ระบบขอบภาพด้วยอุปกรณ์ฮาร์ดแวร์

EDGE DETECTION VIA FPGA

โดย

นางสาวอานันตยา ทองเพ็ง รหัวประจำตัว 43015870

นายพรเทพ สงวนถ้อย รหัสประจำตัว 43015880

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขาวิชาเทคโนโลยีโทรคมนาคม ภาควิชาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

EDGE DETECTION VIA FPGA

MISS ANANTA TONGPENG

Mr.PORNTHEP SAHOUNTUY

A THESIS SUBMITTED IN PARTIAL FULFILLMET OF THE

ERQUIREMENT FOR THE DEGREE OF BACHELOR OF THE

TECHNOLOGY ELECTRONIC DEPARTMENT OF INFORMATION

ENGINEERING FACULITY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKABANG

2002

หัวข้อปริญญานิพนธ์	**ระบบขอบภาพด้วยอุปกรณ์ฮาร์ดแวร์**
	Edge detection via FPGA
นักศึกษา	นางสาวอานันตยา ทองเพ็ง รหัสประจำตัว 43015870
	นายพรเทพ สงวนถ้อย รหัสประจำตัว 43015880
อาจารย์ผู้ควบคุมปริญญานิพนธ์	ผศ.ดร.อรรถสิทธิ์ หล่าสกุล
ระดับการศึกษา	ปริญญาอุตสาหกรรมศาสตร์บัณฑิต
	สาขาวิชาเทคโนโลยีโทรคมนาคม
ภาควิชา	วิศวกรรมสารสนเทศ
ปีการศึกษา	2545

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังอนุมัติ
ให้รับปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตร์บัณฑิต

.....

(ผศ.ดร.อรรถสิทธิ์ หล่าสกุล)
อาจารย์ผู้ควบคุมปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	ระบบขอบภาพด้วยอุปกรณ์ฮาร์ดแวร์
นักศึกษา	นางสาวอานันตยา ทองเพ็ง รหัสประจำตัว 43015870
	นายพรเทพ สงวนถ้อย รหัสประจำตัว 43015880
ระดับการศึกษา	ปริญญาอุตสาหกรรมศาสตร์บัณฑิต
	สาขาวิชาเทคโนโลยีโทรคมนาคม
ภาควิชา	วิศวกรรมสารสนเทศ
ปีการศึกษา	2545
อาจารย์ผู้ควบคุมปริญญานิพนธ์	ผศ.ดร.อรรถสิทธิ์ หล่าสกุล

บทคัดย่อ

ในการทำงานของระบบการตรวจสอบอัตโนมัติ โดยใช้ระบบการมองเห็นของเครื่องจักรโดยใช้กล้อง วงจรปิด ในโรงงานอุตสาหกรรม ต่างๆเช่น ระบบตรวจสอบการประกอบแผ่นวงจร การนับของเม็ดยา การตรวจสอบแผ่นเวเฟอร์ในงานสร้างสารกึ่งตัวนำ การตรวจสอบจอภาพของเหลวเป็นต้น นั้นนับมีการใช้งานอย่างกว้างขวาง และเนื่องจากมักจะเป็นการผลิตเป็นจำนวนมาความต้องการระบบการตรวจสอบที่รวดเร็ว และถูกต้องจึงเป็นระบบที่จำเป็น และขบวนการหนึ่งที่เป็นขั้นตอนที่สำคัญของการตรวจสอบนั้นก็คือ การหาขอบภาพเพื่อการประมวลผลภาพในขั้นตัดสินใจต่อไป ดังนั้นในปริญญานิพนธ์ฉบับนี้จึงนำเสนอถึงการออกแบบและสร้างระบบการหาขอบภาพโดยชุ้ปกรณ์ฮาร์แวร์ ซึ่งจะทำงานเร็วกว่าซอร์ฟแวร์มาก อุปกรณ์ฮาร์ดแวร์ที่ใช้งานคือ Field Programmable Gate Array (FPGA) ระบบมีลักษณะเป็นการ์ดใช้งานประกอบกับคอมพิวเตอร์แบบ IBM PC/AT และที่เครื่องที่เหมือนกัน การคำนวณส่วนใหญ่จะเป็นหน้าที่ของ FPGA ดังนั้นจึงทำงานได้อย่างรวดเร็ว และอีกทั้งราคาถูกสามารถที่จะนำไปสู่การใช้งานจริงได้

THESIS TITLE	EDGE DETECTION VIA FPGA
STUDENT	Miss. Anataya Tongpeng ID. 43015870
	Mr. Pornthep Sahountuy ID. 43015880
COURSE	Bachelor of Industrial Technology in Telecommunication
DEPARTMENT	Information Engineering
YEAR	2002
ADVISOR	Assit.Prof.Dr.Attasit Lasakul

ABSTRACT

Nowadays, machine vision system has applied to many application for industrial purpose. An example of those applications are assembly verification, counting and sizing. Semiconductor wafer thickness measurement and liquid crystal display(LCD) inspection etc. The system that provide faster and high – accuracy is require. In this thesis, we purpose designing and construction of those system by implement on Field Programmable Gate Array (PGAs). The system is very small size as input/output-card for IBM PC/AT compatible. Furthermore, the system is easy to use. Low cost and can be implemented for the real application.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงเป็นอย่างดี เนื่องด้วยคำแนะนำและคำปรึกษาที่เป็นประโยชน์อย่างสูงในการทำปริญญานิพนธ์และโครงงานนี้จากท่าน ผศ.ดร.อรรถสิทธิ์ หล่าสกุล ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ทางคณะผู้จัดทำได้รู้สึกซาบซึ้งในความอนุเคราะห์ที่ดีเยี่ยมจากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณพี่วัชรากรณ์ หนูทอง ศูนย์พัฒนาธุรกิจออกแบบวงจรรวม(TIDI) ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

ขอขอบคุณรุ่นพี่ปริญญาโทในทีมงาน DSL lab สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่านที่ให้ความช่วยเหลือ รวมทั้งคำแนะนำต่างๆ

ขอขอบคุณภาควิชาวิศวกรรมสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ประสิทธิประสาทวิชาความรู้ รวมทั้งให้โอกาสในการทำวิทยานิพนธ์นี้ขึ้นมา

สุดท้ายนี้ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ผู้เป็นผู้ให้ทุกสิ่งทุกอย่างเสมอมา

คุณค่า และประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ทางคณะผู้จัดทำขอมอบแด่ผู้มีพระคุณทุกท่าน ไว้ ณ โอกาสนี้

คณะผู้จัดทำปริญญานิพนธ์

สารบัญ

สารบัญรูป

หน้า

บทที่1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปริญญานิพนธ์ฉบับนี้เสนอวิธีการออกแบบการ์ดประมวลผลภาพดิจิตอลสำหรับหาขอบของภาพ ในการประมวลผลภาพด้วย Soft ware จะเกิดปัญหาในเรื่องของความล่าช้ามาก ดังนั้นจึงจำเป็นต้องใช้เครื่องที่มีความเร็วสูง ซึ่งราคาก็ต้องสูงตามมาด้วย ดังนั้นจึงมีการนำเอาการประมวลภาพดิจิตอลสำหรับหาขอบภาพชนิด Sobel ซึ่งปกติแล้วการหาขอบภาพจะเป็นงานพื้นฐานที่มีอยู่ในแทบทุกขบวนการของการประมวลผลภาพ ฉะนั้นหากเราสามารถนำ FPGA มาทำงานในส่วนของการหาขอบภาพ ก็จะเป็นผลดี เนื่องจากสามารถประมวลผลด้วยความเร็วสูงซึ่งสามารถประยุกต์ใช้งานกับข้อมูลภาพซึ่งมีปริมาณมาก เหมาะสำหรับการผลิตในเชิงอุตสาหกรรมทำให้ผลิตภัณฑ์มีราคาถูกลง

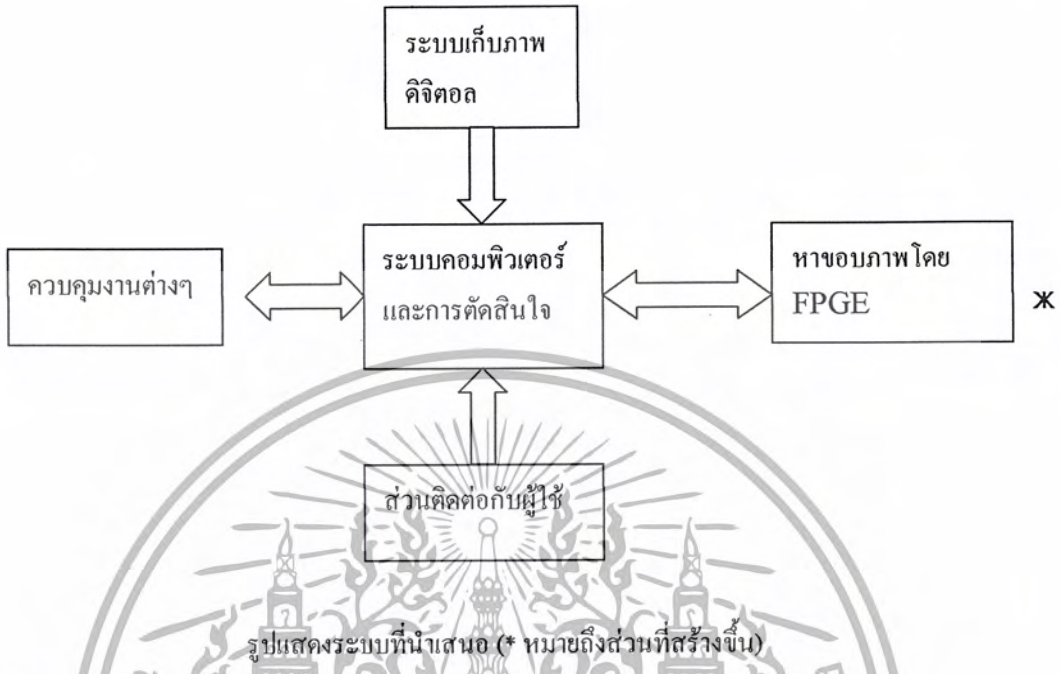
1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้เป็นการเสนอการออกแบบสร้างเครื่องหาขอบภาพโดยในการออกแบบจะใช้ชิฟเอฟพีจีเอเป็นตัวควบคุมกระบวนการทำงานทางดิจิตอลทุกอย่าง ในการออกแบบเครื่องแปลงสัญญาณภาพจากเครื่องเก็บภาพด้วยเอฟพีจีเอทำให้ต้นทุนในการออกแบบต่ำ ในการเก็บข้อมูลภาพสามารถเก็บให้อยู่ในรูปของไฟล์ข้อมูลภาพตามมาตรฐานทั่วไป ซึ่งสามารถนำไฟล์ดังกล่าวไปใช้กับโปรแกรมแสดงภาพหรือประมวลผลภาพที่มีใช้อยู่ทั่วไปได้ โดยมีความละเอียดของสัญญาณเท่ากับ 128 x 128 ระดับ และการเก็บในรูปของไฟล์ที่สามารถเปิดดูได้ด้วยโปรแกรมที่มีใช้อยู่ในปัจจุบัน ความเร็วของการประมวลผลด้วย FPGA ซึ่งมีความเร็วมากกว่า Solf ware ทำให้สามารถนำไปประยุกต์ใช้งานได้จริงและในการออกแบบเครื่องเก็บข้อมูลภาพจากเครื่องเก็บภาพสามารถเลือกขนาดและตำแหน่งของข้อมูลภาพที่จะทำการจัดเก็บได้อีกด้วย

1.3 ขอบเขตของปริญญานิพนธ์

ออกแบบเครื่องหาขอบภาพโดยใช้ชิฟเอฟพีจีเอเป็นตัวควบคุมการทำงานซึ่งเครื่องเก็บข้อมูลภาพจะมีลักษณะเป็นแผงวงจรอิเล็กทรอนิกส์และมีการควบคุมด้วยโปรแกรมบนเครื่องคอมพิวเตอร์โดยที่แผงวงจรอิเล็กทรอนิกส์ที่ออกแบบมาจะมีความสามารถในการหาขอบภาพข้อมูลภาพได้สูงสุด 128X128 จุดต่อภาพ ระดับความเข้มของภาพ 8 บิทต่อภาพ เชื่อมต่อกับ

คอมพิวเตอร์ผ่านทางสล็อต ISA แบบ 16 บิท มีตำแหน่งในการอ้างอิงกับแผงวงจรที่ E000:000 ~ E00:3FFF



รูปแสดงระบบที่นำเสนอ (* หมายถึงส่วนที่สร้างขึ้น)

1.4 เนื้อหาของปริญญานิพนธ์โดยสังเขป

บทที่1 บทนำ

บทที่2 ความรู้พื้นฐานของการประมวลผลภาพ

บทที่3 การหาขอบภาพ

บทที่4 การออกแบบวงจรในลักษณะโครงสร้างและการบรรยายพฤติกรรม

บทที่5 ผลการทดลอง

บทที่6 สรุป

บทที่2
ความรู้พื้นฐานของการประมวลผลภาพ

ในการประมวลสัญญาณด้วยระบบคอมพิวเตอร์ จำเป็นต้องเปลี่ยนข้อมูลหรือสัญญาณที่อยู่ในรูป อนาลอกให้เป็นสัญญาณทางดิจิตอล เพื่อประโยชน์ในการคำนวณและประมวลผลได้ง่าย ในบทนี้จะกล่าวถึงความหมายของภาพในระบบดิจิตอลและคณิตศาสตร์พื้นฐานที่เกี่ยวข้อง

2.1 ความหมายของ Image processing

Image processing เรียกอีกอย่างหนึ่งว่า การประมวลผลภาพ หมายถึง การใช้ขั้นตอน หรือกรรมวิธีใดๆมากระทำกับภาพ โดยมีวัตถุประสงค์ให้ได้ภาพใหม่ที่มีคุณภาพตามที่ต้องการ เช่น ความคมชัดหรือการประหยัดพื้นที่ในการเก็บข้อมูลดังรายการเล่มนี้จะเรียกว่า Image processing ใหม่ว่ากระบวนการประมวลผลภาพ ซึ่งมีเนื้อที่กว้างมากในการศึกษา

2.2 กระบวนการประมวลผลภาพ

การประมวลผลภาพแบ่งออกเป็น 2 ส่วนใหญ่ๆคือ

2.2.1 กระบวนการในโดเมนความถี่(Frequency domain)

กระบวนการในโดเมนความความถี่เป็นการนำการแปลงฟูเรียร์ มาประยุกต์ใช้โดยมีขั้นตอน 3 ตอนดังนี้

ก. การนำภาพมาหาผลการแปลงฟูเรียร์

ข. นำผลการแปลงฟูเรียร์ มาผ่านขั้นตอนการปรับปรุงภาพ

ค. การแปลงฟูเรียร์กลับ

2.2.2 กระบวนการสเปเชียลโดเมน(spatial domain)

สเปเชียลโดเมนจึงเป็นกระบวนการที่กระทำกับจุดภาพโดยตรง อย่างเช่นตัวอย่างของกระบวนการในสเปเชียลโดเมนที่เลือกใช้ได้แก่ฮิสโตแกรมอิควอไลเซชั่น(Histogram Equalization), การบีบอัดไดนามิกเรนจ์(Dinamic range compression) การแปลงระดับสีเทา(Gray sacling)

2.3 ความหมายและนิยามของภาพระบบดิจิตอล

ภาพในเชิงคณิตศาสตร์จะหมายถึง ฟังก์ชั่น 2 มิติ f(x,y) โดย x และ y เป็นแกนพิกัดในระนาบ 2 มิติ ค่าฟังก์ชั่น f(x,y) จะเป็นสัดส่วนกันความสว่างหรือความเข้มของภาพ ที่ตำแหน่ง (x,y) ซึ่งเราเรียกว่า ระดับสีเทา ในรูปที่ 2.1 แสดงให้เห็นถึงระนาบและจุดพิกัดของภาพ ซึ่งปกติเราจะให้จุดกำเนิดของแกนพิกัด อยู่ทางมุมบนซ้ายของภาพ

ภาพ2 มิติที่แทนด้วยฟังก์ชั่นที่จุด (x,y) คือความเข้มของแสงที่จุดนั้น เนื่องจากแสงเป็นพลังงานรูปหนึ่ง ดังนั้น f(x,y) ต้องไม่เป็นศูนย์ และมีค่า (finite)นั่นคือ

$$0 < f(x,y) < \infty \qquad (2.3.1)$$

โดยธรรมชาติของแสง ซึ่งจะต้องมีแหล่งกำเนิดและส่วนที่สะท้อนของแสงดังนั้นเราสามารถแยกฟังก์ชั่น f(x,y) ออกเป็น 2 ส่วนคือ illuminstion component i(x,y) และ reflectant component r(x,y) จะได้ว่า

$$f(x,y) = i(x,y)\, r(x,y) \qquad (2.3.2)$$

เมื่อ

$$0 < I(x,y) < \infty \qquad (2.3.3)$$

และ

$$0 < I(x,y) < \infty \qquad (2.3.4)$$

สมการ (2.1.4) แสดงให้เห็นว่า ฟังก์ชั่นการสะท้อนถูกจำกัดขอบเขตระหว่าง 0 (ซึ่งหมายถึง การดูดซึมโดยสมบูรณ์) และ 1 (ซึ่งหมายถึงการสะท้อนโดยสมบูรณ์)ธรรมชาติของ i(x,y) ขึ้นอยู่กับแหล่งกำเนิดแสง ในขณะที่ r(x,y) ขึ้นอยู่กับวัตถุที่สะท้อนแสงมาเข้าตา

ดังกล่าวมาแล้ว ความเข้มของแสงที่จุด (x,y) เราเรียกว่าระดับสีเทา (λ) จากสมการที่ (2.1.2) ถึง (2.1.4) จะเห็นว่า (λ) ควรอยู่ในช่วง

$$L_{min} \leq (\lambda) \leq L_{max} \qquad (2.3.5)$$

ในทางทฤษฎี L_{min} ต้องมีค่าบวก ในขณะที่ L_{max} ต้องมีค่าน้อยกว่าอนันต์แต่ในทางปฏิบัติ $L_{min} = i_{min} r_{min}$ และ $L_{max} = i_{min} r_{min}$ ช่วงของ (L_{min}, L_{max}) เราเรียกว่า ช่วงของระดับสีเทา ในทางปฏิบัติโดยใช้หลักคณิตศาสตร์ เรานิยมปรับช่วง (L_{min}, L_{max}) ให้เป็นช่วง (0, L) โดย $\lambda=0$ หมายถึงดำสนิท และ $\lambda=L$ หมายถึงขาว

2.4 การแทนภาพด้วยข้อมูลดิจิตอล

ภาพดิจิตอลเป็นภาพที่ถูกแปลงมาจากภาพอนาลอกอยู่ในรูปตัวเลข โดยภาพอนาลอก ถูกแบ่งเป็นพื้นที่สี่เหลี่ยมเล็กๆที่เรียกว่า พิกเซล ในแต่ละพิกเซล จะถูกระบุตำแหน่งโดย (x,y) และค่าระดับสีเทาของพิกเซลนั้น คือค่า ของ f(x,y)

2.5 ระบบการประมวลภาพทางดิจิตอล

ระบบกรประมวลภาพทางดิจิตอลประกอบด้วย 3 ส่วนใหญ่ๆ คือ

1. การเปลี่ยนสัญญาณลอกเป็นสัญญาณดิจิตอล หรือที่เรียกว่า ดิจิไทเซอร์(Digitizer)
2. ส่วนประมวลผล (Processing)
3. ส่วนแสดงผล(Display)

ดิจิไทเซอร์ ทำหน้าที่เปลี่ยนสัญญาณภาพให้เป็นตัวเลข เพื่อป้อนข้อมูลให้กับดิจิตอลคอมพิวเตอร์ อุปกรณ์ในส่วนนี้ได้แก่ กล้องโทรทัศน์ดิจิไทเซอร์

ส่วนประมวลผล คือดิจิตอลคอมพิวเตอร์ ซึ่งอาจใช้ขนาดตั้งแต่ไมโครคอมพิวเตอร์ จนถึงเมนเฟรมคอมพิวเตอร์เนื่องจากภาพที่ถูกดิจิไทซ์ จะได้ข้อมูลที่เป็น array ขนาดใหญ่ซึ่งยากต่อการที่เก็บไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์

ส่วนแสดงผล คือ เปลี่ยนข้อมูลตัวเลขที่เก็บเป็น array ในคอมพิวเตอร์ ให้อยู่ในรูปที่เหมาะสม

2.6 การสุ่มแบบสม่ำเสมอและควอนไทเซชั่น

เพื่อที่จะประมวลสัญญาณภาพด้วยระบบคอมพิวเตอร์ ฟังก์ชั่นของภาพ f(x,y) จะถูกทำให้เป็นสัญญาณไม่ต่อเนื่อง ทั้งระนาบของภาพ ซึ่งเราเรียกว่า Image sampling ค่าของฟังก์ชั่นที่ได้เรียกว่า gray – level quantization

สมมติว่าสัญญาณภาพต่อเนื่อง f(x,y) ถูกดิจิไทซ์ ในระนาบ x – y เป็นช่วงเท่าๆกัน เราสามารถจัด f(x,y) ในรูปเมตริกซ์ขนาด NxN ได้ดังสมการ (2.4.1)

$$f(x,y) = \begin{vmatrix} F(0,0) & f(0,1) & .... & f(0,N-1) \\ F(1,0) & f(1,1) & .... & f(1,N-1) \\ . & & & . \\ . & & & . \\ F(N-1,0) & f(N-1,1) & .... & f(N-1,N-1) \end{vmatrix} \quad (2.6.1)$$

