

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การออกแบบวงจรรวมเฉพาะกิจสำหรับ
HISTOGRAM EQUALIZATION และ EDGE DETECTION
โดยใช้สถาปัตยกรรมแบบโมดูลาร์และภาษา VHDL
THE ASIC DESIGN FOR HISTOGRAM EQUALIZATION AND EDGE DETECTION
USING A MODULAR ARCHITECTURE AND VHDL LANGUAGE

หนังสืออ้างอิง
ห้ามนำออกนอกห้องสมุด

✓ นายวิรัช เลิศบุษยสารคาม
MR. VIRAT LERDBUSSARAKAM



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2537

ISBN 974-621-160-9

ลิขสิทธิ์ของบัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้ภายในเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และยืมเอกสารทุกครั้งที่มีการนำไปใช้

เลขที่.....	21428
เลขทราบดี.....	
วัน เดือน ปี.....	พ.ศ. 2537

**THE ASIC DESIGN FOR HISTOGRAM EQUALIZATION AND EDGE DETECTION
USING A MODULAR ARCHITECTURE AND VHDL LANGUAGE**



MR. VIRAT LERDBUSSARAKAM

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING ADMINISTRATION
GRADUATE SCHOOL
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
1994**

ISBN 974-621-160-9

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อวิทยานิพนธ์

การออกแบบวงจรรวมเฉพาะกิจสำหรับ Histogram Equalization และ Edge Detection โดยใช้สถาปัตยกรรมแบบโมดูลาร์และภาษา VHDL

นักศึกษา

นายวิรัช เลิศบุษตราตาม

อาจารย์ผู้ควบคุมวิทยานิพนธ์

ผศ. ดร. บุญวัฒน์ อัดชู

ระดับการศึกษา

วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า

ภาควิชา

วิศวกรรมคอมพิวเตอร์

คณะ

วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.

2537

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ได้นำเสนอการออกแบบวงจรรวมเฉพาะกิจสำหรับ Histogram Equalization และ Edge Detection โดยใช้สถาปัตยกรรมแบบโมดูลาร์และภาษา VHDL สถาปัตยกรรมแบบโมดูลาร์เป็นลักษณะโครงสร้างที่มีการแบ่งการทำงานออกเป็นหน่วยย่อยๆ หรือโมดูลที่เป็นอิสระต่อกันซึ่งเหมาะสมสำหรับการนำมาใช้กับวงจรการประมวลผลภาพ เนื่องจากกรรมวิธีและขั้นตอนในการประมวลผลภาพนั้นมีความแตกต่างกันตามลักษณะของข้อมูลภาพ สถาปัตยกรรมแบบโมดูลาร์จะช่วยอำนวยความสะดวกและกำหนดลักษณะขั้นตอนในการประมวลผลภาพได้ตรงเป้าหมายมากกว่า ในการออกแบบวงจรได้ทำการออกแบบวงจรเพื่อการประมวลผลภาพไว้ 2 ลักษณะคือวงจร Histogram Equalization ซึ่งเป็นเทคนิคที่ใช้ในการปรับปรุงคุณภาพของภาพเพื่อบันทึกแสดงรายละเอียดของภาพให้เห็นเด่นชัดขึ้นและวงจร Edge Detection ซึ่งเป็นเทคนิคที่ใช้ในการตรวจหาขอบของวัตถุเพื่อการจำแนกรายละเอียดของวัตถุในภาพ

การออกแบบในลักษณะของวงจรรวมเฉพาะกิจทำให้แต่ละโมดูลเป็นอิสระต่อกันและสามารถเลือกใช้เฉพาะโมดูลที่เหมาะสมตามลักษณะของงานที่ต้องการได้ ในการออกแบบได้ใช้ภาษา VHDL ซึ่งเป็นภาษามาตรฐานที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ในลักษณะของข้อความและใช้วิธีการเลียนแบบการทำงานของวงจร (simulation) โดยใช้ซอฟต์แวร์ Mentor Graphics System-1076 บนเครื่อง HP-Workstation และผลที่ได้ก็แสดงให้เห็นถึงประโยชน์ที่ได้รับทั้งในด้านความเร็วในการประมวลผลภาพที่ดีกว่าการใช้ซอฟต์แวร์และความสามารถในการควบคุมการทำงานของแต่ละโมดูลอย่างอิสระทำให้ง่ายต่อการวางลักษณะของงานแต่ละอย่าง

Thesis Title	The ASIC Design for Histogram Equalization and Edge Detection Using a Modular Architecture and VHDL Language
Student	Mr. Virat Lerdbussarakam
Thesis Advisor	Assit. Prof. Dr. Boonwat Attachoo
Level of Study	Master of Engineering in Electrical Engineering Administration
Department	Computer Engineering
Faculty	Engineering King Mongkut's Institute of Technology Ladkrabang
Year	1994

ABSTRACT

This thesis presents the ASIC Design for Histogram Equalization and Edge Detection Using A Modular Architecture and VHDL Language . The modular Architecture is the module structure that each module can be working independently. This is suitable for the digital image processing circuits since processing of digital images involves procedure that is characterized by specific solutions. This design present the histogram equalization and edge detection technique that is the enhancement technique to process an image so that the result is more suitable than the original image and the segmentation technique to detect meaningful discontinuities in gray level, respectively.

When combine a modular architecture with the ASIC designs, it provides an excellent starting point for the solution of specific image processing problems. The Standard VHDL Hardware Description Language is used to describe hardware for the purpose of simulation, modeling, testing in a design process. The VHDL design and simulation system was run by using the Mentor Graphics System - 1076 and performed on a HP-Workstation. The result shows the advantage of hardware design in much more speed porcess than software and the performance of modular architecture that is optimized to carry out specific image processing problems.

กิตติกรรมประกาศ

ขอขอบพระคุณท่านผู้ช่วยศาสตราจารย์ ดร.บุญวัฒน์ อัดชู ที่ได้กรุณารับเป็นอาจารย์ที่ปรึกษาและเป็นผู้ที่ให้คำแนะนำและการสนับสนุนในการทำวิทยานิพนธ์จนกระทั่งสำเร็จลุล่วงลงได้ด้วยดี ผู้วิจัยรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณศูนย์ CAD/CAM ที่ได้ให้ความสะดวกในเรื่องของอุปกรณ์และเครื่องคอมพิวเตอร์ในการทำวิจัย

สุดท้ายนี้ผู้วิจัยขอขอบคุณมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสารรวมถึงบัณฑิตวิทยาลัยที่ได้มอบทุนสนับสนุนในการศึกษาและการวิจัยจนวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงลงได้ด้วยดี

นายวิรัช เลิศบุษตราคม



สารบัญ

หน้า

บทคัดย่อภาษาไทย	i
บทคัดย่อภาษาอังกฤษ	ii
กิตติกรรมประกาศ	iii
สารบัญ	iv
สารบัญตาราง	x
สารบัญรูป	xi

บทที่ 1 บทนำ	1
1.1 วัตถุประสงค์และแนวความคิดในการวิจัย	1
1.2 ปัญหาและหลักการใหม่ที่นำเสนอ	1
1.3 การประมวลผลภาพดิจิทัล (Digital Image Processing)	2
1.4 สถาปัตยกรรมแบบโมดูลาร์ (Modular Architecture)	2
1.5 ภาษา VHDL (VHSIC Hardware Description Language)	3
1.6 ลำดับการเรียบเรียงภายในวิทยานิพนธ์ (Oranization of The Thesis)	4

บทที่ 2 ทฤษฎีการประมวลผลภาพดิจิทัล	5
2.1 การปรับปรุงคุณภาพของภาพโดยวิธี Histogram Equalization	5
2.2 การตรวจหาขอบของวัตถุในภาพโดยวิธี Edge Detection	9
2.3 สรุป (Summary)	11

บทที่ 3 โครงสร้างของ VHDL	12
3.1 แนะนำ VHDL (Introduction to VHDL)	13
3.1.1 ข้อกำหนด (VHDL Requirements)	14
3.1.1.1 ลักษณะทั่วไป (General Features)	14
3.1.1.2 สนับสนุนการออกแบบแบบลำดับชั้น (Support for Design Hierarchy)	14
3.1.1.3 ไลบรารี (Library Support)	15
3.1.1.4 ลำดับคำสั่ง (Sequential Statement)	15
3.1.1.5 การกำหนดคุณสมบัติ (Generic Design)	15
3.1.1.6 ชนิดของข้อมูล (Type Declaration and Usage)	15
3.1.1.7 โปรแกรมย่อย (Use of Subprograms)	15
3.1.1.8 การควบคุมเวลา (Timing Control)	15
3.1.1.9 การกำหนดแบบโครงสร้าง (Structural Specification)	15
3.1.2 องค์ประกอบพื้นฐานใน VHDL (Basic Concepts in VHDL)	16
3.1.2.1 การกำหนดการเชื่อมต่อ (Interface Description)	16
3.1.2.2 การกำหนดรูปแบบการบรรยาย (Architectural Description)	17
3.1.2.3 โปรแกรมย่อย (Subprograms)	17
3.1.2.4 โอเปอเรเตอร์ (VHDL Operators)	18
3.1.2.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)	18
3.1.2.6 สัญญาณและตัวแปร (Signals and Variables)	18
3.2 การบรรยายเชิงพฤติกรรม (Behavioral Description of Hardware)	19
3.2.1 โพรเซส (Process Statement)	19

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้เพื่อการศึกษาเท่านั้น ไม่อนุญาตให้แก้ไขหรือใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2.1.1	การกำหนดตัวกระทำภายในโปรเซส (Declarative Part of a Process)	19
3.2.1.2	การกำหนดการกระทำภายในโปรเซส (Statement Part of a process)	20
3.2.1.3	การกระตุ้นและยับยั้งการกระทำของโปรเซส (Sensitivity List)	20
3.2.2	การตรวจสอบการกระทำ (Assertion Statement)	22
3.2.3	การหยุดรอ (Sequential Wait Statements)	23
3.3	การบรรยายเชิงกระแสข้อมูล (Dataflow Description of Hardware)	23
3.3.1	การกำหนดเลือกข้อมูล (Multiplexing and Data Selection)	23
3.3.2	การกำหนดการ์ด (Guarded Signal Assignments)	24
3.3.3	การกำหนดโครงข่ายของการ์ด (Nesting Guarded Blocks)	25
3.3.4	การเลือกสรรข้อมูล (Resolving Between Several Driving Values)	26
3.4	การบรรยายเชิงโครงสร้าง (Structural Specification of Hardware)	27
3.4.1	การกำหนดไลบรารี (Parts Library)	28
3.4.1.1	อินเวอร์เตอร์โมเดล (Inverter Model)	28
3.4.1.2	NAND เกตโมเดล (NAND Gate Models)	29
3.4.2	การเชื่อมต่ออุปกรณ์พื้นฐาน (Wiring of Primitive)	30
3.4.2.1	ตัวอย่างการออกแบบวงจรเปรียบเทียบ (Logic Design of Comparator)	30
3.4.2.2	ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบทีละบิต (VHDL Description of Bit-Comparator)	31
3.4.3	การเชื่อมต่ออุปกรณ์ชนิดเดียวกัน (Wiring Iterative Networks)	33
3.4.3.1	ตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิต (Design of a 4-Bit Comparator)	33
3.4.3.2	ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบขนาด 4 บิต (VHDL Description of a 4-Bit Comparator)	33
3.5	การทดสอบและเลียนแบบการทำงานของวงจร (A Test Bench and Simulation)	36
3.5.1	รูปแบบการทดสอบ (Modeling a Test Bench)	36
3.5.2	รูปแบบการทำงานร่วมกับเพิ่มข้อความ (Formatted ASCII I/O Operations)	37
3.6	สรุป (Summary)	39

บทที่ 4 สถาปัตยกรรมการออกแบบ (Design Architecture) 40

4.1	สถาปัตยกรรมของระบบ (System Interface Description)	40
4.1.1	พฤติกรรมของระบบ (Behavioral of System)	40
4.1.1.1	ชุดคำสั่ง (Instruction Set)	40
4.1.1.2	กำหนดไลบรารี (Design Libraries)	41
4.1.1.3	แพ็คเกจ (Packages)	41
4.1.1.3.1	Basic_Uilities Package	41
4.1.1.3.2	Image_utilities Package	46
4.1.1.3.3	Image_Parameters Package	48
4.1.1.3.4	Gates_Flipflop Package	49
4.1.1.3.5	Register_Logic Package	59
4.1.1.4	กำหนดเวลาของข้อมูลและเหตุการณ์ (Timing of Data and Control Events)	69

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้ภายในหน่วยงานเท่านั้น ไม่อนุญาตให้เผยแพร่ภายนอก

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.1.2	โครงสร้างของระบบ (Structural of System)	69
4.1.2.1	โครงสร้างบัส (System Bussing Structure)	70
4.1.2.2	การเชื่อมต่อระหว่างอุปกรณ์ (Interconnection of Components)	69
4.1.2.3	การบรรยายของระบบ (Description of System)	71
4.2	การจัดระบบหน่วยความจำ (Memory Organization)	74
4.2.1	พฤติกรรมของวงจรหน่วยความจำ	74
4.2.1.1	การบรรยายการเชื่อมต่อของหน่วยความจำ (Interface Description of Memory)	75
4.2.1.2	การบรรยายการทำงานระดับพฤติกรรมของหน่วยความจำ (Memory Behavioral Architecture)	75
4.2.2	โครงสร้างของวงจรหน่วยความจำ (Structural of Memory)	77
4.2.3	การบรรยายการทำงานขององค์ประกอบ (Description of Components)	78
4.2.3.1	Byte_Memory	78
4.2.3.2	Memory_Array	79
4.2.3.3	Select_Memory	80
4.2.4	การบรรยายการทำงานระดับโครงสร้างของหน่วยความจำ (Memory Structural Architecture)	81
4.3	สรุป (Summary)	82

บทที่ 5 สถาปัตยกรรมของวงจรควบคุม (Controller Architecture) 83

5.1	พฤติกรรมของวงจรควบคุม (Behavioral of Controller)	83
5.1.1	การบรรยายการเชื่อมต่อของวงจรควบคุม (Interface Description of Controller)	83
5.1.2	การบรรยายการทำงานระดับพฤติกรรมของวงจรควบคุม (Controller Behavioral Architecture)	84
5.2	โครงสร้างของวงจรควบคุม (Structural of Controller)	89
5.2.1	โครงสร้างบัสของวงจรควบคุม (Controller Bussing Structure)	90
5.2.2	การบรรยายองค์ประกอบ (Description of Components)	89
5.2.2.1	การบรรยายระดับข้อมูลของส่วนควบคุมภายใน (Dataflow of Logic_of_Controller)	89
5.2.2.2	โครงสร้างของส่วนควบคุมภายใน (General Outline of Logic_of_Controller)	96
5.2.2.3	การบรรยายระดับโครงสร้างของส่วนควบคุมภายใน (Structural of Logic_of_Controller)	96
5.2.3	การบรรยายระดับโครงสร้างของวงจรควบคุม (Controller Structural Architecture)	106
5.3	สรุป (Summary)	108

บทที่ 6 สถาปัตยกรรมของวงจรปรับปรุงคุณภาพของภาพ (Histogram Equalization Architecture) 109

6.1	พฤติกรรมของวงจรปรับปรุงคุณภาพของภาพ (Behavioral of Histogram Equalization)	109
6.1.1	การบรรยายการเชื่อมต่อของวงจรปรับปรุงคุณภาพของภาพ (Interface Description of Histogram Equalization)	109

6.1.2 การบรรยายการทำงานในระดับพฤติกรรมของวงจรปรับปรุง คุณภาพของภาพ (Histogram Equalization Behavioral Architecture)	110
6.2 โครงสร้างของวงจรปรับปรุงคุณภาพของภาพ (Structural of Histogram Equalization)	113
6.2.1 โครงสร้างบัส (Histogram Equalization Bussing Structure)	114
6.2.2 การบรรยายองค์ประกอบ (Description of Components)	115
6.2.2.1 การบรรยายระดับข้อมูลของวงจรควบคุมภายใน (Dataflow of Histogram Controller)	115
6.2.2.2 โครงร่างของวงจรควบคุมภายใน (General Outline of Histogram Controller)	119
6.2.2.3 การบรรยายระดับโครงสร้างของวงจรควบคุมภายใน (Structural of Histogram Controller)	119
6.2.2.4 วงจร Tri_State_256	126
6.2.2.5 วงจร Tri_State_8_256	127
6.2.2.6 วงจร Decoder_8_to_256	128
6.2.2.7 วงจร New_Level Unit	128
6.2.2.8 วงจร Pixel_Counter Unit	129
6.2.3 การบรรยายระดับโครงสร้างของวงจรปรับปรุงคุณภาพ (Histogram Equalization Structural Architecture)	129
6.3 สรุป (Summary)	132

บทที่ 7 สถาปัตยกรรมของวงจรตรวจหาขอบของวัตถุในภาพ 133

(Edge Detection Architecture)

7.1 พฤติกรรมของวงจรตรวจหาขอบของวัตถุในภาพ (Behavioral of Edge Detection)	133
7.1.1 การบรรยายการเชื่อมต่อของวงจรตรวจหาขอบของวัตถุในภาพ (Interface Description of Edge Detection)	133
7.1.2 การบรรยายระดับพฤติกรรมของวงจรตรวจหาขอบของวัตถุในภาพ (Edge Detection Behavioral Architecture)	134
7.2 โครงสร้างของวงจรตรวจหาขอบของวัตถุในภาพ (Structural of Edge-detection)	137
7.2.1 โครงสร้างบัส (Edge Detection Bussing Structure)	138
7.2.2 การบรรยายองค์ประกอบ (Description of Components)	137
7.2.2.1 การบรรยายระดับข้อมูลของส่วนควบคุมภายใน (Dataflow of Edge Controller)	137
7.2.2.2 โครงร่างของส่วนควบคุมภายใน (General Outline of Edge Controller)	144
7.2.2.3 การบรรยายระดับโครงสร้างของส่วนควบคุมภายใน (Structural of Edge Controller)	144
7.2.2.4 วงจร Gradient Unit	157
7.2.2.5 วงจร Absolute_Gxy Unit	158
7.2.2.6 วงจร Gxy_Adder Unit	159
7.2.2.7 วงจร Bit_Comparator	160
7.2.2.8 วงจร 10_Bit_comparator	161
7.2.2.9 วงจร Sobel_Address_Register Unit	162
7.2.2.10 วงจร Sar_Adder Unit	165
7.2.2.11 วงจร Sobel_Data_Register Unit	166

7.2.3 การบรรยายระดับโครงสร้างของวงจรตรวจหาขอบของวัตถุในภาพ

(Edge Detection Structural Architecture) 166

7.3 สรุป (Summary) 169

บทที่ 8 รูปแบบและผลการทดสอบ (Modeling a Test Bench) 1708.1 รูปแบบการทดสอบสถาปัตยกรรมของระบบ
(Configuring for Testing Architecture of System) 170

8.1.1 แบบทดสอบ (Test Bench) 170

8.1.2 การกำหนดระดับของการบรรยาย (Configuration) 171

8.1.3 ผลการทดสอบ (Simulation Result) 172

8.2 รูปแบบการทดสอบสถาปัตยกรรมของหน่วยความจำ
(Configuring for Testing Architecture of Memory) 172

8.2.1 แบบทดสอบ (Test Bench) 172

8.2.2 การกำหนดระดับของการบรรยาย (Configuration) 174

8.2.3 ผลการทดสอบ (Simulation Result) 175

8.3 รูปแบบการทดสอบสถาปัตยกรรมของวงจรควบคุมหลัก
(Configuring for Testing Architecture of Controller) 176

8.3.1 แบบทดสอบ (Test Bench) 176

8.3.2 การกำหนดระดับของการบรรยาย (Configuration) 180

8.3.3 ผลการทดสอบ (Simulation Result) 180

8.4 รูปแบบการทดสอบสถาปัตยกรรมของวงจรปรับปรุงคุณภาพ
(Configuring for Testing Architecture of Histogram Equalization) 187

8.4.1 แบบทดสอบ (Test Bench) 187

8.4.2 การกำหนดระดับของการบรรยาย (Configuration) 189

8.4.3 ผลการทดสอบ (Simulation Result) 190

8.5 รูปแบบการทดสอบสถาปัตยกรรมของวงจรตรวจหาขอบของวัตถุ
(Configuring for Testing Architecture of Edge Detection) 191

8.5.1 แบบทดสอบ (Test Bench) 191

8.5.2 การกำหนดระดับของการบรรยาย (Configuration) 193

8.5.3 ผลการทดสอบ (Simulation Result) 194

8.6 สรุป (Summary) 194

บทที่ 9 สรุปและวิจารณ์ (Summary and Discusses) 196**ภาคผนวก (Appendix) 198**

A. แพคเกจและยูทิลิตี้ (Utilities Packages) 198

A.1 THE STANDARD PACKAGE 198

A.2 THE TEXTIO PACKAGE 199

A.3 THE BASIC UTILITIES PACKAGE 200

A.4 THE IMAGE UTILITIES PACKAGE 204

A.5 THE IMAGE PARAMETER PACKAGE 206

A.6 THE GATES FLIPFLOP PACKAGE 206

A.7 THE REGISTER LOGIC PACKAGE 207

B. การบรรยายของวงจรรวม (The Description of Design) 210

B.1 การบรรยายของระบบ (The Description of Modular System) 210

B.2 การบรรยายของหน่วยความจำ (The Description of Memory) 213

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์และสงวนสิทธิ์ในเนื้อหาและข้อมูลทั้งหมดไว้ใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้คัดลอกเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารอ้างอิง	260
ประวัติผู้เขียน	261

B.2.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)	213
B.2.2 การบรรยายในระดับโครงสร้าง (The Structural Description)	214
B.3 การบรรยายของวงจรควบคุมหลัก (The Description of Controller)	215
B.3.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)	215
B.3.2 การบรรยายในระดับโครงสร้าง (The Structural Description)	219
B.3.3 การบรรยายในระดับข้อมูลของส่วนควบคุมภายใน (The Dataflow Description of Logic of Controller)	221
B.4 การบรรยายของวงจรปรับปรุงคุณภาพของภาพ (The Description of Histogram Equalization)	227
B.4.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)	227
B.4.2 การบรรยายในระดับโครงสร้าง (The Structural Description)	240
B.4.3 การบรรยายในระดับข้อมูลของส่วนควบคุมภายใน (The Dataflow Description of Histogram Equalization Controller)	243
B.5 การบรรยายของวงจรตรวจจับขอบของวัตถุ (The Description of Edge Detection)	247
B.5.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)	247
B.5.2 การบรรยายในระดับโครงสร้าง (The Structural Description)	249
B.5.3 การบรรยายในระดับข้อมูลของส่วนควบคุมภายใน (The Dataflow Description of Edge Detection Controller)	252

สารบัญตาราง

	หน้า
ตารางที่ 2.1 แสดงจำนวนจุดภาพในแต่ละระดับความเข้ม และความน่าจะเป็นของระดับความเข้ม	7
ตารางที่ 2.2 แสดงค่าระดับความเข้มเดิมและใหม่	9



สารบัญรูป

	หน้า
รูปที่ 1.1 แสดงลักษณะโครงสร้างทั่วไปของสถาปัตยกรรมแบบโมดูลาร์	3
รูปที่ 2.1 แสดงฟังก์ชันการแปลงของระดับความเข้ม	6
รูปที่ 2.2 แสดง Histogram ของภาพ 4 แบบ	7
รูปที่ 2.3 แสดงภาพต้นแบบและภาพที่ผ่านขั้นตอนการ Histogram Equalization	8
รูปที่ 2.4 แสดงบริเวณพื้นที่ของภาพเป็นแมทริกซ์ขนาด 3 X 3 และ mask สำหรับคำนวณค่า G_x และ G_y	10
รูปที่ 2.5 แสดงภาพการหาขอบของวัตถุในภาพ	10
รูปที่ 3.1 แสดงขั้นตอนการออกแบบระบบดิจิทัล	12
รูปที่ 3.2 แสดงการออกแบบระบบเส้นทางของข้อมูล	13
รูปที่ 3.3 แสดงตัวอย่างการออกแบบแบบลำดับขั้น	14
รูปที่ 3.4 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม	16
รูปที่ 3.5 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อ ของ clock_component	16
รูปที่ 3.6 แสดงการบรรยายเชิงพฤติกรรมของ clock_component	17
รูปที่ 3.7 แสดงการใช้ไพรม์เจอร์	17
รูปที่ 3.8 แสดงการใช้ฟังก์ชัน	18
รูปที่ 3.9 แสดงตัวกระทำใน VHDL	18
รูปที่ 3.10 แสดงรูปแบบของการบรรยายแบบไพเรส	19
รูปที่ 3.11 แสดงตัวอย่างการประกาศตัวกระทำภายในไพเรส	19
รูปที่ 3.12 แสดงเงื่อนไขการกระทำในไพเรส	20
รูปที่ 3.13 แสดงการกระทำในไพเรส	20
รูปที่ 3.14 แสดงตัวอย่างโมเดลและการบรรยายการเชื่อมต่อของ D-FLIPFLOP	21
รูปที่ 3.15 แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop	22
รูปที่ 3.16 แสดงการใช้ ASSERT	22
รูปที่ 3.17 แสดง 8-to-1 มัลติเพล็กซ์เซอร์	24
รูปที่ 3.18 แสดงตัวอย่างการใช้ GUARDED	25
รูปที่ 3.19 แสดงตัวอย่างการใช้ NESTING GUARDED	26
รูปที่ 3.20 แสดงการกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน	26
รูปที่ 3.21 แสดงฟังก์ชันเลือกสรรข้อมูลแบบ anding	27
รูปที่ 3.22 แสดงการใช้ฟังก์ชัน anding กับการเลือกสรรข้อมูล	27
รูปที่ 3.23 แสดงสัญลักษณ์แทนตัวอุปกรณ์	28
รูปที่ 3.24 แสดง Inverter Model	28
รูปที่ 3.25 แสดง NAND2 Model	29
รูปที่ 3.26 แสดง NAND3 Model	29
รูปที่ 3.27 แสดงวงจรเปรียบเทียบทีละบิต	30
รูปที่ 3.28 แสดงสัญลักษณ์และสมการบูลีนของวงจรเปรียบเทียบทีละบิต	30
รูปที่ 3.29 แสดงการบรรยายการเชื่อมต่อ	31
รูปที่ 3.30 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต	31
รูปที่ 3.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิต	32
รูปที่ 3.32 แสดงวงจรเปรียบเทียบขนาด 4 บิต	33
รูปที่ 3.33 แสดง nibble_comparator	34
รูปที่ 3.34 แสดงการบรรยายการทำงานของวงจร	34
รูปที่ 3.35 แสดงลักษณะของวงจรแบบสัญลักษณ์	35

เอกสารนี้เป็นเอกสารสงวนลิขสิทธิ์สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ทางการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 3.26 แสดงรูปของแบบทดสอบวงจรเปรียบเทียบขนาด 4 บิต	36
รูปที่ 3.27 แสดงการบรรยายแบบทดสอบ	37
รูปที่ 3.28 แสดงตัวอย่างการทำงานร่วมกับเพิ่มข้อความ	38
รูปที่ 3.29 แสดงรายงานผลที่ได้จากการทดสอบและเลียนแบบ	39
รูปที่ 4.1 แสดงชุดคำสั่ง	40
รูปที่ 4.2 แสดงโครงสร้างของแพดเกจ	41
รูปที่ 4.3 แสดงส่วนของการประกาศใน <i>basic_utilities</i>	42
รูปที่ 4.4 แสดงไพรมีเจอร์ <i>qit2int</i> และ <i>int2qit</i>	42
รูปที่ 4.5 ฟังก์ชัน <i>inc</i>	43
รูปที่ 4.6 ฟังก์ชัน "AND"	43
รูปที่ 4.7 ฟังก์ชัน "OR"	44
รูปที่ 4.8 ฟังก์ชัน "NOT"	44
รูปที่ 4.9 ฟังก์ชัน <i>wire</i>	45
รูปที่ 4.10 ฟังก์ชัน <i>anding</i>	45
รูปที่ 4.11 ฟังก์ชัน <i>oring</i>	45
รูปที่ 4.12 ฟังก์ชัน <i>wiring</i>	46
รูปที่ 4.13 แสดงส่วนการประกาศของแพดเกจ <i>image_utilities</i>	46
รูปที่ 4.14 ฟังก์ชัน "AND"	47
รูปที่ 4.15 ฟังก์ชัน "OR"	47
รูปที่ 4.16 ฟังก์ชัน "NOT"	47
รูปที่ 4.17 ไพรมีเจอร์ <i>qit2char</i>	48
รูปที่ 4.18 ไพรมีเจอร์ <i>qit_vector2str</i>	48
รูปที่ 4.19 ไพรมีเจอร์ <i>wqit_vector2str</i>	48
รูปที่ 4.20 แสดงแพดเกจ <i>image_parameters</i>	49
รูปที่ 4.21 แสดงการประกาศส่วนอุปกรณ์	49
รูปที่ 4.22 แสดงแพดเกจ <i>gates_flipflop</i>	49
รูปที่ 4.23 แสดงการบรรยายการทำงานของเกท <i>inverter</i>	50
รูปที่ 4.24 แสดงการบรรยายการทำงานของเกท <i>and-2</i>	50
รูปที่ 4.25 แสดงการบรรยายการทำงานของเกท <i>and-3</i>	51
รูปที่ 4.26 แสดงการบรรยายการทำงานของเกท <i>and-4</i>	51
รูปที่ 4.27 แสดงการบรรยายการทำงานของเกท <i>or-2</i>	52
รูปที่ 4.28 แสดงการบรรยายการทำงานของเกท <i>or-3</i>	52
รูปที่ 4.29 แสดงการบรรยายการทำงานของเกท <i>or-4</i>	53
รูปที่ 4.30 แสดงการบรรยายการทำงานของเกท <i>nand-2</i>	53
รูปที่ 4.31 แสดงการบรรยายการทำงานของเกท <i>nand-3</i>	54
รูปที่ 4.32 แสดงการบรรยายการทำงานของเกท <i>nand-4</i>	54
รูปที่ 4.33 แสดงการบรรยายการทำงานของ <i>tri_state</i>	54
รูปที่ 4.34 แสดงการบรรยายการทำงานในระดับข้อมูลของ <i>d-flipflop</i>	55
รูปที่ 4.35 แสดงการบรรยายการทำงานในระดับโครงสร้างของ <i>d-flipflop</i>	56
รูปที่ 4.36 แสดงการบรรยายการทำงานของ <i>d-latch</i>	56
รูปที่ 4.37 แสดงการบรรยายการทำงานของ <i>master-slave-dff</i>	57
รูปที่ 4.38 แสดงการบรรยายการทำงานของ <i>jk-flipflop</i>	58
รูปที่ 4.39 แสดงการบรรยายการทำงานของ <i>signal_delay</i>	58
รูปที่ 4.40 แสดงแพดเกจ <i>Register_Logic</i>	60
รูปที่ 4.41 แสดงการบรรยายการทำงานของรีจิสเตอร์ขนาด 8 บิตในระดับข้อมูล	60
รูปที่ 4.42 แสดงการบรรยายการทำงานของรีจิสเตอร์ขนาด 8 บิตในระดับโครงสร้าง	61
รูปที่ 4.43 แสดงการบรรยายการทำงานของ <i>program_counter_unit</i> ในระดับพฤติกรรม	62

รูปที่ 4.44 แสดงการบรรยายการทำงานของ <i>program_counter_unit</i> ในระดับโครงสร้าง	62
รูปที่ 4.45 แสดงการบรรยายการทำงานของ <i>memory_address_register_unit</i> ระดับพฤติกรรม	63
รูปที่ 4.46 แสดงการบรรยายการทำงานของ <i>memory_address_register_unit</i> ระดับโครงสร้าง	64
รูปที่ 4.47 แสดงการบรรยายการทำงานของวงจรเปรียบเทียบ <i>bit_compare_row_col</i>	64
รูปที่ 4.48 แสดงการบรรยายการทำงานของวงจรเปรียบเทียบขนาด 8 บิต	65
รูปที่ 4.49 แสดงการบรรยายการทำงานของวงจรบวก	66
รูปที่ 4.50 แสดงการบรรยายการทำงานของวงจรบวกเลขขนาด n บิต	67
รูปที่ 4.51 แสดงการบรรยายการทำงานของ <i>2_to_4_decoder</i>	68
รูปที่ 4.52 แสดงการบรรยายการทำงานของ <i>4_to_16_decoder</i>	68
รูปที่ 4.53 แสดงการบรรยายการทำงานของ <i>8_to_256_decoder</i>	69
รูปที่ 4.54 แสดงความสัมพันธ์ของสัญญาณ <i>clock</i> และสัญญาณควบคุม	69
รูปที่ 4.55 แสดงไออะแกรมโครงสร้างบัสของระบบ	70
รูปที่ 4.56 แสดงการบรรยายระดับโครงสร้างของระบบ	74
รูปที่ 4.57 แสดงการอ้างอิงตำแหน่งของหน่วยความจำ	75
รูปที่ 4.58 แสดงการบรรยายการเชื่อมต่อของหน่วยความจำ	75
รูปที่ 4.59 แสดงการบรรยายพฤติกรรมของหน่วยความจำ	76
รูปที่ 4.60 แสดงไออะแกรมโครงสร้างของหน่วยความจำ	77
รูปที่ 4.61 แสดงการบรรยายหน่วยความจำขนาด 1 ไบต์	79
รูปที่ 4.62 แสดงการบรรยายโครงสร้างของหน่วยความจำอะเรย์ขนาด 256×256 ไบต์	80
รูปที่ 4.63 แสดงการบรรยายการทำงานในระดับข้อมูลของ <i>Select_Memory</i>	80
รูปที่ 4.64 แสดงการบรรยายการทำงานระดับโครงสร้างของหน่วยความจำ	82
รูปที่ 5.1 แสดงสัญลักษณ์และรูปแบบการบรรยายส่วนเชื่อมต่อของวงจรควบคุมหลัก	84
รูปที่ 5.2 แสดงโครงสร้างของการบรรยายพฤติกรรมของวงจรควบคุม	85
รูปที่ 5.3 แสดงการกำหนดตัวแปร	85
รูปที่ 5.4 แสดง <i>Handle interrupt</i>	85
รูปที่ 5.5 แสดงขั้นตอนการอ่านคำสั่ง	85
รูปที่ 5.6 แสดงขั้นตอนการปฏิบัติคำสั่ง <i>clm</i>	86
รูปที่ 5.7 แสดงขั้นตอนการปฏิบัติคำสั่ง <i>swm</i>	87
รูปที่ 5.8 แสดงขั้นตอนการปฏิบัติคำสั่ง <i>mod1-mod8</i>	88
รูปที่ 5.9 แสดงขั้นตอนการอ่านคำสั่งไบต์ที่สอง	88
รูปที่ 5.10 แสดงการทำคำสั่งไบต์ที่สอง	89
รูปที่ 5.11 แสดงโครงสร้างบัสของวงจรควบคุม	90
รูปที่ 5.12 แสดงการบรรยายการเชื่อมต่อของส่วนควบคุมภายใน	91
รูปที่ 5.13 แสดงการกำหนดสัญญาณสำหรับการบรรยาย	91
รูปที่ 5.14 แสดงการกำหนดค่าให้กับเอาต์พุตจริงของวงจร	92
รูปที่ 5.15 แสดง <i>s1</i> บล็อกสำหรับสเตทที่ 1	93
รูปที่ 5.16 แสดง <i>s2</i> บล็อกสำหรับสเตทที่ 2	93
รูปที่ 5.17 แสดง <i>s3</i> บล็อกสำหรับสเตทที่ 3	95
รูปที่ 5.18 แสดง <i>s4</i> บล็อกสำหรับสเตทที่ 4	95
รูปที่ 5.19 แสดงการรีเซ็ตสเตท	96
รูปที่ 5.20 แสดงโครงสร้างของส่วนควบคุมภายใน	97
รูปที่ 5.21 แสดงการบรรยายระดับโครงสร้างของส่วนควบคุมภายใน	99
รูปที่ 5.22 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 1	100
รูปที่ 5.23 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 2	100
รูปที่ 5.24 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 3	104
รูปที่ 5.25 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 4	106
รูปที่ 5.26 แสดงการบรรยายระดับโครงสร้างของวงจรควบคุม	108

รูปที่ 6.1 แสดงสัญลักษณ์และการบรรยายการเชื่อมต่อของวงจรปรับปรุงคุณภาพของภาพ	110
รูปที่ 6.2 แสดงโครงสร้างการบรรยายในระดับพิกติกรรม	110
รูปที่ 6.3 แสดงการกำหนดตัวแปรที่จำเป็น	111
รูปที่ 6.4 แสดงการเซตค่าตัวแปร	111
รูปที่ 6.5 แสดงการรับข้อมูลขนาดภาพ	111
รูปที่ 6.6 แสดงขั้นตอนในการอ่านข้อมูลภาพจากหน่วยความจำต้นทางที่ละจุดภาพ	111
รูปที่ 6.7 แสดงการเปรียบเทียบและนับจำนวนจุดภาพ	112
รูปที่ 6.8 แสดงการหาระดับความเข้มใหม่	112
รูปที่ 6.9 แสดงการเปลี่ยนค่าระดับความเข้มเก่าเป็นค่าระดับความเข้มใหม่	113
รูปที่ 6.10 แสดงการโอนมอบการควบคุมกลับสู่วงจรควบคุมหลัก	113
รูปที่ 6.11 แสดงไดอะแกรมโครงสร้างบัสของวงจรทั้งหมด	114
รูปที่ 6.12 แสดงการบรรยายการเชื่อมต่อของวงจร <i>histogram_controller</i>	115
รูปที่ 6.13 แสดงการกำหนดสัญญาณและการกำหนดค่าของสัญญาณให้กับเอาต์พุต	116
รูปที่ 6.14 แสดงการทำงานในสเตทที่ 1	117
รูปที่ 6.15 แสดงการทำงานในสเตทที่ 2	118
รูปที่ 6.16 แสดงการทำงานในสเตทที่ 3	119
รูปที่ 6.17 แสดงการรีเซตสถานะของสเตท	119
รูปที่ 6.18 แสดงโครงสร้างของวงจรควบคุมภายใน	120
รูปที่ 6.19 แสดงการบรรยายระดับโครงสร้างของวงจรควบคุมภายใน	122
รูปที่ 6.20 แสดงการบรรยายวงจร <i>state_hctrl_1</i>	123
รูปที่ 6.21 แสดงการบรรยายวงจร <i>state_hctrl_2</i>	125
รูปที่ 6.22 แสดงการบรรยายวงจร <i>state_hctrl_3</i>	126
รูปที่ 6.23 แสดงการบรรยายวงจร <i>Tri_State_256</i>	127
รูปที่ 6.24 แสดงการบรรยายวงจร <i>Tri_State_8_256</i>	127
รูปที่ 6.25 แสดงการบรรยายวงจร <i>New_Level Unit</i>	128
รูปที่ 6.26 แสดงการบรรยายวงจร <i>Pixel_Counter Unit</i>	129
รูปที่ 6.27 แสดงการบรรยายระดับโครงสร้างของวงจรปรับปรุงคุณภาพ	132
รูปที่ 7.1 แสดงการบรรยายการเชื่อมต่อของวงจรตรวจหาขอบของวัตถุในภาพ	134
รูปที่ 7.2 แสดงโครงสร้างการบรรยายในระดับพิกติกรรม	134
รูปที่ 7.3 แสดงการกำหนดตัวแปรที่จำเป็น	135
รูปที่ 7.4 แสดงการเซตค่าตัวแปรเริ่มต้น	135
รูปที่ 7.5 แสดงการรับข้อมูลขนาดภาพ	135
รูปที่ 7.6 แสดงการอ่านข้อมูลขนาด 3X3 และหาค่าเกรเดียน	136
รูปที่ 7.7 แสดงการหาค่า <i>absolute</i>	136
รูปที่ 7.8 แสดงการเขียนข้อมูล	137
รูปที่ 7.9 แสดงการส่งสัญญาณแจ้งการเสร็จภารกิจ	137
รูปที่ 7.10 แสดงโครงสร้างบัสของวงจรตรวจหาขอบของวัตถุในภาพ	138
รูปที่ 7.11 แสดงการบรรยายการเชื่อมต่อของวงจร <i>edge_controller</i>	139
รูปที่ 7.12 แสดงการกำหนดสัญญาณและการกำหนดค่าของสัญญาณให้กับเอาต์พุต	140
รูปที่ 7.13 แสดงการทำงานในสเตทที่ 1	141
รูปที่ 7.14 แสดงการทำงานในสเตทที่ 2	141
รูปที่ 7.15 แสดงการทำงานในสเตทที่ 3	142
รูปที่ 7.16 แสดงการทำงานในสเตทที่ 4	143
รูปที่ 7.17 แสดงการทำงานในสเตทที่ 5	143
รูปที่ 7.18 แสดงการรีเซตสถานะของสเตท	144
รูปที่ 7.19 แสดงโครงสร้างของส่วนควบคุมภายใน	145
รูปที่ 7.20 แสดงการบรรยายระดับโครงสร้างของส่วนควบคุมภายใน	148

รูปที่ 7.21 แสดงการบรรยายวงจร <i>state_ectrl_1</i>	149
รูปที่ 7.22 แสดงการบรรยายวงจร <i>state_ectrl_2</i>	150
รูปที่ 7.23 แสดงโครงสร้างของสเตทที่ 3	151
รูปที่ 7.24 แสดงการบรรยายวงจร <i>state_ectrl_3</i>	155
รูปที่ 7.25 แสดงการบรรยายวงจร <i>state_ectrl_4</i>	156
รูปที่ 7.26 แสดงการบรรยายวงจร <i>state_ectrl_5</i>	156
รูปที่ 7.27 แสดงการบรรยายวงจร <i>Gradient Unit</i>	158
รูปที่ 7.28 แสดงการบรรยายวงจร <i>Absolute_Gxy Unit</i>	159
รูปที่ 7.29 แสดงการบรรยายวงจร <i>Gxy_Adder Unit</i>	160
รูปที่ 7.30 แสดงการบรรยายวงจร <i>Bit_Comparator</i>	161
รูปที่ 7.31 แสดงการบรรยายวงจร <i>10_Bit_Comparator</i>	162
รูปที่ 7.32 แสดงการบรรยายวงจร <i>Sobel_Address_Register Unit</i>	165
รูปที่ 7.33 แสดงการบรรยายวงจร <i>Sar_Adder Unit</i>	165
รูปที่ 7.34 แสดงการบรรยายวงจร <i>Sobel_Data_Register Unit</i>	166
รูปที่ 7.35 แสดงการบรรยายระดับโครงสร้างของวงจรตรวจหาขอบของวัตถุในภาพ	169
รูปที่ 8.1 แสดงแบบทดสอบของระบบ	171
รูปที่ 8.2 แสดงการกำหนดระดับของการบรรยายของระบบ	172
รูปที่ 8.3 แสดงแบบทดสอบของหน่วยความจำ	174
รูปที่ 8.4 แสดงการกำหนดระดับของการบรรยายของหน่วยความจำ	175
รูปที่ 8.5 แสดงผลการทดสอบหน่วยความจำ	176
รูปที่ 8.6 แสดงแบบทดสอบของวงจรควบคุมหลัก	179
รูปที่ 8.7 แสดงการกำหนดระดับของการบรรยายของวงจรควบคุมหลัก	180
รูปที่ 8.8 แสดงผลการทดสอบวงจรควบคุมหลัก	186
รูปที่ 8.9 แสดงแบบทดสอบของวงจรปรับปรุงคุณภาพ	189
รูปที่ 8.10 แสดงการกำหนดระดับของการบรรยายของวงจรปรับปรุงคุณภาพ	190
รูปที่ 8.11 แสดงผลการทดสอบวงจรปรับปรุงคุณภาพ	190
รูปที่ 8.12 แสดงแบบทดสอบของวงจรตรวจหาขอบของวัตถุ	193
รูปที่ 8.13 แสดงการกำหนดระดับของการบรรยายของวงจรตรวจหาขอบของวัตถุ	194
รูปที่ 8.14 แสดงผลการทดสอบวงจรตรวจหาขอบของวัตถุ	194

บทที่ 1

บทนำ

1.1 วัตถุประสงค์และแนวความคิดในการวิจัย

วิทยานิพนธ์ฉบับนี้ได้นำเสนอการออกแบบวงจรรวมเฉพาะกิจสำหรับ Histogram Equalization และ Edge Detection โดยใช้สถาปัตยกรรมแบบโมดูลาร์และภาษา VHDL โดยอาศัยความสามารถของภาษา VHDL ในการกำหนดพฤติกรรมและออกแบบโครงสร้างการทำงานของวงจรในระดับต่างๆได้อย่างรวดเร็วและถูกต้องเหมาะสมตามความต้องการและยังสามารถปรับเปลี่ยนตัววงจรเพื่อให้เหมาะสมกับงานเฉพาะอย่างได้ด้วย และจากแนวความคิดอันนี้เองจึงได้นำมาใช้ในการออกแบบระบบวงจรรวมเพื่อการประมวลผลภาพขึ้นมาระบบหนึ่ง ภายในระบบประกอบด้วยส่วนที่สำคัญ 3 ส่วนด้วยกันคือ ส่วนแรกเป็นส่วนของหน่วยความจำที่ประกอบด้วยหน่วยความจำขนาด 64 กิโลไบต์(Kbytes) จำนวน 2 ตัวสำหรับเก็บรายละเอียดของข้อมูลภาพต้นแบบและภาพที่ได้ผ่านกระบวนการประมวลผลภาพมาแล้ว ซึ่งสามารถเก็บข้อมูลภาพที่มีขนาดได้ถึง 256X256 จุดภาพ(Pixels) ส่วนที่สองเป็นส่วนของวงจรควบคุมที่ใช้ในการควบคุมและสั่งการให้วงจรในส่วนที่สามซึ่งเป็นวงจรสำหรับการประมวลผลภาพทำงานตามลำดับคำสั่ง และในส่วนที่สามซึ่งเป็นส่วนของวงจรสำหรับการประมวลผลภาพ ได้ออกแบบให้สามารถบรรจุวงจรรวมสำหรับการประมวลผลภาพที่แตกต่างกันได้จำนวน 8 วงจรซึ่งแต่ละวงจรรวมจะทำงานเป็นอิสระต่อกันและสามารถถอดเปลี่ยนได้ตามแต่ลักษณะของงาน

ในการออกแบบวงจรได้ทำการออกแบบวงจรเพื่อการประมวลผลภาพไว้ 2 ลักษณะคือ วงจร Histogram Equalization ซึ่งเป็นเทคนิคที่ใช้ในการปรับปรุงคุณภาพของภาพเพื่อเน้นแสดงรายละเอียดของภาพให้เห็นเด่นชัดขึ้นและวงจร Edge Detection ซึ่งเป็นเทคนิคที่ใช้ในการตรวจหาขอบของวัตถุเพื่อการจำแนกรายละเอียดของวัตถุในภาพ

ในการออกแบบได้ใช้ภาษา VHDL ซึ่งเป็นภาษามาตรฐานที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ในลักษณะของข้อความและใช้วิธีการเลียนแบบการทำงานของวงจร (simulation) โดยใช้ซอฟต์แวร์ Mentor Graphics System-1076 บนเครื่อง HP-Workstation และผลที่ได้ก็แสดงให้เห็นถึงประโยชน์ที่ได้รับทั้งในด้านความเร็วในการประมวลผลภาพที่ดีกว่าการใช้ซอฟต์แวร์และความสามารถในการควบคุมการทำงานของแต่ละโมดูลอย่างอิสระทำให้ง่ายต่อการวางลักษณะของงานแต่ละอย่าง

1.2 ปัญหาและหลักการใหม่ที่นำเสนอ

ปัจจุบันหลักการในการประมวลผลสัญญาณภาพดิจิทัลได้ถูกนำมาใช้กันอย่างกว้างขวางทั้งในแวดวงวิชาการและอุตสาหกรรมทั่วไป การประมวลผลภาพโดยทั่วไปมักจะใช้ซอฟต์แวร์เป็นหลักในการประมวลผล ซึ่งมักจะประสบกับปัญหาความล่าช้าในการประมวลผลและการแสดงผลโดยเฉพาะอย่างยิ่งกับภาพที่มีความละเอียดสูงมาก ๆ การนำเอาวงจรที่ถูกออกแบบมาโดยเฉพาะมาใช้ทำให้สามารถประมวลผลสัญญาณภาพได้เร็วขึ้น อีกทั้งความสามารถในการควบคุมการทำงานของแต่ละโมดูลอย่างอิสระทำให้ง่ายต่อการวางลักษณะของงานแต่ละอย่าง

การนำเอาสถาปัตยกรรมแบบโมดูลาร์มาใช้ในการออกแบบวงจรรวมเฉพาะกิจเพื่อการประมวลผลภาพดิจิทัลและการนำเอาภาษา VHDL ซึ่งมีความสามารถในการกำหนดพฤติกรรมการทำงานของวงจรได้ทั้งในระดับบนและล่าง ทำให้ง่ายในการตรวจสอบถึงความถูกต้องและความเป็นไปได้ของแนวความคิดหลัก ๆ ในการออกแบบได้ตั้งแต่ในระยะเริ่มต้นและยังสามารถเลียนแบบการทำงานของวงจรได้ในทุก ๆ ขั้นตอนของการออกแบบ ซึ่งสนับสนุนการออกแบบในลักษณะบนลงล่าง(Top-Down design)

การประมวลผลภาพโดยใช่วงจรรวมทางด้านฮาร์ดแวร์จะใช้เวลาในการประมวลผลน้อยกว่าเมื่อเปรียบเทียบกับวิธีการประมวลผลโดยใช้ซอฟต์แวร์ สถาปัตยกรรมแบบโมดูลาร์ช่วยให้การกำหนดลักษณะของงานเป็นไปอย่างสะดวกและตรงจุดมากกว่า ศึกษาการปรับเปลี่ยนขั้นตอนหรือตัววงจรในการไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดลอกเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประมวลผลภาพทำได้ง่าย การปรับปรุงและแก้ไขวงจรรวมให้มีประสิทธิภาพและก้าวตามทันเทคโนโลยีในการผลิตทำได้สะดวกและใช้เวลาน้อยเมื่อใช้ภาษา VHDL ในการออกแบบ

1.3 การประมวลผลภาพดิจิทัล (Digital Image Processing)

ความสนใจในการนำเอาวิธีการประมวลผลภาพมาใช้ประโยชน์ได้มีมานานแล้ว ในยุคแรก ๆ การประมวลผลภาพจะถูกนำมาใช้ในการปรับปรุงคุณภาพของข้อมูลภาพให้ดีขึ้นหรือเป็นการเตรียมข้อมูลภาพให้เหมาะสมกับอุปกรณ์ในเครื่องจักรต่าง ๆ แต่ในปัจจุบันเทคนิคในการประมวลผลภาพได้ถูกนำมาใช้ประโยชน์กันอย่างกว้างขวางไม่เพียงแต่ในแวดวงคอมพิวเตอร์เอง อีกทั้งในอุตสาหกรรมต่าง ๆ หรือแม้แต่ในงานธุรกิจ การโฆษณา และวงการบันเทิงต่าง ๆ ก็ตาม เทคนิคใหม่ๆ ที่เกิดขึ้นไม่เพียงแต่จะถูกนำมาใช้ในการปรับปรุงคุณภาพของภาพเท่านั้น แต่ยังถูกนำมาใช้ในงานสร้างสรรค์ต่าง ๆ อีกด้วย

กระบวนการในการประมวลผลภาพดิจิทัลจะเริ่มจากการรับเอาข้อมูลภาพเข้ามาเสียก่อน ซึ่งอุปกรณ์ที่ใช้ในการรับภาพได้แก่ กล้องถ่ายภาพวิดีโอ เครื่องกวาดภาพ(scanner) หรืออุปกรณ์ที่ทำหน้าที่เปลี่ยนพลังงานแม่เหล็กไฟฟ้าในช่วงคลื่นต่าง ๆ เช่น เอ็กซเรย์(x-ray) อินฟราเรด(infrared) เป็นต้น ให้เป็นสัญญาณไฟฟ้า หลังจากได้ข้อมูลภาพมาแล้วอาจจะนำไปเก็บไว้ในอุปกรณ์ที่ใช้เก็บข้อมูลก่อนหรือจะนำไปผ่านกระบวนการประมวลผลภาพเสียก็ได้ ภาพที่ได้ก็สามารถนำไปแสดงผลผ่านทางจอมอนิเตอร์ เครื่องพิมพ์หรืออุปกรณ์แสดงผลอื่น ๆ

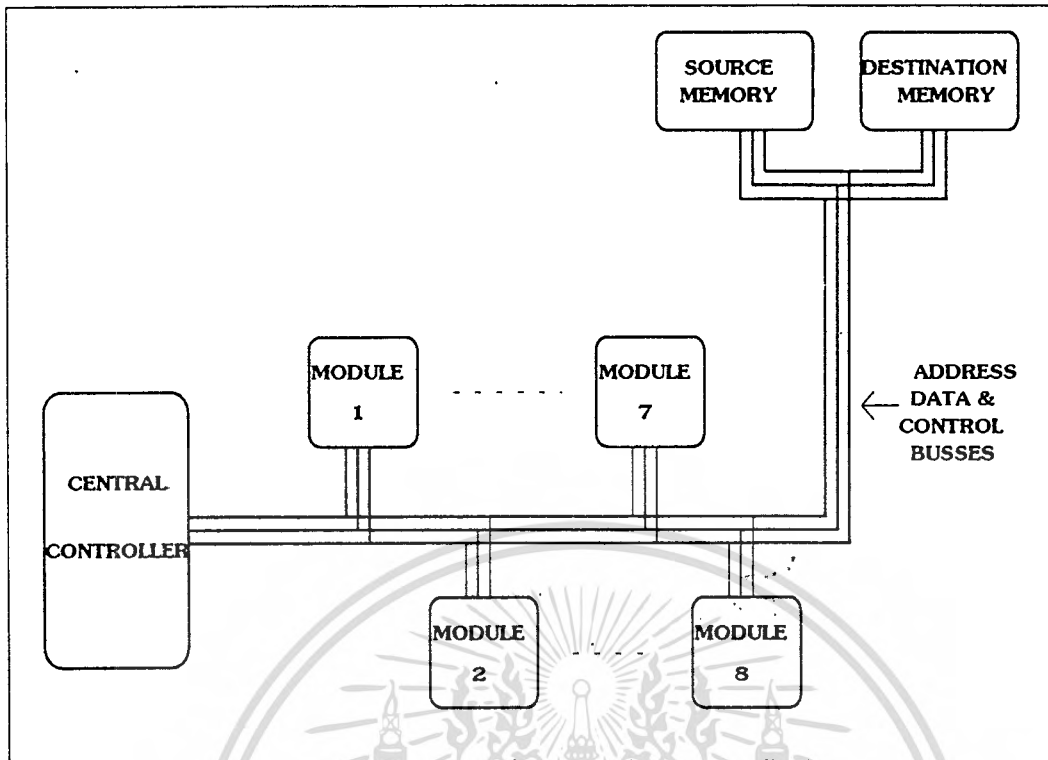
การประมวลผลภาพดิจิทัลมักเกี่ยวข้องกับกระบวนการที่อยู่ในรูปแบบของอัลกอริธึม(algorithmic form)เสียเป็นส่วนใหญ่ ซึ่งถ้าไม่รวมเอาอุปกรณ์ในการรับและแสดงภาพแล้วฟังก์ชันที่ใช้ในการประมวลผลภาพเกือบทั้งหมดสามารถปฏิบัติได้ด้วยการใช้ซอฟต์แวร์ ดังนั้นจึงมีเพียงเหตุผลเดียวที่จะนำเอาฮาร์ดแวร์มาใช้ในการประมวลผลภาพก็คือความต้องการในด้านความเร็วหรือเพื่อให้อยู่เหนือขีดจำกัดความสามารถของเครื่องคอมพิวเตอร์พื้นฐานที่ใช้กันอยู่ อย่างไรก็ตามเทคนิคที่ใช้ในการประมวลผลภาพแต่ละวิธีก็จะเหมาะสมและใช้ได้กับงานในแต่ละลักษณะเท่านั้น เทคนิคที่ใช้ได้ดีกับงานในลักษณะหนึ่งอาจจะไม่ให้ผลที่ดีพอเมื่อนำไปใช้กับงานในอีกลักษณะหนึ่ง การเลือกใช้ซอฟต์แวร์หรือฮาร์ดแวร์ที่ออกแบบมาเฉพาะงานจึงน่าจะเป็นทางเลือกที่จะคุ้มค่าและน่าสนใจอีกทางหนึ่ง

1.4 สถาปัตยกรรมแบบโมดูลาร์ (Modular Architecture)

สถาปัตยกรรมแบบโมดูลาร์เป็นลักษณะโครงสร้างที่มีการแบ่งการทำงานออกเป็นหน่วยย่อย ๆ หรือโมดูลที่เป็นอิสระต่อกัน ซึ่งเหมาะสมสำหรับการนำมาใช้กับวงจรการประมวลผลภาพเนื่องจากธรรมชาติและขั้นตอนในการประมวลผลภาพนั้นมีความแตกต่างกันตามลักษณะของข้อมูลภาพ สถาปัตยกรรมแบบโมดูลาร์จะช่วยอำนวยความสะดวกและกำหนดลักษณะขั้นตอนในการประมวลผลภาพได้ตรงเป้าหมายมากกว่า

ในแต่ละโมดูลสามารถออกแบบให้ตอบสนองต่อรูปแบบของการประมวลผลภาพเฉพาะอย่าง ดังรูปที่ 1.1 แสดงให้เห็นลักษณะโครงสร้างทั่วไปของสถาปัตยกรรมแบบโมดูลาร์ แต่ละโมดูลสามารถติดต่อกับหน่วยความจำต้นทางและปลายทางได้ หน่วยควบคุมหลักจะทำหน้าที่ควบคุมดูแลหน่วยความจำทั้งสองและการปฏิบัติงานของโมดูลอิสระทั้งหมด โดยการควบคุมผ่านทางคำสั่งต่าง ๆ เมื่อโมดูลใดได้รับคำสั่งก็จะเริ่มต้นปฏิบัติงานตามลักษณะหน้าที่ที่ได้รับการออกแบบไว้ การออกคำสั่งสามารถเลือกปฏิบัติกับโมดูลใดโมดูลหนึ่งหรือหลาย ๆ โมดูลตามลำดับก็ได้

จากข้อได้เปรียบของสถาปัตยกรรมแบบโมดูลาร์ ลักษณะโครงสร้างแบบนี้จึงมีความยืดหยุ่นและประหยัมากกว่า โมดูลของอัลกอริธึมใหม่ ๆ สามารถนำมาเปลี่ยนแทนที่ลงไปในระบบได้ง่าย การออกแบบในลักษณะของวงจรรวมเฉพาะกิจทำให้แต่ละโมดูลเป็นอิสระต่อกันและสามารถเลือกใช้เฉพาะโมดูลที่เหมาะสมตามลักษณะของงานที่ต้องการได้ และยังสามารถรองรับการขยายตัวของงานในอนาคตได้อีกด้วย ท้ายที่สุดผู้ใช้อย่างมีทางเลือกระหว่างความเร็ว ประสิทธิภาพ และการประหยัดค่าใช้จ่ายได้โดยการเลือกที่จะใช้อัลกอริธึมใหม่ ๆ ในรูปของโมดูลใหม่หรือในรูปของซอฟต์แวร์



รูปที่ 1.1 แสดงลักษณะโครงสร้างทั่วไปของสถาปัตยกรรมแบบโมดูลาร์

1.5 ภาษา VHDL (VHSIC Hardware Description Language)

VHDL เป็นภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ในลักษณะของข้อความ เป็นภาษาที่ถูกนำมาใช้เป็นประโยชน์ในการออกแบบอุปกรณ์ฮาร์ดแวร์ต่าง ๆ ทั้งในระดับเทกองถึงระบบดิจิทัลที่ซับซ้อน ทั้งนี้เนื่องจาก VHDL เป็นภาษาที่เหมาะสมในการนำมาใช้ในการเลียนแบบการทำงานของอุปกรณ์ อีกทั้งยังมีความยืดหยุ่นและไม่ถูกจำกัดโดยความสามารถทางเทคโนโลยีใด ๆ ทำให้ประหยัดเวลาและลดค่าใช้จ่ายในการออกแบบ จึงได้มีการนำมาใช้อย่างกว้างขวางในวงการอุตสาหกรรม

ปัจจุบันเทคโนโลยีในการผลิตอุปกรณ์ฮาร์ดแวร์ที่ใช้กันในอุตสาหกรรมการผลิตอุปกรณ์ชิ้นส่วนทางอิเล็กทรอนิกส์และคอมพิวเตอร์ได้มีการพัฒนาและก้าวล้ำไปเป็นอย่างมาก อุปกรณ์ที่ผลิตขึ้นมีขนาดเล็กแต่มีความซับซ้อนภายในประกอบด้วยวงจรขนาดเล็กจำนวนมากมายิ่ง ดังนั้นในการออกแบบวงจรที่มีขนาดใหญ่ (VLSI: Very Large Scale Integrated Circuit) และมีความซับซ้อนเช่นนี้ จำเป็นที่จะต้องอาศัยวิธีการออกแบบวงจรที่มีประสิทธิภาพและใช้อุปกรณ์หรือเครื่องมือที่ทันสมัยเช่นคอมพิวเตอร์ช่วยในการออกแบบเพื่อให้ได้ผลผลิตที่มีคุณภาพรวมทั้งต้องประหยัดเวลาและค่าใช้จ่ายด้วย

คงจะเป็นการเสียเวลาและไม่เกิดประโยชน์อันใด ถ้ากระบวนการในการออกแบบยังคงดำเนินไปโดยไม่รู้ว่าในแต่ละขั้นตอนหรือในแต่ละส่วนของการออกแบบสามารถทำงานได้ตรงตามที่คาดหวังไว้หรือไม่ เนื่องจากในปัจจุบันวงจรที่มีขนาดใหญ่สามารถจุทรานซิสเตอร์ได้มากถึงจำนวนแสนหรือล้านตัว ซึ่งถ้าหากมีความผิดพลาดเกิดขึ้นนั้นย่อมหมายถึงค่าใช้จ่ายและเวลาที่ต้องสูญเสียไป ดังนั้นการออกแบบในแต่ละขั้นตอนเราจะต้องแน่ใจว่าแต่ละส่วนสามารถทำงานได้ถูกต้อง และยังคงทำงานได้ถูกต้องเมื่อนำมาประกอบกันเป็นวงจรรวมขนาดใหญ่

การออกแบบวงจรในอดีตนักออกแบบวงจรส่วนมากมักจะเริ่มต้นด้วยการออกแบบวงจรในระดับเกต (gate-level) ก่อนซึ่งเป็นลักษณะของการออกแบบจากล่างขึ้นบน (Bottom-Up design) การออกแบบในลักษณะนี้เหมาะสำหรับวงจรที่มีความสลับซับซ้อนน้อย ๆ และสามารถแก้ไขคนเดียวในขั้นตอนการออกแบบทั้งหมดได้ แต่ในปัจจุบันเรากำลังเกี่ยวข้องกับการออกแบบวงจรรวมเฉพาะกิจ (ASIC: Application Specific Integrated Circuit) ที่มีขนาดใหญ่และมีความสลับซับซ้อนเกินกว่าที่คน

เพียงคนเดียวจะสามารถจัดการได้ ดังนั้นเราจึงจำเป็นต้องมีวิธีหรือหนทางใหม่ ๆ เพื่อนำมาใช้ในการจัดการกับการกับการออกแบบวงจรในลักษณะนี้และ VHDL ก็เป็นวิธีหนึ่งที่น่ามาใช้ในการจัดการกับปัญหาในลักษณะนี้ได้เป็นอย่างดี [1]

1.6 ลำดับการเรียบเรียงภายในวิทยานิพนธ์ (Organization of The Thesis)

เนื้อหาภายในวิทยานิพนธ์ฉบับนี้ได้มีการเรียบเรียงแบ่งออกเป็น 9 บท บทที่ 1 กล่าวถึงวัตถุประสงค์ แนวความคิดและหลักการที่นำเสนอ รวมถึงการแนะนำให้รู้จักกับสถาปัตยกรรมแบบโมดูลาร์และภาษา VHDL บทที่ 2 กล่าวถึงเทคนิคที่ใช้ในการประมวลผลภาพคือ Histogram Equalization ซึ่งเป็นเทคนิคที่ใช้ในการปรับปรุงคุณภาพของภาพเพื่อบันทึกแสดงรายละเอียดของภาพให้เห็นเด่นชัดขึ้นและ Edge Detection ซึ่งเป็นเทคนิคที่ใช้ในการตรวจหาขอบของวัตถุเพื่อการจำแนกรายละเอียดของวัตถุในภาพ บทที่ 3 แนะนำให้รู้จัก VHDL อย่างละเอียด โดยจะกล่าวถึงลักษณะโครงสร้างและองค์ประกอบต่าง ๆ ไวยากรณ์ การบรรยายการทำงานของอุปกรณ์ทั้งในลักษณะเชิงพฤติกรรม เชิงข้อมูล และเชิงโครงสร้าง รวมถึงการทดสอบและเลียนแบบการทำงานของอุปกรณ์

บทที่ 4 กล่าวถึงโครงสร้างสถาปัตยกรรมของระบบ ลักษณะการทำงาน การออกแบบ ยูทิลิตี้(utilities)และอุปกรณ์ย่อยต่าง ๆ โครงสร้างสถาปัตยกรรมของหน่วยความจำ การบรรยายการทำงานของระบบและหน่วยความจำทั้งในเชิงพฤติกรรมและเชิงโครงสร้าง บทที่ 5, 6, 7 กล่าวถึงโครงสร้างสถาปัตยกรรม ลักษณะการทำงานและการออกแบบของวงจรควบคุม วงจรปรับปรุงคุณภาพของภาพ และวงจรตรวจหาขอบของวัตถุในภาพตามลำดับ การบรรยายในเชิงพฤติกรรมและเชิงโครงสร้าง การบรรยายเชิงข้อมูลของวงจรควบคุมภายในแต่ละวงจร

บทที่ 8 แสดงรูปแบบและผลการทดสอบของวงจรรวมทั้งในระดับพฤติกรรมและระดับโครงสร้าง การกำหนดระดับในการทดสอบ บทที่ 9 สรุปและวิจารณ์

ภาคผนวกแบ่งออกเป็น 2 บท ภาคผนวก A รวบรวมและแสดงยูทิลิตี้ทั้งหมดที่ใช้ ภาคผนวก B รวบรวมและแสดงการบรรยายการทำงานของวงจรหลักทั้งหมด

บทที่ 2

ทฤษฎีการประมวลผลภาพดิจิทัล

ในบทนี้จะได้กล่าวถึงหลักการของเทคนิคในการประมวลผลภาพดิจิทัล 2 วิธีได้แก่ เทคนิคในการปรับปรุงคุณภาพของภาพโดยวิธี Histogram Equalization และเทคนิคในการตรวจหาขอบของวัตถุในภาพโดยวิธี Edge Detection ซึ่งเป็นหลักการขั้นพื้นฐานที่ถูกนำมาใช้กันโดยทั่วไป

2.1 การปรับปรุงคุณภาพของภาพโดยวิธี Histogram Equalization

วัตถุประสงค์หลักของเทคนิคในการปรับปรุงคุณภาพของภาพหรือบางครั้งเรียกว่า การเน้นภาพ (Image Enhancement) ก็เพื่อให้ภาพที่ผ่านกระบวนการปรับปรุงแล้วมีคุณภาพดีกว่าภาพต้นแบบเดิม เป็นวิธีการที่เน้นแสดงรายละเอียดของภาพให้เห็นเด่นชัดขึ้น ในการเน้นอาจเลือกเน้นเฉพาะบางส่วนหรือเน้นข้อมูลส่วนใหญ่ที่บรรจุอยู่ในภาพ เป็นการเน้นในแง่คุณภาพเพื่อให้สามารถประมวลผลด้วยสายตาได้ดีขึ้นหรือดึงลักษณะบางอย่างให้เห็นเด่นชัดจากส่วนประกอบภูมิหลังอื่น ๆ มากกว่าที่จะเป็นการวิเคราะห์ในแง่ของปริมาณหรือการจำแนกประเภท

ภาพเป็นสื่อที่ปรากฏจากการที่แสงตกกระทบวัตถุและสะท้อนสู่จากรับภาพ ภาพดังกล่าวจำเป็นต้องผ่านกระบวนการแปลงให้อยู่ในรูปของสัญญาณไฟฟ้า จากนั้นจึงแปลงเป็นตัวเลขโดยการกำหนดการเรียงต่อของจุดแต่ละจุดเพื่อให้ปรากฏเป็นภาพ ซึ่งจุดต่าง ๆ เหล่านี้เรียกว่าพิกเซล (pixels) การกำหนดการเก็บค่าในพิกเซลนั้นเป็นการเก็บระดับความเข้มของแสง ซึ่งหากมีการใช้จำนวนบิต (bit) ในการเก็บค่าของจุดมากก็จะทำให้ระดับของความเข้มจำแนกได้มากระดับขึ้น เช่น พิกเซลที่มีขนาด 4 บิตจะให้ระดับความเข้ม $2^4 = 16$ ระดับและพิกเซลขนาด 8 บิตจะให้ระดับความเข้ม $2^8 = 256$ ระดับ

สำหรับในภาพขาวดำจะเป็นความแตกต่างในระดับความเข้มสีเทา จากสีขาวซึ่งมีการสะท้อนแสงสูงจนถึงสีดำซึ่งการสะท้อนต่ำมากเข้าใกล้ศูนย์ เป็นการแสดงถึงความสามารถในการดูดกลืนพลังงานในระดับที่มีมากกว่าการสะท้อน ในกรณีของข้อมูลภาพตัวเลขเช่น ภาพที่มีความแตกต่างในระดับความเข้มถึง 256 ระดับจะหมายถึงความแตกต่างของระดับสีเทากถึง 256 ระดับ ทำให้การแยกแยะระดับสีเทาเป็นไปอย่างยากลำบาก เทคนิคการเน้นภาพจะช่วยให้เห็นความแตกต่างระหว่างระดับสีเทาให้มากขึ้น

เทคนิคการเน้นภาพโดยเพิ่มความแตกต่างของระดับความเข้ม (Histogram Equalization) เป็นวิธีที่ใช้ในการแปลงระดับความเข้ม (gray level) ของพิกเซลแต่ละพิกเซลให้อยู่ในค่าที่เหมาะสมเพื่อให้เห็นความแตกต่างของภาพให้เด่นชัดขึ้น [8] ซึ่งบางพิกเซลอาจจะถูกแปลงให้มีระดับความเข้มสูงขึ้นหรือต่ำลงหรือบางพิกเซลอาจจะไม่มีการเปลี่ยนแปลงระดับความเข้มเลยก็ได้

ให้ r แทนระดับความเข้มของภาพที่จะนำมาเน้น

สมมติว่าค่าของพิกเซลมีความต่อเนื่องและถูกนอร์มอลไลซ์ (normalized) แล้ว ดังนั้นค่าของพิกเซลจะอยู่ในช่วง $[0, 1]$ โดย $r = 0$ แทนสีดำและ $r = 1$ แทนสีขาว

สำหรับ r ใด ๆ ในช่วง $[0, 1]$ เราสามารถเขียนคณิตศาสตร์ของการแปลงได้ว่า

$$s = T(r) \quad (1)$$

ซึ่งจะให้ค่าระดับ s สำหรับค่าของพิกเซล r ทุก ๆ ค่าที่มีอยู่ในภาพต้นแบบ

เราสามารถกล่าวได้ว่าฟังก์ชันการแปลง (transformation function) ของความสัมพันธ์

(1) สอดคล้องกับ เงื่อนไข

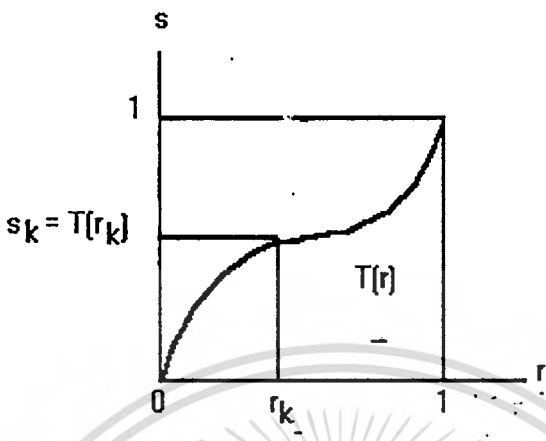
(a) $T(r)$ เป็นค่า ๆ เดียวและมีค่าเพิ่มขึ้นอย่างสม่ำเสมอ (monotonically increasing) ในช่วง

$$0 \leq r \leq 1$$

(b) $0 \leq T(r) \leq 1$ สำหรับ $0 \leq r \leq 1$

ในรูปที่ 2.1 แสดงความสอดคล้องของฟังก์ชันการแปลงกับเงื่อนไข

เราสามารถบอกลักษณะของระดับความเข้มของภาพต้นแบบและภาพที่ถูกแปลงแล้วได้จากฟังก์ชันแสดงความน่าจะเป็นของความหนาแน่น (probability density function) ของพิกเซลที่ระดับต่าง ๆ $P_r(r)$ และ $P_s(s)$



