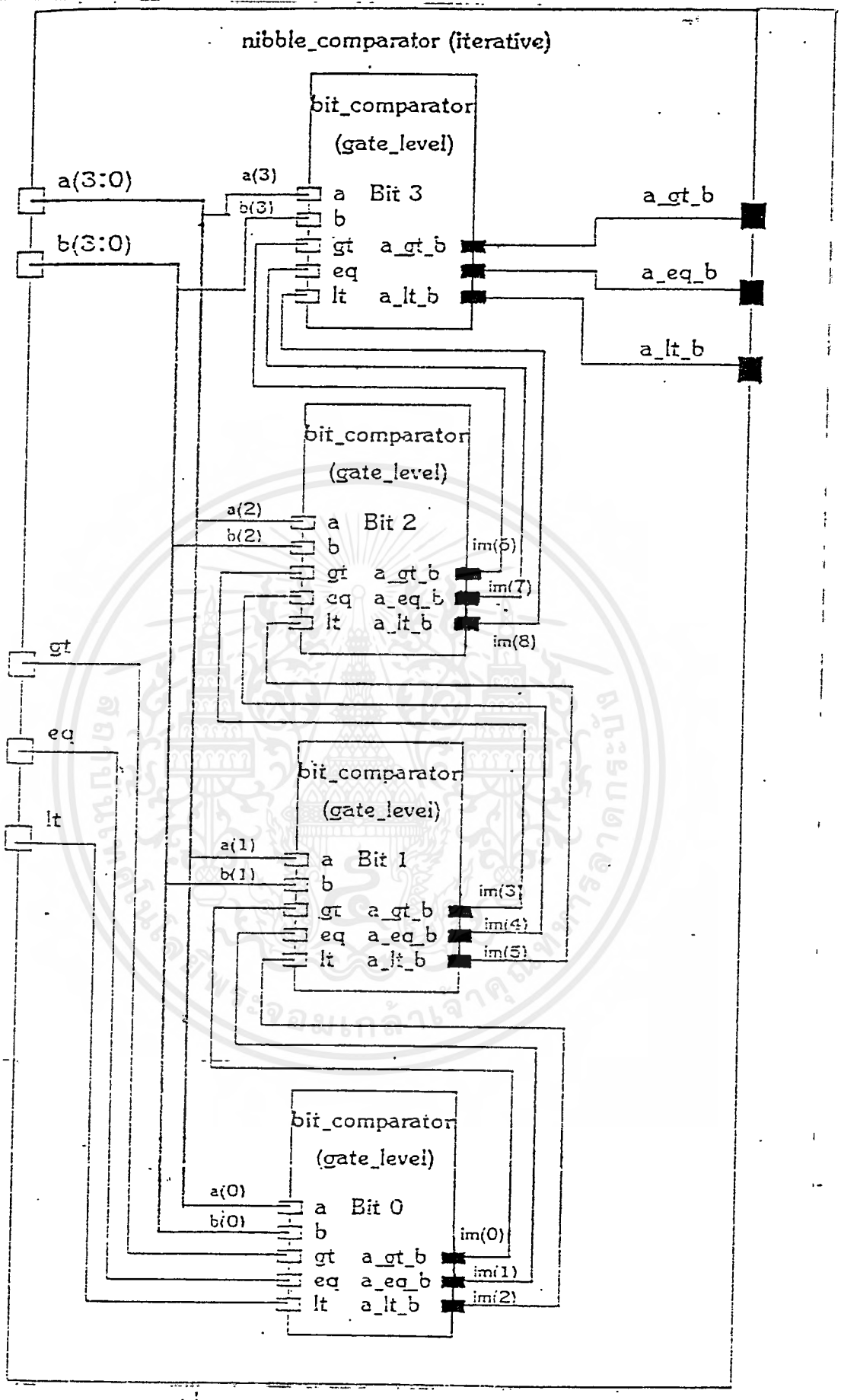
รูปที่ 4.33 แสดง nibble_comparator
(a) สัญลักษณ์ของวงจร
(b) การบรรยายการเชื่อมต่อ

รูปที่ 4.34 แสดงการบรรยายการทำงานของวงจร โดยใช้ comp1 ซึ่งอ้างอิงถึงวงจรเปรียบเทียบทีละบิต ค่า n เป็นค่าคงทีที่ใช้กำหนดจำนวนของวงจร comp1 ซึ่งในกรณีนี้เป็นวงจรเปรียบเทียบขนาด 4 บิต ดังนั้นจึงกำหนดค่า n เป็น 4. การบรรยายได้ใช้ FOR loop และ GENERATE ในการกำหนดการเชื่อมต่อของอุปกรณ์ชนิดเดียวกัน สังเกตว่าการบรรยายในลักษณะนี้ช่วยประหยัด

เวลาและสะดวกมากขึ้น ทั้งนี้เราสามารถกำหนดจำนวนอุปกรณ์ที่มาเชื่อมต่อกันจำนวนเท่าใดก็ได้ โดยไม่ต้องเปลี่ยนรูปแบบใดๆเลยนอกจากค่าของ n เท่านั้น รูปที่ 4.35 แสดงลักษณะของวงจรแบบสัญลักษณ์

```
ARCHITECTURE iterative OF nibble_comparator IS
  COMPONENT compl
    PORT ( a , b , gt , eq , lt  : IN BIT ; a_gt_b , a_eq_b , a_lt_b : OUT BIT);
  END COMPONENT ;
 FOR ALL : compl  USE  ENTITY  WORK.bit_comparpato (gate_level);
 CONSTANT  n : INTEGER := 4 ;
 SIGNAL  im  : BIT_VECTOR( 0 TO (n - 1) * 3 - 1);
BEGIN
 c_all : FOR  i  IN  0  TO  n-1  GENERATE
     l :  IF   i = 0  GENERATE
       least : compl  PORT MAP(a(i) , b(i) , gt , eq , lt , im(0) , im(1) , im(2));
       END GENERATE ;
    m :  IF   i =  0  GENERATE
        most : compl PORT MAP(a(i) , b(i) , im(i*3-3) , im(i*3-2) , im(i*3-1),
                                  a_gt_b,a_eq_b,a_lt_b);
        END GENERATE;
     r :  IF   i >  0 AND i <  n-1 GENERATE
        least : compl  PORT MAP(a(i) , b(i) , im(i*3-3) , im(i*3-2) , im(i*3-1),
                                  im(i*3 + 0) , im(i*3 + 1) , im(i*3  + 2));
        END GENERATE ;
    END GENERATE ;
END  iterative ;
```

รูปที่ 4.34 แสดงการบรรยายการทำงานของวงจร



รูปที่ 4.35 แสดงลักษณะของวงจรแบบสัญลักษณ์

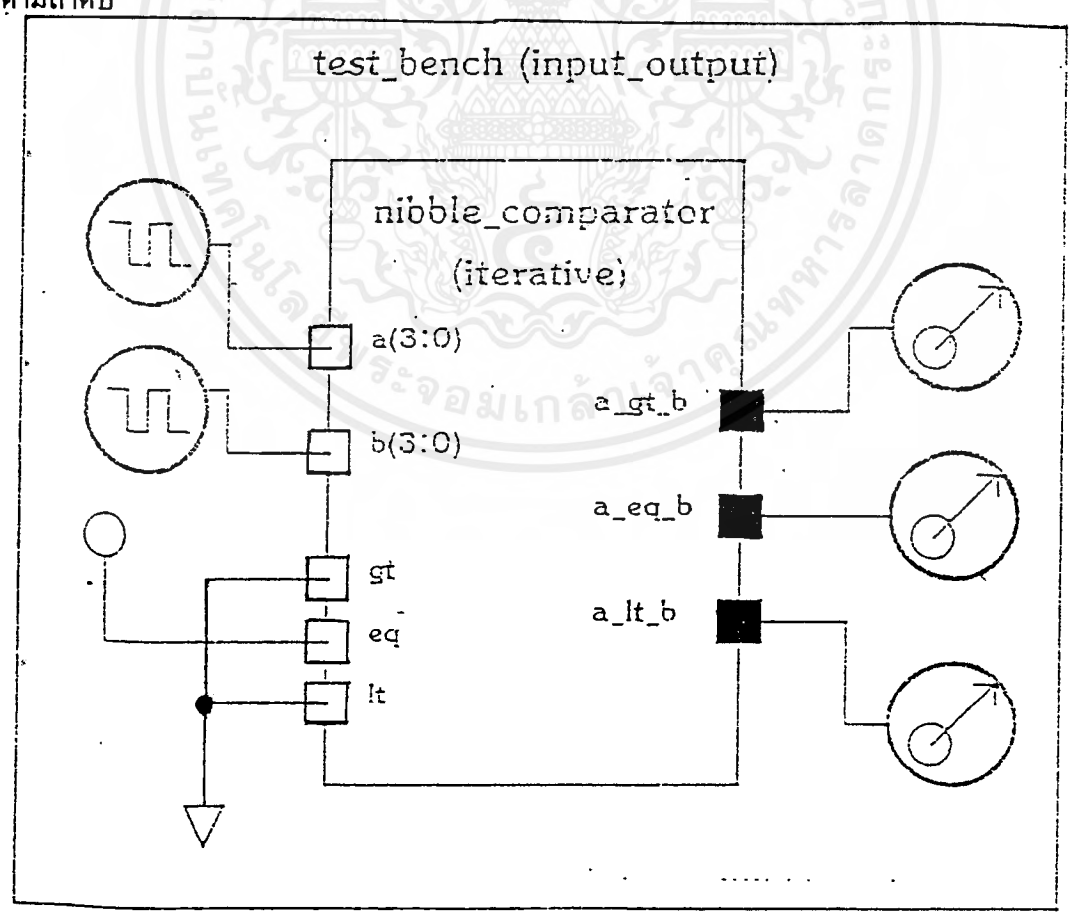
4.5 การทดสอบและเลียนแบบการทำงานของวงจร

(A Test Bench and Simulation)

นอกจากจะใช้ในการออกแบบวงจรแล้ว VHDL ยังสามารถนำมาใช้ในการสร้างแบบทดสอบเพื่อทดสอบและเลียนแบบการทำงานของอุปกรณ์นั้นๆได้ด้วย หัวข้อต่อไปนี้จะแสดงให้เห็นตัวอย่างแบบทดสอบเพื่อทำการทดสอบและเลียนแบบการทำงานของวงจรเปรียบเทียบขนาด 4 บิต การจัดเตรียมข้อมูลสำหรับส่งให้อินพุทของวงจร การแสดงผลการเลียนแบบและการทำงานร่วมกับแฟ้มข้อความเพื่อใช้เก็บรายงานผลการเลียนแบบ

4.5.1 รูปแบบการทดสอบ (Modeling a Test Bench)

แบบทดสอบจะต้องประกอบด้วยการกำหนดวงจรที่จะใช้ในการทดสอบและต้องจัดเตรียมข้อมูลที่จะส่งให้กับอินพุทของวงจร รูปที่ 4.36 แสดงรูปแบบของแบบทดสอบวงจรเปรียบเทียบขนาด 4 บิต แบบทดสอบไม่มีการเชื่อมต่อกับพอร์ตภายนอก ภายในได้จัดเตรียมรูปแบบของข้อมูลสำหรับอินพุท a และ b และเชื่อมต่ออินพุทควบคุม gn eq และ lt เข้ากับ gnd, vdd และ gnd ตามลำดับ



รูปที่ 4.36 แสดงรูปแบบของแบบทดสอบวงจรเปรียบเทียบขนาด 4 บิท

รูปที่ 4.37 แสดงการบรรยายแบบทดสอบ สังเกตว่าในส่วนของการบรรยายการเชื่อมต่อไม่ได้กำหนดพอร์ตเอาไว้เนื่องจากในแบบทดสอบไม่มีการติดต่อกับอุปกรณ์ภายนอก ดังนั้นในส่วนนี้จึงเว้นว่างไว้ ในส่วนของการบรรยายการเลียนแบบกำหนดให้ comp4 เป็นชื่ออุปกรณ์อ้างอิงถึงวงจร nibble_comparator ในส่วนของการเตรียมข้อมูลกำหนดให้ส่งข้อมูลเข้าสู่อินพุท a และ b ทุกๆ ช่วงเวลา 500 NS.

```
ENTITY  nibble_comparator_test_bench IS
END nibble_comparator_test_bench ;

ARCHITECTURE input _output OF nibble_comparator_test_bench IS
   COMPONENT  comp4 PORT ( a,b : IN BIT_VECTOR (3 DOWNTO 0);
                           gt , eq , lt : IN BIT;
                           a_gt_b , a_eq_b , a_lt_b : OUT  BIT) ;
   END COMPONENT;
   FOR  a1 : comp4  USE  ENTITY WORK.nibble_comparator ( iterative);
   SIGNAL  a , b : BIT_VECTOR(3 DOWNTO 0);
   SIGNAL  eql , lss , gtr : BIT ;
   SIGNAL  vdd : BIT := '1' ;
   SIGNAL   gnd : BIT := '0' ;
BEGIN
  a1  : comp4 PORT MAP(a , b , gnd , vdd , gnd , gtr , eql , lss );
  a2  : a <= "0000";                          -- a = b (steady state)
        "1111"  AFTER  0500 NS ;              -- a > b (worst case)
        "1110" AFTER  1500 NS ;               -- a < b (worst case)
        "1110" AFTER  2500 NS ;               -- a > b (need  bit 1 info)
        "1010" AFTER  3500 NS ;               -- a < b (need  bit 2  info)
        "0000" AFTER  4000 NS ;               -- a < b (steady state , prepare for next)
        "1111" AFTER  4500 NS ;               -- a = b (worst case)
        "0000" AFTER  5000 NS ;               -- a < b (need  bit 3 only , best case)
        "0000" AFTER  5500 NS ;               -- a = b (worst case)
        "1111" AFTER  6000 NS ;               -- a > b (need  bit 3 only , best case)

  a3  : b  <= "000"                           -- a = b (steady state)
        "1110"  AFTER  0500 NS ;              -- a > b (worst case)
        "1111" AFTER  1500 NS ;               -- a < b (worst case)          (มีต่อ..)
```

```
(ต่อ..)     "1100" AFTER  2500 NS ;        -- a > b (need  bit 1  info)
            "1100" AFTER  3500 NS ;        -- a < b (need  bit 2  info)
            "1111" AFTER  4000 NS ;        -- a < b (steady state , prepare for next)
            "1111" AFTER  4500 NS ;        -- a = b (worst case)
            "1111" AFTER  5000 NS ;        -- a < b (need  bit 3 only , best case)
            "0000" AFTER  5500 NS ;        -- a = b (worst case)
            "0000" AFTER  6000 NS ;        -- a > b (need  bit 3 only , best case)
    END   input_output
```

รูปที่ 4.37 แสดงการบรรยายแบบทดสอบ

4.5.2 รูปแบบการทำงานร่วมกับแฟ้มข้อความ (Formatted ASCII I/O Operations)

VHDL ได้จัดเตรียมรูปแบบของการติดต่อกับแฟ้มข้อความไว้ใน STD.TEXTIO. ไลบรารี รวมทั้งโพรซีเจอร์ในการอ่านและเขียน ต่างๆเอาไว้ โพรซีเจอร์ READLINE (f, 1) จะอ่านข้อความจากไฟล์ (ทีละหนึ่งบรรทัดและนำมาเก็บไว้ในบัฟเฟอร์ 1 โพรซีเจอร์ READ (1, v, ...) อ่านค่าจาก 1 ทีละค่า (หากในหนึ่งบรรทัดมีข้อมูลหลายค่าให้แยกข้อมูลด้วย หรือ ' ' (space)) นำมากำหนดค่าให้ v และจะเปลี่ยนชนิดของข้อมูลเป็นไปตามชนิดของ v ด้วย โพรซีเจอร์ WRITE (1,v,...) เขียนข้อมูลจาก v ลงในบัฟเฟอร์ LINE 1 และ โพรซีเจอร์ WRITELINE (f, 1) เขียนข้อมูลจากบัฟเฟอร์ 1 ลงสู่ไฟล์ f ทีละบรรทัดเช่นกัน ฟังก์ชั่น ENDFILE (f) ใช้สำหรับตรวจสอบจุดสิ้นสุดของไฟล์โดยจะให้ค่าเป็น TRUE เมื่อพบจุดสิ้นสุดของไฟล์ ชนิดของข้อมูลที่สามารถกำหนดให้กับค่า v ได้มีดังนี้ BIT, BIT_VECTOR,BOOLEAN, CHARACTOR, INTEGER, REAL, STRING และ TIME ถ้าหากเป็นชนิดของข้อมูลที่กำหนดขึ้นเองก็สามารถสร้างโพรซีเจอร์เพื่อเปลี่ยนชนิดของข้อมูลเสียก่อนได้

รูปที่ 4.28 แสดงตัวอย่างการทำงานร่วมกับแฟ้มข้อความ ในการอ้างอิงถึงไลบรารีให้ใช้คำสั่ง USE ชนิดของไฟล์กำหนดโดยคำสั่ง FILE ถ้าเป็นไฟล์เพื่อการอ่านก็กำหนดเป็น TEXT IS IN และกำหนดเป็น TEXT IS OUT ถ้าเป็นไฟล์เพื่อการเขียน ชนิดของข้อมูลที่ใช้ในการอ่านและเขียนกำหนดเป็น LINE ดังนั้นในการอ่านหรือเขียนจะกระทำทีละบรรทัด รูปที่ 4.29 แสดงรายงานผลที่ได้

```
USE STD.TEXTIO.ALL;
ENTITY two_phase_clock IS
END two_phase_clock ;

ARCHITECTURE input_output OF two_phase_clock IS
  SIGNAL c1 : BIT := '1';
  SIGNAL c2 : BIT := '0';
BEGIN
 phase1 : c1 <= NOT c1 AFTER 500 NS;
 phase2 : PROCESS
          BEGIN
            WAIT UNTIL c1 = '0' ;
            WAIT FOR 10 NS ;
             c2 <= '1' ;
            WAIT FOR 480 NS;
             c2 <= '0';
          END PROCESS phase2 ;
 writing : PROCESS(c1 , c2 )
          FILE flush : TEXT IS OUT " clock.out"
          VARIABLE filter : STRING(1 TO 3) ;
          VARIABLE i : LINE ;
          BEGIN
           WRITE( l.NOW.RIGHT.8.NS);
           IF c1'EVENT THEN
            WRITE( l.c1.RIGHT.3);
            WRITE( l.filter.LEFT.0);
           ELSE
            WRITE( l.filter.LEFT.0);
            WRITE( l.c2.RIGHT.3);
           END IF;
           WRITELINE(flush.l);
          END PEOCESS writing;
END input output;
```

รูปที่ 4.38 แสดงตัวอย่างการทำงานร่วมกับเพิ่มข้อความ

```
USE STD.TEXTIO.ALL;
ENTITY two_phase_clock IS
END two_phase_clock ;

ARCHITECTURE input_output OF two_phase_clock IS
  SIGNAL c1 : BIT := '1';
  SIGNAL c2 : BIT := '0';
BEGIN
 phase1 : c1 <= NOT c1 AFTER 500 NS;
 phase2 : PROCESS
          BEGIN
            WAIT UNTIL c1 = '0' ;
            WAIT FOR 10 NS ;
              c2 <= '1' ;
            WAIT FOR 480 NS;
              c2 <= '0';
          END PROCESS phase2 ;
 writing : PROCESS(c1 , c2 )
          FILE flush : TEXT IS OUT " clock.out"
          VARIABLE filter : STRING(1 TO 3) ;
          VARIABLE i : LINE ;
          BEGIN
           WRITE( l.NOW.RIGHT.8.NS);
           IF c1'EVENT THEN
            WRITE( l.c1.RIGHT.3);
            WRITE( l.filter.LEFT.0);
           ELSE
            WRITE( l.filter.LEFT.0);
            WRITE( l.c2.RIGHT.3);
           END IF;
           WRITELINE(flush.l);
          END PEOCESS writing;
END input output;
```

รูปที่ 4.38 แสดงตัวอย่างการทำงานร่วมกับเพิ่มข้อความ

0	ns	..	0
500	ns	0	..
510	ns	..	1
990	ns	..	0
1000	ns	1	..
1500	ns	0	..
1510	ns	..	1
1990	ns	..	0
2000	ns	1	..
2500	ns	0	..
2510	ns	..	1
2990	ns	..	0
3000	ns	1	..
3500	ns	0	..
3510	ns	..	1
3990	ns	..	0
4000	ns	1	..