ทางขวามของสมการ จะเรียกว่า ภาพดิจิตอล และทุกๆ สมาชิกของเมตริกซ์จะเรียกว่า พิกเซล จากขบวนการสร้างภาพดิจิตอลข้างต้น จะเห็นว่า เราต้องการทราบขนาดของภาพ NxN พิกเซล และจำนวนระดับของระดับสีเทา ในทางปฏิบัติการทำควอท ไทเซชั่นในระบบภาพดิจิตอลจะเป็นค่าของ 2 ยกกำลังจำนวนเต็ม คือ

$$N = 2^n \quad (2.6.2)$$

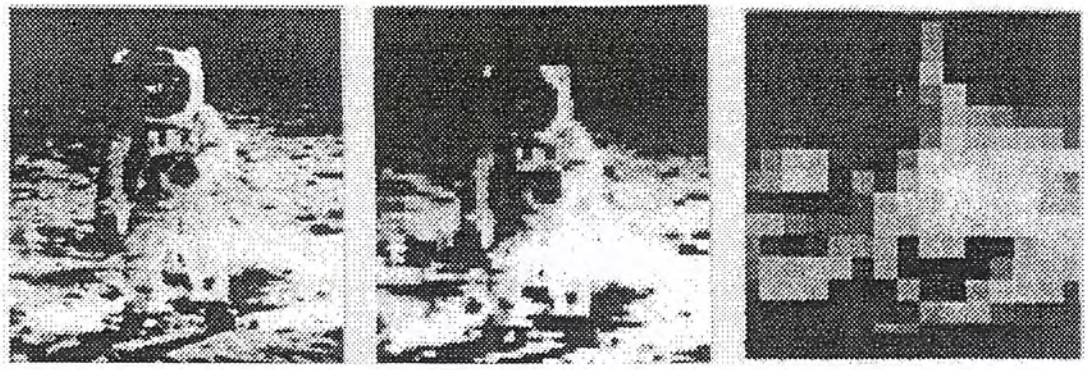
และ

$$G = 2^m \quad (2.6.3)$$

เมื่อ G คือ จำนวนระดับของสีเทา ดังนั้นจำนวน bit (b) ที่ใช้เก็บภาพหนึ่งภาพที่ถูกดิจิไทซ์ คือ

$$B = N \times N \times m \quad \text{บิท} \quad (2.6.4)$$

ตัวอย่างภาพขนาด 128 x 128 พิกเซล และระดับสีเทาจำนวน 64 ระดับ ต้องใช่หน่วยความจำขนาด 98.304 บิท ในรูปที่ 2.1 ได้แสดงการเปรียบเทียบภาพ เมื่อลดความละเอียดของภาพลง



รูปที่ 2.1 เปรียบเทียบเมื่อลดความละเอียดของภาพลง

2.7 เทคนิคต่างๆสำหรับ image processing

สามารถแบ่งออกเป็น 4 พวกใหญ่ๆ คือ

1. image digitization
2. image enhancement and restoration
3. image encoding
4. image reconstruction

2.7.1 image digitization

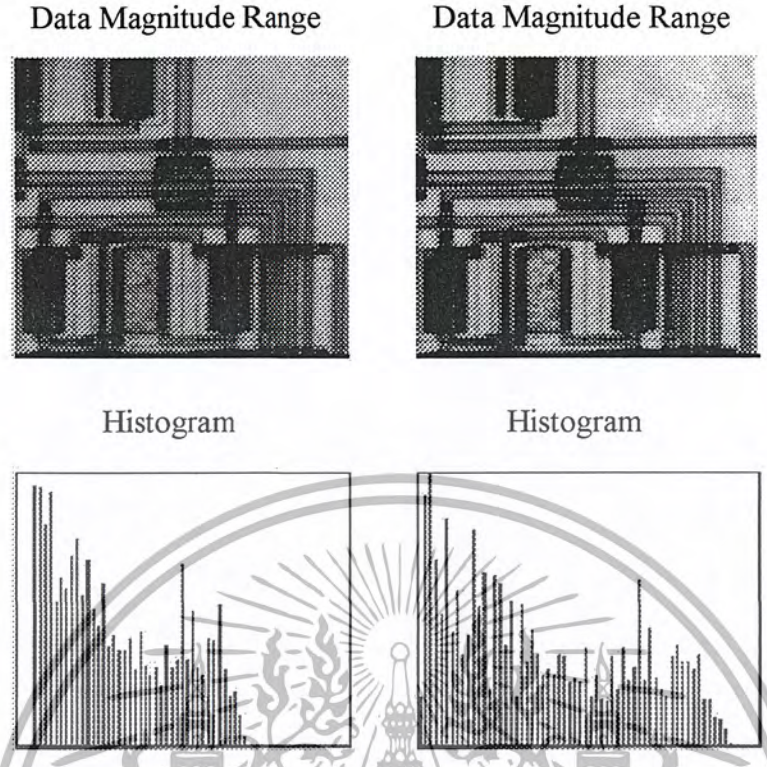
ดังได้กล่าวมาแล้วถึงความหมายของการดิจิไทซ์ภาพ ซึ่งขนาดความละเอียดของภาพที่ได้ก็ขึ้นอยู่กับการควอนไทซ์ภาพ ซึ่งในปัจจุบันเครื่องมือที่ใช้ทำขบวนการนี้ที่เรียกว่า digitizer มีราคาแพง หากไม่มีการใช้งานที่คุ้มค่า ก็ควรจะทำภาพด้วยวิธีจำลองภาพเอา

2.7.2 image enhancement and restoration

image enhancement เป็นการทำภาพให้อยู่ในรูปที่เหมาะสมขึ้น สำหรับงานเฉพาะอย่าง กล่าวคือ วิธีที่ทำภาพ หรือปรับปรุงภาพเอกซ์เรย์อาจจะไม่เป็นวิธีที่ดีเมื่อนำมาปรับปรุงภาพถ่ายดาวเคราะห์ที่ส่งมาจากการสำรวจทางอวกาศ

วิธีปรับปรุงคุณภาพของภาพ(enhancement) มีดังนี้

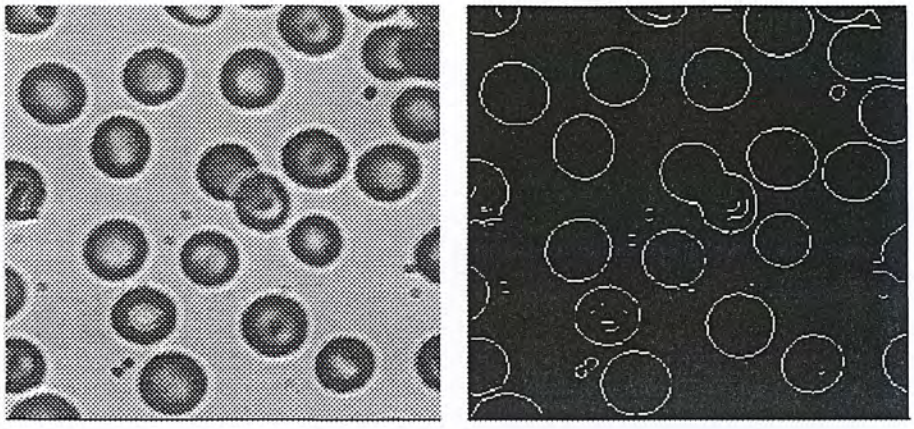
1. Contrast enhancement เป็นวิธีที่ทำให้ภาพคมชัดขึ้น โดยอาศัยฮิสโตรแกรม อาจใช้แบบ Linrae stretch , piecewise linear stretch หรือ equalization



รูปที่ 2.2 การแสดง contrast enhancement จากภาพที่มีการกระจายของ gray tone ต่ำซึ่งภาพจะมืด ให้มีการกระจายสูงขึ้น จะได้ภาพที่ชัดเจน

2. pseudo-coler image processing เป็นการใช้เทคนิคของการทำdensity slicing และการใส่สีเทียมให้กับภาพขาวดำ ที่มีระดับสีเทาต่างๆกัน

3. Edge enhancement เป็นการแยกความแตกต่างของจุดภาพที่ใกล้เคียงกัน เพื่อหาขอบเขตของภาพ

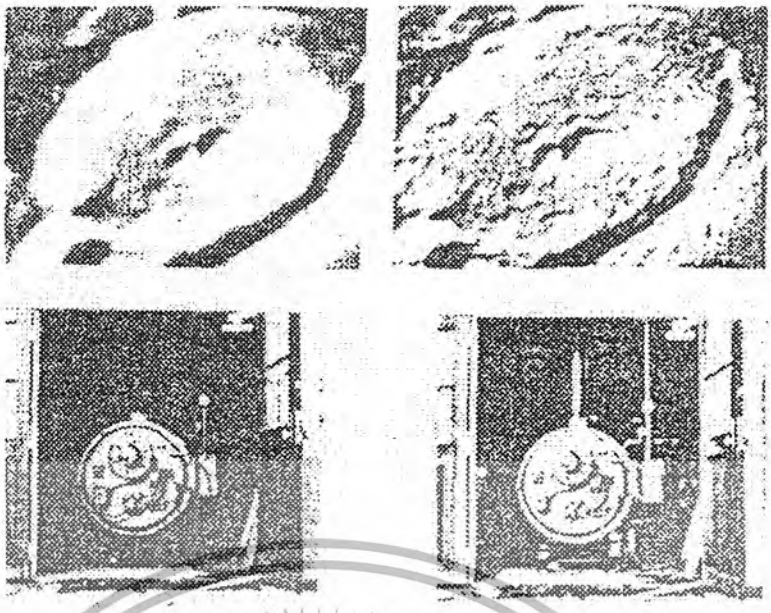


(a)

(b)

รูปที่ 2.3 แสดงการ contrast enhancement เพื่อให้ได้ลักษณะโครงร่างของภาพออกมา 2.3 a) สามารถใช้ edge enhancement นับจำนวนเม็ดเลือดในภาพได้ 2.3 b) แสดงโครงร่างของเมล็ดข้าวได้ชัดเจนขึ้น

4. Filtering เพื่อให้ภาพ smoothing หรือ sharpening โดยใช้ Low pass filter หรือ high pass filter



รูปที่ 2.4 แสดงการทำ filtering enhancement

a) แสดงการทำ sharpening ของพื้นผิวดวงจันทร์ซึ่งได้ภาพคมชัดขึ้น

b) แสดงการทำ high pass filter กับภาพเพื่อให้แสดงรายละเอียดภายในห้องได้ชัดเจนขึ้น

Image restoration เป็นขบวนการในการสร้างภาพกลับคืน โดยการหาค่าชดเชย และแก้ความคลาดเคลื่อน เนื่องจากข้อมูลในภาพผิดพลาดไปหรือเป็นขบวนการสร้างภาพกลับคืน จากภาพที่ถูกทำให้เสียไป เนื่องจากปรากฏการต่างๆโดยใช้หลักการของ linear algebra

2.7.3 image encodeing

เป็นการใช้เทคนิคต่างๆเพื่อเข้าโค้ดข้อมูล เนื่องจากข้อมูลภาพที่ได้จะถูกเก็บในลักษณะเป็นจำนวนไบท์ ซึ่งถ้าภาพมีขนาดใหญ่ ก็ต้องใช้พื้นที่ในการเก็บมากด้วยข้อจำกัดของเครื่องไมโครคอมพิวเตอร์ ที่มีขนาดหน่วยความจำจำกัด การเข้าโค้ดข้อมูลจึงมีประโยชน์ ในด้านการลดพื้นที่ในการเก็บข้อมูลภาพดังกล่าวมากผลของการเข้าโค้ดข้อมูลนี้เรียกว่าเป็น Data reduction ซึ่งเป็นเนื้อหาที่ทำใน Projeact นอกจากนี้การเข้าโค้ดยังมีประโยชน์ในการช่วยลดปริมาณข้อมูลภาพที่ใช้ในระบบการสื่อสาร เช่น การส่งภาพถ่ายจากอวกาศมายังโลก เป็นต้น

2.7.4 image Reconstruction

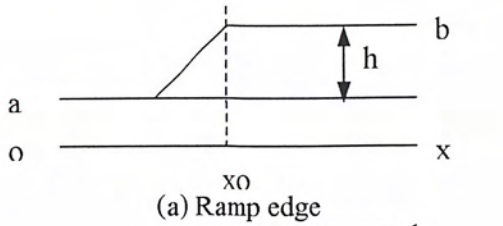
เป็นวิธีการสร้างภาพของวัตถุ โดยไม่ต้องผ่า หรือทำลายวัตถุ เพื่อประมวลผลโดยใช้คอมพิวเตอร์ เราเรียกการสร้างภาพตัดขวางด้วยคอมพิวเตอร์ว่า Computer tomography

บทที่3
การหาขอบภาพ

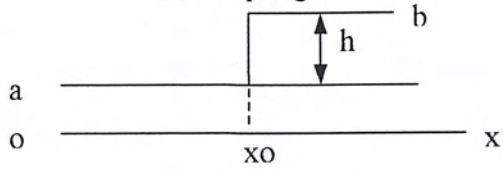
ขบวนการหรือขั้นตอนที่สำคัญอย่างหนึ่งของการประมวลผลภาพคือการหาขอบภาพเพื่อนำมาปรับปรุงภาพ ซึ่งคำว่าขอบในที่นี้จะหมายถึงจุดที่มีระดับความเข้ม(Gray-level) เปลี่ยนแปลงอย่างเห็นได้ชัด และจุดที่มีการเปลี่ยนแปลงนี้จะเป็นกรอบของรายละเอียดในส่วนต่างๆของภาพซึ่งขอบที่กล่าวมานนี้สามารถเกิดขึ้นได้ทุกทิศทางและอาจมีความเข้มไม่เท่ากันก็ได้ส่วนที่เป็นขอบนี้จะแสดงสเปคตรัมที่ความถี่สูงเท่านั้น

3.1 รูปแบบของขอบภาพ(Edge Model)

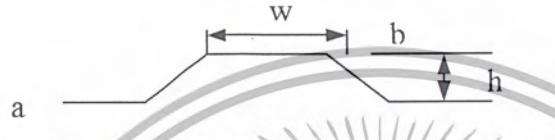
ในรูปที่ 3.1 (a) แสดงรูปร่างของขอบในหนึ่งมิติซึ่งแสดงในขอบเขตที่ต่อเนื่อง(continuous domain) โดยขอบที่มีลักษณะดังรูปนี้จะเรียกว่า แรมป์(ramp) ซึ่งค่าของมันจะเพิ่มขึ้นตามระดับแอมปลิจูดของภาพจากต่ำไปสูง สิ่งที่สามารถบ่งบอกลักษณะของขอบคือ ความสูง ความชัน(slop) มุม(angle)และพิกัดในแนวนอน ณ จุดกึ่งกลางของความชัน ถ้าหากค่าของความชันของมุมมีค่าเท่ากับ 90 องศา จะเรียกขอบชนิดนี้ว่า ขอบแบบขั้นบันได(step edge) แสดงได้ดังรูปที่ 3.1(b) ซึ่งในระบบดิจิตอลจะนำขอบแบบขั้นบันไดไปใช้ในการสร้างภาพอิมเมจชนิดที่เรียกว่า มนุษย์เป็นผู้สร้างขึ้น(Artifical image) ตัวอย่างเช่น แบบการทดลอบ(Test pattern) และข้อมูลในทางกราฟฟิก(bi level graphic data)และในขอบเขตที่มีความต่อเนื่องนี้รูปแบบของภาพในระบบสองมิติจะถือว่าแอมปลิจูดที่มีลักษณะไม่ต่อเนื่องนั้นเป็นค่าคงที่ในจุดข้างเคียงทั้งแปดจุด(neighborhood orthogonal) ในรูปที่ 3.2(a) แสดงรูปร่างของขอบภาพในระบบสองมิติ นอกเหนือจากพารามิเตอร์ในระบบหนึ่งมิติแล้วในระบบสองมิตินี้ยังประกอบด้วยสิ่งที่สำคัญคือ ความชันของขอบที่มีความสัมพันธ์กับแนวแกนและในรูปที่ 3.2(b) แสดงทางเดินของขอบของวัตถุที่มีรูปร่างแปดเหลี่ยมที่มีค่าแอมปลิจูดสูงกว่าบริเวณที่เป็นพื้นที่(back ground) และในรูปที่ 3.3 แสดงลักษณะของขอบที่มีรูปแบบเป็นแรมป์และยูนิต(unit) ในขอบเขตที่ไม่ต่อเนื่องจากรูปจะพบว่ารูปแบบของขอบแบบแรมป์ในแนวตั้งจะประกอบด้วยพิกเซล(pixel) ที่มีค่าเดียวกันซึ่งค่าแอมปลิจูดของมันมีค่าเท่ากับค่ากลางโดยเฉลี่ยของจุดข้างเคียง จากรูปที่แสดงให้เห็นจะมีขอบแบบแรมป์ที่อยู่ในแนวทแยงอยู่ 2 แบบ คือ ซิงเกิลพิกเซลลทรานส์ซชัน(single pixel transition) ซึ่งจุดที่เป็นขอบเขตอยู่ระหว่างแอมปลิจูดสูงกับแอมปลิจูดต่ำและอีกแบบหนึ่งเรียกว่า สมูททรานส์ชิชัน(smoothed transition) ซึ่งวิธีการหาขอบจะหาจากการหาค่าเฉลี่ยใน 2*2 จุดจากทุกๆจุดของขอบในแนวทแยง



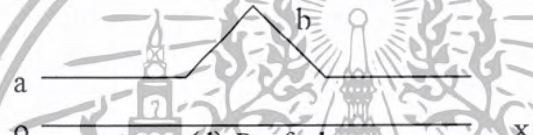
(a) Ramp edge



(b) Step edge



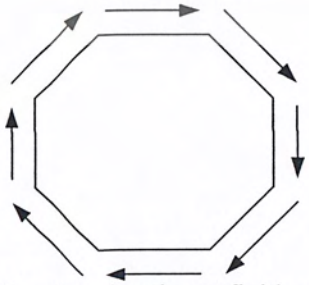
(c) Line



(d) Roof edge

รูป 3.1 แสดงรูปแบบของขอบใน 1 มิติ

(a) edge model



(d) edge direction cefinition

รูป 3.2 แสดงรูปแบบของขอบใน 2 มิติ

a a a a a b b b b b	a a a b b b b b b b	a a a a a a a a a a
a a a a a b b b b b	a a a a b b b b b b	a a a a a a a a a a
a a a a a b b b b b	a a a a a b b b b b	a a a a a b b b b b
a a a a a b b b b b	a a a a a a a b b b	a a a a a b b b b b
a a a a a b b b b b	a a a a a a a a b b	a a a a a b b b b b
Vertical step edge	Digonal step edge	Corner step edge

a a a a c b b b b b	a a c b b b b b b b	a a a a a a a a a a
a a a a c b b b b b	a a a c b b b b b b	a a a a c c c c c c
a a a a c b b b b b	a a a a c b b b b b	a a a a c b b b b b
a a a a c b b b b b	a a a a a c b b b b	a a a a c b b b b b
a a a a c b b b b b	a a a a a a c b b b	a a a a c b b b b b
Vertical ramp edge	Digonal ramp edge	Corner step edge

a a a a c b b b b b	a a d e b b b b b b	a a a a a a a a a a
a a a a c b b b b b	a a a d e b b b b b	a a a a d c c c c c
a a a a c b b b b b	a a a a d e b b b b	a a a a c b b b b b
a a a a c b b b b b	a a a a a d e b b b	a a a a c b b b b b
a a a a c b b b b b	a a a a a a d e b b	a a a a c b b b b b
Vertical ramp edge	Digonal ramp edge	Corner step edge

SMOOTHED TRANSITION

รูปที่ 3.3 แสดงรูปแบบของขอบในระบบ 2 มิติ ซึ่งอยู่ในขอบเขตที่ไม่ต่อเนื่อง

ในรูปที่ 3.3 นี้จะพบว่าสำหรับขอบแบบขั้นบันได(discrete step edge) และขอบแบบคอนเนอร์แรมป์(coner ramp edge) ตำแหน่งของขอบอยู่ที่มีแอมปลิจูดสูงกว่ารอบข้าง สำหรับขอบแบบซิงเกิลพิกเซลทรานซิชั่นตำแหน่งที่เหมาะสมของขอบจะอยู่ตรงที่จุดที่อยู่ในแนวที่มีการเปลี่ยนแปลงและในแบบของแนวทแยงจะมีจุด 2 จุดที่อยู่ใกล้กันในทิศทางเดินของขอบภาพ ดังนั้นตำแหน่งของขอบที่แท้จริงจะอยู่ตรงจุดที่มีแอมปลิจูดสูงกว่าอีกจุดหนึ่งในระหว่างจุดคู่นั้น และในรูปที่ 3.3 นี้จุดที่เป็นตำแหน่งของขอบภาพจะพิมพ์เป็นตัวเอียง

3.2 การหาขอบภาพ(Edge Detection)

วิธีการเบื้องต้นที่ใช้สำหรับการหาขอบภาพในลำดับแรก(First order Derivative Edge Detection) มีอยู่ 2 วิธีด้วยกัน วิธีแรกคือ การหาเกรเดียนท์(gradient) ในทิศทางมุมฉากคือตามแนวนอนและแนวตั้ง และอีกวิธีเราจะทำการหาเกรเดียนท์ในหลายๆทิศทาง

3.2.1 การสร้างเกรเดียนท์มุมฉาก(Orthogonal Gradient Generation)

ถ้ากำหนดให้G(x,y) คือเกรเดียนท์ใน 1 มิติ และ θ คือมุมที่พิจารณาโดยเทียบกับแนวระดับดังนั้นจะทำการคำนวณหาเกรเดียนท์สำหรับขอบภาพจาก

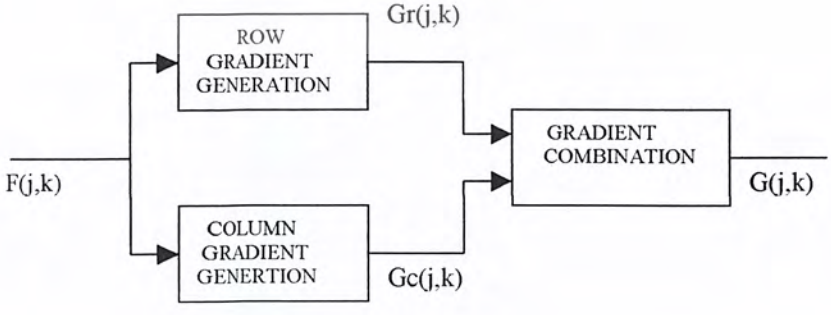
$$G(x,y) = \frac{\partial F(x,y)}{\partial x}\cos\theta + \frac{\partial F(x,y)}{\partial y}\sin\theta \tag{3.1}$$