รูปที่ 2.1 แสดงฟังก์ชันการแปลงของระดับความเข้มจากทฤษฎีความน่าจะเป็น ถ้าเรารู้ค่า $P_r(r)$ และ $T(r)$ แล้วจะได้ว่า

$$P_s(s) = P_r(r) \frac{dr}{ds} \quad r=T^{-1}(s) \tag{2}$$

เทคนิคการปรับปรุงคุณภาพแบบนี้จะปรับปรุงภาพที่ปรากฏโดยการควบคุมฟังก์ชันแสดงความน่าจะเป็นของความหนาแน่นของระดับความเข้มโดยใช้ฟังก์ชันการแปลง $T(r)$ พิจารณาฟังก์ชันการแปลง

$$s = T(r) = \int_0^r P_r(w) dw \quad 0 \leq r \leq 1 \tag{3}$$

w เป็นตัวแปร dummy ในการอินทิเกรต

จากความสัมพันธ์ (3) อนุพันธ์ของ s เมื่อเทียบกับ r คือ

$$\frac{ds}{dr} = P_r(r) \tag{4}$$

แทนค่า dr/ds ลงในความสัมพันธ์ (2) จะได้

$$\begin{aligned} P_s(s) &= P_r(r) \frac{1}{P_r(r)} \quad r=T^{-1}(s) \\ &= 1 \quad 0 \leq s \leq 1 \end{aligned} \tag{5}$$

ซึ่งแสดงให้เห็นว่าแม้ภาพจะผ่านการแปลงแล้ว ความหนาแน่นก็ยังคงเหมือนเดิม คืออยู่ในช่วง $0 \leq s \leq 1$ เช่นเดิม

เพื่อนำไปใช้ประโยชน์ในการประมวลผลทางดิจิทัลก็ต้องปรับรูปแบบของความสัมพันธ์เหล่านี้ให้อยู่ในรูปของดิสครีต (discrete form) จะได้

$$P_r(r_k) = \frac{n_k}{n} \quad 0 \leq r_k \leq 1 ; k = 0, 1, \dots, L-1 \tag{6}$$

- โดยที่ L = จำนวนของระดับความเข้ม
- $P_r(r_k)$ = ความน่าจะเป็นของระดับความเข้มที่ k
- n_k = จำนวนครั้งที่มึระดับความเข้มนี้ปรากฏในภาพ
- n = จำนวนพิกเซลทั้งหมดในภาพ

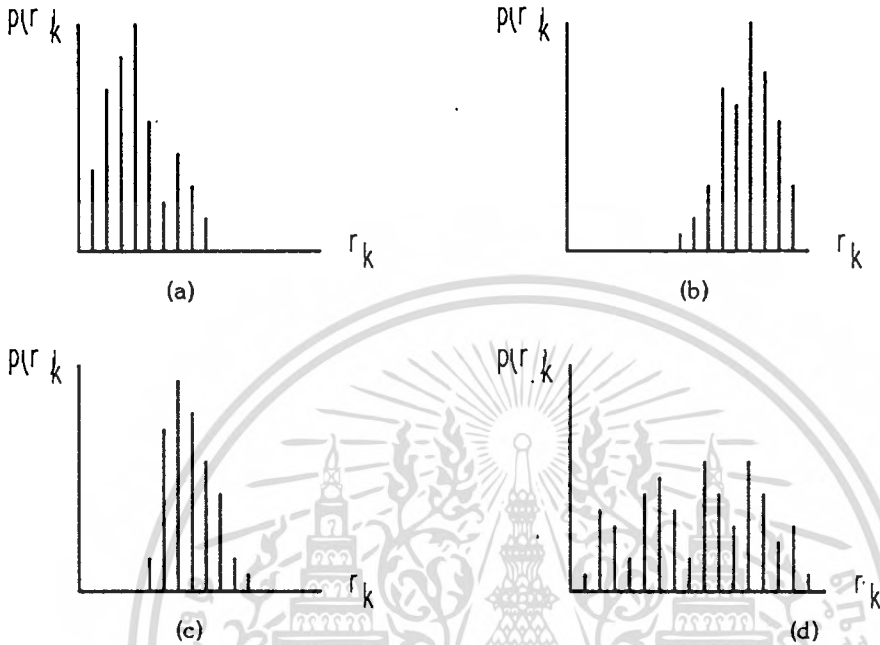
เราเรียกรูปพล็อตแสดงความสัมพันธ์ระหว่าง $P_r(r_k)$ และ r_k ว่า Histogram และเรียกเทคนิคที่ใช้ว่า Histogram Equalization หรือ Histogram Linearization

เอกสารนี้เป็นเอกสารลิขสิทธิ์ส่วนบุคคลซึ่งใช้ในการศึกษาเท่านั้น ไม่อนุญาตให้เผยแพร่โดยไม่ได้รับอนุญาตจากการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปแบบดีสคริตของความสัมพันธ์ (3) เขียนใหม่ได้เป็น

$$s_k = T(r_k) = \sum_{j=0}^k \frac{n_j}{n} = \sum_{j=0}^k P_r(r_j) \quad 0 \leq r_k \leq 1 ; \quad k = 0, 1, \dots, L-1 \quad (7)$$



รูปที่ 2.2 แสดง Histogram ของภาพ 4 แบบ

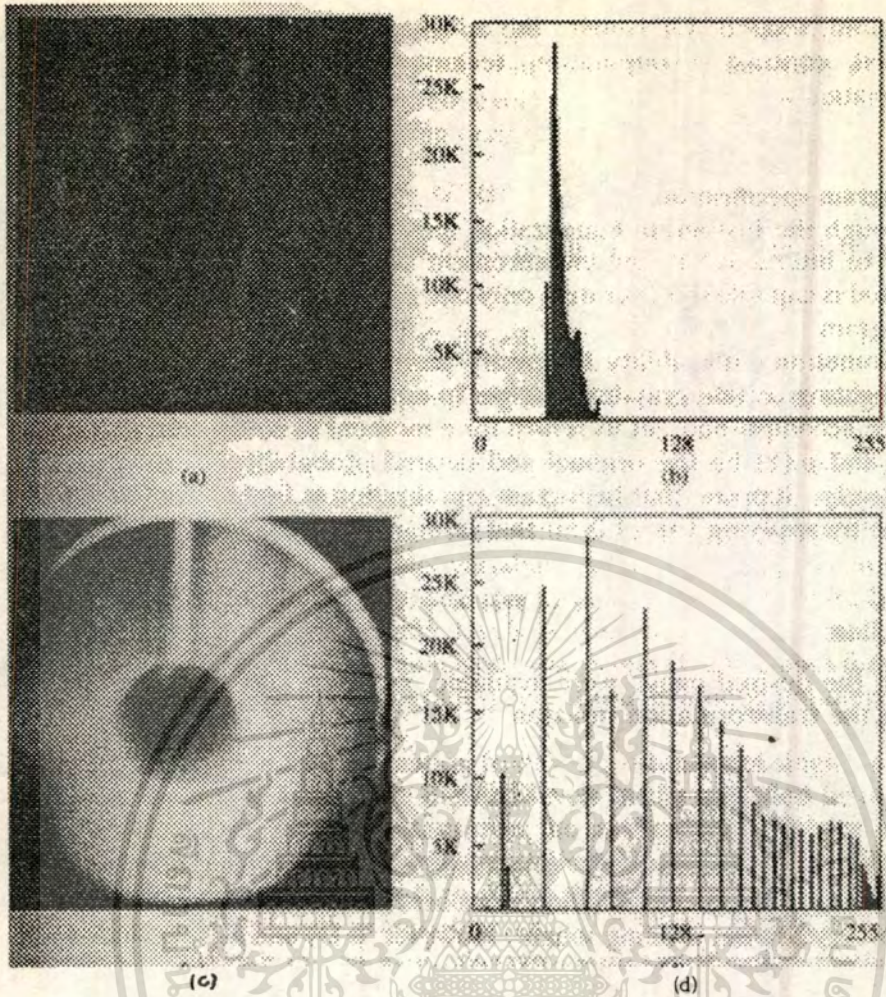
(a) ระดับความเข้มต่อเนื่องทางด้านมืด ซึ่ง Histogram แบบนี้จะได้จากภาพที่มีลักษณะค่อนข้างมืดตลอดทั้งภาพ (b) ลักษณะตรงข้ามกับรูป (a), (c) ระดับความเข้มต่อเนื่องข้างแคบแสดงว่ามีความแตกต่างของระดับความเข้มต่ำ (d) ระดับความเข้มมีการกระจายออกแสดงว่ามีความแตกต่างของระดับความเข้มมาก

รูปที่ 2.3 แสดงภาพให้เห็นประโยชน์ของขบวนการ Histogram Equalization รูป (a) แสดงภาพขนาด 512 X 512, 8 บิต ของการเชื่อมโยงด้วยความร้อนสูง ลักษณะของภาพค่อนข้างดำในช่วงการกระจายความเข้มแคบและอยู่ต่อเนื่องทางด้านมืดดังในรูป (b) แกนบนแสดงค่าช่วงความเข้ม [0, 255] ซึ่งเป็นค่าช่วงที่เป็นไปได้ของภาพขนาด 8 บิต แกนตั้งแสดงจำนวนของพิกเซลที่ระดับความเข้มต่าง ๆ รูป (c) และ (d) แสดงผลที่ได้จากขบวนการ Histogram Equalization

ตัวอย่าง สมมติว่ามีภาพขนาด 64x64 และมีระดับความเข้มแตกต่างกัน 8 ระดับที่มีการกระจายดังแสดงในตาราง

r_k	n_k	$P_r(r_k) = n_k/n$
$r_k = 0$	790	0.19
$r_k = 1/7$	1023	0.25
$r_k = 2/7$	850	0.21
$r_k = 3/7$	656	0.16
$r_k = 4/7$	329	0.08
$r_k = 5/7$	245	0.06
$r_k = 6/7$	122	0.03
$r_k = 1$	81	0.02

ตารางที่ 2.1 แสดงจำนวนจุดภาพในแต่ละระดับความเข้มและจะน้ำหนักจะเป็นของระดับความเข้ม



รูปที่ 2.3

(a) ภาพต้นแบบ (b) ฮิสโตแกรมของภาพต้นแบบ

(c) ภาพที่ผ่านขั้นตอนการ Histogram Equalization

(d) ฮิสโตแกรมของภาพที่ผ่านขั้นตอนการ Histogram Equalization

เราสามารถหาฟังก์ชันการแปลงได้จากความสัมพันธ์ (7)

$$\begin{aligned} s_0 = T(r_0) &= \sum_{j=0}^0 P_r(r_j) \\ &= P_r(r_0) \\ &= 0.19 \end{aligned}$$

$$\begin{aligned} s_1 = T(r_1) &= \sum_{j=0}^1 P_r(r_j) \\ &= P_r(r_0) + P_r(r_1) \\ &= 0.44 \end{aligned}$$

ในทำนองเดียวกัน

$$s_2 = 0.65$$

$$s_5 = 0.95$$

$$s_3 = 0.81$$

$$s_6 = 0.98$$

$$s_4 = 0.89$$

$$s_7 = 1.00$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เนื่องจากมีระดับความเข้มเพียง 8 ระดับเท่านั้นจึงต้องมีการกำหนดให้ค่าแปลงที่ได้มีค่าเท่ากับค่าระดับที่ใกล้เคียงที่สุด ดังนั้นจะได้ว่า

$$s_0 \approx 1/7 \quad s_4 \approx 6/7$$

$$s_1 \approx 3/7 \quad s_5 \approx 1$$

$$s_2 \approx 5/7 \quad s_6 \approx 1$$

$$s_3 \approx 6/7 \quad s_7 \approx 1$$

จะเห็นได้ว่ามีเพียง 5 ระดับที่แตกต่างกัน เราจะกำหนดสัญลักษณ์ใหม่ดังนี้

$$s_0 \approx 1/7 \quad s_3 \approx 6/7$$

$$s_1 \approx 3/7 \quad s_4 \approx 1$$

$$s_2 \approx 5/7$$

ในการแปลงค่าระดับความเข้มใหม่ จะเป็นดังนี้

ระดับความเข้มเดิม	จำนวนพิกเซล	แปลงเป็นระดับความเข้มใหม่
r_0	790	s_0
r_1	1023	s_1
r_2	850	s_2
$r_3 + r_4$	$656 + 329$	s_3
$r_5 + r_6 + r_7$	$245 + 122 + 81$	s_4

ตารางที่ 2.2 แสดงค่าระดับความเข้มเดิมและใหม่

ระดับความเข้มของภาพที่ผ่านขั้นตอนการ Histogram Equalization จะมีการกระจายออก และมักจะมีการกระจายไปทางสีขาวเสมอ ขั้นตอนการนี้ไม่เพียงแต่จะเพิ่มช่วงของค่าระดับความเข้มเท่านั้น แต่ยังเพิ่มความแตกต่างระหว่างสีให้มากขึ้นด้วย

2.2 การตรวจหาขอบของวัตถุในภาพโดยวิธี Edge Detection

การตรวจหาขอบของวัตถุในภาพเป็นหลักการที่สำคัญอย่างหนึ่งที่มีคนนำมาใช้ในการประมวลผลภาพเพื่อการเปรียบเทียบ การจดจำหรือเพื่อการจำแนกรายละเอียดของวัตถุในภาพ โดยอาศัยหลักการพื้นฐานของความเหมือนหรือต่างกันและความไม่ต่อเนื่องของระดับสีของจุดภาพ (pixels) วิธีการที่นำมาใช้ก็คือการหาค่าอนุพันธ์อันดับที่หนึ่งของภาพโดยหาค่าเกรเดียนต์ (gradient) ของภาพ $f(x,y)$ ที่ตำแหน่ง (x,y) ในวิทยานิพนธ์นี้ได้เลือกใช้วิธีของโซเบล (Sobel operators) [7] ดังแสดงในรูปที่ 2.4 ซึ่งวิธีนี้มีข้อดีที่ให้ผลดีทั้งในด้านความแตกต่างและความราบเรียบของภาพ (differencing and smoothing effect) เนื่องจากการหาอนุพันธ์จะไปขยายข้อมูลรบกวน (noise) ที่มีอยู่ในภาพให้เพิ่มมากขึ้น ดังนั้นเรื่องผลของความราบเรียบจึงเป็นคุณสมบัติที่น่าสนใจของวิธีโซเบล

จากรูปที่ 2.4 นำมาเขียนเป็นสมการสำหรับหาค่าเกรเดียนต์ได้ดังสมการที่ (1)–(3) โดย Z_1 – Z_9 เป็นค่าระดับความเข้มที่ตำแหน่งต่าง ๆ ในบริเวณพื้นที่ของภาพซึ่งเป็นแมทริกซ์ขนาด 3×3 ส่วน G_x และ G_y เป็นค่าเกรเดียนต์ในแนวแกน X และแกน Y และ Vf เป็นค่าเกรเดียนต์ในแนวแกนทั้งสองรวมกัน

$$G_x = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3) \quad (1)$$

$$G_y = (Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z_7) \quad (2)$$

$$Vf \approx |G_x| + |G_y| \quad (3)$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

(a)

-1	-2	-1
0	0	0
1	2	1

(b)

-1	0	1
-2	0	2
-1	0	1

(c)

รูปที่ 2.4

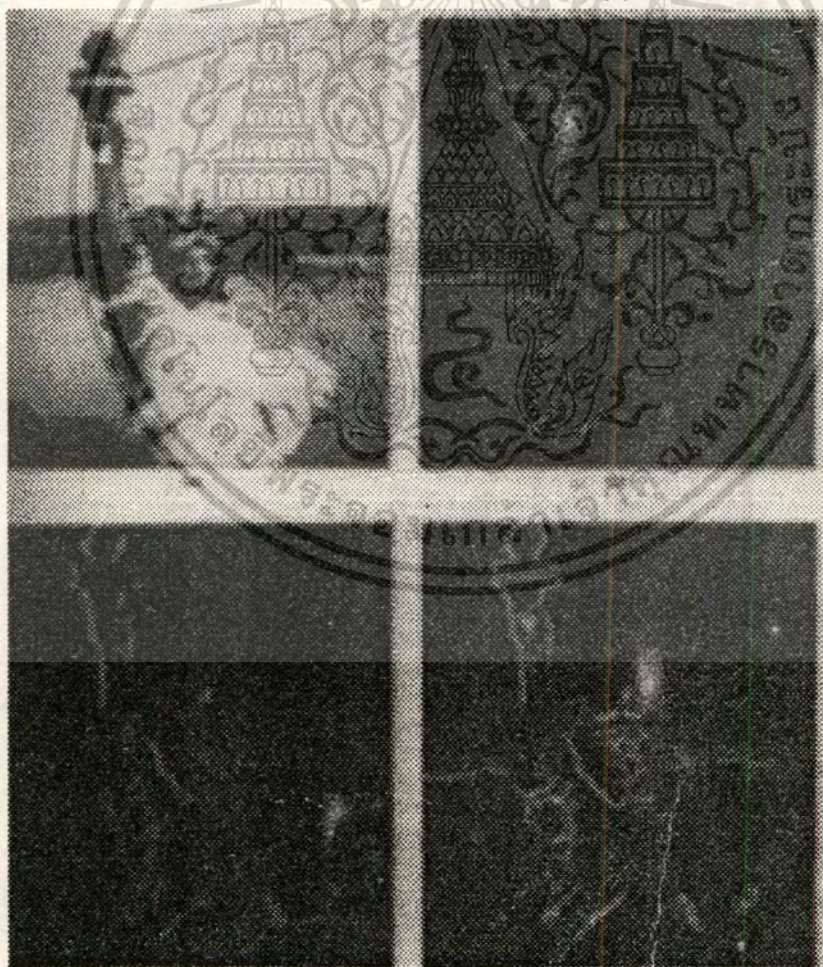
(a) แสดงบริเวณพื้นที่ของภาพเป็นแมทริกซ์ขนาด 3×3

(b) mask สำหรับคำนวณค่า G_x ที่จุดกลางของแมทริกซ์ขนาด 3×3

(c) mask สำหรับคำนวณค่า G_y ที่จุดกลางของแมทริกซ์ขนาด 3×3

(a)

(b)



(c)

(d)

รูปที่ 2.5

(a) ภาพต้นแบบ (b) ภาพที่ได้จากการหา G_x

เอกสารนี้เป็นเอกสารที่สงวนไว้ (c) ภาพที่ได้จากการหา G_y (d) ภาพที่ได้จากการหาเกรเดียนต์

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 2.5 แสดงภาพการหาขอบของวัตถุในภาพ รูป (a) แสดงภาพต้นแบบ รูป (b) แสดงผลที่ได้จากการคำนวณค่า I_{G_x} โดยใช้ mask จากรูป 2.4(b) สังเกตว่าแถบในแนวตั้งคือแถบ X ดังนั้นภาพที่ได้จะมีการตอบสนองของขอบวัตถุที่ตั้งฉากกับแถบ X รูป (c) แสดงผลที่ได้จากการคำนวณค่า I_{G_y} โดยใช้ mask จากรูป 2.4(c) แถบในแนวตั้งคือแถบ Y ดังนั้นภาพที่ได้จะมีการตอบสนองของขอบวัตถุที่ตั้งฉากกับแถบ Y รูป (d) แสดงภาพที่ได้จากการรวมผลการคำนวณทั้งสองเข้าด้วยกัน

ซึ่งจะได้ค่าเกรเดียนของตำแหน่งกลางของแมทริกซ์ขนาด 3×3 หนึ่งค่า สำหรับค่าต่อไปสามารถหาได้โดยเลื่อนตำแหน่งของแมทริกซ์ขนาด 3×3 ไปทีละหนึ่งตำแหน่งและคำนวณซ้ำใหม่ ซึ่งผลที่ได้จะเป็นค่าเกรเดียนของภาพทั้งหมดที่มีขนาดเท่ากับภาพต้นแบบเดิม

2.3 สรุป (Summary)

วัตถุประสงค์หลักของการปรับปรุงคุณภาพของภาพก็เพื่อให้สามารถประมวลผลด้วยสายตาได้ดีขึ้น อย่างไรก็ตามแม้ว่าจะมีเทคนิคต่าง ๆ มากมายที่ใช้ในการปรับปรุงคุณภาพของภาพ ในขั้นตอนสุดท้ายผู้ที่ตัดสินใจว่าวิธีการใดที่เหมาะสมที่สุดก็ควรจะเป็นผู้นั้นเอง ไม่มีวิธีการใดที่จะใช้ได้กับภาพทุกรูปแบบ การจะเลือกเอาเทคนิคใดเทคนิคหนึ่งมาใช้ที่เหมาะสมก็ขึ้นอยู่กับเงื่อนไขและลักษณะเฉพาะของข้อมูลภาพนั้นๆ ผลที่ได้จึงเป็นสิ่งยืนยันความถูกต้องในการเลือก Histogram Equalization ก็เป็นเทคนิคอย่างหนึ่งซึ่งมักจะถูกนำมาใช้ในการปรับปรุงภาพประเภทที่ค่อนข้างดำหรือมืดให้มีความสว่างมากขึ้น ทั้งนี้เนื่องจากวิธีการนี้ทำให้ระดับความเข้มของภาพมีการกระจายออกและแนวการกระจายจะไปในทางสว่างเสมอ

การตรวจหาขอบของวัตถุในภาพเป็นหนึ่งในเทคนิคที่นำมาใช้กันในสาขาที่เรียกว่าการวิเคราะห์ภาพ(image analysis) โดยทั่วไปขั้นตอนแรกในการวิเคราะห์ภาพจะต้องทำการแบ่งภาพออกเป็น ส่วน ๆ เสียก่อน หลักในการวิเคราะห์จะขึ้นอยู่กับคุณสมบัติของค่าระดับความเข้มคือ ความต่างและความเหมือน(discontinuity and similarity)ของแต่ละพิกเซล Edge Detection ก็อาศัยคุณสมบัตินี้ในการหาขอบของวัตถุโดยดูจากลักษณะการเปลี่ยนแปลงอย่างทันทีทันใดของค่าระดับความเข้ม เพราะขอบของวัตถุก็คือเส้นแบ่งเขตระหว่างสองพื้นที่ที่มีระดับความเข้มที่แตกต่างกันนั่นเอง

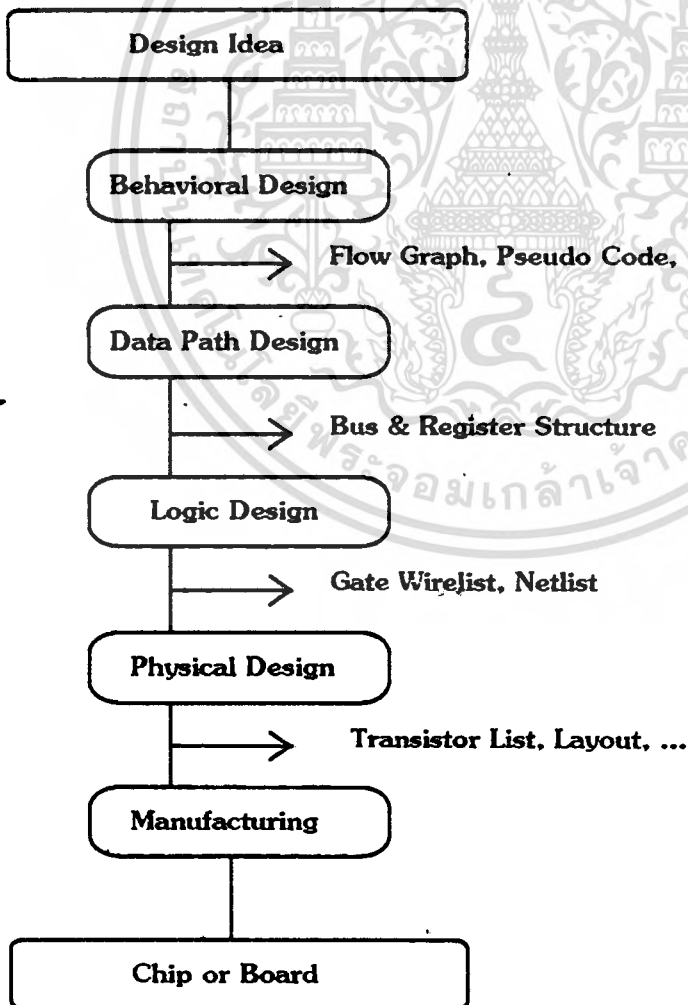
บทที่ 3

โครงสร้างของ VHDL

ในขณะที่ขนาดและความซับซ้อนของระบบดิจิทัลเพิ่มมากขึ้น คอมพิวเตอร์เพื่อช่วยในการออกแบบ (CAD: Computer Aided Design) ก็ได้ถูกนำเข้ามาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มมากขึ้นเช่นกัน อุปกรณ์และวิธีการออกแบบใหม่ๆ ได้ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบและภาษาสำหรับบรรยายอุปกรณ์ฮาร์ดแวร์ (HDL : Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการในการออกแบบระบบดิจิทัล

ในการออกแบบระบบดิจิทัล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและก็เข้าสู่กระบวนการออกแบบในขั้นต่อไป

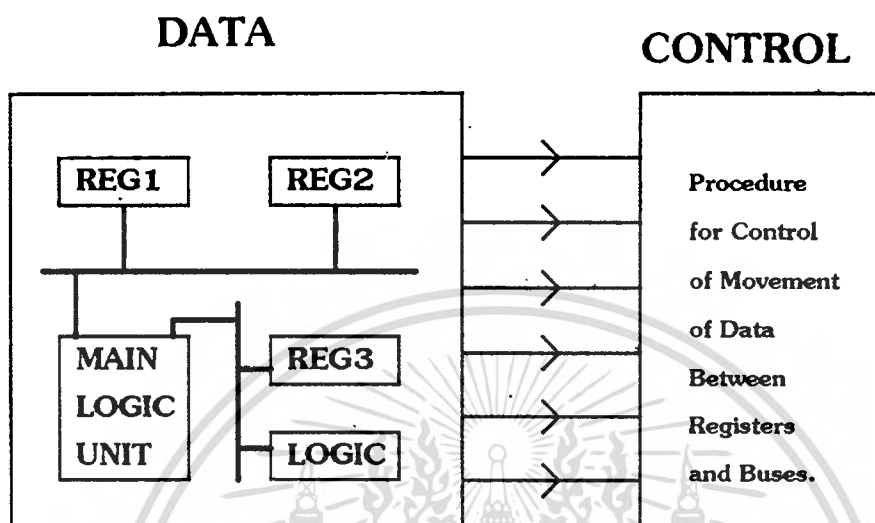
รูปที่ 3.1 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิทัลทั่วไป[5] ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบเสียก่อนและทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นภายในขั้นตอนนี้จึงมีความจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจจะเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (pseudo code) ก็ได้



เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ 2564 โดยมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์(register)และวงจรตรรกะ(logic)ที่จำเป็นทั้งหมดเพื่อนำมาประกอบกันเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง(unidirectional or bidirectional bus) กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรตรรกะก็จะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 3.2



รูปที่ 3.2 แสดงการออกแบบระบบเส้นทางของข้อมูล

การออกแบบวงจรตรรกะจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิปฟล็อป(flip flop)เป็นส่วนประกอบของอุปกรณ์ย่อยต่าง ๆ ได้แก่ รีจิสเตอร์เก็บข้อมูล บัสวงจรตรรกะ และส่วนควบคุมฮาร์ดแวร์ ซึ่งในขั้นสุดท้ายจะต้องออกมาเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟล็อปนั่นเอง

ต่อมาเป็นขั้นตอนของการเปลี่ยนเครือข่ายการโยงใยในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์และเลย์เอาต์(transistor list and layout) ในขั้นตอนนี้จะเกี่ยวข้องกับการจัดวางเกทและฟลิปฟล็อปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซลล์

ขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

3.1 แนะนำ VHDL (Introduction to VHDL)

ในช่วงฤดูร้อนของปี 1981 สถาบันเพื่อการป้องกัน (The Institute for Defence Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่งเพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า **VHDL** (VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวข้องกับวงจรรวมที่มีความเร็วสูงมาก(Very High Speed Integrated Circuit) ต่อมาในปี 1985, IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL [3] ในด้านของการกำหนดพฤติกรรมการทำงานของวงจร ทำให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งของวงจรดิจิทัลทั่วไปและในระบบที่แตกต่างออกไปเช่นพฤติกรรมการทำงานของระบบเรดาร์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ได้ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้สามารถที่จะถูกใช้ได้ตลอดในทุก ๆ ระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง(system level) จนถึงในระดับที่ต่ำกว่า(lower hardware

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไมอนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

levels)สามารถใช้ภาษาเดียวกันได้โดยตลอดทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

3.1.1 ข้อกำหนด (VHDL Requirements)

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

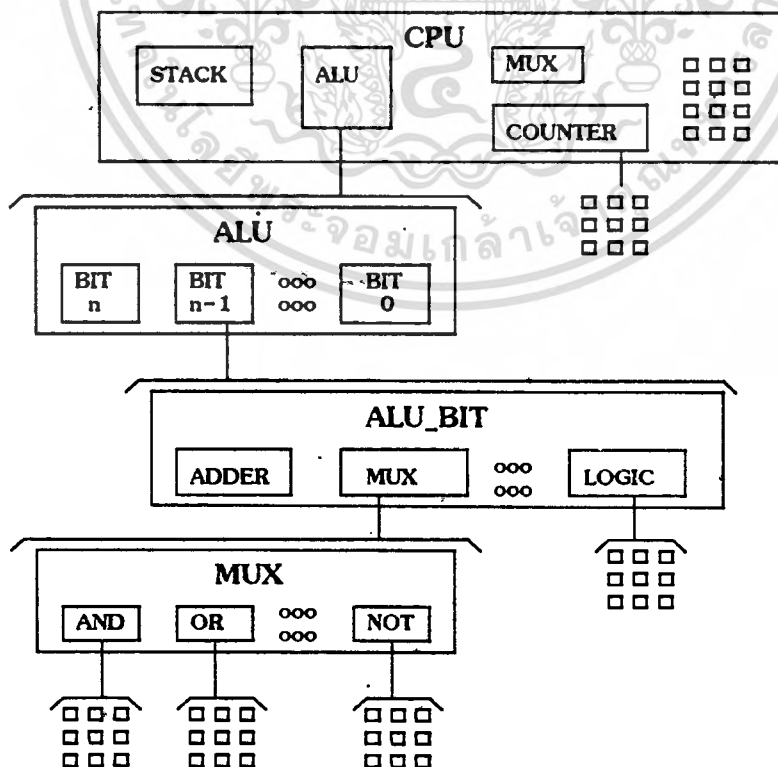
3.1.1.1 ลักษณะทั่วไป (General Features)

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

นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับบนคือระบบจนถึงระดับเกทอีกด้วย เนื่องจากการทำงานของระบบดิจิทัลจริง ๆ ทุก ๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อม ๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งใน VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้วความพร้อมเพรียงจะหมายถึงทุก ๆ คำสั่ง องค์ประกอบ เกทหรือวงจรตรรกะจะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อม ๆ กัน)

3.1.1.2 สนับสนุนการออกแบบแบบลำดับชั้น (Support for Design Hierarchy)

การออกแบบแบบลำดับชั้นก็เป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลาย ๆ ระดับ ในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตัวเองหรือถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อย ๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ ดังรูป 3.3



เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ 3.3 แสดงตัวอย่างการออกแบบแบบลำดับชั้นขนาดให้นำไปใช้ประโยชน์ด้านการคำนวณว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.1.1.3 ไบรารี (Library Support)

VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารีหรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่น ๆ สามารถนำไปใช้ได้ด้วย

3.1.1.4 ลำดับคำสั่ง (Sequential Statement)

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

การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

3.1.1.5 การกำหนดคุณสมบัติ (Generic Design)

นอกจากการกำหนดอินพุตและเอาต์พุตแล้ว เจ็อนไซอื่น ๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้น ๆ ภาษาสำหรับการออกแบบที่ดีควรจะช่วยให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วยเช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่น ๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

3.1.1.6 ชนิดของข้อมูล (Type Declaration and Usage)

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

3.1.1.7 โปรแกรมย่อย (Use of Subprograms)

ความสามารถในการใช้ฟังก์ชันและโพรซีเจอร์ (procedure) ก็เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL เราสามารถใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก การกำหนดตัวกระทำต่าง ๆ ทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

3.1.1.8 การควบคุมเวลา (Timing Control)

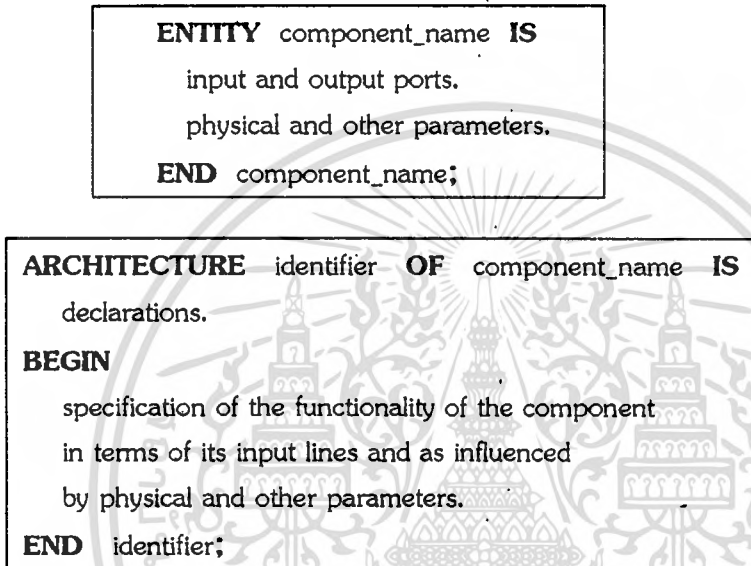
VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลา ก็สามารถทำได้โดยการกำหนดช่วงเวลาที่เหมาะสมหรือกำหนดให้มีการรูดอยเหตุการณ์ (event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

3.1.1.9 การกำหนดแบบโครงสร้าง (Structural Specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุก ๆ ระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวมทั้งเกิดจากองค์ประกอบย่อย ๆ ที่แตกต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

3.1.2 องค์ประกอบพื้นฐานใน VHDL (Basic Concepts in VHDL)

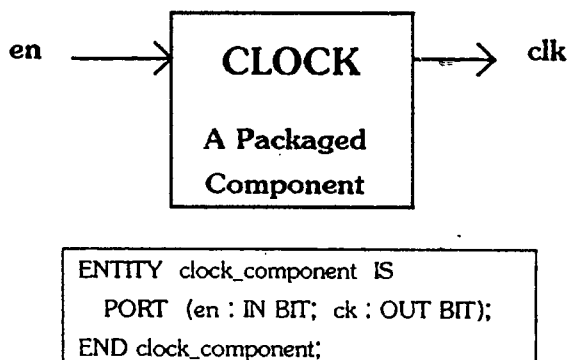
รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ(interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม(architecture) ดังแสดงในรูปที่ 3.4 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ **ENTITY** ตามด้วยชื่อขององค์ประกอบและคำ **IS** ภายในบรรยายถึงพอร์ตการติดต่อ อินพุต-เอาต์พุตพอร์ตขององค์ประกอบ ส่วนลักษณะภายนอกอื่น ๆ เช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า **ARCHITECTURE** ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุต-เอาต์พุตและพารามิเตอร์อื่น ๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูป การบรรยายหน้าที่ขององค์ประกอบเริ่มต้นหลังจากคำว่า **BEGIN** เป็นต้นไป



รูปที่ 3.4 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม

3.1.2.1 การกำหนดการเชื่อมต่อ (Interface Description)

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอกอื่น ๆ ดังตัวอย่างในรูปที่ 3.5 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ *clock_component* ตามด้วยคำว่า **PORT** และชื่อของพอร์ตอยู่ภายในวงเล็บ **IN** และ **OUT** กำหนดโหมดของสัญญาณเป็นอินพุตหรือเอาต์พุต **BIT** แสดงชนิดของข้อมูล



รูปที่ 3.5 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อของ *clock_component*

3.1.2.2 การกำหนดรูปแบบการบรรยาย (Architectural Description)

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเอาต์พุตในเทอมของอินพุตหรือในรูปขององค์ประกอบอื่น ๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ *clock_component* ในรูปที่ 3.6 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี *en* เป็นอินพุตและ *ck* เป็นเอาต์พุต **PROCESS** เป็นคำเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม ภายในไพรเซสกำหนดให้ *periodic* เป็นตัวแปรที่มีค่าเริ่มต้นเป็น '0' ถ้าสัญญาณ *en* มีค่าเป็น '1' ค่าของ *periodic* จะถูกคอมพลีเมนต์(complement)และส่งค่าให้กับ *ck* ซึ่งเป็นสัญญาณเอาต์พุต คำสั่ง **WAIT** กำหนดให้สัญญาณมีคาบเป็นเวลา 1 ไมโครวินาที

```

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

```

รูปที่ 3.6 แสดงการบรรยายเชิงพฤติกรรมของ *clock_component*

3.1.2.3 โปรแกรมย่อย (Subprograms)

การใช้ฟังก์ชันและไพรเซจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาขั้นสูงทั่วไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าเราใช้ฟังก์ชันแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรจริง ๆ ในขณะที่ถ้าเราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 3.7 แสดงการใช้ไพรเซจอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิตเป็นค่าจำนวนเต็ม รูปที่ 3.8 แสดงการใช้ฟังก์ชันโดยกำหนดให้ *X* เป็นตัวแปรชนิดบิตแทนการกระทำในสมการบูลีน

```

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 3.7 แสดงการใช้ไพรเซจอร์

```

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 3.8 แสดงการใช้ฟังก์ชัน

3.1.2.4 โอเปอเรเตอร์ (VHDL Operators)

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกับกับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 3.9

PREDEFINED OPERATORS	
LOGICAL OPERATORS :	NOT AND OR NAND NOR XOR
OPERAND TYPE :	BIT BOOLEAN
RESULT TYPE :	BIT BOOLEAN
RELATIONAL OPERATORS :	= /= < <= > >=
OPERAND TYPE :	any type
RESULT TYPE :	Boolean
ARITHMETIC OPERATORS :	+ - * / ** MOD REM ABS
OPERAND TYPE :	INTEGER REAL Physical
RESULT TYPE :	INTEGER REAL Physical
CONCATENATION OPERATOR :	&
OPERAND TYPE :	array of any type
RESULT TYPE :	array of any type

รูปที่ 3.9 แสดงตัวกระทำใน VHDL

3.1.2.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

ในวงจรอิเล็กทรอนิกส์ อุปกรณ์ทุกตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (always active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องเสมอในทุกๆ เหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของสถาปัตยกรรมบรรยาย (architecture) จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่ไพรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายไพรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆ ไพรเซสก็จะทำงานไปพร้อมๆ กันด้วย

3.1.2.6 สัญญาณและตัวแปร (Signals and Variables)

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ $\leq$ ในการส่งค่าและสามารถให้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น $w \leq a$ AFTER 12 NS หมายถึงกำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 NS.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในทางตรงข้าม ตัวแปรที่มีลักษณะเป็นเสมือนตัวกลางซอฟต์แวร์ที่ใช้ในการส่งผ่านข้อมูล และไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ := ตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชัน โปรซีเจอร์ และโมดูล

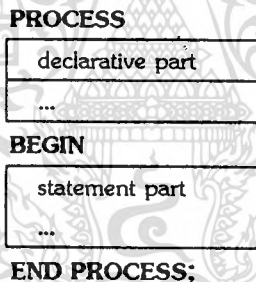
3.2 การบรรยายเชิงพฤติกรรม (Behavioral Description of Hardware)

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรมเป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นสืบเนื่องมาจากการเปลี่ยนแปลงสถานะของข้อมูลที่เข้ามาโดยไม่คำนึงถึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในจะเป็นอย่างไร ในหัวข้อนี้จะแสดงให้เห็นถึงประโยชน์ในการใช้โปรแกรมย่อยที่จำลองรูปแบบอินพุตและเอาต์พุตในระดับพฤติกรรมแทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่าง ๆ ที่ควรรู้

3.2.1 โพรเซส (Process Statement)

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญญาณ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติตามคำสั่งพร้อม ๆ กับโพรเซสอื่น ๆ ที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ทางด้านขาเข้าของสัญญาณการกำหนดค่าให้กับสัญญาณ($\leq$)

การบรรยายโพรเซสจะเริ่มต้นด้วยคำสั่ง **PROCESS** และจบด้วยคำสั่ง **END PROCESS** รูปที่ 3.10 แสดงส่วนประกอบของการบรรยายแบบโพรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้และส่วนของการปฏิบัติตามคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 3.10 แสดงรูปแบบของการบรรยายแบบโพรเซส

3.2.1.1 การกำหนดตัวกระทำภายในโพรเซส (Declarative Part of a Process)

ตัวกระทำภายในโพรเซสมี 3 ชนิดคือ ตัวแปร(variable) ไฟล์(file) และตัวคงที่(constant) ซึ่งตัวกระทำทั้งสามชนิดนี้หากมีการประกาศไว้ในโพรเซสใดก็จะใช้ได้เฉพาะในโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซสสามารถทำได้โดยใช้สัญญาณ(signal)หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ **ARCHITECTURE**

```

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

```

รูปที่ 3.11 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส

รูปที่ 3.11 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง **PROCESS** และ **BEGIN** คำเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียวเท่านั้น แต่คำเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้โปรแกรมย่อยนั้น ๆ

3.2.1.2 การกำหนดการกระทำภายในโพรเซส (Statement Part of a process)

การกระทำใด ๆ ภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับ(sequential)เสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการกระทำซ้ำได้เช่น IF-THEN-ELSE, CASE-WHEN, FOR LOOP และ WHILE LOOP ดังตัวอย่างในรูปที่ 3.12 และ 3.13

```

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

```

รูปที่ 3.12 แสดงเงื่อนไขการกระทำในโพรเซส

```

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

```

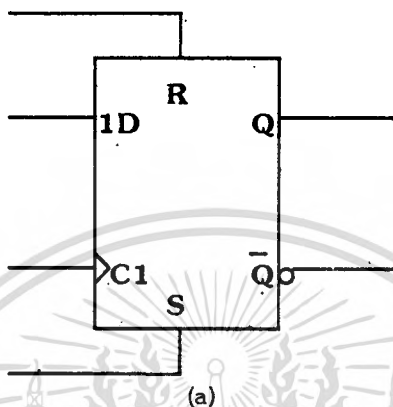
รูปที่ 3.13 แสดงการกระทำในโพรเซส

3.2.1.3 การกระตุ้นและยับยั้งการกระทำของโพรเซส (Sensitivity List)

การกระทำภายในโพรเซสจะเตรียมพร้อมและมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์เกิดขึ้น อย่างไรก็ตามเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโพรเซสได้ เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โดยการกำหนดรายการของสัญญาณที่ต้องการให้โฟรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้น เหตุการณ์ใดๆที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโฟรเซส รายการของสัญญาณนี้เรียกว่า Sensitivity List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS

รูปที่ 3.14 แสดง (a) ตัวอย่างโมเดลและ (b) ตัวอย่างการบรรยายการเชื่อมต่อของ D-FlipFlop รูปที่ 3.15 แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop (a) การใช้ตัวกระทำภายนอกโฟรเซส และ (b) การใช้ตัวกระทำภายในโฟรเซส โดยมีรายการของสัญญาณ (*rst*, *set*, *clk*) เป็นตัวกระตุ้นการปฏิบัติงานภายในโฟรเซส



(a)

```

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

```

(b)

รูปที่ 3.14 (a) ตัวอย่างโมเดล D-FlipFlop

(b) การบรรยายการเชื่อมต่อของ D-FLIPFLOP

```

ARCHITECTURE behavioral OF d_sr_flipflop IS
    SIGNAL state : BIT := '0';
BEGIN
    dff : PROCESS (rst, set, clk)
    BEGIN
        IF set = '1' THEN
            state <= '1' AFTER sq_delay;
        ELSIF rst = '1' THEN
            state <= '0' AFTER rq_delay;
        ELSIF clk = '1' AND clk'EVENT THEN
            state <= d AFTER cq_delay;
        END IF;
    END PROCESS dff;

    q <= state;
    qb <= NOT state;
END behavioral;

```

```

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

```

(b)

รูปที่ 3.15 แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop

(a) การใช้ตัวกระทำภายนอกโพรเซส

(b) การใช้ตัวกระทำภายในโพรเซส

3.2.2 การตรวจสอบการกระทำ (Assertion Statement)

VHDL ได้จัดเตรียมรูปแบบคำสั่งเพื่อใช้ในการตรวจดูการกระทำต่างๆที่เกิดขึ้นภายในอุปกรณ์ไว้ดังนี้

```

ASSERT assertion_condition REPORT "reporting_message" SEVERITY severity_level;

```

คำสั่งนี้จะถูกปฏิบัติเมื่อเงื่อนไข *assertion_condition* มีค่าเป็น FALSE และจะแสดงข้อความ "*reporting_message*" ส่วนพารามิเตอร์ *severity_level* จะเป็นตัวกำหนดระดับความรุนแรงที่เกิดขึ้น ซึ่งแบ่งเป็น 4 ระดับคือ NOTE, WARNING, ERROR และ FAILURE. ระดับความรุนแรง ERROR หรือ FAILURE จะทำให้การเลียนแบบการทำงานหยุดทันทีหลังจากที่ได้แสดงข้อความ *reporting_message*

```

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

```

รูปที่ 3.16 แสดงการใช้ ASSERT

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แล้ว ส่วนระดับ NOTE หรือ WARNING จะแสดงข้อความ *reporting_message* แต่ยังคงเลียนแบบการทำงานต่อไป สมมติว่าเราต้องการตรวจสอบการทำงานของ D-FlipFlop ในตัวอย่างข้างต้นว่าในขณะที่กำลังทำงานได้เกิดสัญญาณในกรณีที่ไม่มีปรากฏขึ้นหรือไม่เช่น สัญญาณ set และ rst มีค่าเป็น '1' พร้อมกันก็สามารถตรวจสอบได้โดยแทรกคำสั่ง **ASSERT** เพื่อตรวจสอบสัญญาณในโปรแกรมได้ดังรูปที่ 3.16

3.2.3 การหยุดรอ (Sequential Wait Statements)

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณมีอยู่ 4 รูปแบบดังนี้

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

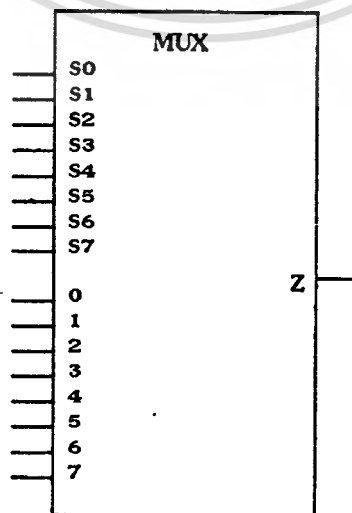
คำสั่งหยุดรอสามารถใช้ได้ภายในโปรแกรมและโปรแกรมเมอร์ที่ไม่ถูกกำหนด Sensitivity List ไว้เท่านั้น **WAIT FOR** จะหยุดรอเป็นเวลาเท่ากับ *waiting_time* **WAIT ON** จะหยุดรอจนกว่าจะมีเหตุการณ์เกิดขึ้นกับสัญญาณ *waiting_sensitivity_list* **WAIT UNTIL** จะหยุดรอจนกว่าเงื่อนไข *waiting_condition* เปลี่ยนจาก FALSE เป็น TRUE **WAIT** จะหยุดรอตลอดไป

3.3 การบรรยายเชิงกระแสข้อมูล (Dataflow Description of Hardware)

การบรรยายเชิงกระแสข้อมูลเป็นการบรรยายถึงการไหลของข้อมูลผ่านรีจิสเตอร์และบัลลของระบบ เป็นระดับขั้นของการบรรยายที่อยู่ตรงกลางระหว่างการบรรยายเชิงพฤติกรรมและการบรรยายเชิงโครงสร้าง เครื่องมือที่ใช้ในการควบคุมการเคลื่อนไหวของข้อมูลได้แก่ *conditional*, *selected* และ *guarded* ในหัวข้อนี้จะแสดงให้เห็นถึงลักษณะและรูปแบบของการบรรยายเชิงกระแสข้อมูลรวมถึงข้อกำหนดและเงื่อนไขต่างพร้อมทั้งตัวอย่างประกอบ

3.3.1 การกำหนดเลือกข้อมูล (Multiplexing and Data Selection)

ในระบบดิจิทัล โครงสร้างของอุปกรณ์ฮาร์ดแวร์ส่วนใหญ่จะถูกใช้สำหรับการคัดเลือกและการนำข้อมูลเข้าสู่บัลลและรีจิสเตอร์ รูปที่ 3.17 แสดงตัวอย่างโมเดลและการบรรยายเชิงกระแสข้อมูลของ 8-to-1 มัลติเพล็กซ์เซอร์



(a)

```

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

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

```

(b)

รูปที่ 3.17 แสดง 8-to-1 มัลติเพล็กซ์

(a) โมเดล

(b) การบรรยายเชิงกระแสข้อมูล

sel_lines เป็นสัญญาณที่กำหนดขึ้นมาเพื่อรวมสัญญาณเลือกทั้ง 8 อินพุต (s7 - s0) เข้าด้วยกันและนำไปใช้เป็นสัญญาณในการกำหนดเลือกอินพุตตัวใดตัวหนึ่งให้กับเอาต์พุต z ถ้าไม่มีอินพุตใดมีค่าเป็น '1' เลย ค่า '0' จะถูกส่งให้กับเอาต์พุต z หลังจากเวลาผ่านไป 3 NS ส่วนอินพุตอื่น ๆ (i7, i6, i5, i4, i3, i2, i1, i0) จะถูกส่งให้ z ขึ้นอยู่กับสัญญาณเลือกใด (s7, s6, s5, s4, s3, s2, s1, s0) มีค่าเป็น '1'

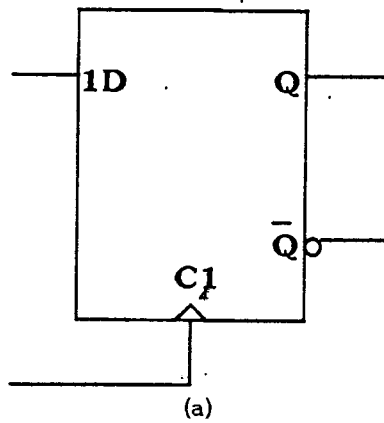
3.3.2 การกำหนดการ์ด (Guarded Signal Assignments)

ในการบรรยายเชิงกระแสข้อมูล เราสามารถตั้งเงื่อนไขสำหรับการกำหนดค่าสัญญาณใด ๆ ได้โดยใช้คำสั่ง **GUARDED** ซึ่งมีรูปแบบในการเขียนดังนี้

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

รูปที่ 3.18 แสดงตัวอย่างการใช้ **GUARDED** ในการบรรยายการทำงานของ d_flipflop การกำหนด **GUARDED** ทำได้โดยใช้รูปแบบของคำสั่ง **BLOCK** ตามด้วยเงื่อนไขภายในวงเล็บ ซึ่งผลลัพธ์ที่ได้จากวงเล็บนี้ก็คือ **GUARDED** นั่นเอง ในรูปเป็นการบรรยายการทำงานของ d_flipflop ที่มีการทำงานเมื่อสัญญาณนาฬิกามีการเปลี่ยนแปลงจาก '0' เป็น '1' หรือสัญญาณขาขึ้น จะเห็นว่า **GUARDED** ถูกใช้เป็นเงื่อนไขในการกำหนดค่าให้กับ q และ qb นั่นคือถ้าเงื่อนไขภายในวงเล็บมีค่าเป็น TRUE (**GUARDED** เป็น TRUE) ค่า d และ NOT d จะถูกส่งค่าให้กับ q และ qb หากเงื่อนไขภายในวงเล็บมีค่าเป็น FALSE (**GUARDED** เป็น FALSE) ค่า d และ NOT d ก็จะไม่ถูกส่งค่าให้กับ q และ qb

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



```

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

ARCHITECTURE guarding OF d_flipflop IS
BEGIN
    ff : BLOCK ( c = '1' AND NOT c'STABLE )
    BEGIN
        q <= GUARDED d AFTER delay1;
        qb <= GUARDED NOT d AFTER delay2;
    END BLOCK ff;
END guarding;

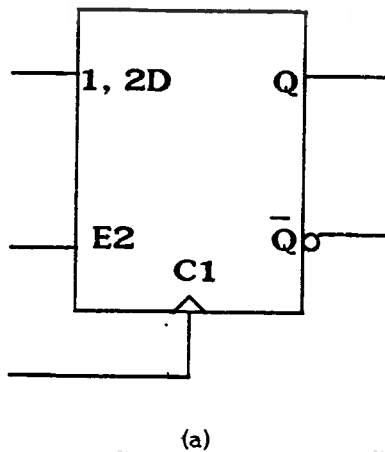
```

(b)

รูปที่ 3.18 แสดงตัวอย่างการใช้ GUARDED

3.3.3 การกำหนดโครงข่ายของการ์ด (Nesting Guarded Blocks)

เราสามารถกำหนดการ์ดเป็นโครงข่ายซ้อนๆกันได้โดยการกำหนด BLOCK ซ้อนๆกัน ซึ่งจะทำให้เราสามารถออกแบบอุปกรณ์ที่มีการทำงานที่ซับซ้อนมากกว่าได้ พิจารณา *d_flipflop* ในรูปที่ 3.19 แสดงตัวอย่างการบรรยายการทำงานที่ซับซ้อนมากกว่า *d_flipflop* ในตัวอย่างข้างต้น ซึ่งนอกจากสัญญาณอินพุต *d* จะถูกส่งผ่านไปยัง *q* โดยมีเงื่อนไขว่า ($c = '1' \text{ AND NOT } c'STABLE$) ต้องเป็น TRUE แล้วในกรณีนี้ยังมีเงื่อนไขเพิ่มเติมอีกว่าสัญญาณ *enable* (*e*) ต้องเป็น TRUE ด้วย



```

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

ARCHITECTURE guarding OF de_flipflop IS
BEGIN
    edge : BLOCK ( c = '1' AND NOT c'STABLE )
        BEGIN
            gate : BLOCK ( e = '1' AND GUARD )
                BEGIN
                    q <= GUARDED d AFTER delay1;
                    qb <= GUARDED NOT d AFTER delay2;
                END BLOCK gate;
            END BLOCK edge;
        END guarding;

```

(b)

รูปที่ 3.19 แสดงตัวอย่างการใช้ NESTING GUARDED

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

3.3.4 การเลือกสรรข้อมูล (Resolving Between Several Driving Values)

ในทางฮาร์ดแวร์มีหลาย กรณีที่เรามีความจำเป็นต้องเชื่อมต่อหลาย ๆ เอาต์พุตไปยังอินพุตได้อินพุตหนึ่ง ซึ่งในกรณีเช่นนี้อาจจะทำให้เกิดความสับสนของสัญญาณที่ปะปนกันทำให้ไม่สามารถรู้ค่าที่แน่นอนได้ ในทาง VHDL หมายถึงการกำหนดค่าจากหลาย ๆ สัญญาณให้กับสัญญาณเดียวซึ่งก็จะให้ผลลัพธ์ที่สับสนได้เหมือนกัน ดังตัวอย่างในรูปที่ 3.20 ซึ่งจะไม่สามารถบอกได้เลยว่าค่าของ *circuit_node* มีค่าเป็นอะไรถ้าสมมติว่าค่าอินพุต a เป็น '1' และค่าอินพุตอื่น ๆ เป็น '0'

```

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

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

```

รูปที่ 3.20 แสดงการกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน

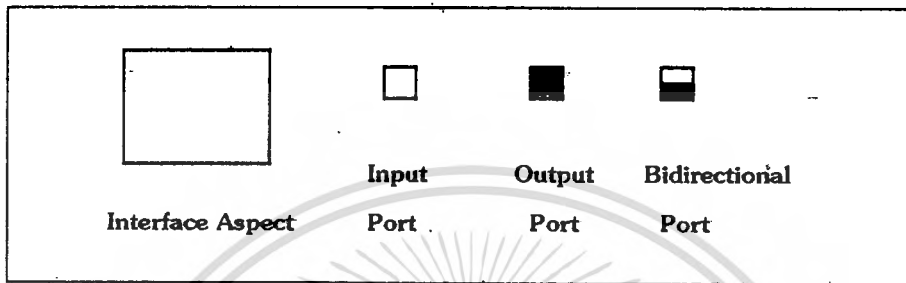
การบรรยายการทำงานของระบบในเชิงโครงสร้างจะต้องแสดงรายการของอุปกรณ์ทั้งหมดที่ใช้ในระบบนั้นและต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ ด้วย เพราะว่าการบรรยายในระดับนี้เป็นการบรรยายที่ใกล้เคียงลักษณะของฮาร์ดแวร์จริงที่สุด VHDL ได้จัดเตรียมเครื่องมือและลักษณะโครงสร้างของการบรรยายในระดับนี้ไว้ที่สำคัญ 4 ลักษณะคือ 1) ความสามารถในการเลือกหรือกำหนดอุปกรณ์ที่ต้องการได้จากไลบรารี 2) การสร้างไลบรารีเพื่อเก็บอุปกรณ์ที่ผู้ใช้ออกแบบไว้เองได้ 3) กลไกในการเชื่อมต่อระหว่างอุปกรณ์ 4) โครงสร้างการกำหนดอุปกรณ์ชนิดเดียวกันซ้ำ ๆ กัน ในหัว

ข้อนี้จะได้แสดงรูปแบบและการบรรยายเชิงโครงสร้างโดยใช้ตัวอย่างการออกแบบวงจรเปรียบเทียบกับติดตั้งแต่ระดับเทกองถึงระดับวงจร

3.4.1 การกำหนดไลบรารี (Parts Library)

ในหัวข้อนี้จะแสดงการออกแบบเกทพื้นฐาน 3 ชนิดคือ **INVERTER**, **NAND** เกทชนิด 2 อินพุตและ 3 อินพุต ซึ่งเกทพื้นฐานเหล่านี้จะถูกเก็บไว้ในไลบรารีและนำมาใช้เป็นส่วนประกอบเพื่อออกแบบเป็นวงจรที่ใหญ่ขึ้นต่อไป

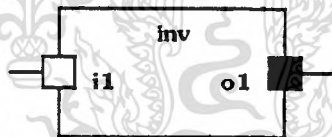
ในการออกแบบได้กำหนดสัญลักษณ์ที่เกี่ยวข้องไว้ดังนี้



รูปที่ 3.23 แสดงสัญลักษณ์แทนตัวอุปกรณ์

3.4.1.1 อินเวอร์เตอร์โมเดล (Inverter Model)

รูปที่ 3.24 (a) แสดงสัญลักษณ์ของ **Inverter** รูป (b) แสดงการบรรยายการเชื่อมต่อโดยกำหนดให้อุปกรณ์มีชื่อว่า **inv** และมีพอร์ตการติดต่อ **i1** เป็นอินพุตและ **o1** เป็นเอาต์พุตมีชนิดของข้อมูลที่ผ่านเข้าออกเป็น **BIT** เราสามารถกำหนดพอร์ตเป็นแบบสองทิศทาง(bidirectional)ได้โดยการกำหนดโหมดเป็น **INOUT** รูป (c) แสดงการบรรยายการทำงานของอุปกรณ์



(a)

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

(b)

```
ARCHITECTURE single_delay OF inv IS
BEGIN
    o1 <= NOT i1 AFTER 4 NS;
END single_delay;
```

(c)

รูปที่ 3.24 แสดง Inverter Model

(a) แสดงสัญลักษณ์ของ **Inverter**

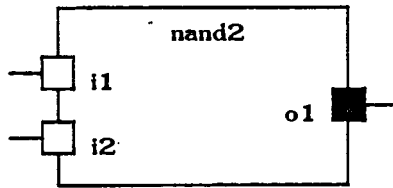
(b) แสดงการบรรยายการเชื่อมต่อ

(c) แสดงการบรรยายการทำงานของ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4.1.2 NAND เกทโมเดล (NAND Gate Models)

รูปที่ 3.25 แสดงสัญลักษณ์และการบรรยายของ NAND เกทชนิด 2 อินพุต



(a)

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

(b)

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

(c)

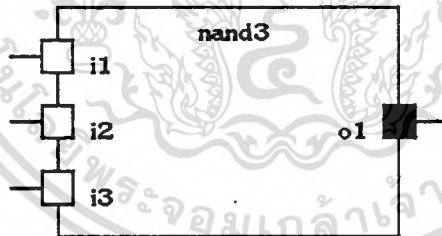
รูปที่ 3.25 แสดง NAND2 Model

(a) แสดงสัญลักษณ์ของ NAND2

(b) แสดงการบรรยายการเชื่อมต่อ

(c) แสดงการบรรยายการทำงาน

รูปที่ 3.26 แสดงสัญลักษณ์และการบรรยายของ NAND เกทชนิด 3 อินพุต



(a)

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

(b)

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

(c)

รูปที่ 3.26 แสดง NAND3 Model

(a) แสดงสัญลักษณ์ของ NAND3

(b) แสดงการบรรยายการเชื่อมต่อ

(c) แสดงการบรรยายการทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการ (c) อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

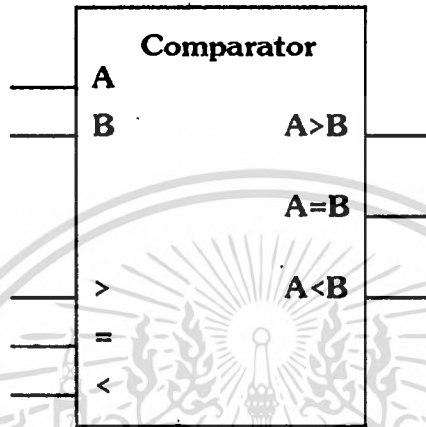
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4.2 การเชื่อมต่ออุปกรณ์พื้นฐาน (Wiring of Primitive)

เมื่อเราได้ออกแบบแบบเกทพื้นฐานเรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานเหล่านี้มาเชื่อมต่อกันเป็นวงจร ในหัวข้อนี้จะแสดงการออกแบบและการบรรยายเชิงโครงสร้างของวงจรเปรียบเทียบโดยใช้ Inverter และ NAND เกท

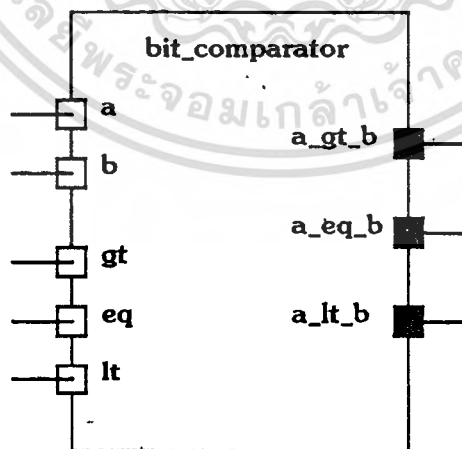
3.4.2.1 ตัวอย่างการออกแบบวงจรเปรียบเทียบ (Logic Design of Comparator)

วงจรเปรียบเทียบทีละบิต(bit-comparator) ประกอบด้วยสัญญาณข้อมูล 2 อินพุต, สัญญาณควบคุม 3 อินพุต และสัญญาณผลเปรียบเทียบ 3 เอาต์พุต ดังรูปที่ 3.27



รูปที่ 3.27 แสดงวงจรเปรียบเทียบทีละบิต

เอาต์พุต $A > B$ มีค่าเป็น '1' เมื่ออินพุต A มีค่ามากกว่า B ($AB = 10$) หรือถ้า A เท่ากับ B และอินพุต $>$ มีค่าเป็น '1' เอาต์พุต $A = B$ มีค่าเป็น '1' เมื่ออินพุต A เท่ากับ B และอินพุต $=$ มีค่าเป็น '1' ส่วนเอาต์พุต $A < B$ จะตรงข้ามกับเอาต์พุต $A > B$ นั่นคือมีค่าเป็น '1' เมื่ออินพุต A มีค่าน้อยกว่า B ($AB = 01$) หรือถ้า A เท่ากับ B และอินพุต $<$ มีค่าเป็น '1' ซึ่งสามารถเขียนเป็นสมการบูลีนในรูปของ NAND เกทได้ดังรูปที่ 3.28



$$\begin{aligned} a_gt_b &= ((a \cdot gt)' \cdot (b' \cdot gt)')' \cdot (a \cdot b')' \\ a_eq_b &= ((a \cdot b \cdot eq)' \cdot (a' \cdot b' \cdot eq)')' \\ a_lt_b &= ((a' \cdot lt)' \cdot (b \cdot lt)')' \cdot (a' \cdot b)' \end{aligned}$$

รูปที่ 3.28 แสดงสัญลักษณ์และสมการบูลีนของวงจรเปรียบเทียบทีละบิต

3.4.2.2 ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบทีละบิต

(VHDL Description of Bit-Comparator)

รูปที่ 3.29 แสดงการบรรยายการเชื่อมต่อของวงจรเปรียบเทียบทีละบิตตามสัญลักษณ์ในรูปที่ 3.28 เครื่องหมาย “--” ใช้แทนข้อความอธิบาย(comment)

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

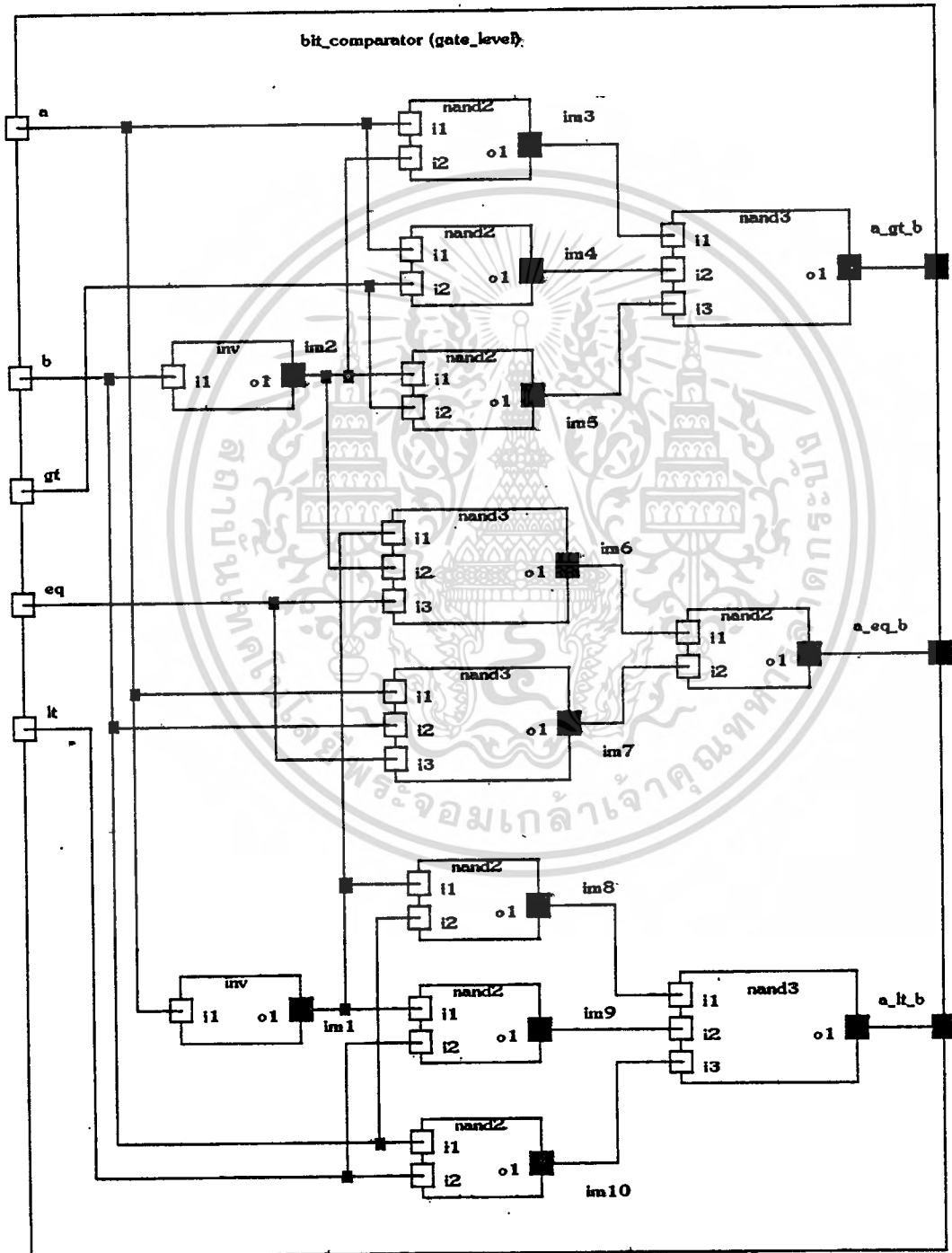
รูปที่ 3.29 แสดงการบรรยายการเชื่อมต่อ

```
ARCHITECTURE gate_level OF bit_comparator IS
    COMPONENT n1 PORT (i1 : IN BIT; o1 : OUT BIT); END COMPONENT;
    COMPONENT n2 PORT (i1, i2 : IN BIT; o1 : OUT BIT); END COMPONENT;
    COMPONENT n3 PORT (i1, i2, i3 : IN BIT; o1 : OUT BIT); END COMPONENT;

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

รูปที่ 3.30 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต

รูปที่ 3.30 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบโดยกำหนดชื่อการบรรยายเป็น **gate_level** ภายในมีการกำหนดรายการเกทพื้นฐานที่ต้องใช้เอาไว้ 3 ชนิดและกำหนดชื่อเป็น **n1, n2, n3** โดยกำหนดให้ **n1** อ้างอิงถึง inverter เกท **n2** อ้างอิงถึง nand2 เกท และ **n3** อ้างอิงถึง nand3 เกท คำสั่ง **FOR ALL : n1** เป็นการกำหนดให้อุปกรณ์ทุกตัวที่ขึ้นต้นชื่อด้วย **n1** ให้อ้างอิง inverter เกท ในกรณีของ **n2** และ **n3** ก็เช่นเดียวกัน **WORK** เป็นการกำหนดตำแหน่งที่อยู่หรือไลบรารีที่เก็บอุปกรณ์ที่อ้างอิงถึง ซึ่งอาจจะเป็นชื่อใดเรีกทอริใดๆ (กรณีที่ใช้คำ **WORK** จะหมายถึงไดเร็กทอรีปัจจุบัน) รูปที่ 3.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบที่ละบิตตามสมการบูลีนโดยใช้เกทพื้นฐานที่ได้ออกแบบไว้แล้ว



รูปที่ 3.31 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบที่ละบิต

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

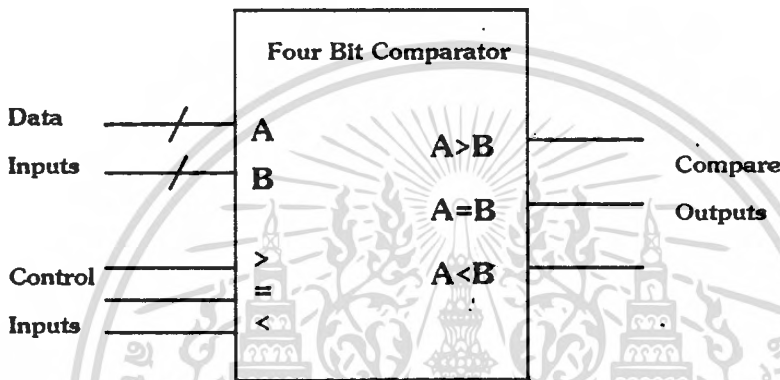
3.4.3 การเชื่อมต่ออุปกรณ์ชนิดเดียวกัน (Wiring Iterative Networks)

ในการเชื่อมต่ออุปกรณ์ต่าง ๆ เข้าด้วยกันจะต้องอ้างอิงถึงอุปกรณ์ทีละตัว แต่หากเป็นการเชื่อมต่ออุปกรณ์ชนิดเดียวกันจำนวนหลาย ๆ ตัวเข้าด้วยกันแล้ว VHDL ได้จัดเตรียมเครื่องมือซึ่งเปรียบเสมือนเส้นทางลัดในการบรรยายโดยไม่จำเป็นต้องอ้างอิงถึงอุปกรณ์ทุก ๆ ตัว ในหัวข้อต่อไปจะได้ยกตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิตที่เรียกว่า *nibble_comparator* ซึ่งประกอบด้วยวงจรเปรียบเทียบทีละบิตที่ได้ออกแบบไว้ในหัวข้อก่อนหน้านี้จำนวน 4 ตัวต่อเข้าด้วยกัน

3.4.3.1 ตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิต

(Design of a 4-Bit Comparator)

วงจรเปรียบเทียบขนาด 4 บิตมีสัญญาณข้อมูลขนาด 4 บิตจำนวน 2 อินพุต, สัญญาณควบคุม 3 อินพุต และสัญญาณผลเปรียบเทียบ 3 เอาต์พุต ดังรูปที่ 3.32

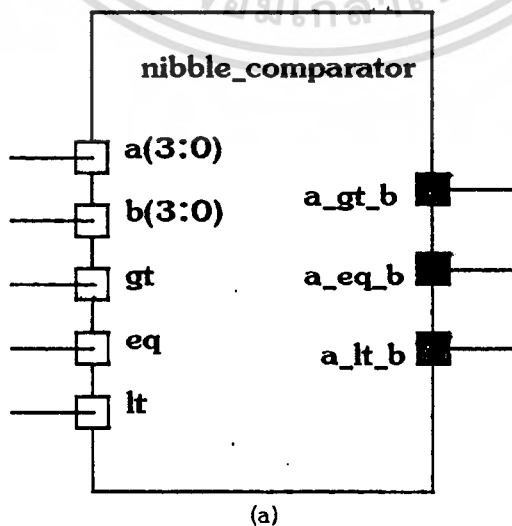


รูปที่ 3.32 แสดงวงจรเปรียบเทียบขนาด 4 บิต

3.4.3.2 ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบขนาด 4 บิต

(VHDL Description of a 4-Bit Comparator)

รูปที่ 3.33 แสดงสัญลักษณ์ของวงจร *nibble_comparator* และการบรรยายการเชื่อมต่อของวงจร อินพุต a และ b เป็นอินพุตสำหรับข้อมูลขนาด 4 บิตเราสามารถกำหนดชนิดของอินพุตเป็น **BIT_VECTOR** ซึ่งเป็นอาเรย์ของ **BIT** ได้ด้วย