รูปที่ 4.39 แสดงรายงานผลที่ได้จากการทดสอบและเลียนแบบ

4.6 สรุป (Summary)

ในบทนี้ได้กล่าวถึงโครงสร้างพื้นฐานและข้อกำหนดต่างๆของภาษา VHDL พอสังเขปเพื่อความเข้าใจในการศึกษาโครงสร้าง การออกแบบและการบรรยายของวงจรในบทต่อๆไป รูปแบบหลักในการบรรยายการทำงานของระบบซึ่ง VHDL ได้จัดเตรียมไว้แบ่งเป็น 3 ระดับ คือ ระดับพฤติกรรม ระดับข้อมูล และระดับโครงสร้าง

การบรรยายในระดับพฤติกรรม (Behavioral Level) ช่วยอำนวยประโยชน์ในการออกแบบเป็นขั้นตอน เช่น การออกแบบพอร์ตติดต่อ (I/O port) ที่จำเป็น ใช้อธิบายการทำงานของวงจรในระดับสูง สิ่งที่สนใจเป็นเพียงสัญญาณที่อินพุทและเอาท์พุทเท่านั้น การออกแบบในระดับนี้จะช่วยตรวจสอบและแก้ไขโครงสร้างของวงจรทั้งในเรื่องของพอร์ต จำนวนอุปกรณ์ที่จำเป็นสำหรับการนำมาเชื่อมต่อให้เป็นวงจรที่สมบูรณ์ การประสานงานของแต่ละอุปกรณ์เพื่อรวมกันเป็นระบบ

การบรรยายในระดับกระแสข้อมูล (Dataflow Level) เป็นระดับกลางที่อยู่ระหว่างระดับพฤติกรรมและระดับโครงสร้าง การออกแบบในระดับนี้จะใช้แทนการทำงานของอุปกรณ์ได้ดี ประโยชน์ที่ได้คืออธิบายการเลื่อนไหลของสัญญาณข้อมูลต่างๆระหว่างลอจิกและรีจิสเตอร์โดยผ่านตัวกลางคือบัสและช่วยในการตรวจสอบและแก้ไขการประสานงานระหวางลอจิกและรีจิสเตอร์ในลักษณะของการปฏิบัติงานพร้อมกัน (concurrent) ซึ่งเป็นการเลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์จริงๆ

การบรรยายในระดับโครงสร้าง (Structure Level) เป็นระดับล่างสุดเมื่อการออกแบบในระดับบนทั้งสองถูกต้องแล้ว การออกแบบในระดับนี้เป็นการออกแบบในระดับเกทใช้อธิบายการทำงานที่ประสานกันของเกทต่างๆ เพื่อให้ได้สัญญาณออกมาตามต้องการ เมื่อถึงขั้นนี้แล้วการตรวจสอบและแก้ไขจะเป็นเพียงการดูแลความถูกต้องในเชิงตรรกเท่านั้น

บทที่ 5

การออกแบบDCT

5.1 หลักการของ Discrete Cosine Transform

ในการที่ทำการแปลงข้อมูลแบบ Discreat Cosine Transform ในที่นี้จะทำการแปลงข้อมูลเมตริกซ์ขนาด 8 × 8 โดยมี สมการหรือรูปแบบของ DCT สามารถแสดงได้ดังนี้

$$g = DCT(f)$$

$$g = A f A^{T} \qquad \ldots(1)$$

โดยแสดงถึง discrete cosine transform ของ matrix f

รูปที่ 5.1 แสดงแผนภาพการแปลงเมตริกซ์

โดย A เป็น matrix ซึ่งมีค่าดังนี้

$$a_{jk} = \begin{cases} \dfrac{1}{\sqrt{N}} & ; \ k = 0 \\ \sqrt{\dfrac{2}{N}}\left[\cos\left[\dfrac{(2j+1)k\pi}{2N}\right]\right] & ; k = 1, 2, \ldots, N\text{-}1 \end{cases}$$

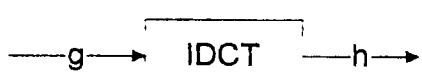
$$j = 0, 1, \ldots, N\text{-}1$$

และ A^{T} คือ transpose ของ matrix

และ inverse discrete cosine transform ของ g แสดงได้ดังสมการที่(2)

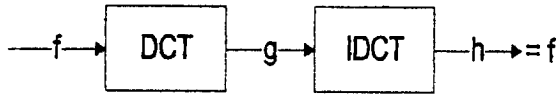
$$h = IDCT(g)$$

$$h = A^{T} g A \qquad \ldots(2)$$



รูปที่ 5.2 แสดงแผนภาพการแปลงกลับเมตริกซ์

เมื่อ f saturated จะได้



รูปที่ 5.3 แสดงแผนภาพการแปลงกลับเมตริกซ์

ในการแปลง matrix ขนาด 8 ×8 (N = 8)จะได้ค่า A ดังนี้

$$A = \begin{bmatrix} 0.354 & 0.49 & 0.462 & 0.416 & 0.354 & 0.278 & 0.191 & 0.098 \\ 0.354 & 0.416 & 0.191 & -0.098 & -0.354 & -0.49 & -0.462 & -0.278 \\ 0.354 & 0.278 & -0.191 & -0.49 & -0.354 & 0.098 & 0.462 & 0.416 \\ 0.354 & 0.098 & -0.462 & -0.278 & 0.354 & 0.416 & -0.191 & -0.49 \\ 0.354 & -0.098 & -0.462 & 0.278 & 0.354 & -0.416 & -0.191 & 0.49 \\ 0.354 & -0.278 & -0.191 & 0.49 & -0.354 & -0.098 & 0.462 & -0.416 \\ 0.354 & -0.416 & 0.191 & 0.098 & -0.354 & 0.49 & -0.462 & 0.278 \\ 0.354 & -0.49 & 0.462 & -0.416 & 0.354 & -0.278 & 0.191 & -0.098 \end{bmatrix}$$

โดยมีผลการทดลองดังนี้

f คือข้อมูลภาพ

$$f1 := \begin{bmatrix} 97 & 97 & 97 & 102 & 102 & 102 & 102 & 102 \\ 88 & 88 & 97 & 97 & 97 & 97 & 102 & 97 \\ 88 & 88 & 97 & 88 & 88 & 97 & 88 & 88 \\ 88 & 88 & 88 & 88 & 88 & 88 & 88 & 88 \\ 80 & 92 & 88 & 88 & 88 & 88 & 88 & 88 \\ 80 & 88 & 83 & 88 & 82 & 88 & 88 & 88 \\ 80 & 83 & 80 & 86 & 88 & 86 & 86 & 86 \\ 80 & 80 & 83 & 83 & 80 & 80 & 83 & 88 \end{bmatrix}$$

$$f2 := \begin{bmatrix} 50 & 97 & 97 & 102 & 102 & 102 & 102 & 102 \\ 50 & 88 & 97 & 97 & 97 & 97 & 102 & 97 \\ 50 & 88 & 97 & 88 & 88 & 97 & 88 & 88 \\ 50 & 88 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 92 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 88 & 83 & 88 & 82 & 88 & 88 & 88 \\ 50 & 83 & 80 & 86 & 88 & 86 & 86 & 86 \\ 50 & 80 & 83 & 83 & 80 & 80 & 83 & 88 \end{bmatrix}$$

$$f3 := \begin{bmatrix} 50 & 50 & 97 & 102 & 102 & 102 & 102 & 102 \\ 50 & 50 & 97 & 97 & 97 & 97 & 102 & 97 \\ 50 & 50 & 97 & 88 & 88 & 97 & 88 & 88 \\ 50 & 50 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 50 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 50 & 83 & 88 & 82 & 88 & 88 & 88 \\ 50 & 50 & 80 & 86 & 88 & 86 & 86 & 86 \\ 50 & 50 & 83 & 83 & 80 & 80 & 83 & 88 \end{bmatrix}$$

g คือ discrete cosine transform ของ matrix f จะได้

$$g1 := A \cdot f1 \cdot A^T$$

$$g1 = \begin{bmatrix} 630.341 & -183.1 & 133.915 & -53.075 & 68.138 & -9.024 & 42.518 & 16.025 \\ -134.673 & 35.677 & -30.899 & 11.755 & -17.532 & 3.886 & -5.842 & 3.108 \\ 134.481 & -45.825 & 29.573 & -13.742 & 20.283 & 0.138 & 8.577 & 3.64 \\ -30.204 & 5.62 & -6.366 & 2.372 & -2.562 & 1.176 & -7.182 & -5.883 \\ 74.19 & -19.866 & 16.842 & -8.131 & 13.898 & -1.432 & 2.538 & 2.443 \\ 0.831 & -0.663 & -0.107 & 5.453 & 4.457 & 1.594 & -0.739 & -0.15 \\ 43.659 & -8.6 & 12.061 & -3.921 & 4.912 & 0.175 & 6.995 & -0.56 \\ 25.078 & -4.716 & 3.57 & 1.433 & 0.385 & -3.554 & 3.064 & -0.45 \end{bmatrix}$$

$$g2 := A \cdot f2 \cdot A^T$$

$$g2 = \begin{bmatrix} 596.326 & -217.116 & 99.899 & -87.09 & 34.122 & -43.04 & 8.502 & -17.99 \\ -130.597 & 39.754 & -26.822 & 15.831 & -13.455 & 7.963 & -1.765 & 7.185 \\ 127.439 & -52.867 & 22.531 & -20.784 & 13.241 & -6.904 & 1.535 & -3.402 \\ -28.38 & 7.444 & -4.542 & 4.197 & -0.738 & 3 & -5.358 & -4.058 \\ 69.587 & -24.469 & 12.24 & -12.734 & 9.295 & -6.035 & -2.065 & -2.16 \\ -1.035 & -2.529 & -1.973 & 3.586 & 2.591 & -0.272 & -2.605 & -2.016 \\ 39.99 & -12.269 & 8.392 & -7.59 & 1.243 & -3.494 & 3.326 & -4.229 \\ 23.373 & -6.422 & 1.864 & -0.273 & -1.321 & -5.26 & 1.358 & -2.156 \end{bmatrix}$$

$$g3 := A \cdot f3 \cdot A^{T}$$

$$g3 = \begin{bmatrix} 545.693 & -260.04 & 71.218 & -97.161 & 44.194 & -14.359 & 51.426 & 32.642 \\ -120.258 & 48.518 & -20.966 & 17.888 & -15.511 & 2.106 & -10.53 & -3.154 \\ 118.69 & -60.284 & 17.575 & -22.524 & 14.982 & -1.948 & 8.952 & 5.347 \\ -29.28 & 6.681 & -5.052 & 4.018 & -0.559 & 3.51 & -4.595 & -3.158 \\ 63.035 & -30.023 & 8.528 & -14.037 & 10.598 & -2.324 & 3.49 & 4.393 \\ -1.84 & -3.211 & -2.429 & 3.426 & 2.751 & 0.184 & -1.923 & -1.211 \\ 35.738 & -15.874 & 5.983 & -8.436 & 2.089 & -1.085 & 6.931 & 0.024 \\ 19.734 & -9.507 & -0.197 & -0.997 & -0.597 & -3.199 & 4.442 & 1.482 \end{bmatrix}$$

h คือ inverse discrete cosine transform ของ g จะได้

$$h1 := A^{T} \cdot g1 \cdot A$$

$$h1 = \begin{bmatrix} 97 & 97 & 97 & 102 & 102 & 102 & 102 & 102 \\ 88 & 88 & 97 & 97 & 97 & 97 & 102 & 97 \\ 88 & 88 & 97 & 88 & 88 & 97 & 88 & 88 \\ 88 & 88 & 88 & 88 & 88 & 88 & 88 & 88 \\ 80 & 92 & 88 & 88 & 88 & 88 & 88 & 88 \\ 80 & 88 & 83 & 88 & 82 & 88 & 88 & 88 \\ 80 & 83 & 80 & 86 & 88 & 86 & 86 & 86 \\ 80 & 80 & 83 & 83 & 80 & 80 & 83 & 88 \end{bmatrix}$$

$$h2 := A^{T} \cdot g2 \cdot A$$

$$h2 = \begin{bmatrix} 50 & 97 & 97 & 102 & 102 & 102 & 102 & 102 \\ 50 & 88 & 97 & 97 & 97 & 97 & 102 & 97 \\ 50 & 88 & 97 & 88 & 88 & 97 & 88 & 88 \\ 50 & 88 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 92 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 88 & 83 & 88 & 82 & 88 & 88 & 88 \\ 50 & 83 & 80 & 86 & 88 & 86 & 86 & 86 \\ 50 & 80 & 83 & 83 & 80 & 80 & 83 & 88 \end{bmatrix}$$

$$h3 := A^{T} \cdot g3 \cdot A$$

$$h3 = \begin{bmatrix} 50 & 50 & 97 & 102 & 102 & 102 & 102 & 102 \\ 50 & 50 & 97 & 97 & 97 & 97 & 102 & 97 \\ 50 & 50 & 97 & 88 & 88 & 97 & 88 & 88 \\ 50 & 50 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 50 & 88 & 88 & 88 & 88 & 88 & 88 \\ 50 & 50 & 83 & 88 & 82 & 88 & 88 & 88 \\ 50 & 50 & 80 & 86 & 88 & 86 & 86 & 86 \\ 50 & 50 & 83 & 83 & 80 & 80 & 83 & 88 \end{bmatrix}$$

เมื่อนำไปใช้สร้างจริงค่าของ A ซึ่งเป็นจุดทศนิยมต้องทำการคูณด้วยค่า 1024 เพื่อทำให้มีส่วนของจำนวนเต็มเพื่อสะดวกที่จะนำไปทำการคูณ ในงานที่เป็นดิจิตอล และสามารถที่จะหารกลับได้ง่าย โดยจะได้ค่าของ A × 1024 ซึ่งเป็นค่า Coeff ที่จะนำไปใช้จริงในวงจร ดังนี้

$$coeff := \begin{bmatrix} 362 & 502 & 473 & 426 & 362 & 284 & 196 & 100 \\ 362 & 426 & 196 & -100 & -362 & -502 & -473 & -284 \\ 362 & 284 & -196 & -502 & -362 & 100 & 473 & 426 \\ 362 & 100 & -473 & -284 & 362 & 426 & -196 & -502 \\ 362 & -100 & -473 & 284 & 362 & -426 & -196 & 502 \\ 362 & -284 & -196 & 502 & -362 & -100 & 473 & -426 \\ 362 & -426 & 196 & 100 & -362 & 502 & -473 & 284 \\ 362 & -502 & 473 & -426 & 362 & -284 & 196 & -100 \end{bmatrix}$$

ซึ่งจะได้ discrete cosine transform ของ matrix f (ซึ่งค่าที่ส่งออกในวงจรจริงจะส่งเฉพาะค่าที่เป็นจำนวนเต็ม) ดังนี้

$$gg1 := \frac{coeff \cdot f1 \cdot coeff^{T}}{1024 \cdot 1024}$$

$$gg1 = \begin{bmatrix} 630.222 & -182.874 & 133.909 & -52.856 & 67.866 & -9.074 & 42.352 & 16.069 \\ -134.466 & 35.573 & -30.856 & 11.689 & -17.452 & 3.891 & -5.791 & 3.104 \\ 134.461 & -45.773 & 29.578 & -13.695 & 20.225 & 0.123 & 8.541 & 3.645 \\ -29.999 & 5.554 & -6.323 & 2.346 & -2.528 & 1.178 & -7.161 & -5.88 \\ 73.931 & -19.768 & 16.788 & -8.085 & 13.837 & -1.435 & 2.502 & 2.44 \\ 0.796 & -0.658 & -0.115 & 5.455 & 4.451 & 1.596 & -0.743 & -0.149 \\ 43.5 & -8.538 & 12.027 & -3.889 & 4.874 & 0.173 & 6.972 & -0.561 \\ 25.113 & -4.722 & 3.578 & 1.437 & 0.379 & -3.555 & 3.059 & -0.448 \end{bmatrix}$$

$$gg2 := \frac{coeff \cdot f2 \cdot coeff^{T}}{1024 \cdot 1024}$$

$$gg2 = \begin{bmatrix} 596.213 & -216.883 & 99.9 & -86.865 & 33.856 & -43.084 & 8.343 & -17.941 \\ -130.4 & 39.64 & -26.789 & 15.756 & -13.385 & 7.958 & -1.725 & 7.171 \\ 127.421 & -52.813 & 22.538 & -20.735 & 13.185 & -6.917 & -1.501 & -3.395 \\ -28.186 & 7.367 & -4.51 & 4.159 & -0.715 & 2.991 & -5.348 & -4.067 \\ 69.342 & -24.357 & 12.199 & -12.673 & 9.248 & -6.024 & -2.087 & -2.149 \\ -1.068 & -2.522 & -1.98 & 3.591 & 2.587 & -0.268 & -2.607 & -2.013 \\ 39.839 & -12.199 & 8.366 & -7.55 & 1.214 & -3.488 & 3.311 & -4.222 \\ 23.406 & -6.429 & 1.871 & -0.27 & -1.328 & -5.261 & 1.352 & -2.155 \end{bmatrix}$$

$$gg3 := \frac{coeff \cdot f3 \cdot coeff^{T}}{1024 \cdot 1024}$$

$$gg3 = \begin{bmatrix} 545.602 & -259.832 & 71.268 & -96.947 & 43.938 & -14.451 & 51.291 & 32.67 \\ -120.078 & 48.398 & -20.95 & 17.812 & -15.441 & 2.119 & -10.483 & -3.15 \\ 118.675 & -60.235 & 17.59 & -22.477 & 14.927 & -1.969 & 8.922 & 5.35 \\ -29.102 & 6.59 & -5.028 & 3.977 & -0.533 & 3.509 & -4.571 & -3.151 \\ 62.812 & -29.899 & 8.505 & -13.974 & 10.549 & -2.33 & 3.454 & 4.381 \\ -1.87 & -3.203 & -2.433 & 3.431 & 2.747 & 0.185 & -1.926 & -1.212 \\ 35.599 & -15.797 & 5.967 & -8.395 & 2.058 & -1.089 & 6.909 & 0.019 \\ 19.766 & -9.518 & -0.189 & -0.995 & -0.603 & -3.202 & 4.441 & 1.486 \end{bmatrix}$$

และ ให้ hh เป็น inverse discrete cosine transform ของ gg (ซึ่งค่าที่ส่งออกในวงจรจริงจะส่งเฉพาะค่าที่เป็นจำนวนเต็ม) จะได้

$$hh1 := \frac{coeff^{T} \cdot gg1 \cdot coeff}{1024 \cdot 1024}$$

$$hh1 = \begin{bmatrix} 96.959 & 96.951 & 96.98 & 101.946 & 101.956 & 101.841 & 101.979 & 101.838 \\ 87.96 & 87.949 & 96.975 & 96.939 & 96.954 & 96.844 & 101.971 & 96.839 \\ 87.982 & 87.97 & 97.001 & 87.975 & 87.982 & 96.877 & 88.001 & 87.875 \\ 87.96 & 87.953 & 87.982 & 87.957 & 87.965 & 87.863 & 87.985 & 87.859 \\ 79.966 & 91.954 & 87.982 & 87.958 & 87.962 & 87.858 & 87.982 & 87.86 \\ 79.869 & 87.858 & 82.879 & 87.852 & 81.863 & 87.758 & 87.875 & 87.754 \\ 79.983 & 82.976 & 80.001 & 85.974 & 87.982 & 85.884 & 86.001 & 85.882 \\ 79.869 & 79.86 & 82.884 & 82.852 & 79.868 & 79.767 & 82.881 & 87.766 \end{bmatrix}$$

$$hh2 := \frac{coeff^{T} \cdot gg2 \cdot coeff}{1024 \cdot 1024}$$

$$hh2 = \begin{bmatrix} 49.979 & 96.951 & 96.98 & 101.946 & 101.956 & 101.841 & 101.979 & 101.838 \\ 49.975 & 87.949 & 96.975 & 96.939 & 96.954 & 96.844 & 101.971 & 96.839 \\ 49.99 & 87.97 & 97.001 & 87.975 & 87.982 & 96.877 & 88.001 & 87.875 \\ 49.975 & 87.953 & 87.982 & 87.957 & 87.965 & 87.863 & 87.985 & 87.859 \\ 49.979 & 91.954 & 87.982 & 87.958 & 87.962 & 87.858 & 87.982 & 87.86 \\ 49.921 & 87.858 & 82.879 & 87.852 & 81.863 & 87.758 & 87.875 & 87.754 \\ 49.99 & 82.976 & 80.001 & 85.974 & 87.982 & 85.884 & 86.001 & 85.882 \\ 49.921 & 79.86 & 82.884 & 82.852 & 79.868 & 79.767 & 82.881 & 87.766 \end{bmatrix}$$

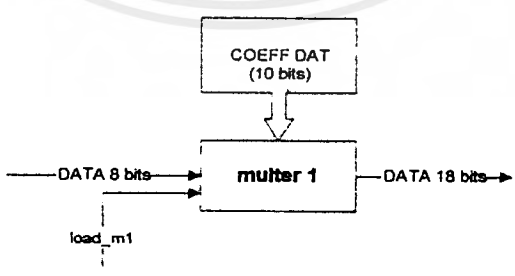
$$hh3 := \frac{coeff^{T} \cdot gg3 \cdot coeff}{1024 \cdot 1024}$$

$$hh3 = \begin{bmatrix} 49.979 & 49.975 & 96.98 & 101.92 & 101.956 & 101.866 & 101.979 & 101.838 \\ 49.975 & 49.971 & 96.975 & 96.918 & 96.954 & 96.864 & 101.971 & 96.839 \\ 49.99 & 49.981 & 97.001 & 87.954 & 87.982 & 96.897 & 88.001 & 87.875 \\ 49.975 & 49.971 & 87.982 & 87.936 & 87.965 & 87.883 & 87.985 & 87.859 \\ 49.979 & 49.975 & 87.982 & 87.935 & 87.962 & 87.881 & 87.982 & 87.86 \\ 49.921 & 49.917 & 82.879 & 87.831 & 81.863 & 87.778 & 87.875 & 87.754 \\ 49.99 & 49.986 & 80.001 & 85.956 & 87.982 & 85.902 & 86.001 & 85.882 \\ 49.921 & 49.918 & 82.884 & 82.836 & 79.868 & 79.784 & 82.881 & 87.766 \end{bmatrix}$$