ในรูปที่ 3.4 แสดงการสร้างเกรเดียนท์ของขอบภาพ[G(j,k)] ในขอบเขตที่ไม่ต่อเนื่อง โดยจะแยกเป็นเกรเดียนท์ของแถว(row edge dradient) [Gr(j,k)] และเกรเดียนท์ของหลัก(column edge gradient) [Gc(j,k)] และค่าอแมปลิจูดของเกรเดียนท์ ซึ่งหมายความถึงความเข้มของภาพแสดงได้ดังนี้

$$G(j,k) = \{[Gr(j,k)]+[Gc(j,k)]\}^{\frac{1}{2}} \tag{3.2}$$

การคำนวณค่าเกรเดียนท์บางครั้งอาจประมาณค่าจาก

$$G(j,k) = |Gr(j,k)| + |Gc(j,k)| \tag{3.3}$$



รูป 3.4 แสดงการหาเกรเดียนท์มุมฉาก

และทิศทางของเกรเดียนท์ตามแนวนอนคือ

$$\theta(j,k) = \tan^{-1}\left\{\frac{Gc(j,k)}{Gr(j,k)}\right\} \tag{3.4}$$

ซึ่งวิธีการที่ง่ายที่สุดสำหรับการสร้างเกรเดียนท์คือ การหาค่าความแตกต่างของความเข้มในแต่ละจุดตามแนวแกนนอน(row) และตามแนวแกนตั้ง(column) ซึ่งเกรเดียนท์ตามแนวแกนนอนหาได้จาก

$$Gr(j,k) = F(j,k) - F(j,k-1) \tag{3.5a}$$

และเกรเดียนท์ในแนวแกนตั้งหาได้จาก

$$Gc(j,k) = F(j,k) - F(j+1,k) \tag{3.5b}$$

โดยเกรเดียนท์นี้จะนำมาใช้หาขอบภาพก็ต่อเมื่อ Gc และ Gr มีค่าเป็นบวกจากซ้ายไปขวาหรือจากล่างขึ้นบนของภาพอิมเมจ

ตัวอย่างเช่นจะหาค่าผลต่างความเข้มโดยใช้เกรเดียนท์ตามแนวแกนนอนของรูปที่ 3.3 เป็นขอบของภาพแบบแรมป์ในแนวดิ่ง(vertical ramp edge model) ได้ดังนี้

0 0 0 0 h 0 0 0 0

โดยที่ h = b-a ซึ่งหมายถึง ความสูงของขั้นบันไดและเกรเดียนท์ในแนวแกนนอนของขอบภาพแบบแรมป์ในแนวดิ่งคือ

0 0 0 0 h/2 h/2 0 0 0

จากค่าข้างบนนี้จะมีค่าที่เท่ากันอยู่ 1 คู่

สำหรับเกรเดียนท์ของขอบในแนวทแยง(diagonal edge gradient) จะหาได้จากผลต่างของจุดแต่ละคู่ตามแนวทแยง ซึ่งในเรื่องนี้จะใช้วิธีการที่เรียกว่า โรเบิร์ตดิฟเฟอเรนซ์โอเปอเรเตอร์ (Robert Difference operator)

3.2.2 โรเบิร์ตดิฟเฟอเรนซ์โอเปอเรเตอร์ (ROBERT DIFFERENCE OPERATOR)

สำหรับโอเปอเรเตอร์ชนิดนี้หาได้จาก

$$G(j,k) = \{[G1(j,k)] + [G2(j,k)]\} \tag{3.6a}$$

หรืออาจหาได้จาก

$$G(j,k) = \{[Gr(j,k)]^2 + [Gc(j,k)]^2\}^{\frac{1}{2}} \tag{3.6b}$$

เมื่อ

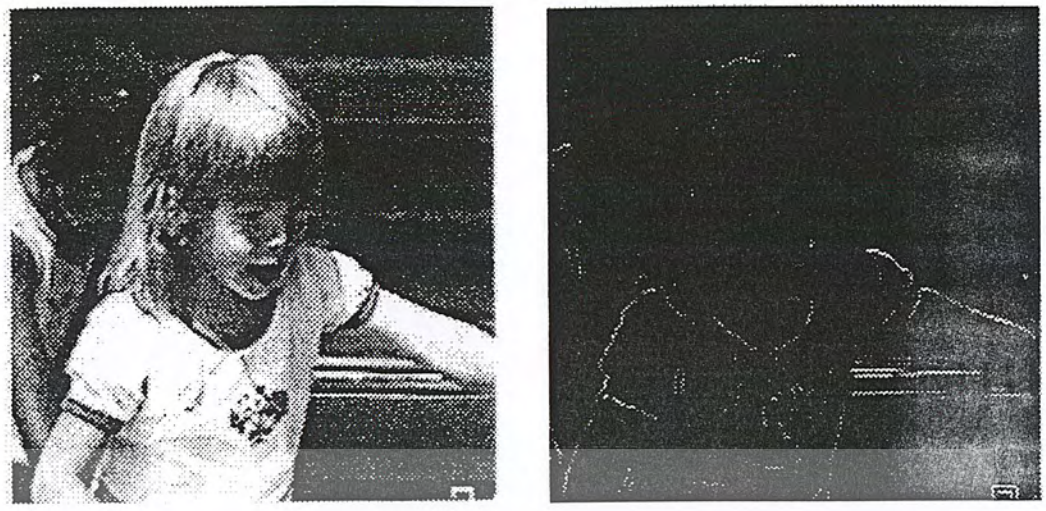
$$G1(j,k) = F(j,k) - F(j+1,k+1) \tag{3.6c}$$

$$G2(j,k) = F(j,k+1) - F(j+1,k) \tag{3.6d}$$

และทิศทางของขอบภาพคือ

$$\theta(j,k) = \frac{\pi}{4} + \tan^{-1}\left\{\frac{Gc(j,k)}{Gr(j,k)}\right\} \tag{3.7}$$

การหาขอบในทิศทางเดียวเช่นนี้จะไม่ค่อยสมบูรณ์นักสำหรับภาพอิมเมจที่มีความส่องสว่างเพียงเล็กน้อย ซึ่งการแก้ปัญหานี้ทำได้โดยการเกรเดียนท์ใน 2 มิติ คือ ทำการหาผลต่างใน 1 ทิศทางแต่จะแสดงค่าออกมาเป็นค่าเฉลี่ยทั้งหมดในทิศทางมุมฉาก



(a) (b)

รูปที่ 3.5 แสดงตัวอย่างของการหาขอบภาพโดยใช้ Robert

(a) ภาพตัวอย่างที่นำมาหาขอบภาพ

(b) แสดงขอบภาพที่ได้จากการใช้ Robert

3.2.3 พรีวิทท์โอเปอเรเตอร์ (Prewitt operator)

A0	A1	A2
A7	F(j,k)	A3
A6	A5	A4

รูป 3.6 ลักษณะของเทมเพลทที่มีขนาด 3X3

โอเปอร์เรเตอร์ชนิดนี้จะใช้เทมเพลท (Template) ที่มีขนาด 3*3 ในการหาเกรเดียนท์โอเปอเรเตอร์ ถ้าสมมุติให้แต่ละจุดมีการเรียงลำดับตามรูปที่ ดังนั้นเกรเดียนท์ของขอบคือ

$$G(j,k) = \left\{ [Gr(j,k)]^2 + [Gc(j,k)]^2 \right\}^{1/2} \tag{3.8a}$$

โดยที่

$$G(j,k) = \frac{1}{k+2}\left[(A2 + KA3 + A4) - (A0 + KA7 + A6)\right] \tag{3.8b}$$

(3.8c)

$$G(j,k) = \frac{1}{k+2}\left[(A0 + KA1 + A2) - (A6 + KA5 + A4)\right]$$

การหาโอเปอเรเตอร์แบบพรีวิทท์ค่า K ที่นำมาใช้มีค่าเท่ากับ 1 เมื่อแทนค่าลงไปในสูตรข้างต้นของเกรเดียนท์ในแนวนอนและแนวตั้ง สุดท้ายจะได้ยูนิตเกน (unit gain) ที่มีค่าเป็นบวกและลบของค่าเฉลี่ยความเข้มเพื่อหาตำแหน่งของขอบภาพ

(a) (b)

รูปที่3.7 แสดงตัวอย่างของการหาขอบภาพโดยใช้ Prewitt
(a) ภาพตัวอย่างที่นำมาหาขอบภาพ
(b) แสดงขอบภาพที่ได้จากการใช้ Prewitt

(a) (b)

รูปที่3.8 แสดงตัวอย่างของการหาขอบภาพโดยใช้ Prewitt
(a) ภาพดาวเสาร์ที่นำมาหาขอบภาพ
(b) แสดงขอบภาพของดาวเสาร์ที่นำมาหาขอบภาพโดยใช้ Prewitt

3.2.4 โซเบลโอเปอเรเตอร์ (Sobel operator)

การหาขอบภาพของโซเบลนี้จะแตกต่างจากแบบพรีวิทท์ตรงที่ค่าความเข้มของภาพในทิศเหนือ,ใต้,ตะวันออก,ตะวันตก จะมีค่าเพิ่มขึ้นอีกเท่าตัวในที่นี้คือค่า K ในสมการที่ (3.8b) และ (3.8c) มีค่าเท่ากับ 2 ดังนั้นค่าน้ำหนัก (weight) ที่เปลี่ยนไปในแต่ละจุดคือตัวที่แสดงการกระจายของเกรเดียนท์

พรีวิทท์โอเปอร์เรเตอร์จะใช้สำหรับการหาขอบภาพในแนวตั้งและแนวนอนได้ดีกว่าโอเปเรเตอร์แบบโซเบล แต่โซเบลเรเตอร์จะใช้ในการหาขอบภาพในทิศทางแนวทแยงได้ดี การใช้โอเปอเรเตอร์แบบพรีวิทท์และโซเบล จะทำให้ได้ขอบภาพของวัตถุที่มีความชัดเจนกว่าการใช้โรเบิร์ตโอเปอเรเตอร์ ทั้งนี้เพราะว่ามีขนาดของโอเปอเรเตอร์ที่ใหญ่กว่าทำให้การเฉลี่ยค่าความเข้มในส่วนที่มีความส่องสว่างน้อยได้ดีกว่าเกรเดียนท์ในแนวนอนและแนวตั้งของดีแทคเตอร์ (detector) ชนิดต่าง ๆ ที่กล่าวมาข้างต้นจะประกอบด้วยการนำความเข้มรอบ ๆ จุดที่ต้องการหาค่ามารวมกัน ดังนั้นการเดียนท์ในแนวนอนและแนวตั้งจะคำนวณจากความสัมพันธ์ดังนี้คือ

$$Gr(j,k) = F(j,k) * Hr(j,k) \tag{3.9a}$$

$$Gc(j,k) = F(j,k) * Hc(j,k) \tag{3.9b}$$

โดยที่ $Hr(j,k)$ คือ อิมพัลส์เรสปอนส์ในแนวแกนนอน

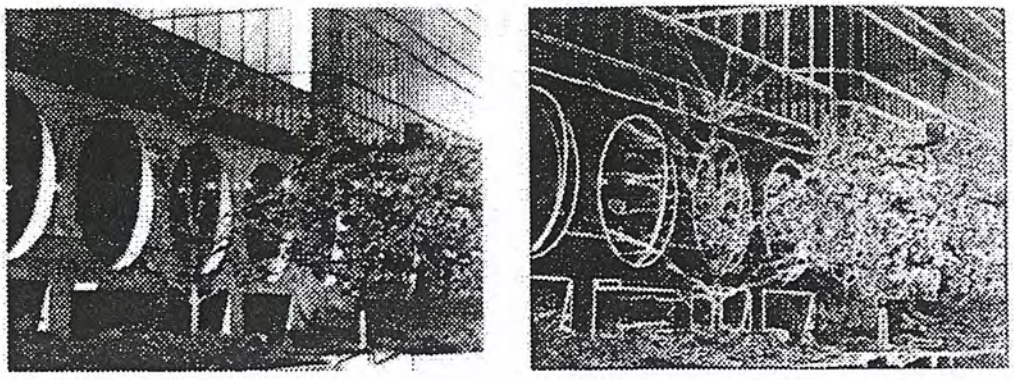
และ $Hc(j,k)$ แนวแกนตั้งของโอเปอเรเตอร์ชนิดต่าง ๆ

operator	Row gradient	Column gradient
Pixel difference	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
Robert	$\begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

operator	Row gradient	Column gradient
Prewitt	$\frac{1}{3}\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$	$\frac{1}{3}\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$
Sobe	$\frac{1}{4}\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$	$\frac{1}{4}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$

รูป 3.9 แสดงอิมพัลส์เรสปอนส์ของเกรเดียนท์โอเปอเรเตอร์ในแนวมุมฉาก

โอเปอเรเตอร์ที่กล่าวมาข้างต้นทั้งหมดนี้มีข้อจำกัดตรงที่จะไม่สามารถ หาขอบภาพได้ถูกต้องแม่นยำในสภาพที่มีการรบกวนสูง(noise) ซึ่งปัญหานี้อาจแก้ไข ได้โดยการใช้โอเปอเรเตอร์ที่มีขนาดใหญ่ขึ้น



(a) (b)

รูปที่ 3.10 แสดงตัวอย่างของการหาขอบภาพโดย Sobel

(a) ภาพตัวอย่างที่นำมาหาขอบภาพ

(b) แสดงขอบภาพที่ได้จากการใช้ Sobel

(c) (d)

รูปที่ 3.11 แสดงตัวอย่างการหาขอบภาพโดยใช้ Sobel

(a) รูปน๊อตและสกลู

(b) แสดงขอบภาพน๊อตและสกรูที่ได้จากการใช้ Sobel

3.2.5 การสร้างเทมเพลทของเกรเดียนท์สำหรับการหาขอบภาพ (Edge Template Gradient)

การหาขอบภาพโดยการใช้เทคนิคของเกรเดียนท์ซึ่งเป็นวิธีที่ทำการหาการเปลี่ยนแปลงความเข้มของภาพในทิศทางมุมฉากคือในแนวแกนนอนและตามแนวแกนตั้ง ดังนั้นวิธีการหาทิศทางของขอบภาพนั้นจะทำได้โดยการคำนวณค่าเวกเตอร์ (vector) จากผลบวกของเกรเดียนท์เหล่านั้น และต่อไปจะคำนวณทิศทางของขอบภาพในหลาย ๆ ทิศทางโดยการคอนโวลูชั่นระหว่างภาพกับเทมเพลทเกรเดียนท์ของอิมพัลส์เรสปอนส์ (template gradient impulse response array) ซึ่งเทมเพลทเหล่านั้นจะได้มาจาก

$$G(j,k) = \text{MAX}\left[\left|G1(j,k)\right|,\ldots,\left|Gm(j,k)\right|,\ldots,\left|GM(j,k)\right|\right] \qquad (3.10a)$$

โดยที่

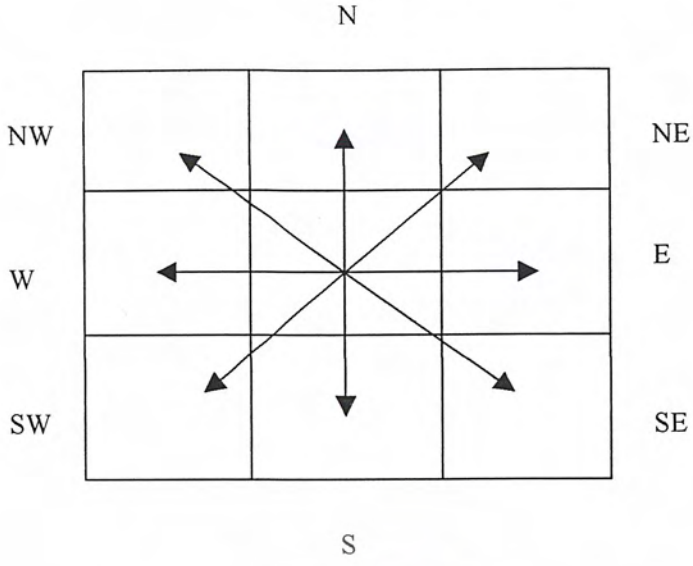
$$Gm(j,k) = F(j,k) * Hm(j,k) \qquad (3.10b)$$

โดยที่ m^{th} คือจำนวนทิศทางในการคำนวณ ดังนั้นเกรเดียนท์จะได้มาจากการคอนโวลูชั่นระหว่างภาพกับ เกรเดียนท์ของอิมพัลส์เรสปอนส์คือ Hm(j,k) สุดท้ายขอบภาพที่ได้จะอยู่ในทิศทางที่มีผลต่างความเข้มมากที่สุด

Gradient direction	Prewitt compass gradient	Kirsch compass gradient
East(H1)	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{pmatrix}$
Northeast(H2)	$\begin{pmatrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{pmatrix}$
North(H3)	$\begin{pmatrix} -1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{pmatrix}$

Gradient direction	Prewitt compass gradient	Kirsch compass gradient
Northwest(H4)	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{pmatrix}$
West(H5)	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{pmatrix}$
Southwest(H6)	$\begin{pmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} -3 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}$
South(H7)	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}$
Southeast(H8)	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{pmatrix}$	$\begin{pmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}$
Scale factor	1/5	1/15

รูป 3.12 เทมเพลทเกรเดียนท์ของอิมพัลส์เรสปอนส์ที่มีขนาด 3x3



รูปที่ 3.13 แสดงการกระจายของทิศทางของภาพ (compass direction)

ในรูปที่ 3.12 แสดงการเกรเดียนท์ 8 ของโอเปอร์เรเตอร์ชนิดต่าง ๆ และในเคริท์ (kirsch) นั้นทิศทางของเกรเดียนท์จะหาได้จาก

$$G(j,k) = \underset{i=0}{\overset{7}{MAX}}\left[5Si - 3Ti\right] \tag{3.11a}$$

โดยที่

$$Si=Ai+Ai+Ai+1+Ai+2 \tag{3.11b}$$

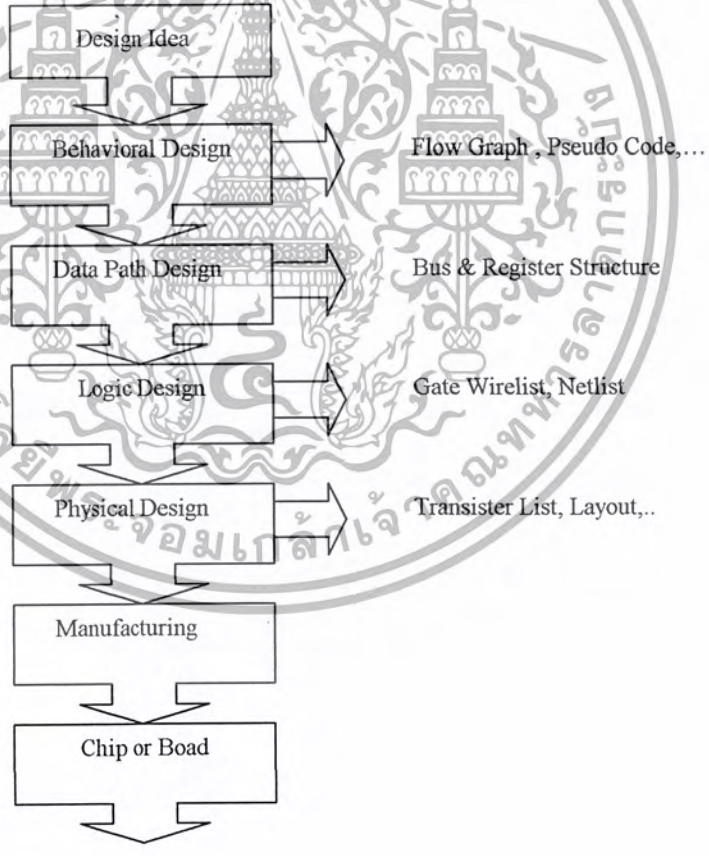
$$Ti=Ai+3Ai+4Ai+5Ai+6Ai+7 \tag{3.11c}$$

และเคิร์ทเกรเดียนท์อาจคำนวณได้จากสมการที่ 3.10 b

บทที่4
การออกแบบวงจรในลักษณะโครงสร้างและการบรรยายพฤติกรรม

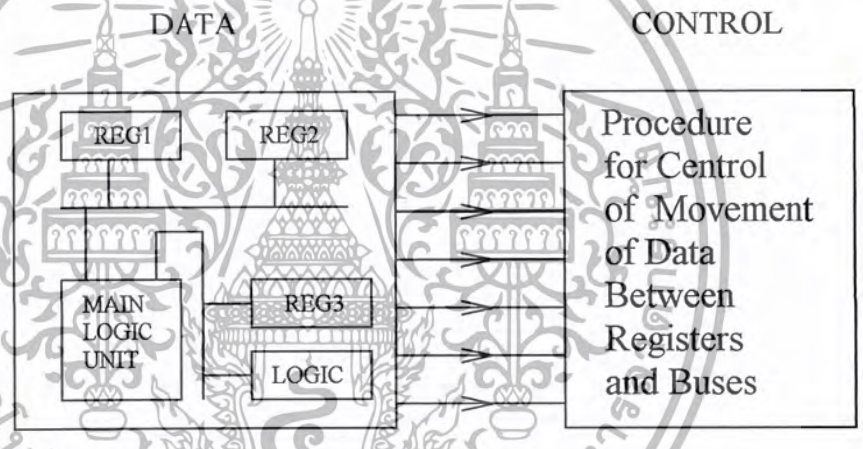
ความซับซ้อนและขนาดของระบบดิจิตอลในปัจจุบันได้เพิ่มมากขึ้นทุกขณะ ส่งผลให้มีการนำคอมพิวเตอร์เพื่อช่วยในการออกแบบหรือ CAD มาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มขึ้นเช่นกัน อีกทั้งอุปกรณ์และวิธีการออกแบบใหม่ๆก็ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบมากขึ้นด้วย สำหรับภาษาบรรยายพฤติกรรมฮาร์ดแวร์ (HDL : Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยให้การปรับปรุงขบวนการออกแบบระบบดิจิตอลเป็นอย่างมีประสิทธิภาพ

4.1 การออกแบบระบบดิจิตอล[7]



รูปที่ 4.1 ขั้นตอนการออกแบบระบบดิจิตอล

ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวรที่ใช้งานได้จะต้องป่านขั้นตอนต่างๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้นตอนก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป รูปที่ 4.1 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบแล้วทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ซึ่งภายในขั้นตอนนี้ผู้ออกแบบ(Flow Graph)หรือรหัสคำสั่งเทียม(pseudo Code)ก็ได้ ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล(Bus) ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์และวงจรลอจิกที่จำเป็นทั้งหมดเพื่อนำมาประกอบเป็นระบบที่สมบูรณ์ โดยแต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบังหนึ่งหรือสองทิศทาง(Unidirectional or Bidirectional Bus)ส่วนกระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรลอจิกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 4.2