```

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

```

(b)

รูปที่ 3.33 แสดง nibble_comparator

(a) สัญลักษณ์ของวงจร

(b) การบรรยายการเชื่อมต่อ

รูปที่ 3.34 แสดงการบรรยายการทำงานของวงจร โดยใช้ comp1 ซึ่งอ้างอิงถึงวงจรเปรียบเทียบที่ละบิต ค่า n เป็นค่าคงที่ที่ใช้กำหนดจำนวนของวงจร comp1 ซึ่งในกรณีนี้เป็นวงจรเปรียบเทียบขนาด 4 บิต ดังนั้นจึงกำหนดค่า n เป็น 4 การบรรยายได้ใช้ FOR loop และ GENERATE ในการกำหนดการเชื่อมต่อของอุปกรณ์ชนิดเดียวกัน สังเกตว่าการบรรยายในลักษณะนี้ช่วยประหยัดเวลาและสะดวกมากขึ้น ทั้งนี้เราสามารถกำหนดจำนวนอุปกรณ์ที่มาเชื่อมต่อกันจำนวนเท่าใดก็ได้โดยไม่ต้องเปลี่ยนรูปแบบใดๆเลยนอกจากค่าของ n เท่านั้น รูปที่ 3.35 แสดงลักษณะของวงจรแบบสัญลักษณ์

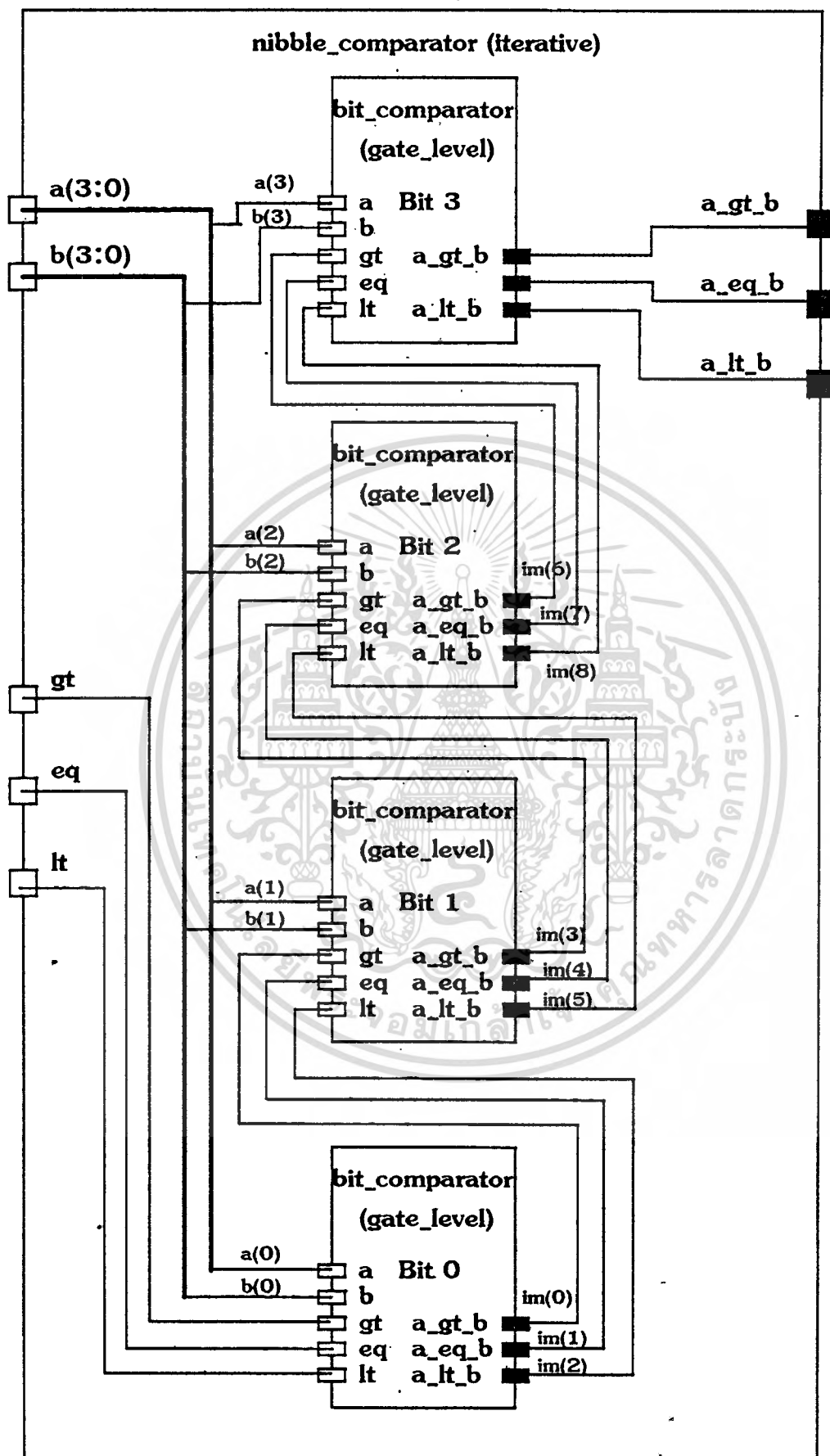
```

ARCHITECTURE iterative OF nibble_comparator IS
  COMPONENT comp1
    PORT ( a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
  CONSTANT n : INTEGER := 4;
  SIGNAL im : BIT_VECTOR (0 TO (n-1)*3-1);
BEGIN
  c_all : FOR i IN 0 TO n-1 GENERATE
    l : IF i = 0 GENERATE
      least : comp1 PORT MAP (a(i), b(i), gt, eq, lt, im(0), im(1), im(2));
    END GENERATE;
    m : IF i = n-1 GENERATE
      most : comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2), im(i*3-1),
        a_gt_b, a_eq_b, a_lt_b);
    END GENERATE;
    r : IF i > 0 AND i < n-1 GENERATE
      rest : comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2), im(i*3-1),
        im(i*3+0), im(i*3+1), im(i*3+2));
    END GENERATE;
  END GENERATE;
END iterative;

```

รูปที่ 3.34 แสดงการบรรยายการทำงานของวงจร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ 3.35 แสดงลักษณะของวงจรแบบสัญญาณนำเข้าไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

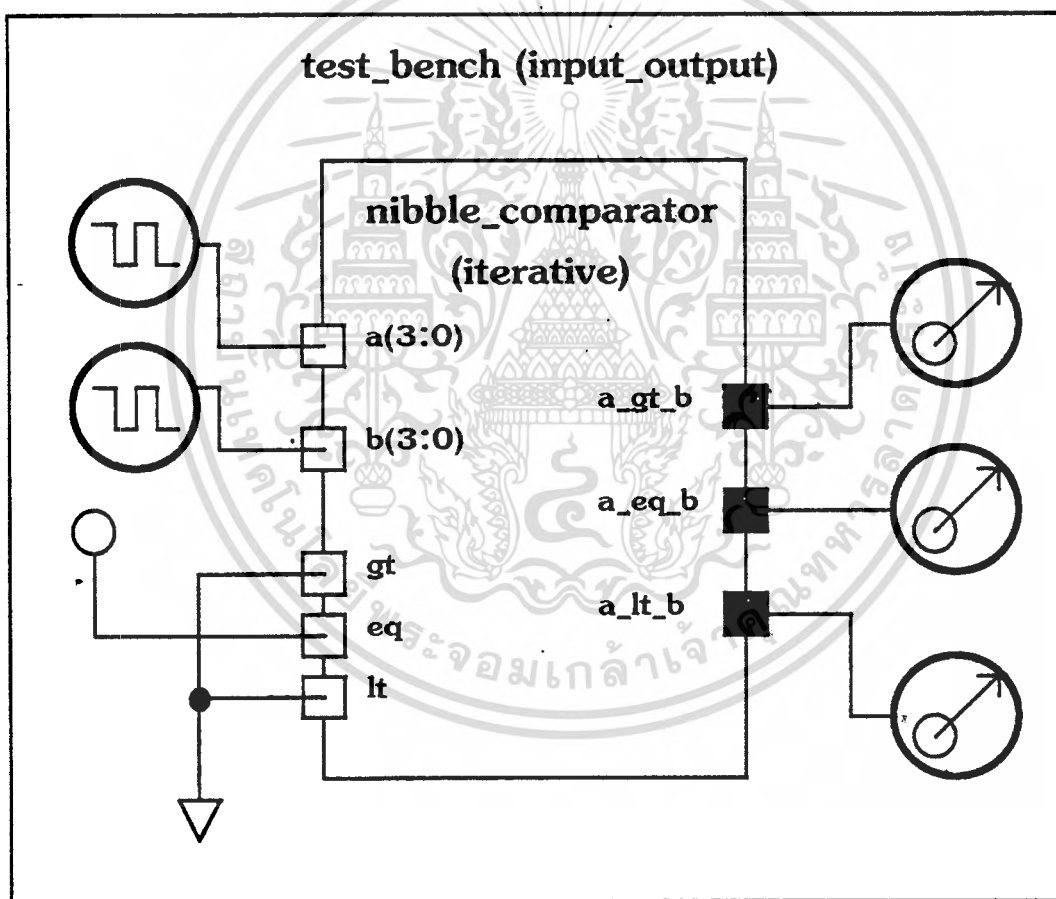
3.5 การทดสอบและเลียนแบบการทำงานของวงจร

(A Test Bench and Simulation)

นอกจากจะใช้ในการออกแบบวงจรแล้ว VHDL ยังสามารถนำมาใช้ในการสร้างแบบทดสอบเพื่อทดสอบและเลียนแบบการทำงานของอุปกรณ์นั้น ๆ ได้ด้วย หัวข้อต่อไปนี้จะแสดงให้เห็นตัวอย่างแบบทดสอบเพื่อทำการทดสอบและเลียนแบบการทำงานของวงจรเปรียบเทียบขนาด 4 บิต การเตรียมข้อมูลสำหรับส่งให้อินพุตของวงจร การแสดงผลการเลียนแบบและการทำงานร่วมกับเพิ่มข้อความเพื่อใช้เก็บรายงานผลการเลียนแบบ

3.5.1 รูปแบบการทดสอบ (Modeling a Test Bench)

แบบทดสอบจะต้องประกอบด้วยกำหนดวงจรที่จะใช้ในการทดสอบและต้องจัดเตรียมข้อมูลที่จะส่งให้กับอินพุตของวงจร รูปที่ 3.26 แสดงรูปของแบบทดสอบวงจรเปรียบเทียบขนาด 4 บิต แบบทดสอบไม่มีการเชื่อมต่อกับพอร์ตกายนอก ภายในได้จัดเตรียมรูปแบบของข้อมูลสำหรับอินพุต a และ b และเชื่อมต่อกับอินพุตควบคุม gt, eq และ lt เข้ากับ gnd, vdd และ gnd ตามลำดับ



รูปที่ 3.26 แสดงรูปของแบบทดสอบวงจรเปรียบเทียบขนาด 4 บิต

รูปที่ 3.27 แสดงการบรรยายแบบทดสอบ สังเกตว่าในส่วนของการบรรยายการเชื่อมต่อไม่ได้กำหนดพอร์ตเอาต์เนื่องจากในแบบทดสอบไม่มีการติดต่อกับอุปกรณ์ภายนอกดังนั้นในส่วนนี้จึงเว้นว่างไว้ ในส่วนของการบรรยายการเลียนแบบกำหนดให้ `comp4` เป็นชื่ออุปกรณ์อ้างอิงถึงวงจร `nibble_comparator` ในส่วนของการเตรียมข้อมูลกำหนดให้ส่งข้อมูลเข้าสู่อินพุต a และ b ทุก ๆ ช่วงเวลา 500 NS.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

ENTITY nibble_comparator_test_bench IS
END nibble_comparator_test_bench;

ARCHITECTURE input_output OF nibble_comparator_test_bench IS
    COMPONENT comp4 PORT (a, b : IN BIT_VECTOR (3 DOWNT0 0);
                           gt, eq, lt : IN BIT;
                           a_gt_b, a_eq_b, a_lt_b : OUT BIT);

    END COMPONENT;
    FOR a1 : comp4 USE ENTITY WORK.nibble_comparator (iterative);
    SIGNAL a, b : BIT_VECTOR (3 DOWNT0 0);
    SIGNAL eql, lss, gtr : BIT;
    SIGNAL vdd : BIT := '1';
    SIGNAL gnd : BIT := '0';
BEGIN
    a1 : comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
    a2 : a <= "0000", -- a = b (steady state)
        "1111" AFTER 0500 NS, -- a > b (worst case)
        "1110" AFTER 1500 NS, -- a < b (worst case)
        "1110" AFTER 2500 NS, -- a > b (need bit 1 info)
        "1010" AFTER 3500 NS, -- a < b (need bit 2 info)
        "0000" AFTER 4000 NS, -- a < b (steady state, prepare for next)
        "1111" AFTER 4500 NS, -- a = b (worst case)
        "0000" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
        "0000" AFTER 5500 NS, -- a = b (worst case)
        "1111" AFTER 6000 NS; -- a > b (need bit 3 only, best case)

    a3 : b <= "0000", -- a = b (steady state)
        "1110" AFTER 0500 NS, -- a > b (worst case)
        "1111" AFTER 1500 NS, -- a < b (worst case)
        "1100" AFTER 2500 NS, -- a > b (need bit 1 info)
        "1100" AFTER 3500 NS, -- a < b (need bit 2 info)
        "1111" AFTER 4000 NS, -- a < b (steady state, prepare for next)
        "1111" AFTER 4500 NS, -- a = b (worst case)
        "1111" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
        "0000" AFTER 5500 NS, -- a = b (worst case)
        "0000" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
END input_output;

```

รูปที่ 3.27 แสดงการบรรยายแบบทดสอบ

3.5.2 รูปแบบการทำงานร่วมกับแฟ้มข้อความ (Formatted ASCII I/O Operations)

VHDL ได้จัดเตรียมรูปแบบของการติดต่อกับแฟ้มข้อความไว้ใน **STD.TEXTIO** ไบรารี รวมทั้งโปรซีเจอร์ในการอ่านและเขียน (ดูภาคผนวก A) ต่าง ๆ เอาไว้ โปรซีเจอร์ **READLINE (f, l)** จะอ่านข้อความจากไฟล์ *f* ทีละหนึ่งบรรทัดและนำมาเก็บไว้ในบัฟเฟอร์ *l* โปรซีเจอร์ **READ (l, v, ...)** อ่านค่าจาก *l* ทีละค่า (หากในหนึ่งบรรทัดมีข้อมูลหลายค่าให้แยกข้อมูลด้วย ';' หรือ ' ' (space)) นำมากำหนดค่าให้ *v* และจะเปลี่ยนชนิดของข้อมูลเป็นไปตามชนิดของ *v* ด้วย โปรซีเจอร์ **WRITE (l, v, ...)** เขียนข้อมูลจาก *v* ออกสู่แฟ้มเป็นแฟ้มข้อความที่ส่งมาไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น เมื่อผู้ใช้เห็นประโยชน์ในการค้าไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลงในบัฟเฟอร์ **LINE** และ โปรซีเจอร์ **WRITELINE** (f, l) เขียนข้อมูลจากบัฟเฟอร์ l ลงสู่ไฟล์ f ทีละบรรทัดเช่นกัน ฟังก์ชัน **ENDFILE** (f) ใช้สำหรับตรวจสอบจุดสิ้นสุดของไฟล์โดยจะให้ค่าเป็น TRUE เมื่อพบจุดสิ้นสุดของไฟล์ ชนิดของข้อมูลที่สามารถกำหนดให้กับค่า v ได้มีดังนี้ BIT, BIT_VECTOR, BOOLEAN, CHARACTER, INTEGER, REAL, STRING, และ TIME ถ้าหากเป็นชนิดของข้อมูลที่กำหนดขึ้นเองก็สามารถสร้างโปรซีเจอร์เพื่อเปลี่ยนชนิดของข้อมูลเสียก่อนได้

รูปที่ 3.28 แสดงตัวอย่างการทำงานร่วมกับแฟ้มข้อความ ในการอ้างอิงถึงไลบรารีให้ใช้คำสั่ง **USE** ชนิดของไฟล์กำหนดโดยคำสั่ง **FILE** ถ้าเป็นไฟล์เพื่อการอ่านก็กำหนดเป็น **TEXT IS IN** และกำหนดเป็น **TEXT IS OUT** ถ้าเป็นไฟล์เพื่อการเขียน ชนิดของข้อมูลที่ใช้ในการอ่านและเขียนกำหนดเป็น **LINE** ดังนั้นในการอ่านหรือเขียนจะกระทำทีละบรรทัด รูปที่ 3.29 แสดงรายงานผลที่ได้

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

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

รูปที่ 3.28 แสดงตัวอย่างการทำงานร่วมกับแฟ้มข้อความ

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

รูปที่ 3.29 แสดงรายงานผลที่ได้จากการทดสอบและเลียนแบบ

3.6 สรุป (Summary)

ในบทนี้ได้กล่าวถึงโครงสร้างพื้นฐานและข้อกำหนดต่างๆของภาษา VHDL พอสังเขป เพื่อความเข้าใจในการศึกษาโครงสร้าง การออกแบบและการบรรยายของวงจรในบทต่อไป รูปแบบหลักในการบรรยายการทำงานของระบบซึ่ง VHDL ได้จัดเตรียมไว้แบ่งเป็น 3 ระดับคือ ระดับพฤติกรรม ระดับข้อมูล และระดับโครงสร้าง

การบรรยายในระดับพฤติกรรม(Behavioral Level) ช่วยอำนวยความสะดวกในการออกแบบเป็นขั้นตอนเช่น การออกแบบพอร์ตติดต่อ(I/O port)ที่จำเป็น ใช้อธิบายการทำงานของวงจรในระดับสูง สิ่งที่น่าสนใจเป็นเพียงสัญญาณที่อินพุตและเอาต์พุตเท่านั้น การออกแบบในระดับนี้จะช่วยตรวจสอบและแก้ไขโครงสร้างของวงจรทั้งในเรื่องของพอร์ต จำนวนอุปกรณ์ที่จำเป็นสำหรับการนำมาเชื่อมต่อให้เป็นวงจรที่สมบูรณ์ การประสานงานของแต่ละอุปกรณ์เมื่อรวมกันเป็นระบบ

การบรรยายในระดับกระแสข้อมูล(Dataflow Level) เป็นระดับกลางที่อยู่ระหว่างระดับพฤติกรรมและระดับโครงสร้าง การออกแบบในระดับนี้จะใช้แทนการทำงานของอุปกรณ์ได้ดี ประโยชน์ที่ได้คืออธิบายการไหลของสัญญาณข้อมูลต่างๆระหว่างลอจิกและรีจิสเตอร์โดยผ่านตัวกลางคิบบัส และช่วยในการตรวจสอบและแก้ไขการประสานงานระหว่างลอจิกและรีจิสเตอร์ในลักษณะของการปฏิบัติงานพร้อมกัน(concurrent)ซึ่งเป็นการเลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์จริงๆ

การบรรยายในระดับโครงสร้าง(Structure Level) เป็นระดับล่างสุดเมื่อการออกแบบในระดับบนทั้งสองถูกต้องแล้ว การออกแบบในระดับนี้เป็นการออกแบบในระดับเกทใช้อธิบายการทำงานของประสานกันของเกทต่างๆเพื่อให้ได้สัญญาณออกมาตามต้องการ เมื่อถึงขั้นนี้แล้วการตรวจสอบและแก้ไขจะเป็นเพียงการดูแลความถูกต้องในเชิงตรรกเท่านั้น

บทที่ 4

สถาปัตยกรรมการออกแบบ

(Design Architecture)

ในบทนี้จะได้กล่าวถึงรูปแบบการจัดเรียงและโครงสร้างของระบบทั้งหมดรวมถึงลักษณะทางพฤติกรรมและองค์ประกอบ สถาปัตยกรรมการออกแบบ ข้อกำหนด ยุทิลิตี้ที่จำเป็น การกำหนดไลบรารี การจัดระบบหน่วยความจำรวมทั้งลักษณะทางพฤติกรรมและโครงสร้าง สำหรับลักษณะภายในขององค์ประกอบหรือโมดูลต่างๆและการทดสอบจะได้กล่าวถึงในบทต่อไป

4.1 สถาปัตยกรรมของระบบ (System Interface Description)

ระบบในที่นี้หมายถึงระบบวงจรรวมที่ประกอบด้วยวงจรย่อยต่างๆที่ทำหน้าที่ในการประมวลผลภาพดิจิทัล มีลักษณะโครงสร้างประกอบกันเป็นสถาปัตยกรรมแบบโมดูลาร์ที่ประกอบด้วยโมดูลหลักทำหน้าที่ในการควบคุมดูแลการทำงานและการจ่ายงานให้กับโมดูลย่อยๆตามลำดับคำสั่งที่ได้รับ โมดูลย่อยจำนวน 8 โมดูลทำหน้าที่ในการประมวลผลข้อมูลภาพดิจิทัลตามลักษณะงาน และหน่วยความจำขนาด 64K จำนวน 2 ตัวเพื่อเก็บข้อมูลภาพ ขนาดของบัสข้อมูลมีขนาด 8 บิตและบัสแอดเดรสมีขนาด 16 บิต

4.1.1 พฤติกรรมของระบบ (Behavioral of System)

พฤติกรรมการทำงานของระบบถูกกำหนดให้แต่ละโมดูลทำงานเป็นอิสระต่อกัน โดยมีโมดูลหลักทำหน้าที่รับคำสั่งจากภายนอกและควบคุมให้มีการปฏิบัติเป็นไปตามลำดับคำสั่ง แต่ละโมดูลใช้บัลแอตเตรสและบัสข้อมูลร่วมกันในการติดต่อกับหน่วยความจำ

4.1.1.1 ชุดคำสั่ง (Instruction Set)

รูปที่ 4.1 แสดงชุดคำสั่งที่ใช้ในการควบคุมการทำงานของระบบจำนวน 12 คำสั่ง แต่ละคำสั่งมีความหมายดังนี้ คำสั่ง CLM ใช้ลบข้อมูลในหน่วยความจำทั้งสองตัว คำสั่ง SWM ใช้ในการย้ายข้อมูลภาพจากหน่วยความจำปลายทางไปยังหน่วยความจำต้นทางเพื่อเตรียมข้อมูลสำหรับการทำงานของโมดูลต่อไป คำสั่ง MOD no. ตามด้วยหมายเลขของโมดูลใช้สำหรับกำหนดเลือกให้โมดูลใด ๆ

Instruction Mnemonic	Brief Description	Bits	Opcode																
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
CLM	Clear Both Memory	8	0	0	0	0	0	0	0	1									-
SWM	Swap Memory	8	0	0	0	0	0	0	0	1	0							-	
MOD1	Select Module 1	8	0	0	0	0	0	1	0	0	0							-	
MOD2	Select Module 2	8	0	0	0	0	0	1	0	0	1							-	
MOD3	Select Module 3	8	0	0	0	0	0	1	0	1	0							-	
MOD4	Select Module 4	8	0	0	0	0	0	1	0	1	1							-	
MOD5	Select Module 5	8	0	0	0	0	0	1	1	0	0							-	
MOD6	Select Module 6	8	0	0	0	0	0	1	1	0	1							-	
MOD7	Select Module 7	8	0	0	0	0	0	1	1	1	0							-	
MOD8	Select Module 8	8	0	0	0	0	0	1	1	1	1							-	
ROW no.	Number of Rows	16	1	1	1	1	0	0	1	1							no.		
COL no.	Number of Columns	16	1	1	1	1	1	1	0	0							no.		

รูปที่ 4.1 แสดงชุดคำสั่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาดูเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

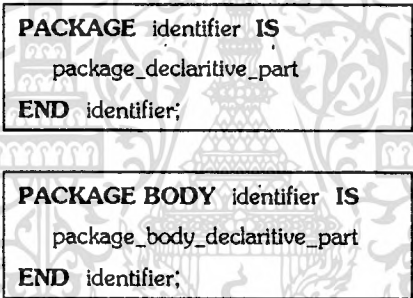
ทำงาน(การทำงานของโมดูลจะกระทำกับข้อมูลที่อยู่ในหน่วยความจำต้นทางและนำผลที่ได้เก็บลงในหน่วยความจำปลายทางเสมอ) คำสั่ง ROW no. และ COL no. ใช้กำหนดขนาดของข้อมูลภาพเป็นจำนวนแถวและคอลัมน์ซึ่งสามารถกำหนดได้สูงสุดจำนวน 256 แถวในแต่ละแถวกำหนดได้ 256 คอลัมน์

4.1.1.2 กำหนดไลบรารี (Design Libraries)

ในการออกแบบได้กำหนดไลบรารีสำหรับยูทิลิตี้ ฟังก์ชัน ไพรซีเจอร์ เกท รีจิสเตอร์ และอุปกรณ์ต่างๆไว้ 7 ไลบรารีด้วยกันคือ ไลบรารี “util” เป็นที่เก็บรายละเอียดของชนิดข้อมูล ชุดคำสั่ง ฟังก์ชันและไพรซีเจอร์ต่างๆ ไลบรารี “gates” เก็บรายละเอียดของเกทพื้นฐานและฟลิปฟล็อปต่างๆ ไลบรารี “reg” เก็บรายละเอียดของรีจิสเตอร์และลอจิกต่างๆ ไลบรารี “memo” เก็บรายละเอียดของหน่วยความจำ ไลบรารี “ctrl” เก็บรายละเอียดของโมดูลควบคุมหลัก ไลบรารี “edge” เก็บรายละเอียดของโมดูล edge detection ไลบรารี “hgram” เก็บรายละเอียดของโมดูล histogram equalization การอ้างอิงและการเรียกใช้ไลบรารีใช้คำสั่ง LIBRARY และ USE

4.1.1.3 แพคเกจ (Packages)

แพคเกจคือกลุ่มของชนิดข้อมูล โปรแกรมย่อยหรืออุปกรณ์ย่อยต่างๆที่ผู้ออกแบบได้ออกแบบไว้แล้วนำมารวบรวมไว้เป็นกลุ่ม อยู่นภายในไลบรารีเพื่อให้ผู้ออกแบบสามารถเรียกใช้ได้โดยสะดวกดังรูปที่ 4.2



รูปที่ 4.2 แสดงโครงสร้างของแพคเกจ

4.1.1.3.1 Basic_Uilities Package

แพคเกจ basic_utilities เป็นแพคเกจที่อยู่ในไลบรารี “util” ภายในได้กำหนดชนิดของข้อมูล ฟังก์ชัน และไพรซีเจอร์ต่างๆเอาไว้เพื่อใช้ในการบรรยายการทำงานของวงจรในระดับพฤติกรรมดังรูปที่ 4.3 ซึ่งแสดงการกำหนดชนิดของข้อมูล รูปแบบของฟังก์ชันและไพรซีเจอร์

```
PACKAGE basic_utilities IS
    TYPE qit IS ('0', '1', 'Z', 'X');
    TYPE qit_2d IS ARRAY (qit, qit) OF qit;
    TYPE qit_1d IS ARRAY (qit) OF qit;
    TYPE qit_vector IS ARRAY (NATURAL RANGE <>) OF qit;
    PROCEDURE qit2int (qin : IN qit_vector; int : OUT INTEGER);
    PROCEDURE int2qit (int : IN INTEGER; qin : OUT qit_vector);
    FUNCTION inc (x : qit_vector) RETURN qit_vector;
    FUNCTION "AND" (a, b : qit) RETURN qit;
    FUNCTION "OR" (a, b : qit) RETURN qit;
    FUNCTION "NOT" (a : qit) RETURN qit;
    FUNCTION wire (a, b : qit) RETURN qit;
```

```

FUNCTION oring ( drivers : qit_vector) RETURN qit;
SUBTYPE ored_qit IS oring qit;
TYPE ored_qit_vector IS ARRAY (NATURAL RANGE <>) OF ored_qit;
FUNCTION anding ( drivers : qit_vector) RETURN qit;
SUBTYPE anded_qit IS anding qit;
TYPE anded_qit_vector IS ARRAY (NATURAL RANGE <>) OF anded_qit;
FUNCTION wiring ( drivers : qit_vector) RETURN qit;
SUBTYPE wired_qit IS wiring qit;
TYPE wired_qit_vector IS ARRAY (NATURAL RANGE <>) OF wired_qit;
END basic_utilities;

```

รูปที่ 4.3 แสดงส่วนของการประกาศใน *basic_utilities*

qit เป็นชนิดของข้อมูลที่ประกอบด้วยค่าของลอจิก 4 ค่าคือ '0', '1', 'Z' และ 'X' ในขณะที่ *qit_1d*, *qit_2d* และ *qit_vector* เป็นชนิดของข้อมูลแบบอะเรย์ โปรซีเจอร์ *qit2int* และ *int2qit* ใช้สำหรับเปลี่ยนชนิดของข้อมูลจาก *qit* เป็น *integer* และ *integer* เป็น *qit* ตามลำดับ ดังแสดงการทำงานในรูปที่ 4.4 (a) และ (b)

```

PROCEDURE qit2int (qin : IN qit_vector; int : OUT INTEGER) IS
    VARIABLE result : INTEGER;
BEGIN
    result := 0;
    FOR i IN 0 TO (qin'LENGTH - 1) LOOP
        IF qin(i) = '1' THEN
            result := result + 2**i;
        END IF;
    END LOOP;
    int := result;
END qit2int;

```

(a)

```

PROCEDURE int2qit (int : IN INTEGER; qin : OUT qit_vector) IS
    VARIABLE tmp : INTEGER;
BEGIN
    tmp := int;
    FOR i IN 0 TO (qin'LENGTH - 1) LOOP
        IF (tmp MOD 2 = 1) THEN
            qin(i) := '1';
        ELSE qin(i) := '0';
        END IF;
        tmp := tmp / 2;
    END LOOP;
END int2qit;

```

(b)

รูปที่ 4.4 (a) โปรซีเจอร์ *qit2int*

(b) โปรซีเจอร์ *int2qit*

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 4.5 แสดงการทำงานของฟังก์ชัน *inc* ซึ่งเป็นฟังก์ชันสำหรับการเพิ่มค่าทีละหนึ่ง

```

FUNCTION inc (x : qit_vector) RETURN qit_vector IS
    VARIABLE i : INTEGER;
    VARIABLE t : qit_vector(x'RANGE);
BEGIN
    qit2int (x, i);
    i := i + 1;
    IF i >= 2**x'LENGTH THEN
        i := 0;
    END IF;
    int2qit (i, t);
    RETURN t;
END inc;

```

รูปที่ 4.5 ฟังก์ชัน *inc*

ฟังก์ชัน "AND", "OR", "NOT" และ *wire* ใช้แทนการกระทำทางลอจิกของข้อมูลที่มีชนิดเป็น *qit* รูปที่ 4.6 - 4.9 แสดงการกระทำของฟังก์ชันทั้งสี่

a:	0	1	Z	X
b:	0	0	0	0
1	0	1	1	X
Z	0	1	1	X
X	0	X	X	X

$$z = a . b$$

```

FUNCTION "AND" (a, b : qit) RETURN qit IS
    CONSTANT qit_and_table : qit_2d := (
        ('0','0','0','0'),
        ('0','1','1','X'),
        ('0','1','1','X'),
        ('0','X','X','X'));
BEGIN
    RETURN qit_and_table (a, b);
END "AND";

```

รูปที่ 4.6 ฟังก์ชัน "AND"

a:	0	1	Z	X
b:	0	1	1	X
	1	1	1	1
	1	1	1	1
	X	1	1	X

$z = a + b$

```

FUNCTION "OR" (a, b : qit) RETURN qit IS
  CONSTANT qit_or_table : qit_2d := (
    ('0','1','1','X'),
    ('1','1','1','1'),
    ('1','1','1','1'),
    ('X','1','1','X'));
BEGIN
  RETURN qit_or_table (a, b);
END "OR";

```

รูปที่ 4.7 ฟังก์ชัน "OR"

a:	0	1
	1	0
	Z	0
	X	X

$z = a'$

```

FUNCTION "NOT" (a : qit) RETURN qit IS
  CONSTANT qit_not_table : qit_1d := ('1','0','0','X');
BEGIN
  RETURN qit_not_table (a);
END "NOT";

```

รูปที่ 4.8 ฟังก์ชัน "NOT"

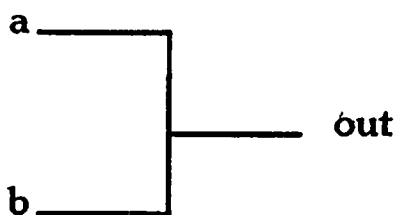
a:	0	1	Z	X
b:	0	X	0	X
	1	X	1	X
	Z	0	Z	X
	X	X	X	X

```

FUNCTION wire (a, b : qit) RETURN qit IS
    CONSTANT qit_wire_table : qit_2d := (
                                                ('0','X','0','X'),
                                                ('X','1','1','X'),
                                                ('0','1','Z','X'),
                                                ('X','X','X','X'));

    BEGIN
        RETURN qit_wire_table (a, b);
    END wire;

```



รูปที่ 4.9 ฟังก์ชัน wire

ฟังก์ชัน *anding*, *oring* และ *wiring* เป็นฟังก์ชันสำหรับการเลือกสรรข้อมูล (resolving between several driving values) ดังแสดงในรูปที่ 4.10-4.12

```

FUNCTION anding (drivers : qit_vector) RETURN qit IS
    VARIABLE accumulate : qit := '1';
    BEGIN
        FOR i IN drivers'RANGE LOOP
            accumulate := accumulate AND drivers(i);
        END LOOP;
        RETURN accumulate;
    END anding;

```

รูปที่ 4.10 ฟังก์ชัน anding

```

FUNCTION oring (drivers : qit_vector) RETURN qit IS
    VARIABLE accumulate : qit := '0';
    BEGIN
        FOR i IN drivers'RANGE LOOP
            accumulate := accumulate OR drivers(i);
        END LOOP;
        RETURN accumulate;
    END oring;

```

รูปที่ 4.11 ฟังก์ชัน oring

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษานั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

FUNCTION wiring (drivers : qit_vector) RETURN qit IS
    VARIABLE accumulate : qit := 'Z';
BEGIN
    FOR i IN drivers'RANGE LOOP
        accumulate := wire(accumulate, drivers(i));
    END LOOP;
    RETURN accumulate;
END wiring;

```

รูปที่ 4.12 ฟังก์ชัน *wiring*

4.1.1.3.2 Image_utilities Package

แพ็คเกจ *image_utilities* เป็นแพ็คเกจในไลบรารี *util* เก็บรายละเอียดเกี่ยวกับชนิดข้อมูลฟังก์ชัน และไพรอซีเจอร์ที่จำเป็นต้องใช้ในการบรรยายการทำงานของวงจรต่าง ๆ และไม่มีอยู่ในแพ็คเกจ *basic_utilities* รูปที่ 4.13 แสดงส่วนการประกาศของแพ็คเกจนี้ ในแพ็คเกจมีการเรียกใช้และอ้างอิงถึงรายละเอียดที่อยู่ในแพ็คเกจ *basic_utilities* และแพ็คเกจมาตรฐาน(standard package)ด้วย

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE STD.ALL;
PACKAGE image_utilities IS
    SUBTYPE nibble IS qit_vector (3 DOWNT0 0);
    SUBTYPE byte IS qit_vector (7 DOWNT0 0);
    SUBTYPE word IS qit_vector (15 DOWNT0 0);
    TYPE byte256 IS ARRAY (0 TO 255) OF byte;
    TYPE word256 IS ARRAY (0 TO 255) OF word;
    SUBTYPE wired_nibble IS wired_qit_vector (3 DOWNT0 0);
    SUBTYPE wired_byte IS wired_qit_vector (7 DOWNT0 0);
    SUBTYPE wired_ten IS wired_qit_vector (9 DOWNT0 0);
    SUBTYPE wired_word IS wired_qit_vector (15 DOWNT0 0);
    SUBTYPE ored_nibble IS ored_qit_vector (3 DOWNT0 0);
    SUBTYPE ored_byte IS ored_qit_vector (7 DOWNT0 0);
    SUBTYPE ored_word IS ored_qit_vector (15 DOWNT0 0);
    CONSTANT zero_4 : nibble := "0000";
    CONSTANT zero_8 : byte := "00000000";
    CONSTANT zero_16 : word := "0000000000000000";
    FUNCTION "AND" (a, b : qit_vector) RETURN qit_vector;
    FUNCTION "OR" (a, b : qit_vector) RETURN qit_vector;
    FUNCTION "NOT" (a : qit_vector) RETURN qit_vector;
    PROCEDURE qit2char (qin : IN qit; char : OUT CHARACTER);
    PROCEDURE qit_vector2str (qin : IN qit_vector; str : OUT STRING);
    PROCEDURE wqit_vector2str (qin : IN wired_qit_vector; str : OUT STRING);
END image_utilities;

```

รูปที่ 4.13 แสดงส่วนการประกาศของแพ็คเกจ *image_utilities*

เอกสารนี้เป็นเอกสารลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในที่นี้ฟังก์ชัน "AND", "OR" และ "NOT" ถูกกำหนดอีกครั้งแต่เป็นการกระทำกับข้อมูลชนิดอะเรย์ ดังรูปที่ 4.14-4.16

```
FUNCTION "AND" (a, b : qit_vector) RETURN qit_vector IS
  VARIABLE r : qit_vector (a'RANGE);
BEGIN
  loop1: FOR i IN a'RANGE LOOP
    r(i) := a(i) AND b(i);
  END LOOP loop1;
  RETURN r;
END "AND";
```

รูปที่ 4.14 ฟังก์ชัน "AND"

```
FUNCTION "OR" (a, b : qit_vector) RETURN qit_vector IS
  VARIABLE r : qit_vector (a'RANGE);
BEGIN
  loop1: FOR i IN a'RANGE LOOP
    r(i) := a(i) OR b(i);
  END LOOP loop1;
  RETURN r;
END "OR";
```

รูปที่ 4.15 ฟังก์ชัน "OR"

```
FUNCTION "NOT" (a : qit_vector) RETURN qit_vector IS
  VARIABLE r : qit_vector (a'RANGE);
BEGIN
  loop1: FOR i IN a'RANGE LOOP
    r(i) := NOT a(i);
  END LOOP loop1;
  RETURN r;
END "NOT";
```

รูปที่ 4.16 ฟังก์ชัน "NOT"

โพรซีเจอร์ *qit2char*, *qit_vector2str* และ *wqit_vector2str* เป็นโพรซีเจอร์สำหรับการเปลี่ยนชนิดของข้อมูลจาก *qit* เป็น CHARACTER, *qit_vector* เป็น STRING และ *wired_qit_vector* เป็น STRING ตามลำดับ ดังรูปที่ 4.17-4.19

```
PROCEDURE qit2char (qin: IN qit; char: OUT CHARACTER) IS
  VARIABLE ch : CHARACTER;
BEGIN
  IF qin = '0' THEN ch := '0';
  ELSIF qin = '1' THEN ch := '1';
  ELSIF qin = 'Z' THEN ch := 'Z';
```

```

ELSE ch := 'X';
END IF;
char := ch;
END qit2char;

```

รูปที่ 4.17 โพรซีเจอร์ qit2char

```

PROCEDURE qit_vector2str (qin: IN qit_vector; str: OUT STRING) IS
  VARIABLE st : STRING (qin'RANGE);
BEGIN
  FOR i IN qin'RANGE LOOP
    IF qin(i) = '0' THEN st(i) := '0';
    ELSIF qin(i) = '1' THEN st(i) := '1';
    ELSIF qin(i) = 'Z' THEN st(i) := 'Z';
    ELSE st(i) := 'X';
    END IF;
  END LOOP;
  str := st;
END qit_vector2str;

```

รูปที่ 4.18 โพรซีเจอร์ qit_vector2str

```

PROCEDURE wqit_vector2str (qin: IN wired_qit_vector; str: OUT STRING) IS
  VARIABLE st : STRING (qin'RANGE);
BEGIN
  FOR i IN qin'RANGE LOOP
    IF qin(i) = '0' THEN st(i) := '0';
    ELSIF qin(i) = '1' THEN st(i) := '1';
    ELSIF qin(i) = 'Z' THEN st(i) := 'Z';
    ELSE st(i) := 'X';
    END IF;
  END LOOP;
  str := st;
END wqit_vector2str;

```

รูปที่ 4.19 โพรซีเจอร์ wqit_vector2str

4.1.1.3.3 Image_Parameters Package

image_parameters เป็นแพ็คเกจที่เก็บรวบรวมคำสั่งในการควบคุมทั้งหมดอยู่ในไลบรารี "util" ดังรูปที่ 4.20

```

LIBRARY util;
USE util.basic_utilities.ALL;
PACKAGE image_parameters IS
  CONSTANT single_byte_instructions : qit_vector (3 DOWNT0 0) := '0000';

```

```

CONSTANT dlm : qit_vector (3 DOWNT0 0) := "0001";
CONSTANT swm : qit_vector (3 DOWNT0 0) := "0010";
CONSTANT mod1 : qit_vector (3 DOWNT0 0) := "1000";
CONSTANT mod2 : qit_vector (3 DOWNT0 0) := "1001";
CONSTANT mod3 : qit_vector (3 DOWNT0 0) := "1010";
CONSTANT mod4 : qit_vector (3 DOWNT0 0) := "1011";
CONSTANT mod5 : qit_vector (3 DOWNT0 0) := "1100";
CONSTANT mod6 : qit_vector (3 DOWNT0 0) := "1101";
CONSTANT mod7 : qit_vector (3 DOWNT0 0) := "1110";
CONSTANT mod8 : qit_vector (3 DOWNT0 0) := "1111";
CONSTANT row : qit_vector (3 DOWNT0 0) := "0011";
CONSTANT col : qit_vector (3 DOWNT0 0) := "1100";

END image_parameters;

```

รูปที่ 4.20 แสดงแพ็คเกจ *image_parameters*

4.1.1.3.4 Gates_Flipflop Package

เป็นแพ็คเกจที่เก็บรวบรวมอุปกรณ์ประเภทเกทพื้นฐานและฟลิปฟล็อปต่างๆอยู่ในไลบรารี "gates" ในการอ้างอิงถึงอุปกรณ์จะใช้คำว่า COMPONENT และ END COMPONENT ดังรูปที่ 4.21 และ 4.22

```

COMPONENT identifier
    PORT ( interface description );
END COMPONENT;

```

รูปที่ 4.21 แสดงการประกาศส่วนอุปกรณ์

```

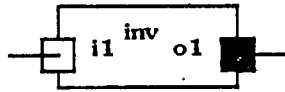
LIBRARY util;
USE util.basic_utilities.ALL;
PACKAGE gates_flipflop IS
    COMPONENT df PORT ( d, en, clk, clr : IN qit; q : OUT qit); END COMPONENT;
    COMPONENT dl PORT ( d, en, clk : IN qit; q : OUT qit); END COMPONENT;
    COMPONENT jkcf PORT ( j, k, clk, clr : IN qit; q : OUT qit); END COMPONENT;
    COMPONENT msdff PORT ( d, clk : IN qit; q : OUT qit); END COMPONENT;
    COMPONENT inv PORT ( i1 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT a2 PORT ( i1, i2 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT a3 PORT ( i1, i2, i3 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT a4 PORT ( i1, i2, i3, i4 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT o2 PORT ( i1, i2 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT o3 PORT ( i1, i2, i3 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT o4 PORT ( i1, i2, i3, i4 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT n2 PORT ( i1, i2 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT n3 PORT ( i1, i2, i3 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT n4 PORT ( i1, i2, i3, i4 : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT ts PORT ( i1, c : IN qit; o1 : OUT qit); END COMPONENT;
    COMPONENT delay PORT ( i1 : IN qit; o1 : OUT qit); END COMPONENT;

END gates_flipflop;

```

รูปที่ 4.22 แสดงแพ็คเกจ *gates_flipflop*

รูปที่ 4.23 แสดงการบรรยายการทำงานของเกท *inverter* ในการบรรยายได้กำหนดค่าเวลาหน่วงไว้ 2 ค่าคือ *tphl* และ *tph* ซึ่งเป็นค่าเวลาหน่วงของสัญญาณขาขึ้นและขาลง (rise and fall propagation delays)



```

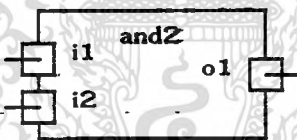
LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY inv_q IS
    GENERIC ( tphl : TIME := 5 NS; tph : TIME := 3 NS);
    PORT ( i1 :IN qit; o1 : OUT qit);
END inv_q;
ARCHITECTURE double_delay OF inv_q IS
BEGIN
    o1 <= '1' AFTER tphl WHEN i1 = '0' ELSE
        '0' AFTER tphl WHEN i1 = '1' OR i1 = 'Z' ELSE
        'X' AFTER tphl;
END double_delay;

```

รูปที่ 4.23 แสดงการบรรยายการทำงานของเกท *inverter*

รูปที่ 4.24-4.26 แสดงการบรรยายการทำงานของ AND เกท 2,3 และ 4 อินพุตตาม

ลำดับ



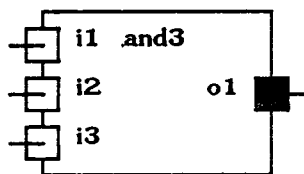
```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY and2_q IS
    GENERIC ( tphl : TIME := 7 NS; tph : TIME := 5 NS);
    PORT ( i1, i2 :IN qit; o1 : OUT qit);
END and2_q;
ARCHITECTURE double_delay OF and2_q IS
BEGIN
    o1 <= '0' AFTER tphl WHEN i1 = '0' OR i2 = '0' ELSE
        '1' AFTER tphl WHEN (i1 = '1' AND i2 = '1') OR
            (i1 = '1' AND i2 = 'Z') OR
            (i1 = 'Z' AND i2 = '1') OR
            (i1 = 'Z' AND i2 = 'Z') ELSE
        'X' AFTER tphl;
END double_delay;

```

รูปที่ 4.24 แสดงการบรรยายการทำงานของเกท *and-2*

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

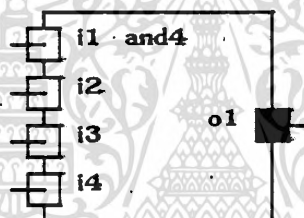


```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY and3_q IS
    GENERIC ( tplh : TIME := 8 NS; tphl : TIME := 6 NS);
    PORT ( i1, i2, i3 :IN qit; o1 : OUT qit);
END and3_q;
ARCHITECTURE average_delay OF and3_q IS
BEGIN
    o1 <= i1 AND i2 AND i3 AFTER (tplh + tphl) / 2;
END average_delay;

```

รูปที่ 4.25 แสดงการบรรยายการทำงานของเกท and-3



```

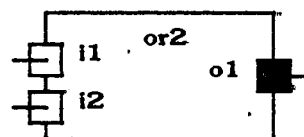
LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY and4_q IS
    GENERIC ( tplh : TIME := 9 NS; tphl : TIME := 7 NS);
    PORT ( i1, i2, i3, i4 :IN qit; o1 : OUT qit);
END and4_q;
ARCHITECTURE average_delay OF and4_q IS
BEGIN
    o1 <= i1 AND i2 AND i3 AND i4 AFTER (tplh + tphl) / 2;
END average_delay;

```

รูปที่ 4.26 แสดงการบรรยายการทำงานของเกท and-4

รูปที่ 4.27-4.29 แสดงการบรรยายการทำงานของ OR เกท 2,3 และ 4 อินพุตตาม

ลำดับ

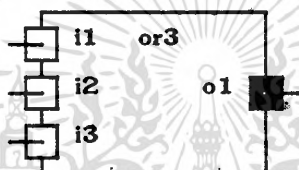


```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY or2_q IS
    GENERIC ( tph : TIME := 7 NS; tphl : TIME := 5 NS);
    PORT ( i1, i2 :IN qit; o1 : OUT qit);
END or2_q;
ARCHITECTURE double_delay OF or2_q IS
BEGIN
    o1 <= '0' AFTER tph WHEN i1 = '0' AND i2 = '0' ELSE
        '1' AFTER tph WHEN (i1 = '1' OR i2 = '1') OR
            (i1 = 'Z' OR i2 = 'Z') ELSE
        'X' AFTER tph;
END double_delay;

```

รูปที่ 4.27 แสดงการบรรยายการทำงานของเกท or-2

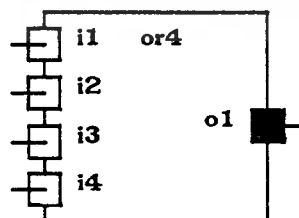


```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY or3_q IS
    GENERIC ( tph : TIME := 8 NS; tphl : TIME := 6 NS);
    PORT ( i1, i2, i3 :IN qit; o1 : OUT qit);
END or3_q;
ARCHITECTURE average_delay OF or3_q IS
BEGIN
    o1 <= i1 OR i2 OR i3 AFTER (tph + tphl) / 2;
END average_delay;

```

รูปที่ 4.28 แสดงการบรรยายการทำงานของเกท or-3



```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY or4_q IS
    GENERIC ( tph : TIME := 9 NS; tphl : TIME := 7 NS);
    PORT ( i1, i2, i3, i4 :IN qit; o1 : OUT qit);
END or4_q;

```

ARCHITECTURE average_delay OF or4_q IS

BEGIN

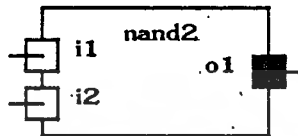
o1 <= i1 OR i2 OR i3 OR i4 AFTER (tphl + tphi) / 2;

END average_delay;

รูปที่ 4.29 แสดงการบรรยายการทำงานของเกท or-4

รูปที่ 4.30-4.32 แสดงการบรรยายการทำงานของ NAND เกท 2,3 และ 4 อินพุต

ตามลำดับ



LIBRARY util;

USE util.basic_utilities.ALL;

ENTITY nand2_q IS

GENERIC (tphl : TIME := 7 NS; tphi : TIME := 5 NS);

PORT (i1, i2 :IN qit; o1 : OUT qit);

END nand2_q;

ARCHITECTURE double_delay OF nand2_q IS

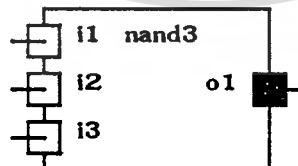
BEGIN

o1 <= '1' AFTER tphl WHEN i1 = '0' OR i2 = '0' ELSE
 '0' AFTER tphi WHEN (i1 = '1' AND i2 = '1') OR
 (i1 = '1' AND i2 = 'Z') OR
 (i1 = 'Z' AND i2 = '1') OR
 (i1 = 'Z' AND i2 = 'Z') ELSE

'X' AFTER tphi;

END double_delay;

รูปที่ 4.30 แสดงการบรรยายการทำงานของเกท nand-2



LIBRARY util;

USE util.basic_utilities.ALL;

ENTITY nand3_q IS

GENERIC (tphl : TIME := 8 NS; tphi : TIME := 6 NS);

PORT (i1, i2, i3 :IN qit; o1 : OUT qit);

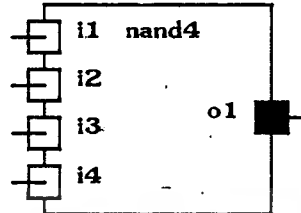
END nand3_q;

```

ARCHITECTURE average_delay OF nand3_q IS
BEGIN
  o1 <= NOT (i1 AND i2 AND i3) AFTER (tphl + tphi) / 2;
END average_delay;

```

รูปที่ 4.31 แสดงการบรรยายการทำงานของเกท nand-3



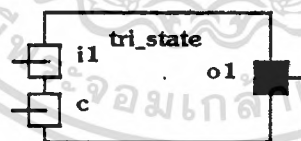
```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY nand4_q IS
  GENERIC ( tphl : TIME := 9 NS; tphi : TIME := 7 NS);
  PORT ( i1, i2, i3, i4 :IN qit; o1 : OUT qit);
END nand4_q;
ARCHITECTURE average_delay OF nand4_q IS
BEGIN
  o1 <= NOT (i1 AND i2 AND i3 AND i4) AFTER (tphl + tphi) / 2;
END average_delay;

```

รูปที่ 4.32 แสดงการบรรยายการทำงานของเกท nand-4

รูปที่ 4.33 แสดงการบรรยายการทำงานของ tri_state เกท ซึ่งทำหน้าที่ในการส่งผ่านสัญญาณเมื่อได้รับอนุญาต($c = 1$)



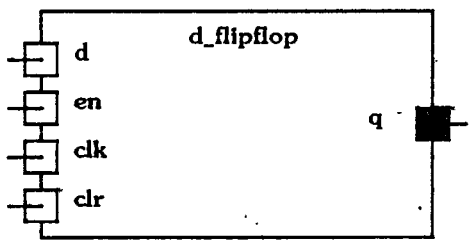
```

LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY tri_state_q IS
  GENERIC ( delay : TIME := 10 NS);
  PORT ( i1, c :IN qit; o1 : OUT qit);
END tri_state_q;
ARCHITECTURE single_delay OF tri_state_q IS
BEGIN
  o1 <= '0' AFTER delay WHEN i1 = '0' AND c = '1' ELSE
    '1' AFTER delay WHEN i1 = '1' AND c = '1' ELSE
    'Z' AFTER delay;
END single_delay;

```

รูปที่ 4.33 แสดงการบรรยายการทำงานของ tri_state
รูปที่ 4.34 และ 4.35 แสดงการบรรยายการทำงานของ d-flipflop ในระดับข้อมูลและ

โครงสร้าง



```
LIBRARY util;
USE util.basic_utilities.ALL;
ENTITY dff IS
    GENERIC ( delay : TIME := 10 NS);
    PORT ( d, en, clk, clr :IN qit;  q : OUT qit);
END dff;
ARCHITECTURE dataflow OF dff IS
BEGIN
    clear : BLOCK (clr = '1')
    BEGIN
        q <= GUARDED '0' AFTER delay;
    END BLOCK clear;
    clocking : BLOCK (clk = '0' AND NOT clk'STABLE)
    BEGIN
        enable : BLOCK (en = '1' AND GUARD)
        BEGIN
            q <= GUARDED d AFTER delay;
        END BLOCK enable;
    END BLOCK clocking;
END dataflow;
```

รูปที่ 4.34 แสดงการบรรยายการทำงานในระดับข้อมูลของ d-flipflop

```
LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY dff IS
    PORT ( d, en, clk, clr :IN qit;  q : OUT qit);
END dff;
ARCHITECTURE gate_level OF dff IS
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
    FOR ALL : n2 USE ENTITY gates.nand2_q (double_delay);
    SIGNAL im1,im2,im3,im4,im5,im6,im7,im8,im9 : qit;
BEGIN
    g0 : inv PORT MAP (d, im1);
```

```

g1 : inv PORT MAP (clk, im2);
g2 : inv PORT MAP (clr, im6);
g3 : a2 PORT MAP (en, im2, im3);
g4 : n2 PORT MAP (d, im3, im4);
g5 : n2 PORT MAP (im1, im3, im5);
g6 : n2 PORT MAP (im4, im8, im7);
g7 : n2 PORT MAP (im5, im9, im8);
g8 : a2 PORT MAP (im6, im7, im9);
q <= im9;
END gate_level;

```

รูปที่ 4.35 แสดงการบรรยายการทำงานในระดับโครงสร้างของ *d-flipflop*

รูปที่ 4.36 แสดงการบรรยายการทำงานของ *d-latch* ซึ่งคล้ายกับการทำงานของ *d-flipflop*. แต่ไม่มีอินพุต *clr*



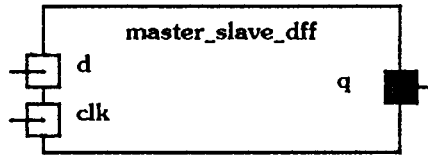
```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY d_latch IS
  PORT ( d, en, clk :IN qit; q : OUT qit);
END d_latch;
ARCHITECTURE gate_level OF d_latch IS
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
  FOR ALL : n2 USE ENTITY gates.nand2_q (double_delay);
  SIGNAL im1,im2,im3,im4,im5,im6,im7 : qit;
BEGIN
  g0 : inv PORT MAP (d, im1);
  g1 : inv PORT MAP (clk, im2);
  g2 : a2 PORT MAP (en, im2, im3);
  g3 : n2 PORT MAP (d, im3, im4);
  g4 : n2 PORT MAP (im1, im3, im5);
  g5 : n2 PORT MAP (im4, im7, im6);
  g6 : n2 PORT MAP (im5, im6, im7);
  q <= im6;
END gate_level;

```

รูปที่ 4.36 แสดงการบรรยายการทำงานของ *d-latch*

รูปที่ 4.37 แสดงการบรรยายการทำงานของ *master-slave-dff* ซึ่งจะนำมาใช้ในส่วนของการส่งสัญญาณควบคุม โดยจะรับอินพุตเมื่อสัญญาณ clock เป็น '1' และเปลี่ยนเอาต์พุตเมื่อสัญญาณ clock เปลี่ยนเป็น '0'

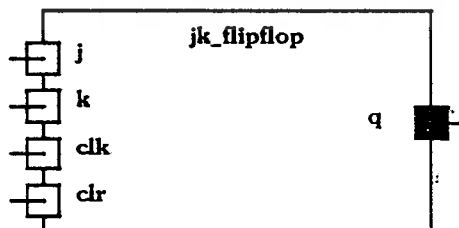


```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY master_slave_dff IS
    PORT ( d, clk :IN qit; q : OUT qit);
END master_slave_dff;
ARCHITECTURE gate_level OF master_slave_dff IS
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : n2 USE ENTITY gates.nand2_q (double_delay);
    FOR ALL : n3 USE ENTITY gates.nand3_q (average_delay);
    SIGNAL im0,im1,im2,im3,im4,im5,im6,im7,im8,im9 : qit;
BEGIN
    g0 : inv PORT MAP (d, im0);
    g1 : inv PORT MAP (clk, im1);
    g2 : n3 PORT MAP (d, clk, im9, im2);
    g3 : n3 PORT MAP (im0, clk, im8, im3);
    g4 : n2 PORT MAP (im2, im5, im4);
    g5 : n2 PORT MAP (im3, im4, im5);
    g6 : n2 PORT MAP (im1, im4, im6);
    g7 : n2 PORT MAP (im1, im5, im7);
    g8 : n2 PORT MAP (im6, im9, im8);
    g9 : n2 PORT MAP (im7, im8, im9);
    q <= im8;
END gate_level;

```

รูปที่ 4.37 แสดงการบรรยายการทำงานของ *master-slave-dff*



```

LIBRARY util;
LIBRARY gates
USE util.basic_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY jk_clr_ff IS

```

```

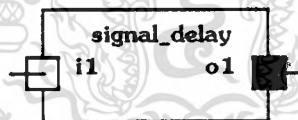
PORT ( j, k, clk, clr :IN qit; q : OUT qit);
END jk_clr_ff;

ARCHITECTURE gate_level OF jk_clr_ff IS
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
  FOR ALL : n2 USE ENTITY gates.nand2_q (double_delay);
  FOR ALL : n3 USE ENTITY gates.nand3_q (average_delay);
  SIGNAL im1,im2,im3,im4,im5,im6,im7,im8,im9 : qit;
BEGIN
  g0 : inv PORT MAP (clr, im1);
  g1 : inv PORT MAP (k, im2);
  g2 : inv PORT MAP (clk, im3);
  g3 : a2 PORT MAP (j, im1, im4);
  g4 : n2 PORT MAP (im1, im2, im5);
  g5 : n3 PORT MAP (im3, im4, im9, im6);
  g6 : n3 PORT MAP (im3, im5, im8, im7);
  g7 : n2 PORT MAP (im6, im9, im8);
  g8 : n2 PORT MAP (im7, im8, im9);
  q <= im8;
END gate_level;

```

รูปที่ 4.38 แสดงการบรรยายการทำงานของ *jk-flipflop*

รูปที่ 4.39 แสดงการบรรยายการทำงานของวงจรหน่วงสัญญาณโดยใช้ *inverter*



```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE gates.gates_flipflop.ALL;

ENTITY signal_delay IS
  PORT (i1 : IN qit; o1 : OUT qit);
END signal_delay;

ARCHITECTURE gate_level OF signal_delay IS
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  SIGNAL im : qit;
BEGIN
  g0 : inv PORT MAP (i1, im);
  g1 : inv PORT MAP (im, o1);
END gate_level;

```

4.1.1.3.5 Register_Logic Package

แพ็คเกจ *Register_Logic* เป็นแพ็คเกจในไลบรารี “reg” รวบรวมรายละเอียดเกี่ยวกับรีจิสเตอร์และวงจรพื้นฐานต่าง ๆ เอาไว้ดังแสดงในรูปที่ 4.40

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
PACKAGE register_logic IS
  COMPONENT reg -- register_unit
    PORT ( i8 : IN byte; o8 : OUT byte; load, ck : IN qit);
  END COMPONENT;
  COMPONENT rg8 -- register_unit_8
    PORT ( d : IN qit_vector(7 DOWNTO 0);
          q : OUT qit_vector(7 DOWNTO 0); load, ck : IN qit);
  END COMPONENT;
  COMPONENT pc -- program_counter_unit
    PORT ( o16 : OUT word; increment, reset, ck : IN qit);
  END COMPONENT;
  COMPONENT rpc -- row_col_counter_unit
    PORT ( o8 : OUT byte; increment, reset, ck : IN qit);
  END COMPONENT;
  COMPONENT mar -- memory_address_register_unit
    PORT ( i16 : IN word; o16 : OUT word;
          load_page, load_offset, ck : IN qit);
  END COMPONENT;
  COMPONENT cpc -- compare_col
    PORT ( i1, i2 : IN byte; o1 : OUT qit);
  END COMPONENT;
  COMPONENT cprc -- compare_row_col
    PORT ( i1, i2 : IN word; o1 : OUT qit);
  END COMPONENT;
  COMPONENT ts8 -- tri_state_8
    PORT ( i8 : IN byte; c : IN qit; o8 : OUT byte);
  END COMPONENT;
  COMPONENT ts10 -- tri_state_10
    PORT ( i : IN qit_vector(9 DOWNTO 0); c : IN qit; o : OUT qit_vector(9 DOWNTO 0));
  END COMPONENT;
  COMPONENT ts16 -- tri_state_16
    PORT ( i16 : IN word; c : IN qit; o16 : OUT word);
  END COMPONENT;
  COMPONENT inv10 -- 10_bit_inv_q
    PORT ( i : IN qit_vector(9 DOWNTO 0); o : OUT qit_vector(9 DOWNTO 0));
  END COMPONENT;
  COMPONENT add -- adder_8
    PORT ( i1, i2 : IN byte; o1 : OUT byte);
  END COMPONENT;

```

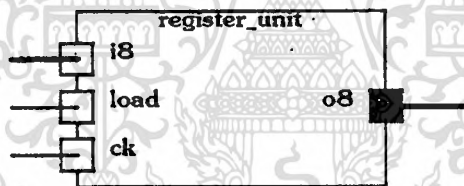
```

COMPONENT add8 -- 8_bit_adder
  PORT ( a, b : IN qit_vector(7 DOWNT0 0); cin : IN qit;
        sum : OUT qit_vector(7 DOWNT0 0); cout : OUT qit);
END COMPONENT;
COMPONENT add9 -- 9_bit_adder
  PORT ( a, b : IN qit_vector(8 DOWNT0 0); cin : IN qit;
        sum : OUT qit_vector(8 DOWNT0 0); cout : OUT qit);
END COMPONENT;
COMPONENT add10 -- 10_bit_adder
  PORT ( a, b : IN qit_vector(9 DOWNT0 0); cin : IN qit;
        sum : OUT qit_vector(9 DOWNT0 0); cout : OUT qit);
END COMPONENT;
COMPONENT d8_256 -- decoder_8_to_256
  PORT ( i8 : IN byte; en : IN qit;
        o : OUT qit_vector (0 TO 255));
END COMPONENT;
END register_logic;

```

รูปที่ 4.40 แสดงแพตเทจ Register_Logic

รูปที่ 4.41 และ 4.42 แสดงการบรรยายการทำงานของรีจิสเตอร์ขนาด 8 บิตในระดับข้อมูลและโครงสร้าง



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
ENTITY register_unit IS
  PORT ( i8 : IN byte; o8 : OUT byte; load, ck : IN qit);
END register_unit;

ARCHITECTURE dataflow OF register_unit IS
BEGIN
  enable : BLOCK (load = '1')
  BEGIN
    clocking : BLOCK ((ck = '0' AND NOT ck'STABLE) AND GUARD)
    BEGIN
      o8 <= GUARDED i8;
    END BLOCK clocking;
  END BLOCK enable;
END dataflow;

```

เอกสารนี้เป็นรูปที่ 4.41 แสดงการบรรยายการทำงานของรีจิสเตอร์ขนาด 8 บิตในระดับข้อมูล ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

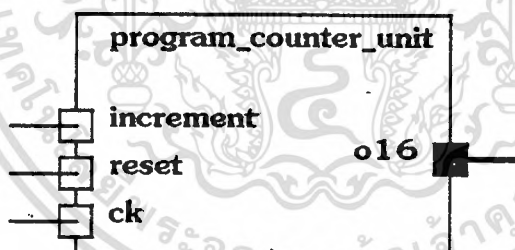
```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY register_unit_8 IS
  PORT ( d : IN qit_vector(7 DOWNT0 0);
        q : OUT qit_vector(7 DOWNT0 0);
        load, ck : IN qit);
END register_unit_8;
ARCHITECTURE structural OF register_unit_8 IS
  FOR ALL : di USE ENTITY gates.d_latch (gate_level);
BEGIN
  dr : FOR i IN d'RANGE GENERATE
    di : di PORT MAP (d(i), load, ck, q(i));
  END GENERATE;
END structural;

```

รูปที่ 4.42 แสดงการบรรยายการทำงานของรีจิสเตอร์ขนาด 8 บิตในระดับโครงสร้าง

รูปที่ 4.43 และ 4.44 แสดงการบรรยายการทำงานของ *program_counter_unit* ในระดับพฤติกรรมและโครงสร้าง การบรรยายในโปรเซสจะให้ความสำคัญกับ *ck* ตามด้วย *reset* และ *increment* และมีการเรียกใช้ฟังก์ชัน *inc* ในแพ็คเกจ *basic_utilities* ด้วย ส่วนการบรรยายในระดับโครงสร้างจะใช้ *jk-flipflop* เป็นตัวเก็บข้อมูล



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
ENTITY program_counter_unit IS
  PORT ( o16 : OUT word; increment, reset, ck : IN qit);
END program_counter_unit;
ARCHITECTURE behavioral OF program_counter_unit IS
BEGIN
  PROCESS (ck)
    VARIABLE internal_state : word := zero_16
  BEGIN
    IF (ck = '0') THEN
      IF reset = '1' THEN
        internal_state := zero_16;
      ELSIF increment = '1' THEN

```

```

        internal_state := inc(internal_state);
    END IF;
    o16 <= internal_state;
END IF;
END PROCESS;
END behavioral;

```

รูปที่ 4.43 แสดงการบรรยายการทำงานของ *program_counter_unit* ในระดับพฤติกรรม

```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY program_counter_unit IS
    PORT ( o16 : OUT word; increment, reset, ck : IN qit);
END program_counter_unit;
ARCHITECTURE structural OF program_counter_unit IS
    FOR ALL : jkcf USE ENTITY gates.jk_clr_ff (gate_level);
    CONSTANT n : INTEGER := 16;
    SIGNAL im : qit_vector (0 TO (n-1));
    SIGNAL am : qit_vector (1 TO (n-2));
BEGIN
    jk_all : FOR i IN 0 TO n-1 GENERATE
        i0 : IF i = 0 GENERATE
            least0 : jkcf PORT MAP (increment, increment, ck, reset, im(i));
        END GENERATE;
        i1 : IF i = 1 GENERATE
            least1 : jkcf PORT MAP (im(i-1), im(i-1), ck, reset, im(i));
        END GENERATE;
        m : IF i = n-1 GENERATE
            most : jkcf PORT MAP (am(i-1), am(i-1), ck, reset, im(i));
        END GENERATE;
        r : IF i > 1 AND i < n-1 GENERATE
            ag : a2 PORT MAP (im(i-2), im(i-1), am(i-1));
            rest : jkcf PORT MAP (am(i-1), am(i-1), ck, reset, im(i));
        END GENERATE;
    END GENERATE;
    o_all : FOR k IN 0 TO n-1 GENERATE
        o16(k) <= im(k);
    END GENERATE;
END structural;

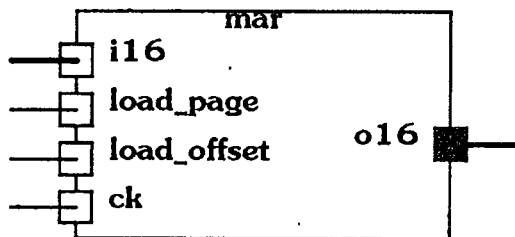
```

รูปที่ 4.44 แสดงการบรรยายการทำงานของ *program_counter_unit* ในระดับโครงสร้าง

รูปที่ 4.45 และ 4.46 แสดงการบรรยายการทำงานของ *memory_address_register_unit*

ซึ่งเป็นรีจิสเตอร์ขนาด 16 บิตใช้สำหรับเก็บตำแหน่งของหน่วยความจำ

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
ENTITY memory_address_register_unit IS
    PORT ( i16 : IN word; o16 : OUT word;
           load_page, load_offset, ck : IN qit);
END memory_address_register_unit;
ARCHITECTURE behavioral OF memory_address_register_unit IS
BEGIN
    PROCESS (ck)
        VARIABLE internal_state : word := zero_16;
    BEGIN
        IF (ck = '0') THEN
            IF load_page = '1' THEN
                internal_state (15 DOWNT0 8) := i16 (15 DOWNT0 8);
            END IF;
            IF load_offset = '1' THEN
                internal_state (7 DOWNT0 0) := i16 (7 DOWNT0 0);
            END IF;
            o16 <= internal_state;
        END IF;
    END PROCESS;
END behavioral;

```

รูปที่ 4.45 แสดงการบรรยายการทำงานของ *memory_address_register_unit* ระดับพฤติกรรม

```

LIBRARY util;
LIBRARY reg;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE reg.register_logic.ALL;
ENTITY memory_address_register_unit IS
    PORT ( i16 : IN word; o16 : OUT word;
           load_page, load_offset, ck : IN qit);
END memory_address_register_unit;

ARCHITECTURE structural OF memory_address_register_unit IS
    FOR ALL : rg8 USE ENTITY reg.register_unit_8 (structural);
    SIGNAL page_in, offset_in, page_out, offset_out : byte;
BEGIN

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

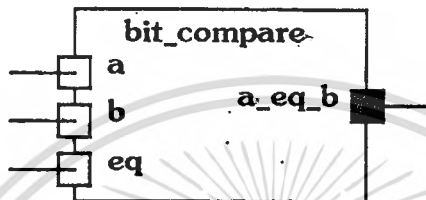
```

page_in <= i16 (15 DOWNT0 8);
offset_in <= i16 (7 DOWNT0 0);
r0 : rg8 PORT MAP ( offset_in, offset_out , load_offset, ck);
r1 : rg8 PORT MAP ( page_in, page_out, load_page, ck);
o16 <= page_out & offset_out;
END structural;

```

รูปที่ 4.46 แสดงการบรรยายการทำงานของ *memory_address_register_unit* ระดับโครงสร้าง

รูปที่ 4.47 แสดงการบรรยายการทำงานของวงจรเปรียบเทียบ *bit_compare_row_col* ซึ่งจะให้อาห์พุทเป็น '1' เมื่ออินพุทมีค่าเท่ากัน



```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE gates.gates_flipflop.ALL;
ENTITY bit_compare_row_col IS
  PORT ( a, b, eq : IN qit; a_eq_b : OUT qit);
END bit_compare_row_col;
ARCHITECTURE gate_level OF bit_compare_row_col IS
  FOR ALL : inv USE ENTITY gates.inv_q(double_delay);
  FOR ALL : n2 USE ENTITY gates.nand2_q(double_delay);
  FOR ALL : n3 USE ENTITY gates.nand3_q(average_delay);
  SIGNAL im1,im2,im3,im4 : qit;
BEGIN
  g0 : inv PORT MAP (a, im1);
  g1 : inv PORT MAP (b, im2);
  g2 : n3 PORT MAP (a, b, eq, im3);
  g3 : n3 PORT MAP (im1, im2, eq, im4);
  g4 : n2 PORT MAP (im3, im4, a_eq_b);
END gate_level;

```

รูปที่ 4.47 แสดงการบรรยายการทำงานของวงจรเปรียบเทียบ *bit_compare_row_col*

รูปที่ 4.48 แสดงการบรรยายการทำงานของวงจรเปรียบเทียบขนาด 8 บิต โดยกำหนดให้ค่า *CONSTANT n* เป็นค่ากำหนดขนาดบิต ในกรณีของวงจรเปรียบเทียบขนาด 16 บิตก็เพียงแต่เปลี่ยนค่าของ *n* เป็น 16 เท่านั้น

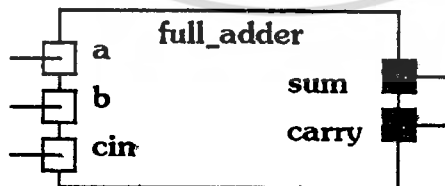
```