โดยจะสังเกตได้ว่าค่าที่ออกมาจะมีค่าน้อยกว่า image f อยู่ในระดับจุดทศนิยม และเมื่อส่งออกเฉพาะค่าของจำนวนเต็ม ค่าที่ออกมาส่วนมากก็จะมีค่าน้อยกว่า image f อยู่ประมาณ 1 เนื่องจากการปัดทศนิยมของ cosine ในค่าของ A (หรือ coeff)ที่ใช้ในการ transform

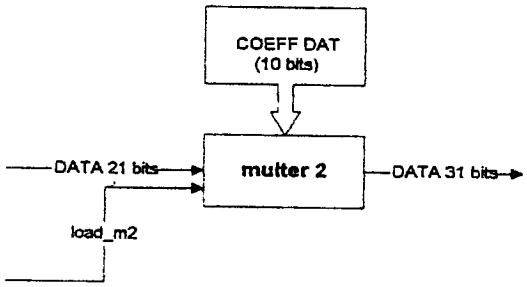
5.2 การออกแบบสร้างเป็นอุปกรณ์

ในการสร้างขั้นแรกต้องทำการสร้างวงจรคูณ ซึ่งเป็นการคูณของเลขฐานสองของอินพุทที่เข้ามากับค่าของ coeff



รูปที่ 5.4 วงจรคูณ multer1

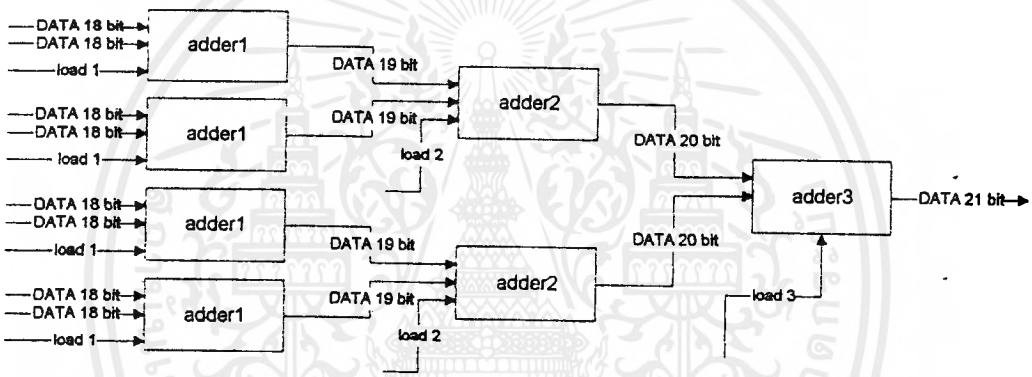
โดยวงจร multer 1 ในรูปที่ 5.4 เป็นการคูณของอินพุทขนาด 8 bits กับcoeff ขนาด 10 bits ซึ่งจะได้เอาท์พุทออกมาขนาด 18 bits โดยมีสัญญาน load_ m1 เป็นสัญญาณ enable



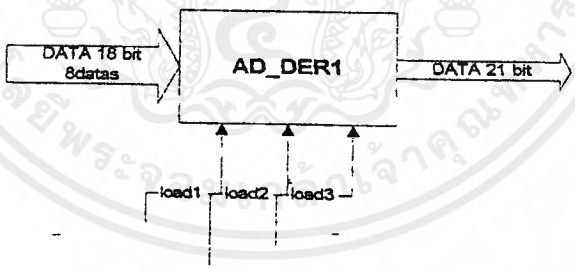
รูปที่ 5.5 วงจรกูณ multer2

ในรูปที่ 5.5 วงจร multer2 ก็มีลักษณะเช่นเดียวกับ multer 1 ต่างกันที่ขนาดของอินพุทและเอาท์พุท

ต่อมาก็เป็นวงจรบวกเลขฐานสอง ดังรูปที่ 5.6 วงจร AD_DER1 เป็นวงจรบวกเลขขนาด 18 bits 8 จำนวน โดยประกอบไปด้วยอุปกรณ์ adder1,adder2 และ adder3 ซึ่งมีสัญญาน load1,load2 เป็นสัญญาณที่ควบคุมการทำงาน



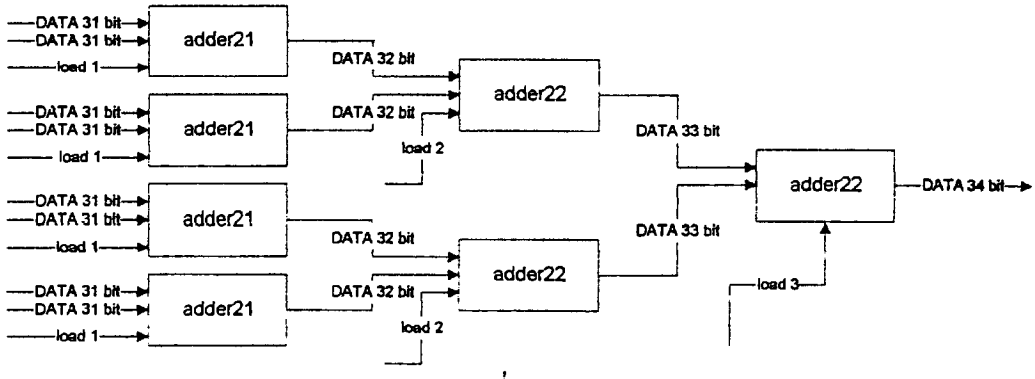
รูปที่ 5.6 (a)



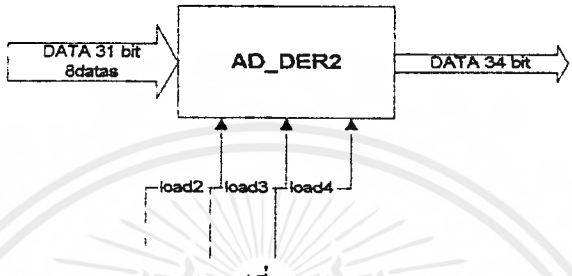
รูปที่ 5.6 (b)

รูปที่ 5.6 วงจรบวก AD_DER1

ต่อมารูปที่ 5.7 วงจร AD_DER2 เป็นวงจรบวกเลขฐานสอง โดยบวกเลขขนาด 31 bits 8 จำนวน โดยประกอบไปด้วยอุปกรณ์ adder21,adder22 และ adder23 ซึ่งมีสัญญาน load1,load2 เป็นสัญญาณที่ควบคุมการทำงาน



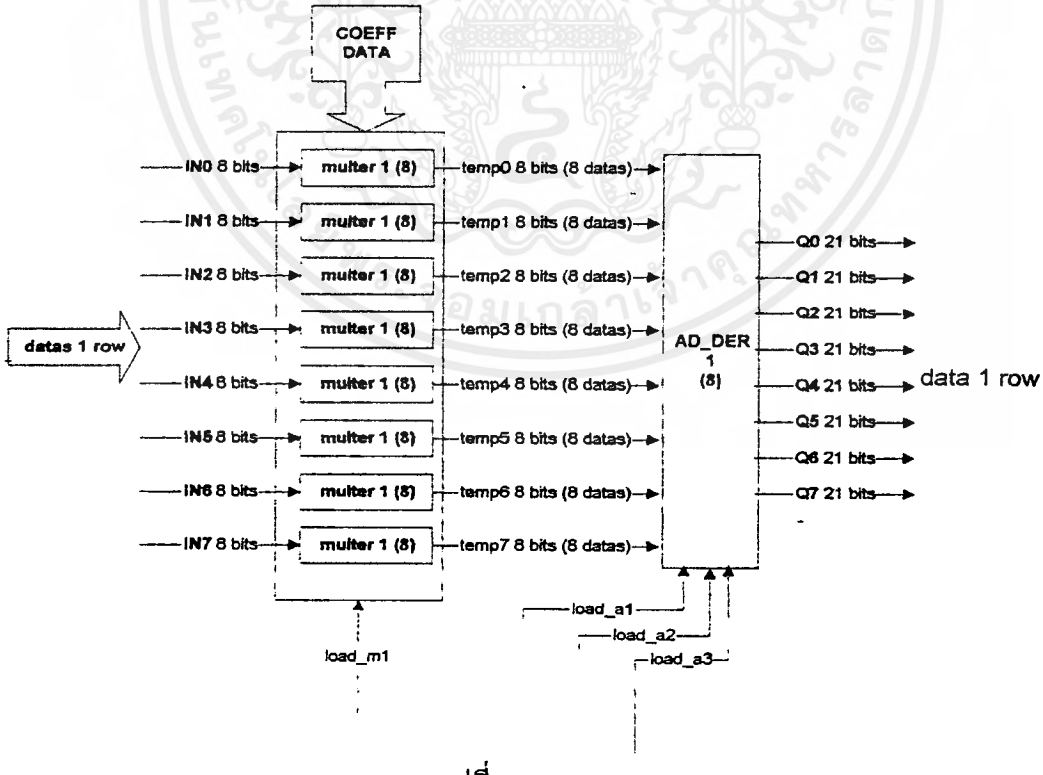
รูปที่ 5.7(a)



รูปที่ 5.7 (b)

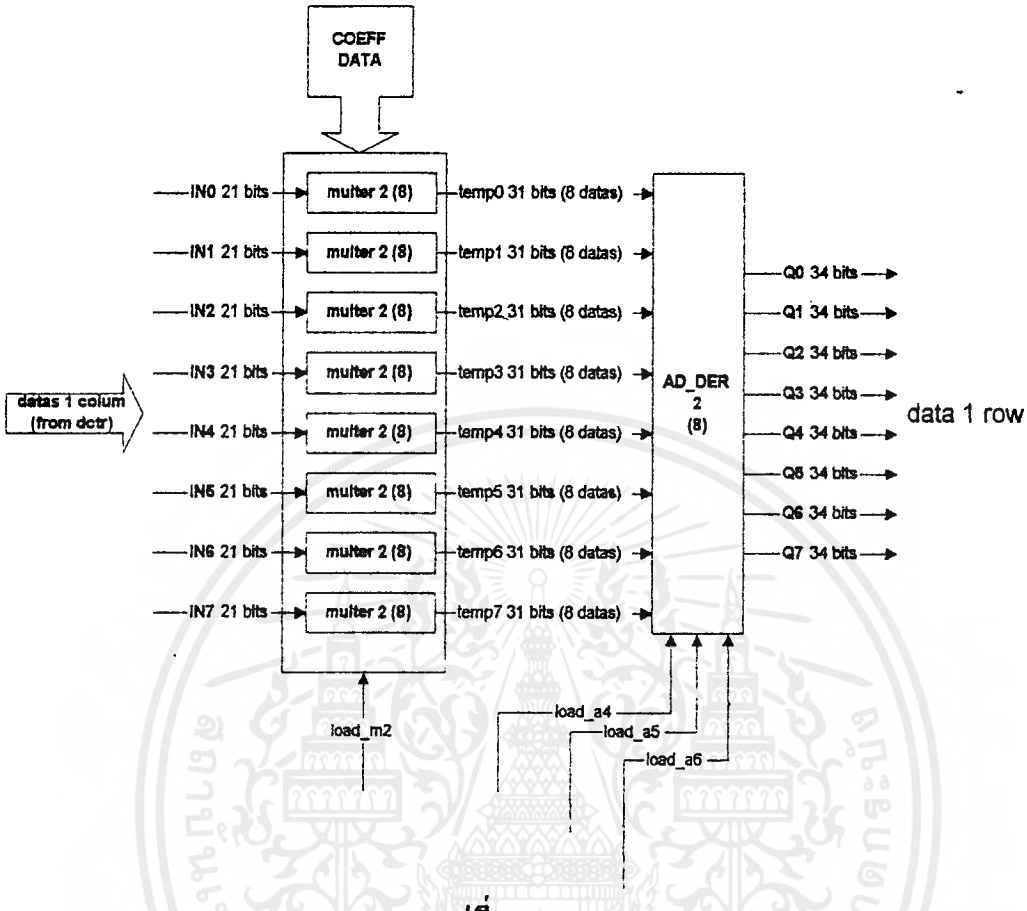
รูปที่ 5.7 วงจรบวก AD_DER2

ต่อมารูปที่ 5.8 วงจร DCTR เป็นวงจรที่นำเอา multer1 และ AD_DER1 มาต่อกันเพื่อสร้างเป็นอุปกรณ์ที่จะนำไปใช้ต่อไป



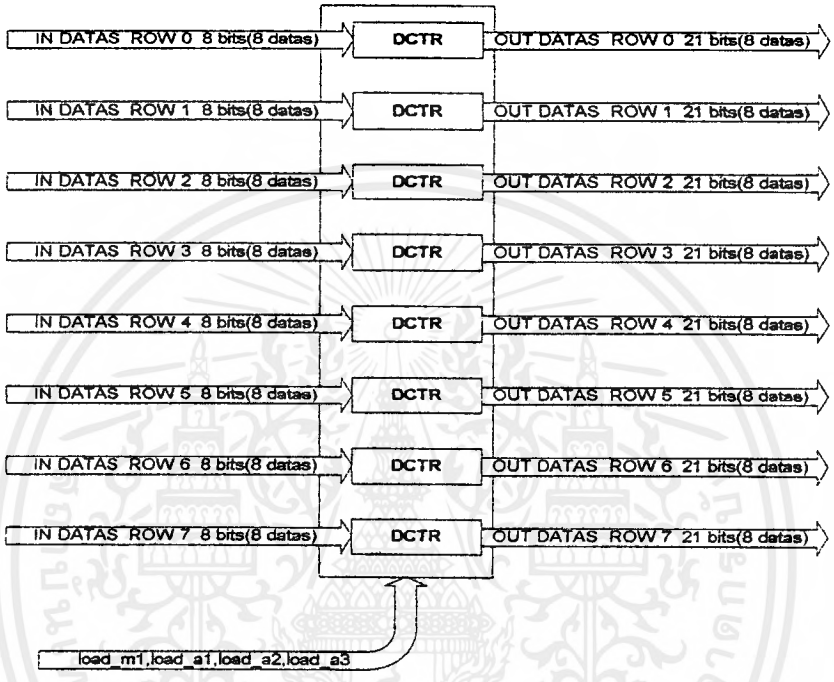
รูปที่ 5.8 วงจร DCTR

ต่อมารูปที่ 5.9 วงจร DCTC เป็นวงจรที่นำเอา multer2 และ AD_DER2 มาต่อกันเพื่อสร้างเป็นอุปกรณ์ที่จะนำไปใช้ต่อไป

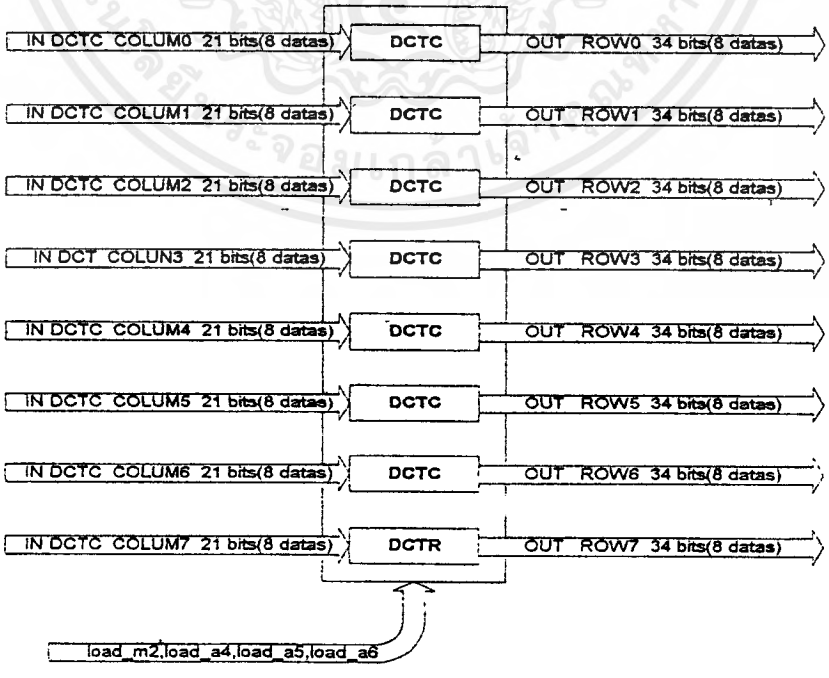


รูปที่ 5.9วงจร DCTC

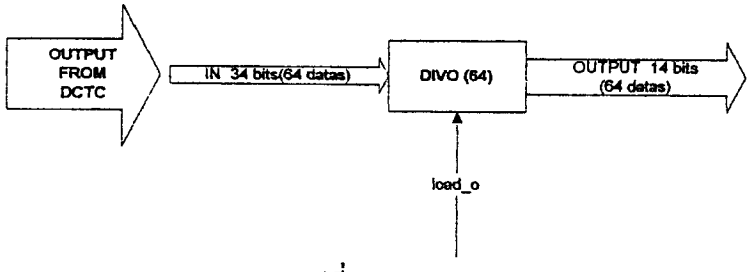
รูปที่ 5.10 เป็นวงจรรวมDCT โดย รูปที่5.10 (a) เป็นการนำ ค่า image f คูณกับค่า coeff ^{T} (f.coeff^{T}) ส่วนรูปที่5.10 (b) เป็นการนำ ค่า coeff คูณกับค่า f.coeff^{T} ที่ได้มา (coeff.[f.coeff^{T}]) ส่วนรูปที่5.10 (c) เป็นการนำค่าที่ได้มาหารด้วย 1024×1024 แล้วส่งออกไปก็จะได้ค่า discrete cosine transform ของ matrix f



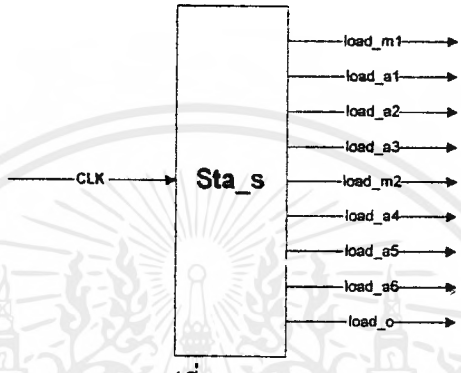
รูปที่ 5.10 (a)



รูปที่ 5.10 (b)



รูปที่ 5.10 (c)



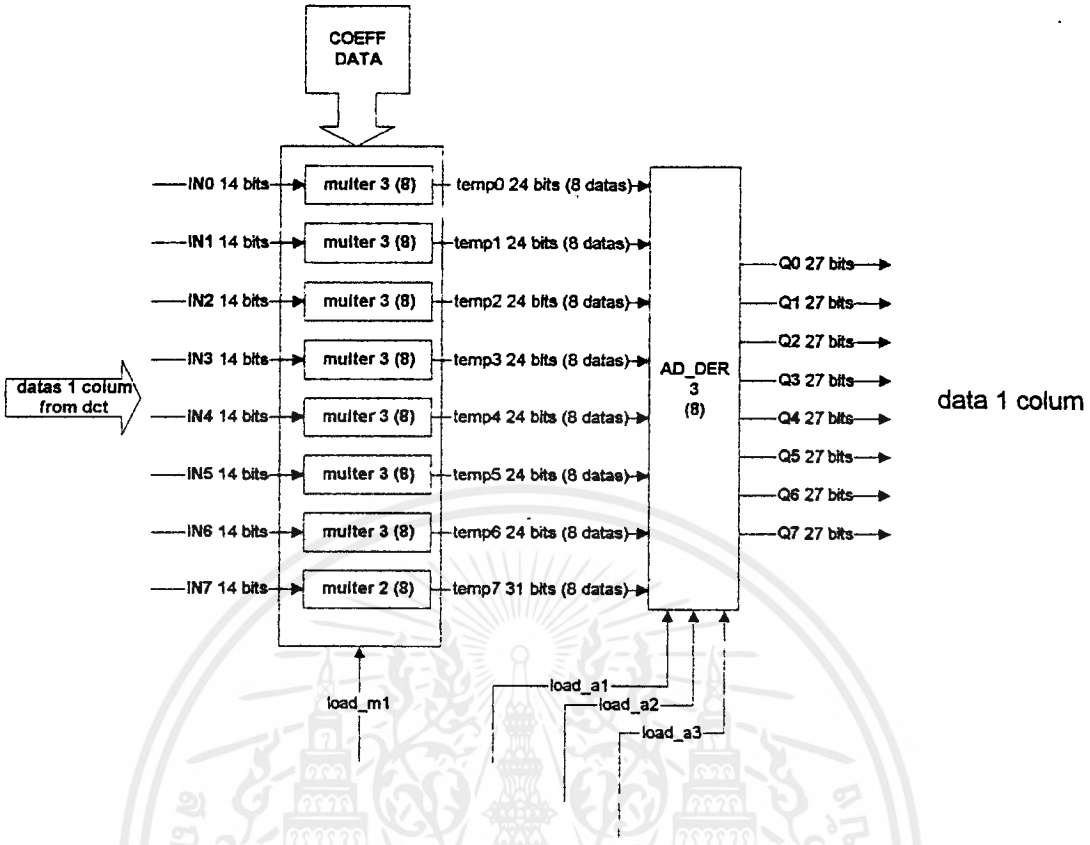
รูปที่ 5.10 (d)