รูปที่ 4.2 การออกแบบระบบเส้นทางของข้อมูล

ขั้นตอนถัดมาเป็นการออกแบบวงจรลอจิก ซึ่งจะเกี่ยวข้องกับการนำเอาเกทดิจิตอลพื้นฐานและฟลิปฟลอป(Flip-flop)มาประกอบเป็นอุปกรณ์ย่อยต่างๆ เช่น รีจิสเตอร์เก็บข้อมูล บัสวงจรลอจิก และส่วนควบคุมฮาร์แวร์ ซึ่งผลลัพธ์ที่ได้ในขั้นตอนนี้จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง การออกแบบในขั้นตอนถัดไปเป็นการเปลี่ยนเครือข่ายการโยงใยที่ได้จากขั้นตอนที่แล้วให้เป็นลำดับของทรานซิสเตอร์(Transistor List) และ Layout ซึ่งขั้นตอนนี้จะเกี่ยวข้องโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อแทนเกทและฟลิปฟลอปต่างๆและในขั้นตอนสุดท้ายจะเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

4.2 ประวัติความเป็นมาของภาษา VHDL

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ด แวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และ สามารถเขียนได้หลายรูปแบบ ด้วยเหตุผลนี้จึงทำให้ภาษา VHDL เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกร ได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชันการทำงานของวงจรอย่างสังเขป โดยยังไม่ต้องคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้น VHDL ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มี ความซับซ้อนได้ทั้งหมด ดังนั้น VHDL จึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง

วิวัฒนาการของภาษา VHDL เริ่มต้นประมาณปี ค.ศ. 1981 เมื่อกระทรวงกลาโหมสหรัฐอเมริกา หรือ DoD (Department of Defense) ได้พยายามปรับปรุงอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร ให้มีความทันสมัยมากขึ้น ประกอบกับเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์มีการพัฒนาไปอย่างรวดเร็วดังจะเห็นได้ จากการนำวงจรดิจิตอลหลายๆ วงจรมาทำการผลิตอยู่บนแผ่นซิลิกอนที่มีพื้นที่เพียง 1 - 2 ตารางเซนติเมตรเท่านั้น ซึ่ง เป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้นตลอดจนความน่าเชื่อถือ ในการทำงานและความคงทนต่อสภาพแวดล้อมสูง แต่เนื่องจากในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ ต้องใช้วิศวกร และเวลาในดำเนินการมาก ฉะนั้นทาง DoD จึงจัดตั้งโครงการขึ้นมาเพื่อศึกษาวิธีการที่ช่วยในการพัฒนา วงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น ซึ่งโครงการดังกล่าวมีชื่อว่า "Very High Speed Integrated Circuits" หรือ VHSIC โดยในระยะแรกนั้นโครงการนี้ถือเป็นความลับทาง ด้านความมั่งคงของประเทศ และอยู่ภายใต้ความควบคุมดูแลของ United States International Traffic and Arms Regulations (ITAR) สำหรับมาตรฐานของภาษาที่ใช้บรรยาย พฤติกรรมวงจรหรือฮาร์ดแวร์ของระบบ สำหรับโครงการ VHSIC ที่ DoDได้ให้ไว้สามารถสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่นเดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรม ภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า "Hardware Description Language"หรือHDL

ในตอนเริ่มแรกนั้น DoD ได้มอบหมายให้บริษัทไอบีเอ็ม เท็กซัสอินสตูเมนท์ และอินเตอร์เมทริกซ์ เป็นผู้ศึกษาและพัฒนา โครงการ ซึ่งการดำเนินงานเป็นไปอย่างต่อเนื่อง จนกระทั่งในปี ค.ศ.1985 ทาง ITAR ได้ยกเลิกข้อจำกัดในการถ่ายทอด เทคโนโลยีทางทหารออกจากโครงการนี้ ดังนั้นภาษา VHDL จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป และประมาณปี ค.ศ. 1987 IEEE ได้ทำการกำหนดมาตรฐานของภาษานี้เป็น IEEE 1076-1987 และมีชื่อเรียกว่า VHDL ซึ่งมาตรฐานนี้ได้รับ การปรับปรุงจนเป็นมาตรฐาน IEEE 1076-1993 หรือ VHDL 1993 เนื่องจากในขณะนั้น DoD เป็นลูกค้ารายใหญ่ ของอุตสาหกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ ดังนั้นจึงมีผู้รับโครงการต่างๆ จาก DoD ไปดำเนินการวิจัยและพัฒนา เป็นจำนวนมาก และเพื่อให้ทุกโครงการอยู่ในมาตรฐานเดียวกันหมด ดังนั้นทาง DoD จึงได้กำหนดว่า ทุกๆ โครงการต้อง เขียนอยู่ในรูปของภาษา VHDLเท่านั้น ซึ่งทำให้ DoD สามารถนำโครงการเหล่านี้ไปจำลองกับเครื่องคอมพิวเตอร์ได้ หลายๆระบบ

4.3 ข้อกำหนดของภาษาวีเอชดีแอล

DoD ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ในเดือนมกราคมปี ค.ศ.1983 ไว้ดังนี้

4.3.1 ลักษณะทั่วไป

DoD ได้กำหนดให้ VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถ ในการอธิบายและออกแบบในระดับสูง การจำลอง (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับบนซึ่งก็คือระบบจนถึง ระดับเกทอีกด้วย

เนื่องจากในการทำงานของระบบดิจิตอลนั้น ทุกๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่ จะทำงานไปพร้อมๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งของ VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์นั้นความพร้อมเพรียงจะหมายถึงทุกๆ คำสั่ง องค์ประกอบ เกทหรือวงจรลอจิกจะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในที่สุดแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไป พร้อมๆ กัน)

4.3.2 สนับสนุนการออกแบบแบบลำดับขั้น

การออกแบบแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบระบบที่มีหลายๆ ระดับ โดยในการ ออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อ และส่วนการบรรยายหน้าที่การทำงาน ซึ่งหน้าที่การทำงาน ของระบบสามารถกำหนดได้ด้วยตัวเอง หรืออาจถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆ ลง ไปได้เช่นกัน แต่ที่ระดับล่างสุด องค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเอง และไม่สามารถกำหนด การทำงานโดยลักษณะแบบโครงสร้างได้

4.3.3 ไลบรารี

VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของ อุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูก ต้องควรจัดเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไป ใช้ได้ด้วย

4.3.4 ลำดับคำสั่ง

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษาเองก็ยังมีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบ ที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายใน ของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วย โครงสร้างแบบ case, if - then - else และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

4.3.5 การกำหนดคุณสมบัติ

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่นๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน โดยสิ่งนี้รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้นๆ ด้วย ซึ่งภาษาสำหรับการออกแบบที่ดีควร ให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาด ลักษณะทางกายภาพเวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่นๆ ซึ่งความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

4.3.6 ชนิดของข้อมูล

VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด BIT และ BOOLEAN เท่านั้น แต่ยังสามารถกำหนดชนิด ของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับ (Enumerate Type) หรือแม้แต่ชนิดของ ข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเองก็ได้

4.3.7 โปรแกรมย่อย

ความสามารถในการใช้ฟังก์ชันและโพรซีเจอร์ (Procedure) ก็เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL ซึ่งผู้ออกแบบ สามารถนำโปรแกรมย่อยมาใช้ในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก การกำหนดตัวกระทำต่างๆ หรือหน้าที่อื่นๆ ตามที่ต้องการได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

4.3.8 การควบคุมเวลา

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอย เหตุการณ์ (Event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

4.3.9 การกำหนดแบบโครงสร้าง

การกำหนด โครงสร้างขององค์ประกอบต่างๆ สามารถกระทำได้ในทุกระดับของการออกแบบ โดยการกำหนดโครง สร้างขององค์ประกอบร่วมที่เกิดจากองค์ประกอบย่อยซึ่งแตกต่างกันหรือ เหมือนกันก็เป็นข้อกำหนดอย่างหนึ่งของ VHDL เช่นกัน

4.4 องค์ประกอบพื้นฐานของ VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบของ VHDL จะประกอบไปด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 4.3 โดยในการบรรยายการเชื่อมต่อจะขึ้น ต้นด้วยคำว่า ENTITY แล้วตามด้วยชื่อขององค์ประกอบ จากนั้นตามด้วยคำว่า IS และถัดมาจะเป็นการบรรยายถึงพอร์ต การติดต่อ อินพุท - เอาท์พุท ขององค์ประกอบ ส่วนลักษณะภายนอกอื่น ๆ เช่น เวลา อุณหภูมิก็สามารถรวมเข้าไปในส่วนนี้ ได้เช่นกัน

ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้ บรรยายหน้าที่การทำงานขององค์ประกอบ โดยหน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท เอาท์พุทและพารามิเตอร์ อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 4.3 และสำหรับการบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังจาก คำว่า BEGIN เป็นต้นไป

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

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

รูปที่ 4.3 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

4.4.1 การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ โดยในระดับนี้ต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบ ภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 4.4 ซึ่งเป็นบล็อคไดอะแกรม และการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับตัวจ่าย สัญญาณนาฬิกา

ในบรรทัดแรกของการบรรยายการเชื่อมต่อเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดเป็น clock_ component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็บ ส่วน IN และ OUT เป็นการกำหนด โหนดของสัญญาณให้เป็นอินพุทหรือเอาท์พุท และ BIT เป็นการแสดงชนิดของข้อมูล


```
ENTITY clock_component IS
    PORT (en : IN BIT;ck : OUT BIT)
END clock_name;
```

รูปที่ 4.4 บล็อคไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_ component

4.4.2 การกำหนดรูปแบบการบรรยา

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ ซึ่งในการบรรยายสามารถกำหนดค่าของสัญญาณ เอาท์พุทในเทอมของอินพุทหรือในรูปขององค์ประกอบอื่นๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 4.5 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมโดยมี en เป็นอินพุทและ ck เป็นเอาท์พุท PROCESS เป็นคำที่ใช้ในการเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม และภายในโปรเซสกำหนดให้ periodic เป็นตัวแปรที่มีคำเริ่มต้นเป็น "0" ถ้าสัญญาณ en มีค่าเป็น "1"' จะทำให้ตัวแปร periodic ถูกคอมพลีเมนท์ (complement) และส่งค่าให้กับ ck ซึ่งเป็นสัญญาณเอาท์พุท และสำหรับคำสั่ง WAIT จะเป็นการกำหนดให้สัญญาณมีคาบเวลาเท่ากับ 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS

BEGIN
   PEOCESS
      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 4.5 การบรรยายเชิงพฤติกรรมของ clock _ component

4.4.3 หน่วยการออกแบบแพ็กเก็ต

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบการบรรยายระบบดิจิตอล สามารถเก็บไว้ใน ส่วนของแพ็กเก็ต ซึ่งหน่วยการออกแบบต่างๆ เช่น หน่วยการออกแบบ Entity หน่วยการออกแบบสถาปัตยกรรมหรือ หน่วยการออกแบบแพ็กเกจอื่นๆ สามารถเรียกข้อมูลเหล่านี้ไปใช้ได้ นอกจากนั้นสิ่งที่นิยมทำกันมากคือการนำรูปแบบ มาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐาน (เช่น ไอซีตระกูล 74XX เป็นต้น) มาเก็บไว้ในรูปของแพ็กเกจ ที่ทุกคนสามารถเข้าถึงได้ ตามปกติแล้วแพ็กเกจจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็กเกจ (Package declaration)และ ส่วนของบอดี้แพ็กเกจ (Package body) เนื่องจาก แพ็กเกจถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่ กำลังเขียนอยู่ ฉะนั้นการที่นำแพ็กเกจไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถ กระทำได้ด้วยชุดคำสั่ง USE

4.4.3.1 PACKAGE DECLARATION

ส่วนที่มีความสำคัญที่สุดของแพ็กเกจ (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ ส่วนการประกาศแพ็กเกจ เนื่องจากเป็นส่วนที่ใช้กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็กเกจ สำหรับนำไปใช้ภายนอกตัวของแพ็กเกจเอง ถ้ามี การประกาศสิ่งใดๆ ในส่วนของส่วนบอดี้แพ็กเกจ แต่ไม่ถูกประกาศในส่วนการประกาศแพ็กเกจจะทำให้ค่าและพฤติ กรรมไม่สามารถนำไปใช้งานในส่วนนอกได้ซึ่งเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศ Entity คือ จุดเชื่อมต่อ หรือ พอร์ต ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเกจสามารถสร้างขึ้นได้โดยไม่จำเป็น ต้องมีส่วนบอดี้ และยังสามารถนำไปใช้งานจากรูปแบบภายนอกได้เช่น ใช้สำหรับประกาศ ชนิด (Type) หรือสัญญาณ เช่นเดียวกับ ส่วนบอดี้แพ็กเกจที่ไม่จำเป็นต้องมี ส่วนของการประกาศแพ็กเกจ แต่แพ็กเกจนั้นจะไม่สามารถนำไปใช้จาก รูปแบบอื่นได้

```
PACKAGE package_name IS
    Package_declarative_part
END package_name;
```

รูปที่ 4.6 โครงสร้างทั่วไปของส่วนการประกาศแพ็กเกจ

4.4.3.2

โครงสร้างซึ่งประกอบด้วยลำดับคำสั่งที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อยทั้งหลาย ซึ่งชื่อของโปรแกรมย่อยนั้นๆ ได้ถูกประกาศไปแล้วในส่วนของการประกาศแพ็กเกจ จะถูกเก็บไว้ในส่วนของบอดี้แพ็กเกจ ทั้งนี้รวมถึง การกำหนดค่าคงที่ต่างๆ อันได้แก่ค่าคงที่ที่ถูกประกาศชื่อไว้ก่อนในส่วนของการประกาศแพ็กเกจ และถูกกำหนดค่าใน ส่วนของบอดี้แพ็กเกจ ฉะนั้นในส่วนของบอดี้แพ็กเกจจึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็กเกจไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อย หรือค่าคงที่ การเขียนบอดี้แพ็กเกจนั้นจะเป็นไปตามกฎเกณฑ์ดังแสดงในรูปที่ 4.7

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

รูปที่ 4.7 โครงสร้างของบอดี้แพ็กเกจ

4.4.4 หน่วยการออกแบบ Configuration

ดังที่ทราบกันแล้วว่าระบบดิจิตอลรูปแบบหนึ่งไม่ว่าจะเป็นอะไรก็ตาม จะสามารถมีหน่วยการออกแบบ Entity ได้ เพียงหนึ่งเดียวเท่านั้น ซึ่งในหน่วยการออกแบบ Entity หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลาย หน่วย ดังนั้นจะต้องมีหน่วยการออกแบบ Configuration มาเพื่อกำหนดการใช้ Configuration ของการประกอบ Entity กับหน่วยการออกแบบสถาปัตยกรรมหน่วยใดๆ เข้าด้วยกัน

```
COMFIGURTRION identifier OF entiry_name IS
    Configuretion_declarative_part
END ;
```

รูปที่ 4.8 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

4.4.5 โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆ ไปค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าใช้ฟังก์ชัน แทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรลอจิกจริงๆในขณะที่ถ้าใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูล หรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่ 4.9 แสดงการใช้โพรซีเจอร์ เพื่อเปลี่ยนข้อมูลชนิด 8 บิตเป็นค่าจำนวนเต็ม และรูปที่ 4.10 แสดงการใช้ฟังก์ชันโดยกำหนดให้ X เป็นตัวแปรชนิด บิตแทนการกระทำในสมการบูลีน

```
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
```

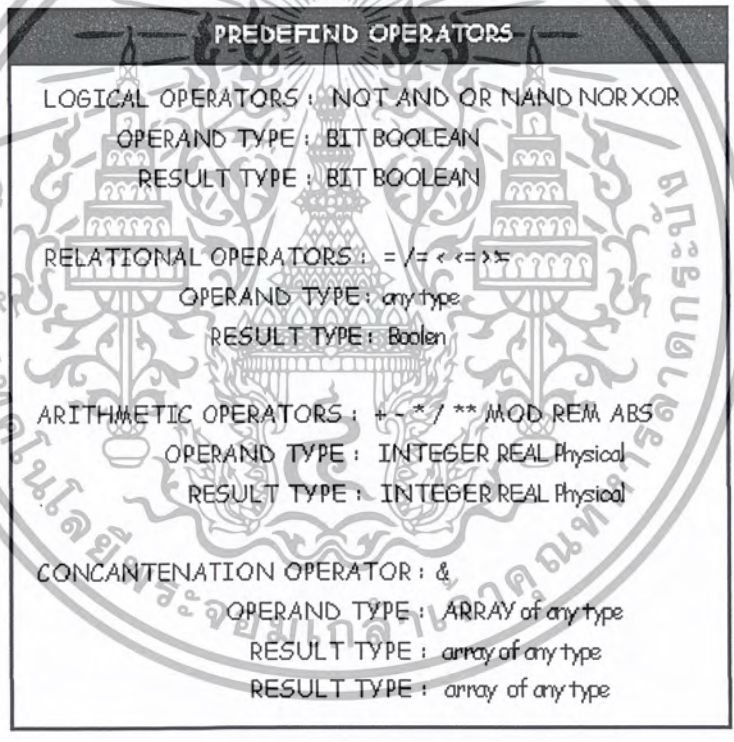
รูปที่ 4.9 การใช้โพรซีเจอร์

```
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 4.10 การใช้ฟังก์ชัน

4.4.6 โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา VHDL มีตัวดำเนินการหรือโอเปอร์เรเตอร์ทางลอจิกและคณิตศาสตร์เช่นเดียว กับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 4.11



รูปที่ 4.11 ตัวดำเนินการใน VHDL

4.4.7 เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์อุปกรณ์ทุกๆ ตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลา เข้ามาเกี่ยวข้องในทุกๆเหตุการณ์ที่เกิดขึ้นเสมอ VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อให้สามารถบรรยายรูป แบบและการพ้องกันของเวลาสำหรับการทำงานของ

อุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วน ของการบรรยายสถาปัตยกรรม จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โปรเซสซึ่งมีการทำงานภายในเป็น แบบลำดับคำสั่งก็ตาม ซึ่งหากมีหลายๆ โปรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆ โปรเซสก็จะทำงานไปพร้อมๆ กันด้วย

4.4.8 สัญญาณและตัวแปร

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยการ กำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการ ส่งผ่านค่าของสัญญาณ เช่น w <= a AFTER 12 NS หมายถึงการกำหนดค่าสัญญาณ a ให้กับ w หลังจากเวลา ผ่านไป 12 นาโนวินาที ในทางตรงข้ามตัวแปรมีลักษณะเป็นเสมือนตัวกลางที่ใช้ในการส่งผ่านข้อมูลและ ไม่มีเรื่องของ เวลาเข้ามาเกี่ยวข้องด้วย ซึ่งตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชัน โพรซีเจอร์ และโปรเซส สำหรับการกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ :=

4.5 การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรม เป็นการบรรยายลักษณะการเปลี่ยนแปลงของ ข้อมูลในรูปแบบของอัลกอริธึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้น ซึ่งสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูล ที่เข้ามาโดยไม่คำนึงถึงลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในว่าจะเป็นอย่างไร ในหัวข้อนี้จะ แสดงถึงการบรรยายเชิงพฤติกรรม แทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่างๆ ที่ควรรู้

4.6 โปรเซส

โปรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดให้กับสัญญาณ โปรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอ และจะปฏิบัติคำสั่งพร้อมๆ กันกับโปรเซสอื่นๆ ที่อยู่ในสถาปัตยกรรมบรรยายเดียวกัน โดยโปรเซสจะปฏิบัติงานตามคำ สั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์กำหนดค่าให้กับสัญญาณ (<=)

การบรรยาย โปรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS ในรูปที่ 4.12 เป็นการแสดงส่วน ประกอบของการบรรยายแบบโปรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้และส่วนของการปฏิบัติ คำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ

รูปที่ 4.12 รูปแบบของการบรรยายแบบโปรเซส

4.7 การกำหนดตัวดำเนินการภายในโปรเซส

ตัวดำเนินการภายในโปรเซสมี 3 ชนิดคือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวดำเนิน การทั้งสามชนิดนี้หากมีการประกาศไว้ในโปรเซสใดก็จะใช้ได้เฉพาะภายในโปรเซสนั้นเท่านั้นสำหรับการติดต่อกับภาย นอกหรือระหว่างโปรเซสสามารถทำได้โดยใช้สัญญาณ (Signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE ในรูปที่ 4.13 แสดงตัวอย่างการประกาศตัวกระทำภายในโปรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN และค่าเริ่มต้นที่ถูกกำหนดให้กับตัวดำเนินการภายในโปรเซสจะถูกนำมาใช้ในตอนเริ่มต้น ของการปฏิบัติเพียงครั้งเดียวเท่านั้น ต่างกับค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้ โปรแกรมย่อยนั้น ๆ

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

รูปที่ 4.13 ตัวอย่างการประกาศตัวดำเนินการภายในโปรเซส

4.8 การกำหนดการกระทำภายในโปรเซส

การกระทำใดๆ ภายในโปรเซสจะเป็นการปฏิบัติแบบลำดับ (Sequential) เสมอ ซึ่งภายในโปรเซสสามารถใช้ประโยค เงื่อนไขหรือการทำซ้ำได้เช่น IF-THEN - ELSE,CASE - WHEN, FOR LOOP และ WHILE LOOP ดังตัวอย่างในรูปที่ 4.14 และ 4.15

```
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 4.14 เงื่อนไขการกระทำในโปรเซส

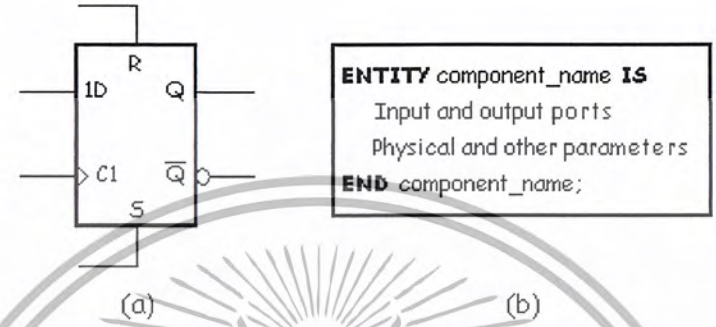
```
ARCHITECTURE demo OF partial_process IS
 ...
BEGIN
  PROCESS
   BEGIN
          ...
          x <= a AFTER 10 NS;
          y <= b AFTER 6 NS;
          ...
  END PROCESS;
END demo;
```