LIBRARY util;
LIBRARY reg;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
ENTITY compare_col IS
  PORT ( a, b : IN byte; eq : IN qit; a_eq_b : OUT qit);
END compare_col;
ARCHITECTURE structural OF compare_col IS
  COMPONENT bcp
    PORT ( a, b, eq : IN qit; a_eq_b : OUT qit);
  END COMPONENT;
  FOR ALL : bcp USE ENTITY reg.bit_compare_row_col (gate_level);
  CONSTANT n : INTEGER := 8;
  SIGNAL im : qit_vector (0 TO (n-2));
BEGIN
  c_all : FOR i IN 0 TO n-1 GENERATE
    l : IF i = 0 GENERATE
      least : bcp PORT MAP ( a(i), b(i), eq, im(i));
    END GENERATE;
    m : IF i = n-1 GENERATE
      most : bcp PORT MAP ( a(i), b(i), im(i-1), a_eq_b);
    END GENERATE;
    r : IF i > 0 AND i < n-1 GENERATE
      rest : bcp PORT MAP ( a(i), b(i), im(i-1), im(i));
    END GENERATE;
  END GENERATE;
END structural;

```

รูปที่ 4.48 แสดงการบรรยายการทำงานของวงจรเปรียบเทียบขนาด 8 บิต

รูปที่ 4.49 แสดงการบรรยายการทำงานของวงจรบวก(full_adder)ในระดับเกต ซึ่งจะนำไปใช้เป็นองค์ประกอบย่อยในการออกแบบวงจรบวกเลขขนาดต่าง ๆ ต่อไป



$$\text{SUM} = (a'b'cin) + (a'b.cin') + (a.b'.cin') + (a.b.cin)$$

$$\text{CARRY} = (a'b.cin) + (a.b'.cin) + (a.b.cin') + (a.b.cin)$$

```

LIBRARY util;
LIBRARY gates;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE gates.gates_flipflop.ALL;

```

```

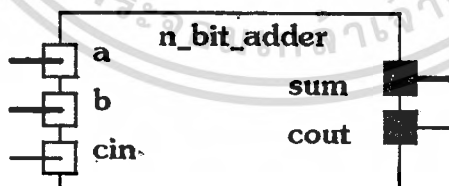
ENTITY full_adder IS
  PORT ( a, b, cin : IN qit; sum, carry : OUT qit);
END full_adder;

ARCHITECTURE gate_level OF full_adder IS
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : n2 USE ENTITY gates.nand2_q (double_delay);
  FOR ALL : n3 USE ENTITY gates.nand3_q (average_delay);
  FOR ALL : n4 USE ENTITY gates.nand4_q (average_delay);
  SIGNAL im1,im2,im3,im4,im5,im6,im7,im8,im9,im10 : qit;
BEGIN
  g0 : inv PORT MAP (a, im1);
  g1 : inv PORT MAP (b, im2);
  g2 : inv PORT MAP (cin, im3);
  -- sum output
  g3 : n3 PORT MAP (im1, im2, cin, im4);
  g4 : n3 PORT MAP (im1, b, im3, im5);
  g5 : n3 PORT MAP (a, im2, im3, im6);
  g6 : n3 PORT MAP (a, b, cin, im7);
  g7 : n4 PORT MAP (im4, im5, im6, im7, sum);
  -- carry output
  g8 : n2 PORT MAP (a, b, im8);
  g9 : n2 PORT MAP (a, cin, im9);
  g10 : n2 PORT MAP (b, cin, im10);
  g11 : n3 PORT MAP (im8, im9, im10, carry);
END gate_level;

```

รูปที่ 4.49 แสดงการบรรยายการทำงานของวงจรบวก

รูปที่ 4.50 แสดงการบรรยายการทำงานของวงจรบวกเลขขนาด n บิต โดยการเปลี่ยนค่า n ตามจำนวนบิตที่ต้องการ



```

LIBRARY util;
LIBRARY reg;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE reg.register_logic.ALL;
ENTITY 8_bit_adder IS
  PORT ( a, b : IN qit_vector(7 DOWNTO 0); cin : IN qit;
        sum : OUT qit_vector(7 DOWNTO 0); cout : OUT qit);
END 8_bit_adder;
ARCHITECTURE structural OF 8_bit_adder IS

```

```

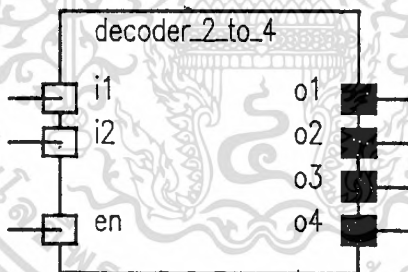
COMPONENT fadd
  PORT ( a, b, cin : IN qit;  sum, carry : OUT qit);
END COMPONENT;

FOR ALL : fadd USE ENTITY reg.full_adder (gate_level);
CONSTANT n : INTEGER := 8;
SIGNAL im : qit_vector (0 TO (n-2));
BEGIN
  c_all : FOR i IN 0 TO n-1 GENERATE
    l : IF i = 0 GENERATE
      least : fadd PORT MAP (a(i), b(i), cin, sum(i), im(i));
    END GENERATE;
    m : IF i = n-1 GENERATE
      most : fadd PORT MAP (a(i), b(i), im(i-1), sum(i), cout);
    END GENERATE;
    r : IF i > 0 AND i < n-1 GENERATE
      rest : fadd PORT MAP (a(i), b(i), im(i-1), sum(i), im(i));
    END GENERATE;
  END GENERATE;
END structural;

```

รูปที่ 4.50 แสดงการบรรยายการทำงานของวงจรบวกเลขขนาด n บิต

รูปที่ 4.51-4.53 แสดงการบรรยายการทำงานของ 2_to_4, 4_to_16 และ 8_to_256_decoder



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY decoder_2_to_4 IS
  PORT ( i2 : IN qit_vector (1 DOWNT0 0); en : IN qit;
        o4 : OUT qit_vector (0 TO 3));
END decoder_2_to_4;

ARCHITECTURE structural OF decoder_2_to_4 IS
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
  SIGNAL in0, in1 : qit;

```

```

BEGIN
  g0 : inv PORT MAP (i2(1), in1);
  g1 : inv PORT MAP (i2(0), in0);
  g2 : a3 PORT MAP (en, in1, in0, o(0));
  g3 : a3 PORT MAP (en, in1, i2(0), o(1));
  g4 : a3 PORT MAP (en, i2(1), in0, o(2));
  g5 : a3 PORT MAP (en, i2(1), i2(0), o(3));
END structural;

```

รูปที่ 4.51 แสดงการบรรยายการทำงานของ 2_to_4_decoder

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY decoder_4_to_16 IS
  PORT ( i4 : IN nibble; en : IN qit;
         o16 : OUT qit_vector (0 TO 15));
END decoder_4_to_16;
ARCHITECTURE structural OF decoder_4_to_16 IS
  COMPONENT d2_4
    PORT ( i2 : IN qit_vector (1 DOWNT0 0); en : IN qit;
          o4 : OUT qit_vector (0 TO 3));
  END COMPONENT;
  FOR ALL : d2_4 USE ENTITY reg.decoder_2_to_4 (structural);
  SIGNAL im : qit_vector (0 TO 3);
BEGIN
  c0 : d2_4 PORT MAP (i4(3 DOWNT0 2), en, im);
  c1_to_4 : FOR k IN 1 TO 4 GENERATE
    c : d2_4 PORT MAP (i4(1 DOWNT0 0), im(k-1),
                      o16((k-1)*4 TO (k-1)*4+3));
  END GENERATE;
END structural;

```

รูปที่ 4.52 แสดงการบรรยายการทำงานของ 4_to_16_decoder

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY decoder_8_to_256 IS
  PORT ( i8 : IN byte; en : IN qit;
         o : OUT qit_vector (0 TO 255));
END decoder_8_to_256;

```

ARCHITECTURE structural OF decoder_8_to_256 IS

```

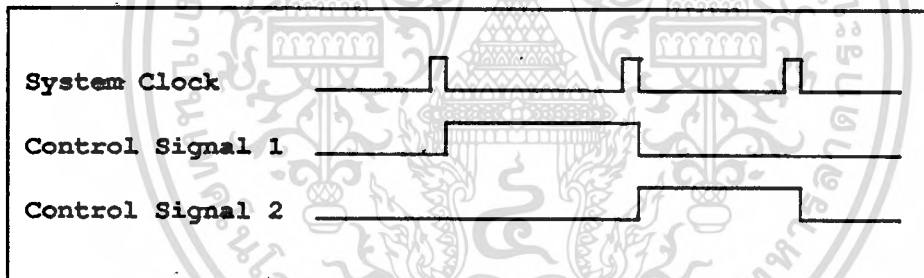
COMPONENT d4_16
  PORT ( i4 : IN nibble; en : IN qit;
         o16 : OUT qit_vector (0 TO 15));
END COMPONENT;
FOR ALL : d4_16 USE ENTITY reg.decoder_4_to_16 (structural);
SIGNAL im : qit_vector (0 TO 15);
BEGIN
  c0 : d4_16 PORT MAP (i8(7 DOWNT0 4), en, im);
  c1_to_16 : FOR k IN 1 TO 16 GENERATE
    c : d4_16 PORT MAP (i8(3 DOWNT0 0), im(k-1),
                       o((k-1)*16 TO (k-1)*16+15));
  END GENERATE;
END structural;

```

รูปที่ 4.53 แสดงการบรรยายการทำงานของ 8_to_256_decoder

4.1.1.4 กำหนดเวลาของข้อมูลและเหตุการณ์(Timing of Data and Control Events)

ในส่วนของวงจรเก็บข้อมูลและวงจรควบคุมจะถูกกระตุ้นโดยสัญญาณนาฬิกาเดียวกันที่สัญญาณขาของ clock สัญญาณที่ออกมาจากวงจรควบคุมจะเปลี่ยนสถานะในขณะที่รีจิสเตอร์ก็จะรับข้อมูลใหม่เข้ามา รูปที่ 4.54 แสดงความสัมพันธ์ของสัญญาณ clock และสัญญาณควบคุม ในส่วนของวงจรควบคุมจะใช้ *master-slave-D-flipflop* เป็นตัวรับและส่งสัญญาณ



รูปที่ 4.54 แสดงความสัมพันธ์ของสัญญาณ clock และสัญญาณควบคุม

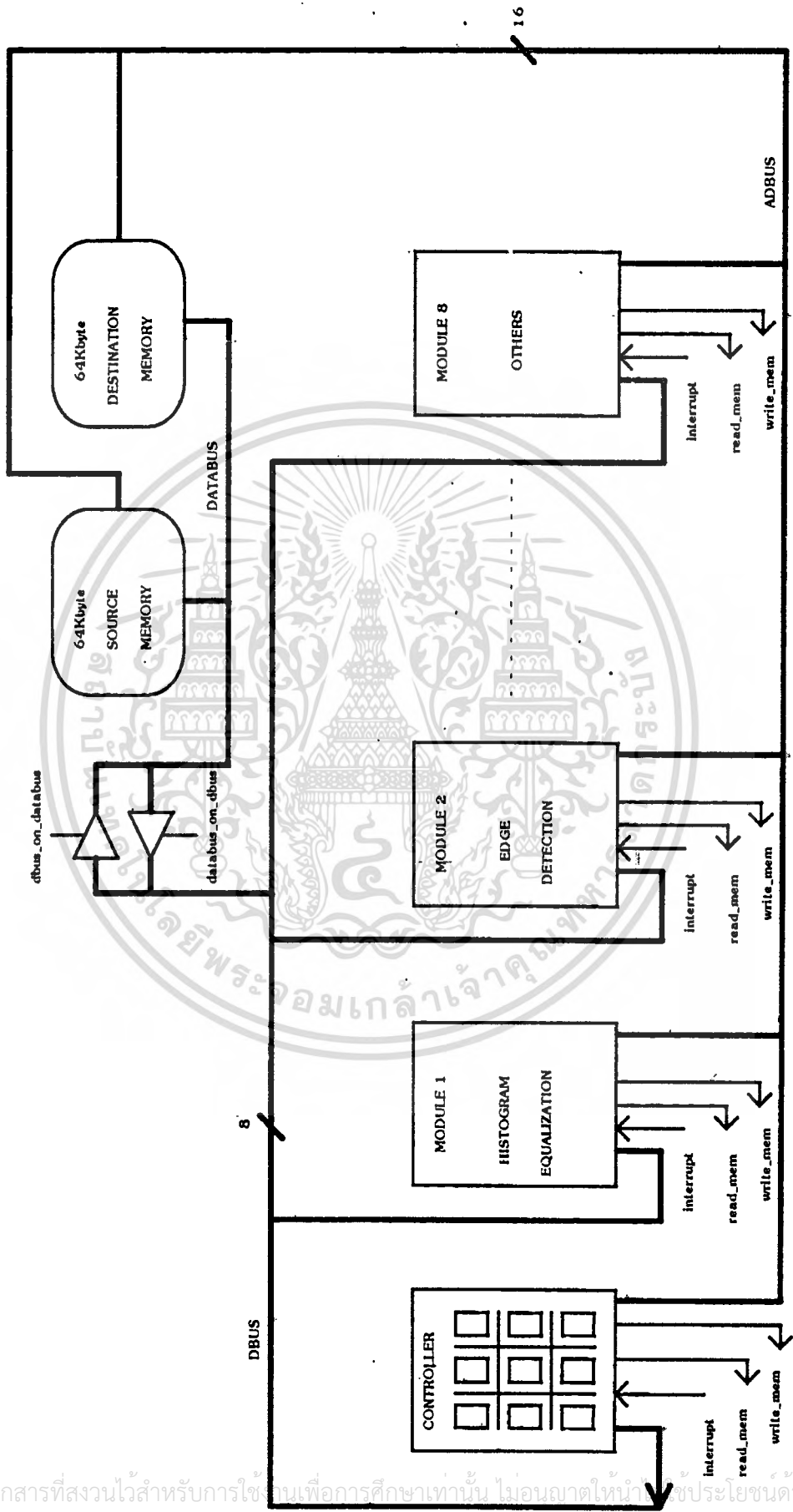
4.1.2 โครงสร้างของระบบ (Structural of System)

หลังจากที่ได้กำหนดลักษณะทางพฤติกรรมของระบบเรียบร้อยแล้ว ขั้นตอนต่อไปจะเป็นการออกแบบโครงสร้างการเชื่อมต่อระหว่างโมดูลต่าง ๆ ในหัวข้อต่อไปจะได้แสดงไต่อะแกรมของโครงสร้างระบบที่ได้ออกแบบไว้ ซึ่งไต่อะแกรมนี้จะมีประโยชน์ในการนำมาพัฒนาเพื่อใช้ในการบรรยายการทำงานของระบบต่อไป

4.1.2.2 การเชื่อมต่อระหว่างอุปกรณ์ (Interconnection of Components)

องค์ประกอบหลักของระบบประมวลผลภาพดิจิทัลจะประกอบด้วย MAIN MODULE ซึ่งทำหน้าที่เป็นตัวควบคุมหลัก(CONTROLLER)ของระบบ ส่วนของวงจรประมวลผลภาพที่ออกแบบไว้มี 2 ส่วนคือ MODULE 1 ซึ่งเป็นวงจรปรับปรุงคุณภาพของภาพและ MODULE 2 เป็นวงจรตรวจหาขอบวัตถุของภาพ องค์ประกอบทั้งหมดเชื่อมต่อกันด้วยบัสและเส้นวงจร(hard_wired)ดังรูปที่ 4.55 DBUS และ DATABUS จะเชื่อมต่อกันได้ก็ต่อเมื่อมีสัญญาณ dbus_on_databus หรือ databus_on_dbus เท่านั้น

4.1.2.1 โครงสร้างบัส (System Bussing Structure)



รูปที่ 4.55 แสดงโครงสร้างโปรแกรมสร้างบัสของระบบ

4.1.2.3 การบรรยายของระบบ (Description of System)

เมื่อได้ออกแบบโครงสร้างบัสของระบบเสร็จแล้วก็นำมาใช้ในการบรรยายการเชื่อมต่อในรูปแบบของ VHDL ดังรูปที่ 4.56 ระบบนี้ให้ชื่อว่า **modular** การติดต่อกับภายนอกของระบบประกอบด้วยอินพุต *clk* รับสัญญาณนาฬิกา อินพุต *interrupt* เพื่อเริ่มการทำงานของระบบ อินพุต *instruct* เป็นเส้นทางสำหรับรับคำสั่งต่าง ๆ และเอาต์พุต *read_inst* เมื่อปฏิบัติแต่ละคำสั่งเรียบร้อยแล้วและพร้อมที่จะรับคำสั่งใหม่ต่อไป การบรรยายเป็นในระดับโครงสร้าง สำหรับการบรรยายการทำงานของแต่ละโมดูลจะได้กล่าวถึงในบทต่อไป

```

LIBRARY util;
USE util.basic.utilities.ALL;
USE util.image.utilities.ALL;
USE STD.TEXTIO.ALL;

ENTITY modular IS
    GENERIC ( read_high_time, read_low_time,
              write_high_time, write_low_time : TIME := 2 US);
    PORT ( clk : IN qit;
           load_image, write_image : IN qit;
           interrupt : IN qit; -- Signal to start controller module
           instruct : IN byte; -- instruction
           load_ready : OUT qit;
           read_inst : OUT qit); -- Signal to fetch next instruction
END modular;

ARCHITECTURE structural OF modular IS
    COMPONENT memo
        PORT ( clk, read, write,
              clear, enable : IN qit;
              abus : IN word;
              dbus : INOUT wired_byte BUS);
    END COMPONENT;

    COMPONENT ctrl
        PORT ( clk, interrupt, mod_end : IN qit;
              instruct : IN byte;
              clear_mem, read_mem, write_mem : OUT qit;
              dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
              abus : OUT word;
              read_instruction : OUT qit;
              en_sm, en_dm : OUT qit;
              m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
              r_c : OUT qit;
              r_c_data : OUT word;
              mod_begin : OUT qit;
              dbus_on_databus, databus_on_dbus : OUT qit);
    END COMPONENT;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

COMPONENT module

```

PORT ( clk, interrupt, r_c, mod_begin : IN qit;
      r_c_data : IN word;
      dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
      read_mem, write_mem : OUT qit;
      en_sm, en_dm, mod_end : OUT qit;
      abus : OUT word;
      dbus_on_databus, databus_on_dbus : OUT qit);

```

```

END COMPONENT;

```

```

SIGNAL mod_end : qit;
SIGNAL clear, sread, swrite : qit;
SIGNAL data : wired_byte BUS := "ZZZZZZZZ";
SIGNAL address : word;
SIGNAL read_inst : qit;
SIGNAL en_sm, en_dm : qit;
SIGNAL m1, m2, m3, m4, m5, m6, m7, m8 : qit;
SIGNAL r_c : qit;
SIGNAL r_c_data : word;
SIGNAL mod_begin : qit;
SIGNAL dbus_on_databus, databus_on_dbus : qit;
SIGNAL load_image, write_image, load_ready : qit;

```

```

BEGIN

```

```

  source_memory : memo
    PORT MAP ( clk, sread, swrite, clear, en_sm, address, data);
  dest_memory : memo
    PORT MAP ( clk, sread, swrite, clear, en_dm, address, data);
  ctrl_module : ctrl
    PORT MAP ( clk, interrupt, mod_end, instruct,
              clear, sread, swrite, data, address,
              read_inst, en_sm, en_dm,
              m1, m2, m3, m4, m5, m6, m7, m8,
              r_c, r_c_data, mod_begin,
              dbus_on_databus, databus_on_dbus);

```

```

  histogram_module : module

```

```

    PORT MAP ( clk, interrupt, r_c, mod_begin,
              r_c_data, data, sread, swrite,
              en_sm, en_dm, mod_end, address,
              dbus_on_databus, databus_on_dbus);

```

```

  edge_module : module

```

```

    PORT MAP ( clk, interrupt, r_c, mod_begin,
              r_c_data, data, sread, swrite,
              en_sm, en_dm, mod_end, address,
              dbus_on_databus, databus_on_dbus);

```

```
-- read image data from file and write to memory
```

```
read : PROCESS (load_image)
```

```
FILE source : TEXT IS IN "input.dat";
```

```
VARIABLE sl : LINE;
```

```
VARIABLE page, offset : byte;
```

```
VARIABLE dt : INTEGER;
```

```
VARIABLE ram_init : BOOLEAN := FALSE;
```

```
VARIABLE i, j : INTEGER := 0;
```

```
BEGIN
```

```
IF ram_init = FALSE THEN
```

```
WHILE NOT ENDFILE (source) AND (i < 256) LOOP
```

```
READLINE (source, sl);
```

```
WHILE (j < 256) LOOP
```

```
READ (sl, dt);
```

```
page <= int2qit(i);
```

```
offset <= int2qit(j);
```

```
en_sm <= '1';
```

```
en_dm <= '1';
```

```
address <= page & offset;
```

```
data <= int2qit(dt);
```

```
swrite <= '1'; WAIT FOR write_high_time;
```

```
swrite <= '0'; WAIT FOR write_low_time;
```

```
data <= "ZZZZZZZZ";
```

```
j := j + 1;
```

```
END LOOP;
```

```
j := 0;
```

```
i := i + 1;
```

```
END LOOP;
```

```
ram_init := TRUE;
```

```
END IF;
```

```
load_ready <= '1';
```

```
END PROCESS read;
```

```
-- read image data from memory and write to file
```

```
write : PROCESS
```

```
FILE dest : TEXT IS OUT "output.dat";
```

```
VARIABLE dl : LINE;
```

```
VARIABLE page, offset, dt : byte;
```

```
VARIABLE dd : INTEGER;
```

```
VARIABLE i, j : INTEGER := 0;
```

```
BEGIN
```

```
WAIT UNTIL write_image = '1';
```

```
i := 0;
```

```
j := 0;
```

```
WHILE i < 256 LOOP
```

```
WHILE j < 256 LOOP
```

```
page <= int2qit(i);
```

เอกสารนี้เป็นเอกสารของสำนักงานคณะกรรมการกฤษฎีกา กระทรวงยุติธรรม ห้ามเผยแพร่หรือคัดลอกข้อมูลในส่วนนี้โดยไม่ได้รับอนุญาตจากสำนักงานกฤษฎีกา

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

offset <= int2qit(j);
en_sm <= '0';
en_dm <= '1';
address <= page & offset;
sread <= '1'; WAIT FOR read_high_time;
dt <= byte(data);
sread <= '0'; WAIT FOR read_low_time;
dd <= qit2int(dt);
WRITE (dl, dd);
j := j + 1;
END LOOP;
Writeln (dest, dl);
j := 0;
i := i + 1;
END LOOP;
END PROCESS write;
END structural;

```

รูปที่ 4.56 แสดงการบรรยายระดับโครงสร้างของระบบ

โปรเซส *read* เป็นโปรเซสสำหรับการอ่านข้อมูลภาพจากไฟล์เข้าสู่หน่วยความจำต้นทาง
 โปรเซส *write* เป็นโปรเซสสำหรับการเขียนข้อมูลภาพจากหน่วยความจำปลายทางลงไฟล์ แบบทดสอบนี้
 ใช้กับข้อมูลภาพที่มีขนาด 256X256 จุดภาพ

4.2 การจัดระบบหน่วยความจำ (Memory Organization)

หน่วยความจำของระบบประกอบด้วย 2 ส่วนคือหน่วยความจำต้นทางสำหรับเก็บ
 ข้อมูลของภาพต้นแบบและหน่วยความจำปลายทางสำหรับเก็บข้อมูลภาพที่ผ่านการประมวลผลแล้ว
 หน่วยความจำทั้ง 2 ส่วนมีขนาด 64 Kbytes การอ้างอิงตำแหน่งแอดเดรสโดยผ่านบัสขนาด 16 บิต

4.2.1 พฤติกรรมของวงจรหน่วยความจำ (Behavioral of Memory)

หน่วยความจำถูกแบ่งเป็น 256 เพจ(pages) ในแต่ละเพจจะมี 256 ไบต์ดังรูปที่ 4.57
 8-bit สูงจะใช้ในการอ้างอิงตำแหน่งของเพจและ 8-bit ต่ำจะใช้ในการอ้างอิงตำแหน่งของออฟเซต(offset)

ADDRESS:

1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
Page								Offset							

MEMORY:							
	7	6	5	4	3	2	1 0
0:00 - 0:FF	page 0 ...						
1:00 - 1:FF	page 1 ...						
2:00 - 2:FF	page 2 ...						
.							
.							
.							
FE:00 - FEFF	page 254 ...						
FF:00 - FFFF	page 255 ...						

รูปที่ 4.57 แสดงการอ้างอิงตำแหน่งของหน่วยความจำ

4.2.1.1 การบรรยายการเชื่อมต่อของหน่วยความจำ
(Interface Description of Memory)

การบรรยายส่วนติดต่อภายนอกอินพุตและเอาต์พุตของหน่วยความจำดังรูปที่ 4.58 ได้กำหนดชนิดของข้อมูลเป็น *qit*, *word* และ *wired_byte* ที่กำหนดไว้ในแพดเกจ *basic_utilities* และ *image_utilities* *abus* เป็นอินพุตเพื่อรับค่าตำแหน่งแอดเดรส *dbus* เป็นอินพุตเอาต์พุตเพื่อรับและส่งข้อมูล

```
LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
ENTITY memory IS
  PORT ( clk, read, write, clear, enable : IN qit;
        abus : IN word;
        dbus : INOUT wired_byte BUS);
END memory;
```

รูปที่ 4.58 แสดงการบรรยายการเชื่อมต่อของหน่วยความจำ

4.2.1.2 การบรรยายการทำงานระดับพฤติกรรมของหน่วยความจำ
(Memory Behavioral Architecture)

รูปที่ 4.59 แสดงการบรรยายพฤติกรรมของหน่วยความจำ ในการบรรยายได้กำหนดอะเรย์ขนาด 256X256 ขึ้นมาเพื่อใช้แทนหน่วยเก็บข้อมูล การทำงานจะเริ่มเมื่อมีสัญญาณ *read*, *write* หรือ *clear* เกิดขึ้น เนื่องจากในการอ้างอิงตำแหน่งของอะเรย์ต้องเป็นเลขจำนวนเต็มจึงต้องแปลงแอดเดรสจากชนิด *qit* เป็น *INTEGER* เสียก่อนโดยใช้ไพร์ซีเจอร์ *qit2int* การอ่านหรือเขียนข้อมูลลงหน่วยความจำกระทำเมื่อสัญญาณ *enable* เป็น '1' เท่านั้น

```
ARCHITECTURE`behavioral OF memory IS
  TYPE byte_memory IS ARRAY (0 TO 255, 0 TO 255) OF byte;
BEGIN
  PROCESS
    VARIABLE mem : byte_memory;
```

```

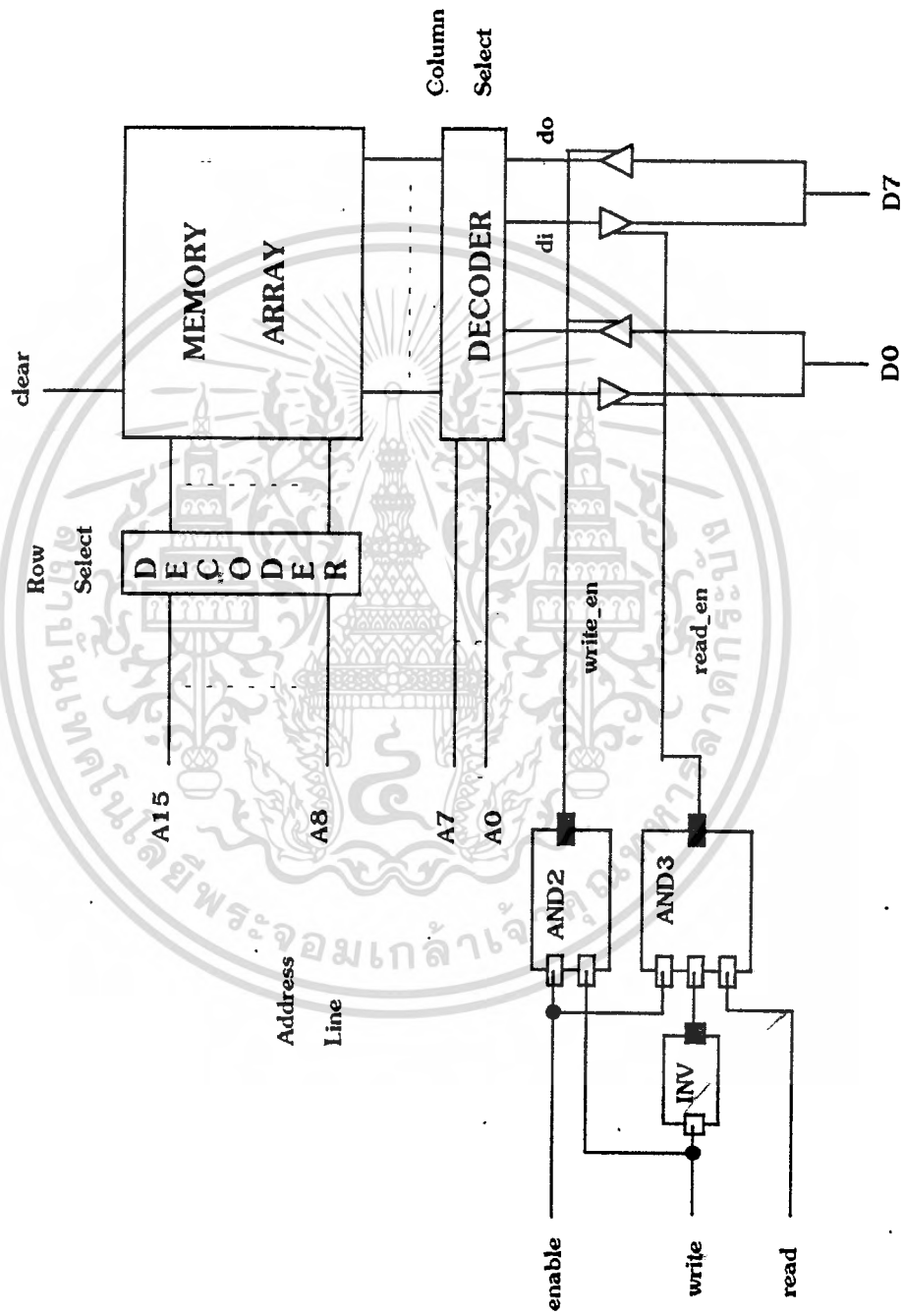
VARIABLE ia, ib : INTEGER;
VARIABLE page, offset : byte;
BEGIN
  WAIT ON read, write, clear;
  page := abus (15 DOWNT0 8);
  offset := abus (7 DOWNT0 0);
  qit2int (page, ia);
  qit2int (offset, ib);
  IF (enable = '1') THEN
    IF read = '1' THEN
      IF (ia >= 256) OR (ib >= 256) THEN
        dbus <= "ZZZZZZZZ";
      ELSE dbus <= wired_byte (mem(ia, ib));
      END IF;
      WAIT UNTIL read = '0';
      dbus <= "ZZZZZZZZ";
    ELSIF write = '1' THEN
      IF (ia < 256) AND (ib < 256) THEN
        mem (ia, ib) := byte(dbus);
      END IF;
      WAIT UNTIL write = '0';
    ELSIF clear = '1' THEN
      FOR i IN 0 TO 255 LOOP
        FOR j IN 0 TO 255 LOOP
          mem(i,j) := "00000000";
        END LOOP;
      END LOOP;
    END IF;
    ELSE dbus <= "ZZZZZZZZ";
    END IF;
  END PROCESS;
END behavioral;

```

รูปที่ 4.59 แสดงการบรรยายพฤติกรรมของหน่วยความจำ

4.2.2 โครงสร้างของวงจรหน่วยความจำ (Structural of Memory)

รูปที่ 4.60 แสดงโครงสร้างและอุปกรณ์ที่เป็นส่วนประกอบของหน่วยความจำ



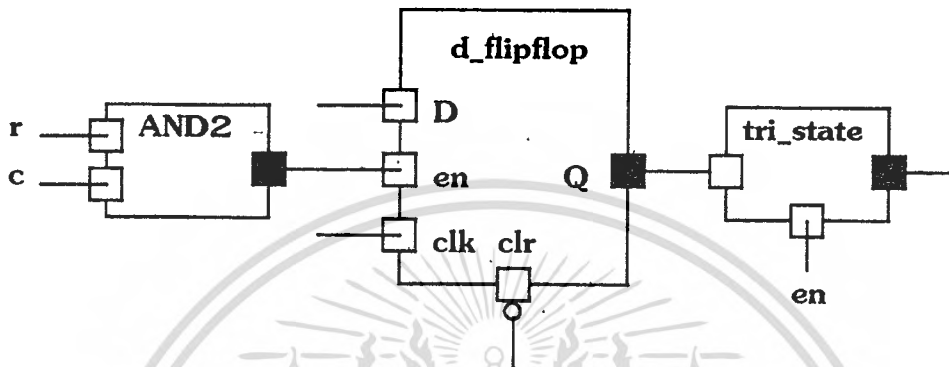
รูปที่ 4.60 แสดงโครงสร้างของหน่วยความจำ

4.2.3 การบรรยายการทำงานขององค์ประกอบ (Description of Components)

วงจรหน่วยความจำมีองค์ประกอบที่สำคัญ 2 ส่วนคือ ส่วนความจำเก็บข้อมูลซึ่งเป็นอะเรย์ขนาด 256X256 ไบต์ และส่วน DECODER สำหรับกำหนดตำแหน่งของอะเรย์หน่วยความจำ

4.2.3.1 Byte_Memory

Byte_Memory เป็นหน่วยความจำสำหรับเก็บข้อมูลขนาด 1 ไบต์และใช้เป็นองค์ประกอบสำหรับอะเรย์ความจำอีกทีหนึ่ง ในการเก็บข้อมูลใช้รีจิสเตอร์ DFF จำนวน 8 ตัว ดังรูปที่ 4.61



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY byte_memory IS
  PORT ( i8 : IN byte;
         clk, clr, r, c : IN qit;
         o8 : OUT byte);
END byte_memory;

ARCHITECTURE structural OF byte_memory IS
  FOR ALL : df USE ENTITY gates.dff(dataflow);
  FOR ALL : a2 USE ENTITY gates.and2_q(double_delay);
  FOR ALL : ts USE ENTITY gates.tri_state_q(single_delay);
  SIGNAL rc : qit;
  SIGNAL im : byte;
BEGIN
  g0 : a2 PORT MAP (r, c, rc);
  c0 : df PORT MAP (i8(0), rc, clk, clr, im(0));
  c1 : df PORT MAP (i8(1), rc, clk, clr, im(1));
  c2 : df PORT MAP (i8(2), rc, clk, clr, im(2));
  c3 : df PORT MAP (i8(3), rc, clk, clr, im(3));
  c4 : df PORT MAP (i8(4), rc, clk, clr, im(4));
  c5 : df PORT MAP (i8(5), rc, clk, clr, im(5));
  c6 : df PORT MAP (i8(6), rc, clk, clr, im(6));
  c7 : df PORT MAP (i8(7), rc, clk, clr, im(7));
  g1 : ts PORT MAP (im(0), rc, o8(0));

```

```

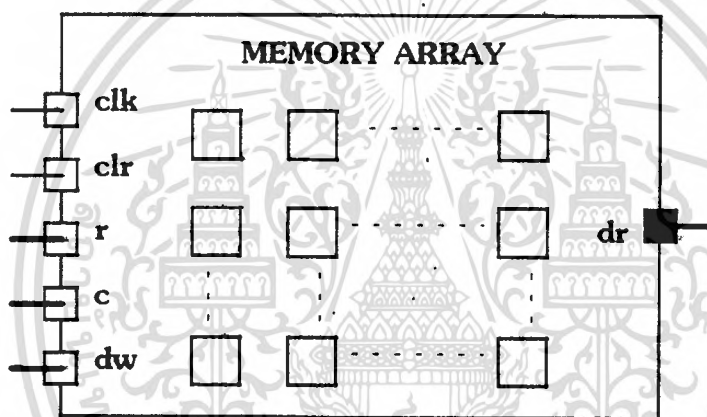
g2 : ts PORT MAP (im(1), rc, o8(1));
g3 : ts PORT MAP (im(2), rc, o8(2));
g4 : ts PORT MAP (im(3), rc, o8(3));
g5 : ts PORT MAP (im(4), rc, o8(4));
g6 : ts PORT MAP (im(5), rc, o8(5));
g7 : ts PORT MAP (im(6), rc, o8(6));
g8 : ts PORT MAP (im(7), rc, o8(7));
END structural;

```

รูปที่ 4.61 แสดงการบรรยายหน่วยความจำขนาด 1 ไบต์

4.2.3.2 Memory_Array

Memory_Array เป็นหน่วยความจำอะเรย์ขนาด 256X256 ไบต์ การบรรยายโครงสร้างใช้หน่วยความจำขนาด 1 ไบต์ของ *Byte_Memory* มาเป็นส่วนประกอบ เนื่องจากเป็นการอ้างอิงอุปกรณ์ชนิดเดียวกันซ้ำกันจึงใช้รูปแบบการบรรยายแบบ *iterative* ดังรูปที่ 4.62



```

LIBRARY util;
LIBRARY memo;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

```

```

ENTITY memory_array IS
  PORT ( clk, clr : IN qit;
         r, c : IN qit_vector(0 TO 255);
         dw : IN byte;
         dr : OUT byte);
END memory_array;

```

```

ARCHITECTURE structural OF memory_array IS
  COMPONENT reg PORT (i8 : IN byte;
                       clk, dr, r, c : IN qit;
                       o8 : OUT byte);
  END COMPONENT;
  FOR ALL : reg USE ENTITY memo.byte_memory(structural);

```

```

BEGIN
  row : FOR i IN 0 TO 255 GENERATE
    col : FOR j IN 0 TO 255 GENERATE
      mem : reg PORT MAP (dw, clk, clr, r(i), c(j), dr);
    END GENERATE;
  END GENERATE;
END structural;

```

รูปที่ 4.62 แสดงการบรรยายโครงสร้างของหน่วยความจำอะเรย์ขนาด 256X256 ไบต์

4.2.3.3 Select_Memory

Select_Memory เป็น DECODER ทำหน้าที่แปลงข้อมูลแอดเดรสที่ได้รับเป็นสัญญาณเพื่อเลือก row และ column สำหรับการอ่านและเขียนข้อมูลในหน่วยความจำ รูปที่ 4.63 แสดงการบรรยายการทำงานในระดับข้อมูลของ *Select_Memory* ซึ่งจะให้สัญญาณออกมาเมื่ออินพุต *enable* เป็น '1'

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY select_memory IS
  PORT ( addr : IN byte; en : IN qit;
        sel : OUT qit_vector(0 TO 255));
END select_memory;

ARCHITECTURE dataflow OF select_memory IS
BEGIN
  BLOCK (en = '1')
  BEGIN
    WITH addr SELECT
      sel(0) <= GUARDED '1' WHEN '00000000';
      sel(1) <= GUARDED '1' WHEN '00000001';
      sel(2) <= GUARDED '1' WHEN '00000010';
      .
      .
      .
      .
      .
      sel(254) <= GUARDED '1' WHEN '11111110';
      sel(255) <= GUARDED '1' WHEN '11111111';
    END BLOCK;
  END dataflow; -

```

รูปที่ 4.63 แสดงการบรรยายการทำงานในระดับข้อมูลของ *Select_Memory*

ในกรณีของการบรรยายการทำงานในระดับโครงสร้างของ *Select_Memory* สามารถใช้ วงจร decoder_8_to_256 ที่ได้กำหนดไว้ในแพตเทจ *Register_Logic* ดังรูปที่ 4.53 แทนได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.2.4 การบรรยายการทำงานระดับโครงสร้างของหน่วยความจำ

(Memory Structural Architecture)

รูปที่ 4.64 แสดงการบรรยายการทำงานระดับโครงสร้างของหน่วยความจำโดยใช้องค์ประกอบที่ได้ออกแบบไว้ข้างต้นมาประกอบกันตามรูปไดอะแกรม 4.60

```

LIBRARY util;
LIBRARY gates;
LIBRARY memo;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE gates.gates_flipflop.ALL;

ENTITY memory IS
    PORT ( clk, read, write, clear, enable : IN qit;
          abus : IN word;
          dbus : INOUT wired_byte BUS);
END memory;

ARCHITECTURE structural OF memory IS
    COMPONENT s1 PORT (addr : IN byte; en : IN qit;
                       sel : OUT qit_vector(0 TO 255));
    END COMPONENT;
    COMPONENT ma PORT (clk, clr : IN qit;
                       r, c : IN qit_vector(0 TO 255);
                       dw : IN byte;
                       dr : OUT byte);
    END COMPONENT;

    FOR ALL : s1 USE ENTITY memo.select_memory(dataflow);
    FOR ALL : ma USE ENTITY memo.memory_array(structural);
    FOR ALL : inv USE ENTITY gates.inv_q(double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q(double_delay);
    FOR ALL : a3 USE ENTITY gates.and3_q(average_delay);
    FOR ALL : ts USE ENTITY gates.tri_state_q(single_delay);

    SIGNAL rs, cs : qit_vector(0 TO 255);
    SIGNAL in1, read_en, write_en : qit;
    SIGNAL di, do : byte;
    SIGNAL page, offset : byte;

BEGIN
    page <= abus(15 DOWNTO 8);
    offset <= abus(7 DOWNTO 0);
    c0 : s1 PORT MAP (page, enable, rs);
    c1 : s1 PORT MAP (offset, enable, cs);
    c2 : inv PORT MAP (write, in1);
    c3 : a2 PORT MAP (enable, write, write_en);
    c4 : a3 PORT MAP (enable, read, in1, read_en);
    c5to12 : FOR i IN 0 TO 7 GENERATE

```

```

cw : ts PORT MAP (dbus(i), write_en, do(i));
END GENERATE;
c13to20 : FOR j IN 0 TO 7 GENERATE
  cr : ts PORT MAP (di(j), read_en, dbus(j));
END GENERATE;
c21 : ma PORT MAP (clk, clear, rs, cs, do, di);
END structural;

```

รูปที่ 4.64 แสดงการบรรยายการทำงานระดับโครงสร้างของหน่วยความจำ

4.3 สรุป (Summary)

ในบทนี้ได้กล่าวถึงลักษณะโครงสร้างและสถาปัตยกรรมของระบบและหน่วยความจำ รวมถึงแพดเกจ ยูทิลิตี้ เกจพื้นฐาน ฟลิปฟล็อป รีจิสเตอร์และลอจิกที่จำเป็นต้องใช้ในการออกแบบทั้งหมด การกำหนดลักษณะทางพฤติกรรมและการออกแบบโครงสร้างการเชื่อมต่อภายในระบบทำให้สามารถมองเห็นภาพโดยรวมของระบบทั้งหมดได้ชัดเจนยิ่งขึ้น

การบรรยายพฤติกรรมการทำงานจนถึงการบรรยายเชิงโครงสร้างของหน่วยความจำ การบรรยายองค์ประกอบย่อยตั้งแต่ระดับพื้นฐานจนถึงระดับรวม เหล่านี้แสดงให้เห็นถึงลำดับขั้นตอนในการพัฒนาตัววงจรตั้งแต่การกำหนดแนวความคิดในเชิงพฤติกรรมจนถึงการกำหนดโครงสร้างรวมในเชิงฮาร์ดแวร์ที่จะสามารถนำไปใช้ประโยชน์ได้จริง สำหรับลักษณะการออกแบบของแต่ละโมดูลจะได้แยกกล่าวถึงในบทต่อไป

บทที่ 5

สถาปัตยกรรมของวงจรควบคุม (Controller Architecture)

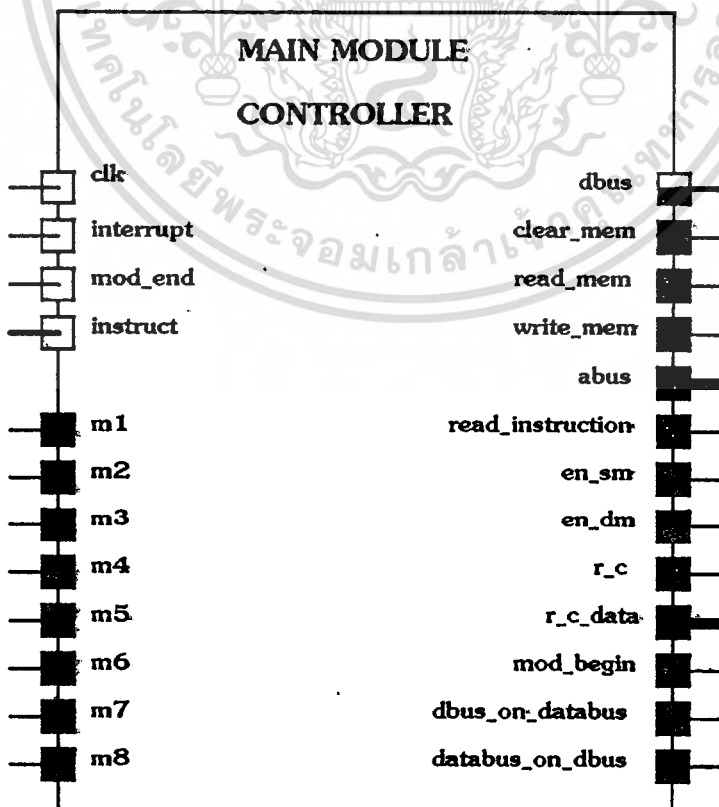
วงจรควบคุมเป็นโมดูลหลักของระบบทำหน้าที่ควบคุมในการประสานงานระหว่างโมดูลต่าง ๆ และหน่วยความจำ การควบคุมการทำงานของระบบสามารถกระทำผ่านทางวงจรควบคุมโดยคำสั่งต่าง ๆ ดังที่ได้แสดงไว้ในบทที่ 4

5.1 พฤติกรรมของวงจรควบคุม (Behavioral of Controller)

ในการออกแบบได้กำหนดพฤติกรรมการทำงานของวงจรควบคุมให้เป็นส่วนของการรับอินพุตซึ่งเป็นคำสั่งจากภายนอกแล้วนำมาปฏิบัติโดยส่งเป็นสัญญาณควบคุมไปยังส่วนต่าง ๆ ของระบบตามลำดับคำสั่งอีกทีหนึ่ง หัวข้อต่อไปจะแสดงให้เห็นถึงส่วนการติดต่อภายนอกและการบรรยายหน้าที่การทำงานของวงจรในระดับพฤติกรรมเพื่อเป็นการสอดประสานแนวความคิดในการออกแบบและลำดับการปฏิบัติให้สอดคล้องกันก่อนที่จะพัฒนาไปสู่การออกแบบในระดับโครงสร้างต่อไป

5.1.1 การบรรยายการเชื่อมต่อของวงจรควบคุม (Interface Description of Controller)

รูปที่ 5.1 แสดงสัญลักษณ์และรูปแบบการบรรยายส่วนเชื่อมต่อของวงจรควบคุมหลัก ส่วนของ *GENERIC* กำหนดพารามิเตอร์เพื่อควบคุมเวลาที่ใช้ในการอ่านและเขียนหน่วยความจำ ส่วนของ *PORT* เป็นส่วนที่ใช้กำหนดอินพุตและเอาต์พุตของวงจร บัส *dbus* เป็นบัสชนิด 2 กิตทางซึ่งเป็นสัญญาณแบบที่ผ่านการเลือกสรรข้อมูลโดยใช้ฟังก์ชัน *wiring* ที่อยู่ในแพ็คเกจ *basic_utilities* การกำหนดในลักษณะนี้ทำให้ *dbus* สามารถรับและส่งข้อมูลที่มาจากหลายอินพุตได้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;

ENTITY controller IS
  GENERIC ( read_high_time, read_low_time,
            write_high_time, write_low_time : TIME := 2 US;
            cycle_time : TIME := 4 US);
  PORT ( clk, interrupt, mod_end : IN qit;
        instruct : IN byte;
        clear_mem, read_mem, write_mem : OUT qit;
        dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
        abus : OUT word;
        read_instruction : OUT qit;
        en_sm, en_dm : OUT qit;
        m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
        r_c : OUT qit;
        r_c_data : OUT word;
        mod_begin : OUT qit;
        dbus_on_databus, databus_on_dbus : OUT qit);
END controller;

```

รูปที่ 5.1 แสดงสัญลักษณ์และรูปแบบการบรรยายส่วนเชื่อมต่อของวงจรควบคุมหลัก

5.1.2 การบรรยายการทำงานระดับพฤติกรรมของวงจรควบคุม (Controller Behavioral Architecture)

รูปที่ 5.2 แสดงโครงสร้างของการบรรยายพฤติกรรมของวงจรควบคุม การบรรยายประกอบด้วยโครงสร้างแบบไพเรสที่ใช้ IF-ELSE ในการแยกหน้าที่การปฏิบัติของคำสั่งหรือกลุ่มคำสั่งออกจากกัน

```

ARCHITECTURE behavioral OF controller IS
BEGIN
  PROCESS
    Declare necessary variables; รูปที่ 5.3
  BEGIN
    IF interrupt = '1' THEN
      Handle interrupt; รูปที่ 5.4
    ELSIF (interrupt /= '1' OR mod_end = '1') THEN -- no interrupt
      Read first byte into byte1; รูปที่ 5.5
      IF byte1 (7 DOWNT0 4) = single_byte_instruction THEN
        Execute single-byte instruction; รูปที่ 5.6
      ELSE -- two-byte instructions
        Read second byte into byte2; รูปที่ 5.9
        IF byte1 (3 DOWNT0 0) = row THEN
          Keep the number of row to no_row; รูปที่ 5.10
        END IF
      END IF
    END IF
  END PROCESS
END behavioral;

```

```

ELSE
    . Keep the number of column to no_col; รูปที่ 5.10
END IF;
END IF;
END IF;
END PROCESS;
END behavioral;

```

รูปที่ 5.2 แสดงโครงสร้างของการบรรยายพฤติกรรมของวงจรควบคุม

```

VARIABLE page, offset : byte;
VARIABLE byte1, byte2 : byte;
VARIABLE no_row, no_col : byte;
VARIABLE buff : byte;

```

รูปที่ 5.3 แสดงการกำหนดตัวแปร

รูปที่ 5.4 แสดงลำดับการปฏิบัติเมื่ออินพุต *interrupt* เป็น '1' หลังจากเซตค่าของตัวแปรเป็นศูนย์แล้ว วงจรจะหยุดรอเป็นเวลา *cycle_time* ก่อนที่จะตรวจสอบค่าของ *interrupt* อีกครั้ง

```

page := zero_8;
offset := zero_8;
no_row := zero_8;
no_col := zero_8;
WAIT FOR cycle_time;

```

รูปที่ 5.4 แสดง Handle interrupt

เมื่อ *interrupt* เป็น '0' ไบต์แรกของคำสั่งจะถูกอ่านเข้ามาเก็บไว้ที่ตัวแปร *byte1* ดังรูปที่ 5.5 เอาท์พุท *read_instruction* จะถูกส่งออกมาเพื่อแจ้งให้รู้ว่าขณะนี้วงจรพร้อมที่จะรับคำสั่งต่อไปแล้ว การหยุดคอยเป็นเวลา *read_high_time* เพื่อให้มีเวลาเพียงพอต่อการตอบสนองของสัญญาณ

```

-- read first byte
read_instruction <= '1'; WAIT FOR read_high_time;
byte1 := instruct;
read_instruction <= '0'; WAIT FOR read_low_time;

```

รูปที่ 5.5 แสดงขั้นตอนการอ่านคำสั่ง

หลังจากอ่านไบต์แรกของคำสั่งเข้ามาแล้วก็ทำการตรวจสอบว่าเป็นคำสั่งชนิดหนึ่งหรือสองไบต์ ถ้าเป็นคำสั่งชนิดไบต์เดียวก็จะทำการตรวจสอบบิตที่ 3 ถึง 0 ว่าตรงกับคำสั่งใดก็จะปฏิบัติตามคำสั่งนั้น

คำสั่ง *clm* (clear memory) เป็นคำสั่งสำหรับลบข้อมูลในหน่วยความจำทั้งสองตัว การส่งค่า '1' ให้เอาท์พุท *en_sm* และ *en_dm* เพื่อบุญญาตให้สามารถกระทำการอ่านหรือเขียนกับหน่วยความจำต้นทางและปลายทางได้ ดังรูปที่ 5.6

```

CASE byte1 (3 DOWNT0 0) IS
  WHEN clm =>      -- clear both memory
    ASSERT
      (NOT (byte1 (3 DOWNT0 0) = clm))
    REPORT
      "Being clear memory"
    SEVERITY NOTE;
    en_sm <= '1';
    en_dm <= '1';
    clear_mem <= '1'; WAIT FOR cycle_time;
    clear_mem <= '0';
    en_sm <= '0';
    en_dm <= '0';

```

รูปที่ 5.6 แสดงขั้นตอนการปฏิบัติคำสั่ง *clm*

คำสั่ง *swm* (swap memory) เป็นคำสั่งสำหรับการย้ายข้อมูลภาพจากหน่วยความจำปลายทางไปยังหน่วยความจำต้นทาง เนื่องจากการทำงานของแต่ละโมดูลจะกระทำจากหน่วยความจำต้นทางไปยังหน่วยความจำปลายทางเสมอ ดังนั้นถ้าต้องการประมวลผลข้อมูลภาพต่อเนื่องกันไปจากโมดูลหนึ่งไปยังอีกโมดูลหนึ่งโดยใช้ข้อมูลภาพจากการประมวลผลอันก่อนจะต้องทำการย้ายข้อมูลจากหน่วยความจำปลายทางไปยังหน่วยความจำต้นทางเสียก่อน ลำดับการปฏิบัติจะประกอบด้วยโครงสร้างแบบลูปเพื่อย้ายข้อมูลภาพทีละตำแหน่ง ดังรูปที่ 5.7

```

WHEN swm =>      -- swap image data from destination to source memory
  ASSERT
    (NOT (byte1 (3 DOWNT0 0) = swm))
  REPORT
    "Being swap data in memory"
  SEVERITY NOTE;
  page := zero_8;
  offset := zero_8;
  WHILE page <= no_row LOOP
    WHILE offset <= no_col LOOP
      en_sm <= '0';
      en_dm <= '1';
      abus <= page & offset;
      read_mem <= '1'; WAIT FOR read_high_time;
      databus_on_dbus <= '1';
      buff := byte(dbus);
      read_mem <= '0'; WAIT FOR read_low_time;
      databus_on_dbus <= '0';
      en_sm <= '1';
      en_dm <= '0';
      dbus <= wired_byte(buff);
      dbus_on_databus <= '1';
      write_mem <= '1'; WAIT FOR write_high_time;
      write_mem <= '0'; WAIT FOR write_low_time;

```

เอกสารนี้เป็นเอกสารลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dbus_on_databus <= '0';
dbus.<= "ZZZZZZZZ";
offset := inc(offset);
END LOOP;
offset := zero_8;
page := inc(page);
END LOOP;
en_sm <= '0';

```

รูปที่ 5.7 แสดงขั้นตอนการปฏิบัติคำสั่ง *swm*

คำสั่ง **mod1 - mod8** เป็นคำสั่งสำหรับโอนย้ายการปฏิบัติงานให้กับโมดูลต่างๆชั่วคราวเพื่อให้แต่ละโมดูลปฏิบัติตามหน้าที่ที่ได้รับการออกแบบไว้จนกว่าแต่ละโมดูลจะปฏิบัติงานเสร็จจึงโอนมอบการควบคุมคืนแก่วงจรควบคุมอีกครั้งหนึ่ง ลำดับการปฏิบัติของแต่ละคำสั่งจะคล้ายคลึงกันเริ่มด้วยการส่งสัญญาณอนุญาต(m1 - m8)ให้ปฏิบัติงานได้แก่โมดูลที่ต้องการทราบและส่งข้อมูลแจ้งขนาดของภาพไปให้ สุดท้ายก็ส่งสัญญาณให้เริ่มปฏิบัติงานได้ ดังรูปที่ 5.8

```

WHEN mod1 => -- select module 1
  m1 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN mod2 => -- select module 2
  m2 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
* WHEN mod3 => -- select module 3
  m3 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN mod4 => -- select module 4
  m4 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN mod5 => -- select module 5
  m5 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';

```

```

WHEN mod6 => -- select module 6
  m6 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN mod7 => -- select module 7
  m7 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN mod8 => -- select module 8
  m8 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';

```

รูปที่ 5.8 แสดงขั้นตอนการปฏิบัติคำสั่ง *mod1-mod8*

สำหรับคำสั่งที่ต้องใช้ขนาด 2 ไบต์แล้วไบต์ที่สองจะถูกอ่านเข้ามาเก็บไว้ที่ตัวแปร *byte2*

ดังรูปที่ 5.9

```

-- read second byte
read_instruction <= '1'; WAIT FOR read_high_time;
byte2 := instruct;
read_instruction <= '0'; WAIT FOR read_low_time;

```

รูปที่ 5.9 แสดงขั้นตอนการอ่านคำสั่งไบต์ที่สอง

คำสั่งไบต์ที่สองจะเป็นข้อมูลแสดงขนาดของภาพในแนว *row* และ *column* โดยดูจากบิตที่ 3 ถึง 0 ของคำสั่งไบต์แรกแล้วจึงเก็บค่าลงในตัวแปร *no_row* หรือ *no_col* ดังรูปที่ 5.10

```

-- execute second byte instruction
IF byte1 (3 DOWNT0 0) = row THEN
  ASSERT
    (NOT (byte1 (3 DOWNT0 0) = row))
  REPORT
    "Being keep the number of rows"
  SEVERITY NOTE;
  no_row := byte2;
ELSE
  ASSERT
    (NOT (byte1 (3 DOWNT0 0) = col))
  REPORT

```

```

"Being keep the number of cols"
SEVERITY NOTE;
no_col := byte2;
END IF;

```

รูปที่ 5.10 แสดงการทำคำสั่งไบต์ที่สอง

5.2 โครงสร้างของวงจรควบคุม (Structural of Controller)

หลังจากผ่านขั้นตอนการออกแบบในระดับพฤติกรรมเรียบร้อยแล้วก็มาสู่ขั้นตอนการออกแบบในระดับข้อมูลและโครงสร้าง รูปที่ 5.11 แสดงโครงสร้างบัสของวงจรควบคุมซึ่งประกอบด้วยส่วนที่สำคัญ 2 ส่วนคือ ส่วนควบคุมภายใน (Logic of Controller) ทำหน้าที่แปลคำสั่งที่ได้รับเป็นสัญญาณควบคุมส่งออกไป และส่วนข้อมูลใช้เก็บข้อมูลต่างๆที่เป็นเช่น Instruction Register (IR) เป็นที่พักของคำสั่งที่ได้จากภายนอก Row Unit และ Col Unit สำหรับเก็บค่าขนาดของข้อมูลภาพ Buffer ใช้เป็นที่พักของข้อมูลในระหว่างขั้นตอนการย้ายข้อมูลระหว่างหน่วยความจำทั้งสอง Program Counter (PC) เป็นตัวกำหนดตำแหน่งหน่วยความจำ Memory Address Register Unit (MAR) เป็นที่พักข้อมูลตำแหน่งหน่วยความจำก่อนส่งเข้าสู่บัส ABUS Compare เป็นวงจรเปรียบเทียบขนาดของข้อมูลภาพ สำหรับรายละเอียดการบรรยายหน้าที่การทำงานของส่วนข้อมูลต่างๆสามารถดูได้จากหัวข้อ 4.1.1.3 บทที่ 4

5.2.2 การบรรยายองค์ประกอบ (Description of Components)

ส่วนควบคุมภายในเป็นส่วนที่ใช้ควบคุมการทำงานของวงจรควบคุมอีกทีหนึ่ง ในหัวข้อนี้จะได้กล่าวถึงการบรรยายระดับข้อมูลของส่วนควบคุมภายในซึ่งทำให้มองเห็นลักษณะสำคัญของการประสานงานภายใน การทำงานที่ขึ้นกับเวลาและสัญญาณนาฬิกาเหมือนกับการทำงานของวงจรฮาร์ดแวร์จริงๆ การบรรยายในระดับนี้จะเป็นส่วนสำคัญที่เชื่อมประสานระหว่างแนวคิดในระดับพฤติกรรมไปสู่ฮาร์ดแวร์ในระดับโครงสร้าง

5.2.2.1 การบรรยายระดับข้อมูลของส่วนควบคุมภายใน

(Dataflow of Logic_of_Controller)

ในการบรรยายได้กำหนดขั้นตอนการทำงานเป็น 4 สเตท(series of states) แต่ละสเตทจะให้สัญญาณควบคุมเป็นเวลาหนึ่งคาบสัญญาณนาฬิกา เมื่อเสร็จสิ้นการปฏิบัติในแต่ละสเตทก็จะมี การส่งสัญญาณกระตุ้นการทำงานในสเตทต่อไปตามแต่เงื่อนไขในขณะนั้นๆ รูปที่ 5.12 แสดงการบรรยายการเชื่อมต่อของส่วนควบคุมภายใน ใน GENERIC จะกำหนดค่าเวลาหน่วงสำหรับสัญญาณการอ่านหรือเขียนกับหน่วยความจำ ในส่วนของ PORT จะเป็นส่วนของอินพุตและเอาต์พุตที่ใช้ติดต่อกับส่วนข้อมูล

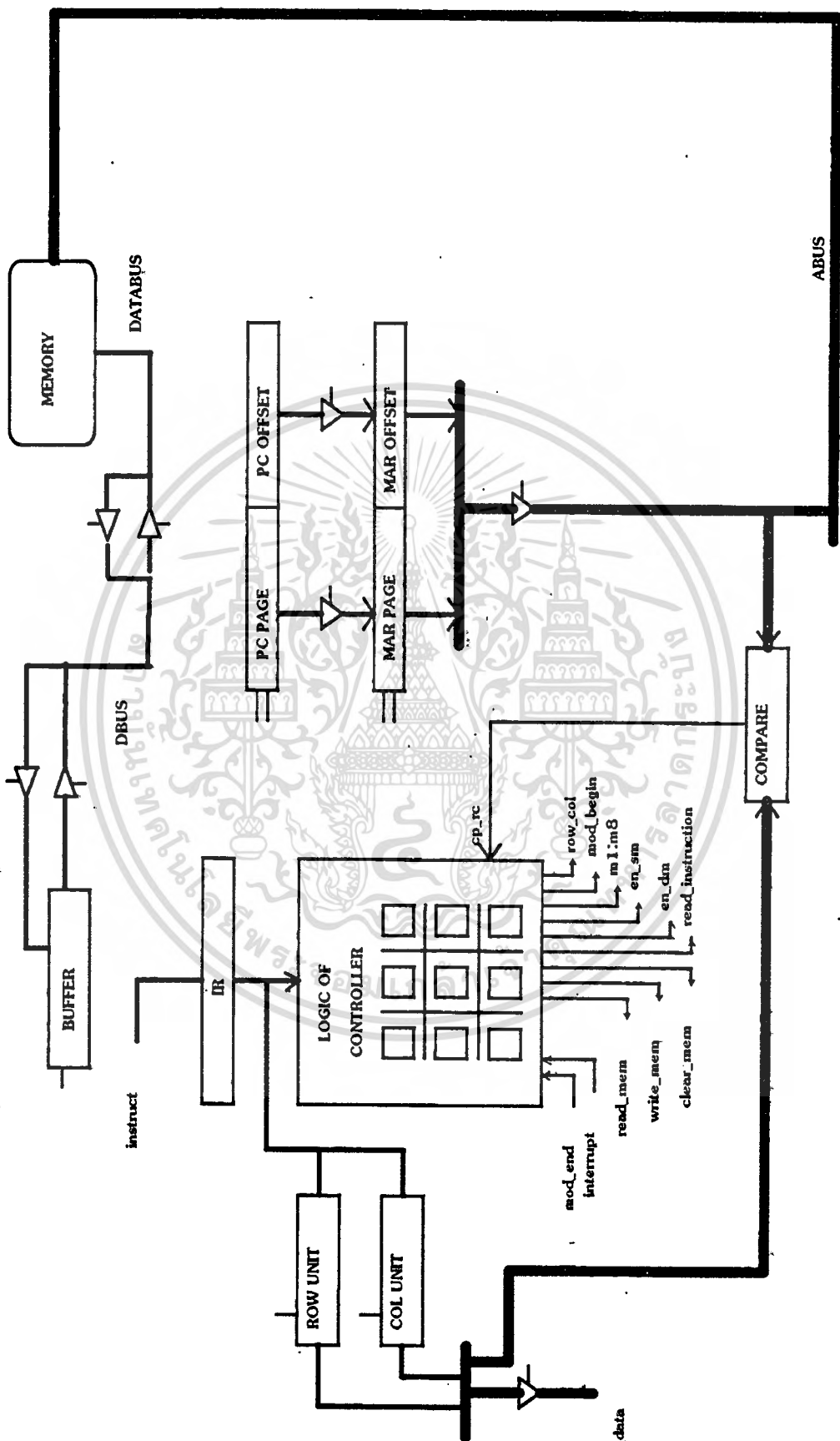
```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY logic_of_controller IS
  GENERIC ( read_delay, write_delay : TIME : 3 NS);
  PORT ( clk : IN qit;
        interrupt, mod_end, cp_rc : IN qit;
        ir_lines : IN byte;

```

5.2.1 โครงสร้างบัสของวงจรควบคุม (Controller Bussing Structure)



รูปที่ 5.1.1 แสดงโครงสร้างบัสของวงจรควบคุม

```

-- register control signals :
load_ir, load_row, load_col, load_buffer,
increment_pc, reset_pc,
load_page_mar, load_offset_mar,
-- bus connection control signals :
rc_on_rcdata_bus,
dbus_on_buffer, buffer_on_dbus,
dbus_on_databus, databus_on_dbus,
pc_on_mar_page_bus, pc_on_mar_offset_bus,
mar_on_abus,
-- memory control signals :
read_mem, write_mem, clear_mem,
en_sm, en_dm,
-- other control signals :
read_instr, mod_begin, row_col,
m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit );
END logic_of_controller;

```

รูปที่ 5.12 แสดงการบรรยายการเชื่อมต่อของส่วนควบคุมภายใน

การบรรยายการทำงานในระดับข้อมูลแสดงดังรูปที่ 5.13 ได้กำหนดสัญญาณการ์ดโดยใช้ฟังก์ชัน *oring* ที่อยู่ในแพ็คเกจ *basic_utilities* ตัวอักษร “_oi” ที่ต่อท้ายชื่อสัญญาณทุกตัวเพื่อแสดงว่ามีการใช้ฟังก์ชัน *oring* สัญญาณนั้น ๆ สัญญาณ *s* ใช้แทนสถานะแต่ละสเตท

```

ARCHITECTURE dataflow OF logic_of_controller IS
-- oring is implied in the following signals (oi)
SIGNAL load_ir_oi, load_row_oi, load_col_oi, load_buffer_oi,
increment_pc_oi, reset_pc_oi,
load_page_mar_oi, load_offset_mar_oi,
rc_on_rcdata_bus_oi,
dbus_on_buffer_oi, buffer_on_dbus_oi,
dbus_on_databus_oi, databus_on_dbus_oi,
pc_on_mar_page_bus_oi, pc_on_mar_offset_bus_oi,
mar_on_abus_oi,
read_mem_oi, write_mem_oi, clear_mem_oi,
en_sm_oi, en_dm_oi,
read_instr_oi, mod_begin_oi, row_col_oi,
m1_oi, m2_oi, m3_oi, m4_oi,
m5_oi, m6_oi, m7_oi, m8_oi : ored_qit BUS;
SIGNAL s : ored_qit_vector(4 DOWNT0 1) REGISTER := '0001';

```

รูปที่ 5.13 แสดงการกำหนดสัญญาณสำหรับการบรรยาย

รูปที่ 5.14 แสดงการกำหนดค่าจากสัญญาณเอาต์พุตที่ผ่านฟังก์ชัน *oring* ให้กับเอาต์พุตจริงของวงจร

```

BEGIN
load_ir <= load_ir_o;
load_row <= load_row_o;
load_col <= load_col_o;
load_buffer <= load_buffer_o;
increment_pc <= increment_pc_o;
reset_pc <= reset_pc_o;
load_page_mar <= load_page_mar_o;
load_offset_mar <= load_offset_mar_o;
rc_on_rcdata_bus <= rc_on_rcdata_bus_o;
dbus_on_buffer <= dbus_on_buffer_o;
buffer_on_dbus <= buffer_on_dbus_o;
dbus_on_databus <= dbus_on_databus_o;
databus_on_dbus <= databus_on_dbus_o;
pc_on_mar_page_bus <= pc_on_mar_page_bus_o;
pc_on_mar_offset_bus <= pc_on_mar_offset_bus_o;
mar_on_abus <= mar_on_abus_o;
read_mem <= read_mem_o;
write_mem <= write_mem_o;
clear_mem <= clear_mem_o;
en_sm <= en_sm_o;
en_dm <= en_dm_o;
read_instr <= read_instr_o;
mod_begin <= mod_begin_o;
row_col <= row_col_o;
m1 <= m1_o;
m2 <= m2_o;
m3 <= m3_o;
m4 <= m4_o;
m5 <= m5_o;
m6 <= m6_o;
m7 <= m7_o;
m8 <= m8_o;

```

รูปที่ 5.14 แสดงการกำหนดค่าให้กับเอาต์พุตจริงของวงจร

รูปที่ 5.15 แสดง *s1* บล็อกสำหรับสเตทที่ 1 ในบล็อกนี้จะเริ่มต้นการทำงานโดยการวางข้อมูลจาก *PC* ไปบน *MAR* ผ่านบัส ถ้าสัญญาณ *interrupt* เป็น '1' อินพุต *reset* ของ *PC* ก็จะได้รับสัญญาณและสเตทจะถูกเซตกลับไปสู่สถานะที่ 1 อีกครั้ง แต่ถ้าไม่มีสัญญาณ *interrupt* สเตทจะถูกเปลี่ยนไปสู่สถานะที่ 2 ในตอนกลางของสัญญาณนาฬิกา

```

s1 : BLOCK (s(1) = '1')
BEGIN -- start of fetch
-- pc to mar
pc_on_mar_page_bus_o <= GUARDED '1';
pc_on_mar_offset_bus_o <= GUARDED '1';
load_page_mar_o <= GUARDED '1';

```

```

load_offset_mar_oi <= GUARDED '1';
-- reset pc if interrupt
reset_pc_oi <= GUARDED '1' WHEN interrupt = '1' ELSE '0';
-- goto 2 if interrupt is off
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(1) <= GUARDED '1' WHEN interrupt = '1' ELSE '0';
    s(2) <= GUARDED '1' WHEN interrupt /= '1' ELSE '0';
END BLOCK ck;
END BLOCK s1;

```

รูปที่ 5.15 แสดง s1 บล็อกสำหรับสเตทที่ 1

เมื่อเข้าสู่สเตทที่ 2 สัญญาณพร้อมรับคำสั่ง *read_instr_oi* จะถูกส่งออกไปและตามด้วยสัญญาณ *load_ir_oi* เพื่อเก็บค่าของคำสั่งลงใน *IR* ในตอนกลางของสัญญาณนาฬิกาสถานะของสเตทจะเปลี่ยนไปสู่สถานะที่ 3 ต่อไป ดังรูปที่ 5.16

```

s2 : BLOCK (s(2) = '1')
BEGIN -- fetching continue
    -- read instruction into ir
    read_instr_oi <= GUARDED '1';
    load_ir_oi <= GUARDED '1' AFTER read_delay;
    -- goto 3
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        s(3) <= GUARDED '1'
    END BLOCK ck;
END BLOCK s2;

```

รูปที่ 5.16 แสดง s2 บล็อกสำหรับสเตทที่ 2

สเตทที่ 3 เป็นสเตทของการเลือกปฏิบัติตามคำสั่งประเภทไบต์เดียว แต่ถ้าเป็นคำสั่งประเภทสองไบต์ก็จะข้ามไปที่สเตท 4 เลย ภายในสเตทที่ 3 ประกอบด้วยบล็อกของการตรวจเช็คคำสั่งว่าเป็นคำสั่งใดก็จะปฏิบัติบล็อกนั้นๆ รูปที่ 5.17 แสดงบล็อกการทำงานของคำสั่งต่างๆ สำหรับบล็อกของคำสั่งในการเลือกโมดูลได้แสดงให้ดูเพียงบล็อก *mod1* เนื่องจากบล็อกโมดูลอื่นๆก็มีลักษณะคล้ายคลึงกัน

```

s3 : BLOCK (s(3) = '1')
BEGIN
    -- pc to mar
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';
    load_page_mar_oi <= GUARDED '1';
    load_offset_mar_oi <= GUARDED '1';
    -- goto 4 if not single byte instruction
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN

```

s(4) <= GUARDED '1' WHEN ir_lines(7 DOWNT0 4) /= '0000' ELSE '0';

```

END BLOCK ck;

-- perform single byte instruction
sb : BLOCK ((ir_lines(7 DOWNT0 4) = "0000") AND GUARD)
BEGIN
    -- clear instruction
    clm : BLOCK ((ir_lines(3 DOWNT0 0) = "0001") AND GUARD)
    BEGIN
        en_sm_oi <= GUARDED '1';
        en_dm_oi <= GUARDED '1';
        clear_mem_oi <= GUARDED '1' AFTER write_delay;
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(1) <= GUARDED '1';
        END BLOCK ck;
    END BLOCK clm;

    -- swap memory instruction
    swm : BLOCK ((ir_lines(3 DOWNT0 0) = "0010") AND GUARD)
    BEGIN
        -- read from destination memory
        en_sm_oi <= GUARDED '0';
        en_dm_oi <= GUARDED '1';
        mar_onabus_oi <= GUARDED '1';
        read_mem_oi <= GUARDED '1' AFTER read_delay;
        databus_ondbus_oi <= GUARDED '1';
        dbus_onbuffer_oi <= GUARDED '1';
        load_buffer_oi <= GUARDED '1';
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            -- write to source memory
            en_sm_oi <= GUARDED '1';
            en_dm_oi <= GUARDED '0';
            mar_onabus_oi <= GUARDED '1';
            buffer_ondbus_oi <= GUARDED '1';
            dbus_ondatabus_oi <= GUARDED '1';
            write_mem_oi <= GUARDED '1' AFTER write_delay;

            -- continue swaping if not ready
            chk : BLOCK (cp_rc /= '1' AND GUARD)
            BEGIN
                -- increment pc
                increment_pc_oi <= GUARDED '1';
                cck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
                BEGIN
                    s(3) <= GUARDED '1';
                END BLOCK cck;
            END BLOCK chk;
        END BLOCK ck;
    END BLOCK swm;

```

```

-- module 1 instruction
mod1 : BLOCK ((ir_lines(3 DOWNT0 0) = "1000") AND GUARD)
BEGIN
  m1_oi <= GUARDED '1';
  rc_on_rcdata_bus_oi <= GUARDED '1';
  row_col_oi <= GUARDED '1' AFTER write_delay;
  ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
  BEGIN
    mod_begin_oi <= GUARDED '1';
  END BLOCK ck;
END BLOCK mod1;

```

รูปที่ 5.17 แสดง s3 บล็อกสำหรับสเตทที่ 3

สเตทที่ 4 เป็นบล็อกของการนำข้อมูลไบต์ที่สองซึ่งเป็นข้อมูลแสดงขนาดของภาพไปเก็บไว้ใน row unit และ col unit หลังจากนั้นก็จะกลับไปสู่สเตทที่ 1 เพื่อรับคำสั่งใหม่ต่อไป

```

s4 : BLOCK (s(4) = '1')
BEGIN
  lr : BLOCK ((ir_lines(3 DOWNT0 0) = "0011") AND GUARD)
  BEGIN
    read_instr_oi <= GUARDED '1';
    load_ir_oi <= GUARDED '1' AFTER read_delay;
    load_row_oi <= GUARDED '1' AFTER read_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
      s(1) <= GUARDED '1';
    END BLOCK ck;
  END BLOCK lr;
  lc : BLOCK ((ir_lines(3 DOWNT0 0) = "1100") AND GUARD)
  BEGIN
    read_instr_oi <= GUARDED '1';
    load_ir_oi <= GUARDED '1' AFTER read_delay;
    load_col_oi <= GUARDED '1' AFTER read_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
      s(1) <= GUARDED '1';
    END BLOCK ck;
  END BLOCK lc;
END BLOCK s4;

```

รูปที่ 5.18 แสดง s4 บล็อกสำหรับสเตทที่ 4

ในบล็อกสุดท้ายจะต้องกำหนดค่า "0000" ให้กับทุกสเตท s การทำเช่นนี้ก็เพื่อให้ทุกสเตททริกเกอร์หลังจากที่ได้กระตุ้นให้เกิดสเตทต่อไป ดังรูปที่ 5.19

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

ck : BLOCK (clk = '0' AND NOT clk'STABLE)
BEGIN
    s(4 DOWNT0 1) <= GUARDED "0000";
END BLOCK ck;
END dataflow;

```

รูปที่ 5.19 แสดงการรีเซตสเตท

5.2.2.2 โครงร่างของส่วนควบคุมภายใน (General Outline of Logic_of_Controller)

รูปที่ 5.20 แสดงโครงสร้างที่สมบูรณ์ของส่วนควบคุมภายในตามลักษณะการบรรยายในระดับข้อมูล ลักษณะของส่วนสเตทที่ใช้ฟลิปฟล็อปในการกำหนดสถานะและใช้บับเบิล(bubble)แทนส่วนของลอจิกบัสลอค อินพุทของลอจิกบัสลอคได้มาจากสเตทฟลิปฟล็อปและจากอินพุทภายนอกซึ่งจะมีผลต่อการเปลี่ยนสถานะของสเตท เอาท์พุทที่ได้ก็คือสัญญาณควบคุม

5.2.2.3 การบรรยายระดับโครงสร้างของส่วนควบคุมภายใน (Structural of Logic_of_Controller)

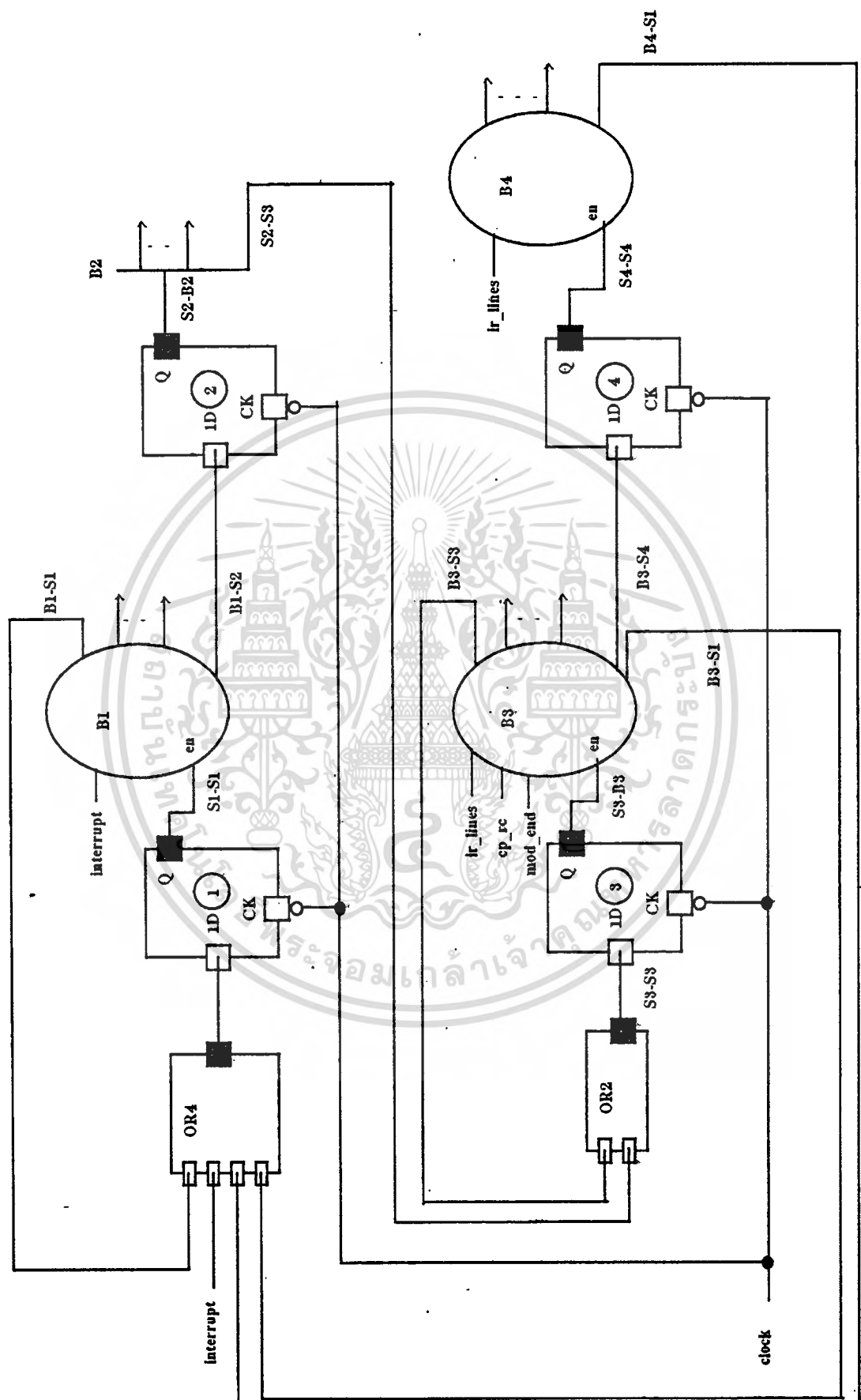
รูปที่ 5.21 แสดงการบรรยายระดับโครงสร้างของส่วนควบคุมภายในเป็นไปตามโครงสร้างในรูปที่ 5.20

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY ctrl;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY logic_of_controller IS
    PORT ( clk, interrupt, mod_end, cp_rc : IN qit;
          ir_lines : IN byte;
          -- register control signals :
          load_ir, load_row, load_col, load_buffer, increment_pc, reset_pc,
          load_page_mar, load_offset_mar;
          -- bus connection control signals :
          rc_on_rcdata_bus, dbus_on_buffer, buffer_on_dbus,
          dbus_on_databus, databus_on_dbus,
          pc_on_mar_page_bus, pc_on_mar_offset_bus, mar_on_abus,
          -- memory control signals :
          read_mem, write_mem, clear_mem, en_sm, en_dm,
          -- other control signals :
          read_instr, mod_begin, row_col,
          m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit );
END logic_of_controller;

```



รูปที่ 5.20 แสดงโครงสร้างของส่วนควบคุมภายใน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้ภายในเท่านั้น ไม่ควรเผยแพร่ให้ผู้อื่นได้ใช้โดยไม่ได้รับอนุญาตจากผู้จัดทำเอกสาร
ไม่ว่ากรณีใดก็ตาม หากมีข้อผิดพลาดหรือข้อสงสัย กรุณาแจ้งให้เราทราบ และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ARCHITECTURE structural OF logic_of_controller IS

COMPONENT cs1

PORT (s1, int, clk : IN qit;
b1_reset_pc, b1_load_page_mar, b1_load_offset_mar,
b1_pc_on_mar_page_bus, b1_on_mar_offset_bus,
b1_s1, b1_s2 : OUT qit);

END COMPONENT;

COMPONENT cs2

PORT (s2, clk : IN qit;
b2_load_ir, b2_read_instr, s2_s3 : OUT qit);

END COMPONENT;

COMPONENT cs3

PORT (s3, clk, cp_rc, mod_end : IN qit;
ir_lines : IN byte;
b3_load_buffer, b3_increment_pc,
b3_load_page_mar, b3_load_offset_mar,
b3_rc_on_rcdata_bus, b3_dbus_on_buffer, b3_buffer_on_dbus,
b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus, b3_mar_on_abus,
b3_read_mem, b3_write_mem, b3_clear_mem, b3_en_sm, b3_en_dm,
b3_mod_begin, b3_row_col,
b3_m1, b3_m2, b3_m3, b3_m4, b3_m5, b3_m6, b3_m7, b3_m8,
b3_s1, b3_s3, b3_s4 : OUT qit);

END COMPONENT;

COMPONENT cs4

PORT (s4, clk : IN qit;
ir_lines : IN nibble;
b4_load_ir, b4_load_row, b4_load_col, b4_read_instr, b4_s1);

END COMPONENT;

FOR s1 : cs1 USE ENTITY ctrl.state_control_1 (structural);

FOR s2 : cs1 USE ENTITY ctrl.state_control_2 (structural);

FOR s3 : cs1 USE ENTITY ctrl.state_control_3 (structural);

FOR s4 : cs1 USE ENTITY ctrl.state_control_4 (structural);

FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);

FOR ALL : o4 USE ENTITY gates.or4_q (average_delay);

SIGNAL b2_load_ir, b4_load_ir : qit;

SIGNAL b1_load_page_mar, b1_load_offset_mar,

b3_load_page_mar, b3_load_offset_mar : qit;

SIGNAL b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,

b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus : qit;

SIGNAL b2_read_instr, b4_read_instr : qit;

SIGNAL s1_s1, b1_s1, b1_s2 : qit;

SIGNAL s2_b3 : qit;

SIGNAL s3_s3, b3_s1, b3_s3, b3_s4 : qit;

SIGNAL b4_s1 : qit;

BEGIN

g0 : o4 PORT MAP (interrupt, b1_s1, b3_s1, b4_s1, s1_s1);

s1 : cs1 PORT MAP (s1_s1, interrupt, clk, reset_pc,

```

        b1_load_page_mar, b1_load_offset_mar,
        b1_pc_on_mar_page_bus, b1_on_mar_offset_bus,
        b1_s1, b1_s2);
s2 : cs2 PORT MAP (b1_s2, clk, b2_load_ir, b2_read_instr, s2_s3);
g1 : o2 PORT MAP (s2_s3, b3_s3, s3_s3);
s3 : cs3 PORT MAP (s3_s3, clk, cp_rc, mod_end, ir_lines,
        load_buffer, increment_pc,
        b3_load_page_mar, b3_load_offset_mar,
        rc_on_rcdata_bus, dbus_on_buffer, buffer_on_dbus,
        b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus,
        mar_on_abus,
        read_mem, write_mem, clear_mem, en_sm, en_dm,
        mod_begin, row_col, m1, m2, m3, m4, m5, m6, m7, m8,
        b3_s1, b3_s3, b3_s4);
s4 : cs4 PORT MAP (b3_s4, clk, ir_lines, b4_load_ir,
        load_row, load_col, b4_read_instr, b4_s1);
g2 : o2 PORT MAP (b2_load_ir, b4_load_ir, load_ir);
g3 : o2 PORT MAP (b1_load_page_mar, b3_load_page_mar, load_page_mar);
g4 : o2 PORT MAP (b1_load_offset_mar, b3_load_offset_mar, load_offset_mar);
g5 : o2 PORT MAP (b1_pc_on_mar_page_bus, b3_pc_on_mar_page_bus,
        pc_on_mar_page_bus);
g6 : o2 PORT MAP (b1_pc_on_mar_offset_bus, b3_pc_on_mar_offset_bus,
        pc_on_mar_offset_bus);
g7 : o2 PORT MAP (b2_read_instr, b4_read_instr, read_instr);
END structural;

```

รูปที่ 5.21 แสดงการบรรยายระดับโครงสร้างของส่วนควบคุมภายใน

องค์ประกอบภายใน $s1 - s4$ เป็นวงจรรหัสที่ออกแบบให้ทำหน้าที่แทนบล็อกสเตทแต่ละบล็อกที่ใช้ในการบรรยายในระดับข้อมูลที่ผ่านมา รูปที่ 5.22 - 5.25 แสดงการบรรยายในระดับโครงสร้างของแต่ละสเตท ภายในแต่ละสเตทจะใช้ master-slave D-flipflop เป็นตัวกำหนดสัญญาณสถานะที่สเตทใด ๆ

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY state_control_1 IS
    PORT ( s1, int, clk : IN qit;
        b1_reset_pc,
        b1_load_page_mar, b1_load_offset_mar,
        b1_pc_on_mar_page_bus, b1_on_mar_offset_bus,
        b1_s1, b1_s2 : OUT qit);
END state_control_1;

```

```

ARCHITECTURE structural OF state_control_1 IS
  FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
  FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
  SIGNAL s1_b1 : qit;
  SIGNAL inv_clk, inv_int : qit;
BEGIN
  control_signal : msdff PORT MAP (s1, clk, s1_b1);
  g0 : inv PORT MAP (clk, inv_clk);
  g1 : inv PORT MAP (int, inv_int);
  g2 : a2 PORT MAP (s1_b1, int, b1_reset_pc);
  g3 : b1_load_page_mar <= s1_b1;
  g4 : b1_load_offset_mar <= s1_b1;
  g5 : b1_pc_on_mar_page_bus <= s1_b1;
  g6 : b1_on_mar_offset_bus <= s1_b1;
  g7 : a3 PORT MAP (inv_clk, s1_b1, int, b1_s1);
  g8 : a3 PORT MAP (inv_clk, s1_b1, inv_int, b1_s2);
END structural;

```

รูปที่ 5.22 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 1

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY state_control_2 IS
  PORT ( s2, clk : IN qit;
         b2_load_ir, b2_read_instr, s2_s3 : OUT qit);
END state_control_2;
ARCHITECTURE structural OF state_control_2 IS
  FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
  FOR ALL : delay USE ENTITY gates.signal_delay (structural);
  SIGNAL inv_clk : qit;
  SIGNAL s2_b2 : qit;
BEGIN
  control_signal : msdff PORT MAP (s2, clk, s2_b2);
  g0 : inv PORT MAP (clk, inv_clk);
  g1 : b2_read_instr <= s2_b2;
  g2 : delay PORT MAP (s2_b2, b2_load_ir);
  g3 : a2 PORT MAP (s2_b2, inv_clk, s2_s3);
END structural;

```

รูปที่ 5.23 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 2

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_control_3 IS
    PORT ( s3, clk, cp_rc, mod_end : IN qit;
          ir_lines : IN byte;
          b3_load_buffer, b3_increment_pc,
          b3_load_page_mar, b3_load_offset_mar,
          b3_rc_on_rcdata_bus,
          b3_dbus_on_buffer, b3_buffer_on_dbus,
          b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus,
          b3_mar_on_abus,
          b3_read_mem, b3_write_mem, b3_clear_mem,
          b3_en_sm, b3_en_dm,
          b3_mod_begin, b3_row_col,
          b3_m1, b3_m2, b3_m3, b3_m4, b3_m5, b3_m6, b3_m7, b3_m8,
          b3_s1, b3_s3, b3_s4 : OUT qit);
END state_control_3;

ARCHITECTURE structural OF state_control_3 IS
    FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
    FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
    FOR ALL : a4 USE ENTITY gates.and4_q (average_delay);
    FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);
    FOR ALL : o4 USE ENTITY gates.or4_q (average_delay);
    FOR ALL : delay USE ENTITY gates.signal_delay (structural);
    SIGNAL inv_clk, inv_cp_rc : qit;
    SIGNAL ir0, ir1, ir2, ir3, ir4, ir5, ir6, ir7 : qit;
    SIGNAL inv_ir0, inv_ir1, inv_ir2, inv_ir3 : qit;
    SIGNAL inv_ir4, inv_ir5, inv_ir6, inv_ir7 : qit;
    SIGNAL im1, im2, im3, im4, im5, im6, im7, im8, im9 : qit;
    SIGNAL im10, im11, im12, im13, im14, im15 : qit;
    SIGNAL cc, ss, mm1, mm2, mm3, mm4, mm5, mm6, mm7, mm8 : qit;
    SIGNAL cdm_en_sm, cdm_en_dm : qit;
    SIGNAL swmr_en_sm, swmr_en_dm, swmr_mar_on_abus : qit;
    SIGNAL swmw_en_sm, swmw_en_dm, swmw_mar_on_abus : qit;
    SIGNAL m1_rc_on_rcdata_bus, m2_rc_on_rcdata_bus, m3_rc_on_rcdata_bus : qit;
    SIGNAL m4_rc_on_rcdata_bus, m5_rc_on_rcdata_bus, m6_rc_on_rcdata_bus : qit;
    SIGNAL m7_rc_on_rcdata_bus, m8_rc_on_rcdata_bus : qit;
    SIGNAL m14_rc_on_rcdata_bus, m58_rc_on_rcdata_bus : qit;

```

```

SIGNAL m1_row_col, m2_row_col, m3_row_col, m4_row_col : qit;
SIGNAL m5_row_col, m6_row_col, m7_row_col, m8_row_col : qit;
SIGNAL m14_row_col, m58_row_col : qit;
SIGNAL m1_mod_begin, m2_mod_begin, m3_mod_begin, m4_mod_begin : qit;
SIGNAL m5_mod_begin, m6_mod_begin, m7_mod_begin, m8_mod_begin : qit;
SIGNAL m14_mod_begin, m58_mod_begin : qit;
SIGNAL b3_s1_1, b3_s1_2 : qit;
SIGNAL s3_b3 : qit;

```

```

BEGIN

```

```

    ir0 <= ir_lines(0);
    ir1 <= ir_lines(1);
    ir2 <= ir_lines(2);
    ir3 <= ir_lines(3);
    ir4 <= ir_lines(4);
    ir5 <= ir_lines(5);
    ir6 <= ir_lines(6);
    ir7 <= ir_lines(7);
    control_signal : msdff PORT MAP (s3, clk, s3_b3);
    g0 : b3_load_page_mar <= s3_b3;
    g1 : b3_load_offset_mar <= s3_b3;
    g2 : b3_pc_on_mar_page_bus <= s3_b3;
    g3 : b3_pc_on_mar_offset_bus <= s3_b3;
    g4 : inv PORT MAP (clk, inv_clk);
    g5 : inv PORT MAP (cp_rc, inv_cp_rc);
    g6 : inv PORT MAP (ir0, inv_ir0);
    g7 : inv PORT MAP (ir1, inv_ir1);
    g8 : inv PORT MAP (ir2, inv_ir2);
    g9 : inv PORT MAP (ir3, inv_ir3);
    g10 : inv PORT MAP (ir4, inv_ir4);
    g11 : inv PORT MAP (ir5, inv_ir5);
    g12 : inv PORT MAP (ir6, inv_ir6);
    g13 : inv PORT MAP (ir7, inv_ir7);
    g14 : o4 PORT MAP (ir7, ir6, ir5, ir4, im1);
    g15 : a3 PORT MAP (inv_clk, s3_b3, im1, b3_s4);
    g16 : inv PORT MAP (im1, im2);
    -- clear memory instruction
    g17 : o4 PORT MAP (inv_ir3, inv_ir2, inv_ir1, ir0, im3);
    g18 : a3 PORT MAP (s3_b3, im2, im3, cc);
    g19 : delay PORT MAP (cc, b3_clear_mem);
    g20 : clm_en_sm <= cc;
    g21 : clm_en_dm <= cc;
    g22 : a2 PORT MAP (inv_clk, cc, b3_s1_1);
    -- swap memory instruction
    -- read
    g23 : o4 PORT MAP (inv_ir3, inv_ir2, ir1, inv_ir0, im4);
    g24 : a3 PORT MAP (s3_b3, im2, im4, ss);
    g25 : b3_load_buffer <= ss;

```

```

g26 : b3_dbus_on_buffer <= ss;
g27 : b3_databus_on_dbus <= ss;
g28 : swmr_mar_on_abus <= ss;
g29 : delay PORT MAP (ss, b3_read_mem);
g30 : inv PORT MAP (ss, swmr_en_sm);
g31 : swmr_en_dm <= ss;
    -- write
g32 : a2 PORT MAP (ss, inv_clk, im5);
g33 : b3_buffer_on_dbus <= im5;
g34 : b3_dbus_on_databus <= im5;
g35 : swmw_mar_on_abus <= im5;
g36 : delay PORT MAP (im5, b3_write_mem);
g37 : swmw_en_sm <= im5;
g38 : inv PORT MAP (im5, swmw_en_dm);
g39 : a3 PORT MAP (inv_cp_rc, ss, inv_clk, im6);
g40 : b3_increment_pc <= im6;
g41 : a2 PORT MAP (im6, inv_clk, b3_s3);
    -- module 1 instruction
g42 : o4 PORT MAP (ir3, inv_ir2, inv_ir1, inv_ir0, im7);
g43 : a3 PORT MAP (s3_b3, im2, im7, mm1);
g44 : m1_rc_on_rcdata_bus <= mm1;
g45 : delay PORT MAP (mm1, m1_row_col);
g46 : b3_m1 <= mm1;
g47 : a2 PORT MAP (inv_clk, mm1, m1_mod_begin);
    -- module 2 instruction
g48 : o4 PORT MAP (ir3, inv_ir2, inv_ir1, ir0, im8);
g49 : a3 PORT MAP (s3_b3, im2, im8, mm2);
g50 : m2_rc_on_rcdata_bus <= mm2;
g51 : delay PORT MAP (mm2, m2_row_col);
g52 : b3_m2 <= mm2;
g53 : a2 PORT MAP (inv_clk, mm2, m2_mod_begin);
    -- module 3 instruction
g54 : o4 PORT MAP (ir3, inv_ir2, ir1, inv_ir0, im9);
g55 : a3 PORT MAP (s3_b3, im2, im9, mm3);
g56 : m3_rc_on_rcdata_bus <= mm3;
g57 : delay PORT MAP (mm3, m3_row_col);
g58 : b3_m3 <= mm3;
g59 : a2 PORT MAP (inv_clk, mm3, m3_mod_begin);
    -- module 4 instruction
g60 : o4 PORT MAP (ir3, inv_ir2, ir1, ir0, im10);
g61 : a3 PORT MAP (s3_b3, im2, im10, mm4);
g62 : m4_rc_on_rcdata_bus <= mm4;
g63 : delay PORT MAP (mm4, m4_row_col);
g64 : b3_m4 <= mm4;
g65 : a2 PORT MAP (inv_clk, mm4, m4_mod_begin);
    -- module 5 instruction
g66 : o4 PORT MAP (ir3, ir2, inv_ir1, inv_ir0, im11);

```

เอกสารนี้เป็นเอกสารของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

g67 : a3 PORT MAP (s3_b3, im2, im11, mm5);
g68 : m5_rc_on_rcdata_bus <= mm5;
g69 : delay PORT MAP (mm5, m5_row_col);
g70 : b3_m5 <= mm5;
g71 : a2 PORT MAP (inv_clk, mm5, m5_mod_begin);
-- module 6 instruction
g72 : o4 PORT MAP (ir3, ir2, inv_ir1, ir0, im12);
g73 : a3 PORT MAP (s3_b3, im2, im12, mm6);
g74 : m6_rc_on_rcdata_bus <= mm6;
g75 : delay PORT MAP (mm6, m6_row_col);
g76 : b3_m6 <= mm6;
g77 : a2 PORT MAP (inv_clk, mm6, m6_mod_begin);
-- module 7 instruction
g78 : o4 PORT MAP (ir3, ir2, ir1, inv_ir0, im13);
g79 : a3 PORT MAP (s3_b3, im2, im13, mm7);
g80 : m7_rc_on_rcdata_bus <= mm7;
g81 : delay PORT MAP (mm7, m7_row_col);
g82 : b3_m7 <= mm7;
g83 : a2 PORT MAP (inv_clk, mm7, m7_mod_begin);
-- module 8 instruction
g84 : o4 PORT MAP (ir3, ir2, ir1, ir0, im14);
g85 : a3 PORT MAP (s3_b3, im2, im14, mm8);
g86 : m8_rc_on_rcdata_bus <= mm8;
g87 : delay PORT MAP (mm8, m8_row_col);
g88 : b3_m8 <= mm8;
g89 : a2 PORT MAP (inv_clk, mm8, m8_mod_begin);
g90 : o2 PORT MAP (cp_rc, mod_end, im15);
g91 : a4 PORT MAP (s3_b3, inv_clk, im2, im15, b3_s1_2);
-- output OR together
g92 : o2 PORT MAP (b3_s1_1, b3_s1_2, b3_s1);
g93 : o2 PORT MAP (swmr_mar_on_abus, swmw_mar_on_abus, b3_mar_on_abus);
g94 : o3 PORT MAP (clm_en_sm, swmr_en_sm, swmw_en_sm, b3_en_sm);
g95 : o3 PORT MAP (clm_en_dm, swmr_en_dm, swmw_en_dm, b3_en_dm);
g96 : o4 PORT MAP (m1_rc_on_rcdata_bus, m2_rc_on_rcdata_bus,
                  m3_rc_on_rcdata_bus, m4_rc_on_rcdata_bus,
                  m14_rc_on_rcdata_bus);
g97 : o4 PORT MAP (m5_rc_on_rcdata_bus, m6_rc_on_rcdata_bus,
                  m7_rc_on_rcdata_bus, m8_rc_on_rcdata_bus,
                  m58_rc_on_rcdata_bus);
g98 : o2 PORT MAP (m14_rc_on_rcdata_bus, m58_rc_on_rcdata_bus,
                  b3_rc_on_rcdata_bus);
g99 : o4 PORT MAP (m1_row_col, m2_row_col, m3_row_col,
                  m4_row_col, m14_row_col);
g100 : o4 PORT MAP (m5_row_col, m6_row_col, m7_row_col,
                  m8_row_col, m58_row_col);
g101 : o2 PORT MAP (m14_row_col, m58_row_col, b3_row_col);
g102 : o4 PORT MAP (m1_mod_begin, m2_mod_begin, m3_mod_begin,

```

เอกสารนี้เป็นเอกสารของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ไม่ควรนำออกนอกระบบโดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

        m4_mod_begin, m14_mod_begin);
g103 : o4 PORT MAP (m5_mod_begin, m6_mod_begin, m7_mod_begin,
        m8_mod_begin, m58_mod_begin);
g104 : o2 PORT MAP (m14_mod_begin, m58_mod_begin, b3_mod_begin);
END structural;

```

รูปที่ 5.24 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 3

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_control_4 IS
    PORT ( s4, clk : IN qit;
          ir_lines : IN nibble;
          b4_load_ir, b4_load_row,
          b4_load_col, b4_read_instr, b4_s1 : OUT qit);
END state_control_4;

ARCHITECTURE structural OF state_control_4 IS
    FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
    FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);
    FOR ALL : o4 USE ENTITY gates.or4_q (average_delay);
    FOR ALL : delay USE ENTITY gates.signal_delay (structural);
    SIGNAL inv_clk : qit;
    SIGNAL ir0, ir1, ir2, ir3 : qit;
    SIGNAL inv_ir0, inv_ir1, inv_ir2, inv_ir3 : qit;
    SIGNAL rr, cc : qit;
    SIGNAL lr_load_ir, lr_read_instr, lr_b4_s1 : qit;
    SIGNAL lc_load_ir, lc_read_instr, lc_b4_s1 : qit;
    SIGNAL im1, im2 : qit;
    SIGNAL s4_b4 : qit;
BEGIN
    ir0 <= ir_lines(0);
    ir1 <= ir_lines(1);
    ir2 <= ir_lines(2);
    ir3 <= ir_lines(3);
    control_signal : msdff PORT MAP (s4, clk, s4_b4);
    g0 : inv PORT MAP (clk, inv_clk);
    g1 : inv PORT MAP (ir0, inv_ir0);
    g2 : inv PORT MAP (ir1, inv_ir1);

```

```

g3 : inv PORT MAP (ir2, inv_ir2);
g4 : inv PORT MAP (ir3, inv_ir3);
-- load row instruction
g5 : o4 PORT MAP (inv_ir3, inv_ir2, ir1, ir0, im1);
g6 : a2 PORT MAP (s4_b4, im1, rr);
g7 : delay PORT MAP (rr, lr_load_ir);
g8 : delay PORT MAP (rr, b4_load_row);
g9 : lr_read_instr <= rr;
g10 : a2 PORT MAP (inv_clk, rr, lr_b4_s1);
-- load col instruction
g11 : o4 PORT MAP (ir3, ir2, inv_ir1, inv_ir0, im2);
g12 : a2 PORT MAP (s4_b4, im2, cc);
g13 : delay PORT MAP (cc, lc_load_ir);
g14 : delay PORT MAP (cc, b4_load_col);
g15 : lc_read_instr <= cc;
g16 : a2 PORT MAP (inv_clk, cc, lc_b4_s1);
-- OR together
g17 : o2 PORT MAP (lr_load_ir, lc_load_ir, b4_load_ir);
g18 : o2 PORT MAP (lr_read_instr, lc_read_instr, b4_read_instr);
g19 : o2 PORT MAP (lr_b4_s1, lc_b4_s1, b4_s1);
END structural;

```

รูปที่ 5.25 แสดงการบรรยายในระดับโครงสร้างของสเตทที่ 4

5.2.3 การบรรยายระดับโครงสร้างของวงจรควบคุม

(Controller Structural Architecture)

หลังจากที่ได้ออกแบบองค์ประกอบแต่ละส่วนเรียบร้อยแล้วก็นำมาประกอบกันเป็นวงจรควบคุมตามลักษณะโครงสร้างบัสในรูปที่ 5.11 ดังแสดงในรูปที่ 5.26

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;

ENTITY controller IS
  PORT ( clk, interrupt, mod_end : IN qit;
        instruct : IN byte;
        clear_mem, read_mem, write_mem : OUT qit;
        dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
        abus : OUT word;
        read_instruction : OUT qit;
        en_sm, en_dm : OUT qit;
        m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
        r_c : OUT qit;
        r_c_data : OUT word;

```

```

    mod_begin : OUT qit;
    dbus_on_databus, databus_on_dbus : OUT qit);
END controller;

```

ARCHITECTURE structural OF controller IS

BEGIN

COMPONENT l_ctrl

```

    PORT ( clk : IN qit;
          interrupt, mod_end, cp_rc : IN qit;
          ir_lines : IN byte;
          load_ir, load_row, load_col, load_buffer,
          increment_pc, reset_pc,
          load_page_mar, load_offset_mar,
          rc_on_rcdata_bus,
          dbus_on_buffer, buffer_on_dbus,
          dbus_on_databus, databus_on_dbus,
          pc_on_mar_page_bus, pc_on_mar_offset_bus,
          mar_on_abus,
          read_mem, write_mem, clear_mem,
          en_sm, en_dm,
          read_instr, mod_begin, row_col,
          m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit );

```

END COMPONENT;

```

FOR logic : l_ctrl USE ENTITY ctrl.logic_of_controller(dataflow);
FOR ALL : reg USE ENTITY reg.register_unit(dataflow);
FOR counter : pc USE ENTITY reg.program_counter_unit(behavioral);
FOR memory_addr : mar USE ENTITY reg.memory_address_register_unit(behavioral);
FOR compare : cprc USE ENTITY reg.compare_row_col(dataflow);
FOR ALL : ts8 USE ENTITY reg.tri_state_8 (structural);
FOR ALL : ts16 USE ENTITY reg.tri_state_16 (structural);
SIGNAL cp_rc : qit;
SIGNAL ir_lines : byte;
SIGNAL rc_lines : word;
SIGNAL pc1_lines, pc2_lines, mar1_lines, mar2_lines : word;
SIGNAL d_buff, buff_d : byte;
SIGNAL dbus1, dbus2 : byte;
SIGNAL load_ir, load_row, load_col, load_buffer,
    increment_pc, reset_pc,
    load_page_mar, load_offset_mar : qit;
SIGNAL rc_on_rcdata_bus,
    dbus_on_buffer, buffer_on_dbus,
    pc_on_mar_page_bus, pc_on_mar_offset_bus,
    mar_on_abus : qit;

```

BEGIN

```

    ir : reg PORT MAP (instruct, ir_lines, load_ir, clk);
    row : reg PORT MAP (ir_lines, rc_lines(15 DOWNT0 8), load_row, clk);
    col : reg PORT MAP (ir_lines, rc_lines(7 DOWNT0 0), load_col, clk);

```

```

rc_data : ts16 PORT MAP (rc_lines, rc_on_rcdata_bus, r_c_data);
buff : reg PORT MAP (d_buff, buff_d, load_buffer, clk);
d_to_buff_1 : dbus1 <= qit_vector (dbus);
d_to_buff_2 : ts8 PORT MAP (dbus1, dbus_on_buffer, d_buff);
buff_to_d_1 : ts8 PORT MAP (buff_d, buffer_on_dbus, dbus2);
buff_to_d_2 : dbus <= wired_byte (dbus2);
counter : pc PORT MAP (pc1_lines, increment_pc, reset_pc, clk);
pc_to_mar_page : ts8 PORT MAP (pc1_lines(15 DOWNT0 8), pc_on_mar_page_bus,
                                pc2_lines(15 DOWNT0 8));
pc_to_mar_offset : ts8 PORT MAP (pc1_lines(7 DOWNT0 0), pc_on_mar_offset_bus,
                                pc2_lines(7 DOWNT0 0));
memory_addr : mar PORT MAP (pc2_lines, mar1_lines, load_page_mar, load_offset_mar, clk);
mar_to_abus_1 : ts16 PORT MAP (mar1_lines, mar_on_abus, mar2_lines);
mar_to_abus_2 : abus <= mar2_lines;
compare : cprc PORT MAP (rc_lines, mar2_lines, cp_rc);
logic : l_ctrl
    PORT MAP ( clk, interrupt, mod_end, cp_rc, ir_lines,
               load_ir, load_row, load_col, load_buffer,
               increment_pc, reset_pc,
               load_page_mar, load_offset_mar,
               rc_on_rcdata_bus,
               dbus_on_buffer, buffer_on_dbus,
               dbus_on_databus, databus_on_dbus,
               pc_on_mar_page_bus, pc_on_mar_offset_bus,
               mar_on_abus,
               read_mem, write_mem, clear_mem,
               err_sm, en_dm,
               read_instruction, mod_begin, r_c,
               m1, m2, m3, m4, m5, m6, m7, m8 );
END structural;

```

รูปที่ 5.26 แสดงการบรรยายระดับโครงสร้างของวงจรควบคุม

5.3 สรุป (Summary)

วงจรควบคุมเป็นโมดูลหลักของระบบทำหน้าที่ในการรับคำสั่งจากภายนอกและควบคุมให้มีการปฏิบัติตามลำดับคำสั่งที่ได้รับ ภายในวงจรประกอบด้วยส่วนควบคุมหลักภายในและส่วนข้อมูลภายในส่วนควบคุมภายในแบ่งการปฏิบัติออกเป็น 4 สเตท แต่ละสเตทจะเริ่มปฏิบัติหน้าที่เมื่อได้รับการกระตุ้นจากสเตทอื่น ๆ ตามเงื่อนไขที่กำหนด อย่างไรก็ตามการควบคุมการทำงานของวงจรควบคุมมิได้รุกร้าเข้าไปควบคุมการทำงานของโมดูลอื่น ๆ ใดๆ เนื่องจากแต่ละโมดูลทำงานเป็นอิสระต่อกัน ดังนั้นแต่ละโมดูลจึงมีส่วนควบคุมการทำงานเป็นของตนเอง หน้าที่ของวงจรควบคุมในส่วนที่เกี่ยวข้องกับโมดูลอื่น ๆ ก็เพียงแต่ทำหน้าที่ในการกระตุ้นและโอนย้ายการควบคุมให้กับโมดูลอื่น ๆ เป็นการชั่วคราวและจะรับมอบการควบคุมกลับเมื่อเสร็จสิ้นภารกิจของแต่ละโมดูลแล้ว

บทที่ 6

สถาปัตยกรรมของวงจรปรับปรุงคุณภาพของภาพ (Histogram Equalization Architecture)

ในการออกแบบสถาปัตยกรรมของวงจรปรับปรุงคุณภาพของภาพก็เช่นเดียวกับการออกแบบวงจรควบคุมที่ผ่านมา ซึ่งจะเริ่มต้นด้วยการกำหนดพฤติกรรมการทำงานของวงจรตามลักษณะแนวความคิดและวิธีที่ตั้งไว้ เมื่อทำการทดสอบจนเป็นที่พอใจแล้วจึงจะขยับเข้าไปให้ใกล้กับความเป็นฮาร์ดแวร์ให้มากยิ่งขึ้นโดยการกำหนดการทำงานในระดับข้อมูลไปจนถึงระดับโครงสร้าง หลังจากทดสอบเรียบร้อยแล้วก็จะได้วงจรที่มีลักษณะการทำงานเหมือนกับวงจรฮาร์ดแวร์จริงๆ

6.1 พฤติกรรมของวงจรปรับปรุงคุณภาพของภาพ

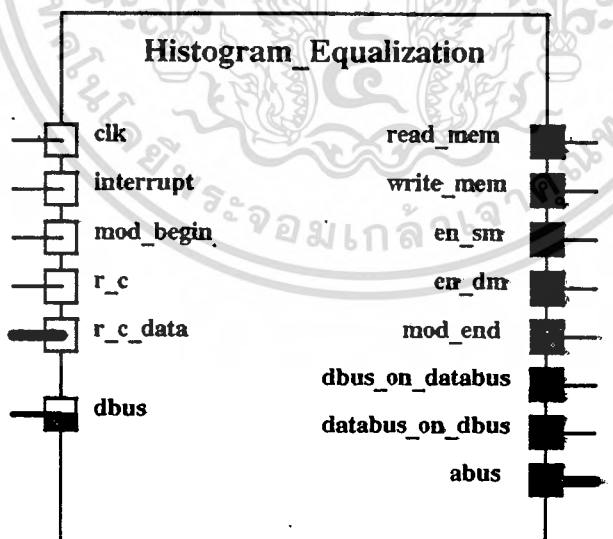
(Behavioral of Histogram Equalization)

พฤติกรรมการทำงานของวงจรปรับปรุงคุณภาพของภาพเริ่มต้นด้วยการอ่านข้อมูลของภาพต้นแบบจากหน่วยความจำต้นทางมาทีละจุดภาพ นำมาผ่านขั้นตอนการประมวลผลภายใน เสร็จแล้วก็จะเขียนข้อมูลภาพที่ได้ใหม่ลงในหน่วยความจำปลายทางเพื่อนำผลที่ได้ไปใช้ในงานอื่นๆต่อไป

6.1.1 การบรรยายการเชื่อมต่อของวงจรปรับปรุงคุณภาพของภาพ

(Interface Description of Histogram Equalization)

รูปที่ 6.1 แสดงสัญลักษณ์และการบรรยายการเชื่อมต่อของวงจรปรับปรุงคุณภาพของภาพ ส่วนของ *GENERIC* กำหนดเวลาที่ใช้ในการติดต่อกับหน่วยความจำ ส่วนของ *PORT* กำหนดส่วนการติดต่อภายนอกที่เป็น



```
LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;
```

```

ENTITY histogram_equalization IS
  GENERIC ( read_high_time, read_low_time,
            write_high_time, write_low_time : TIME := 2 US;
            cycle_time : TIME := 4 US);
  PORT ( clk, interrupt, r_c, mod_begin : IN qit;
        r_c_data : IN word;
        dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
        read_mem, write_mem : OUT qit;
        en_sm, en_dm, mod_end : OUT qit;
        abus : OUT word;
        dbus_on_databus, databus_on_dbus : OUT qit);
END histogram_equalization;

```

รูปที่ 6.1 แสดงสัญลักษณ์และการบรรยายการเชื่อมต่อของวงจรปรับปรุงคุณภาพของภาพ

6.1.2 การบรรยายการทำงานในระดับพฤติกรรมของวงจรปรับปรุง

คุณภาพของภาพ (Histogram Equalization Behavioral Architecture)

รูปที่ 6.2 แสดงโครงสร้างการบรรยายในระดับพฤติกรรม การทำงานจะเริ่มเมื่อได้รับสัญญาณกระตุ้นจากวงจรควบคุมคือ *interrupt* เป็น '1'

```

ARCHITECTURE behavioral OF histogram_equalization IS
BEGIN
  PROCESS
    Declare necessary variables; รูปที่ 6.3
  BEGIN
    IF interrupt = '1' THEN
      Handle interrupt; รูปที่ 6.4
      IF r_c = '1' THEN
        Read number of image rows and cols; รูปที่ 6.5
      ELSIF mod_begin = '1' THEN
        Read the grey level of each pixel; รูปที่ 6.6
        Compare and count the number of pixels in each level; รูปที่ 6.7
        Find new grey level; รูปที่ 6.8
        Change to new grey level; รูปที่ 6.9
        Compare and send the new grey level of each pixels to memory; รูปที่ 6.9
        Send finished signal; รูปที่ 6.10
      END IF;
    END IF;
  END PROCESS;
END behavioral;

```

รูปที่ 6.2 แสดงโครงสร้างการบรรยายในระดับพฤติกรรม

รูปที่ 6.3 แสดงการกำหนดตัวแปรที่จำเป็น *qit16_array* และ *qit8_array* เป็นชนิดของข้อมูลแบบอะเรย์ที่กำหนดขึ้นมาใหม่มีขนาด 256X16 และ 256X8 ตามลำดับ เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

TYPE qit16_array IS ARRAY ( 0 TO 255 ) OF word;
TYPE qit8_array IS ARRAY ( 0 TO 255 ) OF byte;
VARIABLE nrow, ncol : INTEGER;
VARIABLE pg, offt : INTEGER := 0;
VARIABLE page, offset, buff : byte;
VARIABLE rc15_8, rc7_0 : byte;
VARIABLE pixel_count : qit16_array;
VARIABLE new_level : qit8_array;
VARIABLE newj1, pix15_8, newj : INTEGER;

```

รูปที่ 6.3 แสดงการกำหนดตัวแปรที่จำเป็น

รูปที่ 6.4 แสดงการเซตค่าตัวแปรเริ่มต้นเมื่อ *interrupt* เป็น '1'

```

page := zero_8;
offset := zero_8;
buff := zero_8;

```

รูปที่ 6.4 แสดงการเซตค่าตัวแปร

เมื่อได้รับสัญญาณ *r_c* เป็น '1' วงจรจะทำการรับข้อมูลแสดงขนาดของภาพที่ส่งมาจาก วงจรควบคุมหลักไปเก็บไว้ที่ตัวแปร *nrow* และ *ncol* โดยต้องเปลี่ยนชนิดของข้อมูลจาก *qit* เป็น *integer* เสียก่อนโดยใช้ฟังก์ชัน *qit2int()* ดังรูปที่ 6.5

```

rc15_8 := r_c_data(15 DOWNT0 8);
rc7_0 := r_c_data(7 DOWNT0 0);
qit2int(rc15_8, nrow);
qit2int(rc7_0, ncol);

```

รูปที่ 6.5 แสดงการรับข้อมูลขนาดภาพ

```

--read the grey level of each pixel
WHILE pg <= nrow LOOP
  WHILE offt <= ncol LOOP
    en_sm <= '1';
    en_dm <= '0';
    int2qit(pg, page);
    int2qit(offt, offset);
    abus <= page & offset;
    read_mem <= '1'; WAIT FOR read_high_time;
    buff := byte(dbus);
    read_mem <= '0'; WAIT FOR read_low_time;
  
```

รูปที่ 6.6 แสดงขั้นตอนในการอ่านข้อมูลภาพจากหน่วยความจำต้นทางทีละจุดภาพ

เมื่ออ่านข้อมูลภาพเข้ามาแล้วก็จะนำมาเปรียบเทียบกับระดับความเข้มเป็นเท่าใดและทำการนับจำนวนจุดภาพที่ระดับความเข้มนั้น ๆ เก็บไว้ที่ตัวแปร *pixel_count()* ดังรูปที่ 6.7

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CASE buff IS

WHEN "00000000" => pixel_count(0) := inc(pixel_count(0));

WHEN "00000001" => pixel_count(1) := inc(pixel_count(1));

WHEN "00000010" => pixel_count(2) := inc(pixel_count(2));

.

WHEN "11111110" => pixel_count(254) := inc(pixel_count(254));

WHEN OTHERS => pixel_count(255) := inc(pixel_count(255));

END CASE;

offt := offt + 1;

END LOOP;

offt := 0;

pg := pg + 1;

END LOOP;

รูปที่ 6.7 แสดงการเปรียบเทียบและนับจำนวนจุดภาพ

รูปที่ 6.8 แสดงการหาระดับความเข้มใหม่ โดยนำเอาข้อมูลเฉพาะ 8 บิตบนของ pixel_count ที่แต่ละระดับความเข้มมารวมกันเป็นค่าระดับความเข้มใหม่ การนำเอาข้อมูลเฉพาะ 8 บิตบนมาใช้เท่านั้นเป็นขั้นตอนที่ใช้ในการประมาณค่าแทนการหารด้วยจำนวนจุดภาพทั้งหมด(สำหรับวิธีการของการปรับปรุงคุณภาพของภาพดูได้จากบทที่ 2)

--find new grey level

new_level(0) := pixel_count(0)(15 DOWNT0 8);

FOR j IN 1 TO 255 LOOP

qit2int (new_level(j-1), newj1);

qit2int (pixel_count(j)(15 DOWNT0 8), pix15_8);

newj := newj1 + pix15_8;

int2qit (newj, new_level(j));

END LOOP;

รูปที่ 6.8 แสดงการหาระดับความเข้มใหม่

รูปที่ 6.9 แสดงการเปลี่ยนค่าระดับความเข้มเก่าเป็นค่าระดับความเข้มใหม่โดยการอ่านค่าระดับความเข้มเก่าจากหน่วยความจำต้นทางเข้ามาเปรียบเทียบกับค่าระดับความเข้มใดก็ได้ให้ส่งค่าระดับความเข้มใหม่ที่สัมพันธ์กันไปยังหน่วยความจำปลายทางที่ตำแหน่งหน่วยความจำเดียวกัน หลังจากเสร็จสิ้นขั้นตอนทั้งหมดแล้วก็จะได้ข้อมูลภาพที่มีขนาดเท่ากันแต่มีค่าระดับความเข้มที่เปลี่ยนไป

--change to new grey level

pg := 0;

offt := 0;

WHILE pg <= nrow LOOP

WHILE offt <= ncol LOOP

en_sm <= '1';

en_dm <= '0';

int2qit(pg, page);

```

int2qit(offt, offset);
abus <= page & offset;
read_mem <= '1'; WAIT FOR read_high_time;
buff := byte(dbus);
read_mem <= '0'; WAIT FOR read_low_time;
en_sm <= '0';
en_dm <= '1';

--compare and send the new grey level of each pixels to memory
CASE buff IS
  WHEN "00000000" => dbus <= wired_byte(new_level(0));
  WHEN "00000001" => dbus <= wired_byte(new_level(1));
  WHEN "00000010" => dbus <= wired_byte(new_level(2));
  .
  .
  .
  WHEN "11111101" => dbus <= wired_byte(new_level(253));
  WHEN "11111110" => dbus <= wired_byte(new_level(254));
  WHEN OTHERS => dbus <= wired_byte(new_level(255));
END CASE;
offt := offt + 1;
END LOOP;
offt := 0;
pg := pg + 1;
END LOOP;

```

รูปที่ 6.9 แสดงการเปลี่ยนค่าระดับความเข้มเก่าเป็นค่าระดับความเข้มใหม่

เมื่อทำการประมวลผลภาพเสร็จเรียบร้อยแล้ววงจรก็จะส่งสัญญาณแจ้งให้วงจรควบคุมหลักรู้และโอนมอบการควบคุมกลับสู่วงจรควบคุมหลักอีกครั้งหนึ่ง ดังรูปที่ 6.10.

```

mod_end <= '1';
WAIT FOR cycle_time;
mod_end <= '0';

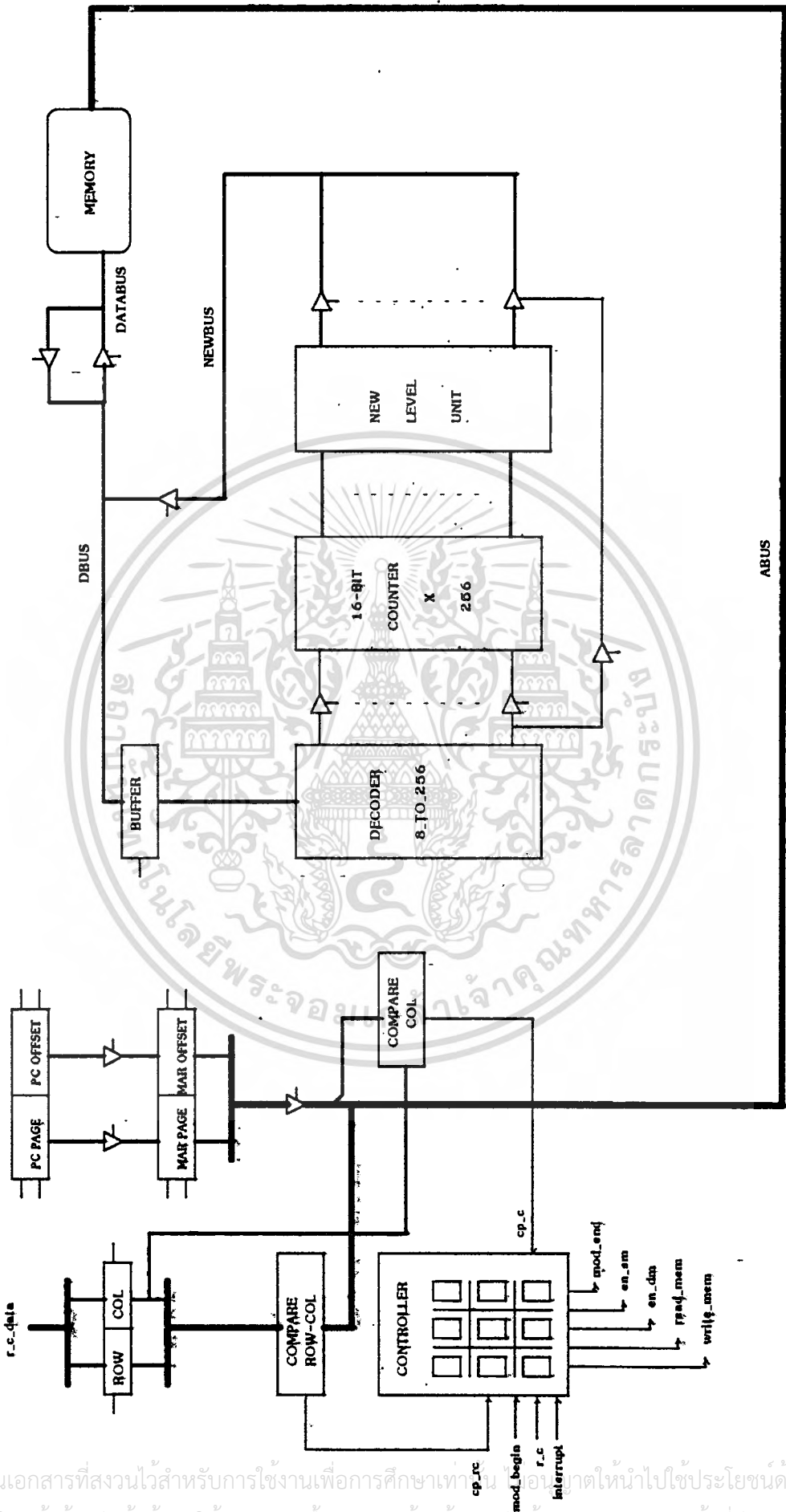
```

รูปที่ 6.10 แสดงการโอนมอบการควบคุมกลับสู่วงจรควบคุมหลัก

6.2 โครงสร้างของวงจรปรับปรุงคุณภาพของภาพ (Structural of Histogram Equalization)

โครงสร้างของวงจรปรับปรุงคุณภาพของภาพประกอบด้วยส่วนสำคัญ 2 ส่วนคือส่วนควบคุมทำหน้าที่ควบคุมการประมวลผลภายในวงจรทั้งหมด และส่วนข้อมูลประกอบด้วยรีจิสเตอร์ที่ใช้ในการเก็บข้อมูลต่าง ๆ รูปที่ 6.11 แสดงไดอะแกรมโครงสร้างบัสของวงจรทั้งหมด

6.2.1 โครงสร้างบัส (Histogram Equalization Bussing Structure)



รูปที่ 6.11 แสดงโปรแกรมโครงสร้างบัสของวงจรทั้งหมด

6.2.2 การบรรยายองค์ประกอบ (Description of Components)

ในหัวข้อนี้จะได้กล่าวถึงการบรรยายการทำงานขององค์ประกอบต่างๆตามรูปที่ 6.11 และในลำดับสุดท้ายจะได้นำเอาองค์ประกอบทั้งหมดมาประกอบรวมกันในการบรรยายระดับโครงสร้าง

6.2.2.1 การบรรยายระดับข้อมูลของวงจรควบคุมภายใน

(Dataflow of Histogram Controller)

การบรรยายระดับข้อมูลของวงจรควบคุมภายในแบ่งการทำงานออกเป็นสเตจๆ รูปที่ 6.12 แสดงการบรรยายการเชื่อมต่อของวงจร

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY histogram_controller IS
  GENERIC ( read_delay, write_delay : TIME := 3 NS);
  PORT ( clk : IN qit;
        interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
        -- register control signals:
        load_row, load_col, load_buffer,
        increment_pc_page, increment_pc_offset,
        reset_pc_page, reset_pc_offset, reset_pxc,
        load_page_mar, load_offset_mar,
        -- bus connection control signals:
        databus_on_dbus, dbus_on_databus,
        newbus_on_dbus, dec_on_newbus, dec_on_counter,
        pc_on_mar_page_bus, pc_on_mar_offset_bus,
        mar_on_abus,
        -- memory control signals:
        read_mem, write_mem,
        en_sm, en_dm,
        -- other control signal:
        dec_enable, mod_end : OUT qit );
END histogram_controller;

```

รูปที่ 6.12 แสดงการบรรยายการเชื่อมต่อของวงจร histogram_controller

ในส่วนของการบรรยายการทำงานประกอบด้วย 3 สเตจ สัญญาณเอาต์พุตกำหนดให้เป็นแบบ *ored_qit* โดยเรียกใช้ฟังก์ชัน *oring* จากแพ็คเกจ *basic_utilities* รูปที่ 6.13 แสดงการกำหนดสัญญาณและการกำหนดค่าของสัญญาณให้กับเอาต์พุต

```

ARCHITECTURE dataflow OF histogram_controller IS
  -- oring is implied in the following signals (oi)
  SIGNAL load_row_oi, load_col_oi, load_buffer_oi,
        increment_pc_page_oi, increment_pc_offset_oi,
        reset_pc_page_oi, reset_pc_offset_oi, reset_pxc_oi,
        load_page_mar_oi, load_offset_mar_oi,

```

```

databus_on_dbus_oi, dbus_on_databus_oi,
newbus_on_dbus_oi, dec_on_newbus_oi, dec_on_counter_oi,
pc_on_mar_page_bus_oi, pc_on_mar_offset_bus_oi,
mar_on_abus_oi,
read_mem_oi, write_mem_oi,
en_sm_oi, en_dm_oi,
dec_enable_oi, mod_end_oi : ored_qit BUS;
SIGNAL s : ored_qit_vector (3 DOWNTO 1) REGISTER := '001';
BEGIN -- start of histogram
load_row <= load_row_oi;
load_col <= load_col_oi;
load_buffer <= load_buffer_oi;
increment_pc_page <= increment_pc_page_oi
increment_pc_offset <= increment_pc_offset_oi
reset_pc_page <= reset_pc_page_oi;
reset_pc_offset <= reset_pc_offset_oi;
reset_pxc <= reset_pxc_oi;
load_page_mar <= load_page_mar_oi;
load_offset_mar <= load_offset_mar_oi;
databus_on_dbus <= databus_on_dbus_oi;
dbus_on_databus <= dbus_on_databus_oi;
newbus_on_dbus <= newbus_on_dbus_oi;
dec_on_newbus <= dec_on_newbus_oi;
dec_on_counter <= dec_on_counter_oi;
pc_on_mar_page_bus <= pc_on_mar_page_bus_oi;
pc_on_mar_offset_bus <= pc_on_mar_offset_bus_oi;
mar_on_abus <= mar_on_abus_oi;
read_mem <= read_mem_oi;
write_mem <= write_mem_oi;
en_sm <= en_sm_oi;
en_dm <= en_dm_oi;
dec_enable <= dec_enable_oi;
mod_end <= mod_end_oi;

```

รูปที่ 6.13 แสดงการกำหนดสัญญาณและการกำหนดค่าของสัญญาณให้กับเอาต์พุต

รูปที่ 6.14 แสดงการทำงานในสเตทที่ 1 เมื่อสัญญาณอินพุต *interrupt* เป็น '1' ซึ่งเป็นสัญญาณที่กำหนดให้โมดูลใดก็ได้ที่ได้รับเริ่มการปฏิบัติงาน สัญญาณอินพุต *r_c* เป็น '1' กำหนดให้วงจรรับค่าขนาด *row* และ *column* ซึ่งเป็นขนาดของข้อมูลภาพ สเตทจะเปลี่ยนสถานะเข้าสู่สเตทที่ 2 เมื่อสัญญาณอินพุต *mod_begin* เป็น '1'

```

s1 : BLOCK (s(1) = '1')
BEGIN
-- pc to mar
pc_on_mar_page_bus_oi <= GUARDED '1';
pc_on_mar_offset_bus_oi <= GUARDED '1';
load_page_mar_oi <= GUARDED '1';

```

```

load_offset_mar_oi <= GUARDED '1';
-- reset pc and pxc if no interrupt
reset_pc_page_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
reset_pc_offset_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
reset_pxc_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
int : BLOCK (interrupt = '1' AND GUARD)
BEGIN -- load rows and cols
load_row_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
load_col_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
-- goto 2 if module begin
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
s(1) <= GUARDED '1' WHEN mod_begin = '0' ELSE '0';
s(2) <= GUARDED '1' WHEN mod_begin /= '0' ELSE '0';
END BLOCK ck;
END BLOCK int;
END BLOCK s1;

```

รูปที่ 6.14 แสดงการทำงานในสเตทที่ 1

รูปที่ 6.15 แสดงการทำงานในสเตทที่ 2 เริ่มต้นด้วยการอ่านข้อมูลภาพจากหน่วยความจำต้นทางเข้ามาที่ละจุดภาพ ข้อมูลที่เข้ามาจะถูกส่งผ่านไปยังวงจร *decoder* เพื่อทำการเปรียบเทียบระดับความเข้มและจะส่งสัญญาณต่อไปยังวงจร *counter* เพื่อบันทึกจำนวนจุดภาพที่ระดับความเข้มนั้นๆ การอ่านข้อมูลภาพจะอ่านทีละคอลัมน์ในแต่ละแถว

```

s2 : BLOCK (s(2) = '1')
BEGIN -- count the number of pixels in each level
-- pc to mar
pc_on_mar_page_bus_oi <= GUARDED '1';
pc_on_mar_offset_bus_oi <= GUARDED '1';
load_page_mar_oi <= GUARDED '1';
load_offset_mar_oi <= GUARDED '1';
en_sm_oi <= GUARDED '1';
en_dm_oi <= GUARDED '0';
mar_on_abus_oi <= GUARDED '1';
read_mem_oi <= GUARDED '1' AFTER read_delay;
databus_on_dbus_oi <= GUARDED '1';
load_buffer_oi <= GUARDED '1';
dec_enable_oi <= GUARDED '1';
dec_on_counter_oi <= GUARDED '1';
chk : BLOCK (cp_rc /= '1' AND GUARD)
BEGIN
-- increment pc offset
increment_pc_offset_oi <= GUARDED '1' WHEN cp_c /= '1';
-- increment pc page and reset pc offset
increment_pc_page_oi <= GUARDED '1' WHEN cp_c = '1';
reset_pc_offset_oi <= GUARDED '1' WHEN cp_c = '1';

```

```

ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(2) <= GUARDED '1';
END BLOCK ck;
END BLOCK chk;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(3) <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
    reset_pc_page_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
    reset_pc_offset_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
END BLOCK ck;
END BLOCK s2;

```

รูปที่ 6.15 แสดงการทำงานในสเตทที่ 2

รูปที่ 6.16 แสดงการทำงานในสเตทที่ 3 ในสเตทนี้จะควบคุมการขั้นตอนการเปลี่ยนค่าระดับความเข้มใหม่โดยอ่านข้อมูลภาพจากหน่วยความจำต้นทางเข้ามาเปรียบเทียบกับระดับความเข้มอีกครั้งและส่งค่าระดับความเข้มใหม่เขียนลงในหน่วยความจำปลายทางในตำแหน่งที่สัมพันธ์กัน

```

s3 : BLOCK (s(3) = '1')
BEGIN -- change to new level
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';
    load_page_mar_oi <= GUARDED '1';
    load_offset_mar_oi <= GUARDED '1';
    -- read from source memory
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    mar_on_abus_oi <= GUARDED '1';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    load_buffer_oi <= GUARDED '1';
    dec_enable_oi <= GUARDED '1';
    dec_on_newbus_oi <= GUARDED '1';
    wck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        -- write to destination memory
        en_sm_oi <= GUARDED '0';
        en_dm_oi <= GUARDED '1';
        mar_on_abus_oi <= GUARDED '1';
        newbus_on_dbus_oi <= GUARDED '1';
        dbus_on_databus_oi <= GUARDED '1';
        write_mem_oi <= GUARDED '1' AFTER write_delay;
        chk : BLOCK (cp_rc /= '1' AND GUARD)
        BEGIN
            -- increment pc offset
            increment_pc_offset_oi <= GUARDED '1' WHEN cp_c /= '1';

```

เอกสารนี้เป็นเอกสารที่จัดทำขึ้นเพื่อใช้ในการศึกษาและวิจัยเท่านั้น ไม่ควรนำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

-- increment pc page and reset pc offset
increment_pc_page_oi <= GUARDED '1' WHEN cp_c = '1';
reset_pc_offset_oi <= GUARDED '1' WHEN cp_c = '1';
cck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(3) <= GUARDED '1';
END BLOCK cck;
END BLOCK chk;
END BLOCK wck;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    mod_end_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
END BLOCK ck;
END BLOCK s3;

```

รูปที่ 6.16 แสดงการทำงานในสเตทที่ 3

รูปที่ 6.17 แสดงการรีเซ็ตสถานะของสเตทในทุกๆครั้งที่มีการเปลี่ยนสถานะจากสเตทหนึ่งไปยังอีกสเตทหนึ่ง

```

ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(3 DOWNT0 1) <= GUARDED '000';
END BLOCK ck;
END dataflow;

```

รูปที่ 6.17 แสดงการรีเซ็ตสถานะของสเตท

6.2.2.2 โครงร่างของวงจรควบคุมภายใน

(General Outline of Histogram Controller)

จากการบรรยายในระดับข้อมูลของวงจรควบคุมนำมาใช้เป็นแนวทางในการออกแบบโครงสร้างได้ดังรูปที่ 6.18 ฟลิปฟล็อป master-slave D-flipflop ถูกใช้เพื่อเป็นตัวเซตสถานะของสเตทและบิตเวิลถูกใช้แทนลอจิกการทำงานในแต่ละสเตท

6.2.2.3 การบรรยายระดับโครงสร้างของวงจรควบคุมภายใน

(Structural of Histogram Controller)

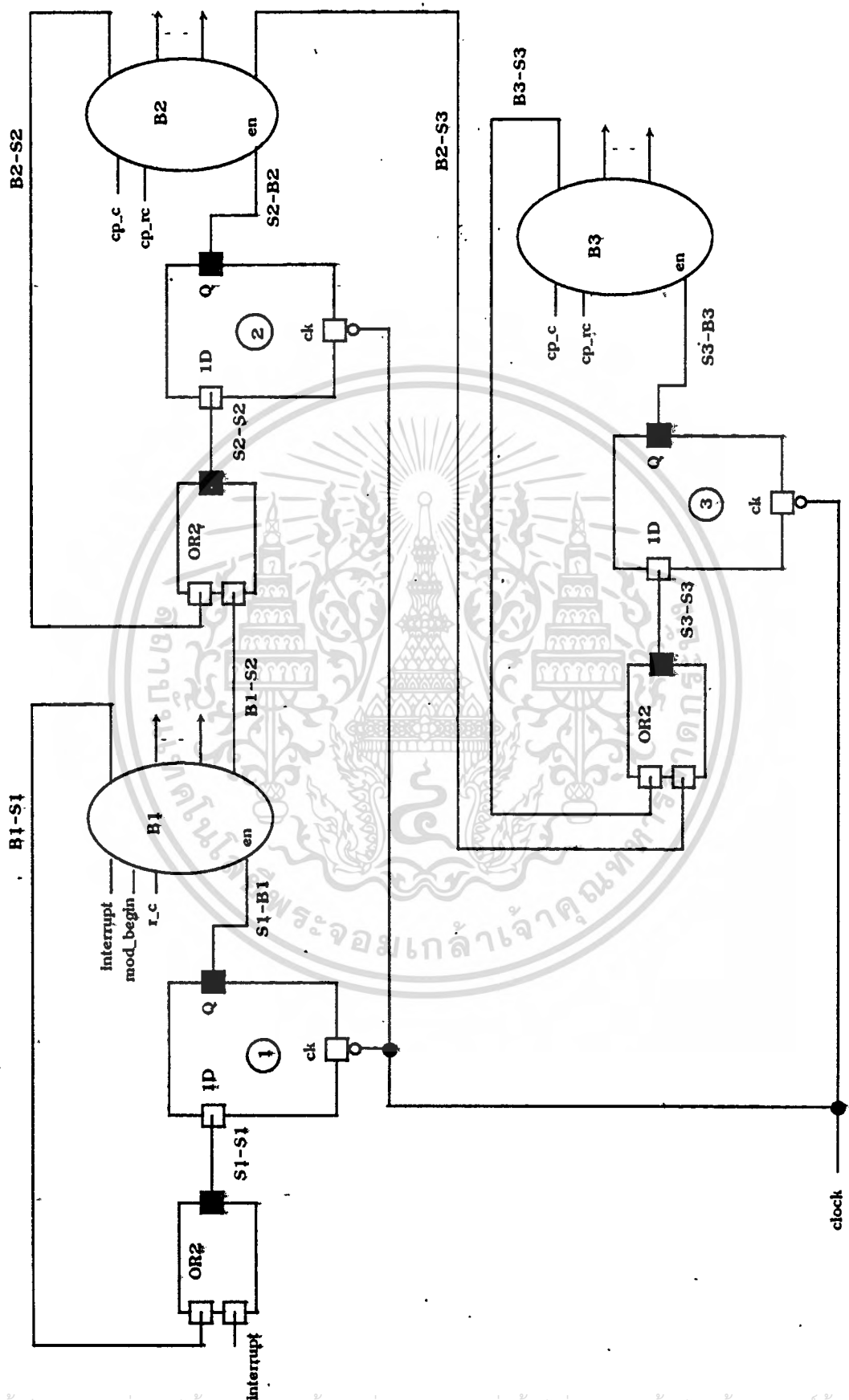
แต่ละสเตทในรูปที่ 6.18 จะถูกนำมาออกแบบแยกเป็นแต่ละองค์ประกอบเพื่อให้ง่ายและสะดวกในการออกแบบ รูปที่ 6.19 แสดงการบรรยายระดับโครงสร้างโดยนำแต่ละองค์ประกอบมารวมกันเป็นโครงสร้างทางฮาร์ดแวร์ที่สมบูรณ์

ARCHITECTURE structural OF histogram_controller IS

```

COMPONENT hs1 PORT ( s1, int, clk, r_c, mod_begin : IN qit;
    b1_load_page_mar, b1_load_offset_mar, b1_pc_on_mar_page_bus,
    b1_pc_on_mar_offset_bus, b1_reset_pc_page, b1_reset_pc_offset,
    b1_reset_pxc, b1_load_row, b1_load_col, b1_s1, b1_s2 : OUT qit);
END COMPONENT;

```



รูปที่ 6.18 แสดงโครงสร้างของวงจรควบคุมภายใน

COMPONENT hs2

```

PORT ( s2, clk, cp_rc, cp_c : IN qit;
      b2_load_page_mar, b2_load_offset_mar,
      b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
      b2_mar_on_abus, b2_en_sm, b2_en_dm,
      b2_load_buffer, b2_databus_on_dbus,
      b2_dec_on_counter, b2_read_mem, b2_dec_enable,
      b2_incrememt_pc_page, b2_incrememt_pc_offset,
      b2_reset_pc_page, b2_reset_pc_offset,
      b2_s2, b2_s3 : OUT qit);

```

END COMPONENT;

COMPONENT hs3

```

PORT ( s3, clk, cp_rc, cp_c : IN qit;
      b3_load_page_mar, b3_load_offset_mar,
      b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus,
      b3_mar_on_abus, b3_en_sm, b3_en_dm,
      b3_load_buffer, b3_databus_on_dbus,
      b3_dbus_on_databus, b3_newbus_on_dbus,
      b3_dec_on_newbus, b3_dec_enable,
      b3_read_mem, b3_write_mem,
      b3_incrememt_pc_page, b3_incrememt_pc_offset,
      b3_reset_pc_offset, b3_mod_end, b3_s3 : OUT qit);

```

END COMPONENT;

FOR s1 : hs1 USE ENTITY hgram.state_hctrl_1 (structural);

FOR s2 : hs2 USE ENTITY hgram.state_hctrl_2 (structural);

FOR s3 : hs3 USE ENTITY hgram.state_hctrl_3 (structural);

FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);

FOR ALL : o3 USE ENTITY gates.or3_q (average_delay);

SIGNAL b2_load_buffer, b3_load_buffer : qit;

```

SIGNAL b2_incrememt_pc_page, b2_incrememt_pc_offset,
      b3_incrememt_pc_page, b3_incrememt_pc_offset : qit;

```

SIGNAL b1_reset_pc_offset, b2_reset_pc_offset, b3_reset_pc_offset : qit;

SIGNAL b1_reset_pc_page, b2_reset_pc_page : qit;

```

SIGNAL b1_load_page_mar, b1_load_offset_mar,
      b2_load_page_mar, b2_load_offset_mar,
      b3_load_page_mar, b3_load_offset_mar : qit;

```

SIGNAL b2_databus_on_dbus, b3_databus_on_dbus : qit;

```

SIGNAL b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
      b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
      b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus : qit;

```

SIGNAL b2_mar_on_abus, b3_mar_on_abus : qit;

SIGNAL b2_read_mem, b3_read_mem : qit;

SIGNAL b2_en_sm, b2_en_dm, b3_en_sm, b3_en_dm : qit;

SIGNAL b2_dec_enable, b3_dec_enable : qit;

SIGNAL s1_s1, s2_s2, s3_s3 : qit;

SIGNAL b1_s1, b1_s2, b2_s2, b2_s3, b3_s3 : qit;