รูปที่ 5.10 วงจรรวม DCT

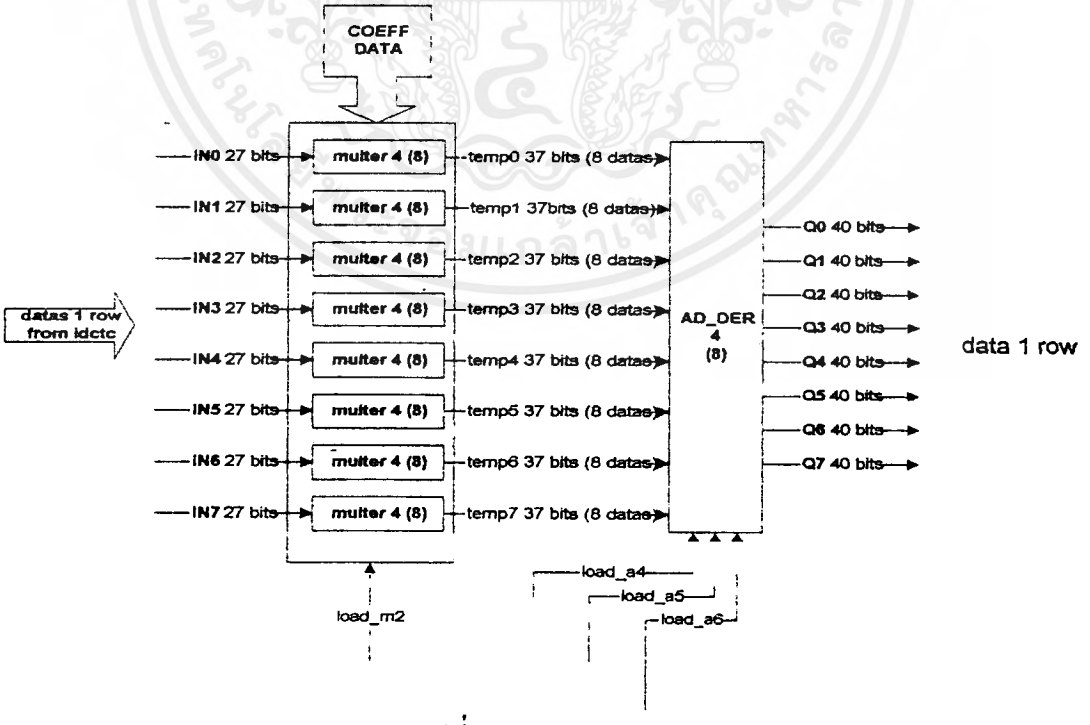
ส่วนในรูปที่ 5.10 (d) เป็นวงจร state machine ที่ใช้ควบคุมการทำงานของวงจรโดยจะส่งสัญญาณเป็นลำดับ คือ load_m1 , load_a1 , loa_a3 , load_m2 , load_a4 , load_a5 , load_a6 และ load_o ตามลำดับ ให้การควบคุมการคูณ การบวก การหารและส่งสัญญาณออก โดยแต่ละสัญญาณจะใช้เวลา 1 cycle ของ สัญญาณ clock ที่ส่งเข้ามา ดังนั้นในการทำงาน 1 ครั้งจะใช้เวลา clock 9 cycle

ซึ่งเวลาของ clock ที่ใช้ใน 1 cycle นั้น ดูจากค่า delay หรือ ค่า critical pathที่ใช้ในอุปกรณ์แต่ละตัวว่าตัวใดมีค่ามากที่สุด ควรจะให้เวลาของ clock ที่ใช้ใน 1 cycle มีค่ามากกว่า critical path นั้น

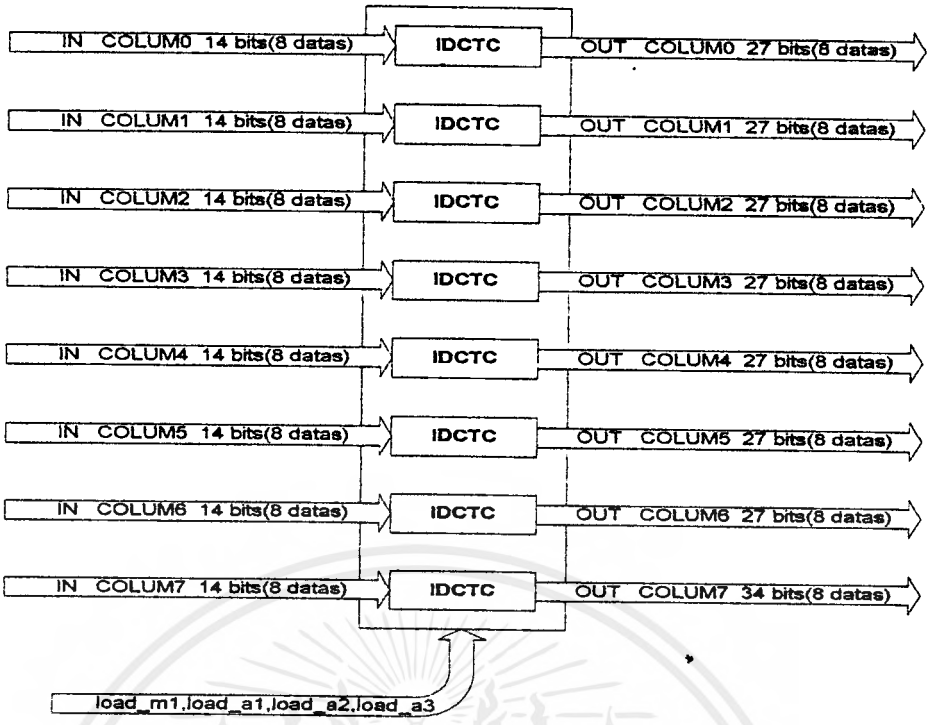
ในส่วนต่อมาจะเป็นการแสดงวงจร IDCT ซึ่งเป็นการ inverse transform ซึ่งมีอุปกรณ์ย่อยคล้ายกับของ DCT ต่างกันที่ขนาดของ bit ที่ใช้ และการเรียงของขาสัญญาณดังรูปที่ 5.11 และ 5.12 เป็นวงจร IDCTC และ IDCTR ตามลำดับซึ่งเป็นวงจรย่อยใน IDCT ในรูปที่ 5.13



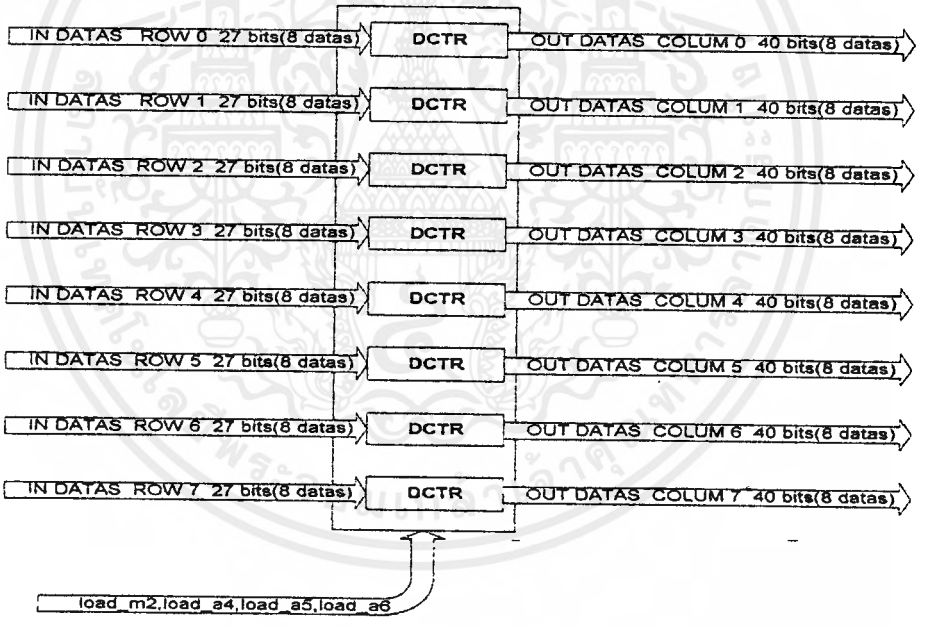
รูปที่ 5.11 วงจร IDCTC



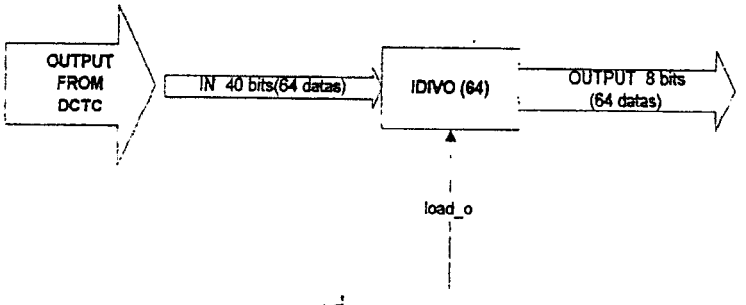
รูปที่ 5.12 วงจร IDCTR



รูปที่ 5.13 (a)

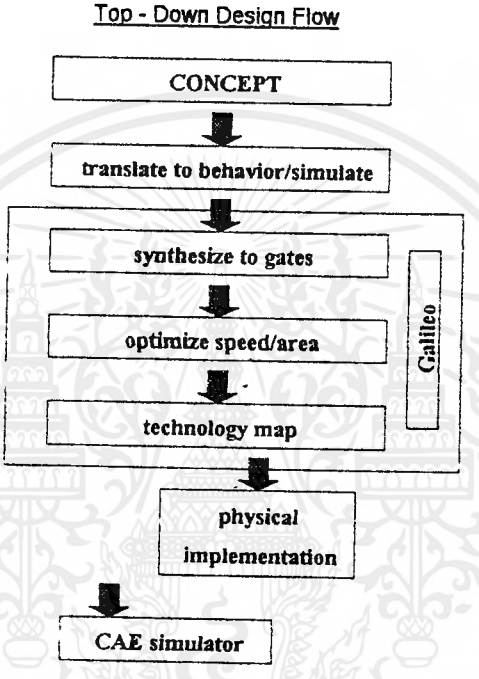


รูปที่ 5.13 (b)



รูปที่ 5.13 (c)

รูปที่ 5.13 เป็นวงจรรวมIDCT โดย รูปที่5.13 (a) เป็นการนำ ค่าcoeff^T คูณกับค่า discrete cosine transform g ส่วนรูปที่5.13 (b) เป็นการนำ ค่า coeff^T. g coeff ที่ได้มาคูณกับค่า coeff (coeff^T.g.coeff) ส่วนรูปที่5.13(c) เป็นการนำค่าที่ได้มาหารด้วย 1024×1024 แล้วส่งออกไปก็จะได้ค่า inverse discrete cosine transform ของ matrix g ส่วนของ state machine นั้นจะเหมือนกับวงจร DCT

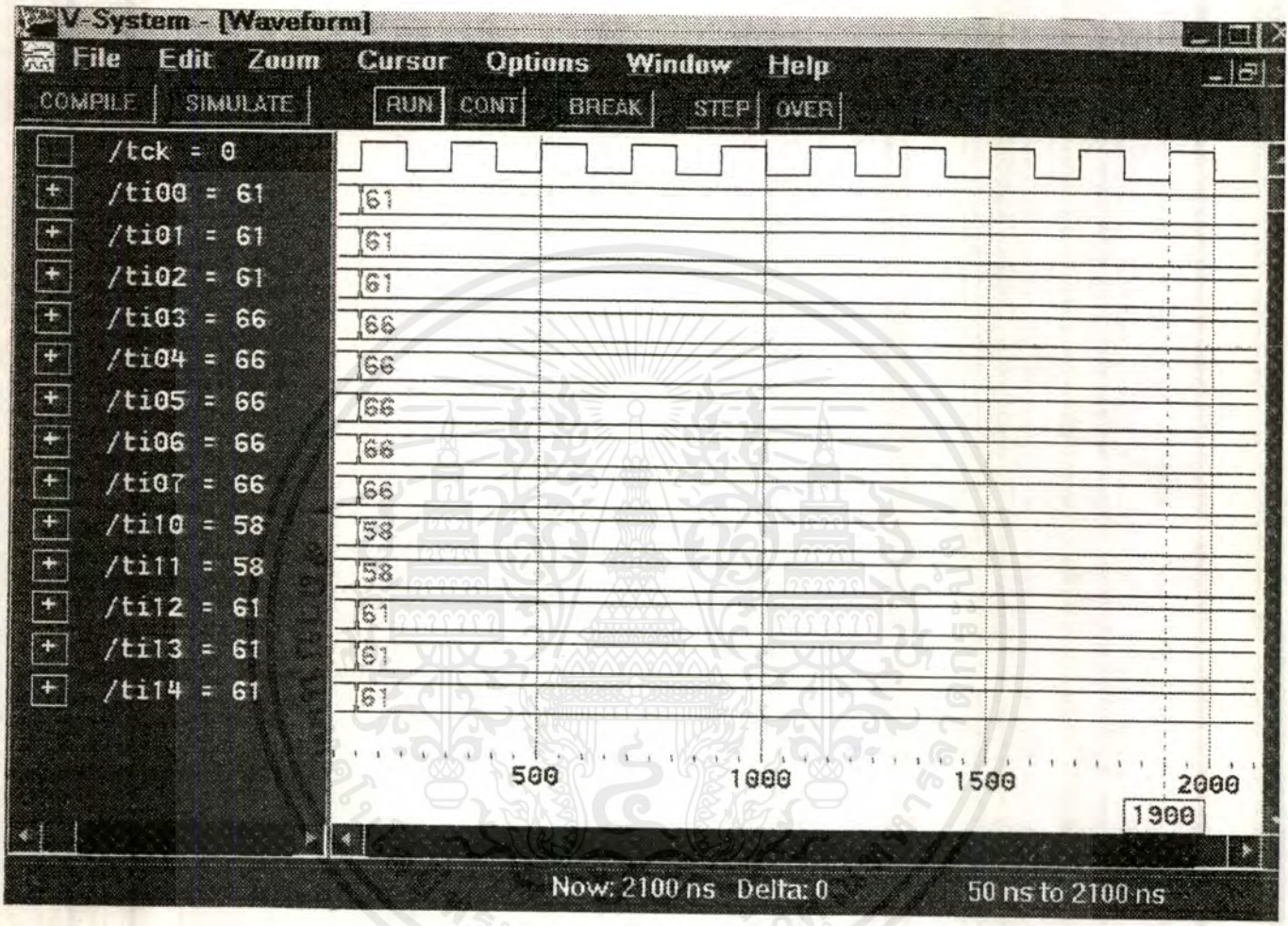


รูปที่ 5.14 ขั้นตอนการสร้างอุปกรณ์

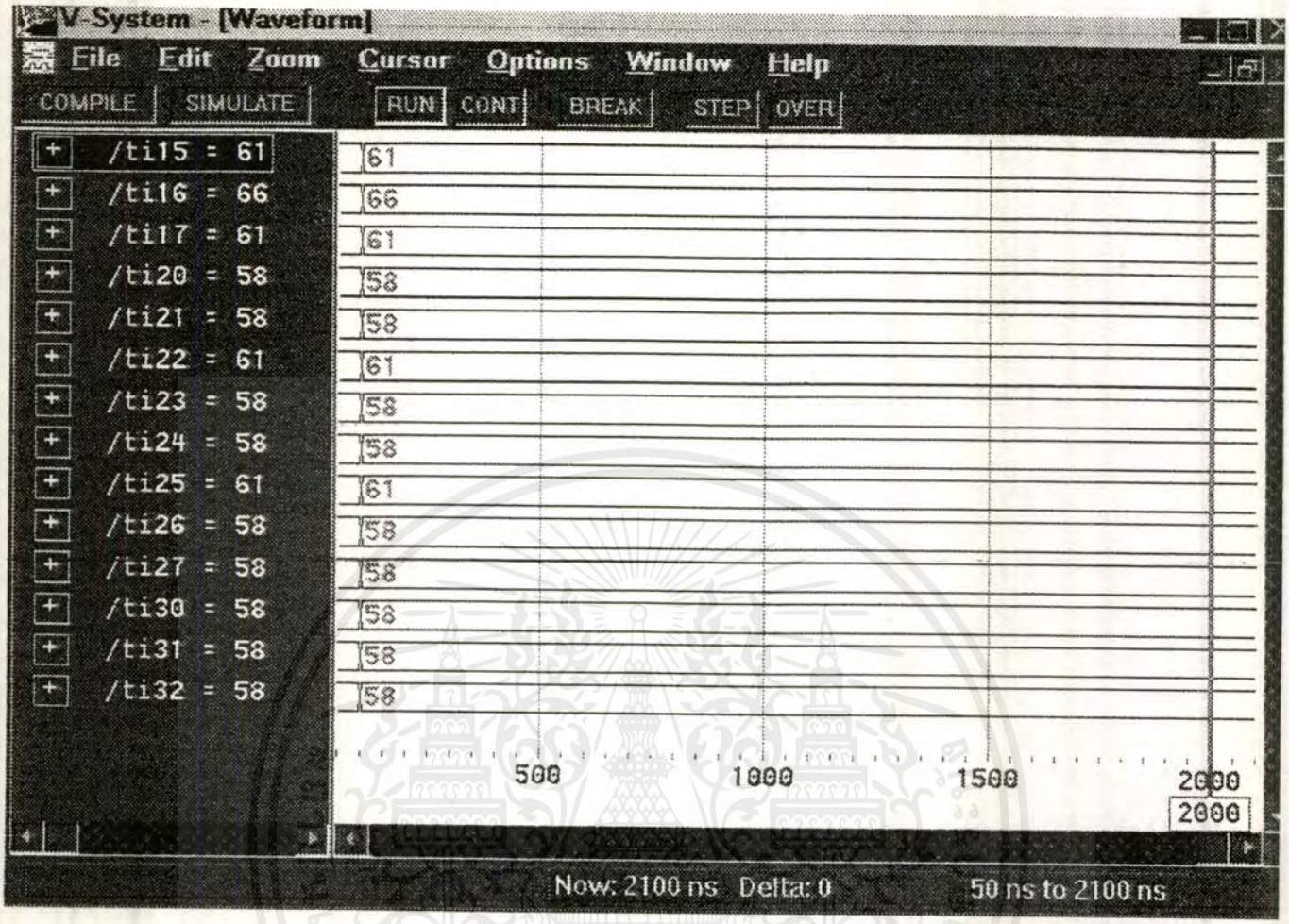
รูปที่ 5.14 จะแสดงขั้นตอนของการที่เมื่อได้หลักการของการทำงานของอุปกรณ์แล้ว ก็ทำการเขียนโปรแกมการทำงานซึ่งในที่นี้เราใช้โปรแกมภาษา VHDL จากนั้นก็ทำการทดสอบการทำงานโดยการเขียนโปรแกม test bench ทดสอบ จากนั้นก็ทำการสังเคราะห์สร้างเป็น gate ขนาดของพื้นที่และความเร็ว และวางโครงข่ายของ technology ที่ใช้ โดยใช้ software ที่ชื่อว่า galileo ซึ่งหลังจากนั้นจะนำไปสร้างเป็นแบบทางกายภาพและทดสอบต่อไป

ผลจากการทดสอบ(Simulate Result)