รูปที่ 4.15 แสดงการกระทำในโปรเซส

4.9 การกระตุ้นและยับยั้งการกระทำของโปรเซส

การกระทำภายในโปรเซสจะอยู่ในสภาพเตรียมพร้อม และมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์ เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโปรเซสได้โดยการกำหนดรายการของสัญญาณที่ต้อง การให้โปรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่กำหนดไว้เท่านั้น ส่วนเหตุการณ์ใดๆ ที่เกิดขึ้นกับสัญญาณ ที่ไม่ได้กำหนดไว้ในรายการก็จะไม่ส่งผลให้มีการกระทำภายในโปรเซส ซึ่งรายการของสัญญาณนี้เรียกว่า Sensitivity

List และจะกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS รูปที่ 4.16 (a) แสดงตัวอย่างโมเดล และรูปที่ 4.16 (b) เป็นตัวอย่างการบรรยายการเชื่อมต่อของ D-Flip Flop ส่วนรูปที่ 4.17 แสดงถึงการบรรยายเชิงพฤติกรรมของ D-Flip Flop โดยในรูปที่ 4.17 (a) เป็นการใช้ตัวกระทำภายนอกโปรเซส และรูปที่ 4.17 (b) เป็นการใช้ตัวกระทำภายในโปรเซส โดยมีรายการของสัญญาณ (rst, set, clk) เป็นตัวกระตุ้นการปฏิบัติงานภายในโปรเซส