BEGIN เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์การค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

g0 : o2 PORT MAP (interrupt, b1_s1, s1_s1);
s1 : hs1 PORT MAP (s1_s1, interrupt, clk, r_c, mod_begin,
    b1_load_page_mar, b1_load_offset_mar,
    b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
    b1_reset_pc_page, b1_reset_pc_offset, reset_pxc,
    load_row, load_col, b1_s1, b1_s2);
g1 : o2 PORT MAP (b1_s2, b2_s2, s2_s2);
s2 : hs2 PORT MAP (s2_s2, clk, cp_rc, cp_c,
    b2_load_page_mar, b2_load_offset_mar,
    b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
    b2_mar_on_abus, b2_en_sm, b2_en_dm,
    b2_load_buffer, b2_databus_on_dbus,
    dec_on_counter, b2_read_mem, b2_dec_enable,
    b2_incrememt_pc_page, b2_incrememt_pc_offset,
    b2_reset_pc_page, b2_reset_pc_offset,
    b2_s2, b2_s3);
g2 : o2 PORT MAP (b2_s3, b3_s3, s3_s3);
s3 : hs3 PORT MAP (s3_s3, clk, cp_rc, cp_c,
    b3_load_page_mar, b3_load_offset_mar,
    b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus,
    b3_mar_on_abus, b3_en_sm, b3_en_dm,
    b3_load_buffer, b3_databus_on_dbus,
    dbus_on_databus, newbus_on_dbus,
    dec_on_newbus, b3_dec_enable,
    b3_read_mem, write_mem,
    b3_incrememt_pc_page, b3_incrememt_pc_offset,
    b3_reset_pc_offset, mod_end, b3_s3);
g3 : o2 PORT MAP (b2_load_buffer, b3_load_buffer, load_buffer);
g4 : o2 PORT MAP (b2_incrememt_pc_page, b3_incrememt_pc_page, incrememt_pc_page);
g5 : o2 PORT MAP (b2_incrememt_pc_offset, b3_incrememt_pc_offset, incrememt_pc_offset);
g6 : o3 PORT MAP (b1_reset_pc_offset, b2_reset_pc_offset, b3_reset_pc_offset, reset_pc_offset);
g7 : o2 PORT MAP (b1_reset_pc_page, b2_reset_pc_page, reset_pc_page);
g8 : o3 PORT MAP (b1_load_page_mar, b2_load_page_mar, b3_load_page_mar, load_page_mar);
g9 : o3 PORT MAP (b1_load_offset_mar, b2_load_offset_mar, b3_load_offset_mar, load_offset_mar);
g10 : o2 PORT MAP (b2_databus_on_dbus, b3_databus_on_dbus, databus_on_dbus);
g11 : o3 PORT MAP (b1_pc_on_mar_page_bus, b2_pc_on_mar_page_bus,
    b3_pc_on_mar_page_bus, pc_on_mar_page_bus);
g12 : o3 PORT MAP (b1_pc_on_mar_offset_bus, b2_pc_on_mar_offset_bus,
    b3_pc_on_mar_offset_bus, pc_on_mar_offset_bus);
g13 : o2 PORT MAP (b2_mar_on_abus, b3_mar_on_abus, mar_on_abus);
g14 : o2 PORT MAP (b2_read_mem, b3_read_mem, read_mem);
g15 : o2 PORT MAP (b2_en_sm, b3_en_sm, en_sm);
g16 : o2 PORT MAP (b2_en_dm, b3_en_dm, en_dm);
g17 : o2 PORT MAP (b2_dec_enable, b3_dec_enable, dec_enable);
END structural;

```

รูปที่ 6.19 แสดงการบรรยายระดับโครงสร้างของวงจรควบคุมภายใน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้ภายในเท่านั้น ไม่ควรเผยแพร่ไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 6.20-6.22 แสดงการบรรยายระดับโครงสร้างของวงจรควบคุมแต่ละสเตจซึ่งมีลำดับการปฏิบัติเป็นไปตามการบรรยายการทำงานในระดับข้อมูลที่ผ่านมา

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY state_hctrl_1 IS
    PORT ( s1, int, clk, r_c, mod_begin : IN qit;
          b1_load_page_mar, b1_load_offset_mar,
          b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
          b1_reset_pc_page, b1_reset_pc_offset, b1_reset_pxc,
          b1_load_row, b1_load_col,
          b1_s1, b1_s2 : OUT qit);
END state_hctrl_1;
ARCHITECTURE structural OF state_hctrl_1 IS
    FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
    FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
    FOR ALL : a4 USE ENTITY gates.and4_q (average_delay);
    SIGNAL inv_clk, inv_int, inv_mod_begin : qit;
    SIGNAL im1, im2 : qit;
    SIGNAL s1_b1 : qit;
BEGIN
    control_signal : msdff PORT MAP (s1, clk, s1_b1);
    g0 : inv PORT MAP (clk, inv_clk);
    g1 : inv PORT MAP (int, inv_int);
    g2 : inv PORT MAP (mod_begin, inv_mod_begin);
    g3 : b1_load_page_mar <= s1_b1;
    g4 : b1_load_offset_mar <= s1_b1;
    g5 : b1_pc_on_mar_page_bus <= s1_b1;
    g6 : b1_pc_on_mar_offset_bus <= s1_b1;
    g7 : a2 PORT MAP (s1_b1, inv_int, im1);
    g8 : b1_reset_pc_page <= im1;
    g9 : b1_reset_pc_offset <= im1;
    g10 : b1_reset_pxc <= im1;
    g11 : a3 PORT MAP (s1_b1, int, r_c, im2);
    g12 : b1_load_row <= im2;
    g13 : b1_load_col <= im2;
    g14 : a4 PORT MAP (inv_clk, s1_b1, int, inv_mod_begin, b1_s1);
    g15 : a4 PORT MAP (inv_clk, s1_b1, int, mod_begin, b1_s2);
END structural;

```

รูปที่ 6.20 แสดงการบรรยายวงจร state_hctrl_1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_hctrl_2 IS
    PORT ( s2, clk, cp_rc, cp_c : IN qit;
          b2_load_page_mar, b2_load_offset_mar,
          b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
          b2_mar_on_abus, b2_eri_sm, b2_en_dm,
          b2_load_buffer, b2_databus_on_dbus,
          b2_dec_on_counter, b2_read_mem, b2_dec_enable,
          b2_increment_pc_page, b2_increment_pc_offset,
          b2_reset_pc_page, b2_reset_pc_offset,
          b2_s2, b2_s3 : OUT qit);

```

```

END state_hctrl_2;

```

```

ARCHITECTURE structural OF state_hctrl_2 IS

```

```

    FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
    FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
    FOR ALL : delay USE ENTITY gates.signal_delay (structural);
    SIGNAL inv_clk, inv_cp_rc, inv_cp_c : qit;
    SIGNAL reset_pc_offset_1, reset_pc_offset_2 : qit;
    SIGNAL im1, im2 : qit;
    SIGNAL s2_b2 : qit;

```

```

BEGIN

```

```

    control_signal : msdff PORT MAP (s2, clk, s2_b2);
    g0 : inv PORT MAP (clk, inv_clk);
    g1 : inv PORT MAP (cp_rc, inv_cp_rc);
    g2 : inv PORT MAP (cp_c, inv_cp_c);
    g3 : b2_load_page_mar <= s2_b2;
    g4 : b2_load_offset_mar <= s2_b2;
    g5 : b2_pc_on_mar_page_bus <= s2_b2;
    g6 : b2_pc_on_mar_offset_bus <= s2_b2;
    g7 : b2_mar_on_abus <= s2_b2;
    g8 : b2_eri_sm <= s2_b2;
    g9 : inv PORT MAP (s2_b2, b2_en_dm);
    g10 : b2_load_buffer <= s2_b2;
    g11 : b2_databus_on_dbus <= s2_b2;
    g12 : b2_dec_on_counter <= s2_b2;
    g13 : delay PORT MAP (s2_b2, b2_read_mem);

```

```

g14 : b2_dec_enable <= s2_b2;
g15 : a3 PORT MAP (s2_b2, inv_cp_rc, inv_cp_c, b2_increment_pc_offset);
g16 : a3 PORT MAP (s2_b2, inv_cp_rc, cp_c, im1);
g17 : b2_increment_pc_page <= im1;
g18 : reset_pc_offset_1 <= im1;
g19 : a3 PORT MAP (s2_b2, inv_clk, inv_cp_rc, b2_s2);
g20 : a3 PORT MAP (s2_b2, inv_clk, cp_rc, im2);
g21 : b2_s3 <= im2;
g22 : b2_reset_pc_page <= im2;
g23 : reset_pc_offset_2 <= im2;
g24 : o2 PORT MAP (reset_pc_offset_1, reset_pc_offset_2, b2_reset_pc_offset);
END structural;

```

รูปที่ 6.21 แสดงการบรรยายวงจร state_hctrl_2

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_hctrl_3 IS
  PORT ( s3, clk, cp_rc, cp_c : IN qit;
         b3_load_page_mar, b3_load_offset_mar,
         b3_pc_on_mar_page_bus, b3_pc_on_mar_offset_bus,
         b3_mar_on_abus, b3_en_sm, b3_en_dm,
         b3_load_buffer, b3_databus_on_dbus,
         b3_dbus_on_databus, b3_newbus_on_dbus,
         b3_dec_on_newbus, b3_dec_enable,
         b3_read_mem, b3_write_mem,
         b3_incremmt_pc_page, b3_incremmt_pc_offset,
         b3_reset_pc_offset, b3_mod_end, b3_s3 : OUT qit);
END state_hctrl_3;

ARCHITECTURE structural OF state_hctrl_3 IS
  FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
  FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
  FOR ALL : a4 USE ENTITY gates.and4_q (average_delay);
  FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);
  FOR ALL : delay USE ENTITY gates.signal_delay (structural);
  SIGNAL inv_clk, inv_cp_rc, inv_cp_c : qit;
  SIGNAL im1, im2, im3 : qit;
  SIGNAL r_en_sm, r_en_dm : qit;

```

```

SIGNAL w_en_sm, w_en_dm : qit;
SIGNAL r_mar_on_abus, w_mar_on_abus : qit;
SIGNAL s3_b3 : qit;
BEGIN
control_signal : msdff PORT MAP (s3, clk, s3_b3);
g0 : inv PORT MAP (clk, inv_clk);
g1 : inv PORT MAP (cp_rc, inv_cp_rc);
g2 : inv PORT MAP (cp_c, inv_cp_c);
g3 : b3_load_page_mar <= s3_b3;
g4 : b3_load_offset_mar <= s3_b3;
g5 : b3_pc_on_mar_page_bus <= s3_b3;
g6 : b3_pc_on_mar_offset_bus <= s3_b3;
g7 : r_mar_on_abus <= s3_b3;
g8 : r_en_sm <= s3_b3;
g9 : inv PORT MAP (s3_b3, r_en_dm);
g10 : b3_load_buffer <= s3_b3;
g11 : b3_databus_on_dbus <= s3_b3;
g12 : b3_dec_on_newbus <= s3_b3;
g13 : delay PORT MAP (s3_b3, b3_read_mem);
g14 : b3_dec_enable <= s3_b3;
g15 : a2 PORT MAP (s3_b3, inv_clk, im1);
g16 : inv PORT MAP (im1, w_en_sm);
g17 : w_en_dm <= im1;
g18 : w_mar_on_abus <= im1;
g19 : b3_dbus_on_databus <= im1;
g20 : b3_newbus_on_dbus <= im1;
g21 : delay PORT MAP (im1, b3_write_mem);
g22 : a4 PORT MAP (s3_b3, inv_clk, inv_cp_rc, inv_cp_c, b3_increment_pc_offset);
g23 : a4 PORT MAP (s3_b3, inv_clk, inv_cp_rc, cp_c, im2);
g24 : b3_increment_pc_page <= im2;
g25 : b3_reset_pc_offset <= im2;
g26 : a3 PORT MAP (s3_b3, inv_clk, inv_cp_rc, im3);
g27 : a2 PORT MAP (im3, inv_clk, b3_s3);
g28 : a3 PORT MAP (s3_b3, inv_clk, cp_rc, b3_mod_end);
g29 : o2 PORT MAP (r_en_sm, w_en_sm, b3_en_sm);
g30 : o2 PORT MAP (r_en_dm, w_en_dm, b3_en_dm);
g31 : o2 PORT MAP (r_mar_on_abus, w_mar_on_abus, b3_mar_on_abus);
END structural;

```

รูปที่ 6.22 แสดงการบรรยายวงจร *state_hctrl_3*

6.2.2.4 วงจร *Tri_State_256*

วงจร *Tri_State_256* เป็นวงจรที่รวมเกท *tri_state* จำนวน 256 ตัวไว้ด้วยกัน ในการบรรยายใช้รูปแบบ GENERATE ในการอ้างอิง ดังรูปที่ 6.23

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_filpflop.ALL;

ENTITY tri_state_256 IS
    PORT ( i : IN qit_vector (0 TO 255); c : IN qit;
           o : OUT qit_vector (0 TO 255));
END tri_state_256;

ARCHITECTURE structural OF tri_state_256 IS
    FOR ALL : ts USE ENTITY gates.tri_state_q(single_delay);
BEGIN
    g0_to_255 : FOR k IN 0 TO 255 GENERATE
        g : ts PORT MAP (i(k), c, o(k));
    END GENERATE;
END structural;

```

รูปที่ 6.23 แสดงการบรรยายวงจร *Tri_State_256*

6.2.2.5 วงจร *Tri_State_8_256*

วงจร *Tri_State_8_256* เป็นวงจรที่รวมเกท *tri_state* ขนาด 8 บิตจำนวน 256 ตัวไว้ด้วยกัน ในการบรรยายใช้รูปแบบ GENERATE ในการอ้างอิง ดังรูปที่ 6.24

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;

ENTITY tri_state_8_256 IS
    PORT ( i : IN byte256;
           c : IN qit_vector (0 TO 255);
           o : OUT byte256);
END tri_state_8_256;

ARCHITECTURE structural OF tri_state_8_256 IS
    FOR ALL : ts8 USE ENTITY reg.tri_state_8 (structural);
BEGIN
    t0_to_255 : FOR k IN 0 TO 255 GENERATE
        t : ts8 PORT MAP (i(k), c(k), o(k));
    END GENERATE;
END structural;

```

รูปที่ 6.24 แสดงการบรรยายวงจร *Tri_State_8_256*

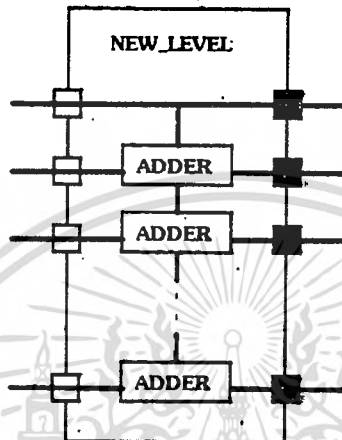
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.2.2.6 วงจร Decoder_8_to_256

วงจร *Decoder_8_to_256* ถูกออกแบบให้รับอินพุตซึ่งเป็นค่าระดับความเข้มของจุดภาพที่มีขนาด 8 บิตและส่งเอาต์พุตออกมาเพื่อกระตุ้นให้วงจรนับทำการนับจำนวนของจุดภาพที่ระดับความเข้มนั้นๆ ซึ่งมีระดับความเข้มที่แตกต่างกัน 256 ระดับ การบรรยายการทำงานของวงจรได้กล่าวถึงไปแล้วในบทที่ 4 รูปที่ 4.53

6.2.2.7 วงจร New_Level Unit

วงจร *New_Level Unit* เป็นวงจรสำหรับคำนวณหาค่าระดับความเข้มใหม่



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
ENTITY new_level IS
  PORT ( i8 : IN byte256; o8 : OUT byt256);
END new_level;
ARCHITECTURE structural OF new_level IS
  FOR ALL : add USE ENTITY reg.adder_8(behavioral);
  SIGNAL im : byte256;
BEGIN
  im_to_o8 : FOR i IN 0 TO 255 GENERATE
    o8(i) <= im(i);
  END GENERATE;
  new0_to_255 : FOR k IN 0 TO 255 GENERATE
    IF k = 0 GENERATE
      new0 : im(k) <= i8(k);
    END GENERATE;
    IF k > 0 GENERATE
      new : add PORT MAP ( im(k-1), i8(k), im(k));
    END GENERATE;
  END GENERATE;
END structural;

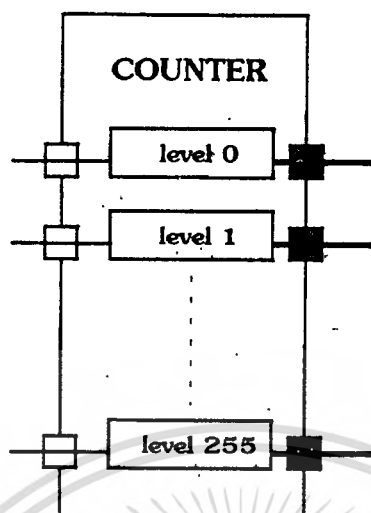
```

รูปที่ 6.25 แสดงการบรรยายวงจร *New_Level Unit*

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.2.2.8 วงจร Pixel_Counter Unit

วงจร *Pixel_Counter Unit* เป็นวงจรนับจำนวนจุดภาพที่ระดับความเข้มต่าง ๆ



```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
ENTITY pixel_counter IS
    PORT ( o8 : OUT byte256;
          inc : IN qit_vector(0 TO 255);
          reset, ck : IN qit);
END pixel_counter;

ARCHITECTURE structural OF pixel_counter IS
    FOR ALL : pc USE ENTITY reg.program_counter_unit(behavioral);
    SIGNAL ow : word256;
BEGIN
    pc_all : FOR i IN 0 TO 255 GENERATE
        pxc : pc PORT MAP ( ow(i), inc(i), reset, ck);
        o8(i) <= ow (i)(15 DOWNT0 8);
    END GENERATE;
END structural;

```

รูปที่ 6.26 แสดงการบรรยายวงจร *Pixel_Counter Unit*

6.2.3. การบรรยายระดับโครงสร้างของวงจรปรับปรุงคุณภาพ

(Histogram Equalization Structural Architecture)

หลังจากที่ได้ออกแบบและตรวจสอบการทำงานขององค์ประกอบต่าง ๆ เรียบร้อยแล้วก็นำเอามาประกอบรวมกันเป็นวงจรฮาร์ดแวร์ที่สมบูรณ์ในระดับโครงสร้างดังรูปที่ 6.27

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY hgram;

ENTITY histogram_equalization IS
  PORT ( clk, interrupt, r_c, mod_begin : IN qit;
         r_c_data : IN word;
         dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
         read_mem, write_mem : OUT qit;
         en_sm, en_dm, mod_end : OUT qit;
         abus : OUT word;
         dbus_on_databus, databus_on_dbus : OUT qit);
END histogram_equalization;

```

```

ARCHITECTURE structural OF histogram_equalization IS

```

```

  COMPONENT ctrl_unit

```

```

    PORT ( clk : IN qit;
           interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
           load_row, load_col, load_buffer,
           increment_pc_page, increment_pc_offset,
           reset_pc_page, reset_pc_offset, reset_pxc,
           load_page_mar, load_offset_mar,
           databus_on_dbus, dbus_on_databus,
           newbus_on_dbus, dec_on_newbus, dec_on_counter,
           pe_on_mar_page_bus, pc_on_mar_offset_bus,
           mar_on_abus,
           read_mem, write_mem,
           en_sm, en_dm,
           dec_enable, mod_end : OUT qit);

```

```

  END COMPONENT;

```

```

  COMPONENT t256

```

```

    PORT ( i : IN qit_vector (0 TO 255); c : IN qit;
           o : OUT qit_vector (0 TO 255));

```

```

  END COMPONENT;

```

```

  COMPONENT t8_256

```

```

    PORT ( i : IN byte256;
           c : IN qit_vector (0 TO 255);
           o : OUT byte256);

```

```

  END COMPONENT;

```

```

  COMPONENT decoder_unit

```

```

    PORT ( i8 : IN byte; en : IN qit;
           o : OUT qit_vector(0 TO 255));

```

```

  END COMPONENT;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับกรณีใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

COMPONENT pxc

PORT (o8 : OUT byte256;

inc : IN qit_vector(0 TO 255);

reset, ck : IN qit);

END COMPONENT;

COMPONENT new_gray_level

PORT (-i8 : IN byte256; o8 : OUT byte256);

END COMPONENT;

FOR control : ctrl_unit USE ENTITY hgram.histogram_controller(dataflow);

FOR pixel : pxc USE ENTITY hgram.pixel_counter(structural);

FOR dec : decoder_unit USE ENTITY hgram.decoder_8_to_256(dataflow);

FOR new_data : t8_256 USE ENTITY hgram.tri_state_8_256(structural);

FOR ALL : t256 USE ENTITY hgram.tri_state_256(structural);

FOR new_pixel : new_gray_level USE ENTITY hgram.new_level(structural);

FOR ALL : reg USE ENTITY reg.register_unit(dataflow);

FOR row_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);

FOR col_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);

FOR memory_addr : mar USE ENTITY reg.memory_address_register_unit(behavioral);

FOR compare_rc : cprc USE ENTITY reg.compare_row_col(dataflow);

FOR compare_c : cpc USE ENTITY reg.compare_col(dataflow);

FOR ALL : ts8 USE ENTITY reg.tri_state_8(structural);

FOR ALL : ts16 USE ENTITY reg.tri_state_16(structural);

SIGNAL cp_rc, cp_c : qit;

SIGNAL load_row, load_col, load_buffer,

increment_pc_page, increment_pc_offset,

reset_pc_page, reset_pc_offset, reset_pxc,

load_page_mar, load_offset_mar : qit;

SIGNAL newbus_on_dbus, new_level_on_newbus,

dec_enable, dec_on_newbus, dec_on_counter,

pc_on_mar_page_bus, pc_on_mar_offset_bus,

mar_on_abus : qit;

SIGNAL pc1_lines, pc2_lines, mar1_lines, mar2_lines : word;

SIGNAL rc_lines : word;

SIGNAL buff_lines, dbuff : byte;

SIGNAL dec1_lines, dec2_lines, dec3_lines : qit_vector(0 TO 255);

SIGNAL count_newlevel_lines : byte256;

SIGNAL new1_lines, new2_lines : byte256;

SIGNAL new_bus : wired_byte BUS;

SIGNAL newbus1_lines, newbus2_lines : byte;

BEGIN

row : reg PORT MAP (r_c_data(15 DOWNT0 8), rc_lines(15 DOWNT0 8), load_row, clk);

col : reg PORT MAP (r_c_data(7 DOWNT0 0), rc_lines(7 DOWNT0 0), load_col, clk);

compare_rc : cprc PORT MAP (rc_lines, mar2_lines, cp_rc);

compare_c : cpc PORT MAP (rc_lines(7 DOWNT0 0), mar2_lines(7 DOWNT0 0), cp_c);

row_counter : rcpc PORT MAP (pc1_lines(15 DOWNT0 8), increment_pc_page, reset_pc_page, clk);

col_counter : rcpc PORT MAP (pc1_lines(7 DOWNT0 0), increment_pc_offset, reset_pc_offset, clk);

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

pc_to_mar_page : ts8 PORT MAP (pc1_lines(15 DOWNT0 8), pc_on_mar_page_bus,
                                pc2_lines(15 DOWNT0 8));
pc_to_mar_offset : ts8 PORT MAP (pc1_lines(7 DOWNT0 0), pc_on_mar_page_bus,
                                pc2_lines(7 DOWNT0 0));
memory_addr : mar PORT MAP (pc2_lines, mar1_lines, load_page_mar, load_offset_mar, clk);
mar_to_abus_1 : ts16 PORT MAP (mar1_lines, mar_on_abus, mar2_lines);
mar_to_abus_2 : abus <= mar2_lines;
buffer_1 : dbuff <= qit_vector(dbus);
buffer_2 : reg PORT MAP (dbuff, buff_lines, load_buffer, clk);
dec : decoder_unit PORT MAP (buff_lines, dec_enable, dec1_lines);
dec_to_count : t256 PORT MAP (dec1_lines, dec_on_counter, dec2_lines);
pixel : pxc PORT MAP (count_newlevel_lines, dec2_lines, reset_pxc, clk);
new_pixel : new_gray_level PORT MAP (count_newlevel_lines, new1_lines);
dec_to_newbus : t256 PORT MAP (dec1_lines, dec_on_newbus, dec3_lines);
new_data_1 : t8_256 PORT MAP (new1_lines, dec3_lines, new2_lines);
new_data_2 : newbus <= wired_byte (new2_lines);
newbus_to_dbus_1 : newbus1_lines <= qit_vector (newbus);
newbus_to_dbus_2 : ts8 PORT MAP (newbus1_lines, newbus_on_dbus, newbus2_lines);
newbus_to_dbus_3 : dbus <= wired_byte (newbus2_lines);
control : ctrl_unit
    PORT MAP ( clk, interrupt, mod_begin, cp_rc, cp_c, r_c,
               load_row, load_col, load_buffer,
               increment_pc_page, increment_pc_offset,
               reset_pc_page, reset_pc_offset, reset_pxc,
               load_page_mar, load_offset_mar,
               databus_on_dbus, dbus_on_databus,
               newbus_on_dbus, dec_on_newbus, dec_on_counter,
               pc_on_mar_page_bus, pc_on_mar_offset_bus,
               mar_on_abus,
               read_mem, write_mem,
               en_sm, en_dm,
               dec_enable, mod_end);

```

END structural;

รูปที่ 6.27 แสดงการบรรยายระดับโครงสร้างของวงจรปรับปรุงคุณภาพ

6.3 สรุป (Summary)

ในการออกแบบวงจรปรับปรุงคุณภาพของภาพได้เริ่มต้นการออกแบบตั้งแต่ในระดับพฤติกรรม การออกแบบโครงสร้างบัสเพื่อวางแบบโครงสร้างขององค์ประกอบและเพื่อกำหนดองค์ประกอบที่จำเป็น การบรรยายในระดับข้อมูลช่วยให้สามารถกำหนดลำดับและความสัมพันธ์ของสัญญาณควบคุมต่าง ๆ ได้ง่ายขึ้น และในขั้นตอนสุดท้ายนำองค์ประกอบทั้งหมดมารวมกันในการบรรยายระดับโครงสร้างเป็นวงจรฮาร์ดแวร์ที่สมบูรณ์

วงจรปรับปรุงคุณภาพของภาพที่ออกแบบนี้สามารถนำมาใช้กับข้อมูลภาพที่มีขนาด 256X256 จุดภาพและค่าระดับความเข้มที่แตกต่างกันได้ 256 ระดับ

บทที่ 7

สถาปัตยกรรมของวงจรตรวจหาขอบของวัตถุในภาพ (Edge Detection Architecture)

การตรวจหาขอบของวัตถุในภาพเป็นหลักการที่สำคัญอย่างหนึ่งที่มีมักจะนำมาใช้ในการประมวลผลภาพเพื่อการเปรียบเทียบ การจดจำหรือเพื่อการจำแนกรายละเอียดของวัตถุในภาพ โดยอาศัยหลักการพื้นฐานของความเหมือนหรือต่างกันและความไม่ต่อเนื่องของระดับสีของจุดภาพ(pixels) ในบทนี้ ได้กล่าวถึงการออกแบบวงจรตรวจหาขอบของวัตถุในภาพตั้งแต่ระดับพฤติกรรมจนถึงระดับโครงสร้างรวมทั้งองค์ประกอบภายในทั้งหมด

7.1 พฤติกรรมของวงจรตรวจหาขอบของวัตถุในภาพ

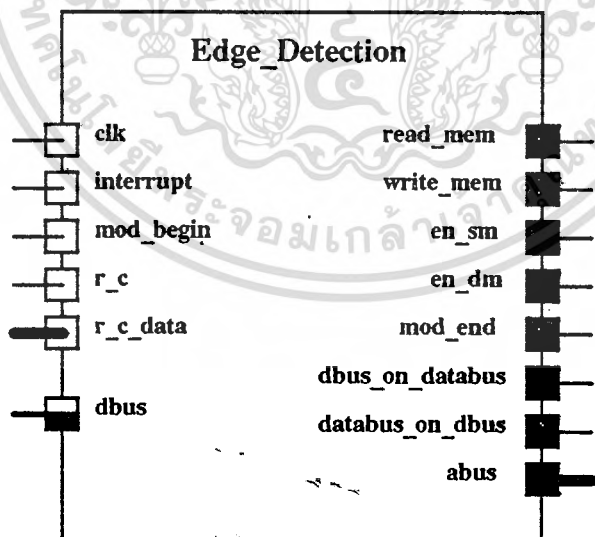
(Behavioral of Edge Detection)

พฤติกรรมการทำงานของวงจรตรวจหาขอบของวัตถุในภาพเริ่มจากการอ่านข้อมูลภาพจากหน่วยความจำต้นทางเข้ามาที่ละจุดภาพ การประมวลผลจะกระทำกับข้อมูลภาพเป็นลักษณะของแมทริกซ์ขนาด 3X3 หลังจากประมวลผลเสร็จก็จะเขียนข้อมูลใหม่ลงในหน่วยความจำปลายทาง จากนั้นก็จะเริ่มการอ่านข้อมูลจากหน่วยความจำต้นทางเข้ามาใหม่ทำเช่นนี้ซ้ำจนกระทั่งหมดจึงจะส่งสัญญาณแจ้งให้วงจรควบคุมหลักทราบต่อไป

7.1.1 การบรรยายการเชื่อมต่อของวงจรตรวจหาขอบของวัตถุในภาพ

(Interface Description of Edge Detection)

รูปที่ 7.1 แสดงสัญลักษณ์และการบรรยายส่วนเชื่อมต่อของวงจร



```
LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;
```

```

ENTITY edge_detection IS
  GENERIC ( read_high_time, read_low_time,
            write_high_time, write_low_time : TIME := 2 US;
            cycle_time : TIME := 4 US);
  PORT ( clk, interrupt, r_c, mod_begin : IN qit;
        r_c_data : IN word;
        dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
        read_mem, write_mem : OUT qit;
        en_sm, en_dm, mod_end : OUT qit;
        abus : OUT word;
        dbus_on_databus, databus_on_dbus : OUT qit);
END edge_detection;

```

รูปที่ 7.1 แสดงการบรรยายการเชื่อมต่อของวงจรตรวจหาขอบของวัตถุในภาพ

7.1.2 การบรรยายระดับพฤติกรรมของวงจรตรวจหาขอบของวัตถุในภาพ (Edge Detection Behavioral Architecture)

รูปที่ 7.2 แสดงโครงสร้างการบรรยายในระดับพฤติกรรม การทำงานจะเริ่มเมื่อได้รับสัญญาณกระตุ้นจากวงจรควบคุมคือ *interrupt* เป็น '1'

```

ARCHITECTURE behavioral OF edge_detection IS
BEGIN
  PROCESS
    Declare necessary variables; รูปที่ 7.3
  BEGIN
    IF interrupt = '1' THEN
      Handle interrupt; รูปที่ 7.4
      IF r_c = '1' THEN
        Read number of image rows and cols; รูปที่ 7.5
      ELSIF mod_begin = '1' THEN
        Read 3X3 matrix and find the gradient that use the sobel operator; รูปที่ 7.6
        Find the absolute of gradient; รูปที่ 7.7
        Send to destination memory; รูปที่ 7.8
        Send finished signal; รูปที่ 7.9
      END IF;
    END IF;
  END PROCESS;
END behavioral;

```

รูปที่ 7.2 แสดงโครงสร้างการบรรยายในระดับพฤติกรรม

รูปที่ 7.3 แสดงการกำหนดตัวแปรที่จำเป็น *sobel* เป็นชนิดข้อมูลที่กำหนดขึ้นเป็นอะเรย์
ของ *integer*

```

TYPE sobel IS ARRAY (INTEGER RANGE <>) OF INTEGER;
VARIABLE nrow, ncol : INTEGER;
VARIABLE page, offset, buff : byte;
VARIABLE i, j, gx, gy : INTEGER := 0;
VARIABLE z : sobel (1 TO 9);
VARIABLE grd : qit_vector(9 DOWNT0 0);
VARIABLE rc15_8, rc7_0 : byte;

```

รูปที่ 7.3 แสดงการกำหนดตัวแปรที่จำเป็น

รูปที่ 7.4 แสดงการเซตค่าตัวแปรเริ่มต้นเมื่อ *interrupt* เป็น '1'

```

page := zero_8;
offset := zero_8;

```

รูปที่ 7.4 แสดงการเซตค่าตัวแปรเริ่มต้น

เมื่อได้รับสัญญาณ *r_c* เป็น '1' วงจรจะทำการรับข้อมูลแสดงขนาดของภาพที่ส่งมาจาก วงจรควบคุมหลักไปเก็บไว้ที่ตัวแปร *nrow* และ *ncol* โดยต้องเปลี่ยนชนิดของข้อมูลจาก *qit* เป็น *integer* เสียก่อนโดยใช้ฟังก์ชัน *qit2int()* ดังรูปที่ 7.5

```

rc15_8 := r_c_data(15 DOWNT0 8);
rc7_0 := r_c_data(7 DOWNT0 0);
qit2int(rc15_8, nrow);
qit2int(rc7_0, ncol);

```

รูปที่ 7.5 แสดงการรับข้อมูลขนาดภาพ

รูปที่ 7.6 แสดงขั้นตอนการอ่านข้อมูลขนาด 3X3 เข้าเก็บไว้ที่ตัวแปร *z(k)* และหาค่าเกรเดียน *gx* และ *gy*

```

-- read 3X3 matrix
WHILE (i < nrow - 2) LOOP
  WHILE (j < ncol - 2) LOOP
    en_sm <= '1';
    en_dm <= '0';
    FOR k IN 1 TO 9 LOOP
      CASE k IS
        WHEN 1 =>
          int2qit (i, page);
          int2qit (j, offset);
        WHEN 2 =>
          int2qit (i, page);
          int2qit (j+1, offset);
        WHEN 3 =>
          int2qit (i, page);
          int2qit (j+2, offset);

```

```

WHEN 4 =>
    int2qit (i+1, page);
    int2qit (j, offset);
WHEN 5 =>
    int2qit (i+1, page);
    int2qit (j+1, offset);
WHEN 6 =>
    int2qit (i+1, page);
    int2qit (j+2, offset);
WHEN 7 =>
    int2qit (i+2, page);
    int2qit (j, offset);
WHEN 8 =>
    int2qit (i+2, page);
    int2qit (j+1, offset);
WHEN 9 =>
    int2qit (i+2, page);
    int2qit (j+2, offset);
END CASE;
abus <= page & offset;
read_mem <= '1'; WAIT FOR read_high_time;
buff := byte(dbus);
qit2int (buff, z(k));
read_mem <= '0'; WAIT FOR read_low_time;
END LOOP;
gx := (z(7) + 2*z(8) + z(9)) - (z(1) + 2*z(2) + z(3));
gy := (z(3) + 2*z(6) + z(9)) - (z(1) + 2*z(4) + z(7));

```

รูปที่ 7.6 แสดงการอ่านข้อมูลขนาด 3X3 และหาค่าเกรเดียน

หลังจากได้ค่าเกรเดียนในแนวแกน x และแกน y แล้วก็นำมาหาค่า *absolute* ดังรูปที่ 7.7

```

IF gx < 0 THEN
    gx := -gx;
END IF;
IF gy < 0 THEN
    gy := -gy;
END IF;

```

รูปที่ 7.7 แสดงการหาค่า *absolute*

รูปที่ 7.8 แสดงการเขียนข้อมูลซึ่งเป็นผลบวกของค่า *absolute gradient* ที่ได้มาลงในหน่วยความจำปลายทางและการส่งสัญญาณแจ้งการเสร็จภารกิจให้กับวงจรควบคุมหลักในรูปที่ 7.9

```

-- send to destination memory
en_sm <= '0';
en_dm <= '1';

```

```

int2qit ((i+1), page);
int2qit ((j+1), offset);
int2qit ((gx + gy), grd);
abus <= page & offset;
dbus <= wired_byte (grd(9 DOWNT0 2));
write_mem <= '1'; WAIT FOR write_high_time;
write_mem <= '0'; WAIT FOR write_low_time;
dbus <= "ZZZZZZZZ";
j := j + 1;
END LOOP;
j := 0;
i := i + 1;
END LOOP;

```

รูปที่ 7.8 แสดงการเขียนข้อมูล

```

mod_end <= '1';
WAIT FOR cycle_time;
mod_end <= '0';

```

รูปที่ 7.9 แสดงการส่งสัญญาณแจ้งการเสร็จภารกิจ

7.2 โครงสร้างของวงจรตรวจหาขอบของวัตถุในภาพ (Structural of Edge detection)

โครงสร้างของวงจรตรวจหาขอบของวัตถุในภาพประกอบด้วยส่วนควบคุมภายในทำหน้าที่ควบคุมการทำงานของวงจรทั้งหมดเมื่อได้รับมอบหมายจากวงจรควบคุมหลักของระบบ ส่วนข้อมูลเป็นรีจิสเตอร์ใช้เก็บข้อมูลต่างๆ ส่วนวงจร *sobel_address_register_unit* ทำหน้าที่คำนวณตำแหน่งของข้อมูลแมทริกซ์ 3X3 วงจร *sobel_data_register_unit* เก็บข้อมูลภาพแมทริกซ์ 3X3 วงจร *gradient* ทำหน้าที่ในการคำนวณหาค่าเกรเดียนต์จากแมทริกซ์ 3X3 รูปที่ 7.10 แสดงโครงสร้างบัสการเชื่อมต่อของส่วนวงจรต่างๆเข้าด้วยกัน

7.2.2 การบรรยายองค์ประกอบ (Description of Components)

ในหัวข้อนี้จะได้กล่าวถึงการบรรยายการทำงานขององค์ประกอบต่างๆตามรูปที่ 7.10 และในลำดับสุดท้ายจะได้นำเอาองค์ประกอบทั้งหมดมาประกอบรวมกันในการบรรยายระดับโครงสร้าง

7.2.2.1 การบรรยายระดับข้อมูลของส่วนควบคุมภายใน (Dataflow of Edge Controller)

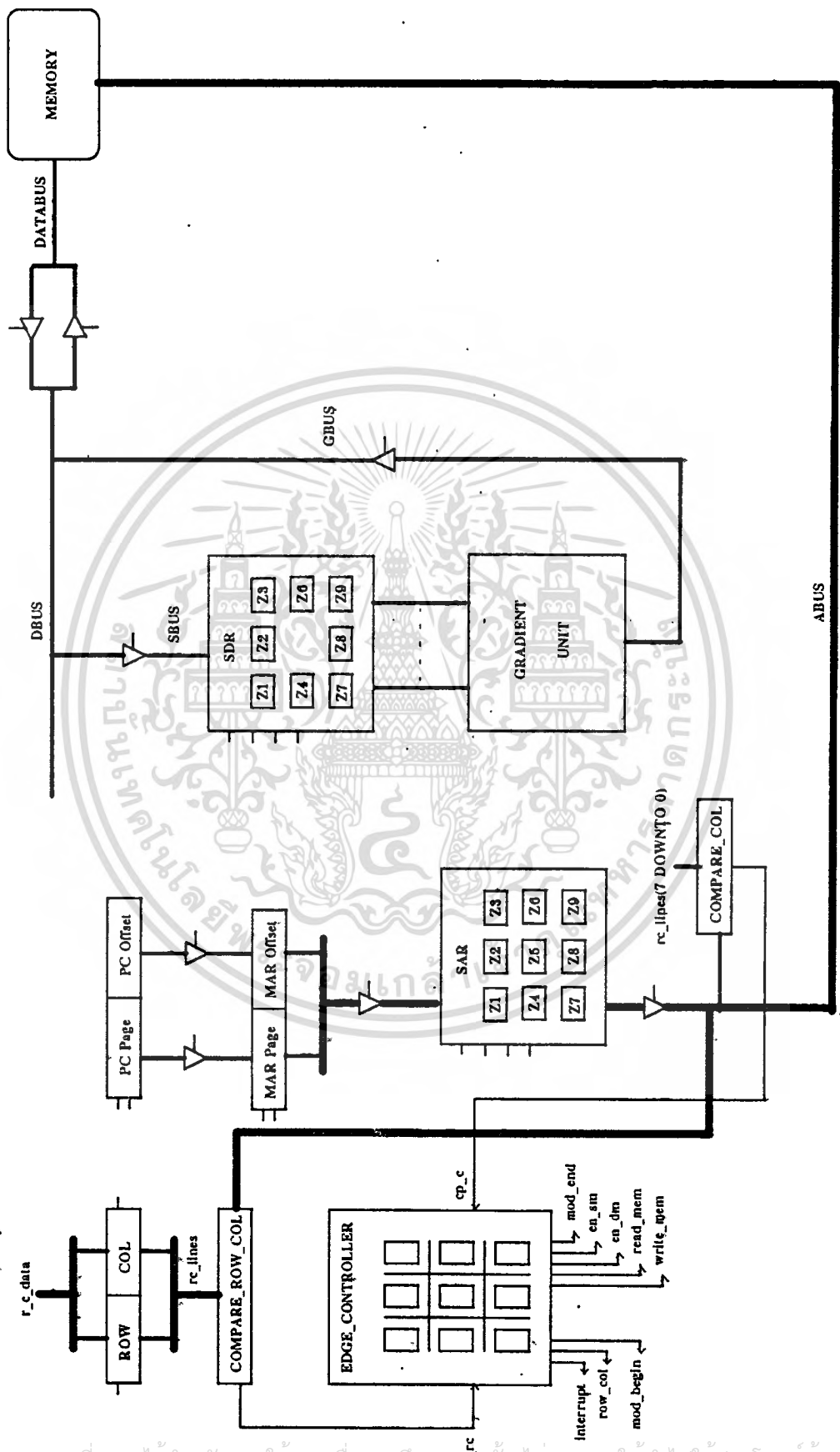
การบรรยายระดับข้อมูลของวงจรควบคุมภายในแบ่งการทำงานออกเป็นสเตจๆ รูปที่ 7.11 แสดงการบรรยายการเชื่อมต่อของวงจร

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

```

7.2.1 โครงสร้างบัส (Edge Detection Bussing Structure)



รูปที่ 7.10 แสดงโครงสร้างบัสของวงจรตรวจจับขอบวัตถุในภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่าในรูปแบบใดก็ตาม หากมีให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

ENTITY edge_controller IS
  GENERIC ( read_delay, write_delay : TIME := 3 NS;
            delay : TIME := 3 NS);
  PORT ( clk : IN qit;
         interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
         -- register control signals:
         load_row, load_col,
         load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,
         load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
         increment_pc_page, increment_pc_offset,
         reset_pc_page, reset_pc_offset,
         load_page_mar, load_offset_mar,
         load_z_sar,
         pass_z1_sar, pass_z2_sar, pass_z3_sar,
         pass_z4_sar, pass_z5_sar, pass_z6_sar,
         pass_z7_sar, pass_z8_sar, pass_z9_sar,
         -- bus connection control signals:
         dbus_on_databus, databus_on_dbus,
         grd_on_dbus, dbus_on_sbus,
         pc_on_mar_page_bus, pc_on_mar_offset_bus,
         mar_on_sar_bus, sar_on_abus,
         -- memory control signals:
         read_mem, write_mem,
         en_sm, en_dm,
         -- other control signal:
         mod_end : OUT qit);
END edge_controller;

```

รูปที่ 7.11 แสดงการบรรยายการเชื่อมต่อของวงจร *edge_controller*

ในส่วนของการบรรยายการทำงานประกอบด้วย 5 สดก สัญญาณเอาต์พุตกำหนดให้เป็นแบบ *ored_qit* โดยเรียกใช้ฟังก์ชัน *oring* จากแพ็คเกจ *basic_utilities* รูปที่ 7.12 แสดงการกำหนดสัญญาณและการกำหนดค่าของสัญญาณให้กับเอาต์พุต

```

ARCHITECTURE dataflow OF edge_controller IS
  -- oring is implied in the following signals (oi)
  SIGNAL load_row_oi, load_col_oi,
         load_z1_sdr_oi, load_z2_sdr_oi, load_z3_sdr_oi, load_z4_sdr_oi,
         load_z6_sdr_oi, load_z7_sdr_oi, load_z8_sdr_oi, load_z9_sdr_oi,
         increment_pc_page_oi, increment_pc_offset_oi,
         reset_pc_page_oi, reset_pc_offset_oi,
         load_page_mar_oi, load_offset_mar_oi, load_z_sar_oi,
         pass_z1_sar_oi, pass_z2_sar_oi, pass_z3_sar_oi,
         pass_z4_sar_oi, pass_z5_sar_oi, pass_z6_sar_oi,
         pass_z7_sar_oi, pass_z8_sar_oi, pass_z9_sar_oi,
         dbus_on_databus_oi, databus_on_dbus_oi,
         grd_on_dbus_oi, dbus_on_sbus_oi,

```

```

pc_on_mar_page_bus_oi, pc_on_mar_offset_bus_oi,
mar_on_sar_bus_oi, sar_on_abus_oi,
read_mem_oi, write_mem_oi, en_sm_oi, en_dm_oi,
mod_end_oi : ored_qit BUS;
SIGNAL s : ored_qit_vector (5 DOWNT0 1) REGISTER := "00001";
SIGNAL z : ored_qit_vector (8 DOWNT0 1) REGISTER := "00000001";
BEGIN
load_row <= load_row_oi;
load_col <= load_col_oi;
load_z1_sdr <= load_z1_sdr_oi;
load_z2_sdr <= load_z2_sdr_oi;
load_z3_sdr <= load_z3_sdr_oi;
load_z4_sdr <= load_z4_sdr_oi;
load_z6_sdr <= load_z6_sdr_oi;
load_z7_sdr <= load_z7_sdr_oi;
load_z8_sdr <= load_z8_sdr_oi;
load_z9_sdr <= load_z9_sdr_oi;
increment_pc_page <= increment_pc_page_oi;
increment_pc_offset <= increment_pc_offset_oi;
reset_pc_page <= reset_pc_page_oi;
reset_pc_offset <= reset_pc_offset_oi;
load_page_mar <= load_page_mar_oi;
load_offset_mar <= load_offset_mar_oi;
load_z_sar <= load_z_sar_oi;
pass_z1_sar <= pass_z1_sar_oi;
pass_z2_sar <= pass_z2_sar_oi;
pass_z3_sar <= pass_z3_sar_oi;
pass_z4_sar <= pass_z4_sar_oi;
pass_z5_sar <= pass_z5_sar_oi;
pass_z6_sar <= pass_z6_sar_oi;
pass_z7_sar <= pass_z7_sar_oi;
pass_z8_sar <= pass_z8_sar_oi;
pass_z9_sar <= pass_z9_sar_oi;
dbus_on_databus <= dbus_on_databus_oi;
databus_on_dbus <= databus_on_dbus_oi;
grd_on_dbus <= grd_on_dbus_oi;
dbus_on_sbus <= dbus_on_sbus_oi;
pc_on_mar_page_bus <= pc_on_mar_page_bus_oi;
pc_on_mar_offset_bus <= pc_on_mar_offset_bus_oi;
mar_on_sar_bus <= mar_on_sar_bus_oi;
sar_on_abus <= sar_on_abus_oi;
read_mem <= read_mem_oi;
write_mem <= write_mem_oi;
en_sm <= en_sm_oi;
en_dm <= en_dm_oi;
mod_end <= mod_end_oi;

```

รูปที่ 7.12 แสดงการกำหนดสัญญาณและการกำหนดค่าของสัญญาณให้กับเอาต์พุต

เอกสารนี้เป็นเอกสารของบริษัทเอกชนที่ออกให้เพื่อใช้ในการดำเนินงานเท่านั้น ไม่ควรเผยแพร่ให้บุคคลภายนอกได้ใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 7.13 แสดงการทำงานในสเตตที่ 1 เมื่อสัญญาณอินพุต *interrupt* เป็น '1' ซึ่งเป็นสัญญาณที่กำหนดให้ไมโครโวลต์ที่ได้รับเริ่มการปฏิบัติงาน สัญญาณอินพุต *r_c* เป็น '1' กำหนดให้วงจรรับค่าขนาด *row* และ *column* ซึ่งเป็นขนาดของข้อมูลภาพ สเตตจะเปลี่ยนสถานะเข้าสู่สเตตที่ 2 เมื่อสัญญาณอินพุต *mod_begin* เป็น '1'

```
s1 : BLOCK (s(1) = '1')
BEGIN
    -- pc to mar
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';
    load_page_mar_oi <= GUARDED '1';
    load_offset_mar_oi <= GUARDED '1';
    -- reset pc if no interrupt
    reset_pc_page_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
    reset_pc_offset_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
    int : BLOCK (interrupt = '1' AND GUARD)
        BEGIN -- load rows and cols
            load_row_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
            load_col_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
            -- goto 2 if module begin
            ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
                BEGIN
                    s(1) <= GUARDED '1' WHEN mod_begin = '0' ELSE '0';
                    s(2) <= GUARDED '1' WHEN mod_begin /= '0' ELSE '0';
                END BLOCK ck;
            END BLOCK int;
        END BLOCK s1;
```

รูปที่ 7.13 แสดงการทำงานในสเตตที่ 1

```
s2 : BLOCK (s(2) = '1')
BEGIN -- set sobel address register
    -- pc to mar to sar
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';
    load_page_mar_oi <= GUARDED '1';
    load_offset_mar_oi <= GUARDED '1';
    mar_on_sar_bus_oi <= GUARDED '1';
    load_z_sar_oi <= GUARDED '1' AFTER delay;
    -- goto 3
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(3) <= GUARDED '1';
        END BLOCK ck;
    END BLOCK s2;
```

รูปที่ 7.14 แสดงการทำงานในสเตตที่ 2

การทำงานในสเตทที่ 2 จะส่งสัญญาณควบคุมไปให้วงจร *sobel_address_register_unit* ทำการคำนวณหาตำแหน่งแอดเดรสของแมทริกซ์ขนาด 3X3 ดังรูปที่ 7.14

```

s3 : BLOCK (s(3) = '1')
BEGIN -- read matrix data into sdr
  z1 : BLOCK (z(1) = '1')
  BEGIN
    pass_z1_sar_oi <= GUARDED '1';
    sar_onabus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_ondbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z1_sdr_oi <= GUARDED '1';
    -- goto z2
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
      z(2) <= GUARDED '1';
    END BLOCK ck;
  END BLOCK z1;
  .
  .
  .
  z8 : BLOCK (z(8) = '1')
  BEGIN
    pass_z9_sar_oi <= GUARDED '1';
    sar_onabus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_ondbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z9_sdr_oi <= GUARDED '1';
    -- goto 4
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
      s(4) <= GUARDED '1';
    END BLOCK ck;
  END BLOCK z8;
  ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
  BEGIN
    z(8 DOWNT0 1) <= GUARDED "00000000";
  END BLOCK ck;
END BLOCK s3;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้ก่อนใช้งานจริงที่ศูนย์วิจัยและพัฒนาเทคโนโลยีสารสนเทศแห่งชาติ ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

รูปที่ 7.15 แสดงการทำงานในสเตทที่ 3

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 7.15 แสดงการทำงานในสเตทที่ 3 สัญญาณควบคุมจะถูกส่งไปควบคุมการอ่านข้อมูลตามตำแหน่งแอดเดรสของเมมทริกซ์ที่คำนวณได้จากวงจร *sobel_address_register_unit* และเก็บค่าไว้ที่รีจิสเตอร์ *sobel_data_register_unit*

รูปที่ 7.16 แสดงการทำงานในสเตทที่ 4 เป็นขั้นตอนการนำข้อมูลเขียนลงในหน่วยความจำปลายทาง

```
s4 : BLOCK (s(4) = '1')
BEGIN -- write gradient data
    pass_z5_sar_oi <= GUARDED '1';
    sar_onabus_oi <= GUARDED '1';
    grd_ondbus_oi <= GUARDED '1';
    dbus_ondatabus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '0';
    en_dm_oi <= GUARDED '1';
    write_mem_oi <= GUARDED '1' AFTER write_delay;
    -- goto 5
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(5) <= GUARDED '1';
END BLOCK ck;
END BLOCK s4;
```

รูปที่ 7.16 แสดงการทำงานในสเตทที่ 4

```
s5 : BLOCK (s(5) = '1')
BEGIN -- check end of image
    pass_z9_sar_oi <= GUARDED '1';
    sar_onabus_oi <= GUARDED '1';
    chk : BLOCK (cp_rc /= '1' AND GUARD)
    BEGIN
        -- increment pc_offset
        increment_pc_offset_oi <= GUARDED '1' WHEN cp_c /= '1';
        -- increment pc page and reset pc offset
        increment_pc_page_oi <= GUARDED '1' WHEN cp_c = '1';
        reset_pc_offset_oi <= GUARDED '1' WHEN cp_c = '1';
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(2) <= GUARDED '1';
        END BLOCK ck;
    END BLOCK chk;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_end_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
    END BLOCK ck;
END BLOCK s5;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
รูปที่ 7.17 แสดงการทำงานในสเตทที่ 5
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 7.17 แสดงการทำงานในสเตทที่ 5 เป็นการตรวจสอบว่าทำการประมวลผลข้อมูลภาพหมดหรือยัง ถ้าเสร็จแล้วก็จะส่งสัญญาณแจ้งให้วงจรควบคุมหลักรับทราบต่อไป

```
ck : BLOCK ((clk = '0' AND NOT ck'STABLE) AND GUARD)
BEGIN
  s(5 DOWNT0 1) <= GUARDED '00000';
END BLOCK ck;
```

รูปที่ 7.18 แสดงการรีเซตสถานะของสเตท

7.2.2.2 โครงร่างของส่วนควบคุมภายใน (General Outline of Edge Controller)

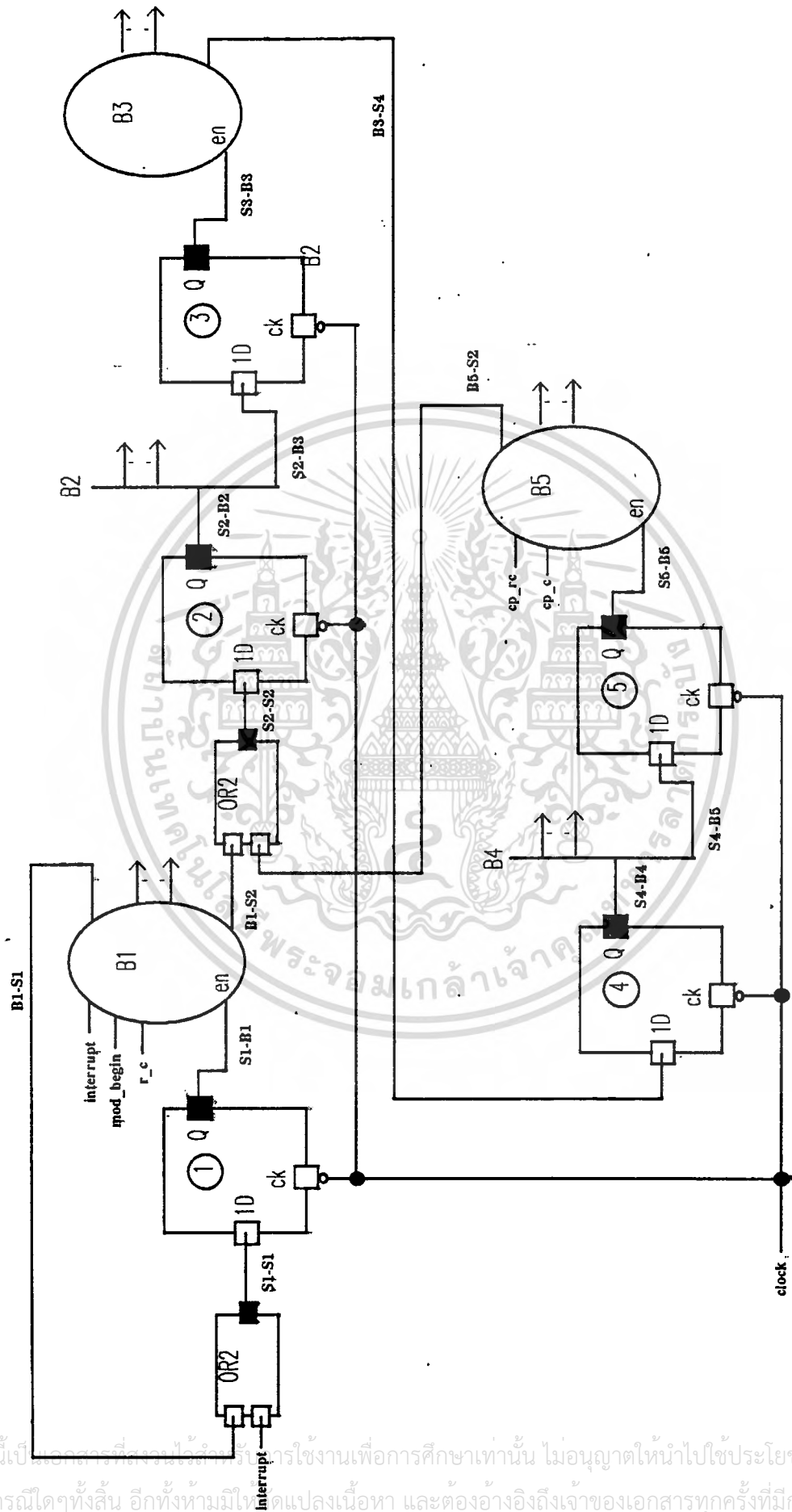
จากการบรรยายในระดับข้อมูลของวงจรควบนํามาใช้เป็นแนวทางในการออกแบบโครงสร้างได้ดังรูปที่ 7.19 ฟลิปฟล็อป master-slave D-flipflop ถูกใช้เพื่อเป็นตัวเซตสถานะของสเตทและบับเบิลถูกใช้แทนลอจิกการทำงานในแต่ละสเตท

7.2.2.3 การบรรยายระดับโครงสร้างของส่วนควบคุมภายใน (Structural of Edge Controller)

แต่ละสเตทในรูปที่ 7.19 จะถูกนํามาออกแบบแยกเป็นแต่ละองค์ประกอบเพื่อให้ง่ายและสะดวกในการออกแบบ รูปที่ 7.20 แสดงการบรรยายระดับโครงสร้างโดยนำแต่ละองค์ประกอบมารวมกันเป็นโครงสร้างทางฮาร์ดแวร์ที่สมบูรณ์

```
LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
LIBRARY edge;

ENTITY edge_controller IS
  PORT ( clk : IN qit;
        interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
        -- register control signals:
        load_row, load_col,
        load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,
        load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
        increment_pc_page, increment_pc_offset,
        reset_pc_page, reset_pc_offset,
        load_page_mar, load_offset_mar, load_z_sar,
        pass_z1_sar, pass_z2_sar, pass_z3_sar,
        pass_z4_sar, pass_z5_sar, pass_z6_sar,
        pass_z7_sar, pass_z8_sar, pass_z9_sar,
        -- bus connection control signals:
        dbus_on_databus, databus_on_dbus,
        grd_on_dbus, dbus_on_sbus,
        pc_on_mar_page_bus, pc_on_mar_offset_bus,
        mar_on_sar_bus, sar_on_abus,
```



รูปที่ 7.19 แสดงโครงสร้างของส่วนควบคุมภายใน

```

    -- memory control signals:
    read_mem, write_mem,
    en_sm, en_dm,
    -- other control signal:
    mod_end : OUT qit);
END edge_controller;

```

ARCHITECTURE structural OF edge_controller IS

COMPONENT es1

```

    PORT ( s1, clk, int, r_c, mod_begin : IN qit;
          b1_load_page_mar, b1_load_offset_mar;
          b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
          b1_reset_pc_page, b1_reset_pc_offset,
          b1_load_row, b1_load_col,
          b1_s1, b1_s2 : OUT qit);

```

END COMPONENT;

COMPONENT es2

```

    PORT ( s2, clk : IN qit;
          b2_load_page_mar, b2_load_offset_mar,
          b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
          b2_mar_on_sar_bus, b2_load_z_sar,
          s2_s3 : OUT qit);

```

END COMPONENT;

COMPONENT es3

```

    PORT ( s3, clk : IN qit;
          b3_databus_on_dbus, b3_dbus_on_sbus,
          b3_sar_on_abus, b3_read_mem,
          b3_en_sm, b3_en_dm,
          b3_load_z1_sdr, b3_load_z2_sdr,
          b3_load_z3_sdr, b3_load_z4_sdr,
          b3_load_z6_sdr, b3_load_z7_sdr,
          b3_load_z8_sdr, b3_load_z9_sdr,
          b3_pass_z1_sar, b3_pass_z2_sar,
          b3_pass_z3_sar, b3_pass_z4_sar,
          b3_pass_z5_sar, b3_pass_z6_sar,
          b3_pass_z7_sar, b3_pass_z8_sar,
          b3_pass_z9_sar, b3_s4 : OUT qit);

```

END COMPONENT;

COMPONENT es4

```

    PORT ( s4, clk : IN qit;
          b4_pass_z5_sar, b4_dbus_on_databus,
          b4_grd_on_dbus, b4_sar_on_abus, b4_write_mem,
          b4_en_sm, b4_en_dm, s4_s5 : OUT qit);

```

END COMPONENT;

COMPONENT es5

```

    PORT ( s5, clk, cp_rc, cp_c : IN qit;
          b5_pass_z9_sar, b5_sar_on_abus,

```

```

        b5_increment_pc_page, b5_increment_pc_offset,
        b5_reset_pc_offset, b5_mod_end, b5_s2 : OUT qit);
END COMPONENT;
FOR s1 : es1 USE ENTITY edge.state_ectrl_1 (structural);
FOR s2 : es2 USE ENTITY edge.state_ectrl_2 (structural);
FOR s3 : es3 USE ENTITY edge.state_ectrl_3 (structural);
FOR s4 : es4 USE ENTITY edge.state_ectrl_4 (structural);
FOR s5 : es5 USE ENTITY edge.state_ectrl_5 (structural);
FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);
FOR ALL : o3 USE ENTITY gates.or3_q (average_delay);
SIGNAL s1_s1, b1_s1, b1_s2 : qit;
SIGNAL s2_s2, s2_s3 : qit;
SIGNAL b3_s4_s4_s5, b5_s2 : qit;
SIGNAL b1_reset_pc_offset, b5_reset_pc_offset : qit;
SIGNAL b1_load_page_mar, b1_load_offset_mar,
        b2_load_page_mar, b2_load_offset_mar : qit;
SIGNAL b3_pass_z9_sar, b5_pass_z9_sar : qit;
SIGNAL b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
        b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus : qit;
SIGNAL b3_sar_on_abus, b4_sar_on_abus, b5_sar_on_abus : qit;
SIGNAL b3_en_sm, b3_en_dm, b4_en_sm, b4_en_dm : qit;
BEGIN
g0 : o2 PORT MAP (interrupt, b1_s1, s1_s1);
s1 : es1 PORT MAP (s1_s1, clk, interrupt, r_c, mod_begin,
        b1_load_page_mar, b1_load_offset_mar,
        b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
        reset_pc_page, b1_reset_pc_offset,
        load_row, load_col, b1_s1, b1_s2);
g1 : o2 PORT MAP (b1_s2, b5_s2, s2_s2);
s2 : es2 PORT MAP (s2_s2, clk,
        b2_load_page_mar, b2_load_offset_mar,
        b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
        mar_on_sar_bus, load_z_sar, s2_s3);
s3 : es3 PORT MAP (s2_s3, clk,
        databus_on_dbus, dbus_on_sbus,
        b3_sar_on_abus, read_mem,
        b3_en_sm, b3_en_dm,
        load_z1_sdr, load_z2_sdr,
        load_z3_sdr, load_z4_sdr,
        load_z6_sdr, load_z7_sdr,
        load_z8_sdr, load_z9_sdr,
        pass_z1_sar, pass_z2_sar,
        pass_z3_sar, pass_z4_sar,
        pass_z5_sar, pass_z6_sar,
        pass_z7_sar, pass_z8_sar,
        b3_pass_z9_sar, b3_s4);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

s4 : es4 PORT MAP (b3_s4, clk,
                    pass_z5_sar, dbus_on_databus,
                    grd_on_dbus, b4_sar_on_abus, write_mem,
                    b4_en_sm, b4_en_dm, s4_s5);
s5 : es5 PORT MAP (s4_s5, clk, cp_rc, cp_c,
                    b5_pass_z9_sar, b5_sar_on_abus,
                    increment_pc_page, increment_pc_offset,
                    b5_reset_pc_offset, mod_end, b5_s2);

-- OR together
g2 : o2 PORT MAP (b1_reset_pc_offset, b5_reset_pc_offset, reset_pc_offset);
g3 : o2 PORT MAP (b1_load_page_mar, b2_load_page_mar, load_page_mar);
g4 : o2 PORT MAP (b1_load_offset_mar, b2_load_offset_mar, load_offset_mar);
g5 : o2 PORT MAP (b3_pass_z9_sar, b5_pass_z9_sar, pass_z9_sar);
g6 : o2 PORT MAP (b1_pc_on_mar_page_bus, b2_pc_on_mar_page_bus,
                    pc_on_mar_page_bus);
g7 : o2 PORT MAP (b1_pc_on_mar_offset_bus, b2_pc_on_mar_offset_bus,
                    pc_on_mar_offset_bus);
g8 : o3 PORT MAP (b3_sar_on_abus, b4_sar_on_abus,
                    b5_sar_on_abus, sar_on_abus);
g9 : o2 PORT MAP (b3_en_sm, b4_en_sm, en_sm);
g10 : o2 PORT MAP (b3_en_dm, b4_en_dm, en_dm);
END structural;

```

รูปที่ 7.20 แสดงการบรรยายระดับโครงสร้างของส่วนควบคุมภายใน

รูปที่ 7.21-7.26 แสดงการบรรยายระดับโครงสร้างของวงจรควบคุมแต่ละสเตจซึ่งมีลำดับการปฏิบัติเป็นไปตามการบรรยายการทำงานในระดับข้อมูลที่ผ่านมา

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

```

```

ENTITY state_ectrl_1 IS
    PORT ( s1, clk, int, r_c, mod_begin : IN qit;
           b1_load_page_mar, b1_load_offset_mar,
           b1_pc_on_mar_page_bus, b1_pc_on_mar_offset_bus,
           b1_reset_pc_page, b1_reset_pc_offset,
           b1_load_row, b1_load_col,
           b1_s1, b1_s2 : OUT qit);
END state_ectrl_1;

```

ARCHITECTURE structural OF state_ectrl_1 IS

```

FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
FOR ALL : a4 USE ENTITY gates.and4_q (average_delay);
SIGNAL inv_clk, inv_int, inv_mod_begin : qit;
SIGNAL im1, im2 : qit;
SIGNAL s1_b1 : qit;

```

BEGIN

```

control_signal : msdff PORT MAP (s1, clk, s1_b1);
g0 : inv PORT MAP (clk, inv_clk);
g1 : inv PORT MAP (int, inv_int);
g2 : inv PORT MAP (mod_begin, inv_mod_begin);
g3 : b1_load_page_mar <= s1_b1;
g4 : b1_load_offset_mar <= s1_b1;
g5 : b1_pc_on_mar_page_bus <= s1_b1;
g6 : b1_pc_on_mar_offset_bus <= s1_b1;
g7 : a2 PORT MAP (s1_b1, inv_int, im1);
g8 : b1_reset_pc_page <= im1;
g9 : b1_reset_pc_offset <= im1;
g10 : a3 PORT MAP (s1_b1, int, r_c, im2);
g11 : b1_load_row <= im2;
g12 : b1_load_col <= im2;
g13 : a4 PORT MAP (s1_b1, int, inv_clk, inv_mod_begin, b1_s1);
g14 : a4 PORT MAP (s1_b1, int, inv_clk, mod_begin, b1_s2);

```

END structural;

รูปที่ 7.21 แสดงการบรรยายวงจร state_ectrl_1

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

```

ENTITY state_ectrl_2 IS

```

PORT ( s2, clk : IN qit;
      b2_load_page_mar, b2_load_offset_mar,
      b2_pc_on_mar_page_bus, b2_pc_on_mar_offset_bus,
      b2_mar_on_sar_bus, b2_load_z_sar,
      s2_s3 : OUT qit);

```

END state_ectrl_2;

ARCHITECTURE structural OF state_ectrl_2 IS

```

FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
FOR ALL : delay USE ENTITY gates.signal_delay (structural);
SIGNAL inv_clk : qit;
SIGNAL s2_b2 : qit;
BEGIN
control_signal : msdff PORT MAP (s2, clk, s2_b2);
g0 : inv PORT MAP (clk, inv_clk);
g1 : b2_load_page_mar <= s2_b2;
g2 : b2_load_offset_mar <= s2_b2;
g3 : b2_pc_on_mar_page_bus <= s2_b2;
g4 : b2_pc_on_mar_offset_bus <= s2_b2;
g5 : b2_mar_on_sar_bus <= s2_b2;
g6 : delay PORT MAP (s2_b2, b2_load_z_sar);
g7 : a2 PORT MAP (s2_b2, inv_clk, s2_s3);
END structural;

```

รูปที่ 7.22 แสดงการบรรยายวงจร state_ectrl_2

ภายในสเตทที่ 3 มีลำดับการทำงานเป็นสเตทย่อย อีก 8 สเตทดังรูปที่ 7.23

```

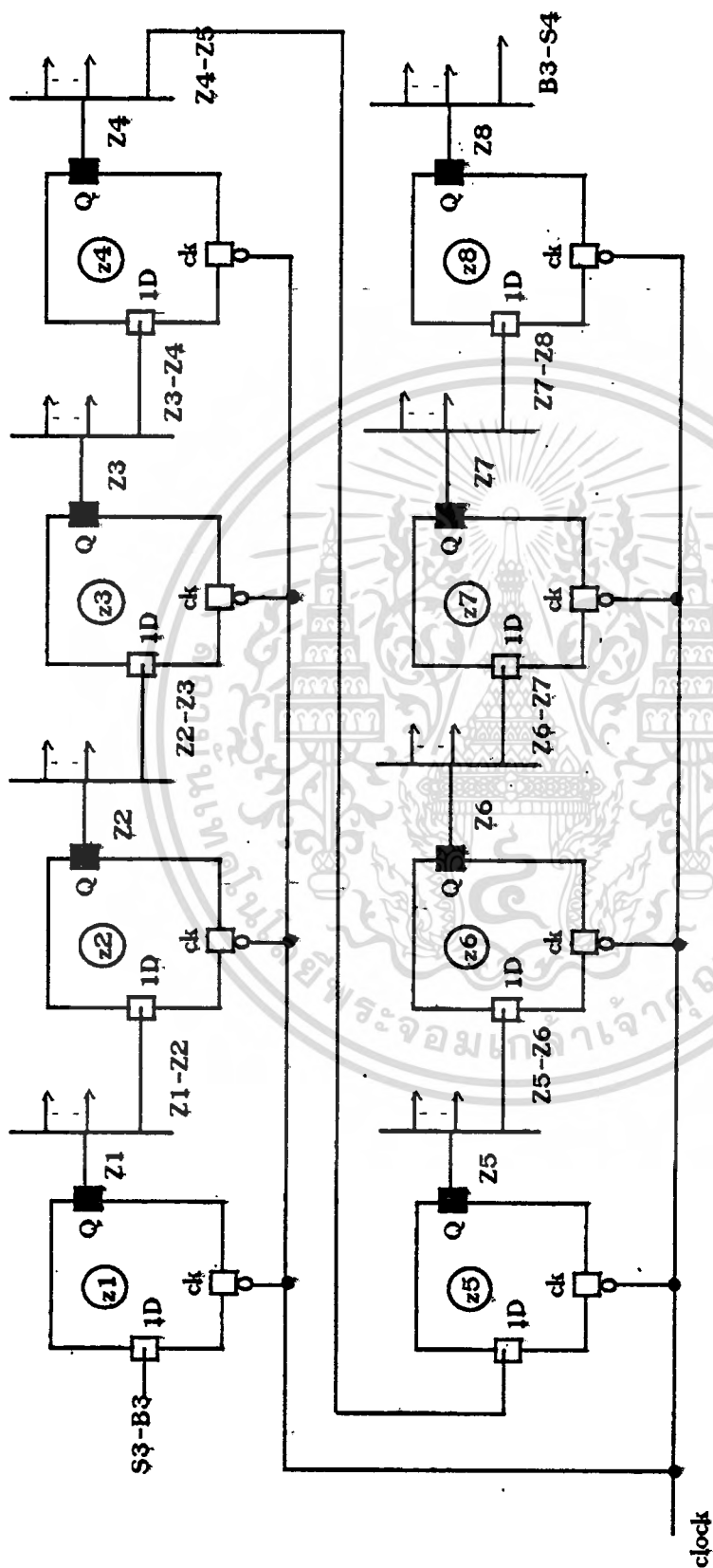
LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_ectrl_3 IS
PORT ( s3, clk : IN qit;
      b3_databus_on_dbus, b3_dbus_on_sbus,
      b3_sar_on_abus, b3_read_mem,
      b3_en_sm, b3_en_dm,
      b3_load_z1_sdr, b3_load_z2_sdr,
      b3_load_z3_sdr, b3_load_z4_sdr,
      b3_load_z6_sdr, b3_load_z7_sdr,
      b3_load_z8_sdr, b3_load_z9_sdr,
      b3_pass_z1_sar, b3_pass_z2_sar,
      b3_pass_z3_sar, b3_pass_z4_sar,
      b3_pass_z5_sar, b3_pass_z6_sar,
      b3_pass_z7_sar, b3_pass_z8_sar,
      b3_pass_z9_sar, b3_s4 : OUT qit);
END state_ectrl_3;

```

เอกสารนี้เป็นทรัพย์สินทางปัญญาของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ไม่ควรนำออกเผยแพร่โดยไม่ได้รับอนุญาตให้วงใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7.23 แสดงโครงสร้างของสแตทก์ 3