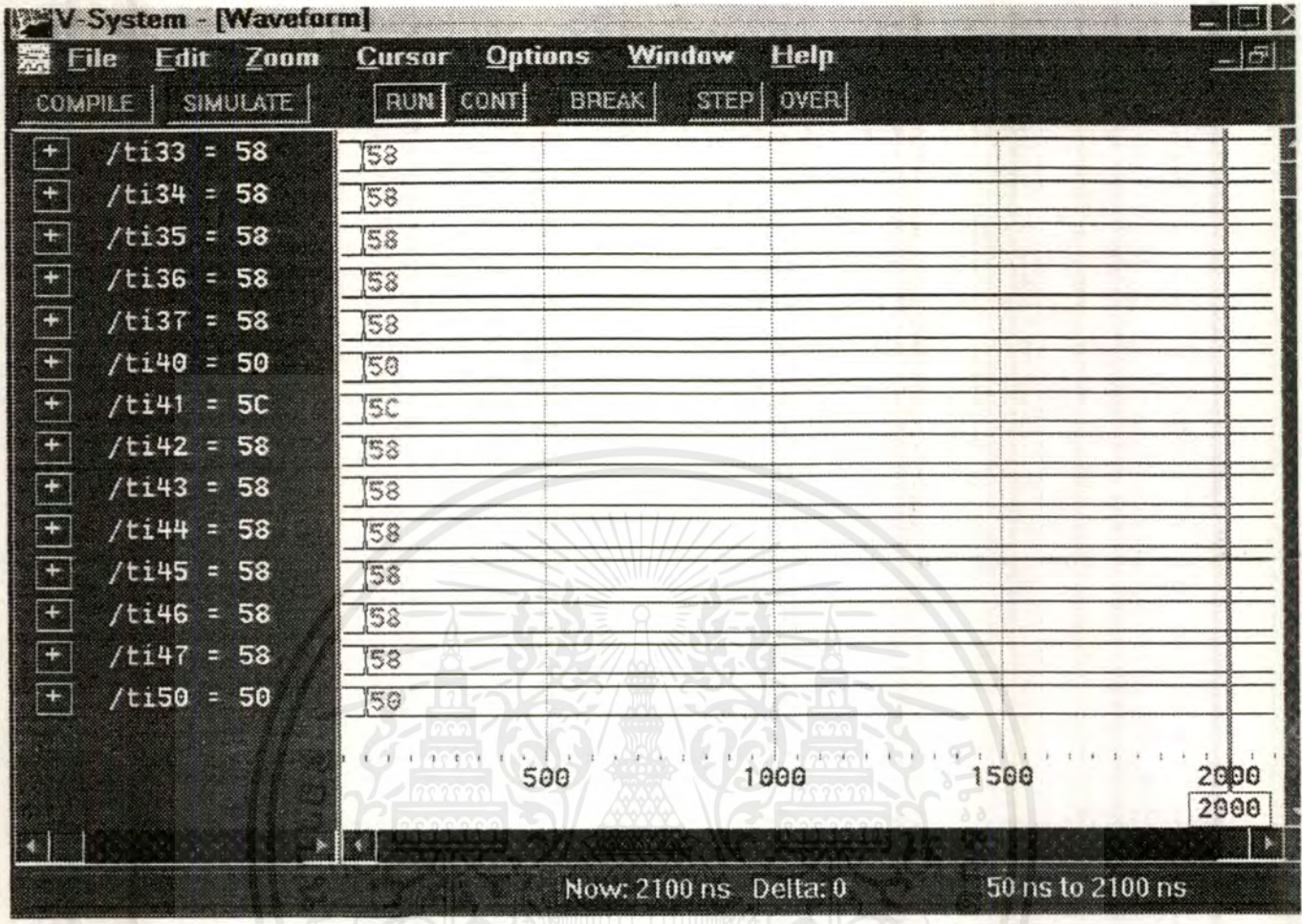
ดังรูปที่ 5.14 เป็นรูปของสัญญาณอินพุท(image)ที่ป้อนเข้าไป(ti00 - ti77) และสัญญาณนาฬิกา(tck)



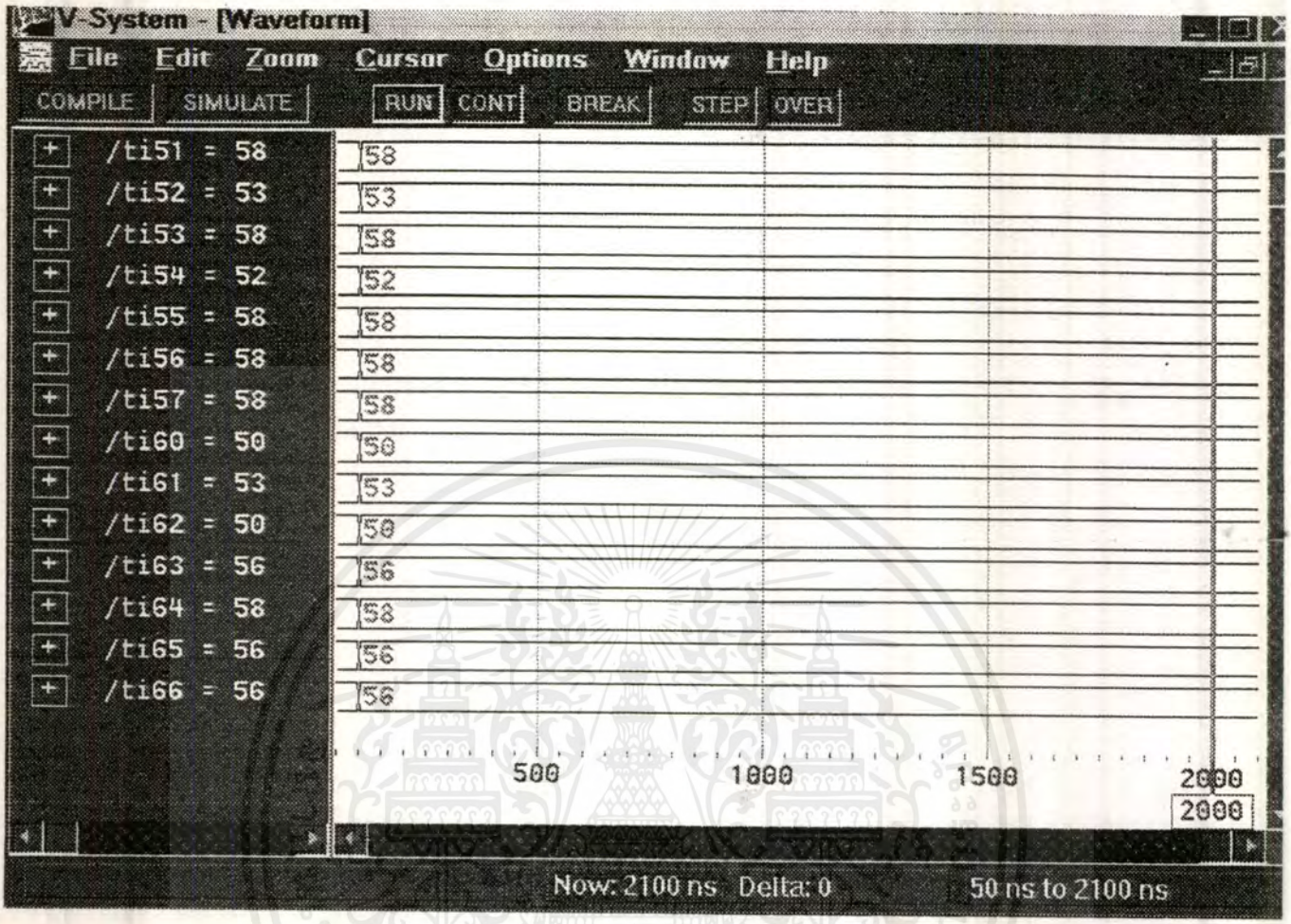
รูปที่ 5.14 สัญญาณอินพุท(image)



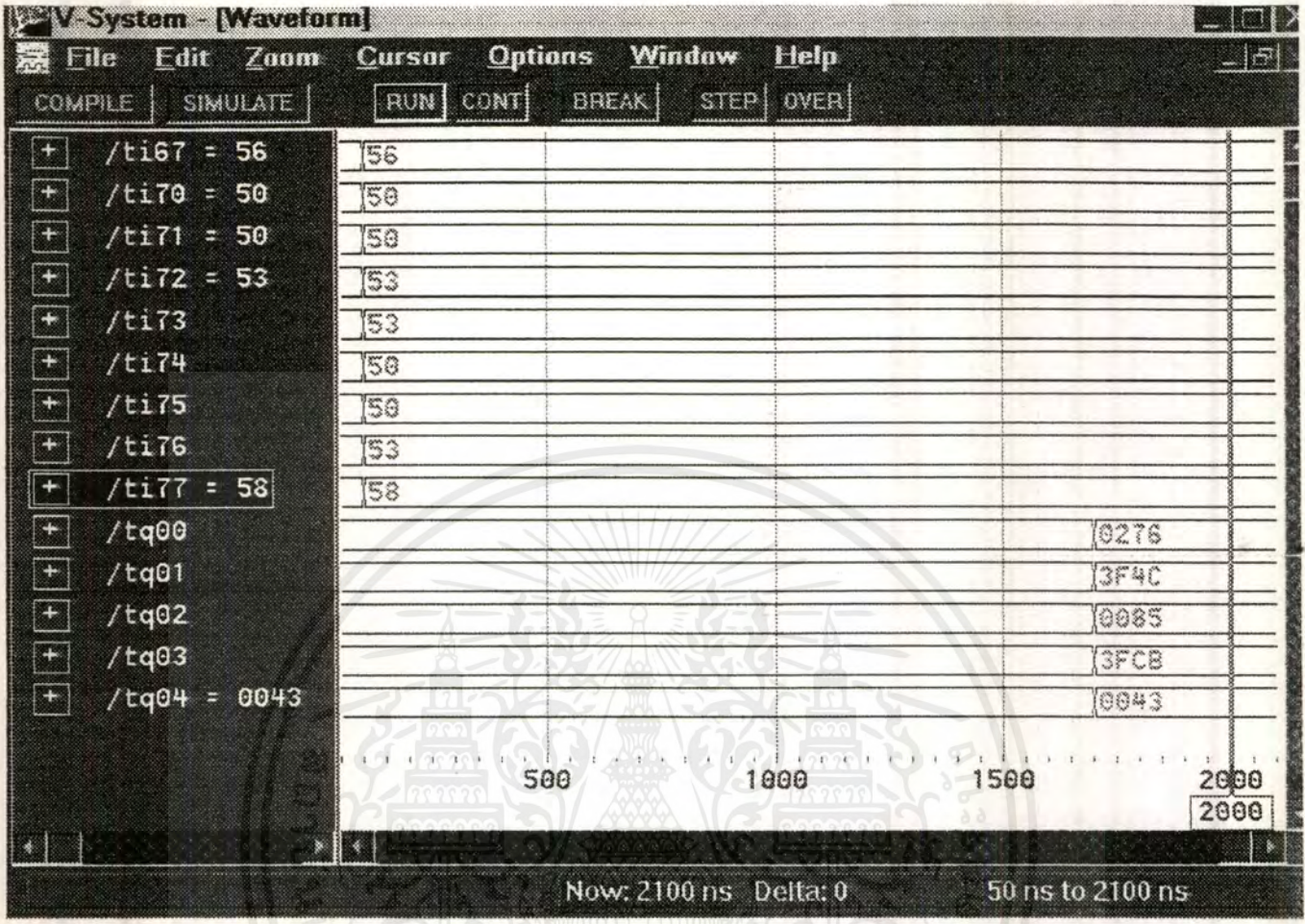
รูปที่ 5.14 สัญญาณอินพุท(ต่อ)



รูปที่ 5.14 สัญญาณอินพุท(ต่อ)

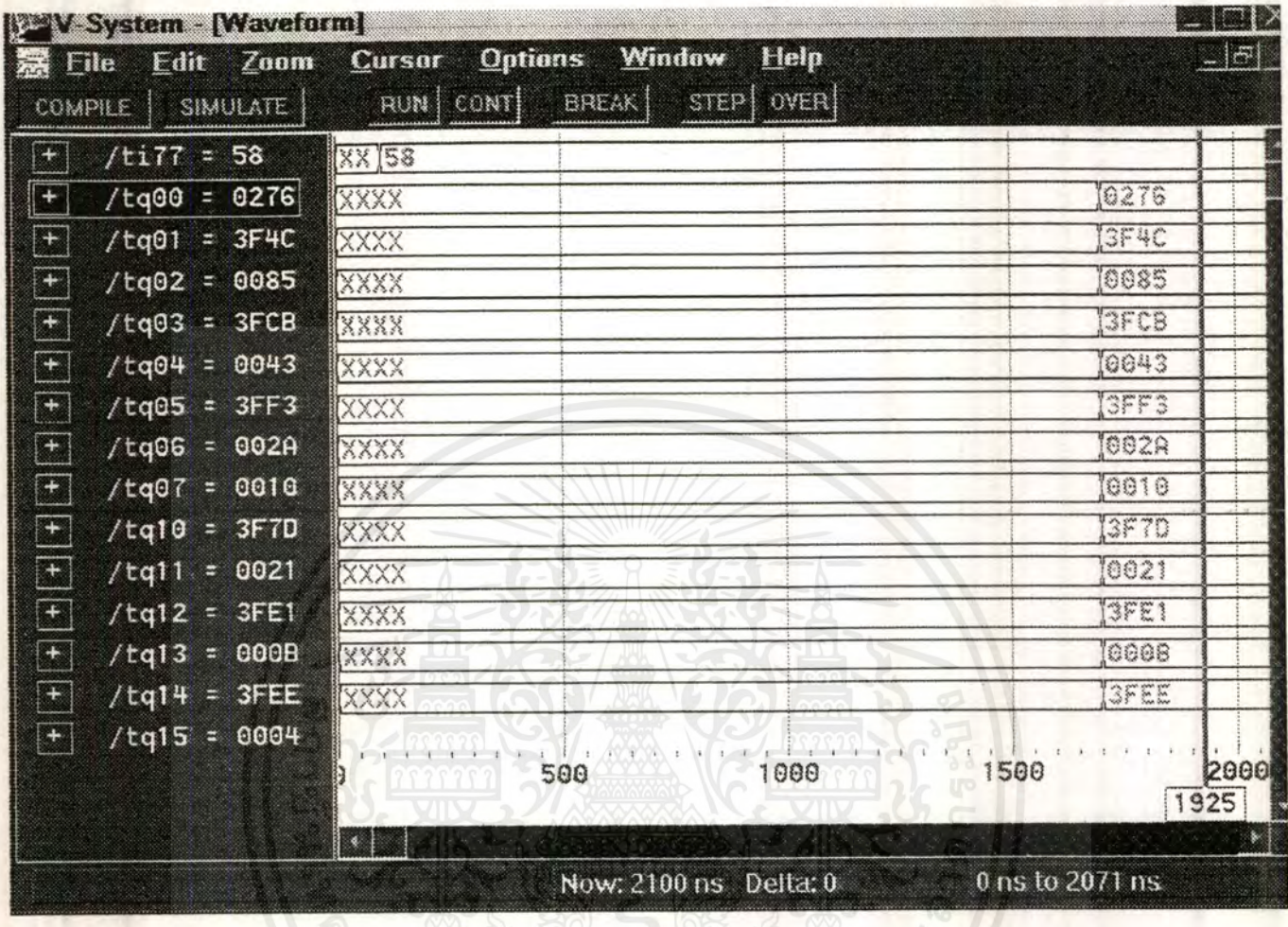


รูปที่ 5.14 สัญญาณอินพุท(ต่อ)

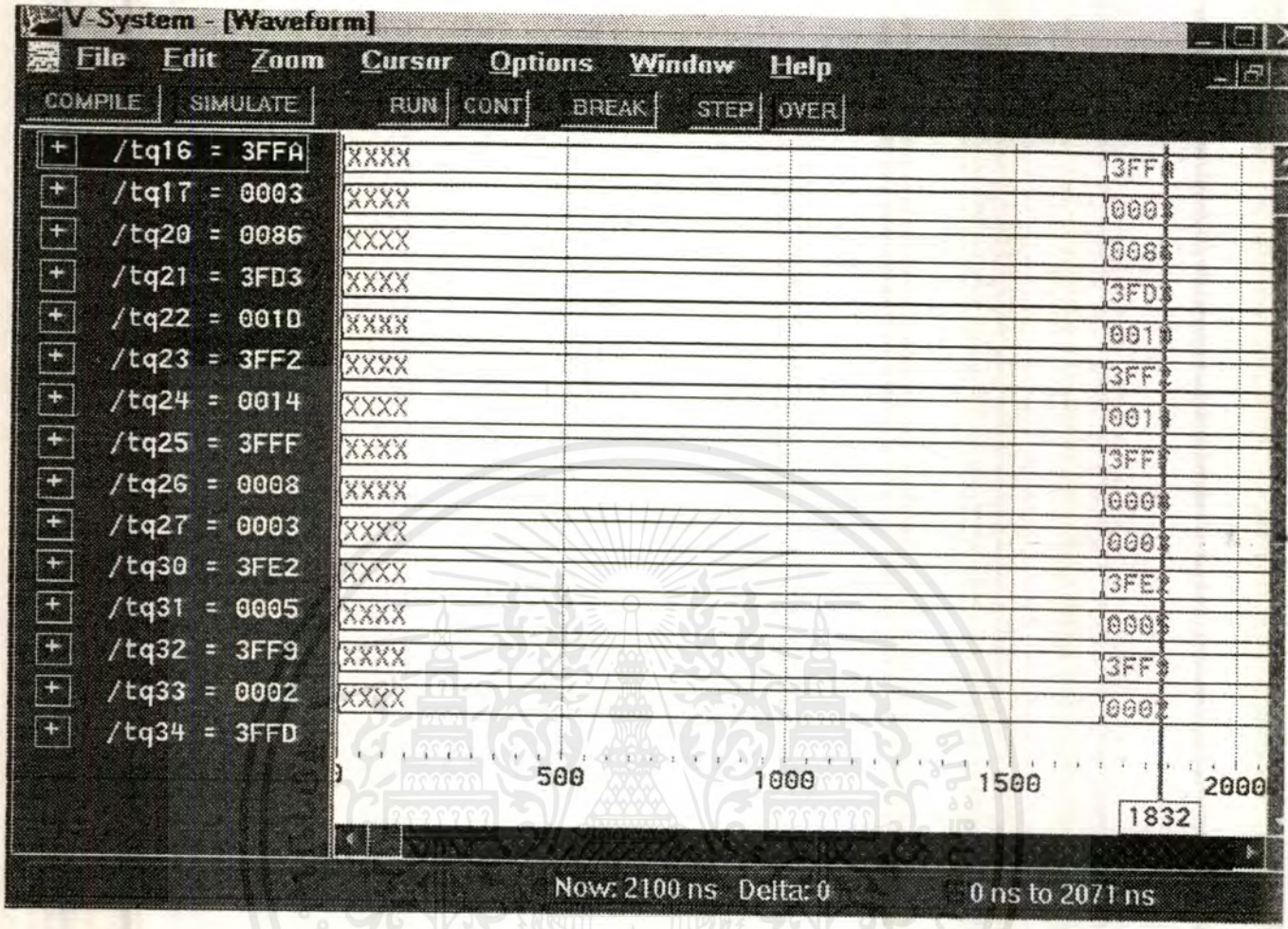


รูปที่ 5.14 สัญญาณอินพุท(ต่อ)

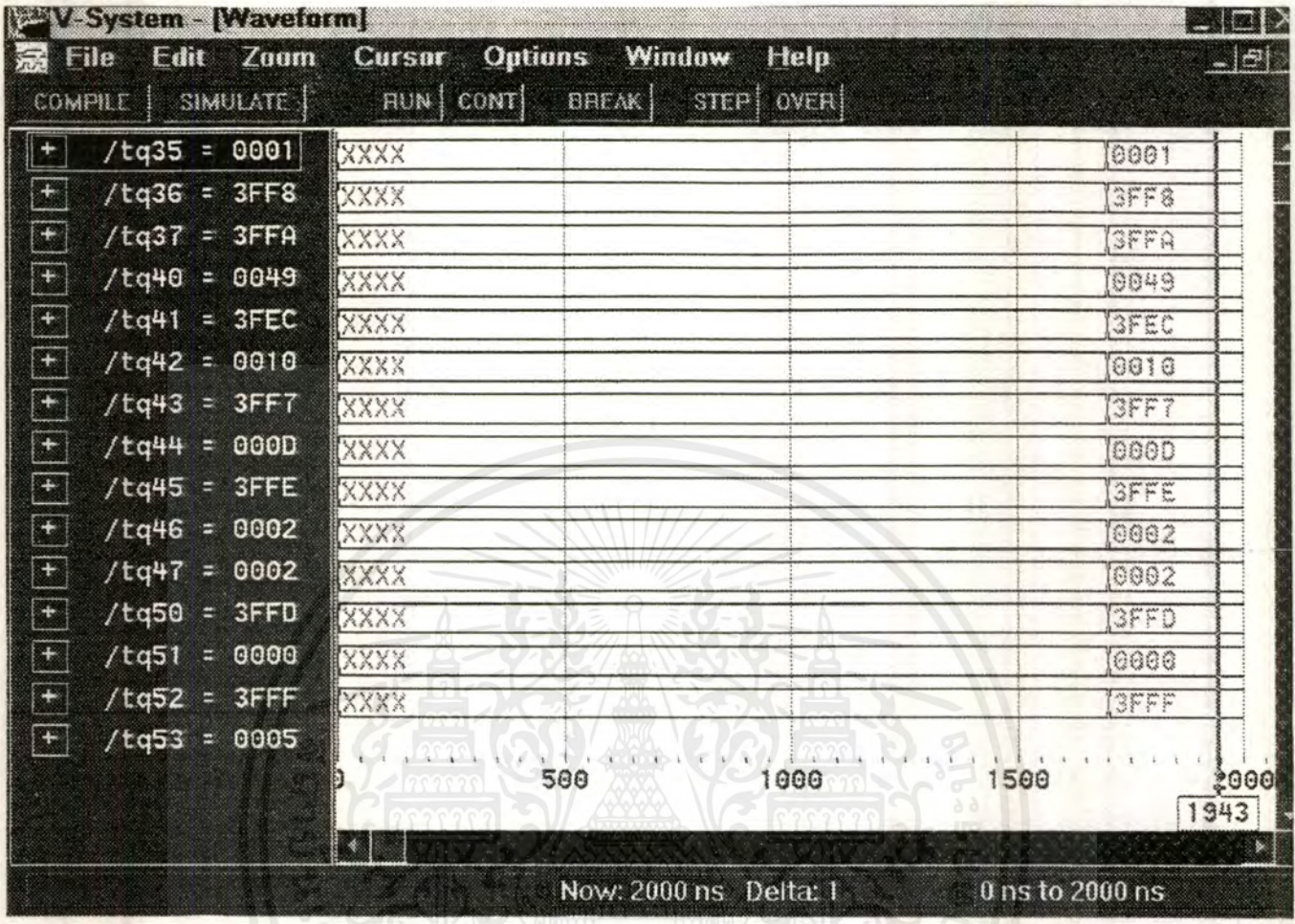
ในรูปที่ 5.15 เป็นรูปของสัญญาณเอาท์พุท(DCT)ที่ได้(tq00 - tq77)