รูปที่ 4.16 (a) ตัวอย่างโมเดล D-Flip Flop
(b) การบรรยายการเชื่อมต่อของ D-Flip Flop

```
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_dalay_ behavioral OF d_sr_flipflop IS
BEGIN
   dff : PEOCESS (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)

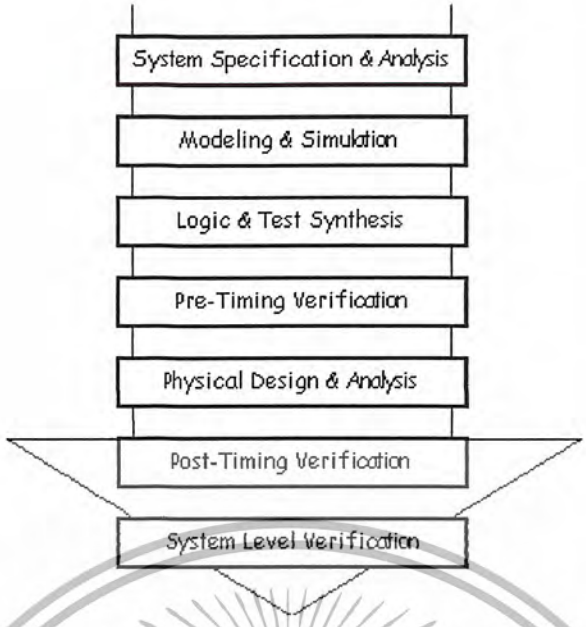
รูปที่ 4.17 การบรรยายเชิงพฤติกรรมของ D-FlipFlop

(a) การใช้ตัวกระทำภายนอกโปรเซส

(b) การใช้ตัวกระทำภายในโปรเซส

4.10 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของ บล็อกไดอะแกรมก่อนที่ทำวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายและวิเคราะห์การ ทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการ นอกจากนี้ยัง สามารถเพิ่มเติมในรายละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่าง (Top - Down Design) นั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom - Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ ก่อนแล้วจึงทำการจำลองการทำงาน และตรวจ สอบความถูกต้อง ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถ ออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 4.18 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 4.18 แสดงถึงขั้นตอนของการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้าง เล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

1.สร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea andConcept) ใน การแก้ปัญหา

2.เขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL หรือ ภาษา HDL อื่น ๆ สำหรับบรรยายพฤติกรรมการ ทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3.หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมา เป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจริง หรือสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับ วงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของ วงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้น หรือ ไม่ก็อยู่ในรูปของ Netlist ที่สามารถนำไปผลิตในอุปกรณ์อื่นได้

4.หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือ Netlist แล้ว ข้อมูลนี้จะถูกใช้สำหรับจำลองการทำงานในเรื่อง ความถูกต้องของฟังก์ชัน พร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาประกอบการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ ทางอิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็น เวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

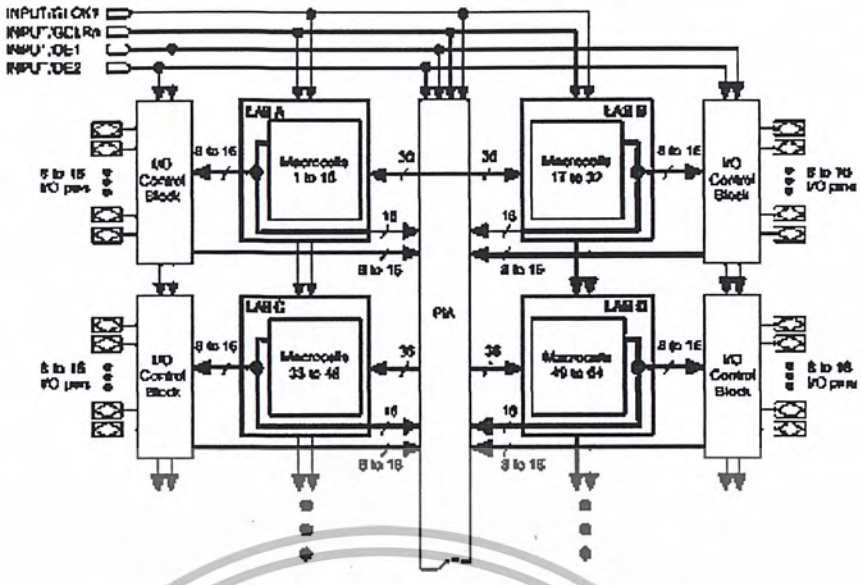
5.ผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจ จะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวมASIC

6.ทำการตรวจสอบการทำงานและตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้ วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

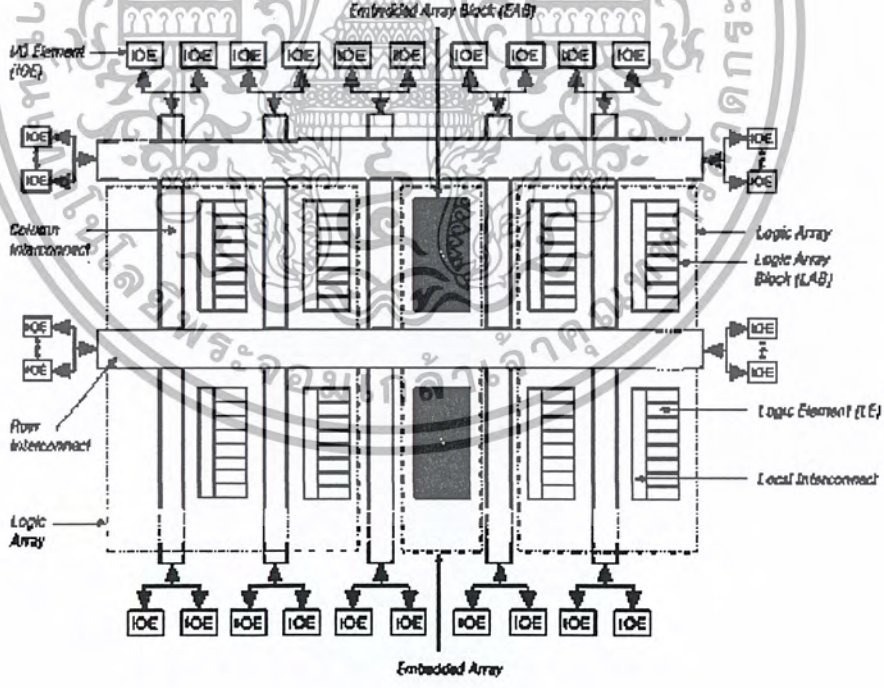
7.นำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบร่วมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

4.11 โครงสร้างภายในของ FPGA

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อคลอจิกที่สามารถทำการโปรแกรมได้ดังรูปที่ 4.19 และ 4.20



รูปที่ 4.19 โครงสร้างภายในของ FPGA ตระกูล MAX7000S



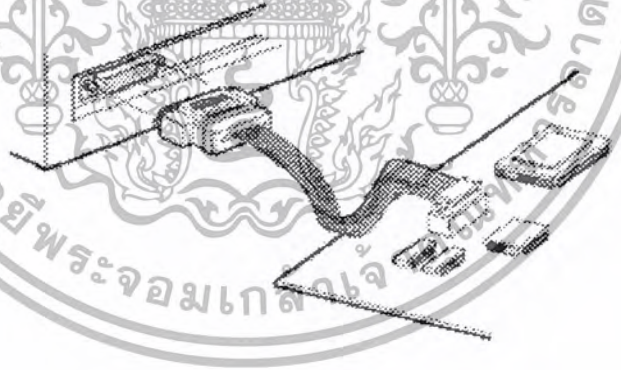
รูปที่ 4.20 โครงสร้างภายในของ FPGA ตระกูล FLEX10K

4.12 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

1.ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออก แบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2.มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language)เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็น ต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตราฐาน เดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3.การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 4.21 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 4.21 การโปรแกรมลงในชิพ

4.13 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ จากที่ได้กล่าวไปแล้วในบทที่ 1 ในกรณีของการออก แบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้วจึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำ ได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจร ใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

4.13.1 การสังเคราะห์วงจร (Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรม ของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์ นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตาม เทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครง สร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบ ด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆจึงทำให้ผล ที่ได้มีประสิทธิภาพและในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดล แต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจร ที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับ เทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำ การสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้น เป็นอย่างไร เช่นมีค่า

ความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้น ตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตาม ที่กำหนด

4.13.2 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อลดความหนาแน่นในตอนทำการเชื่อม ต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำโดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิป-ฟลอบ (flip-flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ในอุปกรณ์ FPGAหลังจากทำขั้นตอน นี้เสร็จแล้วผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนข้อมูลทางเวลานั้นผู้ออกแบบจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความ หน่วงลอจิก(logic delay) ส่วนซอฟต์แวร์จะรวมเอาซอฟต์แวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement & Routing) เป็นไปอย่างต่อเนื่อง

4.13.3 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควร จะอยู่ ณ ตำแหน่งไหน ในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่น วงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะ ได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี ความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณได ไม่หมด การวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กันโดยเฉพาะส่วน ที่มีการเชื่อมต่อสัญญาณด้วยกัน นอกจากนั้นการกำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ก็จะมีผลโดยตรงเลยคือซอฟต์แวร์จะวาง I/O ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่ง บางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขา I/O ควรกำหนดตำแหน่งให้เหมาะสม หรือไม่ก็ให้ซอฟต์แวร์จัดการเอง

4.13.4 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ขั้นตอนนี้จะทำ ต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณ ได้ไม่หมด (เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกิน ค่าที่กำหนดในข้อบังคับ ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือผู้ออกแบบจะทำการ เชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับ ทางเวลา จะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

4.13.5 ความหน่วงด้านเวลา (Delay)

ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วง ที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ

- ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์ FPGA เอง
- ความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay) เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA

โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการ เชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์ และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลาเพื่อให้ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อ ให้ได้ผลลัพธ์ที่ดีขึ้น

ค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความ หน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้น เป็นไปตามข้อกำหนดหรือไม่

4.13.6 การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการ ทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น

Model Sim ของบริษัท Model Technology หรือ Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดล เกิดขึ้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้ เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออก แบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจ สอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจร แล้ว เพื่อจรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตาม ข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง

ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้ จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้อง ตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวมสามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ ซอฟต์แวร์ตัวเดียวกันตอลดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

4.13.7 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement & Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มี ฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับอุปกรณ์ FPGA ของแต่ละบริษัทผู้ผลิตคือ ในกรณีที่เป็นอุปกรณ์ FPGA ชนิดที่ต้องโปรแกรมโดยวิธี SRAM นั้น ในการใช้งานผู้ออกแบบจะต้องเก็บข้อมูลวงจรไว้ในหน่วยความจำประเภท EPROM หรือ serial PROM ด้วยเพื่อจะใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปไม่ ต้องดาวน์โหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีก เพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่แล้ว แต่กรณีที่อุปกรณ์ FPGA เป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Anti fuse ก็ไม่จำ เป็นต้องมีหน่วยความจำสำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูล

วงจรลงไป ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออก แบบไว้ได้เลย

4.14 เครื่องมือสำหรับการออกแบบ FPGA

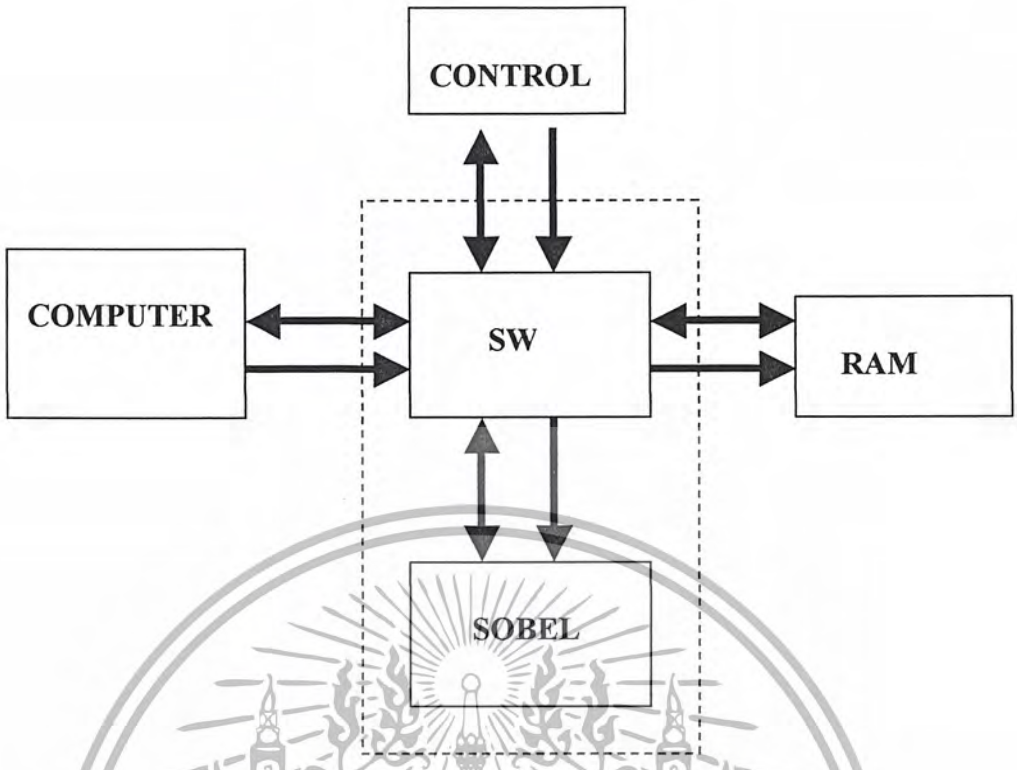
จะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่า ASIC มากเพราะใช้เวลาน้อยกว่ามากด้วย ส่วน สำคัญที่ใช้ในการทำ FPGA คือ ซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงใน อุปกรณ์ FPGA ซึ่งซอฟต์แวร์ที่ใช้ต้องเป็น ซอฟต์แวร์ที่ทำงานต่อเนื่องกันได้ สำหรับซอฟต์แวร์ที่ใช้ทำการ จำลองการทำงานของวงจรนั้น ต้องสามารถใช้งานต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบ เพราะโมเดลที่ได้จาก การทำขั้นตอนต่างๆ (ด้วยซอฟต์แวร์ ต่างๆ ต้องเอามาจำลองการทำงานได้ และในการจำลองการทำงานของ วงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดทั้งระบบ เพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่ จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ ที่ใช้บนพีซี (PC) มากขึ้นซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

บทที่ 5
การออกแบบและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดสอบการทำงานของวงจรที่ออกมาว่าสามารถที่จะทำงานได้จริงตามที่ได้ออกแบบไว้หรือไม่ ในการทดสอบจะแบ่งออกเป็นสองส่วนคือ ส่วนที่หนึ่งจะเป็นส่วนของการจำลองการทำงาน(Simulation)ของวงจรก่อนที่จะโปรแกรมลงชิฟเอฟพีจีเอเพื่อดูผลตอบสนองของวงจรอินพุทต่างๆ ที่ได้กำหนดให้ ส่วนท่าสองจะเป็นการทดสอบการทำงานจริงของวงทั้งหมดที่ได้ออกแบบมา

5.1 การออกแบบวงจรดิจิตอลเพื่อใช้งานกับอุปกรณ์ FPGA

ตามปกติแล้ว การออกแบบวงจรดิจิตอลเพื่อใช้งานร่วมกับอุปกรณ์ประเภทเอฟพีจีเอ นั้นจะมีขั้นตอนที่ไม่แตกต่างจากการอกแบบวงจรดิจิตอลประเภทเอสิกมากนัก โดยเฉพาะอย่างยิ่งการใช้ภาษาอธิบายฮาร์ดแวร์ในการออกแบบวงจร หรือเรียกว่า เอชดีแอล เช่น ภาษา VHDL ซึ่งสามารถเขียนให้อยู่ในรูปของโค้ดภาษาอธิบายฮาร์ดแวร์ แล้วนำไปผ่านการสังเคราะห์วงจร โดยที่เราสามารถเลือกเทคโนโลยีเป้าหมายได้ตามต้องการ เช่น FPGA เป็นต้น แต่จะเลือกใช้ไลบรารีสำหรับการสังเคราะห์อย่างเหมาะสมด้วย เนื่องจากปัญหาสำคัญส่วนหนึ่งของการอกแบบอยู่ตรงที่การกำหนดเทคโนโลยีเป้าหมาย ดังนั้นอาจจะมีบางจุดในโค้ดที่เราได้เขียนขึ้นต้องได้รับการเปลี่ยนแปลงแก้ไขก่อนเพื่อเหมาะสมและถูกต้องเมื่อต้องการจะเปลี่ยนจากเทคโนโลยีเป้าหมายหนึ่งไปยังเทคโนโลยีอื่นๆแต่ก็ทำได้ยาก ซึ่งไม่เหมือนกับวิธีการออกแบบที่ใช้ชีมเมติกเพราะสำหรับวิธีหลังนี้ ดีไซน์จะถูกอธิบายให้อยู่ในรูปของสัญลักษณ์ทางกราฟิกที่ประกอบขึ้นจากองค์ประกอบพื้นฐานและมีอยู่ในไลบารีสำหรับเทคโนโลยีเป้าหมาย แล้วนำมาต่อด้วยกันเป็นบล็อกย่อยหรือโมดูล และสามารถนำมาประกอบเข้าเป็นวงจรที่ซับซ้อนอีกได้หลายระดับขึ้นไปจนกลายเป็นวงจรที่เสร็จสมบูรณ์ ซึ่งการใช้ชีมเมติกนี้เป็นการอธิบายวงจรเชิงโครงสร้างนั้นเอง จุดด้อยของวิธีการออกแบบโดยใช้ชีมเมติกนี้คือ การเปลี่ยนแปลงแก้ไขดีไซน์นั้นทำได้ค่อนข้างยากลำบากและใช้เวลามาก และอีกประการที่สำคัญคือ ดีไซด์ที่ถูกสร้างขึ้นนั้นจะขึ้นอยู่กับไลบารี่เป้าหมายที่ได้เลือกใช้ ถ้าจะเปลี่ยนไลบารี่ก็ย่อมทำได้ลำบากโดยเฉพาะอย่างยิ่งวงจรที่มีขนาดใหญ่เพราะจะต้องมาแก้ไขชีมเมติกใหม่ ดังนั้นจะกล่าวถึงการออกแบบโดยใช้ภาษาฮาร์ดแวร์เท่านั้น



รูปที่ 5.1 บล็อกไดอะแกรมที่ใช้ในการออกแบบ

เนื่องจากว่าบล็อกไดอะแกรมดังรูปที่ 5.1 ถูกควบคุมด้วย Port out ซึ่งจะอยู่ที่ตำแหน่ง 2F2 เพื่อจะแสดงผลออกทางคอมพิวเตอร์ จากบล็อกไดอะแกรมจะประกอบไปด้วย 3 ส่วนคือ ส่วนที่หนึ่งจะเป็น Computer ซึ่งจะเป็นตัวแสดงผล ส่วนที่สองจะเป็นส่วนของ FPGA ส่วนที่สามคือส่วนของ RAM ในการส่งข้อมูลโดยจะผ่าน Data และ Address ซึ่ง Data นั้นจะเป็นทั้ง ตัวอ่านและตัวเขียนในตัวเดียวกันส่วนAddress จะเป็นตัวเก็บข้อมูลไว้ในตำแหน่งตั้งแต่ E000:7FFFในแต่ละส่วนจะถูกกำหนดจาก port out ซึ่งจะมีสวิชต์เป็นตัวเลือกในการใช้งานสามารถเลือกการใช้งานได้ดังนี้

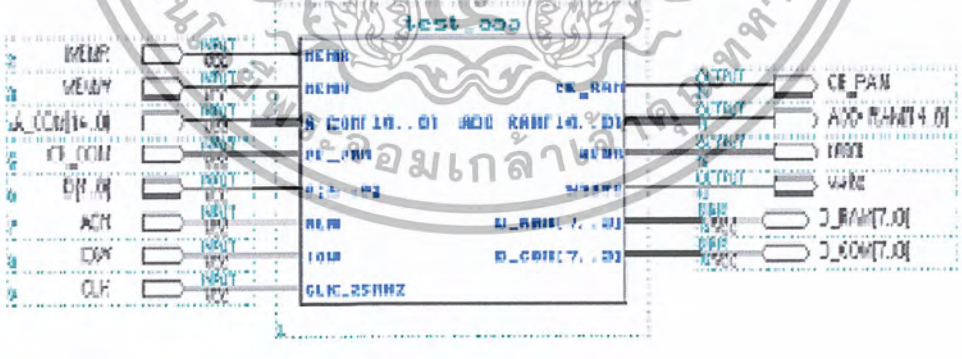
00	คือ การเขียนการคอมพิวเตอร์โดยจะผ่าน Data (8 บิท) และ Address ที่ตำแหน่ง E000:7FFF ไปเก็บไว้ที่ RAM
01	คือ การอ่าน Sobel โดยการที่เอาข้อมูลจาก RAM ไปเข้า Sobel
10	คือการเอาข้อมูลจาก RAM ไปแสดงผลที่คอมพิวเตอร์

11 คือ การเขียน Sobel โดยการเขียนจาก Edge ไปเก็บไว้ใน RAM

5.2 ทดสอบโดยการจำลองการทำงาน(Simulation)

หลังจากที่ได้ออกแบบวงจรโดยการอธิบายลักษณะพฤติกรรมของวงจร(Hardware Description Language) เป็นที่เรียบร้อยแล้ว ผู้ออกแบบจำเป็นจะต้องจำลองการทำงานเพื่อดูผลตอบสนองของวงจรต่างๆ ต่ออินพุทที่กำหนดให้ซึ่งหากผลลัพธ์ที่ได้ไม่ถูกต้องจะได้ทำการแก้ไขต่อไป ส่วนในการออกแบบโดยการอธิบายลักษณะพฤติกรรมของวงจรจะกระทำการออกแบบบนคอมพิวเตอร์ส่วนบุคคล Pentium MMX 233 MHz หน่วยความจำเป็นแบบ DDRAM 64 MB ระบบปฏิบัติการ WINDOWN 98 โดยใช้โปรแกรมในการออกแบบชื่อโปรแกรม MAX+PLUS II ซึ่งเป็น Synthesis Tool ที่ใช้สำหรับการออกแบบเพื่อทำงานบนชิฟเอฟพีจีเอของบริษัท อัลเทร่า จำกัด

ในส่วนนี้จะแสดงผลการทดสอบการทำงานของวงจรที่ได้ออกแบบด้วยภาษาวีเอชดีแอลที่สำคัญสองส่วนคือส่วนของวงจรอินเตอร์เฟสกับคอมพิวเตอร์ผ่านทางสล๊อต ISA และส่วนของ Sobel Edge Dection

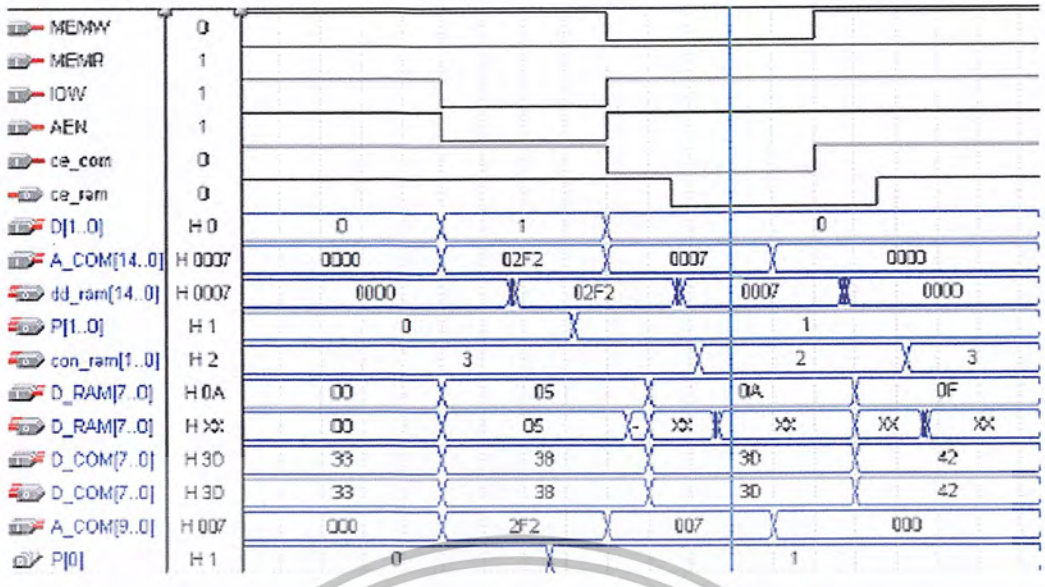


รูปที่5.2 แบบจำลองของวงจรอินเทอร์เฟส ISA แบบ 16 บิท

จากแบบจำลองในรูปที่5.2 จะมีอินพุทและเอาท์พุทดังนี้

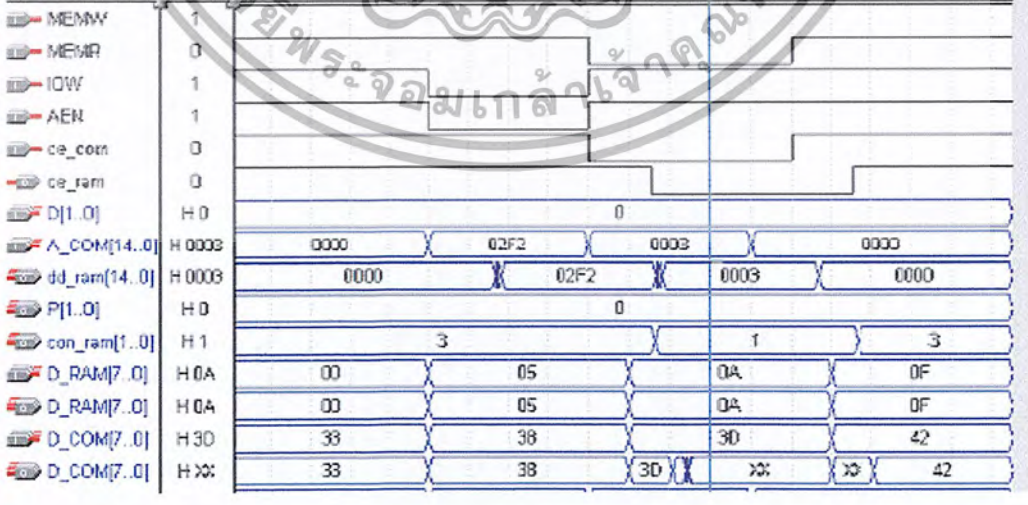
1. MEMR เป็นสัญญาณอินพุทขนาด 1 บิท ใช้สำหรับต้องการอ่านข้อผ่านคอมพิวเตอร์
2. MEMW เป็นสัญญาณอินพุทขนาด 1 บิท ใช้สำหรับในการที่ต้องการจะเขียนข้อมูลจากคอมพิวเตอร์ไปยังหน่วยความจำ
3. A_COM[14..0] เป็นสัญญาณทางอินพุทขนาด 15 บิท ใช้สำหรับระบุแอดแดรสของข้อมูลที่ต้องการจะส่งผ่านและเป็นการติดต่อกับพอร์ทที่ต้องการ
4. CE_COM เป็นสัญญาณอินพุทขนาด 1 บิท เพื่อแสดงว่าบัสไซเคิลที่เกิดขึ้นนั้นเป็นการเขียนหรืออ่านข้อมูลด้วยผ่านคอมพิวเตอร์
5. D[1..0] เป็นสัญญาณอินพุทขนาด 1 บิท ใช้สำหรับส่งข้อเพื่อที่จะบอกว่าจะให้พอร์ทไหนทำงาน
6. AEN เป็นสัญญาณอินพุทขนาด 1 บิท เพื่อแสดงว่าบัสไซเคิลที่เกิดขึ้นเป็นบัสไซเคิลสำหรับพอร์ท I/O
7. IOW เป็นสัญญาณอินพุทขนาด 1 บิท เพื่อแสดงว่าบัสไซเคิลที่เกิดขึ้นเป็นการเขียนข้อมูลลงสู่พอร์ท I/O
8. CLK เป็นสัญญาณอินพุทขนาด 1 บิท ใช้เป็นสัญญาณนาฬิกาของระบบ
9. CE_RAM เป็นสัญญาณเอาท์พุทขนาด 1 บิท ใช้สำหรับการเขียนหรืออ่านข้อมูลผ่านหน่วยความจำ
10. ADD_RAM[14..0] เป็นสัญญาณเอาท์พุทขนาด 15 บิท ใช้สำหรับระบุแอดแดรสของข้อมูลที่ต้องการที่จะเขียนหรืออ่านผ่านหน่วยความ
11. read เป็นสัญญาณเอาท์พุทขนาด 1 บิท เพื่อแสดงการอ่านข้อมูลของหน่วยความจำ
12. write เป็นสัญญาณเอาท์พุทขนาด 1 บิท เพื่อแสดงการเขียนข้อมูลผ่านหน่ววยความจำ
13. D_RAM[7..0] เป็นสัญญาณอินเอาท์ขนาด 8 บิท ใช้สำหรับส่งผ่านข้อมูลจากน่วยความจำ
14. D_COM[7..0] เป็นสัญญาณอินเอาท์ขนาด 8 บิท ใช้สำหรับส่งผ่านข้อมูลผ่านคอมพิวเตอร์

จะเห็นว่าโหมดการทำงานของวงจรอินเตอร์เฟส ISA 15 บิท จะมีอยู่ 2 โหมดการทำงาน คือ โหมดของการเขียนข้อมูลจากคอมพิวเตอร์ไปยังหน่วยความจำ และโหมดการอ่านข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์ ซึ่งจะใช้สัญญาณจากขา IOW และ D[1..0] เป็นสัญญาณควบคุมโหมดการอ่านหรือเขียนข้อมูล ในการจำลองการทำงานจะแสดงผลการจำลองการทำงานของทั้งสองโหมดดังกล่าว สามารถแสดงผลการจำลองการทำงานของโหมดเขียนข้อมูลได้ดังรูปที่5.3 และสามารถที่แสดงผลการจำลองการทำงานของโหมดอ่านข้อมูลได้ดังรูปที่5.4



รูปที่5.3 ผลการจำลองการทำงานของชุดอินเตอร์เฟส ISA แบบ 16 บิท สำหรับโหมดการเขียน

จากรูปที่5.3 เป็นผลการจำลองการทำงานของวงจรอินเตอร์เฟส ISA ในโหมดของการเขียน ถ้าเกิดเราต้องการที่จะทำการเขียนข้อมูลจากคอมพิวเตอร์ไปไว้ในหน่วยความจำภายนอกแล้ว เราจะต้องเริ่มต้นที่สัญญาณ AEN และสัญญาณ IOW จะต้องมีค่าเป็น "0" และ D[1..0] จะต้องส่งค่า 1H ออกไปเพื่อเป็นการแสดงว่าเราต้องการที่จะติดต่อกับหน่วยความจำภายนอก หลังจากนั้นมันก็จะมีการส่งแอดเดรสออกไป เพื่อเป็นการบอกว่าเราจะเขียนข้อมูลจากคอมพิวเตอร์ไปสู่หน่วยความจำ แล้วหลังจากนั้นจะต้องส่งสัญญาณ MEMW เป็น "0" ออกไปเพื่อที่จะใช้สำหรับส่งข้อมูลขนาด 8 บิท ไปที่หน่วยความจำภาย

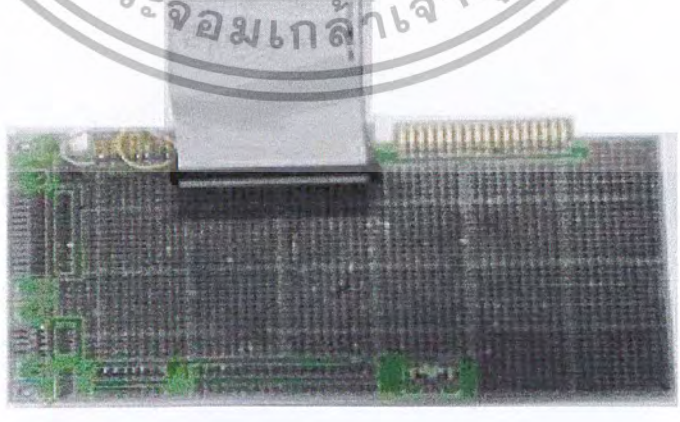


รูปที่5.4 ผลการจำลองการทำงานของชุดอินเตอร์เฟส ISA สำหรับโหมดการอ่าน

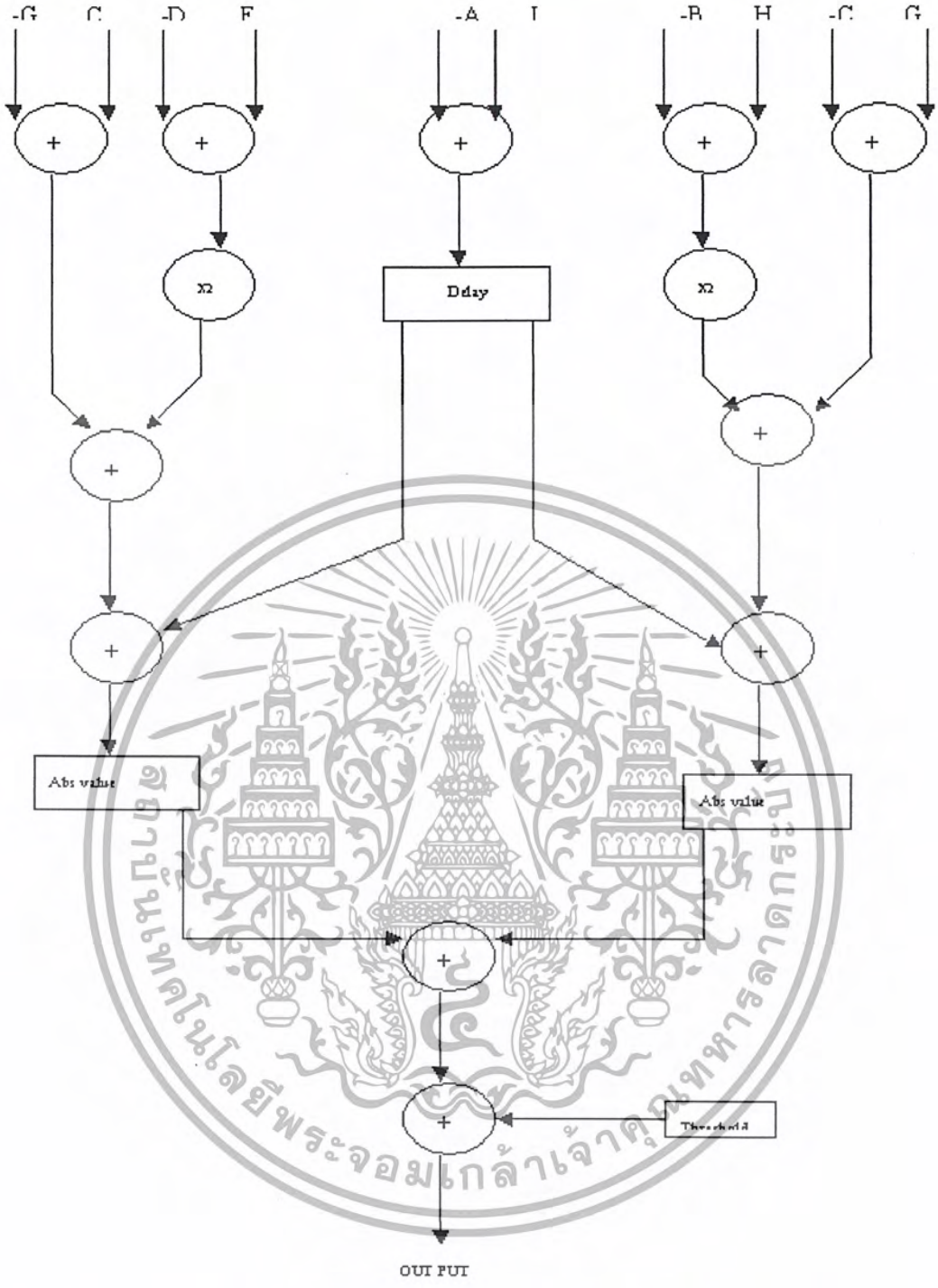
จากรูปที่5.4 เป็นผลการจำลองการทำงานของชุดอินเตอร์เฟส ISA สำหรับโหมดการอ่าน ถ้าเราต้องการที่จะอ่านข้อมูลจากหน่วยความจำภายนอกมายังคอมพิวเตอร์แล้ว โดยทีเราจะต้องเริ่มต้นดังนี้ เราจะต้องให้สัญญาณ IOW และสัญญาณ AEN เป็น “0” ก่อน แล้วให้ D[1..0] เป็น “0H” เพื่อที่จะเป็นการแสดงว่าเราต้องการที่จะอ่านข้อมูลจากหน่วยความจำภายนอกมาแสดงที่หน้าจอคอมพิวเตอร์ หลังจากนั้นสัญญาณก็จะเป็นการส่งสัญญาณแอดเดรสที่ต้องการออกมาเพื่อบอกว่าเราจ้องการที่จะอ่านข้อมูลที่ตำแหน่งไหน หลังจากนั้นก็ส่งสัญญาณ MEMR ออกมาเพื่อเป็นการแสดงการอ่านข้อมูลจากหน่วยความจำและให้เป็นการที่บอกให้ RAM ทำงานด้วย เมื่อRAM ทำงานก็จะส่งสัญญาณข้อมูลออกมาแสดงได้

รูปที่ 5.5 แสดงการ์ด ISA ในส่วนของ RAM

ส่วนต่อไปจะเป็นการจำลองการทำงานของส่วนใช้ในการประมวลผลภาพ(Sobel Edge Dectection) ซึ่งส่วนนี้จะนำเอาข้อมูลมาทำการประมวลผลตามแบบวิธีการของSobel Edge Dection ซึ่งเราสามารที่จะรูปที่5.5 มาเขียนเป็นภาษาอธิบายพฤติกรรมของวงจรได้

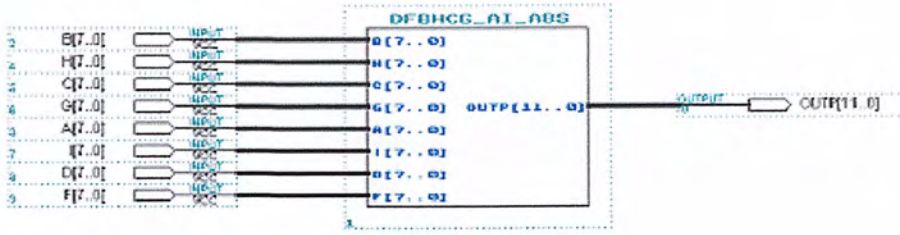


รูปที่ 5.6 แสดงการ์ด ISA ในส่วนของการติดต่อกับคอมพิวเตอร์



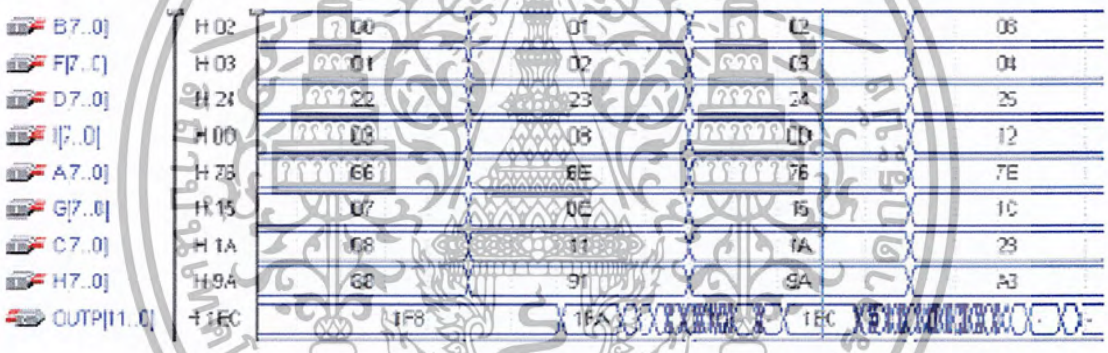
รูปที่5.7 แสดงBlock Diagram ของส่วนประมวลภาพ

1. B[7..0],H[7..0],C[7..0],G[7..0],A[7..0],I[7..0],D[7..0],F[7..0] ทั้งหมดเป็นสัญญาณอินพุทขนาด 8 บิท ที่จะใช้สำหรับการประมวลผลภาพ
2. OUTP[11..0] เป็นสัญญาณเอาท์พุทขนาด 12 บิท ใช้สำหรับการแสดงผลของสัญญาณ ที่ได้รับการประมวลผลภาพเสร็จเรียบร้อยแล้ว



รูปที่5.8 แบบจำลองของการประมวลผลภาพแบบ Sobel Edge Dection

จากรูปที่5.8 เป็นการแสดงแบบการจำลองการทำงานของส่วนประมวลผลภาพ ซึ่งจะมีสัญญาณอินพุทเอาท์ดังต่อไปนี้



รูปที่5.9 ผลการจำลองการทำงานของส่วนการประมวลผลภาพ

จากรูปที่5.9 นั้นเราสามารถที่จะนำไปเปรียบกับรูปที่4 ได้ว่าการทำงานของวงจรที่เราเขียนขึ้นนั้นมีความถูกต้องหรือไม่ เพราะเรานำBlock Diagram จากรูปที่4 นั้นมาเขียนบรรยายด้วยการบรรยายพฤติกรรมของวงจรที่จะใช้ในการคำนวณและประมวลผลภาพแบบ Sobel Edge Dection ซึ่งเมื่อโปรแกรมสามารถที่จะคำนวณเสร็จแล้วจะให้ผลการคำนวณดังรูปที่5.9

5.3 ทดสอบการทำงานจริงของวงจร

หลังจากการทำการทดสอบการทำงานของวงจรที่ได้ออกแบบด้วยภาษาวีเอชดีแอล โดยการจำลองการทำงานจนได้ผลเป็นที่หน้าพอใจแล้ว หลังจากนั้นจะต้องทำการโปรแกรมข้อมูลทางลอจิกที่ได้ลงในตัวชิฟเอฟพีจีเอ จากนั้นก็นำชิฟเอฟพีจีเอไปทดสอบกับวงจรอื่นๆ ที่ประกอบกันเป็นเครื่องประมวลผลภาพตามรูปที่5.10

รูปที่5.10แผงวงจรของเครื่องประมวลผลภาพ

รูปที่ 5.11 แสดงบอร์ด UP1

ในการทดสอบนั้นได้ทำการทดสอบผ่าน Debug ของคอมพิวเตอร์โดยเราสามารถที่จะกำหนดเงื่อนไขในการทำงานของวงจรได้โดยการใช้outport ออกว่าเราต้องจะอ่านข้อมูลหรือว่าเราต้องการที่จะเขียนข้อมูลลงสู่หน่วยความจำภายนอกได้

ผลการทดลองที่แสดงออกทางจอคอมพิวเตอร์

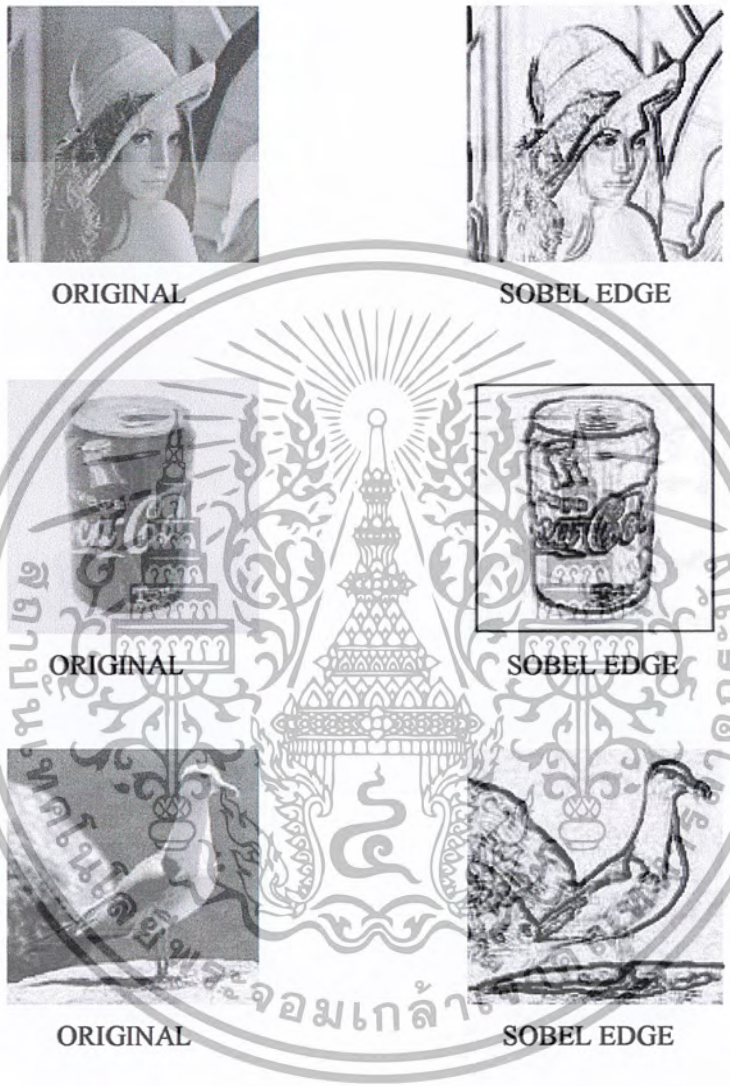
การทดลองจะใช้รูป ORIGINAL ที่เป็นภาพขาวดำขนาด 128x128 พิกเซล โดยที่ยังไม่ผ่านกระบวนการหาขอบภาพด้วยFPGA ดังแสดงในรูปที่ 5.12

รูปที่ 5.12 แสดงรูป ORIGINAL ขนาด 128x128 พิกเซล

จากรูปที่5.12 ทำการทดลองโดยนำภาพORIGINAL ขนาด 128x128 ผ่านกระบวนการหาขอบภาพด้วยFPGA ซึ่งรูปที่ออกมาก็จะมีขนาดเท่าเดิม

รูปที่ 5.13 แสดงโปรแกรมดูรูปภาพที่ผ่านกระบวนการหาขอบภาพด้วยFPGA

จากรูปที่5.13 จะแสดงให้เห็นถึงภาพORIGINAL ขนาด 128x128 และภาพที่ผ่านกระบวนการหาขอบภาพด้วยอุปกรณ์ฮาร์ดแวร์ ซึ่งอุปกรณ์ฮาร์ดแวร์ที่ใช้งานคือ FPGA ภาพที่ผ่านการหาขอบภาพมาและจะออกมาเป็น Sobel edge detection หรือ การหาขอบภาพแบบโซเบล



รูปที่ 5.14 แสดงให้เห็นถึงภาพ ORIGINAL กับภาพที่ผ่านกระบวนการหาขบภาพ

จากรูปที่ 5.14 แสดงความแตกต่างระหว่างภาพ ORIGINAL และภาพที่ผ่านการหาขอบภาพ ในการเก็บข้อมูลภาพสามารถเก็บให้อยู่ในรูปของไฟล์ข้อมูลภาพตามมาตรฐานทั่วไป ซึ่งสามารถนำไฟล์ดังกล่าวไปใช้กับโปรแกรมแสดงภาพโดยมีความระเอียดของสัญญาณเท่ากับ 128x128 ระดับ

บทที่ 6
สรุป

การประมวลผลภาพดิจิตอล เป็นที่กว้างขวางมาก เพราะครอบคลุมถึงเทคนิคต่างๆที่ปรับปรุง แก้ไขภาพในลักษณะที่ข้อมูลเป็นดิจิตอลด้วยคุณสมบัติการทำงานของระบบคอมพิวเตอร์ ค่างานในลักษณะของการใช้สัญญาณดิจิตอล จึงได้มีการนำคอมพิวเตอร์มาประมวลผลภาพ ทำการศึกษาระบบต่างๆได้อย่างมีประสิทธิภาพ

การหาขอบภาพโดยประยุกต์ใช้กับ FPGA มาทำงานในส่วนของการหาขอบภาพ ก็จะเป็นผลดี เนื่องจากสามารถประมวลผลด้วยความเร็วสูงซึ่งสามารถประยุกต์ใช้งานกับข้อมูลภาพซึ่งมีปริมาณมาก เหมาะสำหรับการผลิตในเชิงอุตสาหกรรมในการทำงานของระบบการตรวจสอบอัตโนมัติโดยใช้ระบบการมองเห็นของเครื่องจักร โดยใช้กล้อง วงจรปิด ในโรงงานอุตสาหกรรม อย่างเช่น ระบบตรวจสอบการประกอบแผ่นวงจร เป็นต้น

จากการทดลองการหาขอบภาพชนิด sobel ซึ่งเป็นวิธีที่ให้ผลดีและง่ายที่สุด

บรรณานุกรม

[1] Stephen Brown and Zvonko Vranesic,Fundamentals of Digital Logic with VHDL Design ,McGrawHill,2000

[2] Douglas L.Perry,VHDL,McGrawHill,1998

[3] Charles H.Roth,Digital System Design Using VHDL,PWS Publishing Company,1998

[4] Mark Zwolinski,Digital System Design with VHDL,Prentice Hill,2000

[5] Douglas L.Perry and Thomas J.Wilderotter,A Designer's Guide to VHDL Synthesis,Kluwer Academic Publishers,1998

[6] Frank A.Scarpino,VHDL and AHDL Digital System Implementtation,Prentice Hall PTR,1998

[7] Altera Corporation,"MAX7000 Programmable Logic Device Family.",[Online],Avaible:htt://www.altera.com/literature/manual

[8] Altera Corporation,"FLEX10K Embeddede Logic Device Family.",[Online],Avaible:htt://www.altera.com/literature/manual

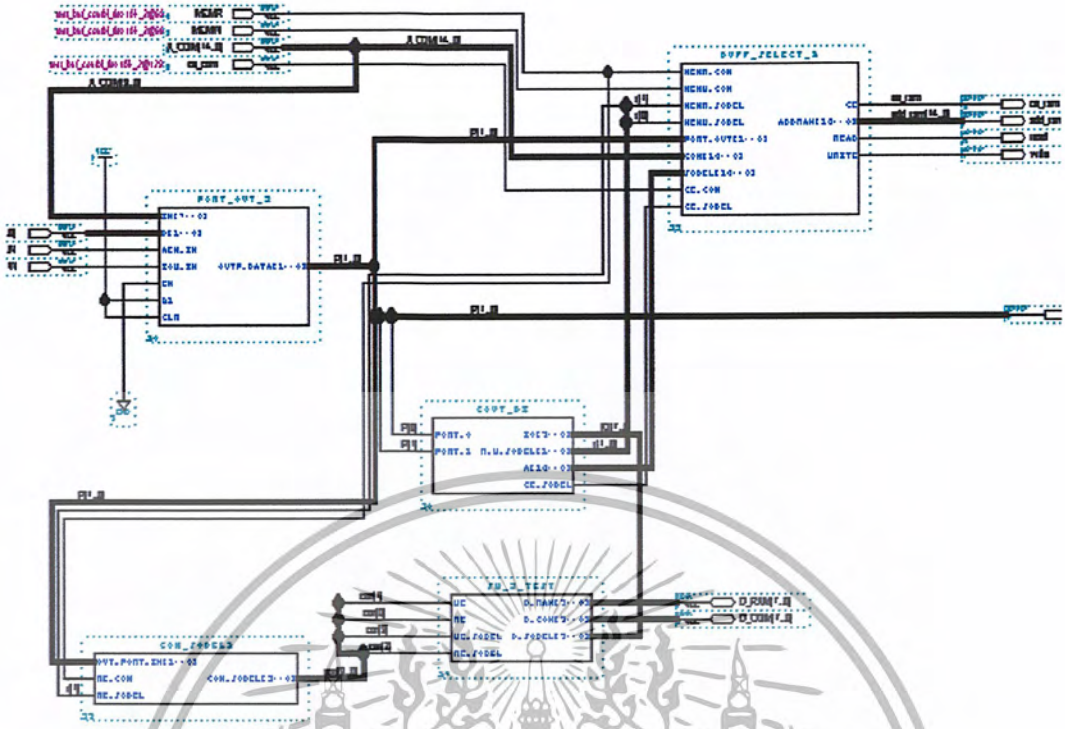
[9]Altera Corporation,"MAX PLUSII Tutorial:htt://www.altera.com/literature/manual

[10] บุญอนันต์ เกียงเอีย ,"การออกแบบเครื่องจับสัญญาณภาพทางการแพทย์โยใช้เอฟพีจีเอ,วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2545

[11]วิรัช เลิศบุษศราคาม การออกแบบวงจรรวมเฉพาะกิจสำหรับ Histrogram Equalization และ Edge Dection โดยใช้สถาปัตยกรรมแบบโมดูลลาร์และภาษา VHDL วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2537

[12] การออกแบบวงจรรวมดิจิตอลเพื่อใช้งานกับอุปกรณ์ FPGA ฝ่ายออกแบบวงจรรวม ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ พ.ศ. 2540

ภาคผนวก



รูปวงจรภายในFPGA

รูปสวิทช์เลือกสอง

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

entity port_out_3 is
        port(A_IN                                   :in std_logic_vector(9 downto
0);
                D                                                       :in
std_logic_vector(1 downto 0);
                AEN_IN,IOW_IN,en,G1,CLR             :in std_logic;
                OUTP_DATA                                       :out