ARCHITECTURE structural OF state_ectrl_3 IS

```

FOR ALL : msdff USE ENTITY gates.master_slave_dff (gate_level);
FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
FOR ALL : o2 USE ENTITY gates.or2_q (double_delay);
FOR ALL : o4 USE ENTITY gates.or4_q (average_delay);
FOR ALL : delay USE ENTITY gates.signal_delay (structural);
SIGNAL inv_clk : qit;
SIGNAL z1_databus_on_dbus, z1_dbus_on_sbus, z1_sar_on_abus,
      z1_read_mem, z1_en_sm, z1_en_dm : qit;
SIGNAL z2_databus_on_dbus, z2_dbus_on_sbus, z2_sar_on_abus,
      z2_read_mem, z2_en_sm, z2_en_dm : qit;
SIGNAL z3_databus_on_dbus, z3_dbus_on_sbus, z3_sar_on_abus,
      z3_read_mem, z3_en_sm, z3_en_dm : qit;
SIGNAL z4_databus_on_dbus, z4_dbus_on_sbus, z4_sar_on_abus,
      z4_read_mem, z4_en_sm, z4_en_dm : qit;
SIGNAL z5_databus_on_dbus, z5_dbus_on_sbus, z5_sar_on_abus,
      z5_read_mem, z5_en_sm, z5_en_dm : qit;
SIGNAL z6_databus_on_dbus, z6_dbus_on_sbus, z6_sar_on_abus,
      z6_read_mem, z6_en_sm, z6_en_dm : qit;
SIGNAL z7_databus_on_dbus, z7_dbus_on_sbus, z7_sar_on_abus,
      z7_read_mem, z7_en_sm, z7_en_dm : qit;
SIGNAL z8_databus_on_dbus, z8_dbus_on_sbus, z8_sar_on_abus,
      z8_read_mem, z8_en_sm, z8_en_dm : qit;
SIGNAL z14_databus_on_dbus, z58_databus_on_dbus : qit;
SIGNAL z14_dbus_on_sbus, z58_dbus_on_sbus : qit;
SIGNAL z14_sar_on_abus, z58_sar_on_abus : qit;
SIGNAL z14_read_mem, z58_read_mem : qit;
SIGNAL z14_en_sm, z58_en_sm : qit;
SIGNAL z14_en_dm, z58_en_dm : qit;
SIGNAL z1_z2, z2_z3, z3_z4, z4_z5, z5_z6, z6_z7, z7_z8 : qit;
SIGNAL z1, z2, z3, z4, z5, z6, z7, z8 : qit;
SIGNAL s3_b3 : qit;

```

BEGIN

```

control_signal : msdff PORT MAP (s3, clk, s3_b3);
g0 : inv PORT MAP (clk, inv_clk);
-- z1 control
r1 : msdff PORT MAP (s3_b3, clk, z1);
g1 : z1_databus_on_dbus <= z1;
g2 : z1_dbus_on_sbus <= z1;
g3 : z1_sar_on_abus <= z1;
g4 : delay PORT MAP (z1, z1_read_mem);
g5 : z1_en_sm <= z1;
g7 : inv PORT MAP (z1, z1_en_dm);
g8 : b3_load_z1_sdr <= z1;
g9 : b3_pass_z1_sar <= z1;
g10 : a2 PORT MAP (z1, inv_clk, z1_z2);

```

```

-- z2 control
r2 : msdff PORT MAP (z1_z2, clk, z2);
g11 : z2_databus_on_dbus <= z2;
g12 : z2_dbus_on_sbus <= z2;
g13 : z2_sar_on_abus <= z2;
g14 : delay PORT MAP (z2, z2_read_mem);
g15 : z2_en_sm <= z2;
g17 : inv PORT MAP (z2, z2_en_dm);
g18 : b3_load_z2_sdr <= z2;
g19 : b3_pass_z2_sar <= z2;
g20 : a2 PORT MAP (z2, inv_clk, z2_z3);
-- z3 control
r3 : msdff PORT MAP (z2_z3, clk, z3);
g21 : z3_databus_on_dbus <= z3;
g22 : z3_dbus_on_sbus <= z3;
g23 : z3_sar_on_abus <= z3;
g24 : delay PORT MAP (z3, z3_read_mem);
g25 : z3_en_sm <= z3;
g27 : inv PORT MAP (z3, z3_en_dm);
g28 : b3_load_z3_sdr <= z3;
g29 : b3_pass_z3_sar <= z3;
g30 : a2 PORT MAP (z3, inv_clk, z3_z4);
-- z4 control
r4 : msdff PORT MAP (z3_z4, clk, z4);
g31 : z4_databus_on_dbus <= z4;
g32 : z4_dbus_on_sbus <= z4;
g33 : z4_sar_on_abus <= z4;
g34 : delay PORT MAP (z4, z4_read_mem);
g35 : z4_en_sm <= z4;
g37 : inv PORT MAP (z4, z4_en_dm);
g38 : b3_load_z4_sdr <= z4;
g39 : b3_pass_z4_sar <= z4;
g40 : a2 PORT MAP (z4, inv_clk, z4_z5);
-- z5 control
r5 : msdff PORT MAP (z4_z5, clk, z5);
g41 : z5_databus_on_dbus <= z5;
g42 : z5_dbus_on_sbus <= z5;
g43 : z5_sar_on_abus <= z5;
g44 : delay PORT MAP (z5, z5_read_mem);
g45 : z5_en_sm <= z5;
g47 : inv PORT MAP (z5, z5_en_dm);
g48 : b3_load_z5_sdr <= z5;
g49 : b3_pass_z5_sar <= z5;
g50 : a2 PORT MAP (z5, inv_clk, z5_z6);
-- z6 control
r6 : msdff PORT MAP (z5_z6, clk, z6);
g51 : z6_databus_on_dbus <= z6;

```

```

g52 : z6_dbus_on_sbus <= z6;
g53 : z6_sar_on_abus <= z6;
g54 : delay PORT MAP (z6, z6_read_mem);
g55 : z6_en_sm <= z6;
g57 : inv PORT MAP (z6, z6_en_dm);
g58 : b3_load_z6_sdr <= z6;
g59 : b3_pass_z6_sar <= z6;
g60 : a2 PORT MAP (z6, inv_clk, z6_z7);
-- z7 control
r7 : msdff PORT MAP (z6_z7, clk, z7);
g61 : z7_databus_on_dbus <= z7;
g62 : z7_dbus_on_sbus <= z7;
g63 : z7_sar_on_abus <= z7;
g64 : delay PORT MAP (z7, z7_read_mem);
g65 : z7_en_sm <= z7;
g67 : inv PORT MAP (z7, z7_en_dm);
g68 : b3_load_z7_sdr <= z7;
g69 : b3_pass_z7_sar <= z7;
g70 : a2 PORT MAP (z7, inv_clk, z7_z8);
-- z8 control
r8 : msdff PORT MAP (z7_z8, clk, z8);
g71 : z8_databus_on_dbus <= z8;
g72 : z8_dbus_on_sbus <= z8;
g73 : z8_sar_on_abus <= z8;
g74 : delay PORT MAP (z8, z8_read_mem);
g75 : z8_en_sm <= z8;
g77 : inv PORT MAP (z8, z8_en_dm);
g78 : b3_load_z8_sdr <= z8;
g79 : b3_pass_z8_sar <= z8;
g80 : a2 PORT MAP (z8, inv_clk, b3_s4);
-- OR together
g81 : o4 PORT MAP (z1_databus_on_dbus, z2_databus_on_dbus,
                  z3_databus_on_dbus, z4_databus_on_dbus, z14_databus_on_dbus);
g82 : o4 PORT MAP (z5_databus_on_dbus, z6_databus_on_dbus,
                  z7_databus_on_dbus, z8_databus_on_dbus, z58_databus_on_dbus);
g83 : o2 PORT MAP (z14_databus_on_dbus, z58_databus_on_dbus, b3_databus_on_dbus);

g84 : o4 PORT MAP (z1_dbus_on_sbus, z2_dbus_on_sbus,
                  z3_dbus_on_sbus, z4_dbus_on_sbus, z14_dbus_on_sbus);
g85 : o4 PORT MAP (z5_dbus_on_sbus, z6_dbus_on_sbus,
                  z7_dbus_on_sbus, z8_dbus_on_sbus, z58_dbus_on_sbus);
g86 : o2 PORT MAP (z14_dbus_on_sbus, z58_dbus_on_sbus, b3_dbus_on_sbus);

g87 : o4 PORT MAP (z1_sar_on_abus, z2_sar_on_abus,
                  z3_sar_on_abus, z4_sar_on_abus, z14_sar_on_abus);
g88 : o4 PORT MAP (z5_sar_on_abus, z6_sar_on_abus,
                  z7_sar_on_abus, z8_sar_on_abus, z58_sar_on_abus);

```

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์โดยสำนักงานคณะกรรมการกฤษฎีกา อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

g89 : o2 PORT MAP (z14_sar_onabus, z58_sar_onabus, b3_sar_onabus);

g90 : o4 PORT MAP (z1_read_mem, z2_read_mem,
                   z3_read_mem, z4_read_mem, z14_read_mem);
g91 : o4 PORT MAP (z5_read_mem, z6_read_mem,
                   z7_read_mem, z8_read_mem, z58_read_mem);
g92 : o2 PORT MAP (z14_read_mem, z58_read_mem, b3_read_mem);

g93 : o4 PORT MAP (z1_en_sm, z2_en_sm, z3_en_sm, z4_en_sm, z14_en_sm);
g94 : o4 PORT MAP (z5_en_sm, z6_en_sm, z7_en_sm, z8_en_sm, z58_en_sm);
g95 : o2 PORT MAP (z14_en_sm, z58_en_sm, b3_en_sm);

g96 : o4 PORT MAP (z1_en_dm, z2_en_dm, z3_en_dm, z4_en_dm, z14_en_dm);
g97 : o4 PORT MAP (z5_en_dm, z6_en_dm, z7_en_dm, z8_en_dm, z58_en_dm);
g98 : o2 PORT MAP (z14_en_dm, z58_en_dm, b3_en_dm);
END structural;

```

รูปที่ 7.24 แสดงการบรรยายวงจร *state_ectrl_3*

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_ectrl_4 IS
    PORT ( s4, clk : IN qit;
           b4_pass_z5_sar, b4_dbus_on_databus,
           b4_grd_on_dbus, b4_sar_onabus, b4_write_mem,
           b4_en_sm, b4_en_dm, s4_s5 : OUT qit);
END state_ectrl_4;

ARCHITECTURE structural OF state_ectrl_4 IS
    FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
    FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
    FOR ALL : delay USE ENTITY gates.signal_delay (structural);
    SIGNAL inv_clk : qit;
    SIGNAL s4_b4 : qit;
BEGIN
    control_signal : msdff PORT MAP (s4, clk, s4_b4);
    g0 : inv PORT MAP (clk, inv_clk);
    g1 : b4_pass_z5_sar <= s4_b4;
    g2 : b4_dbus_on_databus <= s4_b4;
    g3 : b4_grd_on_dbus <= s4_b4;

```

```

g4 : b4_sar_onabus <= s4_b4;
g5 : delay PORT MAP (s4_b4, b4_write_mem);
g6 : inv PORT MAP (s4_b4, b4_en_sm);
g7 : b4_en_dm <= s4_b4;
g8 : a2 PORT MAP (s4_b4, inv_clk, s4_s5);
END structural;

```

รูปที่ 7.25 แสดงการบรรยายวงจร state_ectrl_4

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;

ENTITY state_ectrl_5 IS
  PORT ( s5, clk, cp_rc, cp_c : IN qit;
         b5_pass_z9_sar, b5_sar_onabus,
         b5_increment_pc_page, b5_increment_pc_offset,
         b5_reset_pc_offset, b5_mod_end, b5_s2 : OUT qit);
END state_ectrl_5;

ARCHITECTURE structural OF state_ectrl_5 IS
  FOR control_signal : msdff USE ENTITY gates.master_slave_dff (gate_level);
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : a2 USE ENTITY gates.and2_q (double_delay);
  FOR ALL : a3 USE ENTITY gates.and3_q (average_delay);
  SIGNAL inv_clk, inv_cp_rc, inv_cp_c : qit;
  SIGNAL lm1 : qit;
  SIGNAL s5_b5 : qit;
BEGIN
  control_signal : msdff PORT MAP (s5, clk, s5_b5);
  g0 : inv PORT MAP (clk, inv_clk);
  g1 : inv PORT MAP (cp_rc, inv_cp_rc);
  g2 : inv PORT MAP (cp_c, inv_cp_c);
  g3 : b5_pass_z9_sar <= s5_b5;
  g4 : b5_sar_onabus <= s5_b5;
  g5 : a3 PORT MAP (s5_b5, inv_cp_rc, inv_cp_c, b5_increment_pc_offset);
  g6 : a3 PORT MAP (s5_b5, inv_cp_rc, cp_c, lm1);
  g7 : b5_increment_pc_page <= lm1;
  g8 : b5_reset_pc_offset <= lm1;
  g9 : a2 PORT MAP (s5_b5, inv_clk, b5_mod_end);
  g10 : a3 PORT MAP (s5_b5, inv_cp_rc, inv_clk, b5_s2);
END structural;

```

รูปที่ 7.26 แสดงการบรรยายวงจร state_ectrl_5

7.2.2.4 วงจร Gradient Unit

วงจร *Gradient Unit* เป็นวงจรสำหรับคำนวณหาค่าของเกรเดียนตามสมการ

$$\nabla f \approx |G_x| + |G_y|$$

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY gradient IS
  PORT ( i1, i2, i3, i4, i6, i7, i8, i9 : IN byte;
         o1 : OUT byte);
END gradient;

ARCHITECTURE behavioral OF gradient IS
BEGIN
  PROCESS (i1, i2, i3, i4, i6, i7, i8, i9)
    VARIABLE z1, z2, z3, z4, z6, z7, z8, z9 : INTEGER;
    VARIABLE gx, gx1, gx2, gy, gy1, gy2, gxy : INTEGER;
    VARIABLE grt : qit_vector(10 DOWNT0 0);
  BEGIN
    qit2int (i1, z1);   qit2int (i2, z2);
    qit2int (i3, z3);   qit2int (i4, z4);
    qit2int (i6, z6);   qit2int (i7, z7);
    qit2int (i8, z8);   qit2int (i9, z9);

    gx1 := (z7 + 2*z8 + z9);
    gx2 := (z1 + 2*z2 + z3);
    gy1 := (z3 + 2*z6 + z9);
    gy2 := (z1 + 2*z4 + z7);

    IF (gx1 > gx2) THEN
      gx := gx1 - gx2;
    ELSE
      gx := gx2 - gx1;
    END IF;

    IF (gy1 > gy2) THEN
      gy := gy1 - gy2;
    ELSE
      gy := gy2 - gy1;
    END IF;

    gxy := gx + gy;
    int2qit (gxy, grt);
    o1 <= grt(10 DOWNT0 3);
  END PROCESS;
END behavioral;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

(a)

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY edge;

ENTITY gradient IS
    PORT ( i1, i2, i3, i4, i6, i7, i8, i9 : IN byte;
           o1 : OUT byte);
END gradient;

ARCHITECTURE structural OF gradient IS
    COMPONENT abs
        PORT ( i1, i2, i3, i4, i5, i6 : IN byte;
               o1 : OUT qit_vector(9 DOWNT0 0));
    END COMPONENT;
    FOR ALL : abs USE ENTITY edge.absolute_gxy (structural);
    FOR ALL : add10 USE ENTITY reg.10_bit_adder (structural);
    SIGNAL gx, gy, grd : qit_vector(9 DOWNT0 0);
    SIGNAL gnd : qit := '0';
    SIGNAL c : qit;
BEGIN
    abs_gx : abs PORT MAP (i7, i8, i9, i1, i2, i3, gx);
    abs_gy : abs PORT MAP (i3, i6, i9, i1, i4, i7, gy);
    grd : add10 PORT MAP (gx, gy, gnd, grd, c);
    o1 <= c & grd (9 DOWNT0 3);
END structural;

```

(b)

รูปที่ 7.27 แสดงการบรรยายวงจร *Gradient Unit*

(a) การบรรยายระดับพฤติกรรม

(b) การบรรยายระดับโครงสร้าง

7.2.2.5 วงจร *Absolute_Gxy Unit*

วงจร *Absolute_Gxy Unit* เป็นวงจรสำหรับหาค่า *absolute* โดยการใช้การเปรียบเทียบว่าค่าใดมากกว่าให้ใช้ค่านั้นเป็นตัวตั้งลบด้วยค่าที่น้อยกว่าก็จะได้ผลที่เป็นบวกเสมอ

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
ENTITY absolute_gxy IS
    PORT ( i1, i2, i3, i4, i5, i6 : IN byte;
           o1 : OUT qit_vector(9 DOWNT0 0));
END absolute_gxy;

```

เอกสารนี้เป็นส่วนหนึ่งสำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

ARCHITECTURE structural OF absolute_gxy IS
  COMPONENT gxy
    PORT ( i1, i2, i3 : IN byte; gxy : OUT qit_vector(9 DOWNT0 0));
  END COMPONENT;
  COMPONENT cp10
    PORT ( a, b : IN qit_vector(9 DOWNT0 0);
          gt, eq, lt : IN qit;
          a_gt_b, a_eq_b; a_lt_b : OUT qit);
  END COMPONENT;
  FOR ALL : gxy USE ENTITY edge.gxy_adder (structural);
  FOR compare : cp10 USE ENTITY edge.10_bit_comparator (structural);
  FOR ALL : add10 USE ENTITY reg.10_bit_adder (structural);
  FOR ALL : n2 USE ENTITY gates.nand2_q (double_delay);
  FOR ALL : inv USE ENTITY gates.inv_q (double_delay);
  FOR ALL : inv10 USE ENTITY reg.10_inv_q (structural);
  FOR ALL : ts10 USE ENTITY reg.tri_state_10 (structural);
  SIGNAL vdd : qit := '1';
  SIGNAL gnd : qit := '0';
  SIGNAL g1, g2, g1_2, g2_1 : qit_vector (9 DOWNT0 0);
  SIGNAL g1_cpl, g2_cpl : qit_vector (9 DOWNT0 0);
  SIGNAL gbus : wired_ten BUS := "ZZZZZZZZZZ";
  SIGNAL gtr, eql, lss : qit;
  SIGNAL gtr_eql : qit;
  SIGNAL c1, c2 : qit;
BEGIN
  gxy_1 : gxy PORT MAP (i1, i2, i3, g1);
  gxy_2 : gxy PORT MAP (i4, i5, i6, g2);
  compare : cp10 PORT MAP (g1, g2, gnd, vdd, gnd, gtr, eql, lss);
  gtr_bar : inv PORT MAP (gtr, gtr_b);
  eql_bar : inv PORT MAP (eql, eql_b);
  -- g1 greater than or equal g2
  g2_one_complement : inv10 PORT MAP (g2, g2_cpl);
  g1_sub_g2 : add10 PORT MAP (g1, g2_cpl, vdd, g1_2, c1);
  -- g1 less than g2
  g1_one_complement : inv10 PORT MAP (g1, g1_cpl);
  g2_sub_g1 : add10 PORT MAP (g2, g1_cpl, vdd, g2_1, c2);
  -- absolute
  gt_or_eq : n2 PORT MAP (gtr_b, eql_b, gtr_eql);
  g1_2_abs : ts10 PORT MAP (g1_2, gtr_eql, gbus);
  g2_1_abs : ts10 PORT MAP (g2_1, lss, gbus);
  o1 <= qit_vector (gbus);
END structural;

```

รูปที่ 7.28 แสดงการบรรยายวงจร Absolute_Gxy Unit

7.2.2.6 วงจร Gxy_Adder Unit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าในการทำ Gx และ Gy ดังสมการข้างล่าง

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$$G_x = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3)$$

$$G_y = (Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z_7)$$

สังเกตเห็นได้ว่าพจน์ที่อยู่ในแต่ละวงเล็บจะมีรูปแบบที่เหมือนกันคืออยู่ในรูปแบบ $(_ + 2_ + _)$ ดังนั้นก็จะสามารถใช้วงจร *Gxy_Adder Unit* ได้กับทั้ง 4 วงเล็บดังนี้

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
ENTITY gxy_adder IS
    PORT ( i1, i2, i3 : IN byte;
           gxy : OUT qit_vector(9 DOWNT0 0));
END gxy_adder;

ARCHITECTURE structural OF gxy_adder IS
    FOR ALL : add8 USE ENTITY reg.8_bit_adder (structural);
    FOR ALL : add9 USE ENTITY reg.9_bit_adder (structural);
    SIGNAL im1, im2 : byte;
    SIGNAL im3, im4, im5 : qit_vector(8 DOWNT0 0);
    SIGNAL c1, c2, c3 : qit;
    SIGNAL gnd : qit := '0';
BEGIN
    i1_sum_i2 : add8 PORT MAP (i1, i2, gnd, im1, c1);
    i2_sum_i3 : add8 PORT MAP (i2, i3, gnd, im2, c2);
    im3 <= c1 & im1;
    im4 <= c2 & im2;
    total_sum : add9 PORT MAP (im3, im4, gnd, im5, c3);
    gxy <= c3 & im5;
END structural;

```

รูปที่ 7.29 แสดงการบรรยายวงจร *Gxy_Adder Unit*

7.2.2.7 วงจร Bit_Comparator

วงจร *Bit_Comparator* เป็นวงจรเปรียบเทียบทีละบิต

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
ENTITY bit_comparator IS
    PORT ( a, b, gt, eq, lt : IN qit;
           a_gt_b, a_eq_b, a_lt_b : OUT qit);
END bit_comparator;

```

สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

ARCHITECTURE gate_level OF bit_comparator IS
  FOR ALL : inv USE ENTITY gates.inv_q(double_delay);
  FOR ALL : n2 USE ENTITY gates.nand2_q(double_delay);
  FOR ALL : n3 USE ENTITY gates.nand3_q(average_delay);
  SIGNAL im1,im2,im3,im4,im5,im6,im7,im8,im9,im10 : qit;
BEGIN
  -- a_gt_b output
  g0 : inv PORT MAP (a, im1);
  g1 : inv PORT MAP (b, im2);
  g2 : n2 PORT MAP (a, im2, im3);
  g3 : n2 PORT MAP (a, gt, im4);
  g4 : n2 PORT MAP (im2, gt, im5);
  g5 : n3 PORT MAP (im3, im4, im5, a_gt_b);
  -- a_eq_b output
  g6 : n3 PORT MAP (im1, im2, eq, im6);
  g7 : n3 PORT MAP (a, b, eq, im7);
  g8 : n2 PORT MAP (im6, im7, a_eq_b);
  -- a_lt_b output
  g9 : n2 PORT MAP (im1, b, im8);
  g10 : n2 PORT MAP (im1, lt, im9);
  g11 : n2 PORT MAP (b, lt, im10);
  g12 : n3 PORT MAP (im8, im9, im10, a_lt_b);
END gate_level;

```

รูปที่ 7.30 แสดงการบรรยายวงจร *Bit_Comparator*

7.2.2.8 วงจร 10_Bit_comparator

วงจร *10_Bit_Comparator* เป็นวงจรเปรียบเทียบขนาด 10 บิต

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
LIBRARY edge;

ENTITY 10_bit_comparator IS
  PORT ( a, b : IN qit_vector( 9 DOWNTO 0);
         gt, eq, lt : IN qit;
         a_gt_b, a_eq_b, a_lt_b : OUT qit);
END 10_bit_comparator;

ARCHITECTURE structural OF 10_bit_comparator IS
  COMPONENT comp1
    PORT ( a, b, gt, eq, lt : IN qit;
          a_gt_b, a_eq_b, a_lt_b : OUT qit);
  END COMPONENT;
  FOR ALL : comp1 USE ENTITY edge.bit_comparator(gate_level);

```

```

CONSTANT n : INTEGER := 10;
SIGNAL im : qit_vector ( 0 TO (n-1)*3-1);
BEGIN
  c_all : FOR i IN 0 TO n-1 GENERATE
    l : IF i = 0 GENERATE
      least : comp1 PORT MAP ( a(i), b(i), gt, eq, lt, im(0), im(1), im(2));
    END GENERATE;
    m : IF i = n-1 GENERATE
      most : comp1 PORT MAP ( a(i), b(i),
                             im(i*3-3), im(i*3-2), im(i*3-1),
                             a_gt_b, a_eq_b, a_lt_b);
    END GENERATE;
    r : IF i > 0 AND i < n-1 GENERATE
      rest : comp1 PORT MAP ( a(i), b(i),
                             im(i*3-3), im(i*3-2), im(i*3-1),
                             im(i*3+0), im(i*3+1), im(i*3+2));
    END GENERATE;
  END GENERATE;
END structural;

```

รูปที่ 7.31 แสดงการบรรยายวงจร 10_Bit_Comparator

7.2.2.9 วงจร Sobel_Address_Register Unit

วงจร Sobel_Address_Register Unit เป็นวงจรคำนวณตำแหน่งข้อมูลเป็นแมทริกซ์

ขนาด 3X3

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY sobel_address_register_unit IS
  PORT ( i16 : IN word; o16 : OUT wired_word;
         load_z, ck : IN qit;
         pass_z : IN qit_vector (1 TO 9));
END sobel_address_register_unit;

ARCHITECTURE behavioral OF sobel_address_register_unit IS
BEGIN
  PROCESS (ck)
    VARIABLE x, x1, x2, y, y1, y2 : INTEGER;
    VARIABLE zr1, zr2, zr3, zr4, zr5, zr6, zr7, zr8, zr9 : byte;
    VARIABLE zc1, zc2, zc3, zc4, zc5, zc6, zc7, zc8, zc9 : byte;
    VARIABLE z : wired_word BUS := "ZZZZZZZZZZZZZZZZ";
  BEGIN
    qit2int (i16(15 DOWNT0 8), x);
    qit2int (i16(7 DOWNT0 0), y);
    x1 := x + 1;
    x2 := x + 2;

```

```

y1 := y + 1;
y2 := y + 2;
IF (ck = '0') THEN
  IF load_z = '1' THEN
    int2qit (x, zr1);    int2qit (y, zc1);
    int2qit (x, zr2);    int2qit (y1, zc2);
    int2qit (x, zr3);    int2qit (y2, zc3);
    int2qit (x1, zr4);   int2qit (y, zc4);
    int2qit (x1, zr5);   int2qit (y1, zc5);
    int2qit (x1, zr6);   int2qit (y2, zc6);
    int2qit (x2, zr7);   int2qit (y, zc7);
    int2qit (x2, zr8);   int2qit (y1, zc8);
    int2qit (x2, zr9);   int2qit (y2, zc9);
  END IF;
END IF;
CASE pass_z IS
  WHEN "100000000" =>
    z := zr1 & zc1;
  WHEN "010000000" =>
    z := zr2 & zc2;
  WHEN "001000000" =>
    z := zr3 & zc3;
  WHEN "000100000" =>
    z := zr4 & zc4;
  WHEN "000010000" =>
    z := zr5 & zc5;
  WHEN "000001000" =>
    z := zr6 & zc6;
  WHEN "000000100" =>
    z := zr7 & zc7;
  WHEN "000000010" =>
    z := zr8 & zc8;
  WHEN "000000001" =>
    z := zr9 & zc9;
  WHEN OTHERS => NULL;
END CASE;
o16 <= z;
END PROCESS;
END behavioral;

```

(a)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.register_logic.ALL;
LIBRARY edge;

```

ENTITY sobel_address_register_unit IS

PORT (i16 : IN word; o16 : OUT wired_word;
load_z, ck : IN qit;
pass_z : IN qit_vector (1 TO 9));

END sobel_address_register_unit;

ARCHITECTURE structural OF sobel_address_register_unit IS

COMPONENT add_1_2

PORT (adi : IN byte; ado1, ado2 : OUT byte);

END COMPONENT;

FOR ALL : add_1_2 USE ENTITY edge.sar_adder_unit (structural);

FOR ALL : rg8 USE ENTITY reg.register_unit_8 (structural);

FOR ALL : ts16 USE ENTITY reg.tri_state_16 (structural);

SIGNAL sar_bus : wired_word BUS := "ZZZZZZZZZZZZZZZZ";

SIGNAL r0_lines, c0_lines : byte;

SIGNAL r1_lines, r2_lines, c1_lines, c2_lines, : byte;

SIGNAL z1_lines, z2_lines, z3_lines, z4_lines, z5_lines,

z6_lines, z7_lines, z8_lines, z9_lines : word;

BEGIN

r0_lines <= i16(15 DOWNT0 8);

c0_lines <= i16(7 DOWNT0 0);

row_1_2 : add_1_2 PORT MAP (r0_lines, r1_lines, r2_lines);

col_1_2 : add_1_2 PORT MAP (c0_lines, c1_lines, c2_lines);

z1_row_unit : rg8 PORT MAP (r0_lines, z1_lines(15 DOWNT0 8), load_z, ck);

z1_col_unit : rg8 PORT MAP (c0_lines, z1_lines(7 DOWNT0 0), load_z, ck);

z1_addr : ts16 PORT MAP (z1_lines, pass_z(1), sar_bus);

z2_row_unit : rg8 PORT MAP (r0_lines, z2_lines(15 DOWNT0 8), load_z, ck);

z2_col_unit : rg8 PORT MAP (c1_lines, z2_lines(7 DOWNT0 0), load_z, ck);

z2_addr : ts16 PORT MAP (z2_lines, pass_z(2), sar_bus);

z3_row_unit : rg8 PORT MAP (r0_lines, z3_lines(15 DOWNT0 8), load_z, ck);

z3_col_unit : rg8 PORT MAP (c2_lines, z3_lines(7 DOWNT0 0), load_z, ck);

z3_addr : ts16 PORT MAP (z3_lines, pass_z(3), sar_bus);

z4_row_unit : rg8 PORT MAP (r1_lines, z4_lines(15 DOWNT0 8), load_z, ck);

z4_col_unit : rg8 PORT MAP (c0_lines, z4_lines(7 DOWNT0 0), load_z, ck);

z4_addr : ts16 PORT MAP (z4_lines, pass_z(4), sar_bus);

z5_row_unit : rg8 PORT MAP (r1_lines, z5_lines(15 DOWNT0 8), load_z, ck);

z5_col_unit : rg8 PORT MAP (c1_lines, z5_lines(7 DOWNT0 0), load_z, ck);

z5_addr : ts16 PORT MAP (z5_lines, pass_z(5), sar_bus);

z6_row_unit : rg8 PORT MAP (r1_lines, z6_lines(15 DOWNT0 8), load_z, ck);

z6_col_unit : rg8 PORT MAP (c2_lines, z6_lines(7 DOWNT0 0), load_z, ck);

z6_addr : ts16 PORT MAP (z6_lines, pass_z(6), sar_bus);

```

z7_row_unit : rg8 PORT MAP (r2_lines, z7_lines(15 DOWNT0 8), load_z, ck);
z7_col_unit : rg8 PORT MAP (c0_lines, z7_lines(7 DOWNT0 0), load_z, ck);
z7_addr : ts16 PORT MAP (z7_lines, pass_z(7), sar_bus);

z8_row_unit : rg8 PORT MAP (r2_lines, z8_lines(15 DOWNT0 8), load_z, ck);
z8_col_unit : rg8 PORT MAP (c1_lines, z8_lines(7 DOWNT0 0), load_z, ck);
z8_addr : ts16 PORT MAP (z8_lines, pass_z(8), sar_bus);

z9_row_unit : rg8 PORT MAP (r2_lines, z9_lines(15 DOWNT0 8), load_z, ck);
z9_col_unit : rg8 PORT MAP (c2_lines, z9_lines(7 DOWNT0 0), load_z, ck);
z9_addr : ts16 PORT MAP (z9_lines, pass_z(9), sar_bus);
END structural;

```

(b)

รูปที่ 7.32 แสดงการบรรยายวงจร *Sobel_Address_Register Unit*

(a) การบรรยายระดับพฤติกรรม

(b) การบรรยายระดับโครงสร้าง

7.2.2.10 วงจร Sar_Adder Unit

วงจร *Sar_Adder Unit* เป็นวงจรบวกตำแหน่งแอดเดรสที่นำไปใช้กับวงจร *Sobel_Address_Register Unit* อีกทีหนึ่ง

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;

ENTITY sar_adder_unit IS
  PORT ( adi : IN byte; ado1, ado2 : OUT byte);
END sar_adder_unit;

ARCHITECTURE structural OF sar_adder_unit IS
  FOR ALL : add8 USE ENTITY reg.8_bit_adder (structural);
  SIGNAL im1, im2 : byte;
  SIGNAL c1, c2 : qit;
  SIGNAL vdd : qit := '1';
  SIGNAL gnd : qit := '0';
  SIGNAL zero : byte := "00000000";
BEGIN
  add_1 : add8 PORT MAP (adi, zero, vdd, im1, c1);
  add_2 : add8 PORT MAP (im1, zero, vdd, im2, c2);
  ado1 <= im1;
  ado2 <= im2;
END structural;

```

(b)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น

รูปที่ 7.33 แสดงการบรรยายวงจร *Sar_Adder Unit* ของเอกสารทุกครั้งที่มีการนำไปใช้

7.2.2.11 วงจร Sobel_Data_Register Unit

วงจร *Sobel_Data_Register Unit* เป็นรีจิสเตอร์เก็บข้อมูลภาพตามตำแหน่งที่ได้จากการคำนวณของวงจร *Sobel_Address_Register Unit*

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;

ENTITY sobel_data_register_unit IS
    PORT ( din : IN byte;
          o1, o2, o3, o4, o6, o7, o8, o9 : OUT byte;
          i1, i2, i3, i4, i6, i7, i8, i9 : IN qit;
          ck : IN qit);
END sobel_data_register_unit;

ARCHITECTURE structural OF sobel_data_register_unit IS
    FOR ALL : rg8 USE ENTITY reg.register_unit_8 (structural);
BEGIN
    z1_data : rg8 PORT MAP (din, o1, i1, ck);
    z2_data : rg8 PORT MAP (din, o2, i2, ck);
    z3_data : rg8 PORT MAP (din, o3, i3, ck);
    z4_data : rg8 PORT MAP (din, o4, i4, ck);
    z6_data : rg8 PORT MAP (din, o6, i6, ck);
    z7_data : rg8 PORT MAP (din, o7, i7, ck);
    z8_data : rg8 PORT MAP (din, o8, i8, ck);
    z9_data : rg8 PORT MAP (din, o9, i9, ck);
END structural;

```

รูปที่ 7.34 แสดงการบรรยายวงจร *Sobel_Data_Register Unit*

7.2.3 การบรรยายระดับโครงสร้างของวงจรตรวจจับขอบของวัตถุในภาพ (Edge Detection Structural Architecture)

หลังจากที่ได้ออกแบบและตรวจสอบการทำงานขององค์ประกอบต่างๆ เรียบร้อยแล้วก็นำเอามาประกอบรวมกันเป็นวงจรฮาร์ดแวร์ที่สมบูรณ์ในระดับโครงสร้างดังรูปที่ 7.35

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY edge;

ENTITY edge_detection IS
    PORT ( clk, interrupt, r_c, mod_begin : IN qit;

```

```

r_c_data : IN word;
dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
read_mem, write_mem : OUT qit;
en_sm, en_dm, mod_end : OUT qit;
abus : OUT word;
dbus_on_databus, databus_on_dbus : OUT qit);

```

END edge_detection;

ARCHITECTURE structural OF edge_detection IS

COMPONENT ctrl_unit

```

PORT ( clk : IN qit;
      interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
      load_row, load_col,
      load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,
      load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
      increment_pc_page, increment_pc_offset,
      reset_pc_page, reset_pc_offset,
      load_page_mar, load_offset_mar,
      load_z_sar,
      pass_z1_sar, pass_z2_sar, pass_z3_sar,
      pass_z4_sar, pass_z4_sar, pass_z6_sar,
      pass_z7_sar, pass_z8_sar, pass_z9_sar,
      dbus_on_databus, databus_on_dbus,
      grd_on_dbus, dbus_on_sbus,
      pc_on_mar_page_bus, pc_on_mar_offset_bus,
      mar_on_sar_bus, sar_on_abus,
      read_mem, write_mem,
      en_sm, en_dm,
      mod_end : OUT qit);

```

END COMPONENT;

COMPONENT sar

```

PORT ( i16 : IN word; o16 : OUT wired_word;
      load_z, ck : IN qit;
      pass_z : IN qit_vector (1 TO 9));

```

END COMPONENT;

COMPONENT sdr

```

PORT ( din : IN byte;
      o1, o2, o3, o4, o6, o7, o8, o9 : OUT byte;
      i1, i2, i3, i4, i6, i7, i8, i9 : IN qit;
      ck : IN qit);

```

END COMPONENT;

COMPONENT grd

```

PORT ( i1, i2, i3, i4, i6, i7, i8, i9 : IN byte;
      o1 : OUT byte);

```

END COMPONENT;

FOR control : ctrl_unit USE ENTITY edge.edge_controller(dataflow);

FOR sobel_address : sar USE ENTITY edge.sobel_address_register_unit(behavioral);

```

FOR sobel_matrix : sdr USE ENTITY edge.sobel_data_register_unit(dataflow);
FOR gradient : grd USE ENTITY edge.gradient(behavioral);
FOR ALL : reg USE ENTITY reg.register_unit(dataflow);
FOR row_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);
FOR col_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);
FOR memory_addr : mar USE ENTITY reg.memory_address_register_unit(behavioral);
FOR compare_rc : cprc USE ENTITY reg.compare_row_col(dataflow);
FOR compare_c : cpc USE ENTITY reg.compare_col(dataflow);
FOR ALL : ts8 USE ENTITY reg.tri_state_8(structural);
FOR ALL : ts16 USE ENTITY reg.tri_state_16(structural);
SIGNAL cp_rc, cp_c : qit;
SIGNAL load_row, load_col,
    load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,
    load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
    increment_pc_page, increment_pc_offset,
    reset_pc_page, reset_pc_offset,
    load_page_mar, load_offset_mar,
    load_z_sar,
    pass_z1_sar, pass_z2_sar, pass_z3_sar,
    pass_z4_sar, pass_z4_sar, pass_z6_sar,
    pass_z7_sar, pass_z8_sar, pass_z9_sar : qit;
SIGNAL dbus_on_databus, databus_on_dbus,
    grd_on_dbus, dbus_on_sbus,
    pc_on_mar_page_bus, pc_on_mar_offset_bus,
    mar_on_sar_bus, sar_on_abus : qit;
SIGNAL pc1_lines, pc2_lines, mar1_lines, mar2_lines : word;
SIGNAL rc_lines : word;
SIGNAL sar1_lines, sar2_lines : word;
SIGNAL sbus1_lines, sbus2_lines : byte;
SIGNAL gbus1_lines, gbus2_lines : byte;
SIGNAL sdr1_lines, sdr2_lines, sdr3_lines, sdr4_lines,
    sdr6_lines, sdr7_lines, sdr8_lines, sdr9_lines : byte;
SIGNAL pass_z_sar : qit_vector (1 TO 9);
BEGIN
    row : reg PORT MAP (r_c_data(15 DOWNT0 8), rc_lines(15 DOWNT0 8), load_row, clk);
    col : reg PORT MAP (r_c_data(7 DOWNT0 0), rc_lines(7 DOWNT0 0), load_col, clk);
    compare_rc : cprc PORT MAP (rc_lines, sar2_lines, cp_rc);
    compare_c : cpc PORT MAP (rc_lines(7 DOWNT0 0), sar2_lines(7 DOWNT0 0), cp_c);
    row_counter : rcpc PORT MAP (pc1_lines(15 DOWNT0 8), increment_pc_page, reset_pc_page, clk);
    col_counter : rcpc PORT MAP (pc1_lines(7 DOWNT0 0), increment_pc_offset, reset_pc_offset, clk);
    pc_to_mar_page : ts8 PORT MAP (pc1_lines(15 DOWNT0 8), pc_on_mar_page_bus,
        pc2_lines(15 DOWNT0 8));
    pc_to_mar_offset : ts8 PORT MAP (pc1_lines(7 DOWNT0 0), pc_on_mar_offset_bus,
        pc2_lines(7 DOWNT0 0));
    memory_addr : mar PORT MAP (pc2_lines, mar1_lines, load_page_mar, load_offset_mar, clk);
    mar_to_sar : ts16 PORT MAP (mar1_lines, mar_on_sar_bus, mar2_lines);
    pass_z : pass_z_sar <= pass_z1_sar & pass_z2_sar & pass_z3_sar & pass_z4_sar &

```

```

pass_z5_sar & pass_z6_sar & pass_z7_sar & pass_z8_sar & pass_z9_sar;
sobel_address : sar PORT MAP (mar2_lines, sar1_lines, load_z_sar, clk, pass_z_sar);
sar_to_abus : ts16 PORT MAP (sar1_lines, sar_on_abus, sar2_lines);
dbus_to_sbus_1 : sbus1_lines <= qit_vector (dbus);
dbus_to_sbus_2 : ts8 PORT MAP (sbus1_lines, dbus_on_sbus, sbus2_lines);
sobel_matrix : sdr PORT MAP (sbus2_lines, sdr1_lines, sdr2_lines, sdr3_lines,
                             sdr4_lines, sdr6_lines, sdr7_lines, sdr8_lines,
                             sdr9_lines, load_z1_sdr, load_z2_sdr, load_z3_sdr,
                             load_z4_sdr, load_z6_sdr, load_z7_sdr, load_z8_sdr,
                             load_z9_sdr, clk);
gradient : grd PORT MAP (sdr1_lines, sdr2_lines, sdr3_lines, sdr4_lines,
                         sdr6_lines, sdr7_lines, sdr8_lines, sdr9_lines,
                         gbus1_lines);
gbus_to_dbus_1 : ts8 PORT MAP (gbus1_lines, grd_on_dbus, gbus2_lines);
gbus_to_dbus_2 : dbus <= wired_byte (gbus2_lines);
control : ctrl_unit
    PORT MAP ( clk,interrupt, mod_begin, cp_rc, cp_c, r_c,
               load_row, load_col,
               load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,
               load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
               increment_pc_page, increment_pc_offset,
               reset_pc_page, reset_pc_offset,
               load_page_mar, load_offset_mar,
               load_z_sar,
               pass_z1_sar, pass_z2_sar, pass_z3_sar,
               pass_z4_sar, pass_z4_sar, pass_z6_sar,
               pass_z7_sar, pass_z8_sar, pass_z9_sar,
               dbus_on_databus, databus_on_dbus,
               grd_on_dbus, dbus_on_sbus,
               pc_on_mar_page_bus, pc_on_mar_offset_bus,
               mar_on_sar_bus, sar_on_abus,
               read_mem, write_mem,
               en_sm, en_dm,
               mod_end);
END structural;

```

รูปที่ 7.35 แสดงการบรรยายระดับโครงสร้างของวงจรตรวจหาขอบของวัตถุในภาพ

7.3 สรุป (Summary)

ในบทนี้ได้แสดงการออกแบบและการบรรยายการทำงานของวงจรตรวจหาขอบของวัตถุในภาพทั้งในระดับพฤติกรรม ระดับข้อมูล และระดับโครงสร้าง การทำงานของวงจรสามารถใช้ในการประมวลผลได้กับข้อมูลภาพที่มีขนาด 256X256 จุดภาพ ในการออกแบบเลือกใช้วิธีของโซเบล (Sobel operators) ซึ่งวิธีนี้มีข้อดีที่ให้ผลดีทั้งในด้านความแตกต่างและความราบเรียบของภาพ

บทที่ 8

รูปแบบและผลการทดสอบ

(Modeling a Test Bench)

ในบทนี้จะได้กล่าวถึงรูปแบบและผลการทดสอบการทำงานของระบบ หน่วยความจำและวงจรที่ได้ออกแบบไว้ การกำหนดระดับของการบรรยายนอกจากจะกำหนดไว้ในส่วนของการบรรยายที่มีการอ้างอิงถึงองค์ประกอบอื่นๆแล้ว ยังสามารถเลือกที่จะกำหนดไว้ภายนอกได้อีกด้วยโดยใช้รูปแบบของ **CONFIGURATION** ซึ่งทำให้สะดวกในการปรับเปลี่ยนหรือเลือกที่จะใช้ระดับของการบรรยายขององค์ประกอบใดๆก็ได้

8.1 รูปแบบการทดสอบสถาปัตยกรรมของระบบ

(Configuring for Testing Architecture of System)

8.1.1 แบบทดสอบ (Test Bench)

โปรเซส *processing* เป็นโปรเซสสำหรับการป้อนชุดคำสั่งการทำงานให้กับระบบ

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY modular_test_bench IS
END modular_test_bench;

ARCHITECTURE input_output OF modular_test_bench IS
  COMPONENT modular
  PORT ( clk : IN qit;
        load_image, write_image : IN qit;
        interrupt : IN qit;  -- Signal to start controller module
        instruct : IN byte;  -- instruction
        load_ready : OUT qit;
        read_inst : OUT qit); -- Signal to fetch next instruction
  END COMPONENT;
  SIGNAL clock, interrupt : qit;
  SIGNAL instruct : byte;
  SIGNAL read_inst : qit;
  SIGNAL load_image, load_ready, write_image : qit;
BEGIN
  clk : clock <= NOT clock AFTER 1 US;
  system : modular
    PORT MAP (clock, load_image, write_image, interrupt,
              instruct, load_ready, read_inst);
  processing : PROCESS
  BEGIN

```

```

load_image <= '1';
WAIT UNTIL load_ready = '1';
interrupt <= '1'; WAIT FOR 4 US;
interrupt <= '0';
WAIT UNTIL read_inst = '1';
instruct <= "00000001"; -- clear memory instruction
WAIT UNTIL read_inst = '1';
instruct <= "11110011"; -- row instruction
WAIT UNTIL read_inst = '1';
instruct <= "11111111"; -- 256 rows (0 to 255)
WAIT UNTIL read_inst = '1';
instruct <= "11111100"; -- col instruction
WAIT UNTIL read_inst = '1';
instruct <= "11111111"; -- 256 cols (0 to 255)
WAIT UNTIL read_inst = '1';
instruct <= "00001000"; -- instruction to select module 1
-- (Histogram Equalization)
WAIT UNTIL read_inst = '1';
write_image <= '1';
END PROCESS processing;
END input_output;

```

รูปที่ 8.1 แสดงแบบทดสอบของระบบ

8.1.2 การกำหนดระดับของการบรรยาย (Configuration)

การกำหนดระดับของการบรรยายของระบบในที่นี้จะแสดงการกำหนดในระดับพฤติกรรม โดยเริ่มอ้างอิงจากแบบทดสอบในรูปที่ 8.1 ที่ใช้ชื่อว่า *modular_test_bench* ภายในแบบทดสอบได้อ้างอิงถึงระบบโดยใช้ชื่อว่า *modular* และเลือกระดับการบรรยายแบบ *structural*

ภายในระบบหรือ *modular* ประกอบด้วย 3 โมดูลและ 2 หน่วยความจำ ซึ่งในการทดสอบครั้งนี้ได้เลือกใช้การบรรยายของแต่ละองค์ประกอบอยู่ในระดับพฤติกรรมทั้งหมด ดังแสดงในรูปที่ 8.2 จะเห็นได้ว่าหากเราต้องการทดสอบวงจรในระดับใดก็สามารถเลือกเปลี่ยนได้โดยเพียงการเปลี่ยนระดับการบรรยายที่อยู่ในวงเล็บเท่านั้นโดยจะเปลี่ยนทั้งหมดหรือเลือกเปลี่ยนเฉพาะองค์ประกอบใดองค์ประกอบหนึ่งก็ได้ เช่น ต้องการเปลี่ยนการบรรยายของวงจรควบคุมหลักจากกระดับพฤติกรรมเป็นระดับโครงสร้างก็สามารถทำได้ดังนี้

เปลี่ยนจาก USE ENTITY ctrl.controller (behavioral);
เป็น USE ENTITY ctrl.controller (structural);

CONFIGURATION behavioral OF modular_test_bench IS

FOR input_output

FOR system : modular

USE ENTITY system.modular (structural);

FOR source_memory : memo

USE ENTITY memo.memory (behavioral);

END FOR;

FOR dest_memory : memo

USE ENTITY memo.memory (behavioral);

END FOR;

เอกสารนี้เป็นเอกสารสงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

FOR ctrl_module : ctrl
    USE ENTITY ctrl.controller (behavioral);
END FOR;
FOR histogram_module : module
    USE ENTITY hgram.histogram_equalization (behavioral);
END FOR;
FOR edge_module : module
    USE ENTITY edge.edge_detection (behavioral);
END FOR;
END FOR;
END FOR;
END behavioral;

```

รูปที่ 8.2 แสดงการกำหนดระดับของการบรรยายของระบบ

8.1.3 ผลการทดสอบ (Simulation Result)

เนื่องจากแต่ละโมดูลทำงานเป็นอิสระต่อกันดังนั้นผลจากการทดสอบการเลียนแบบการทำงานจะได้แยกแสดงตามหัวข้อการทดสอบในแต่ละโมดูลต่อไป

8.2 รูปแบบการทดสอบสถาปัตยกรรมของหน่วยความจำ (Configuring for Testing Architecture of Memory)

8.2.1 แบบทดสอบ (Test Bench)

รูปที่ 8.3 แสดงแบบทดสอบการทำงานของวงจรหน่วยความจำโดยการป้อนข้อมูลเข้าทางอินพุตและบันทึกผลการทดสอบที่ได้ลงในไฟล์ชื่อ *memo_ts.dat*

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE STD.TEXTIO.ALL;
USE STD.ALL;

ENTITY memory_test_bench IS
END memory_test_bench;

ARCHITECTURE read_write OF memory_test_bench IS
    COMPONENT m1 PORT ( clk, read, write, clear, enable : IN qit;
                        abus : IN word;
                        dbus : INOUT wired_byte BUS);
    END COMPONENT;
    SIGNAL clock_q, read_q, write_q, clear_q, enable_q : qit;
    SIGNAL addr_q : word;
    SIGNAL data_q : wired_byte := "ZZZZZZZZ";

BEGIN
    enb : enable_q <= '1', '0' AFTER 28 US;

```

```

clk : clock_q <= NOT clock_q AFTER 1 US;
mem : m1 PORT MAP ( clock_q, read_q, write_q, clear_q, enable_q, addr_q, data_q);
s1 : clear_q <= '1', '0' AFTER 6 US;
s2 : read_q <= '0',
      '1' AFTER 6 US,
      '0' AFTER 8 US,
      '1' AFTER 14 US,
      '0' AFTER 16 US,
      '1' AFTER 24 US,
      '0' AFTER 26 US,
      '1' AFTER 30 US,
      '0' AFTER 32 US;

s3 : write_q <= '0',
      '1' AFTER 10 US,
      '0' AFTER 12 US,
      '1' AFTER 20 US,
      '0' AFTER 22 US;

s4 : addr_q <= "0000000000000000" AFTER 6 US,
      "/////////" AFTER 8 US,
      "0000010111111111" AFTER 10 US,
      "/////////" AFTER 12 US,
      "0000010111111111" AFTER 14 US,
      "/////////" AFTER 16 US,
      "1111111110000000" AFTER 20 US,
      "/////////" AFTER 22 US,
      "1111111110000000" AFTER 24 US,
      "/////////" AFTER 26 US,
      "1111111110000000" AFTER 30 US;

s5 : data_q <= "00001111" AFTER 10 US,
      "/////////" AFTER 12 US,
      "11110000" AFTER 20 US,
      "/////////" AFTER 22 US;

```

```

writing : PROCESS (clock_q)
  FILE mdata : TEXT IS OUT "memo_ts.dat";
  VARIABLE l : LINE;
  VARIABLE clock_s, read_s, write_s, clear_s, enable_s : CHARACTER;
  VARIABLE addr_s : STRING (16 DOWNT0 1);
  VARIABLE data_s : STRING (8 DOWNT0 1);
  BEGIN
    qit2char(clock_q, clock_s);
    qit2char(read_q, read_s);
    qit2char(write_q, write_s);
    qit2char(clear_q, clear_s);
    qit2char(enable_q, enable_s);

```

```

qit_vector2str(addr_q, addr_s);
wqit_vector2str(data_q, data_s);

WRITE (l, NOW, RIGHT, 8, NS);
WRITE (l, clock_s, RIGHT, 3);
WRITE (l, read_s, RIGHT, 3);
WRITE (l, write_s, RIGHT, 3);
WRITE (l, clear_s, RIGHT, 3);
WRITE (l, enable_s, RIGHT, 3);
WRITE (l, addr_s, RIGHT, 18);
WRITE (l, data_s, RIGHT, 10);
WRITELINE (mdata, l);
END PROCESS writing;
END read_write;

```

รูปที่ 8.3 แสดงแบบทดสอบของหน่วยความจำ

8.2.2 การกำหนดระดับของการบรรยาย (Configuration)

```

CONFIGURATION structural OF memory_test_bench IS
FOR read_write
FOR mem : m1
USE ENTITY memo.memory (structural);
FOR c0 : s1
USE ENTITY memo.select_memory(dataflow);
END FOR;
FOR c1 : s1
USE ENTITY memo.select_memory(dataflow);
END FOR;
FOR c2 : inv
USE ENTITY gates.inv_q(double_delay);
END FOR;
FOR c3 : a2
USE ENTITY gates.and2_q(double_delay);
END FOR;
FOR c4 : a3
USE ENTITY gates.and3_q(average_delay);
END FOR;
FOR c5to12
FOR cw : ts
USE ENTITY gates.tri_state_q(single_delay);
END FOR;
END FOR;
FOR c13to20
FOR cr : ts
USE ENTITY gates.tri_state_q(single_delay);
END FOR;

```

```

END FOR;
FOR c21 : ma
  USE ENTITY memo.memory_array(structural);
END FOR;
END FOR;
END FOR;
END structural;

```

รูปที่ 8.4 แสดงการกำหนดระดับของการบรรยายของหน่วยความจำ

8.2.3 ผลการทดสอบ (Simulation Result)

รูปที่ 8.5 แสดงผลการทดสอบการทำงานของวงจรหน่วยความจำโดยการป้อนอินพุตและบันทึกผลทุก ๆ คาบของสัญญาณ clock

TIME	ck	r	w	cl	en	addr	data
0 ns	0	0	0	0	0	0000000000000000	00000000
1000 ns	1	0	0	1	1	0000000000000000	00000000
2000 ns	0	0	0	1	1	0000000000000000	00000000
3000 ns	1	0	0	1	1	0000000000000000	00000000
4000 ns	0	0	0	1	1	0000000000000000	00000000
5000 ns	1	0	0	1	1	0000000000000000	00000000
6000 ns	0	1	0	0	1	0000000000000000	00000000
7000 ns	1	1	0	0	1	0000000000000000	00000000
8000 ns	0	0	0	0	1	////////////////	00000000
9000 ns	1	0	0	0	1	////////////////	////////
10000 ns	0	0	1	0	1	0000010111111111	00001111
11000 ns	1	0	1	0	1	0000010111111111	00001111
12000 ns	0	0	0	0	1	////////////////	////////
13000 ns	1	0	0	0	1	////////////////	////////
14000 ns	0	1	0	0	1	0000010111111111	////////
15000 ns	1	1	0	0	1	0000010111111111	00001111
16000 ns	0	0	0	0	1	////////////////	00001111
17000 ns	1	0	0	0	1	////////////////	////////
18000 ns	0	0	0	0	1	////////////////	////////
19000 ns	1	0	0	0	1	////////////////	////////
20000 ns	0	0	1	0	1	1111111110000000	11110000
21000 ns	1	0	1	0	1	1111111110000000	11110000
22000 ns	0	0	0	0	1	////////////////	////////
23000 ns	1	0	0	0	1	////////////////	////////
24000 ns	0	1	0	0	1	1111111110000000	////////
25000 ns	1	1	0	0	1	1111111110000000	11110000
26000 ns	0	0	0	0	1	////////////////	11110000
27000 ns	1	0	0	0	1	////////////////	////////
28000 ns	0	0	0	0	0	////////////////	////////
29000 ns	1	0	0	0	0	////////////////	////////

เอกสารนี้เป็นเอกสารลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี การศึกษาและนำเอกสารนี้ไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

30000	ns	0	1	0	0	0	1111111110000000	ZZZZZZZZ
31000	ns	1	1	0	0	0	1111111110000000	ZZZZZZZZ
32000	ns	0	0	0	0	0	1111111110000000	ZZZZZZZZ
33000	ns	1	0	0	0	0	1111111110000000	ZZZZZZZZ
34000	ns	0	0	0	0	0	1111111110000000	ZZZZZZZZ

รูปที่ 8.5 แสดงผลการทดสอบหน่วยความจำ

8.3 รูปแบบการทดสอบสถาปัตยกรรมของวงจรควบคุมหลัก (Configuring for Testing Architecture of Controller)

8.3.1 แบบทดสอบ (Test Bench)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;
USE STD.TEXTIO.ALL;
USE STD.ALL;

ENTITY controller_test_bench IS
END controller_test_bench;

ARCHITECTURE input_output OF controller_test_bench IS
  COMPONENT ctrl PORT ( clk, interrupt, mod_end : IN qit;
                        instruct : IN byte;
                        clear_mem, read_mem, write_mem : OUT qit;
                        dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
                        abus : OUT word;
                        read_instruction : OUT qit;
                        en_sm, en_dm : OUT qit;
                        m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
                        r_c : OUT qit;
                        r_c_data : OUT word;
                        mod_begin : OUT qit;
                        dbus_on_databus, databus_on_dbus : OUT qit);
  END COMPONENT;

  SIGNAL instruct_q : byte;
  SIGNAL clear_mem_q, read_mem_q, write_mem_q : qit;
  SIGNAL dbus_q : wired_byte BUS := "ZZZZZZZZ";
  SIGNAL abus_q : word;
  SIGNAL read_instruction_q : qit;
  SIGNAL en_sm_q, en_dm_q : qit;
  SIGNAL m1_q, m2_q, m3_q, m4_q, m5_q, m6_q, m7_q, m8_q : qit;
  SIGNAL r_c_q : qit;
  SIGNAL r_c_data_q : word;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

SIGNAL mod_begin_q : qit;
SIGNAL dbus_on_databus_q, databus_on_dbus_q : qit;
BEGIN
int : interrupt_q <= '1', '0' AFTER 4000 NS;
clk : clock_q <= NOT clock_q AFTER 1 US;
cont : ctrl PORT MAP (clock_q, interrupt_q, mod_end_q, instruct_q,
                      clear_mem_q, read_mem_q, write_mem_q, dbus_q, abus_q,
                      read_instruction_q, en_sm_q, en_dm_q,
                      m1_q, m2_q, m3_q, m4_q, m5_q, m6_q, m7_q, m8_q,
                      r_c_q, r_c_data_q, mod_begin_q,
                      dbus_on_databus_q, databus_on_dbus_q);

writing : PROCESS (clock_q)
    FILE ctr1 : TEXT IS OUT "ctrl_ts1.dat";
    FILE ctr2 : TEXT IS OUT "ctrl_ts2.dat";
    VARIABLE l1, l2 : LINE;
    VARIABLE clock_s, interrupt_s, mod_end_s : CHARACTER;
    VARIABLE instruct_s : STRING (8 DOWNT0 1);
    VARIABLE clear_mem_s, read_mem_s, write_mem_s : CHARACTER;
    VARIABLE dbus_s : STRING (8 DOWNT0 1);
    VARIABLE abus_s : STRING (16 DOWNT0 1);
    VARIABLE read_instruction_s : CHARACTER;
    VARIABLE en_sm_s, en_dm_s : CHARACTER;
    VARIABLE m1_s, m2_s, m3_s, m4_s, m5_s, m6_s, m7_s, m8_s : CHARACTER;
    VARIABLE r_c_s : CHARACTER;
    VARIABLE r_c_data_s : STRING (16 DOWNT0 1);
    VARIABLE mod_begin_s : CHARACTER;
    VARIABLE dbus_on_databus_s, databus_on_dbus_s : CHARACTER;

BEGIN
    qit2char(clock_q, clock_s);
    qit2char(interrupt_q, interrupt_s);
    qit2char(mod_end_q, mod_end_s);
    qit_vector2str(instruct_q, instruct_s);
    qit2char(clear_mem_q, clear_mem_s);
    qit2char(read_mem_q, read_mem_s);
    qit2char(write_mem_q, write_mem_s);
    wqit_vector2str(dbus_q, dbus_s);
    qit_vector2str(abus_q, abus_s);
    qit2char(read_instruction_q, read_instruction_s);
    qit2char(en_sm_q, en_sm_s);
    qit2char(en_dm_q, en_dm_s);
    qit2char(m1_q, m1_s);
    qit2char(m2_q, m2_s);
    qit2char(m3_q, m3_s);
    qit2char(m4_q, m4_s);
    qit2char(m5_q, m5_s);

```

เอกสารนี้เป็นเอกสารเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

qit2char(m6_q, m6_s);
qit2char(m7_q, m7_s);
qit2char(m8_q, m8_s);
qit2char(r_c_q, r_c_s);
qit_vector2str(r_c_data_q, r_c_data_s);
qit2char(mod_begin_q, mod_begin_s);
qit2char(dbus_on_databus_q, dbus_on_databus_s);
qit2char(databus_on_dbus_q, databus_on_dbus_s);

```

```

WRITE (l1, NOW, RIGHT, 8, NS);
WRITE (l1, clock_s, RIGHT, 3);
WRITE (l1, interrupt_s, RIGHT, 3);
WRITE (l1, mod_end_s, RIGHT, 3);
WRITE (l1, instruct_s, RIGHT, 10);
WRITE (l1, clear_mem_s, RIGHT, 3);
WRITE (l1, read_mem_s, RIGHT, 3);
WRITE (l1, write_mem_s, RIGHT, 3);
WRITE (l1, dbus_s, RIGHT, 10);
WRITE (l1, abus_s, RIGHT, 18);
WRITE (l1, read_instruction_s, RIGHT, 3);
WRITE (l1, en_sm_s, RIGHT, 3);
WRITE (l1, en_dm_s, RIGHT, 3);

```

```

WRITE (l2, NOW, RIGHT, 8, NS);
WRITE (l2, m1_s, RIGHT, 3);
WRITE (l2, m2_s, RIGHT, 3);
WRITE (l2, m3_s, RIGHT, 3);
WRITE (l2, m4_s, RIGHT, 3);
WRITE (l2, m5_s, RIGHT, 3);
WRITE (l2, m6_s, RIGHT, 3);
WRITE (l2, m7_s, RIGHT, 3);
WRITE (l2, m8_s, RIGHT, 3);
WRITE (l2, r_c_s, RIGHT, 3);
WRITE (l2, r_c_data_s, RIGHT, 20);
WRITE (l2, mod_begin_s, RIGHT, 3);
WRITE (l2, dbus_on_databus_s, RIGHT, 3);
WRITE (l2, databus_on_dbus_s, RIGHT, 3);

```

```

WRITELINE (ctr1, l1);
WRITELINE (ctr2, l2);
END PROCESS writing;

```

```
inst : PROCESS
```

```
BEGIN
```

```
WAIT UNTIL read_instruction_q = '1';
```

```
instruct_q <= "00000001"; -- clear memory instruction
```

```
WAIT UNTIL read_instruction_q = '1';
```

เอกสารนี้เป็นเอกสารที่จัดทำขึ้นเพื่อใช้ในการศึกษาเท่านั้น เมื่อนำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

instruct_q <= "11110011"; -- row instruction
WAIT UNTIL read_instruction_q = '1';
instruct_q <= "00000010"; -- 3 rows
WAIT UNTIL read_instruction_q = '1';
instruct_q <= "11111100"; -- column instruction
WAIT UNTIL read_instruction_q = '1';
instruct_q <= "00000011"; -- 4 cols
WAIT UNTIL read_instruction_q = '1';
instruct_q <= "00001000"; -- module 1 instruction
WAIT UNTIL mod_begin_q = '1';
WAIT FOR 5 US;
- mod_end_q <= '1';
WAIT UNTIL read_instruction_q = '1';
instruct_q <= "00000010"; -- swap memory instruction
WAIT UNTIL read_instruction_q = '1';
instruct_q <= "00001100"; -- module 5 instruction
WAIT UNTIL mod_begin_q = '1';
WAIT FOR 5 US;
mod_end_q <= '1';
END PROCESS inst;
END input_output;

```

รูปที่ 8.6 แสดงแบบทดสอบของวงจรควบคุมหลัก

โปรเซส *writing* สำหรับบันทึกผลการเขียนแบบการทำงานลงในไฟล์เนื่องจากวงจรนี้มีอินพุตและเอาต์พุตค่อนข้างมากจึงได้แยกบันทึกผลเป็น 2 ไฟล์คือ *ctrl_ts1.dat* และ *ctrl_ts2.dat* โปรเซส *inst* สำหรับป้อนคำสั่งเข้าสู่วงจร

8.3.2 การกำหนดระดับของการบรรยาย (Configuration)

```

CONFIGURATION structural OF controller_test_bench IS
  FOR input_output
    FOR cont : ctrl
      USE ENTITY ctrl.controller (structural);
    FOR logic : l_ctrl
      USE ENTITY ctrl.logic_of_controller (dataflow);
    END FOR;
  FOR counter : pc
    USE ENTITY reg.program_counter_unit (behavioral);
  END FOR;
  FOR memory_addr : mar
    USE ENTITY reg.memory_address_register_unit (behavioral);
  END FOR;
  FOR compare : cprc
    USE ENTITY reg.compare_row_col (dataflow);
  END FOR;
  FOR ALL : reg
    USE ENTITY reg.register_unit (dataflow);
  END FOR;
  FOR ALL : ts8
    USE ENTITY reg.tri_state_8 (structural);
  END FOR;
  FOR ALL : ts16
    USE ENTITY reg.tri_state_16 (structural);
  END FOR;
END FOR;
END FOR;
END structural;

```

รูปที่ 8.7 แสดงการกำหนดระดับของการบรรยายของวงจรควบคุมหลัก

8.3.3 ผลการทดสอบ (Simulation Result)

ในการทดสอบได้ป้อนสัญญาณอินพุตซึ่งเป็นคำสั่งต่าง ๆ เข้าไปแล้วบันทึกผลการตอบสนองที่ออกมาทางเอาต์พุตโดยบันทึกผลลงในไฟล์ ดังแสดงผลการทดสอบวงจรควบคุมหลักในรูปที่ 8.8

ck เป็นสัญญาณนาฬิกา, int คือสัญญาณ interrupt, md_end คือสัญญาณ mod_end, inst คือสัญญาณ instruction, cl คือสัญญาณ clear, r คือสัญญาณ read_memory, w คือสัญญาณ write_memory, dbus คือสัญญาณ databus, abus คือสัญญาณ address_bus, rd_inst คือสัญญาณ read_instruction, en_sm คือสัญญาณ enable source memory, en_dm คือสัญญาณ enable destination memory, m1-m8 คือสัญญาณกระตุ้นการทำงานของไมโครทั้ง 8 , r_c คือสัญญาณ row_col, r_c_data คือสัญญาณแสดงขนาดของภาพ, md_beg คือสัญญาณ mod_begin การเปลี่ยนแปลงของสัญญาณจะเป็นไปตามคำสั่ง(instruction)ที่ได้รับ

md_		rd en m m m m m m m m m m										md_dbus_on databus							
TIME	ck lint end	inst	cl r w	dbus	abus	ins sm dm	1	2	3	4	5	6	7	8	r_c	r_c_data	beg	databus	on_dbus
31000 ns	1 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000000000000000	0	0	0
32000 ns	0 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000000000000000	0	0	0
33000 ns	1 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	1	000000100000000011	0	0	0
34000 ns	0 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	1	000000100000000011	0	0	0
35000 ns	1 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	0	0	0
36000 ns	0 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	0	0	0
37000 ns	1 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	1 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
38000 ns	0 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	1 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
39000 ns	1 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
40000 ns	0 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
41000 ns	1 0 0	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
42000 ns	0 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
43000 ns	1 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
44000 ns	0 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
45000 ns	1 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
46000 ns	0 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
47000 ns	1 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
48000 ns	0 0 1	00001000	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
49000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	1 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
50000 ns	0 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	1 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
51000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
52000 ns	0 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
53000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
54000 ns	0 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0
55000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	1
56000 ns	0 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	1
57000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 1 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	1	0
58000 ns	0 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 1 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	1	0
59000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 1 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	1	0
60000 ns	0 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 1 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	1	0
61000 ns	1 0 1	00000010	0 0 0	ZZZZZZZZ	000000000000000000	0 0 0	1	0	0	0	0	0	0	0	0	000000100000000011	1	0	0

TIME	md_		inst	d	r	w	dbus	abus	rd								r_c								md_		dbus	
	ck	int							en	sm	dm	1	2	3	4	5	6	7	8	m	m	m	m	beg	on			
62000 ns	0	0	1	00000010	0	1	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	0	0
63000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	0	1
64000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	0	1
65000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
66000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
67000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
68000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
69000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
70000 ns	0	0	1	00000010	0	1	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
71000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
72000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
73000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
74000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
75000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
76000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
77000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
78000 ns	0	0	1	00000010	0	1	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
79000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
80000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
81000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
82000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
83000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
84000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
85000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
86000 ns	0	0	1	00000010	0	1	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
87000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
88000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
89000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
90000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
91000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0
92000 ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000000000000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0000001000000000	1	1	1	0

md_		rd en en m m m m m m m m m m m m m m m m																md_dbus_on_databus							
TIME	ck	int	end	inst	d	r	w	dbus	abus	ins	sm	dm	1	2	3	4	5	6	7	8	r_c	r_c_data	beg	databus	on_dbus
93000 ns	1	0	1	00000010	0	1	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0
94000 ns	0	1	00000010	0	1	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0
95000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	1
96000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	1
97000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
98000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
99000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
100000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
101000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
102000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
103000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
104000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
105000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
106000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
107000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
108000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
109000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
110000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
111000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
112000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
113000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
114000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
115000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
116000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000001	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
117000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
118000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
119000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
120000 ns	0	1	00000010	0	0	ZZZZZZZZ	0000000100000000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
121000 ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000000100000000	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
122000 ns	0	1	00000010	0	1	ZZZZZZZZ	0000000100000000	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0
123000 ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000000100000000	0	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	1	0

md_		rd										md_dbus_on														
TIME	ck	int	end	inst	cl	r	w	dbus	abus	lms	sm	dm	1	2	3	4	5	6	7	8	r_c	r_c_data	beg	dbus	on	dbus
124000	ns	0	0	1	00000010	0	0	ZZZZZZZZ	0000001000000000	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000000	1	1	0	0
125000	ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000001000000001	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
126000	ns	0	1	00000010	0	1	0	ZZZZZZZZ	0000001000000001	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
127000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000001	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	1	0
128000	ns	0	1	00000010	0	0	0	ZZZZZZZZ	0000001000000001	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	1	0
129000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000001	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
130000	ns	0	1	00000010	0	0	1	ZZZZZZZZ	0000001000000001	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
131000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000001	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
132000	ns	0	1	00000010	0	0	0	ZZZZZZZZ	0000001000000001	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
133000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000001	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
134000	ns	0	1	00000010	0	1	0	ZZZZZZZZ	0000001000000010	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
135000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000010	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
136000	ns	0	1	00000010	0	0	0	ZZZZZZZZ	0000001000000010	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	1	0
137000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000010	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
138000	ns	0	1	00000010	0	0	1	ZZZZZZZZ	0000001000000010	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
139000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000010	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
140000	ns	0	1	00000010	0	0	0	ZZZZZZZZ	0000001000000010	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
141000	ns	1	0	1	00000010	0	1	ZZZZZZZZ	0000001000000011	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
142000	ns	0	1	00000010	0	1	0	ZZZZZZZZ	0000001000000011	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
143000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000011	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	1	0
144000	ns	0	1	00000010	0	0	0	ZZZZZZZZ	0000001000000011	0	0	1	1	0	0	0	0	0	0	0	0	0000001000000001	1	0	1	0
145000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000011	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
146000	ns	0	1	00000010	0	0	1	ZZZZZZZZ	0000001000000011	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
147000	ns	1	0	1	00000010	0	0	ZZZZZZZZ	0000001000000011	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
148000	ns	0	1	00000010	0	0	0	ZZZZZZZZ	0000001000000011	0	1	0	1	0	0	0	0	0	0	0	0	0000001000000001	1	1	0	0
149000	ns	1	0	1	00001100	0	0	ZZZZZZZZ	0000001000000011	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
150000	ns	0	1	00001100	0	0	0	ZZZZZZZZ	0000001000000011	1	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
151000	ns	1	0	1	00001100	0	0	ZZZZZZZZ	0000001000000011	0	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
152000	ns	0	1	00001100	0	0	0	ZZZZZZZZ	0000001000000011	0	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
153000	ns	1	0	1	00001100	0	0	ZZZZZZZZ	0000001000000011	0	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0
154000	ns	0	1	00001100	0	0	0	ZZZZZZZZ	0000001000000011	0	0	1	0	0	0	0	0	0	0	0	0	0000001000000001	1	0	0	0

8.4 รูปแบบการทดสอบสถาปัตยกรรมของวงจรปรับปรุงคุณภาพ

(Configuring for Testing Architecture of Histogram Equalization)

8.4.1 แบบทดสอบ (Test Bench)

ขั้นตอนทดสอบเริ่มด้วยการอ่านข้อมูลของภาพต้นแบบเข้าสู่หน่วยความจำต้นทาง ทำการประมวลผล เก็บผลที่ได้เข้าสู่หน่วยความจำปลายทางและสุดท้ายเขียนข้อมูลภาพจากหน่วยความจำปลายทางลงสู่ไฟล์ที่กำหนด

```

LIBRARY util;
LIBRARY hgram;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;
USE STD.TEXTIO.ALL;
USE STD.ALL;

ENTITY histogram_test_bench IS
END histogram_test_bench;