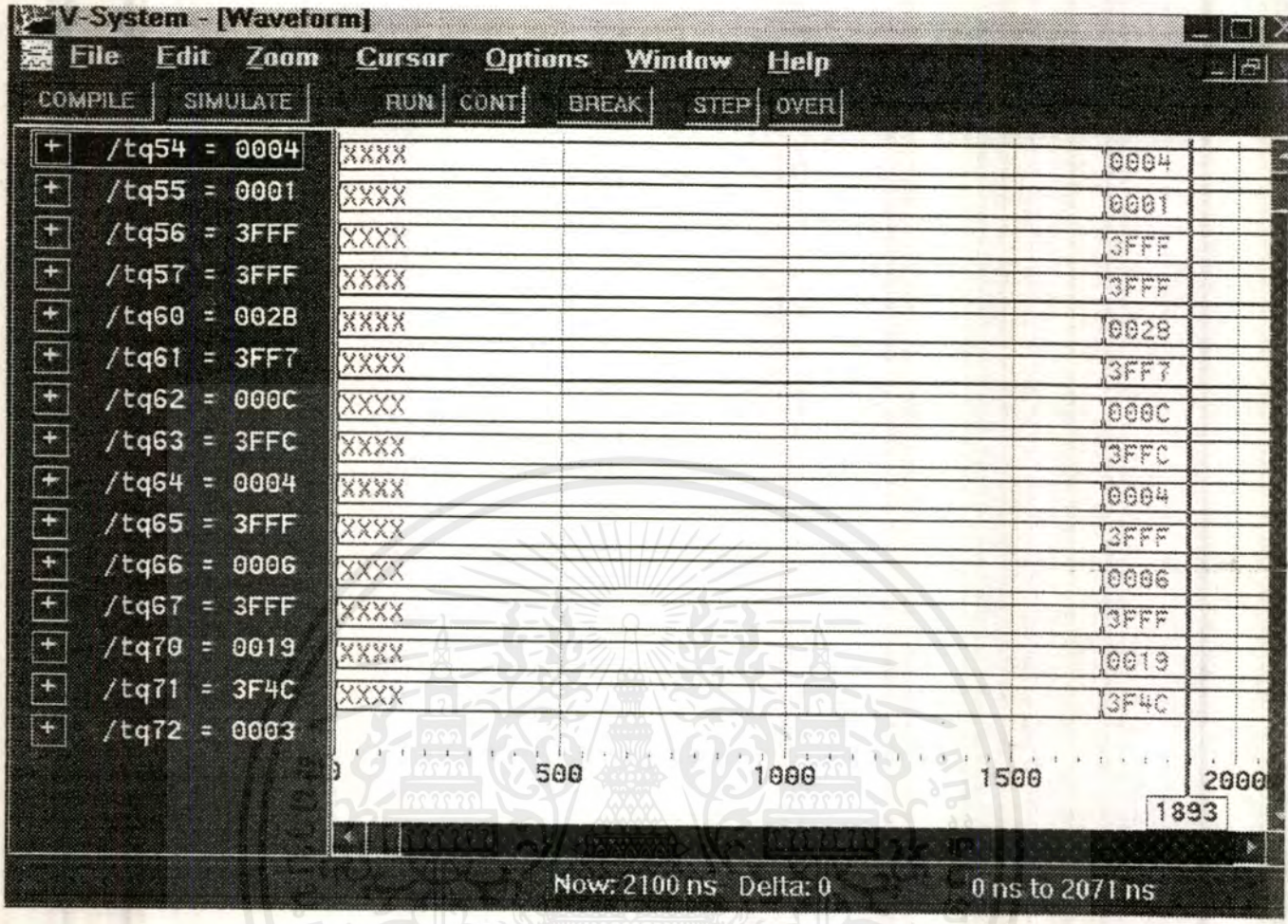
รูปที่ 5.15 สัญญาณเอาท์พุท(DCT)



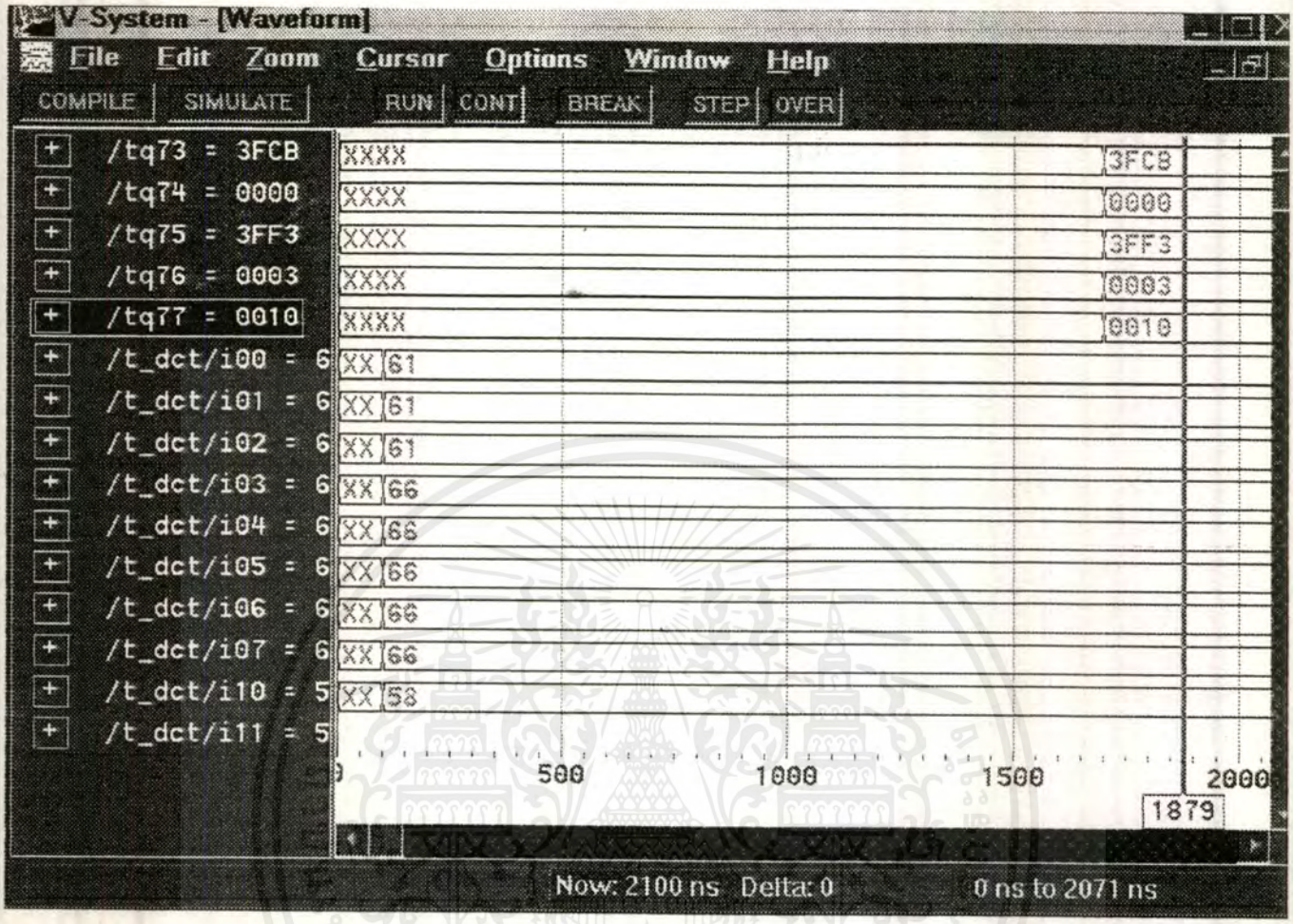
รูปที่ 5.15 สัญญาณเอาท์พุท(ต่อ)



รูปที่ 5.15 สัญญาณเอาท์พุท(ต่อ)

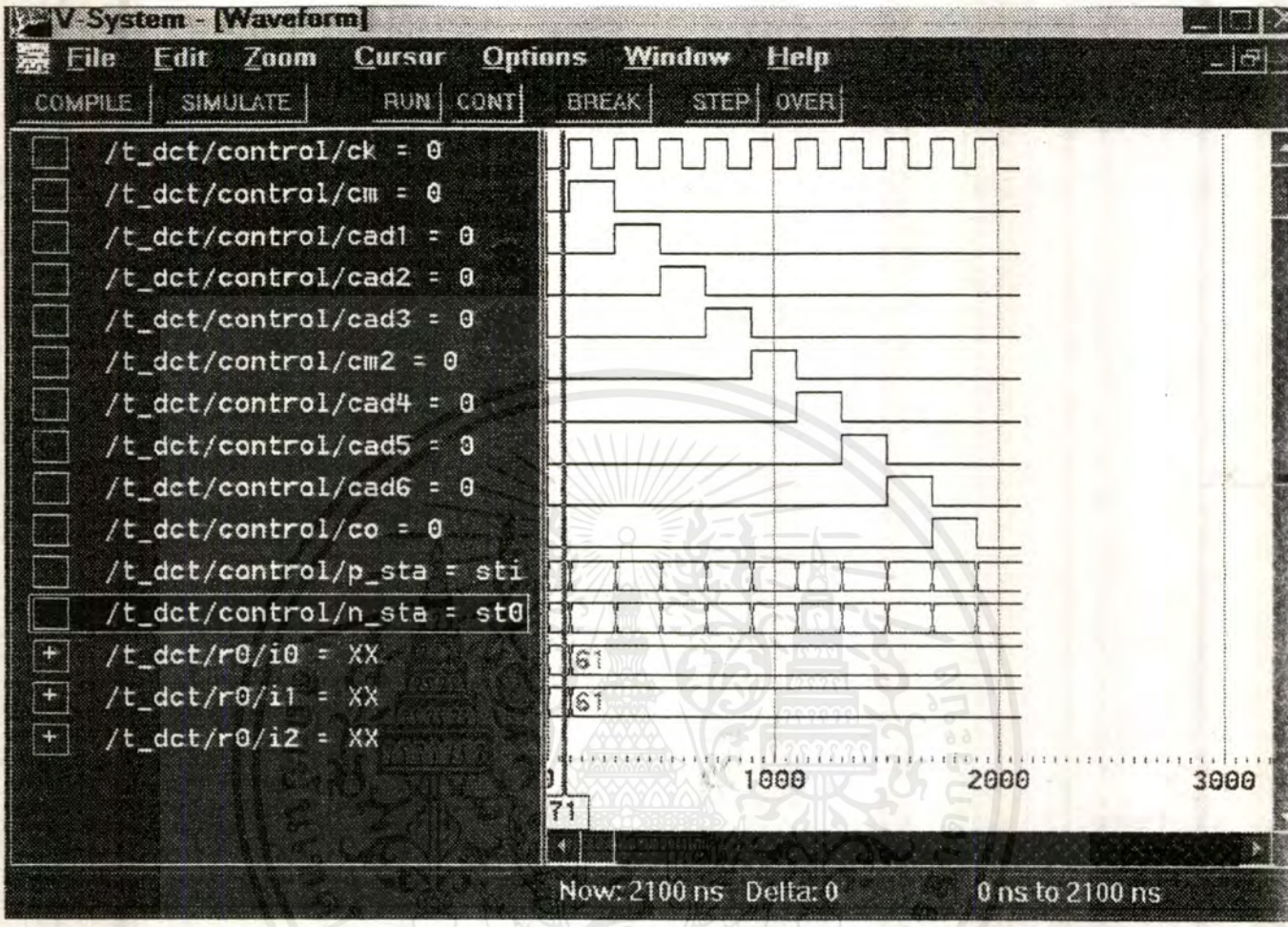


รูปที่ 5.15 สัญญาณเอาท์พุท(ต่อ)



รูปที่ 5.15 สัญญาณเอาท์พุท(ต่อ)

ในรูปที่ 5.16 เป็นรูปสัญญาณในการควบคุมการทำงานของอุปกรณ์ (State machine)



รูปที่ 5.16 สัญญาณควบคุมการทำงานของ state machine

บทที่6
บทสรุปและวิจารณ์

ในโครงงานนี้อุปกรณ์DCT ที่ออกแบบมีความเร็วในการทำงานโดยขึ้นอยู่กับการกำหนดสัญญาณนาฬิกาของอุปกรณ์ เนื่องจากการทำงานของอุปกรณ์ขึ้นอยู่กับ State machineซึ่งใช้สัญญาณนาฬิกาในการควบคุมโดยให้ 1 ไซเกิลของสัญญาณนาฬิกาเป็นขั้นตอนเวลาการทำงานของอุปกรณ์แต่ละตัวเป็นลำดับ โดยมีทั้งหมด 9 ขั้นตอน ดังนั้นเวลาการทำงานของ DCT จะใช้เวลา 9 ไซเกิลของสัญญาณนาฬิกา ในการทำงานจนจบ โดยในการกำหนดระยะเวลาของสัญญาณนาฬิกาจะขึ้นอยู่กับค่า Critical path หรือค่า delay time ของตัวอุปกรณ์ย่อยที่ให้ค่า Critical path หรือค่า delay time ที่มากที่สุด (จากการสังเคราะห์อยู่ประมาณ 160-200ns) โดยต้องให้ค่าของสัญญาณนาฬิกาใน1 ไซเกิลให้มากกว่า Critical path หรือค่า delay time

ในการออกแบบของอุปกรณ์DCT นี้ การออกแบบให้อุปกรณ์ให้มีความเร็วและให้ค่าที่แม่นยำเป็นสิ่งสำคัญ ซึ่งในโครงงานนี้ตัวอุปกรณ์อาจมีปัญหาในด้านนี้อยู่บ้าง โดยในการแก้ปัญหานี้อาจแก้ไขโดยการปรับปรุงอัลกอริทึมหรือมีการออกแบบอัลกอริทึมใหม่ ให้การทำงานของแต่ละส่วนใช้เวลาให้น้อยที่สุดและมีวิธีการคำนวณที่แม่นยำขึ้น

หนังสืออ้างอิง (Reference)

[1] Edward J.Delp and O. Robert Mitcell , "*Image Compression Using Block Trunction Coding*" , IEEE Transaction on communications , VOL.COM27 , 1979 , p. 1335 - 1342

[2] N. Ahmed , T. Natarajan and K.R. Rao , "*Discrete cosine transform*", IEEE Transation on communications , VOL.COM23 , 1974 , p. 90-93

[3] W. Chen , C.H. Smith and S. Fralick , "*A fast computational algorithm for the discrete cosine transform*" , IEEE Tran. Commu. Techrol. , VOL COM25 , 1977 , p. 1258-1291

[4] Tora F. Fliegel , "*The Data Compression Book*" , Mark Nelsan , p. 374-308

[5] Phillip e. Mattition "*PRACTICAL DIGITAL VIDEO WITH PROGRAMING EXAMPLES IN C*" , John wiley & Son,INC.,N.Y.

[6] Jayaram Bhasker ,AT&T Bell Laboratories Division , "*AVHDL Primer*" , Prentice Hall ,1992

[7] Douglas L. Perry , "*VHDL*" , McGraw-Hill ,INC. ,1991

[8] Examplar Logic , "*HDL Synthesis Reference Manual VHDL,Verilog HDL & Module Generation*" , Examplar Logic,INC. , 1994-1995

[9] Examplar Logic , "*Galileo User Manual*" , Examplar Logic,INC. , 1991-1995

กิตติกรรมประกาศ

ในการทำปริญญานิพนธ์นี้จนสำเร็จลุล่วงไปได้ข้าพเจ้าต้องขอขอบคุณ อ. สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา ที่ได้ให้คำแนะนำและให้เอกสารต่างๆที่สำคัญในการค้นคว้า และเพื่อนๆทุกคนในภาควิชาอิเล็กทรอนิกส์ที่เป็นกำลังใจและให้คำปรึกษาโดยเฉพาะนาย นพดล ม่วงเฉย(เอก) ที่ได้ให้คำปรึกษาตลอดมา และนายวีรภัทร วรรณะแสวง เพื่อนภาคโยธาที่ได้ช่วยขับรถนำ computer มาใช้ในวัน Present และท้ายที่สุดก็ขอขอบคุณเพื่อนๆในคณะวิศวะ'ลาดกระบังของข้าพเจ้าทุกคนที่เป็นกำลังใจให้ และอาจารย์ทุกท่านที่ได้สั่งสอนให้ข้าพเจ้ามีความรู้ได้มีความสามารถทำให้สำเร็จไปได้

ภาคผนวก

(APPENDIX)

โปรแกรมVHDL

multer1

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;
PACKAGE memdct IS

    TYPE mem210_data IS
     array(0 to 7,0 to 7) of std_logic_vector(9 downto 0);

  constant dct_coeff : mem210_data:=
   (("0101101010","0111110110","0111011001","0110101010",
    "0101101010","0100011100","0011000100","0001100100"),
    ("0101101010","0110101010","0011000100","1110011100",
    "1010010110","1000011010","1000100111","1011100100"),
    ("0101101010","0100011100","1100111100","1000001010",
    "1010010110","0001100100","0111011001","0110101010"),
    ("0101101010","0001100100","1000100111","1011100100",
    "0101101010","0110101010","1100111100","1000001010"),
    ("0101101010","1110011100","1000100111","0100011100",
    "0101101010","1001010110","1100111100","0111110110"),
    ("0101101010","1011100100","1100111100","0111110110",
    "1010010110","1110011100","0111001001","1001010110"),
    ("0101101010","1001010110","0011000100","0001100100",
    "1010010110","0111110110","1000100111","0100011100"),
    ("0101101010","1000001010","0111011001","1001010110",
    "0101101010","1011100100","0011000100","1110011100"));
END  memdct;

---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;
use work.memdct.ALL;

ENTITY multer1 IS
 port(i : IN  std_logic_vector(7 downto 0) ;
      ld : IN std_logic ;
      c,r:IN integer range 0 to 7;
      q : OUT  std_logic_vector(17 downto 0 ));
END multer1;

 ARCHITECTURE  multer1_behav of multer1 IS
```

```
begin

 m1:process(i,ld,c,r)
    begin
     if ld = '1' and ld'event then
        q <= mult2(i,dct_coeff(c,r));
     end if;
    end process m1;
end multer1_behav;

configuration multer1con of multer1 is
  for multer1_behav
  end for;
end multer1con;
```

multer2

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;
PACKAGE memdct IS

    TYPE mem212_data IS
     array(0 to 7,0 to 7) of std_logic_vector(11 downto 0);

 constant dct_coeff : mem212_data:=
  (("000101101010","000111110110","000111011001","000110101010",
    "000101101010","000100011100","000011000100","000001100100"),
   ("000101101010","000110101010","000011000100","111110011100",
    "111010010110","111000011010","111000100111","111011100100"),
   ("000101101010","000100011100","111100111100","111000001010",
    "111010010110","000001100100","000111011001","000110101010"),
   ("000101101010","000001100100","111000100111","111011100100",
    "000101101010","000110101010","111100111100","111000001010"),
   ("000101101010","111110011100","111000100111","000100011100",
    "000101101010","111001010110","111100111100","000111110110"),
   ("000101101010","111011100100","111100111100","000111110110",
    "111010010110","111110011100","000111001001","111001010110"),
   ("000101101010","111001010110","000011000100","000001100100",
    "111010010110","000111110110","111000100111","000100011100"),
   ("000101101010","111000001010","000111011001","111001010110",
    "000101101010","111011100100","000011000100","111110011100"));
END memdct;

---//--BEGIN PROGRAM
```

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;
use work.memdct.ALL;

ENTITY multer2 IS
 port(i : IN  std_logic_vector(22 downto 0) ;
      ld : IN std_logic ;
      c,r:IN integer range 0 to 7;
      q : OUT  std_logic_vector(34 downto 0 ));
END multer2;

ARCHITECTURE  multer2_behav of multer2 IS
begin

  m2:process(ld,i,c,r)
     begin
      if ld = '1' and ld'event then
         q <= mult2(i,dct_coeff(c,r));
      end if;
     end process m2;
end multer2_behav;

configuration  multer2con  of multer2 is
   for multer2_behav
   end for;
end multer2con;
```

ADDER1

```
---//--BEGIN PROGRAM
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY adder1 IS
 port(i0,i1: IN  std_logic_vector(17 downto 0) ;
      ld : IN std_logic ;
      q : OUT  std_logic_vector(18 downto 0 ));
END adder1;