std_logic_vector(1 downto 0));
end port_out_3;

architecture RTL of port_out_3 is
signal enA,enA_2:std_logic_vector(5 downto 0);
--signal A20     :std_logic_vector(2 downto 0);
--signal A40     :std_logic_vector(3 downto 0);
signal A9_0 :std_logic_vector(9 downto 0);
signal DIN,Y2   :std_logic_vector(1 downto 0);
signal outp :std_logic_vector(2 downto 0);
signal AEN_OUT,IOW,NAND_OR,NAND1,NAND2,clk: std_logic;
begin
        p0:process
                begin
                        if (en ='0') then
                                A9_0<=A_IN;
                                --A40<=AIN_40;
                                --AOUT01<=A0_1;
                                AEN_OUT<=AEN_IN;
                                IOW<=IOW_IN;
                        end if;
        end process p0;
```

```
NAND_and_OR :process
        begin
            NAND1<= A9_0(4) nand A9_0(5);
            NAND2<= A9_0(6) nand A9_0(7);
            NAND_OR<=NAND1 or NAND2;
                --sum<=NAND_OR;
end process NAND_and_OR;


decode3to8:process(enA,AEN_OUT,NAND_OR)
        begin
            enA <= A9_0(9) & A9_0(3 downto 2) & A9_0(8)&AEN_OUT
& NAND_OR;
            case enA is
                when "100000" => outp <= "000";
                when "100001" => outp <= "001";
                when others => outp <= "011";
            end case;

end process decode3to8;

decode_3_to_8_2:process
        begin
            enA_2 <=  G1 & outp(0) & A9_0(1 downto 0)  & IOW &
IOW;
            case enA_2 is
                when "101000" => Y2 <= "10";
                when "101100" => Y2 <= "01";
                when others => Y2 <= "11";
            end case;
                clk <=not Y2(0) ;
                --clkp<=clk;
```

```
end process decode_3_to_8_2;

        DATA_IN :process
            begin
                if (outp(0)='0') then
                    DIN<=D;
                else
                    DIN<="00";
                end if;
        end process DATA_IN;

        DATA_FF :process
            begin
                wait until clk'event and clk='1';
                if CLR='0' then
                    OUTP_DATA<="00";
                else
                    OUTP_DATA<=DIN;
                end if;
        end process DATA_FF;

end RTL;
```

```
----------------------------------------------------------------------------------
--      file name       :                       buff_select_2
--      Createed date   :               12/18/02
--      by      :                                       Mr.Pornthep Sahuntouy
--      detail  :                               selecet memr and memw,ce
--      Modify date     :                       01/16/03
----------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity buff_select_2 is
        port(MEMR_COM,MEMW_COM,MEMR_SOBEL,MEMW_SOBEL    :in std_logic;
        --input memr,memw form com and input memr,memw form sobel--
                PORT_OUT    :in std_logic_vector(1 downto 0);                   --
outport bit0,1--
                com,sobel       :in std_logic_vector(14 downto 0);
        --address sobel and com--
                ce_com,ce_sobel:in std_logic;
                --ce form sobel and com--
                ce                      :out std_logic;
                                --ce output to Ram--
                AddRam                  :out std_logic_vector(14 downto 0);
                --address output to Ram--
                --con_ram       :out std_logic_vector(1 downto 0);
                --read,write signal to Ram--
                read,write:out std_logic
                );
end buff_select_2;

architecture RTL of buff_select_2 is
--      signal den :std_logic_vector(5 downto 0);
        signal en :std_logic_vector(1 downto 0);
```

```
        begin
--------------------------------------------------------------------------------
--              den<=MEMR_COM & MEMW_COM & MEMR_SOBEL & MEMW_SOBEL
& PORT_OUT(1) & PORT_OUT(0);
                        --select memr and memw form sobel and com--
--                  with den select
--                      con_ram <= "10" when "101101",      --memw_com--
--                          "01"  when "110110",            --memr_sobel--
--                          "01"  when "011100",            --memr_com--
--                          "10"  when "111011",            --memwsobel--
--                          "11"  when others;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
            en <= PORT_OUT(1) & PORT_OUT(0) ;
                with en select
                    read <= MEMR_COM when "00",         --read form com
                                MEMR_SOBEL when "10",       --
read form sobel
                                '1' when others;
                with en select
                    write <=    MEMW_COM when "01",      --wrute form
sobel
                                MEMW_SOBEL when "11",       --
write form com
                                '1' when others;
--------------------------------------------------------------------------------
            --select CE to Ram--
                    with PORT_OUT select
                        ce <=   ce_com      when "00",
                                ce_com      when "01",
                                ce_sobel    when "10",
```

```
                                                    ce_sobel          when others;


                --select Address Ram--
        process(com,sobel,PORT_OUT)
        begin
---------case port_out-------------------------------------------------------------------
--      case PORT_OUT is
--              when "00" =>
--                      con_ram(0) <= MEMR_COM;
--                      con_ram(1) <= '1';
--              when "01" =>
--                      con_ram(1) <= MEMW_COM;
--                      con_ram(0) <= '1';
--              when "10" =>
--                      con_ram(0) <= MEMR_SOBEL;
--                      con_ram(1) <= '1';
--              when others =>
--                      con_ram(1) <= MEMW_SOBEL;
--                      con_ram(0) <= '1';
--      end case;
-----------------------------------------------------------------------------------------


                if PORT_OUT(1) = '1' then
                        AddRam <= sobel;
                else
                        AddRam <= com;
                end if;
        end process;


end RTL;
```

```
--------------------------------------------------------------------------------
--      file name       :                       con_data_sobel.vhd
--      Createed date   :               01/16/02
--      by      :                               Mr.Pornthep Sahuntouy
--      detail  :                       control data sobel
--      Modify date     :               01/16/03
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity con_sobel2 is
        port(OUT_PORT_IN    :in std_logic_vector(1 downto 0);--form out port 2 bit --
                re_com,re_sobel:in std_logic;                                   --read com
and read sobel--
                con_sobel           :out std_logic_vector(3 downto 0));--select data com
adn sobel--
end con_sobel2;

architecture RTL of con_sobel2 is
        signal con :std_logic_vector(3 downto 0);
--ignal en:std_logic_vector(1 downto 0);
        begin
--------------------------------------------------------------------------------
                con<=re_com & re_sobel & OUT_PORT_IN(1) & OUT_PORT_IN(0) ; --select
data --
                        with con select
                                con_sobel <= "0001" when "0100",        --read com--
                                        "0010"  when "1101",                    --
write_com--
                                        "0100"  when "1010",                    --read sobel-
-
```

```
                                        "1000"  when "1111",                --write
sobel--
                                        "0000"  when others;
-------------------------------------------------------------------------------
--              en <= OUT_PORT_IN(1) & OUT_PORT_IN(0);
--                      with en select
--                              con_sobel <=    "0001" when "00",               --read com
--                                                      "0010" when "01",
        --write com
--                                                      "0100" when "10",
        --read sobel
--                                                      "1000" when "11",
        --write sobel
--                                                      "0000" when others;
        --others
end RTL;



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity bh is
port(B,H        :in std_logic_vector(7 downto 0);
        OUTP_BH     :out std_logic_vector(8 downto 0));
end bh;


architecture RTL of bh is
begin
```

```
        process(B,H)
                variable Z1     :integer range -255 to 255;
                variable Z2     :integer range -255 to 255;
                variable Z3     :integer range -512 to 511;

begin
                Z1:=conv_integer(B);
                Z2:=conv_integer(H);
                Z3:=(Z2-Z1);
                OUTP_BH<=conv_std_logic_vector(Z3*2,9);
        end process;
end RTL;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity bhcg2 is
port(B,H,C,G    :in std_logic_vector(7 downto 0);
        OUTP_BHCG   :out std_logic_vector(9 downto 0));
end bhcg2;


architecture RTL of bhcg2 is
begin
        process(B,H,C,G)
                variable C_IN   :integer range -255 to 255;
                variable G_IN   :integer range -255 to 255;
                variable B_IN   :integer range -255 to 255;
                variable H_IN   :integer range -255 to 255;
```

```
        variable OUT_BH      :integer range -512 to 511;
        variable OUT_CG      :integer range -512 to 511;
        variable OUT_BHCG1   :integer range -1024 to 1024;


begin
        C_IN :=conv_integer(C);
        G_IN :=conv_integer(G);
        B_IN :=conv_integer(B);
        H_IN :=conv_integer(H);
        OUT_CG :=(G_IN - C_IN );
        OUT_BH :=(H_IN - B_IN)*2;
        OUT_BHCG1:=(OUT_BH + OUT_CG);
        OUTP_BHCG<=conv_std_logic_vector(OUT_BHCG1,10);
    end process;
end RTL;


library ieee;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity bhcg2ai_2 is
port(B,H,C,G,A,I        :in std_logic_vector(7 downto 0);
     OUTP_BHCG_AI       :out std_logic_vector(10 downto 0));
end bhcg2ai_2;
```

```
architecture RTL of bhcg2ai_2 is
begin
        process(B,H,C,G,A,I)
                variable A_IN    :integer range -255 to 255;
                variable I_IN    :integer range -255 to 255;
                variable C_IN    :integer range -255 to 255;
                variable G_IN    :integer range -255 to 255;
                variable B_IN    :integer range -255 to 255;
                variable H_IN    :integer range -255 to 255;
                variable OUT_AI          :integer range -512 to 511;
                variable OUT_BH          :integer range -512 to 511;
                variable OUT_CG          :integer range -512 to 511;
                variable OUT_BHCG1   :integer range -1024 to 1024;
                variable OUT_BHCG_AI_1       :integer range -2048 to 2048;
begin
                A_IN :=conv_integer(A);
                I_IN :=conv_integer(I);
                C_IN :=conv_integer(C);
                G_IN :=conv_integer(G);
                B_IN :=conv_integer(B);
                H_IN :=conv_integer(H);
                OUT_AI :=(I_IN - A_IN);
                OUT_CG :=(G_IN - C_IN );
                OUT_BH :=(H_IN - B_IN)*2;
                OUT_BHCG1:=(OUT_BH + OUT_CG);
                OUT_BHCG_AI_1:=(OUT_BHCG1 + OUT_AI);
                OUTP_BHCG_AI<=conv_std_logic_vector(OUT_BHCG_AI_1,11);
        end process;
end RTL;
```

```
LIBRARY ieee;
library lpm;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
USE ieee.std_logic_1164.all;
use lpm.lpm_components.all;


entity bhcg2ai_2_abs is
port(B,H,C,G,A,I          :in std_logic_vector(7 downto 0);
        outp_abs2                 :out std_logic_vector(10 downto 0));
end bhcg2ai_2_abs;

architecture STRUCT of bhcg2ai_2_abs is


COMPONENT bhcg2ai_2
port(B,H,C,G,A,I          :in std_logic_vector(7 downto 0);
        OUTP_BHCG_AI      :out std_logic_vector(10 downto 0));

END COMPONENT;


COMPONENT  abs_gdf
        PORT
        (
                data            : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                result          : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
                overflow                : OUT STD_LOGIC );

END COMPONENT;
```

```
        signal OUTP_BHCG_AI : std_logic_vector(10 downto 0);

    begin
        U1:bhcg2ai_2 port map(B,H,C,G,A,I,OUTP_BHCG_AI);
        U2:abs_gdf port map(OUTP_BHCG_AI,outp_abs2);
end STRUCT;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity gcdf is
port(C,G,D,F    :in std_logic_vector(7 downto 0);
     OUTP_CGDF  :out std_logic_vector(9 downto 0));
end gcdf;


architecture RTL of gcdf is
begin
    process(C,G,D,F)
        variable C_IN   :integer range -255 to 255;
        variable G_IN   :integer range -255 to 255;
        variable D_IN   :integer range -255 to 255;
        variable F_IN   :integer range -255 to 255;
        variable OUT_DF     :integer range -512 to 511;
        variable OUT_CG     :integer range -512 to 511;
        variable OUT_CGDF1  :integer range -1024 to 1024;
```

```
begin
                C_IN :=conv_integer(C);
                G_IN :=conv_integer(G);
                D_IN :=conv_integer(D);
                F_IN :=conv_integer(F);
                OUT_CG :=(C_IN - G_IN);
                OUT_DF :=(F_IN - D_IN)*2;
                OUT_CGDF1:=(OUT_DF + OUT_CG);
                OUTP_CGDF<=conv_std_logic_vector(OUT_CGDF1,10);
        end process;
end RTL;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity gcdf_ai is
port(A,I,C,G,D,F        :in std_logic_vector(7 downto 0);
        OUTP_CGDF_AI        :out std_logic_vector(10 downto 0));
end gcdf_ai;


architecture RTL of gcdf_ai is
begin
        process(A,I,C,G,D,F)
                variable A_IN   :integer range -255 to 255;
                variable I_IN   :integer range -255 to 255;
                variable C_IN   :integer range -255 to 255;
                variable G_IN   :integer range -255 to 255;
                variable D_IN   :integer range -255 to 255;
```

```
        variable F_IN    :integer range -255 to 255;
        variable OUT_DF          :integer range -512 to 511;
        variable OUT_CG          :integer range -512 to 511;
        variable OUT_AI          :integer range -512 to 511;
        variable OUT_CGDF1   :integer range -1024 to 1024;
        variable OUT_CGDF_AI_1       :integer range -2048 to 2048;

begin
        A_IN :=conv_integer(A);
        I_IN :=conv_integer(I);
        C_IN :=conv_integer(C);
        G_IN :=conv_integer(G);
        D_IN :=conv_integer(D);
        F_IN :=conv_integer(F);
        OUT_AI :=(I_IN - A_IN);
        OUT_CG :=(C_IN - G_IN);
        OUT_DF :=(F_IN - D_IN)*2;
        OUT_CGDF1:=(OUT_DF + OUT_CG);
        OUT_CGDF_AI_1:=(OUT_CGDF1 + OUT_AI);
        OUTP_CGDF_AI<=conv_std_logic_vector(OUT_CGDF_AI_1,11);
    end process;
end RTL;


LIBRARY ieee;
library lpm;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
USE ieee.std_logic_1164.all;
```

```
use lpm.lpm_components.all;


entity gcdf_ai_abs is
port(A,I,C,G,D,F          :in std_logic_vector(7 downto 0);
        outp_abs                  :out std_logic_vector(10 downto 0));
end gcdf_ai_abs;

architecture STRUCT of gcdf_ai_abs is


COMPONENT gcdf_ai
port(A,I,C,G,D,F          :in std_logic_vector(7 downto 0);
        OUTP_CGDF_AI      :out std_logic_vector(10 downto 0));

END COMPONENT;

COMPONENT abs_gdf
        PORT
        (
                data              : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                result            : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
                overflow          : OUT STD_LOGIC );

END COMPONENT;

                signal OUTP_CGDF_AI : std_logic_vector(10 downto 0);


        begin
                U1:gcdf_ai port map(A,I,C,G,D,F,OUTP_CGDF_AI);
                U2:abs_gdf port map(OUTP_CGDF_AI,outp_abs);
end STRUCT;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity df is
port(D,F          :in std_logic_vector(7 downto 0);
     OUTP_DF      :out std_logic_vector(8 downto 0));
end df;

architecture RTL of df is
begin
     process(D,F)
          variable Z1     :integer range -255 to 255;
          variable Z2     :integer range -255 to 255;
          variable Z3     :integer range -512 to 511;
begin
          Z1:=conv_integer(D);
          Z2:=conv_integer(F);
          Z3:=(Z2-Z1);
          OUTP_DF<=conv_std_logic_vector(Z3*2,9);
     end process;
end RTL;

LIBRARY ieee;
library lpm;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
USE ieee.std_logic_1164.all;
```

```
use lpm.lpm_components.all;

entity dfbhcg_ai_abs is
port(B,H,C,G,A,I,D,F     :in std_logic_vector(7 downto 0);
        OUTP                              :out std_logic_vector(11 downto 0));
end dfbhcg_ai_abs;

architecture STRUCT of dfbhcg_ai_abs is

COMPONENT  bhcg2ai_2_abs
port(B,H,C,G,A,I          :in std_logic_vector(7 downto 0);
        outp_abs2                 :out std_logic_vector(10 downto 0));
END COMPONENT;


COMPONENT gedf_ai_abs
port(A,I,C,G,D,F          :in std_logic_vector(7 downto 0);
        outp_abs                  :out std_logic_vector(10 downto 0));
END COMPONENT;


COMPONENT  a12_bit
        port(IN_1,IN_2  :in std_logic_vector(10 downto 0);
                OUTP_12                   :out std_logic_vector(11 downto 0));

END COMPONENT;

signal outp_abs2 :std_logic_vector(10 downto 0);
signal outp_abs          :std_logic_vector(10 downto 0)   ;

begin
U1:bhcg2ai_2_abs port map(B,H,C,G,A,I,outp_abs2);
```

```
U2:gcdf_ai_abs port map(A,I,C,G,D,F,outp_abs);
U3:a12_bit     port map(outp_abs2,outp_abs,OUTP);
end STRUCT ;


library ieee;
library altera;
use ieee.std_logic_1164.all;
use altera.maxplus2.all;
use work.sobel_lib.all;

entity a_write_read is
        port(
                clk_in          :in std_logic;
                wr_sobel        :out std_logic;
                oe_sobel        :out std_logic;
                ce_sobel        :out std_logic;
                sel_con         :in std_logic_vector(1 downto 0);
                bidir           :inout std_logic_vector(7 downto 0);
                data_in         :out std_logic_vector(7 downto 0);
                data_out        :out std_logic_vector(7 downto 0);
                addr_sobel      :out std_logic_vector(14 downto 0)
                );
end a_write_read;

architecture STR of a_write_read is

---------aa_osc.vhd------------------------------------------------------------
        component aa_osc
                port(
                        T_in            :in std_logic;
                clk_25MHz               :in std_logic;
```

```
                out_3MHz            :out std_logic
                );
        end component;
-------------------------------------------------------------------------------
---------P_ADD.vhd-------------------------------------------------------------
        component P_ADD
                port(
                        wr                          :in std_logic;
                        rd                          :in std_logic;
                        outp                        :out addr16
                        );
        end component;
-------------------------------------------------------------------------------
---------g_con_gen-------------------------------------------------------------
        component g_con_gen
                port(
                        sel_wr_oe           :in std_logic_vector(1 downto 0);
                        clk                 :in std_logic;
                        wr                  :out std_logic;
                        oe                  :out std_logic;
                        ce                  :out std_logic
                );
        end component;
-------------------------------------------------------------------------------
---------bi_direc.vhd----------------------------------------------------------
        component bi_direc
                port(
                        bidir               :inout std_logic_vector(7 downto 0);
                        oe,clk              :in std_logic;
                        inp                 :in std_logic_vector(7 downto 0);
                        outp                :out std_logic_vector(7 downto 0)
                        );
```

```
        end component;
-------------------------------------------------------------------------------
---------dff_8.vhd-------------------------------------------------------------
        component dff_8
                port(
                        d                       :in std_logic_vector(7 downto 0);
                        clk                     :in std_logic;
                        Q                       :out std_logic_vector(7 downto 0)
                        );
        end component;
-------------------------------------------------------------------------------

---------signal clock 3MHz-----------------------------------------------------
        signal clk_3MHz:std_logic;
-------------------------------------------------------------------------------
---------signal data 8 bit form add_decode to dff_8----------------------------
        signal data_dff:std_logic_vector(7 downto 0);
        signal data_in_bi:std_logic_vector(7 downto 0);
        signal data_out_bi:std_logic_vector(7 downto 0);
-------------------------------------------------------------------------------
---------signal clock Dff 8 bit------------------------------------------------
        signal clk_dff:std_logic;
        signal write:std_logic;
-------------------------------------------------------------------------------
begin
                oe_sobel <= clk_dff;
                wr_sobel <= write;
        U0:aa_osc port map(sel_con(1),clk_in,clk_3MHz);
        U1:g_con_gen port map(sel_con,clk_3MHz,write,clk_dff,ce_sobel);
        U2:P_ADD port map(sel_con(1),sel_con(0),addr_sobel);
        U3:bi_direc port map(bidir,write,clk_dff,data_in_bi,data_out_bi);
        U4:dff_8 port map(data_out_bi,clk_dff,data_in_bi);
```

```
                data_in <= data_in_bi;
                data_out <= data_out_bi;
end STR;


library ieee;
use ieee.std_logic_1164.all;

entity aa_osc is
        port(
                T_in                    :in std_logic;
                clk_25MHz               :in std_logic;
                out_3MHz                :out std_logic
                );
end aa_osc;

architecture STR of aa_osc is

---------aa_tff.vhd-------------------------------------------------------------
        component aa_tff
                port(
                        T               :in std_logic;
                        clk             :in std_logic;
                        Q               :out std_logic
                        );
        end component;
--------------------------------------------------------------------------------
---------signal in T FF--------------------------------------------------------
        signal Q1:std_logic;
        signal Q2:std_logic;
        signal Q1andQ2:std_logic;
--------------------------------------------------------------------------------
```

```
begin
                Q1andQ2 <= Q1 and Q2;
        U1:aa_tff port map(T_in,clk_25MHz,Q1);
        U2:aa_tff port map(Q1,clk_25MHz,Q2);
        U3:aa_tff port map(Q1andQ2,clk_25MHz,out_3MHz);
end STR;



--------------------------------------------------------------------------------
--      File name       :                       dff_8.vhd
--      Createed date   :               01/18/03
--      By      :                               Mr.pornthep Sanhuntuy
--      Detial  :                       delay data out to mux_8
--      Modify date     :               -
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity dff_8 is
        port(D          :in std_logic_vector(7 downto 0);
                clk                     :in std_logic;
                Q                               :out std_logic_vector(7 downto 0));
end dff_8;

architecture RTL of dff_8 is
        begin
                process(clk,D)
                begin
                if clk'event and clk='1' then
                        Q<=D;
```

```
                end if;
                end process;
end RTL;


--------------------------------------------------------------------------------
--      File name       :               bi_direc.vhd
--      Createed date   :       02/10/03
--      By      :                       Mr.pornthep Sanhuntuy
--      Detial  :                       bidirectional 8 bit
--      Modify date     :               -
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY bi_direc IS
        PORT(
                bidir   : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                oe, clk : IN STD_LOGIC;
                inp     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                outp    :out std_logic_vector(7 downto 0)
                );
END bi_direc;

ARCHITECTURE maxpld OF bi_direc IS
        SIGNAL a : STD_LOGIC_VECTOR (7 DOWNTO 0); -- DFF that stores
-- value from input.
        SIGNAL b : STD_LOGIC_VECTOR (7 DOWNTO 0); -- DFF that stores

BEGIN -- feedback value.
```

```
D_FF:PROCESS(clk)
        BEGIN
                IF clk = '0' AND clk'EVENT THEN -- Creates the flipflops
                        a <= inp ;
                        outp <= b;
                END IF;
END PROCESS D_FF;

TRI1_STATE:PROCESS (oe, bidir) -- Behavioral representation
        BEGIN -- of tri-states.
                IF( oe = '1') THEN
                        bidir <= "ZZZZZZZZ";
                        b <= bidir;
                ELSE
                        bidir <= a;
                        b <= bidir;
                END IF;
        END PROCESS TRI1_STATE;
END maxpld;


library ieee;
use ieee.std_logic_1164.all;

entity g_con_sobel is
        port(
                clk                     :in std_logic;
                in_read         :in std_logic;
                in_write:in std_logic;
                ce_sobel        :out std_logic;
                wr_sobel        :out std_logic;
                oe_sobel        :out std_logic
```

```
        );
end g_con_sobel;

architecture RTL of g_con_sobel is
        signal sel :std_logic_vector(1 downto 0);
        begin
                process(clk,in_read,in_write)
                begin
                        if clk='1' then
                                sel <= in_read & in_write;
                        case sel is
                                when "10" =>
                                        ce_sobel <= '0';
                                        wr_sobel <= '1';
                                        oe_sobel <= '0';
                                when "11" =>
                                        ce_sobel <= '0';
                                        wr_sobel <= '0';
                                        oe_sobel <= '1';
                                when others =>
                                        ce_sobel <= '1';
                                        wr_sobel <= '1';
                                        oe_sobel <= '1';
                                end case;
                                else
                                        ce_sobel <= '1';
                                        wr_sobel <= '1';
                                        oe_sobel <= '1';
                        end if;
                end process;
end RTL;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.sobel_lib.all;

entity g_con_gen is
        port(
                sel_wr_oe               :in std_logic_vector(1 downto 0);
                clk                     :in std_logic;
                wr                      :out std_logic;
                oe                      :out std_logic;
                ce                      :out std_logic
                );
end g_con_gen ;

architecture STR of g_con_gen is

---------mux_count_write.vhd-----------------------------------------------------
        component mux_count_write
                port(
                        sel_con         :in t_sel;
                        clk             :in std_logic;
                        outp_write      :out std_logic
                        );
        end component;
-------------------------------------------------------------------------------
---------mux_count_read.vhd------------------------------------------------------
        component mux_count_read
                port(
                        sel_con         :in t_sel;
                        clk             :in std_logic;
                        outp_read       :out std_logic
                        );
```

```
        end component;
-------------------------------------------------------------------------------
---------count_ce.vhd-----------------------------------------------------------
        component count_ce
                port(
                        clk                             :in std_logic;
                        reset                   :in std_logic;
                        wave                    :out std_logic
                        );
        end component;
-------------------------------------------------------------------------------
---------signal ce--------------------------------------------------------------
        signal not_ce:std_logic;
-------------------------------------------------------------------------------
begin
                not_ce <= not sel_wr_oe(1);
        U1:mux_count_write port map(sel_wr_oe,clk,wr);
        U2:mux_count_read  port map(sel_wr_oe,clk,oe);
        U3:count_ce      port map(not_ce,clk,ce);
end STR;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;


entity count_ce is
        port(
                clk             :in std_logic;
                reset   :in std_logic;
                wave    :out std_logic
```

```
        );
end count_ce;

architecture RTL of count_ce is
        signal count:integer range 0 to 16384;
        constant setwave_1      :integer :=2;           --wave_1 begin 2
        constant resetwave_1:integer :=3;       --wave_1 reset 3
        constant setwave_2      :integer :=6;
        constant resetwave_2:integer :=7;
        constant period:integer :=16384;        --wave period 16384
begin
        PULSEGEN:process(clk,reset)
                begin
                        if rising_edge(clk) then
                                if reset ='1'then
                                        if count=period then
                                                count<=0;
                                        else
                                                count <= count+1;
                                        end if;
----------counter when = setwave----------
--count 0-3 and then reset ,count 128-131 and then reset,count 256-259 and reset
                                        if count=setwave_1 then
                                                wave <='0';
                                        elsif count=resetwave_1 then
                                                wave <='1';
                                        elsif count=setwave_2 then
                                                wave <= '0';
                                        elsif count=resetwave_2 then
                                                wave <= '1';
                                        else
                                                wave <= '1';
```

```
                                    end if;
-------------------------------------------------------------------------------
                                end if;
                            end if;
        end process;
end RTL;


library ieee;
use ieee.std_logic_1164.all;
use work.sobel_lib.all;

entity mux_count_read is
        port(
            sel_con             :in t_sel;
            clk                 :in std_logic;
            outp_read           :out std_logic
            );
end mux_count_read;

architecture STR of mux_count_read is

---------mux_read.vhd-----------------------------------------------------------
        component mux_read
            port(
                sel_con         :in t_sel;
                read            :out std_logic
                );
        end component;
-------------------------------------------------------------------------------

---------count_sobel.vhd--------------------------------------------------------
```

```
        component count_sobel
                port(
                        clk                     :in std_logic;
                        reset           :in std_logic;
                        wave            :out std_logic
                        );
        end component;
-------------------------------------------------------------------------------
---------signal reset-----------------------------------------------------------
        signal rst:std_logic;
        signal not_rst:std_logic;
        signal read:std_logic;
-------------------------------------------------------------------------------

begin
                not_rst <= read xor '1';
                rst <= not not_rst;
        U1:mux_read port map(sel_con,read);
        U2:count_sobel port map(clk,rst,outp_read);
end STR;



library ieee;
use ieee.std_logic_1164.all;
use work.sobel_lib.all;

entity mux_count_write is
        port(
                sel_con                 :in t_sel;
                clk                             :in std_logic;
                outp_write              :out std_logic
```

```
                );
end mux_count_write;

architecture STR of mux_count_write is

---------mux_write.vhd-----------------------------------------------------------------
        component mux_write
                port(
                        sel_con         :in t_sel;
                        write           :out std_logic
                        );
        end component;
-------------------------------------------------------------------------------

---------count_write.vhd---------------------------------------------------------
        component count_write
                port(
                        clk                     :in std_logic;
                        reset           :in std_logic;
                        wave            :out std_logic
                        );
        end component;
-------------------------------------------------------------------------------
---------signal reset------------------------------------------------------------
        signal rst:std_logic;
        signal not_rst:std_logic;
        signal write:std_logic;
-------------------------------------------------------------------------------

begin
                not_rst <= write xor '1';
                rst <= not not_rst;
```

```
        U1:mux_write port map(sel_con,write);
        U2:count_write port map(clk,rst,outp_write);
end STR;


library ieee;
use ieee.std_logic_1164.all;
use work.sobel_lib.all;

entity P_ADD is
        port(
                wr                      :in std_logic;
                rd                      :in std_logic;
                outp            :out addr16
                );
end P_ADD;

architecture RTL of P_ADD is
        signal en :std_logic_vector(1 downto 0);
begin
                        en <= wr & rd;
                with en select
                        outp <= a0 when "10",
                                        a1 when others;

end RTL;

--------------------------------------------------------------------------------
--      File name       :               count_compa_ripple.vhd
--      Createed date   :       02/3/03
--      By      :                       Mr.pornthep Sanhuntuy
--      Detial  :               counter 128 bit to comparater 14 bit to 1 bit
--      Modify date     :               -
```

```
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity count_compa_ripple is
        port(
                inp                     :in std_logic_vector(13 downto 0);
                load            :in std_logic;
                clk                     :in std_logic;
                rst                     :in std_logic;
                out_count       :out std_logic_vector(13 downto 0);
                out_compa       :out std_logic
                );
end count_compa_ripple;

architecture structure of count_compa_ripple is

-------------ripple_count.vhd-----------------------------
        component ripple_count
                port(
                        inp             :in std_logic_vector(13 downto 0);
                        load    :in std_logic;
                        clk             :in std_logic;
                        reset   :in std_logic;
                        outp    :out std_logic_vector(13 downto 0)
                        );
        end component;
-------------------------------------------------------------
```

```
----------count_compa.vhd---------------------------------
        component count_compa
                port(
                        clk             :in std_logic;
                        en              :in std_logic;
                        outp    :out std_logic
                        );
        end component;
-------------------------------------------------------------
signal m:std_logic;

begin
   m <= not load;
        ripple:ripple_count port map(inp,load,clk,rst,out_count);
        compa:count_compa port map(clk,m,out_compa);
end structure;

-------------------------------------------------------------------------
--      File name       :               count_compa.vhd
--      Createed date   :       02/3/03
--      By      :                       Mr.pornthep Sanhuntuy
--      Detial  :                       counter 128 bit to comparater 14 bit to 1 bit
--      Modify date     :               02/04/03
-------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity count_compa is
        port(
```

```
                clk                     :in std_logic;
                en                      :in std_logic;
                outp            :out std_logic
                );
end count_compa;

architecture structure of count_compa is

----------file count128.vhd------------------------------
        component count128
                port(
                        clk             :in std_logic;
                        en              :in std_logic;
                        outp    :out std_logic_vector(13 downto 0)
                        );
        end component;
-------------------------------------------------------------

----------file comp.vhd-----------------------------------
        component comp
                port(
                        inp_compa               :in std_logic_vector(13 downto 0);
                        outp_compa              :out std_logic
                        );
        end component;
-----------------------------------------------------------------
---------input compa-----------------------------------------
        signal inp_compa :std_logic_vector(13 downto 0);
-----------------------------------------------------------------
begin
        count:count128 port map(clk,en,inp_compa);
        compa:comp port map(inp_compa,outp);
```

```
end structure;

--------------------------------------------------------------------------------
--      File name       :                   comp.vhd
--      Createed date   :           02/3/03
--      By      :                           Mr.pornthep Sanhuntuy
--      Detial  :                   comparater 14 bit to 1 bit
--      Modify date     :           -
--------------------------------------------------------------------------------

LIBRARY ieee;
use ieee.std_logic_1164.all;

entity comp is
        port(
                inp_compa               :in std_logic_vector(13 downto 0);
                outp_compa              :out std_logic
                );
end comp;

architecture RTL of comp is
begin
        with inp_compa sclect
                outp_compa <= '1'when "00000000000001",     --00h
                              '1'when "00000000000010",     --01h
                              '1'when "00000000000011",     --00h
                              '1'when "00000100000000",     --100h
                              '1'when "00000100000001",     --101h
                              '1'when "00000100000010",     --102h
                              '1'when "00000110000000",     --fdh
                              '1'when "00000110000001",     --feh
                              '1'when "00000110000010",     --ffh
```

```
                                        '0'when others;
end RTL;


--------------------------------------------------------------------------------
--      File name       :               count128.vhd
--      Createed date   :       01/29/03
--      By      :                               Mr.pornthep Sanhuntuy
--      Detial  :                       counter 14 bit or countter 128
--      Modify date     :               -
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity count128 is
        port(
                clk                     :in std_logic;
                en                      :in std_logic;
                outp            :out std_logic_vector(13 downto 0)
                );
end count128;

architecture RTL of count128 is
signal inp_count:integer range 0 to 16384;
begin
        process(clk,en)
        begin
                        if (clk'event and clk='1') then
                                if en='0' then          --enabale counter----
```

```
                    inp_count <= inp_count + 1; --counter + 1--
                end if;
            end if;
                    outp<=conv_std_logic_vector(inp_count,14);
        end process;
end RTL;


--------------------------------------------------------------------------------
--      File name       :               ripple_count.vhd
--      Createed date   :               01/29/03
--      By      :                       Mr.pornthep Sanhuntuy
--      Detial  :                       ripple counter 14 bit
--      Modify date     :               -
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity ripple_count is
        port(
                inp             :in std_logic_vector(13 downto 0);
                load            :in std_logic;
                clk             :in std_logic;
                reset           :in std_logic;
                outp            :out std_logic_vector(13 downto 0)
                );
end ripple_count;

architecture RTL of ripple_count is
```

```
signal inp_count:integer range 0 to 16384;
begin
        process(inp,load,clk,reset)
                --convter input form std_logic 14 bit to integer--
                variable i1:integer range 0 to 16384;
        begin
                i1 :=conv_integer(inp);
                        if reset='0'then   --clear output--
                                inp_count <=0;
                        elsif (clk'event and clk='1') then
                                if (load='0')then --load vale--
                                        inp_count <= i1;
                                else
                                        inp_count <= inp_count + 1; --counter + 1--
                                end if;
                        end if;
                                outp<=conv_std_logic_vector(inp_count,14);
        end process;
end RTL;
```