ARCHITECTURE  adder1_behav of adder1 IS
begin

   A1:process(i0,i1,ld)
```

```
 A1:process(i0,i1,ld)
    begin
     if ld = '1' and ld'event then
       q <= add2(i0,i1);
     end if;
    end process A1;
end adder1_behav;

configuration adder1con of adder1 is
  for adder1_behav
  end for;
end adder1con;
```

ADDER2

```
---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY adder2 IS
 port(i0,i1: IN  std_logic_vector(18 downto 0) ;
      ld : IN std_logic;
      q : OUT  std_logic_vector(19 downto 0 ));
END adder2;

ARCHITECTURE  adder2_behav of adder2 IS
begin

 A2:process(i0,i1,ld)
    begin
     if ld = '1' and ld'event then
       q <= add2(i0,i1);
     end if;
    end process A2;
end adder2_behav;

configuration adder2con of adder2 is
  for adder2_behav
  end for;
end adder2con;
```

ADDER3

```
---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1 1 64.all;
use ieee.std_logic_1 1 64.all;

ENTITY adder3 IS
 port(i0,i1: IN  std_logic_vector(19 downto 0) ;
      ld : IN std_logic ;
      q : OUT  std_logic_vector(20 downto 0 ));
END adder3;

ARCHITECTURE  adder3_behav of adder3 IS
begin

  A3:process(i0,i1,ld)
     begin
      if ld = '1'  and ld'event then
        q <= add2(i0,i1);
      end if;
     end process A3;
end adder3_behav;

configuration adder3con of adder3 is
   for adder3_behav
   end for;
end adder3con;
```

AD_DER_1

```
---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1 1 64.all;
use ieee.std_logic_1 1 64.all;

ENTITY ad_der1 IS
 port(i0,i1,i2,i3,i4,i5,i6,i7: IN  std_logic_vector(17 downto 0) ;
      ld1,ld2,ld3 : IN std_logic;
      q : OUT  std_logic_vector(20 downto 0 ));
END ad_der1;

ARCHITECTURE  ad_der1_behav of ad_der1 IS
```

```
  component adder1
   port (i0,i1:in std_logic_vector(17 downto 0);
         ld:in std_logic;
         q:out std_logic_vector(18 downto 0));
  end component;

  component adder2
   port (i0,i1:in std_logic_vector(18 downto 0);
         ld:in std_logic;
         q:out std_logic_vector(19 downto 0));
  end component;

  component adder3
   port (i0,i1:in std_logic_vector(19 downto 0);
         ld:in std_logic;
         q:out std_logic_vector(20 downto 0));
  end component;

 signal o11,o12,o13,o14 :std_logic_vector(18 downto 0);
 signal o21,o22 :std_logic_vector(19 downto 0);

begin

 ad11 :adder1  port map (i0,i1,ld1,o11);
 ad12 :adder1  port map (i2,i3,ld1,o12);
 ad13 :adder1  port map (i4,i5,ld1,o13);
 ad14 :adder1  port map (i6,i7,ld1,o14);
 ad21 :adder2  port map (o11,o12,ld2,o21);
 ad22 :adder2  port map (o13,o14,ld2,o22);
 ad31 :adder3  port map (o21,o22,ld3,q);

end ad_der1_behav;

configuration ad_der1con of ad_der1 is

  for ad_der1_behav
   for ad11,ad12,ad13,ad14:adder1 use entity work.adder1(adder1_bahav);
   end for;
   for ad21,ad22:adder2 use entity work.adder2(adder2_bahav);
   end for;
   for ad31:adder3 use entity work.adder3(adder3_bahav);
   end for;
  end for;

end ad_der1con;
```

ADDER21

```
---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY adder21 IS
 port(i0,i1: IN  std_logic_vector(30 downto 0) ;
      ld : IN std_logic ;
      q : OUT  std_logic_vector(31 downto 0 ));
END adder21;

ARCHITECTURE  adder21_behav of adder21 IS
begin

  A1:process(i0,i1,ld)
     begin
      if ld = '1' and ld'event then
        q <= add2(i0,i1);
      end if;
     end process A1;
end adder21_behav;

configuration adder21con of adder21 is
   for adder21_behav
   end for;
end adder21con;
```

ADDER22

```
---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY adder22 IS
 port(i0,i1: IN  std_logic_vector(31 downto 0) ;
      ld : IN std_logic;
      q : OUT  std_logic_vector(32 downto 0 ));
```

```
END adder22;

ARCHITECTURE  adder22_behav of adder22 IS
begin

 A2:process(i0,i1,ld)
    begin
     if ld = '1' and ld'event then
       q <= add2(i0,i1);
     end if;
    end process A2;
end adder22_behav;

configuration adder22con of adder22 is
   for adder22_behav
   end for;
end adder22con;
```

ADDER23

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY adder23 IS
 port(i0,i1: IN  std_logic_vector(32 downto 0) ;
      ld : IN std_logic ;
      q : OUT  std_logic_vector(33 downto 0 ));
END adder23;

ARCHITECTURE  adder23_behav of adder23 IS
begin

  A3:process(i0,i1,ld)
     begin
      if ld = '1'  and ld'event then
        q <= add2(i0,i1);
      end if;
     end process A3;
end adder23_behav;

configuration adder23con of adder23 is
   for adder23_behav
   end for;
end adder23con;
```

AD_DER_2

```
---//--BEGIN PROGRAM

library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY ad_der2 IS
 port(i0,i1,i2,i3,i4,i5,i6,i7: IN  std_logic_vector(30 downto 0) ;
      ld1,ld2,ld3 : IN std_logic;
      q : OUT  std_logic_vector(33 downto 0 ));
END ad_der2;

ARCHITECTURE  ad_der2_behav of ad_der2 IS
   component adder21
    port (i0,i1:in std_logic_vector(30 downto 0);
          ld:in std_logic;
          q:out std_logic_vector(31 downto 0));
   end component;

   component adder22
    port (i0,i1:in std_logic_vector(31 downto 0);
          ld:in std_logic;
          q:out std_logic_vector(32 downto 0));
   end component;

   component adder23
    port (i0,i1:in std_logic_vector(32 downto 0);
          ld:in std_logic;
          q:out std_logic_vector(33 downto 0));
   end component;

  signal o11,o12,o13,o14 :std_logic_vector(31 downto 0);
  signal o21,o22 :std_logic_vector(32 downto 0);

begin

 ad11 :adder21  port map (i0,i1,ld1,o11);
 ad12 :adder21  port map (i2,i3,ld1,o12);
 ad13 :adder21  port map (i4,i5,ld1,o13);
 ad14 :adder21  port map (i6,i7,ld1,o14);
 ad21 :adder22  port map (o11,o12,ld2,o21);
 ad22 :adder22  port map (o13,o14,ld2,o22);
 ad31 :adder23  port map (o21,o22,ld3,q);
```

```
end ad_der2_behav;

configuration ad_der2con of ad_der2 is

  for ad_der2_behav
   for all:adder21 use entity work.adder21(adder21_bahav);
   end for;
   for all:adder22 use entity work.adder22(adder22_bahav);
   end for;
   for all:adder23 use entity work.adder23(adder23_bahav);
   end for;
  end for;

end ad_der2con;
```

DCTR

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY dctr IS
 port(i0,i1,i2,i3,i4,i5,i6,i7: IN  std_logic_vector(7 downto 0) ;
      ld_m,ld_a1,ld_a2,ld_a3 : IN std_logic;
      q0,q1,q2,q3,q4,q5,q6,q7 : OUT  std_logic_vector(20 downto 0 ));
END dctr;

ARCHITECTURE  dctr_behav of dctr IS
  component ad_der1
   port (i0,i1,i2,i3,i4,i5,i6,i7:in std_logic_vector(17 downto 0);
         ld1,ld2,ld3 : IN std_logic;
         q:out std_logic_vector(20 downto 0));
  end component;

  component multer1
   port (i:in std_logic_vector(7 downto 0);
         ld:in std_logic;
         c,r :in integer range 0 to 7;
         q:out std_logic_vector(17 downto 0));
  end component;
 signal t00,t01,t02,t03,t04,t05,t06,t07,
        t10,t11,t12,t13,t14,t15,t16,t17,
        t20,t21,t22,t23,t24,t25,t26,t27,
        t30,t31,t32,t33,t34,t35,t36,t37,
```

```
        t40,t41,t42,t43,t44,t45,t46,t47,
        t50,t51,t52,t53,t54,t55,t56,t57,
        t60,t61,t62,t63,t64,t65,t66,t67,
        t70,t71,t72,t73,t74,t75,t76,t77 : std_logic_vector(17 downto 0);


    signal j0,j1,j2,j3,j4,j5,j6,j7 :integer;

    begin

     j0 <= 0;  j1 <= 1;  j2 <= 2;  j3 <= 3;
     j4 <= 4;  j5 <= 5;  j6 <= 6;  j7 <= 7;

     multi00  :multer1 port map(i0,ld_m,j0,j0,t00);
     multi01  :multer1 port map(i1,ld_m,j0,j1,t01);
     multi02  :multer1 port map(i2,ld_m,j0,j2,t02);
     multi03  :multer1 port map(i3,ld_m,j0,j3,t03);
     multi04  :multer1 port map(i4,ld_m,j0,j4,t04);
     multi05  :multer1 port map(i5,ld_m,j0,j5,t05);
     multi06  :multer1 port map(i6,ld_m,j0,j6,t06);
     multi07  :multer1 port map(i7,ld_m,j0,j7,t07);
     adde0  :ad_der1  port map (t00,t01,t02,t03,t04,t05,t06,t07,
                    ld_a1,ld_a2,ld_a3,q0 );

     multi10  :multer1 port map(i0,ld_m,j1,j0,t10);
     multi11  :multer1 port map(i1,ld_m,j1,j1,t11);
     multi12  :multer1 port map(i2,ld_m,j1,j2,t12);
     multi13  :multer1 port map(i3,ld_m,j1,j3,t13);
     multi14  :multer1 port map(i4,ld_m,j1,j4,t14);
     multi15  :multer1 port map(i5,ld_m,j1,j5,t15);
     multi16  :multer1 port map(i6,ld_m,j1,j6,t16);
     multi17  :multer1 port map(i7,ld_m,j1,j7,t17);
     adde1  :ad_der1  port map (t10,t11,t12,t13,t14,t15,t16,t17,
                    ld_a1,ld_a2,ld_a3,q1 );

     multi20  :multer1 port map(i0,ld_m,j2,j0,t20);
     multi21  :multer1 port map(i1,ld_m,j2,j1,t21);
     multi22  :multer1 port map(i2,ld_m,j2,j2,t22);
     multi23  :multer1 port map(i3,ld_m,j2,j3,t23);
     multi24  :multer1 port map(i4,ld_m,j2,j4,t24);
     multi25  :multer1 port map(i5,ld_m,j2,j5,t25);
     multi26  :multer1 port map(i6,ld_m,j2,j6,t26);
     multi27  :multer1 port map(i7,ld_m,j2,j7,t27);
     adde2  :ad_der1  port map (t20,t21,t22,t23,t24,t25,t26,t27,
                    ld_a1,ld_a2,ld_a3,q2 );

     multi30  :multer1 port map(i0,ld_m,j3,j0,t30);
```

```
multi31  :multer1 port map(i1,ld_m,j3,j1,t31);
multi32  :multer1 port map(i2,ld_m,j3,j2,t32);
multi33  :multer1 port map(i3,ld_m,j3,j3,t33);
multi34  :multer1 port map(i4,ld_m,j3,j4,t34);
multi35  :multer1 port map(i5,ld_m,j3,j5,t35);
multi36  :multer1 port map(i6,ld_m,j3,j6,t36);
multi37  :multer1 port map(i7,ld_m,j3,j7,t37);
adde3  :ad_der1  port map (t30,t31,t32,t33,t34,t35,t36,t37,
                    ld_a1,ld_a2,ld_a3,q3 );

multi40  :multer1 port map(i0,ld_m,j4,j0,t40);
multi41  :multer1 port map(i1,ld_m,j4,j1,t41);
multi42  :multer1 port map(i2,ld_m,j4,j2,t42);
multi43  :multer1 port map(i3,ld_m,j4,j3,t43);
multi44  :multer1 port map(i4,ld_m,j4,j4,t44);
multi45  :multer1 port map(i5,ld_m,j4,j5,t45);
multi46  :multer1 port map(i6,ld_m,j4,j6,t46);
multi47  :multer1 port map(i7,ld_m,j4,j7,t47);
adde4  :ad_der1  port map (t40,t41,t42,t43,t44,t45,t46,t47,
                    ld_a1,ld_a2,ld_a3,q4 );

multi50  :multer1 port map(i0,ld_m,j5,j0,t50);
multi51  :multer1 port map(i1,ld_m,j5,j1,t51);
multi52  :multer1 port map(i2,ld_m,j5,j2,t52);
multi53  :multer1 port map(i3,ld_m,j5,j3,t53);
multi54  :multer1 port map(i4,ld_m,j5,j4,t54);
multi55  :multer1 port map(i5,ld_m,j5,j5,t55);
multi56  :multer1 port map(i6,ld_m,j5,j6,t56);
multi57  :multer1 port map(i7,ld_m,j5,j7,t57);
adde5  :ad_der1  port map (t50,t51,t52,t53,t54,t55,t56,t57,
                    ld_a1,ld_a2,ld_a3,q5 );

multi60  :multer1 port map(i0,ld_m,j6,j0,t60);
multi61  :multer1 port map(i1,ld_m,j6,j1,t61);
multi62  :multer1 port map(i2,ld_m,j6,j2,t62);
multi63  :multer1 port map(i3,ld_m,j6,j3,t63);
multi64  :multer1 port map(i4,ld_m,j6,j4,t64);
multi65  :multer1 port map(i5,ld_m,j6,j5,t65);
multi66  :multer1 port map(i6,ld_m,j6,j6,t66);
multi67  :multer1 port map(i7,ld_m,j6,j7,t67);
adde6  :ad_der1  port map (t60,t61,t62,t63,t64,t65,t66,t67,
                    ld_a1,ld_a2,ld_a3,q6 );

multi70  :multer1 port map(i0,ld_m,j7,j0,t70);
multi71  :multer1 port map(i1,ld_m,j7,j1,t71);
multi72  :multer1 port map(i2,ld_m,j7,j2,t72);
multi73  :multer1 port map(i3,ld_m,j7,j3,t73);
```

```
  multi74  :multer1 port map(i4,ld_m,j7,j4,t74);
  multi75  :multer1 port map(i5,ld_m,j7,j5,t75);
  multi76  :multer1 port map(i6,ld_m,j7,j6,t76);
  multi77  :multer1 port map(i7,ld_m,j7,j7,t77);
  adde7  :ad_der1  port map (t70,t71,t72,t73,t74,t75,t76,t77,
                        ld_a1,ld_a2,ld_a3,q7 );

end dctr_behav;

configuration dctrcon of dctr is

   for dctr_behav
    for all:multer1 use entity work.multer1(multer1_bahav);
    end for;
    for all:ad_der1 use entity work.ad_der1(ad_der1_bahav);
    end for;
   end for;

end dctrcon;
```

DCTC

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

ENTITY dctc IS
 port(i0,i1,i2,i3,i4,i5,i6,i7: IN  std_logic_vector(20 downto 0) ;
      ld_m,ld_a1,ld_a2,ld_a3 : IN std_logic;
      q0,q1,q2,q3,q4,q5,q6,q7 : OUT  std_logic_vector(33 downto 0 ));
END dctc;

ARCHITECTURE  dctc_behav of dctc IS
   component ad_der2
    port (i0,i1,i2,i3,i4,i5,i6,i7:in std_logic_vector(30 downto 0);
          ld1,ld2,ld3 : IN std_logic;
          q:out std_logic_vector(33 downto 0));
   end component;

   component multer2
    port (i:in std_logic_vector(20 downto 0):
          ld:in std_logic;
          c,r :in integer range 0 to 7;
          q:out std_logic_vector(30 downto 0));
   end component;
```

```
signal t00,t01,t02,t03,t04,t05,t06,t07,
       t10,t11,t12,t13,t14,t15,t16,t17,
       t20,t21,t22,t23,t24,t25,t26,t27,
       t30,t31,t32,t33,t34,t35,t36,t37,
       t40,t41,t42,t43,t44,t45,t46,t47,
       t50,t51,t52,t53,t54,t55,t56,t57,
       t60,t61,t62,t63,t64,t65,t66,t67,
       t70,t71,t72,t73,t74,t75,t76,t77 : std_logic_vector(30 downto 0);

signal j0,j1,j2,j3,j4,j5,j6,j7 :integer;

begin

 j0 <= 0; j1 <= 1; j2 <= 2; j3 <= 3;
 j4 <= 4; j5 <= 5; j6 <= 6; j7 <= 7;

 multi00 :multer2 port map(i0,ld_m,j0,j0,t00);
 multi01 :multer2 port map(i1,ld_m,j0,j1,t01);
 multi02 :multer2 port map(i2,ld_m,j0,j2,t02);
 multi03 :multer2 port map(i3,ld_m,j0,j3,t03);
 multi04 :multer2 port map(i4,ld_m,j0,j4,t04);
 multi05 :multer2 port map(i5,ld_m,j0,j5,t05);
 multi06 :multer2 port map(i6,ld_m,j0,j6,t06);
 multi07 :multer2 port map(i7,ld_m,j0,j7,t07);
 adde0 :ad_der2 port map (t00,t01,t02,t03,t04,t05,t06,t07,
                          ld_a1,ld_a2,ld_a3,q0 );

 multi10 :multer2 port map(i0,ld_m,j1,j0,t10);
 multi11 :multer2 port map(i1,ld_m,j1,j1,t11);
 multi12 :multer2 port map(i2,ld_m,j1,j2,t12);
 multi13 :multer2 port map(i3,ld_m,j1,j3,t13);
 multi14 :multer2 port map(i4,ld_m,j1,j4,t14);
 multi15 :multer2 port map(i5,ld_m,j1,j5,t15);
 multi16 :multer2 port map(i6,ld_m,j1,j6,t16);
 multi17 :multer2 port map(i7,ld_m,j1,j7,t17);
 adde1 :ad_der2 port map (t10,t11,t12,t13,t14,t15,t16,t17,
                          ld_a1,ld_a2,ld_a3,q1 );

 multi20 :multer2 port map(i0,ld_m,j2,j0,t20);
 multi21 :multer2 port map(i1,ld_m,j2,j1,t21);
 multi22 :multer2 port map(i2,ld_m,j2,j2,t22);
 multi23 :multer2 port map(i3,ld_m,j2,j3,t23);
 multi24 :multer2 port map(i4,ld_m,j2,j4,t24);
 multi25 :multer2 port map(i5,ld_m,j2,j5,t25);
 multi26 :multer2 port map(i6,ld_m,j2,j6,t26);
 multi27 :multer2 port map(i7,ld_m,j2,j7,t27);
 adde2 :ad_der2 port map (t20,t21,t22,t23,t24,t25,t26,t27,
```

```
                    ld_a1,ld_a2,ld_a3,q2 );

multi30  :multer2 port map(i0,ld_m,j3,j0,t30);
multi31  :multer2 port map(i1,ld_m,j3,j1,t31);
multi32  :multer2 port map(i2,ld_m,j3,j2,t32);
multi33  :multer2 port map(i3,ld_m,j3,j3,t33);
multi34  :multer2 port map(i4,ld_m,j3,j4,t34);
multi35  :multer2 port map(i5,ld_m,j3,j5,t35);
multi36  :multer2 port map(i6,ld_m,j3,j6,t36);
multi37  :multer2 port map(i7,ld_m,j3,j7,t37);
adde3  :ad_der2  port map (t30,t31,t32,t33,t34,t35,t36,t37,
                    ld_a1,ld_a2,ld_a3,q3 );

multi40  :multer2 port map(i0,ld_m,j4,j0,t40);
multi41  :multer2 port map(i1,ld_m,j4,j1,t41);
multi42  :multer2 port map(i2,ld_m,j4,j2,t42);
multi43  :multer2 port map(i3,ld_m,j4,j3,t43);
multi44  :multer2 port map(i4,ld_m,j4,j4,t44);
multi45  :multer2 port map(i5,ld_m,j4,j5,t45);
multi46  :multer2 port map(i6,ld_m,j4,j6,t46);
multi47  :multer2 port map(i7,ld_m,j4,j7,t47);
adde4  :ad_der2  port map (t40,t41,t42,t43,t44,t45,t46,t47,
                    ld_a1,ld_a2,ld_a3,q4 );

multi50  :multer2 port map(i0,ld_m,j5,j0,t50);
multi51  :multer2 port map(i1,ld_m,j5,j1,t51);
multi52  :multer2 port map(i2,ld_m,j5,j2,t52);
multi53  :multer2 port map(i3,ld_m,j5,j3,t53);
multi54  :multer2 port map(i4,ld_m,j5,j4,t54);
multi55  :multer2 port map(i5,ld_m,j5,j5,t55);
multi56  :multer2 port map(i6,ld_m,j5,j6,t56);
multi57  :multer2 port map(i7,ld_m,j5,j7,t57);
adde5  :ad_der2  port map (t50,t51,t52,t53,t54,t55,t56,t57,
                    ld_a1,ld_a2,ld_a3,q5 );

multi60  :multer2 port map(i0,ld_m,j6,j0,t60);
multi61  :multer2 port map(i1,ld_m,j6,j1,t61);
multi62  :multer2 port map(i2,ld_m,j6,j2,t62);
multi63  :multer2 port map(i3,ld_m,j6,j3,t63);
multi64  :multer2 port map(i4,ld_m,j6,j4,t64);
multi65  :multer2 port map(i5,ld_m,j6,j5,t65);
multi66  :multer2 port map(i6,ld_m,j6,j6,t66);
multi67  :multer2 port map(i7,ld_m,j6,j7,t67);
adde6  :ad_der2  port map (t60,t61,t62,t63,t64,t65,t66,t67,
                    ld_a1,ld_a2,ld_a3,q6 );

multi70  :multer2 port map(i0,ld_m,j7,j0,t70);
```

```
    end if;
   end process divout;
end divo_behav;

configuration divocon of divo is
  for divo_behav
  end for;
end divocon;
```

STA_S

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

entity sta_s is
 port(ck :in std_logic; cm,cad1,cad2,cad3,cm2,cad4,
      cad5,cad6,co :out std_logic);
end sta_s;

architecture sta_s_behav of sta_s is
 type stat is (sti,st0,st1,st2,st3,st4,st5,st6,st7,st8);
 signal p_sta,n_sta:stat;

begin

   a:process(ck)
     begin

      if  (ck='1') and (ck'event)  then
          p_sta <= n_sta ;
      end if;

     end process a;

   b:process(p_sta)
     begin
      case p_sta is
       when sti =>
        cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
        cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
        cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
        n_sta <= st0;

       when st0 =>
        cm   <= '1' ; cad1 <= '0' ; cad2 <= '0' ;
```

```
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st1;
when st1 =>
 cm   <= '0' ; cad1 <= '1' ; cad2 <= '0' ;
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st2;
when st2 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '1' ;
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st3;
when st3 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
 cad3 <= '1' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st4;
when st4 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
 cad3 <= '0' ; cm2  <= '1' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st5;
when st5 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '1' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st6;
when st6 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '1' ; cad6 <= '0' ; co   <= '0' ;
 n_sta <= st7;
when st7 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '1' ; co   <= '0' ;
 n_sta <= st8;
when st8 =>
 cm   <= '0' ; cad1 <= '0' ; cad2 <= '0' ;
 cad3 <= '0' ; cm2  <= '0' ; cad4 <= '0' ;
 cad5 <= '0' ; cad6 <= '0' ; co   <= '1' ;
 n_sta <= sti;
 end case;
end process b;

end sta_s_behav;
```

```
CONFIGURATION   sta_scon OF sta_s IS
    FOR sta_s_behav
    END FOR;
END sta_scon;
```

DCT2S

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;
ENTITY dct2s IS
 port(i00,i01,i02,i03,i04,i05,i06,i07,
     i10,i11,i12,i13,i14,i15,i16,i17,
     i20,i21,i22,i23,i24,i25,i26,i27,
     i30,i31,i32,i33,i34,i35,i36,i37,
     i40,i41,i42,i43,i44,i45,i46,i47,
     i50,i51,i52,i53,i54,i55,i56,i57,
     i60,i61,i62,i63,i64,i65,i66,i67,
     i70,i71,i72,i73,i74,i75,i76,i77 : IN  std_logic_vector(7 downto 0) ;
     clock:in std_logic;
     q00,q01,q02,q03,q04,q05,q06,q07,
     q10,q11,q12,q13,q14,q15,q16,q17,
     q20,q21,q22,q23,q24,q25,q26,q27,
     q30,q31,q32,q33,q34,q35,q36,q37,
     q40,q41,q42,q43,q44,q45,q46,q47,
     q50,q51,q52,q53,q54,q55,q56,q57,
     q60,q61,q62,q63,q64,q65,q66,q67,
     q70,q71,q72,q73,q74,q75,q76,q77 : out  std_logic_vector(13 downto 0)) ;


END dct2s;

ARCHITECTURE  dct2s_behav of dct2s IS

component dctr
 port(i0,i1,i2,i3,i4,i5,i6,i7: IN  std_logic_vector(7 downto 0) ;
     ld_m,ld_a1,ld_a2,ld_a3 : IN std_logic;
     q0,q1,q2,q3,q4,q5,q6,q7:out std_logic_vector(20 downto 0));
end component;

component dctc
```

```
 port(i0,i1,i2,i3,i4,i5,i6,i7: IN  std_logic_vector(20 downto 0) ;
      ld_m,ld_a1,ld_a2,ld_a3 : IN std_logic;
      q0,q1,q2,q3,q4,q5,q6,q7:out std_logic_vector(33 downto 0));
end component;

component divo
 port (i:in std_logic_vector(33 downto 0);
       ld:in std_logic;
       q:out std_logic_vector(13 downto 0));
end component;

component sta_s
 port(ck:in std_logic;
      cm,cad1,cad2,cad3,cm2,cad4,cad5,cad6,co:out std_logic);
 end component;


signal  t00,t01,t02,t03,t04,t05,t06,t07,
        t10,t11,t12,t13,t14,t15,t16,t17,
        t20,t21,t22,t23,t24,t25,t26,t27,
        t30,t31,t32,t33,t34,t35,t36,t37,
        t40,t41,t42,t43,t44,t45,t46,t47,
        t50,t51,t52,t53,t54,t55,t56,t57,
        t60,t61,t62,t63,t64,t65,t66,t67,
        t70,t71,t72,t73,t74,t75,t76,t77 : std_logic_vector(20 downto 0) ;

signal  h00,h01,h02,h03,h04,h05,h06,h07,
        h10,h11,h12,h13,h14,h15,h16,h17,
        h20,h21,h22,h23,h24,h25,h26,h27,
        h30,h31,h32,h33,h34,h35,h36,h37,
        h40,h41,h42,h43,h44,h45,h46,h47,
        h50,h51,h52,h53,h54,h55,h56,h57,
        h60,h61,h62,h63,h64,h65,h66,h67,
        h70,h71,h72,h73,h74,h75,h76,h77 : std_logic_vector(33 downto 0) ;

signal  ctm,ctad1,ctad2,ctad3,ctm2,ctad4,ctad5,ctad6,cto : std_logic;

begin


  control :sta_s port map(clock,ctm,ctad1,ctad2,ctad3,
                          ctm2,ctad4,ctad5,ctad6,cto);
--do f*At
    r0  :dctr port map(i00,i01,i02,i03,i04,i05,i06,i07,
                       ctm,ctad1,ctad2,ctad3,
                       t00,t01,t02,t03,t04,t05,t06,t07);
```

```
r1  :dctr port map(i10,i11,i12,i13,i14,i15,i16,i17,
                   ctm,ctad1,ctad2,ctad3,
                   t10,t11,t12,t13,t14,t15,t16,t17);
r2  :dctr port map(i20,i21,i22,i23,i24,i25,i26,i27,
                   ctm,ctad1,ctad2,ctad3,
                   t20,t21,t22,t23,t24,t25,t26,t27);
r3  :dctr port map(i30,i31,i32,i33,i34,i35,i36,i37,
                   ctm,ctad1,ctad2,ctad3,
                   t30,t31,t32,t33,t34,t35,t36,t37);
r4  :dctr port map(i40,i41,i42,i43,i44,i45,i46,i47,
                   ctm,ctad1,ctad2,ctad3,
                   t40,t41,t42,t43,t44,t45,t46,t47);
r5  :dctr port map(i50,i51,i52,i53,i54,i55,i56,i57,
                   ctm,ctad1,ctad2,ctad3,
                   t50,t51,t52,t53,t54,t55,t56,t57);
r6  :dctr port map(i60,i61,i62,i63,i64,i65,i66,i67,
                   ctm,ctad1,ctad2,ctad3,
                   t60,t61,t62,t63,t64,t65,t66,t67);
r7  :dctr port map(i70,i71,i72,i73,i74,i75,i76,i77,
                   ctm,ctad1,ctad2,ctad3,
                   t70,t71,t72,t73,t74,t75,t76,t77);

--do  colum

c0  :dctc port map(t00,t10,t20,t30,t40,t50,t60,t70,
                   ctm2,ctad4,ctad5,ctad6,
                   h00,h10,h20,h30,h40,h50,h60,h70);
c1  :dctc port map(t01,t11,t21,t31,t41,t51,t61,t71,
                   ctm2,ctad4,ctad5,ctad6,
                   h01,h11,h21,h31,h41,h51,h61,h71);
c2  :dctc port map(t02,t12,t22,t32,t42,t52,t62,t72,
                   ctm2,ctad4,ctad5,ctad6,
                   h02,h12,h22,h32,h42,h52,h62,h72);
c3  :dctc port map(t03,t13,t23,t33,t43,t53,t63,t73,
                   ctm2,ctad4,ctad5,ctad6,
                   h03,h13,h23,h33,h43,h53,h63,h73);
c4  :dctc port map(t04,t14,t24,t34,t44,t54,t64,t74,
                   ctm2,ctad4,ctad5,ctad6,
                   h04,h14,h24,h34,h44,h54,h64,h74);
c5  :dctc port map(t05,t15,t25,t35,t45,t55,t65,t75,
                   ctm2,ctad4,ctad5,ctad6,
                   h05,h15,h25,h35,h45,h55,h65,h75);
c6  :dctc port map(t06,t16,t26,t36,t46,t56,t66,t76,
                   ctm2,ctad4,ctad5,ctad6,
                   h06,h16,h26,h36,h46,h56,h66,h76);
c7  :dctc port map(t07,t17,t27,t37,t47,t57,t67,t77,
                   ctm2,ctad4,ctad5,ctad6,
```

```
h07,h17,h27,h37,h47,h57,h67,h77);
--div and out

o00 : divo  port map (h00,cto,q00);o01 : divo  port map (h01,cto,q01);
o02 : divo  port map (h02,cto,q02);o03 : divo  port map (h03,cto,q03);
o04 : divo  port map (h04,cto,q04);o05 : divo  port map (h05,cto,q05);
o06 : divo  port map (h06,cto,q06);o07 : divo  port map (h07,cto,q07);

o10 : divo  port map (h10,cto,q10);o11 : divo  port map (h11,cto,q11);
o12 : divo  port map (h12,cto,q12);o13 : divo  port map (h13,cto,q13);
o14 : divo  port map (h14,cto,q14);o15 : divo  port map (h15,cto,q15);
o16 : divo  port map (h16,cto,q16);o17 : divo  port map (h17,cto,q17);

o20 : divo  port map (h20,cto,q20);o21 : divo  port map (h21,cto,q21);
o22 : divo  port map (h22,cto,q22);o23 : divo  port map (h23,cto,q23);
o24 : divo  port map (h24,cto,q24);o25 : divo  port map (h25,cto,q25);
o26 : divo  port map (h26,cto,q26);o27 : divo  port map (h27,cto,q27);

o30 : divo  port map (h30,cto,q30);o31 : divo  port map (h31,cto,q31);
o32 : divo  port map (h32,cto,q32);o33 : divo  port map (h33,cto,q33);
o34 : divo  port map (h34,cto,q34);o35 : divo  port map (h35,cto,q35);
o36 : divo  port map (h36,cto,q36);o37 : divo  port map (h37,cto,q37);

o40 : divo  port map (h40,cto,q40);o41 : divo  port map (h41,cto,q41);
o42 : divo  port map (h42,cto,q42);o43 : divo  port map (h43,cto,q43);
o44 : divo  port map (h44,cto,q44);o45 : divo  port map (h45,cto,q45);
o46 : divo  port map (h46,cto,q46);o47 : divo  port map (h47,cto,q47);

o50 : divo  port map (h50,cto,q50);o51 : divo  port map (h51,cto,q51);
o52 : divo  port map (h52,cto,q52);o53 : divo  port map (h53,cto,q53);
o54 : divo  port map (h54,cto,q54);o55 : divo  port map (h55,cto,q55);
o56 : divo  port map (h56,cto,q56);o57 : divo  port map (h57,cto,q57);

o60 : divo  port map (h60,cto,q60);o61 : divo  port map (h61,cto,q61);
o62 : divo  port map (h62,cto,q62);o63 : divo  port map (h63,cto,q63);
o64 : divo  port map (h64,cto,q64);o65 : divo  port map (h65,cto,q65);
o66 : divo  port map (h66,cto,q66);o67 : divo  port map (h67,cto,q67);

o70 : divo  port map (h70,cto,q70);o71 : divo  port map (h01,cto,q71);
o72 : divo  port map (h72,cto,q72);o73 : divo  port map (h03,cto,q73);
o74 : divo  port map (h74,cto,q74);o75 : divo  port map (h05,cto,q75);
o76 : divo  port map (h76,cto,q76);o77 : divo  port map (h07,cto,q77);
```

```
end dct2s_behav;

configuration dct2scon of dct2s is

  for dct2s_behav
   for all:dctr use entity work.dctr(dctr_bahav);
   end for;
   for all:dctc use entity work.dctc(dctc_bahav);
   end for;
   for control:sta_s use entity work.sta_s(sta_s_behav);
   end for;
   for all:divo use entity work.divo(divo_behav);
   end for;
  end for;

end dct2scon;
```

testbench

```
library ieee;
library exemplar;
use exemplar.exemplar_1164.all;
use ieee.std_logic_1164.all;

entity testdc is
end testdc;

architecture test_behav of testdc is
  component  dct2s
   port(i00,i01,i02,i03,i04,i05,i06,i07,
        i10,i11,i12,i13,i14,i15,i16,i17,
        i20,i21,i22,i23,i24,i25,i26,i27,
        i30,i31,i32,i33,i34,i35,i36,i37,
        i40,i41,i42,i43,i44,i45,i46,i47,
        i50,i51,i52,i53,i54,i55,i56,i57,
        i60,i61,i62,i63,i64,i65,i66,i67,
        i70,i71,i72,i73,i74,i75,i76,i77
        : IN  std_logic_vector(7 downto 0) ;
        clock:in std_logic;
        q00,q01,q02,q03,q04,q05,q06,q07,
        q10,q11,q12,q13,q14,q15,q16,q17,
        q20,q21,q22,q23,q24,q25,q26,q27,
        q30,q31,q32,q33,q34,q35,q36,q37,
        q40,q41,q42,q43,q44,q45,q46,q47,
        q50,q51,q52,q53,q54,q55,q56,q57,
        q60,q61,q62,q63,q64,q65,q66,q67,
```

```
        q70,q71,q72,q73,q74,q75,q76,q77
        : out  std_logic_vector(13 downto 0)) ;
    end component;

    signal tck:std_logic;
    signal ti00,ti01,ti02,ti03,ti04,ti05,ti06,ti07,
        ti10,ti11,ti12,ti13,ti14,ti15,ti16,ti17,
        ti20,ti21,ti22,ti23,ti24,ti25,ti26,ti27,
        ti30,ti31,ti32,ti33,ti34,ti35,ti36,ti37,
        ti40,ti41,ti42,ti43,ti44,ti45,ti46,ti47,
        ti50,ti51,ti52,ti53,ti54,ti55,ti56,ti57,
        ti60,ti61,ti62,ti63,ti64,ti65,ti66,ti67,
        ti70,ti71,ti72,ti73,ti74,ti75,ti76,ti77
        :std_logic_vector(7 downto 0) ;
    signal tq00,tq01,tq02,tq03,tq04,tq05,tq06,tq07,
        tq10,tq11,tq12,tq13,tq14,tq15,tq16,tq17,
        tq20,tq21,tq22,tq23,tq24,tq25,tq26,tq27,
        tq30,tq31,tq32,tq33,tq34,tq35,tq36,tq37,
        tq40,tq41,tq42,tq43,tq44,tq45,tq46,tq47,
        tq50,tq51,tq52,tq53,tq54,tq55,tq56,tq57,
        tq60,tq61,tq62,tq63,tq64,tq65,tq66,tq67,
        tq70,tq71,tq72,tq73,tq74,tq75,tq76,tq77
        :std_logic_vector(13 downto 0) ;


begin
    t_dct:dct2s port map(ti00,ti01,ti02,ti03,ti04,ti05,ti06,ti07,
                         ti10,ti11,ti12,ti13,ti14,ti15,ti16,ti17,
                         ti20,ti21,ti22,ti23,ti24,ti25,ti26,ti27,
                         ti30,ti31,ti32,ti33,ti34,ti35,ti36,ti37,
                         ti40,ti41,ti42,ti43,ti44,ti45,ti46,ti47,
                         ti50,ti51,ti52,ti53,ti54,ti55,ti56,ti57,
                         ti60,ti61,ti62,ti63,ti64,ti65,ti66,ti67,
                         ti70,ti71,ti72,ti73,ti74,ti75,ti76,ti77,
                         tck,
                         tq00,tq01,tq02,tq03,tq04,tq05,tq06,tq07,
                         tq10,tq11,tq12,tq13,tq14,tq15,tq16,tq17,
                         tq20,tq21,tq22,tq23,tq24,tq25,tq26,tq27,
                         tq30,tq31,tq32,tq33,tq34,tq35,tq36,tq37,
                         tq40,tq41,tq42,tq43,tq44,tq45,tq46,tq47,
                         tq50,tq51,tq52,tq53,tq54,tq55,tq56,tq57,
                         tq60,tq61,tq62,tq63,tq64,tq65,tq66,tq67,
                         tq70,tq71,tq72,tq73,tq74,tq75,tq76,tq77) ;

    x:tck <= '0',
             '1' after 100 ns,
             '0' after 200 ns,
```

```
        '1' after 300 ns,
        '0' after 400 ns,
        '1' after 500 ns,
        '0' after 600 ns,
        '1' after 700 ns,
        '0' after 800 ns,
        '1' after 900 ns,
        '0' after 1000 ns,
        '1' after 1100 ns,
        '0' after 1200 ns,
        '1' after 1300 ns,
        '0' after 1400 ns,
        '1' after 1500 ns,
        '0' after 1600 ns,
        '1' after 1700 ns,
        '0' after 1800 ns,
        '1' after 1900 ns,
        '0' after 2000 ns;

ti00 <= "01100001" after 100 ns ;ti01 <= "01100001" after 100 ns ;
ti02 <= "01100001" after 100 ns ;ti03 <= "01100110" after 100 ns ;
ti04 <= "01100110" after 100 ns ;ti05 <= "01100110" after 100 ns ;
ti06 <= "01100110" after 100 ns ;ti07 <= "01100110" after 100 ns ;

ti10 <= "01011000" after 100 ns ;ti11 <= "01011000" after 100 ns ;
ti12 <= "01100001" after 100 ns ;ti13 <= "01100001" after 100 ns ;
ti14 <= "01100001" after 100 ns ;ti15 <= "01100001" after 100 ns ;
ti16 <= "01100110" after 100 ns ;ti17 <= "01100001" after 100 ns ;

ti20 <= "01011000" after 100 ns ;ti21 <= "01011000" after 100 ns ;
ti22 <= "01100001" after 100 ns ;ti23 <= "01011000" after 100 ns ;
ti24 <= "01011000" after 100 ns ;ti25 <= "01100001" after 100 ns ;
ti26 <= "01011000" after 100 ns ;ti27 <= "01011000" after 100 ns ;

ti30 <= "01011000" after 100 ns ;ti31 <= "01011000" after 100 ns ;
ti32 <= "01011000" after 100 ns ;ti33 <= "01011000" after 100 ns ;
ti34 <= "01011000" after 100 ns ;ti35 <= "01011000" after 100 ns ;
ti36 <= "01011000" after 100 ns ;ti37 <= "01011000" after 100 ns ;

ti40 <= "01010000" after 100 ns ;ti41 <= "01011100" after 100 ns ;
ti42 <= "01011000" after 100 ns ;ti43 <= "01011000" after 100 ns ;
ti44 <= "01011000" after 100 ns ;ti45 <= "01011000" after 100 ns ;
ti46 <= "01011000" after 100 ns ;ti47 <= "01011000" after 100 ns ;

ti50 <= "01010000" after 100 ns ;ti51 <= "01011000" after 100 ns ;
ti52 <= "01010011" after 100 ns ;ti53 <= "01011000" after 100 ns ;
ti54 <= "01010010" after 100 ns ;ti55 <= "01011000" after 100 ns ;
```

```
ti56 <= "01011000" after 100 ns ;ti57 <= "01011000" after 100 ns ;

ti60 <= "01010000" after 100 ns ;ti61 <= "01010011" after 100 ns ;
ti62 <= "01010000" after 100 ns ;ti63 <= "01010110" after 100 ns ;
ti64 <= "01011000" after 100 ns ;ti65 <= "01010110" after 100 ns ;
ti66 <= "01010110" after 100 ns ;ti67 <= "01010110" after 100 ns ;

ti70 <= "01010000" after 100 ns ;ti71 <= "01010000" after 100 ns ;
ti72 <= "01010011" after 100 ns ;ti73 <= "01010011" after 100 ns ;
ti74 <= "01010000" after 100 ns ;ti75 <= "01010000" after 100 ns ;
ti76 <= "01010011" after 100 ns ;ti77 <= "01011000" after 100 ns ;

end test_behav;
```