ARCHITECTURE input_output OF histogram_test_bench IS
  COMPONENT hist PORT ( clk, interrupt, r_c, mod_begin : IN qit;
                        r_c_data : IN word;
                        dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
                        read_mem, write_mem : OUT qit;
                        en_sm, en_dm, mod_end : OUT qit;
                        abus : OUT word;
                        dbus_on_databus, databus_on_dbus : OUT qit);
  END COMPONENT;

  FOR histogram : hist USE ENTITY hgram.histogram_equalization(behavioral);
  TYPE mem_array IS ARRAY (0 TO 255, 0 TO 255) OF INTEGER;
  SIGNAL clock, interrupt, r_c, mod_begin : qit;
  SIGNAL r_c_data : word;
  SIGNAL data : wired_byte := "ZZZZZZZZ";
  SIGNAL sread, swrite, en_sm, en_dm, mod_end : qit;
  SIGNAL addr : word;
  SIGNAL dbus_on_databus, databus_on_dbus : qit;
BEGIN
  clk : clock <= NOT clock AFTER 1 US;
  histogram : hist PORT MAP ( clock, interrupt, r_c, mod_begin,
                              r_c_data, data, sread, swrite,
                              en_sm, en_dm, mod_end, addr,
                              dbus_on_databus, databus_on_dbus);

  mem : PROCESS
    FILE sour : TEXT IS IN "his_ts.img";
    FILE dest : TEXT IS OUT "his_ts.dat";
    VARIABLE sl, dl : LINE;

```

```

VARIABLE smem, dmem : mem_array;
VARIABLE ram_init : BOOLEAN := FALSE;
VARIABLE i, j : INTEGER := 0;
VARIABLE ia, ib : INTEGER;
VARIABLE page, offset, mems : byte;
VARIABLE memd : INTEGER;
BEGIN
  -- read data from file into source memory
  IF ram_init = FALSE THEN
    WHILE NOT ENDFILE (sour) AND (i < 256) LOOP
      READLINE (sour, sl);
      WHILE (j < 256) LOOP
        READ (sl, smem(i,j));
        j := j + 1;
      END LOOP;
      j := 0;
      i := i + 1;
    END LOOP;
    ram_init := TRUE;
    interrupt <= '1'; WAIT FOR 4 US;
    r_c <= '1';
    r_c_data <= '1111111111111111'; WAIT FOR 2 US;
    r_c <= '0'; WAIT FOR 2 US;
    mod_begin <= '1';
  END IF;
  -- write data from destination memory to file
  IF mod_end = '1' THEN
    i := 0;
    j := 0;
    WHILE i < 256 LOOP
      WHILE j < 256 LOOP
        WRITE (dl, dmem(i,j), RIGHT, 4);
        j := j + 1;
      END LOOP;
      WRITELINE (dest, dl);
      j := 0;
      i := i + 1;
    END LOOP;
  END IF;
  -- read and write memory inside histogram equalization process
  WAIT ON sread, swrite;
  page := addr(15 DOWNT0 8);
  offset := addr(7 DOWNT0 0);
  qit2int (page, ia);
  qit2int (offset, ib);
  IF (en_sm = '1') AND (sread = '1') THEN
    int2qit (smem(ia, ib), mems);
  
```

```

data <= wired_byte (mems);
WAIT UNTIL sread = '0';
data <= "ZZZZZZZZ";
ELSIF (en_dm = '1') AND (swrite = '1') THEN
  qit2int (byte(data), memd);
  dmem(ia, ib) := memd;
  WAIT UNTIL swrite = '0';
END IF;
END PROCESS mem;
END input_output;

```

รูปที่ 8.9 แสดงแบบทดสอบของวงจรปรับปรุงคุณภาพ

8.4.2 การกำหนดระดับของการบรรยาย (Configuration)

```

CONFIGURATION structural OF edge_detection_test_bench IS
  FOR input_output
    FOR histogram : hist
      USE ENTITY hgram.histogram_equalization (structural);
      FOR control : ctrl_unit
        USE ENTITY hgram.histogram_controller(dataflow);
      END FOR;
      FOR pixel : pxc
        USE ENTITY hgram.pixel_counter(structural);
      END FOR;
      FOR dec : decoder_unit
        USE ENTITY hgram.decoder_8_to_256(dataflow);
      END FOR;
      FOR new_data : t8_256
        USE ENTITY hgram.tri_state_8_256(structural);
      END FOR;
      FOR ALL : t256
        USE ENTITY hgram.tri_state_256(structural);
      END FOR;
      FOR new_pixel : new_gray_level
        USE ENTITY hgram.new_level(structural);
      END FOR;
      FOR ALL : reg
        USE ENTITY reg.register_unit(dataflow);
      END FOR;
      FOR row_counter, col_counter : rcpc
        USE ENTITY reg.row_col_counter_unit(behavioral);
      END FOR;
      FOR memory_addr : mar
        USE ENTITY reg.memory_address_register_unit(behavioral);
      END FOR;
      FOR compare_rc : cprc

```

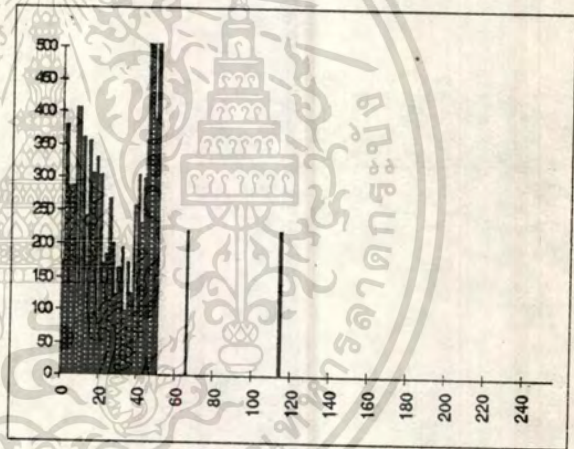
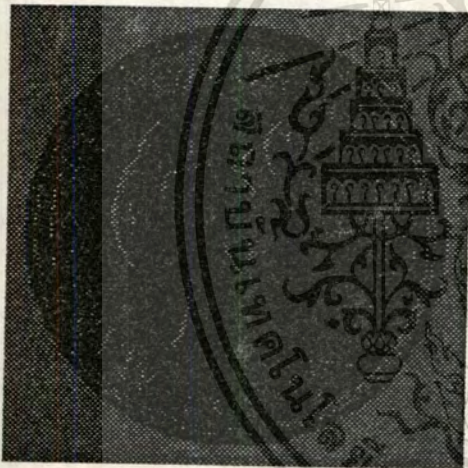
```

USE ENTITY reg.compare_row_col(dataflow);
END FOR;
FOR compare_c : cpc
  USE ENTITY reg.compare_col(dataflow);
END FOR;
FOR ALL : ts8
  USE ENTITY reg.tri_state_8(structural);
END FOR;
FOR ALL : ts16
  USE ENTITY reg.tri_state_16(structural);
END FOR;
END FOR;
END structural;

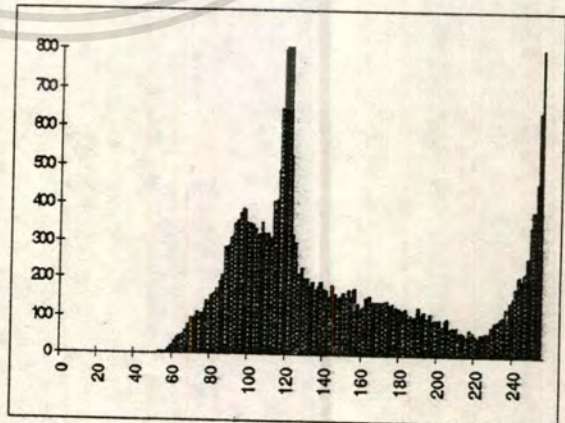
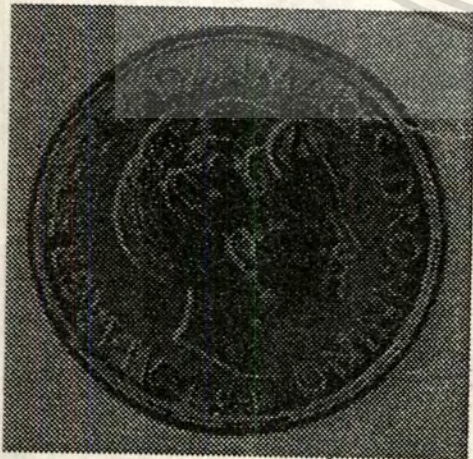
```

รูปที่ 8.10 แสดงการกำหนดระดับของการบรรยายของวงจรปรับปรุงคุณภาพ

8.4.3 ผลการทดสอบ (Simulation Result)



(a) ภาพต้นแบบและฮิสโตแกรมของภาพ



(b) ภาพที่ผ่านการปรับปรุงแล้วและฮิสโตแกรมของภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้รูปที่ 8.11 แสดงผลการทดสอบวงจรปรับปรุงคุณภาพไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 8.11 แสดงผลการทดสอบเปรียบเทียบของภาพต้นแบบและภาพที่ผ่านการประมวลผลของวงจรปรับปรุงคุณภาพแล้ว ภาพที่ใช้เป็นภาพที่มีขนาด 256x256 จุดภาพและฮิสโตแกรมแสดงการกระจายของระดับความเข้มทั้งก่อนและหลังการปรับปรุงคุณภาพ ภาพต้นแบบมีลักษณะค่อนข้างมืดและมีการกระจายของฮิสโตแกรมแคบ ๆ รวมกันอยู่ทางด้านมืด เมื่อผ่านกระบวนการปรับปรุงคุณภาพของภาพแล้วจะเห็นได้ว่าฮิสโตแกรมมีการกระจายออกมาทางด้านสว่างและภาพที่ได้ก็มีความสว่างมากขึ้นทำให้สามารถเห็นรายละเอียดของภาพได้ชัดเจนยิ่งขึ้น

8.5 รูปแบบการทดสอบสถาปัตยกรรมของวงจรตรวจหาขอบของวัตถุ (Configuring for Testing Architecture of Edge Detection)

8.5.1 แบบทดสอบ (Test Bench)

```

LIBRARY util;
LIBRARY edge;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;
USE STD.TEXTIO.ALL;

ENTITY edge_detection_test_bench IS
END edge_detection_test_bench;

ARCHITECTURE input_output OF edge_detection_test_bench IS
    COMPONENT edge PORT ( clk, interrupt, r_c, mod_begin : IN qit;
                          r_c_data : IN word;
                          dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
                          read_mem, write_mem : OUT qit;
                          en_sm, en_dm, mod_end : OUT qit;
                          abus : OUT word;
                          dbus_on_databus, databus_on_dbus : OUT qit);
    END COMPONENT;

    FOR detect : edge USE ENTITY edge.edge_detection(behavioral);

    SIGNAL clock, interrupt, r_c, mod_begin : qit;
    SIGNAL r_c_data : word;
    SIGNAL data : wired_byte := "ZZZZZZZZ";
    SIGNAL sread, swrite, en_sm, en_dm, mod_end : qit;
    SIGNAL addr : word;
    SIGNAL dbus_on_databus, databus_on_dbus : qit;
    TYPE int_mem IS ARRAY (0 TO 255, 0 TO 255) OF INTEGER;
BEGIN
    clk : clock <= NOT clock AFTER 1 US;
    detect : edge PORT MAP ( clock, interrupt, r_c, mod_begin,
                          r_c_data, data, sread, swrite,
                          en_sm, en_dm, mod_end, addr,
                          dbus_on_databus, databus_on_dbus);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

mem : PROCESS
  FILE source : TEXT IS IN "edg_ts.img";
  FILE dest : TEXT IS OUT "edg_ts.dat";
  VARIABLE sl, dl : LINE;
  VARIABLE ia, ib : INTEGER;
  VARIABLE smem, dmem : int_mem;
  VARIABLE ram_init : BOOLEAN := FALSE;
  VARIABLE i, j : INTEGER := 0;
  VARIABLE page, offset, mems : byte;
  VARIABLE memd : INTEGER;
BEGIN
  -- read data from file into source memory
  IF ram_init = FALSE THEN
    WHILE NOT ENDFILE (source) AND (i < 256) LOOP
      READLINE (source, sl);
      WHILE (j < 256) LOOP
        READ (sl, smem(i,j));
        dmem(i,j) := smem(i,j);
        j := j + 1;
      END LOOP;
      i := i + 1;
    END LOOP;
    ram_init := TRUE;
    interrupt <= '1'; WAIT FOR 4 US;
    r_c <= '1';
    r_c_data <= "1111111111111111"; WAIT FOR 2 US;
    r_c <= '0'; WAIT FOR 2 US;
    mod_begin <= '1';
  END IF;
  -- write data from destination memory to file
  IF mod_end = '1' THEN
    i := 0;
    j := 0;
    WHILE i < 256 LOOP
      WHILE j < 256 LOOP
        WRITE (dl, dmem(i,j), RIGHT, 4);
        j := j + 1;
      END LOOP;
      WRITELINE (dest, dl);
      j := 0;
      i := i + 1;
    END LOOP;
  END IF;
  -- read and write memory inside edge detection process
  WAIT ON sread, swrite;
  page := addr(15 DOWNT0 8);

```

```

offset := addr(7 DOWNT0 0);
qit2int (page, ia);
qit2int (offset, ib);
IF (en_sm = '1') AND (sread = '1') THEN
    int2qit (smem(ia, ib), mems);
    data <= wired_byte (mems);
    WAIT UNTIL sread = '0';
    data <= "ZZZZZZZZ";
ELSIF (en_dm = '1') AND (swrite = '1') THEN
    qit2int (byte(data), memd);
    dmem(ia, ib) := memd;
    WAIT UNTIL swrite = '0';
END IF;
END PROCESS mem;
END input_output;

```

รูปที่ 8.12 แสดงแบบทดสอบของวงจรตรวจหาขอบของวัตถุ

8.5.2 การกำหนดระดับของการบรรยาย (Configuration)

```

CONFIGURATION structural OF edge_detection_test_bench IS
FOR input_output
FOR detect : edge
    USE ENTITY edge.edge_detection (structural);
FOR control : ctrl_unit
    USE ENTITY edge.edge_controller (dataflow);
END FOR;
FOR sobel_address : sar
    USE ENTITY edge.sobel_address_register_unit(behavioral);
END FOR;
FOR sobel_matrix : sdr
    USE ENTITY edge.sobel_data_register_unit(dataflow);
END FOR;
FOR gradient : grd
    USE ENTITY edge.gradient(behavioral);
END FOR;
FOR ALL : reg
    USE ENTITY reg.register_unit(dataflow);
END FOR;
FOR row_counter : rcp
    USE ENTITY reg.row_col_counter_unit(behavioral);
END FOR;
FOR col_counter : rcp
    USE ENTITY reg.row_col_counter_unit(behavioral);
END FOR;
FOR memory_addr : mar
    USE ENTITY reg.memory_address_register_unit(behavioral);

```

```

END FOR;
FOR compare_rc : cprc
  USE ENTITY reg.compare_row_col(dataflow);
END FOR;
FOR compare_c : cpc
  USE ENTITY reg.compare_col(dataflow);
END FOR;
FOR ALL : ts8
  USE ENTITY reg.tri_state_8(structural);
END FOR;
FOR ALL : ts16
  USE ENTITY reg.tri_state_16(structural);
END FOR;
END FOR;
END FOR;
END structural;

```

รูปที่ 8.13 แสดงการกำหนดระดับของการบรรยายของวงจรตรวจหาขอบของวัตถุ

8.5.3 ผลการทดสอบ (Simulation Result)



รูปที่ 8.14 แสดงผลการทดสอบวงจรตรวจหาขอบของวัตถุ
(a) ภาพต้นแบบ (b) ภาพที่ผ่านการตรวจหาขอบของวัตถุแล้ว

รูปที่ 8.14 แสดงผลการทดสอบของวงจรตรวจหาขอบของวัตถุ แสดงการเปรียบเทียบภาพต้นแบบและภาพที่ได้จากการประมวลผลซึ่งขอบของวัตถุในภาพจะถูกเน้นให้เห็นเด่นชัดขึ้น

8.6 สรุป (Summary)

ในบทนี้ได้กล่าวถึงวิธีการทดสอบการทำงานของวงจรต่าง ๆ โดยวิธีเลียนแบบการทำงานบนเครื่องคอมพิวเตอร์ แบบทดสอบของวงจรจะเป็นตัวกำหนดลักษณะของสัญญาณและข้อมูลที่ใช้เป็นเพื่อป้อนให้กับวงจร รูปแบบของสัญญาณและข้อมูลสามารถกำหนดได้โดยตรงหรือจะกำหนดให้มีการ

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

อ่านมาจากไฟล์ภายนอกก็ได้เช่นในกรณีของการอ่านข้อมูลภาพจากไฟล์ภายนอกในแบบทดสอบของวงจรประมวลผลภาพ ผลลัพธ์ที่ได้ก็สามารถเก็บลงบนไฟล์ได้เช่นเดียวกัน

ในกรณีของวงจรหน่วยความจำและวงจรควบคุมหลักได้แสดงการป้อนสัญญาณไว้ในแบบทดสอบและแสดงผลที่ได้ในลักษณะของตาราง ส่วนกรณีของวงจรการประมวลผลภาพทั้งสองได้แสดงขั้นตอนในการป้อนสัญญาณและโปรแกรมสำหรับการอ่านและเขียนข้อมูลภาพจากไฟล์ภายนอกไว้ด้วย และผลลัพธ์ที่ได้ก็ได้แสดงไว้ในลักษณะของภาพเพื่อเป็นการเปรียบเทียบระหว่างภาพต้นแบบและภาพที่ได้ผ่านการประมวลผลมาแล้วไว้ด้วยกัน สำหรับภาพที่มีขนาดแตกต่างออกไปก็ยังคงสามารถใช้แบบทดสอบได้เหมือนกันเพียงแต่เปลี่ยนข้อมูลที่เป็นตัวกำหนดขนาดของภาพให้สอดคล้องกันเท่านั้น

นอกจากแบบทดสอบและผลการทดสอบที่ได้แสดงไว้แล้ว ยังได้แสดงรูปแบบของการกำหนดระดับของการบรรยายของแต่ละวงจรไว้ด้วย ซึ่งการกำหนดระดับนี้สามารถใช้ได้กับทุกระดับชั้นของการบรรยายโดยเลือกกำหนดระดับชั้นของการบรรยายในแต่ละวงจรได้ตั้งแต่ระดับบน(behavioral)จนถึงระดับล่าง(structural)ได้ตามต้องการ



บทที่ 9

สรุปและวิจารณ์

(Summary and Discusses)

สิ่งที่วิทยานิพนธ์ฉบับนี้ได้นำเสนอไปทั้งหมดรวม 8 บทนั้นมีวัตถุประสงค์เพื่อต้องการออกแบบระบบวงจรรวมที่นำไปใช้ในการประมวลผลภาพ ซึ่งสามารถจะประมวลผลภาพได้หลาย ๆ ลักษณะ อีกทั้งยังสามารถปรับเปลี่ยนตัววงจรเพื่อให้เหมาะสมกับงานเฉพาะอย่างได้ด้วย แนวความคิดหลักของงานวิจัยในครั้งนี้ประกอบด้วย 3 ส่วนที่สำคัญด้วยกันคือ การนำเอาแนวความคิดแบบสถาปัตยกรรมแบบโมดูลาร์เข้ามาใช้ในการวางรูปแบบของระบบ การนำเอาความสามารถในการออกแบบวงจรในลักษณะของการบรรยายการทำงานเชิงข้อความของภาษา VHDL เข้ามาใช้เพื่อเพิ่มประสิทธิภาพและลดขั้นตอนในการออกแบบให้น้อยลงและสุดท้ายการนำเอาทฤษฎีและอัลกอริธึมต่าง ๆ ในการประมวลผลภาพมาประยุกต์ใช้ในลักษณะของวงจรฮาร์ดแวร์เพื่อเพิ่มประสิทธิภาพและความเร็วในการประมวลผลข้อมูล

การนำเอาแนวความคิดแบบสถาปัตยกรรมแบบโมดูลาร์เข้ามาใช้ในการวางรูปแบบของระบบเพื่อให้เกิดความอิสระในการทำงานและความคล่องตัวในการปรับเปลี่ยนและการนำไปประยุกต์ใช้ให้เหมาะสมกับงานเฉพาะอย่างได้ดี การออกแบบเพิ่มเติมก็สามารถทำได้ง่ายและประหยัดกว่า สามารถตอบสนองต่อหน้าที่และรูปแบบการใช้งานได้ตรงเป้าหมายมากกว่า ลักษณะทางโครงสร้างทางสถาปัตยกรรมแบบโมดูลาร์ที่ใช้กับงานวิจัยในวิทยานิพนธ์ฉบับนี้ได้ถูกออกแบบให้สามารถบรรจุวงจรรวมสำหรับการประมวลผลภาพที่แตกต่างกันได้จำนวน 8 วงจรซึ่งแต่ละวงจรรวมจะทำงานเป็นอิสระต่อกันและสามารถถอดเปลี่ยนได้ตามลักษณะของงาน การควบคุมการทำงานจะมีวงจรควบคุมหลักที่ใช้ในการสั่งงานและควบคุมลำดับขั้นตอนของการประมวลผลภาพให้เป็นไปตามลำดับคำสั่งที่ได้รับมา มีหน่วยความจำที่ใช้ในการเก็บข้อมูลภาพทั้งภาพต้นแบบและภาพที่ได้จากการประมวลผลโดยจะใช้หน่วยความจำในการทำงานร่วมกัน การติดต่อระหว่างวงจรควบคุมหลัก หน่วยความจำ และวงจรประมวลผลภาพต่าง ๆ จะกระทำผ่านทางบัสแอดเดรส บัสข้อมูลและบัสควบคุมร่วมกัน

ความสามารถในการบรรยายเชิงพฤติกรรม(Behavioral Description) การบรรยายเชิงกระแสข้อมูล (Dataflow Description) และการบรรยายเชิงโครงสร้าง(Structural Description) ทำให้ลดลำดับขั้นตอนและความยุ่งยากในการออกแบบลงไปได้มาก ลำดับการออกแบบมีความเป็นระบบและมีระเบียบมากกว่าทำให้การตรวจสอบความถูกต้องสามารถทำได้ง่ายและการแก้ไขสามารถทำได้โดยรวดเร็ว ความยืดหยุ่นและการไม่ถูกจำกัดให้ยึดติดกับเทคโนโลยีใด ๆ ทำให้ประหยัดเวลาและลดค่าใช้จ่ายในการออกแบบในการปรับปรุงและเปลี่ยนแปลงเพื่อให้ก้าวตามทันความสามารถในทางเทคโนโลยีใหม่ ๆ ที่จะมีความมาในอนาคตก็ตาม สิ่งที่ได้กล่าวมาทั้งหมดนี้ล้วนแต่เป็นเหตุผลที่นำมาใช้ในการตัดสินใจเลือกใช้ภาษา VHDL ในการทำงานวิจัยครั้งนี้

ทฤษฎีและอัลกอริธึมในการประมวลผลภาพที่นำมาใช้ในการออกแบบวงจรรวมฮาร์ดแวร์ในงานวิจัยครั้งนี้คือ เทคนิคในการปรับปรุงคุณภาพของภาพโดยวิธี Histogram Equalization และเทคนิคในการจำแนกรายละเอียดของวัตถุในภาพโดยวิธี Edge Detection

การปรับปรุงคุณภาพของภาพโดยวิธี Histogram Equalization เป็นวิธีการที่เน้นแสดงรายละเอียดของภาพให้เห็นเด่นชัดขึ้น โดยเพิ่มความแตกต่างของระดับความเข้ม(Histogram Equalization) เป็นวิธีที่ใช้ในการแปลงระดับความเข้ม (gray level) ของพิกเซลแต่ละพิกเซลให้อยู่ในค่าที่เหมาะสมเพื่อเน้นให้เห็นความแตกต่างของภาพให้เด่นชัดขึ้น ซึ่งบางพิกเซลอาจจะถูกแปลงให้มีระดับความเข้มสูงขึ้นหรือต่ำลงหรือบางพิกเซลอาจจะไม่มีการเปลี่ยนแปลงระดับความเข้มเลยก็ได้

การตรวจหาขอบของวัตถุในภาพโดยวิธี Edge Detection เป็นหลักการที่สำคัญอย่างหนึ่งที่มักจะนำมาใช้ในการประมวลผลภาพเพื่อการเปรียบเทียบ การจดจำหรือเพื่อการจำแนกรายละเอียดของวัตถุในภาพ โดยอาศัยหลักการพื้นฐานของความเหมือนหรือต่างกันและความไม่ต่อเนื่องของระดับสีของจุดภาพ(pixels) วิธีการที่นำมาใช้ก็คือการหาค่าอนุพันธ์อันดับที่หนึ่งของภาพโดยหาค่าเกรเดียนต์ (gradient) ของภาพ $f(x,y)$ ที่ตำแหน่ง (x,y) ในวิทยานิพนธ์นี้ได้เลือกใช้วิธีของโซเบล(Sobel operators) ซึ่งวิธีนี้มีข้อดีที่ช่วยลดทั้งในขั้นตอนการหาความแตกต่างและความราบเรียบของภาพ (differencing and smoothing effect)

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งยังมีให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เนื่องจากการหาอนุพันธ์จะไปขยายข้อมูลรบกวน(noise)ที่มีอยู่ในภาพให้เพิ่มมากขึ้น ดังนั้นเรื่องผลของความราบเรียบจึงเป็นคุณสมบัติที่น่าสนใจของวิธีโซเบล

ในการออกแบบทั้งสองวงจรซึ่งได้แสดงไว้ในบทที่ 6 และบทที่ 7 ลำดับขั้นตอนการออกแบบจะเริ่มตั้งแต่การออกแบบในระดับบนคือการบรรยายในเชิงพฤติกรรมเพื่อยืนยันความถูกต้องในแนวความคิดและลำดับวิธีการประมวลผล จากนั้นจึงเข้าสู่การบรรยายในเชิงข้อมูลและเชิงโครงสร้างในขั้นตอนสุดท้าย ผลการออกแบบได้วงจรรวมสำหรับการประมวลผลภาพที่สามารถนำมาใช้ในการประมวลผลกับข้อมูลภาพที่มีขนาดไม่เกิน 256X256 จุดภาพและใช้กับภาพที่มีระดับความเข้มแตกต่างกันได้ถึง 256 ระดับ

สำหรับวงจรควบคุมหลักได้ออกแบบไว้ให้ทำหน้าที่ในการควบคุมและประสานการทำงานให้เป็นไปตามลำดับคำสั่งที่ได้รับ แต่ก็มีได้เข้าไปก้าวก่ายหน้าที่ในการควบคุมการทำงานภายในของแต่ละโมดูล วงจรควบคุมหลักเพียงแต่ทำหน้าที่ในการแปลและปฏิบัติตามคำสั่งที่ได้รับในส่วนที่เกี่ยวข้องกับตนเองเท่านั้น ในส่วนที่เกี่ยวข้องกับโมดูลอื่น ๆ จะทำหน้าที่ในการกระตุ้นและโอนย้ายการควบคุมให้กับวงจรควบคุมที่อยู่ภายในแต่ละโมดูลและจะรับมอบการควบคุมกลับมาเมื่อได้รับแจ้งการเสร็จสิ้นการปฏิบัติการกิจของแต่ละโมดูลแล้ว สำหรับวงจรหน่วยความจำทั้งสองวงจรจะมีลักษณะการทำงานเหมือนกันและมีเพียงหน้าที่ในการเก็บข้อมูลภาพต้นแบบสำหรับการประมวลผลและข้อมูลภาพที่ผ่านการประมวลผลแล้วเท่านั้น

ในการออกแบบระบบทั้งหมดได้กำหนดให้ตัววงจรมีขนาดไม่ใหญ่จนเกินไปและประหยัดหน่วยความจำที่จะใช้ จึงได้ออกแบบให้ทุก ๆ วงจรในการประมวลผลภาพใช้หน่วยความจำร่วมกัน ดังนั้นในการออกคำสั่งให้มีการประมวลผลภาพจากโมดูลใด ๆ จึงต้องเป็นไปตามลำดับ หากต้องการให้มีการประมวลผลภาพในหลาย ๆ ลักษณะจำเป็นต้องรอให้โมดูลที่กำลังประมวลผลอยู่ได้ทำหน้าที่เสร็จสิ้นเสียก่อนจึงจะสามารถส่งผ่านการควบคุมไปให้กับโมดูลอื่นถัดไปได้ซึ่งการประมวลผลในลักษณะนี้เป็นการประมวลผลแบบตามลำดับ(Sequential Processing) หากต้องการความรวดเร็วมากขึ้นหรือต้องการให้มีการประมวลผลในลักษณะแบบขนาน(Parallel Processing) การออกแบบระบบในลักษณะนี้ก็สามารถทำได้โดยเพิ่มหน่วยความจำรวมเข้าไปภายในวงจรรวมสำหรับการประมวลผลภาพแต่ละโมดูลเลย โดยวิธีนี้แต่ละโมดูลก็สามารถทำงานในการประมวลผลภาพไปได้พร้อม ๆ กันและเป็นอิสระต่อกัน แต่การออกแบบระบบในลักษณะนี้จะทำให้ระบบมีขนาดใหญ่และต้องใช้หน่วยความจำเป็นจำนวนมาก ต้นทุนในการผลิตก็จะสูงตามไปด้วย ดังนั้นในงานวิจัยนี้จึงได้เลือกใช้วิธีการประมวลผลแบบตามลำดับเพื่อให้วงจรมีขนาดเล็กและประหยัดมากกว่า

อย่างไรก็ตามวงจรทั้งหมดที่ได้ออกแบบไว้ในงานวิทยานิพนธ์ฉบับนี้ได้ทำการออกแบบไปจนถึงระดับเกทซึ่งเป็นระดับโครงสร้างของวงจรที่สามารถนำไปใช้ในการทำงานได้จริง ในการออกแบบวงจรในระดับโครงสร้างโดยทั่วไปแล้วรูปแบบที่มักใช้ปฏิบัติกันจะเป็นรูปแบบที่เรียกว่า *Schematic Capture* ซึ่งเป็นการออกแบบในลักษณะที่เป็นการนำเอาสัญลักษณ์ทางวงจรอันได้แก่ เกทและฟลิปฟล็อปต่าง ๆ ที่มีอยู่แล้วในไลบรารีมาวางเรียงประกอบกันและเชื่อมต่อกันด้วยเส้นสัญญาณ ซึ่งเปรียบได้กับการบรรยายในระดับโครงสร้างของ VHDL ที่มีการบรรยายองค์ประกอบและการเชื่อมต่อในลักษณะเดียวกัน ในรูปต่อไปนี้จะเป็นตัวอย่างของวงจรส่วนหนึ่งที่ได้ออกแบบไว้ในงานวิทยานิพนธ์ฉบับนี้และมีการบรรยายลักษณะของวงจรโดยภาษา VHDL เปรียบเทียบให้เห็นในลักษณะของ *Schematic Capture*

```
ENTITY full_adder IS
    PORT ( a, b, cin : IN qit;
           sum, carry : OUT qit);
END full_adder;
ARCHITECTURE gate_level OF full_adder IS
    SIGNAL im1,im2,im3,im4,im5,im6,im7,im8,im9,im10 : qit;
BEGIN
    g0 : inv PORT MAP (a, im1);
    g1 : inv PORT MAP (b, im2);
    g2 : inv PORT MAP (cin, im3);
```

เอกสารนี้เป็นเอกสารลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

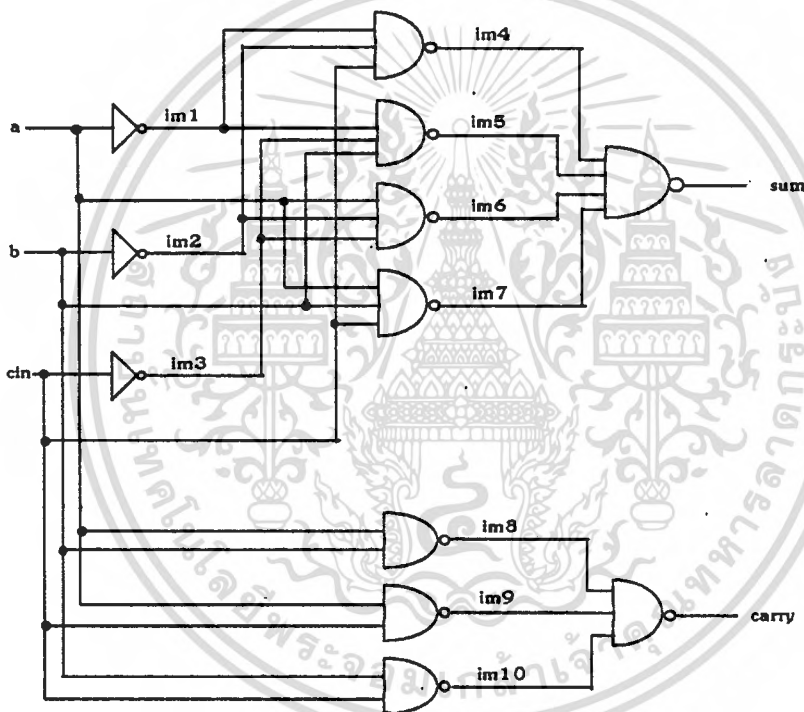
-- sum output
g3 : n3 PORT MAP (im1, im2, cin, im4);
g4 : n3 PORT MAP (im1, b, im3, im5);
g5 : n3 PORT MAP (a, im2, im3, im6);
g6 : n3 PORT MAP (a, b, cin, im7);
g7 : n4 PORT MAP (im4, im5, im6, im7, sum);

-- carry output
g8 : n2 PORT MAP (a, b, im8);
g9 : n2 PORT MAP (a, cin, im9);
g10 : n2 PORT MAP (b, cin, im10);
g11 : n3 PORT MAP (im8, im9, im10, carry);

END gate_level;

```

รูปแสดงการบรรยายของวงจร full_ader



รูปแสดง Schematic ของวงจร full_ader

หากต้องการนำวงจรทั้งหมดที่ได้ออกแบบไว้นี้ไปผลิตเป็นชิปไอซีจะต้องนำไปผ่านกระบวนการแปลงข้อมูลของไฟล์วงจรทั้งหมดให้อยู่ในรูปแบบของไฟล์มาตรฐานหรือ Netlist File เสียก่อน ซึ่งในระบบของ Mentor Graphics System-1076 นี้ก็สามารถแปลงไฟล์ข้อมูลที่อยู่ในรูปของ VHDL ให้เป็น Netlist File ได้โดยใช้ซอฟต์แวร์ที่ชื่อ Enwrite เมื่อได้ Netlist File ของวงจรทั้งหมดแล้วก็สามารถส่งไปให้โรงงานผู้ผลิตได้ทันที แต่ถ้าต้องการออกแบบวงจรให้ต่ำลงไปกว่าระดับเกทหรือลงไปถึงระดับ layout ก่อนที่จะส่งไปให้โรงงานผู้ผลิตก็สามารถทำได้เช่นกันโดยใช้ซอฟต์แวร์ที่ชื่อ IC Station ซึ่งเป็นส่วนหนึ่งในระบบของ Mentor Graphics System-1076 ด้วยเช่นกัน

ผลการทดสอบการทำงานของวงจรทั้งหมดกระทำโดยใช้วิธีการเลียนแบบการทำงานของวงจร(simulation)บนเครื่อง HP-Workstation ด้วยซอฟต์แวร์ Mentor Graphics System-1076 ดังที่ได้แสดงไว้ในบทที่ 8 อย่างไรก็ตามการทดสอบจะได้ผลเป็นที่น่าสนใจก็ตาม หากแต่ก็มีได้ทำการทดสอบด้วยวงจรทางฮาร์ดแวร์จริง การทดสอบความถูกต้องในการทำงานของวงจรจึงหยุดอยู่เพียงขั้นตอนในการเลียนแบบบนเครื่องคอมพิวเตอร์เท่านั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้


```

FUNCTION now RETURN time;
SUBTYPE natural IS integer RANGE 0 TO integer'HIGH;
SUBTYPE positive IS integer RANGE 1 TO integer'HIGH;
TYPE string IS ARRAY (positive RANGE <>) OF character;
TYPE bit_vector IS ARRAY (natural RANGE <>) OF bit;
END standard;

```

A.2 THE TEXTIO PACKAGE

เป็นแพ็คเกจที่กำหนดชนิด โปรซีเจอร์ และฟังก์ชันสำหรับ **standard text I/O** จากไฟล์ **ASCII** และอยู่ในไลบรารี **STD**

```

-- Package TEXTIO as defined in Chapter 14, Section 3 of the IEEE Standard
-- VHDL Language Reference Manual (IEEE Std. 1076-1987)
--
PACKAGE textio IS
    TYPE line IS ACCESS string;
    TYPE text IS FILE OF string;
    TYPE side IS (right, left);
    SUBTYPE width IS natural;

    FILE input : text IS IN "std_input";
    FILE output : text IS OUT "std_output";

    PROCEDURE readline (VARIABLE f : IN text; l : INOUT line);

    PROCEDURE read (l : INOUT line; value : OUT bit; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT bit);

    PROCEDURE read (l : INOUT line; value : OUT bit_vector; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT bit_vector);

    PROCEDURE read (l : INOUT line; value : OUT boolean; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT boolean);

    PROCEDURE read (l : INOUT line; value : OUT character; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT character);

    PROCEDURE read (l : INOUT line; value : OUT integer; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT integer);

    PROCEDURE read (l : INOUT line; value : OUT real; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT real);

    PROCEDURE read (l : INOUT line; value : OUT string; good : OUT boolean);
    PROCEDURE read (l : INOUT line; value : OUT string);

```

```
PROCEDURE read (l : INOUT line; value : OUT time);
```

```
PROCEDURE writeline (f : OUT text; l : INOUT line);
```

```
PROCEDURE write (l : INOUT line; value : IN bit;
                 justified : IN side := right; field : IN width := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN bit_vector;
                 justified : IN side := right; field : IN width := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN boolean;
                 justified : IN side := right; field : IN width := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN character;
                 justified : IN side := right; field : IN width := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN integer;
                 justified : IN side := right; field : IN width := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN real; justified : IN side := right;
                 field : IN width := 0; digits : IN natural := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN string;
                 justified : IN side := right; field : IN width := 0);
```

```
PROCEDURE write (l : INOUT line; value : IN time; justified : IN side := right;
                 field : IN width := 0; unit : IN time := ns);
```

```
FUNCTION endfile (VARIABLE f : IN text) RETURN boolean;
END textio;
```

A.3 THE BASIC UTILITIES PACKAGE

```
PACKAGE basic_utilities IS
```

```
  TYPE qit IS ('0', '1', 'Z', 'X');
```

```
  TYPE qit_2d IS ARRAY (qit, qit) OF qit;
```

```
  TYPE qit_1d IS ARRAY (qit) OF qit;
```

```
  TYPE qit_vector IS ARRAY (NATURAL RANGE <>) OF qit;
```

```
  PROCEDURE qit2int (qin : IN qit_vector; int : OUT INTEGER);
```

```
  PROCEDURE int2qit (int : IN INTEGER; qin : OUT qit_vector);
```

```
  FUNCTION qit2bit_vector (qin : qit_vector) RETURN BIT_VECTOR;
```

```
  FUNCTION bit2qit_vector (bin : BIT_VECTOR) RETURN qit_vector;
```

```
  FUNCTION inc (x : qit_vector) RETURN qit_vector;
```

```
  FUNCTION "AND" (a, b : qit) RETURN qit;
```

```
  FUNCTION "OR" (a, b : qit) RETURN qit;
```

```
  FUNCTION "NOT" (a : qit) RETURN qit;
```

```
  FUNCTION wire (a, b : qit) RETURN qit;
```

```

FUNCTION oring ( drivers : qit_vector) RETURN qit;
SUBTYPE ored_qit IS oring qit;
TYPE ored_qit_vector IS ARRAY (NATURAL RANGE <>) OF ored_qit;

FUNCTION anding ( drivers : qit_vector) RETURN qit;
SUBTYPE anded_qit IS anding qit;
TYPE anded_qit_vector IS ARRAY (NATURAL RANGE <>) OF anded_qit;

FUNCTION wiring ( drivers : qit_vector) RETURN qit;
SUBTYPE wired_qit IS wiring qit;
TYPE wired_qit_vector IS ARRAY (NATURAL RANGE <>) OF wired_qit;
END basic_utilities;

```

```

PACKAGE BODY basic_utilities IS

```

```

  FUNCTION "AND" (a, b : qit) RETURN qit IS

```

```

    CONSTANT qit_and_table : qit_2d := (
      ('0','0','0','0'),
      ('0','1','1','X'),
      ('0','1','1','X'),
      ('0','X','X','X'));

```

```

  BEGIN

```

```

    RETURN qit_and_table (a, b);

```

```

  END "AND";

```

```

  FUNCTION "OR" (a, b : qit) RETURN qit IS

```

```

    CONSTANT qit_or_table : qit_2d := (
      ('0','1','1','X'),
      ('1','1','1','1'),
      ('1','1','1','1'),
      ('X','1','1','X'));

```

```

  BEGIN

```

```

    RETURN qit_or_table (a, b);

```

```

  END "OR";

```

```

  FUNCTION "NOT" (a : qit) RETURN qit IS

```

```

    CONSTANT qit_not_table : qit_1d := ('1','0','0','X');

```

```

  BEGIN

```

```

    RETURN qit_not_table (a);

```

```

  END "NOT";

```

```

  FUNCTION wire (a, b : qit) RETURN qit IS

```

```

    CONSTANT qit_wire_table : qit_2d := (
      ('0','X','0','X'),
      ('X','1','1','X'),
      ('0','1','Z','X'),
      ('X','X','X','X'));

```

```

  BEGIN

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

RETURN qit_wire_table (a, b);
END wire;

```

```

PROCEDURE qit2int (qin : IN qit_vector; int : OUT INTEGER) IS
  VARIABLE result : INTEGER;
BEGIN
  result := 0;
  FOR i IN 0 TO (qin'LENGTH - 1) LOOP
    IF qin(i) = '1' THEN
      result := result + 2**i;
    END IF;
  END LOOP;
  int := result;
END qit2int;

```

```

PROCEDURE int2qit (int : IN INTEGER; qin : OUT qit_vector) IS
  VARIABLE tmp : INTEGER;
BEGIN
  tmp := int;
  FOR i IN 0 TO (qin'LENGTH - 1) LOOP
    IF (tmp MOD 2 = 1) THEN
      qin(i) := '1';
    ELSE qin(i) := '0';
    END IF;
    tmp := tmp / 2;
  END LOOP;
END int2qit;

```

```

FUNCTION qit2bit_vector (qin : qit_vector) RETURN BIT_VECTOR IS
  VARIABLE b : BIT_VECTOR (qin'RANGE);
BEGIN
  FOR i IN qin'RANGE LOOP
    IF qin(i) = '1' THEN
      b(i) := '1';
    ELSE
      b(i) := '0';
    END IF;
  END LOOP;
  RETURN b;
END qit2bit_vector;

```

```

FUNCTION bit2qit_vector (bin : BIT_VECTOR) RETURN qit_vector IS
  VARIABLE q : qit_vector (bin'RANGE);
BEGIN
  FOR i IN bin'RANGE LOOP
    IF bin(i) = '1' THEN
      q(i) := '1';
    ELSE
      q(i) := '0';
    END IF;
  END LOOP;
  RETURN q;
END bit2qit_vector;

```

```

ELSE
    q(i) := '0';
END IF;
END LOOP;
RETURN q;
END bit2qit_vector;

```

```

FUNCTION inc (x : qit_vector) RETURN qit_vector IS
    VARIABLE i : INTEGER;
    VARIABLE t : qit_vector(x'RANGE);
BEGIN
    qit2int (x, i);
    i := i + 1;
    IF i >= 2**x'LENGTH THEN
        i := 0;
    END IF;
    int2qit (i, t);
    RETURN t;
END inc;

```

```

FUNCTION oring (drivers : qit_vector) RETURN qit IS
    VARIABLE accumulate : qit := '0';
BEGIN
    FOR i IN drivers'RANGE LOOP
        accumulate := accumulate OR drivers(i);
    END LOOP;
    RETURN accumulate;
END oring;

```

```

FUNCTION anding (drivers : qit_vector) RETURN qit IS
    VARIABLE accumulate : qit := '1';
BEGIN
    FOR i IN drivers'RANGE LOOP
        accumulate := accumulate AND drivers(i);
    END LOOP;
    RETURN accumulate;
END anding;

```

```

FUNCTION wiring (drivers : qit_vector) RETURN qit IS
    VARIABLE accumulate : qit := 'Z';
BEGIN
    FOR i IN drivers'RANGE LOOP
        accumulate := wire(accumulate, drivers(i));
    END LOOP;
    RETURN accumulate;
END wiring;

```

END basic_utilities;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

A.4 THE IMAGE UTILITIES PACKAGE

```
LIBRARY util;
USE util.basic_utilities.ALL;
USE STD.ALL;
```

```
PACKAGE image_utilities IS
  SUBTYPE nibble IS qit_vector (3 DOWNT0 0);
  SUBTYPE byte IS qit_vector (7 DOWNT0 0);
  SUBTYPE word IS qit_vector (15 DOWNT0 0);
  TYPE byte256 IS ARRAY (0 TO 255) OF byte;
  TYPE word256 IS ARRAY (0 TO 255) OF word;
  SUBTYPE wired_nibble IS wired_qit_vector (3 DOWNT0 0);
  SUBTYPE wired_byte IS wired_qit_vector (7 DOWNT0 0);
  SUBTYPE wired_ten IS wired_qit_vector (9 DOWNT0 0);
  SUBTYPE wired_word IS wired_qit_vector (15 DOWNT0 0);
  SUBTYPE ored_nibble IS ored_qit_vector (3 DOWNT0 0);
  SUBTYPE ored_byte IS ored_qit_vector (7 DOWNT0 0);
  SUBTYPE ored_word IS ored_qit_vector (15 DOWNT0 0);
  CONSTANT zero_4 : nibble := "0000";
  CONSTANT zero_8 : byte := "00000000";
  CONSTANT zero_16 : word := "0000000000000000";
  FUNCTION "AND" (a, b : qit_vector) RETURN qit_vector;
  FUNCTION "OR" (a, b : qit_vector) RETURN qit_vector;
  FUNCTION "NOT" (a : qit_vector) RETURN qit_vector;
  PROCEDURE qit2char (qin: IN qit; char: OUT CHARACTER);
  PROCEDURE qit_vector2str (qin: IN qit_vector; str: OUT STRING);
  PROCEDURE wqit_vector2str (qin: IN wired_qit_vector; str: OUT STRING);
END image_utilities;
```

```
PACKAGE BODY image_utilities IS
  FUNCTION "AND" (a, b : qit_vector) RETURN qit_vector IS
    VARIABLE r : qit_vector (a'RANGE);
  BEGIN
    loop1: FOR i IN a'RANGE LOOP
      r(i) := a(i) AND b(i);
    END LOOP loop1;
    RETURN r;
  END "AND";

  FUNCTION "OR" (a, b : qit_vector) RETURN qit_vector IS
    VARIABLE r : qit_vector (a'RANGE);
  BEGIN
    loop1: FOR i IN a'RANGE LOOP
      r(i) := a(i) OR b(i);
    END LOOP loop1;
    RETURN r;
  END "OR";
```

```

FUNCTION "NOT" (a : qit_vector) RETURN qit_vector IS
  VARIABLE r : qit_vector (a'RANGE);
BEGIN
  loop1: FOR i IN a'RANGE LOOP
    r(i) := NOT a(i);
  END LOOP loop1;
  RETURN r;
END "NOT";

```

```

PROCEDURE qit2char (qin: IN qit; char: OUT CHARACTER) IS
  VARIABLE ch : CHARACTER;
BEGIN
  IF qin = '0' THEN ch := '0';
  ELSIF qin = '1' THEN ch := '1';
  ELSIF qin = 'Z' THEN ch := 'Z';
  ELSE ch := 'X';
  END IF;
  char := ch;
END qit2char;

```

```

PROCEDURE qit_vector2str (qin: IN qit_vector; str: OUT STRING) IS
  VARIABLE st : STRING (qin'RANGE);
BEGIN
  FOR i IN qin'RANGE LOOP
    IF qin(i) = '0' THEN st(i) := '0';
    ELSIF qin(i) = '1' THEN st(i) := '1';
    ELSIF qin(i) = 'Z' THEN st(i) := 'Z';
    ELSE st(i) := 'X';
    END IF;
  END LOOP;
  str := st;
END qit_vector2str;

```

```

PROCEDURE wqit_vector2str (qin: IN wired_qit_vector; str: OUT STRING) IS
  VARIABLE st : STRING (qin'RANGE);
BEGIN
  FOR i IN qin'RANGE LOOP
    IF qin(i) = '0' THEN st(i) := '0';
    ELSIF qin(i) = '1' THEN st(i) := '1';
    ELSIF qin(i) = 'Z' THEN st(i) := 'Z';
    ELSE st(i) := 'X';
    END IF;
  END LOOP;
  str := st;
END wqit_vector2str;
END image_utilities;

```

A.5 THE IMAGE PARAMETER PACKAGE

```

LIBRARY util;
USE util.basic_utilities.ALL;

PACKAGE image_parameters IS
  CONSTANT single_byte_instructions : qit_vector (3 DOWNTO 0) := "0000";
  CONSTANT clm : qit_vector (3 DOWNTO 0) := "0001";
  CONSTANT swm : qit_vector (3 DOWNTO 0) := "0010";
  CONSTANT mod1 : qit_vector (3 DOWNTO 0) := "1000";
  CONSTANT mod2 : qit_vector (3 DOWNTO 0) := "1001";
  CONSTANT mod3 : qit_vector (3 DOWNTO 0) := "1010";
  CONSTANT mod4 : qit_vector (3 DOWNTO 0) := "1011";
  CONSTANT mod5 : qit_vector (3 DOWNTO 0) := "1100";
  CONSTANT mod6 : qit_vector (3 DOWNTO 0) := "1101";
  CONSTANT mod7 : qit_vector (3 DOWNTO 0) := "1110";
  CONSTANT mod8 : qit_vector (3 DOWNTO 0) := "1111";
  CONSTANT row : qit_vector (3 DOWNTO 0) := "0011";
  CONSTANT col : qit_vector (3 DOWNTO 0) := "1100";
END image_parameters;

```

A.6 THE GATES FLIPFLOP PACKAGE

```

LIBRARY util;
USE util.basic_utilities.ALL;

PACKAGE gates_flipflop IS
  COMPONENT df -- dff
    PORT ( d, en, clk, clr : IN qit; q : OUT qit);
  END COMPONENT;
  COMPONENT dl -- d_latch
    PORT ( d, en, clk : IN qit; q : OUT qit);
  END COMPONENT;
  COMPONENT jkff -- jk_ff
    PORT ( j, k, clk :IN qit; q : OUT qit);
  END COMPONENT;
  COMPONENT jkcff -- jk_clr_ff
    PORT ( j, k, clk, clr :IN qit; q : OUT qit);
  END COMPONENT;
  COMPONENT msdff -- master_slave_dff
    PORT ( d, clk :IN qit; q : OUT qit);
  END COMPONENT;
  COMPONENT inv -- inv_q
    PORT ( i1 : IN qit; o1 : OUT qit);
  END COMPONENT;
  COMPONENT a2 -- and2_q
    PORT ( i1, i2 : IN qit; o1 : OUT qit);
  END COMPONENT;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

COMPONENT a3 -- and3_q
  PORT ( i1, i2, i3 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT a4 -- and4_q
  PORT ( i1, i2, i3, i4 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT o2 -- or2_q
  PORT ( i1, i2 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT o3 -- or3_q
  PORT ( i1, i2, i3 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT o4 -- or4_q
  PORT ( i1, i2, i3, i4 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT n2 -- nand2_q
  PORT ( i1, i2 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT n3 -- nand3_q
  PORT ( i1, i2, i3 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT n4 -- nand4_q
  PORT ( i1, i2, i3, i4 : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT ts -- tri_state_q
  PORT ( i1, c : IN qit; o1 : OUT qit);
END COMPONENT;
COMPONENT delay -- signal_delay
  PORT (i1 : IN qit; o1 : OUT qit);
END COMPONENT;
END gates_flipflop;

```

A.7 THE REGISTER LOGIC PACKAGE

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

```

```

PACKAGE register_logic IS
  COMPONENT reg -- register_unit
    PORT ( i8 : IN byte; o8 : OUT byte; load, ck : IN qit);
  END COMPONENT;
  COMPONENT rg8 -- register_unit_8
    PORT ( d : IN qit_vector(7 DOWNT0 0);
          q : OUT qit_vector(7 DOWNT0 0);
          load, ck : IN qit);
  END COMPONENT;

```

เอกสารนี้เป็นเอกสารการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

COMPONENT pc -- program_counter_unit
  PORT ( o16 : OUT word; increment, reset, ck : IN qit);
END COMPONENT;

COMPONENT rpc -- row_col_counter_unit
  PORT ( o8 : OUT byte; increment, reset, ck : IN qit);
END COMPONENT;

COMPONENT mar -- memory_address_register_unit
  PORT ( i16 : IN word; o16 : OUT word;
        load_page, load_offset, ck : IN qit);
END COMPONENT;

COMPONENT cpc -- compare_col
  PORT ( i1, i2 : IN byte; o1 : OUT qit);
END COMPONENT;

COMPONENT cprc -- compare_row_col
  PORT ( i1, i2 : IN word; o1 : OUT qit);
END COMPONENT;

COMPONENT ts8 -- tri_state_8
  PORT ( i8 : IN byte; c : IN qit; o8 : OUT byte);
END COMPONENT;

COMPONENT ts10 -- tri_state_10
  PORT ( i : IN qit_vector(9 DOWNT0 0); c : IN qit; o : OUT qit_vector(9 DOWNT0 0));
END COMPONENT;

COMPONENT ts16 -- tri_state_16
  PORT ( i16 : IN word; c : IN qit; o16 : OUT word);
END COMPONENT;

COMPONENT inv10 -- 10_bit_inv_q
  PORT ( i : IN qit_vector(9 DOWNT0 0); o : OUT qit_vector(9 DOWNT0 0));
END COMPONENT;

COMPONENT add -- adder_8
  PORT ( i1, i2 : IN byte; o1 : OUT byte);
END COMPONENT;

COMPONENT add8 -- 8_bit_adder
  PORT ( a, b : IN qit_vector(7 DOWNT0 0);
        cin : IN qit;
        sum : OUT qit_vector(7 DOWNT0 0);
        cout : OUT qit);
END COMPONENT;

COMPONENT add9 -- 9_bit_adder
  PORT ( a, b : IN qit_vector(8 DOWNT0 0);
        cin : IN qit;
        sum : OUT qit_vector(8 DOWNT0 0);
        cout : OUT qit);
END COMPONENT;

COMPONENT add10 -- 10_bit_adder
  PORT ( a, b : IN qit_vector(9 DOWNT0 0);
        cin : IN qit;
        sum : OUT qit_vector(9 DOWNT0 0);

```

```

        cout : OUT qit);
END COMPONENT;
COMPONENT d8_256 -- decoder_8_to_256
    PORT ( i8 : IN byte; en : IN qit;
          o : OUT qit_vector (0 TO 255));
END COMPONENT;
END register_logic;

```



B. การบรรยายของวงจรรวม (The Description of Design)

ภาคผนวก B รวบรวมการบรรยายการทำงานของระบบและโมดูลทั้งหมด ทั้งในระดับพฤติกรรม ระดับข้อมูล และระดับโครงสร้าง

B.1 การบรรยายของระบบ (The Description of Modular System)

```
LIBRARY util;
```

```
USE util.basic.utilities.ALL;
```

```
USE util.image.utilities.ALL;
```

```
USE STD.TEXTIO.ALL;
```

```
ENTITY modular IS
```

```
  GENERIC ( read_high_time, read_low_time,
            write_high_time, write_low_time : TIME := 2 US);
```

```
  PORT ( clk : IN qit;
```

```
        load_image, write_image : IN qit;
```

```
        interrupt : IN qit;  -- Signal to start controller module
```

```
        instruct : IN byte;  -- instruction
```

```
        load_ready : OUT qit;
```

```
        read_inst : OUT qit); -- Signal to fetch next instruction
```

```
END modular;
```

```
ARCHITECTURE structural OF modular IS
```

```
  COMPONENT memo
```

```
    PORT ( clk, read, write,
```

```
          clear, enable : IN qit;
```

```
          abus : IN word;
```

```
          dbus : INOUT wired_byte BUS);
```

```
  END COMPONENT;
```

```
  COMPONENT ctrl
```

```
    PORT ( clk, interrupt, mod_end : IN qit;
```

```
          instruct : IN byte;
```

```
          clear_mem, read_mem, write_mem : OUT qit;
```

```
          dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
```

```
          abus : OUT word;
```

```
          read_instruction : OUT qit;
```

```
          en_sm, en_dm : OUT qit;
```

```
          m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
```

```
          r_c : OUT qit;
```

```
          r_c_data : OUT word;
```

```
          mod_begin : OUT qit;
```

```
          dbus_on_databus, databus_on_dbus : OUT qit);
```

```
  END COMPONENT;
```

```
  COMPONENT module
```

```
    PORT ( clk, interrupt, r_c, mod_begin : IN qit;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    r_c_data : IN word;
    dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
    read_mem, write_mem : OUT qit;
    en_sm, en_dm, mod_end : OUT qit;
    abus : OUT word;
    dbus_on_databus, databus_on_dbus : OUT qit);
END COMPONENT;

SIGNAL mod_end : qit;
SIGNAL clear, sread, swrite : qit;
SIGNAL data : wired_byte BUS := "ZZZZZZZZ";
SIGNAL address : word;
SIGNAL read_inst : qit;
SIGNAL en_sm, en_dm : qit;
SIGNAL m1, m2, m3, m4, m5, m6, m7, m8 : qit;
SIGNAL r_c : qit;
SIGNAL r_c_data : word;
SIGNAL mod_begin : qit;
SIGNAL dbus_on_databus, databus_on_dbus : qit;
SIGNAL load_image, write_image, load_ready : qit;
BEGIN
    source_memory : memo
        PORT MAP ( clk, sread, swrite, clear, en_sm, address, data);
    dest_memory : memo
        PORT MAP ( clk, sread, swrite, clear, en_dm, address, data);
    ctrl_module : ctrl
        PORT MAP ( clk, interrupt, mod_end, instruct,
            clear, sread, swrite, data, address,
            read_inst, en_sm, en_dm,
            m1, m2, m3, m4, m5, m6, m7, m8,
            r_c, r_c_data, mod_begin,
            dbus_on_databus, databus_on_dbus);
    histogram_module : module
        PORT MAP ( clk, interrupt, r_c, mod_begin,
            r_c_data, data, sread, swrite,
            en_sm, en_dm, mod_end, address,
            dbus_on_databus, databus_on_dbus);
    edge_module : module
        PORT MAP ( clk, interrupt, r_c, mod_begin,
            r_c_data, data, sread, swrite,
            en_sm, en_dm, mod_end, address,
            dbus_on_databus, databus_on_dbus);

```

-- read image data from file and write to memory

read : PROCESS (load_image)

FILE source : TEXT IS IN "input.dat";

VARIABLE sl : LINE;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

VARIABLE page, offset : byte;
VARIABLE dt : INTEGER;
VARIABLE ram_init : BOOLEAN := FALSE;
VARIABLE i, j : INTEGER := 0;
BEGIN
  IF ram_init = FALSE THEN
    WHILE NOT ENDFILE (source) AND (i < 256) LOOP
      READLINE (source, sl);
      WHILE (j < 256) LOOP
        READ (sl, dt);
        page <= int2qit(i);
        offset <= int2qit(j);
        en_sm <= '1';
        en_dm <= '1';
        address <= page & offset;
        data <= int2qit(dt);
        swrite <= '1'; WAIT FOR write_high_time;
        swrite <= '0'; WAIT FOR write_low_time;
        data <= "ZZZZZZZZ";
        j := j + 1;
      END LOOP;
      j := 0;
      i := i + 1;
    END LOOP;
    ram_init := TRUE;
  END IF;
  load_ready <= '1';
END PROCESS read;

```

-- read image data from memory and write to file

write : PROCESS

```

FILE dest : TEXT IS OUT "output.dat";
VARIABLE dl : LINE;
VARIABLE page, offset, dt : byte;
VARIABLE dd : INTEGER;
VARIABLE i, j : INTEGER := 0;

```

BEGIN

```

  WAIT UNTIL write_image = '1';
  i := 0;
  j := 0;
  WHILE i < 256 LOOP
    WHILE j < 256 LOOP
      page <= int2qit(i);
      offset <= int2qit(j);
      en_sm <= '0';
      en_dm <= '1';
      address <= page & offset;

```

```

        sread <= '1'; WAIT FOR read_high_time;
        dt <= byte(data);
        sread <= '0'; WAIT FOR read_low_time;
        dd <= qit2int(dt);
        WRITE (dl, dd);
        j := j + 1;
    END LOOP;
    WRITELINE (dest, dl);
    j := 0;
    i := i + 1;
    END LOOP;
END PROCESS write;
END structural;

```

B.2 การบรรยายของหน่วยความจำ (The Description of Memory)

B.2.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY memory IS
    PORT ( clk, read, write, clear, enable : IN qit;
          abus : IN word;
          dbus : INOUT wired_byte BUS);
END memory;

ARCHITECTURE behavioral OF memory IS
    TYPE byte_memory IS ARRAY (0 TO 255, 0 TO 255) OF byte;
BEGIN
    PROCESS
        VARIABLE mem : byte_memory;
        VARIABLE ia, ib : INTEGER;
        VARIABLE page, offset : byte;
    BEGIN
        WAIT ON read, write, clear;
        page := abus (15 DOWNT0 8);
        offset := abus (7 DOWNT0 0);
        qit2int (page, ia);
        qit2int (offset, ib);
        IF (enable = '1') THEN
            IF read = '1' THEN
                IF (ia >= 256) OR (ib >= 256) THEN
                    dbus <= "ZZZZZZZZ";
                ELSE
                    dbus <= wired_byte (mem(ia, ib));
                END IF;
            ELSE
                WAIT UNTIL read = '0';
            END IF;
        END IF;
    END PROCESS;

```

เอกสารนี้เป็นเอกสารสงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

        dbus <= "ZZZZZZZZ";
    ELSIF write = '1' THEN
        IF (ia < 256) AND (ib < 256) THEN
            mem(ia, ib) := byte(dbus);
        END IF;
        WAIT UNTIL write = '0';
    ELSIF clear = '1' THEN
        FOR i IN 0 TO 255 LOOP
            FOR j IN 0 TO 255 LOOP
                mem(i,j) := "00000000";
            END LOOP;
        END LOOP;
    END IF;
ELSE
    dbus <= "ZZZZZZZZ";
END IF;
END PROCESS;
END behavioral;

```

B.2.2 การบรรยายในระดับโครงสร้าง (The Structural Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY gates;
USE gates.gates_flipflop.ALL;
LIBRARY memo;
ENTITY memory IS
    PORT ( clk, read, write, clear, enable : IN qit;
           abus : IN word;
           dbus : INOUT wired_byte BUS);
END memory;
ARCHITECTURE structural OF memory IS
    COMPONENT s1 PORT (addr : IN byte; en : IN qit;
                        sel : OUT qit_vector(0 TO 255));
    END COMPONENT;
    COMPONENT ma PORT (clk, dr : IN qit;
                        r, c : IN qit_vector(0 TO 255);
                        dw : IN byte; dr : OUT byte);
    END COMPONENT;
    FOR ALL : s1 USE ENTITY memo.select_memory(dataflow);
    FOR ALL : ma USE ENTITY memo.memory_array(structural);
    FOR ALL : inv USE ENTITY gates.inv_q(double_delay);
    FOR ALL : a2 USE ENTITY gates.and2_q(double_delay);
    FOR ALL : a3 USE ENTITY gates.and3_q(average_delay);
    FOR ALL : ts USE ENTITY gates.tri_state_q(single_delay);
    SIGNAL rs, cs : qit_vector(0 TO 255);
    SIGNAL in1, read_en, write_en : qit;

```

```

SIGNAL di, do : byte;
SIGNAL page; offset : byte;
BEGIN
    page <= abus(15 DOWNT0 8);
    offset <= abus(7 DOWNT0 0);
    c0 : s1 PORT MAP (page, enable, rs);
    c1 : s1 PORT MAP (offset, enable, cs);
    c2 : inv PORT MAP (write, in1);
    c3 : a2 PORT MAP (enable, write, write_en);
    c4 : a3 PORT MAP (enable, read, in1, read_en);
    c5to12 : FOR i IN 0 TO 7 GENERATE
        cw : ts PORT MAP (dbus(i), write_en, do(i));
    END GENERATE;
    c13to20 : FOR j IN 0 TO 7 GENERATE
        cr : ts PORT MAP (di(j), read_en, dbus(j));
    END GENERATE;
    c21 : ma PORT MAP (clk, clear, rs, cs, do, di);
END structural;

```

B.3 การบรรยายของวงจรควบคุมหลัก (The Description of Controller)

B.3.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;

ENTITY controller IS
    GENERIC ( read_high_time, read_low_time,
              write_high_time, write_low_time : TIME := 2 US;
              cycle_time : TIME := 4 US);
    PORT ( clk, interrupt, mod_end : IN qit;
           instruct : IN byte;
           clear_mem, read_mem, write_mem : OUT qit;
           dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
           abus : OUT word;
           read_instruction : OUT qit;
           en_sm, en_dm : OUT qit;
           m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
           r_c : OUT qit;
           r_c_data : OUT word;
           mod_begin : OUT qit;
           dbus_on_databus, databus_on_dbus : OUT qit);
END controller;

```

```

ARCHITECTURE behavioral QF controller IS
BEGIN

```

```

    PROCESS

```

```

        VARIABLE page, offset : byte;

```

```

VARIABLE byte1, byte2 : byte;
VARIABLE no_row, no_col : byte;
VARIABLE buff : byte;
BEGIN
IF interrupt = '1' THEN
    page := zero_8;
    offset := zero_8;
    no_row := zero_8;
    no_col := zero_8;
    WAIT FOR cycle_time;
ELSIF (interrupt /= '1' OR mod_end = '1') THEN
    -- read first byte
    read_instruction <= '1'; WAIT FOR read_high_time;
    byte1 := instruct;
    read_instruction <= '0'; WAIT FOR read_low_time;
    -- execute single byte instruction
    IF byte1 (7 DOWNT0 4) = single_byte_instructions THEN
        CASE byte1 (3 DOWNT0 0) IS
            WHEN clm => -- clear both memory
                ASSERT
                    (NOT (byte1 (3 DOWNT0 0) = clm))
                REPORT
                    "Being clear memory"
                SEVERITY NOTE;
                en_sm <= '1';
                en_dm <= '1';
                clear_mem <= '1'; WAIT FOR cycle_time;
                clear_mem <= '0';
                en_sm <= '0';
                en_dm <= '0';
            WHEN swm => -- swap image data from destination to source memory
                ASSERT
                    (NOT (byte1 (3 DOWNT0 0) = swm))
                REPORT
                    "Being swap data in memory"
                SEVERITY NOTE;
                page := zero_8;
                offset := zero_8;
                WHILE page <= no_row LOOP
                    WHILE offset <= no_col LOOP
                        en_sm <= '0';
                        en_dm <= '1';
                        abus <= page & offset;
                        read_mem <= '1'; WAIT FOR read_high_time;
                        databus_on_dbus <= '1';
                        buff := byte(dbus);
                        read_mem <= '0'; WAIT FOR read_low_time;

```

```

databus_on_dbus <= '0';
en_sm <= '1';
en_dm <= '0';
dbus <= wired_byte (buff);
dbus_on_databus <= '1';
write_mem <= '1'; WAIT FOR write_high_time;
write_mem <= '0'; WAIT FOR write_low_time;
dbus_on_databus <= '0';
dbus <= "ZZZZZZZZ";
offset := inc(offset);
END LOOP;
offset := zero_8;
page := inc(page);
END LOOP;
en_sm <= '0';
WHEN mod1 => -- select module 1
m1 <= '1'; WAIT FOR cycle_time;
r_c <= '1';
r_c_data <= no_row & no_col; WAIT FOR read_high_time;
r_c <= '0'; WAIT FOR read_low_time;
mod_begin <= '1';
WHEN mod2 => -- select module 2
m2 <= '1'; WAIT FOR cycle_time;
r_c <= '1';
r_c_data <= no_row & no_col; WAIT FOR read_high_time;
r_c <= '0'; WAIT FOR read_low_time;
mod_begin <= '1';
WHEN mod3 => -- select module 3
m3 <= '1'; WAIT FOR cycle_time;
r_c <= '1';
r_c_data <= no_row & no_col; WAIT FOR read_high_time;
r_c <= '0'; WAIT FOR read_low_time;
mod_begin <= '1';
WHEN mod4 => -- select module 4
m4 <= '1'; WAIT FOR cycle_time;
r_c <= '1';
r_c_data <= no_row & no_col; WAIT FOR read_high_time;
r_c <= '0'; WAIT FOR read_low_time;
mod_begin <= '1';
WHEN mod5 => -- select module 5
m5 <= '1'; WAIT FOR cycle_time;
r_c <= '1';
r_c_data <= no_row & no_col; WAIT FOR read_high_time;
r_c <= '0'; WAIT FOR read_low_time;
mod_begin <= '1';
WHEN mod6 => -- select module 6
m6 <= '1'; WAIT FOR cycle_time;

```

```

r_c <= '1';
r_c_data <= no_row & no_col; WAIT FOR read_high_time;
r_c <= '0'; WAIT FOR read_low_time;
mod_begin <= '1';
WHEN mod7 => -- select module 7
  m7 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN mod8 => -- select module 8
  m8 <= '1'; WAIT FOR cycle_time;
  r_c <= '1';
  r_c_data <= no_row & no_col; WAIT FOR read_high_time;
  r_c <= '0'; WAIT FOR read_low_time;
  mod_begin <= '1';
WHEN OTHERS => NULL;
END CASE;
ELSE
  -- read second byte
  read_instruction <= '1'; WAIT FOR read_high_time;
  byte2 := instruct;
  read_instruction <= '0'; WAIT FOR read_low_time;
  -- execute second byte instruction
  IF byte1 (3 DOWNT0 0) = row THEN
    ASSERT
      (NOT (byte1 (3 DOWNT0 0) = row))
    REPORT
      "Being keep the number of rows"
    SEVERITY NOTE;
    no_row := byte2;
  ELSE
    ASSERT
      (NOT (byte1 (3 DOWNT0 0) = col))
    REPORT
      "Being keep the number of cols"
    SEVERITY NOTE;
    no_col := byte2;
  END IF;
END IF;
END IF;
END PROCESS;
END behavioral;

```

B.3.2 การบรรยายในระดับโครงสร้าง (The Structural Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY ctrl;

```

ENTITY controller IS

```

PORT ( clk, interrupt, mod_end : IN qit;
      instruct : IN byte;
      clear_mem, read_mem, write_mem : OUT qit;
      dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
      abus : OUT word;
      read_instruction : OUT qit;
      en_sm, en_dm : OUT qit;
      m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit;
      r_c : OUT qit;
      r_c_data : OUT word;
      mod_begin : OUT qit;
      dbus_on_databus, databus_on_dbus : OUT qit);

```

END controller;

ARCHITECTURE structural OF controller IS

BEGIN

COMPONENT l_ctrl

```

PORT ( clk : IN qit;
      interrupt, mod_end, cp_rc : IN qit;
      ir_lines : IN byte;
      load_ir, load_row, load_col, load_buffer,
      increment_pc, reset_pc,
      load_page_mar, load_offset_mar,
      rc_on_rcdata_bus,
      dbus_on_buffer, buffer_on_dbus,
      dbus_on_databus, databus_on_dbus,
      pc_on_mar_page_bus, pc_on_mar_offset_bus,
      mar_on_abus,
      read_mem, write_mem, clear_mem,
      en_sm, en_dm,
      read_instr, mod_begin, row_col,
      m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit);

```

END COMPONENT;

FOR logic : l_ctrl USE ENTITY ctrl.logic_of_controller(dataflow);

FOR ALL : reg USE ENTITY reg.register_unit(dataflow);

FOR counter : pc USE ENTITY reg.program_counter_unit(behavioral);

FOR memory_addr : mar USE ENTITY reg.memory_address_register_unit(behavioral);

FOR compare : cprc USE ENTITY reg.compare_row_col(dataflow);

```
FOR ALL : ts8 USE ENTITY reg.tri_state_8 (structural);
FOR ALL : ts16 USE ENTITY reg.tri_state_16 (structural);
```

```
SIGNAL cp_rc : qit;
SIGNAL ir_lines : byte;
SIGNAL rc_lines : word;
SIGNAL pc1_lines, pc2_lines, mar1_lines, mar2_lines : word;
SIGNAL d_buff, buff_d : byte;
SIGNAL dbus1, dbus2 : byte;
SIGNAL load_ir, load_row, load_col, load_buffer,
      increment_pc, reset_pc,
      load_page_mar, load_offset_mar : qit;
SIGNAL rc_on_rcdata_bus,
      dbus_on_buffer, buffer_on_dbus,
      pc_on_mar_page_bus, pc_on_mar_offset_bus,
      mar_on_abus : qit;
```

```
BEGIN
```

```
ir : reg PORT MAP (instruct, ir_lines, load_ir, clk);
row : reg PORT MAP (ir_lines, rc_lines(15 DOWNT0 8), load_row, clk);
col : reg PORT MAP (ir_lines, rc_lines(7 DOWNT0 0), load_col, clk);
rc_data : ts16 PORT MAP (rc_lines, rc_on_rcdata_bus, r_c_data);
buff : reg PORT MAP (d_buff, buff_d, load_buffer, clk);
d_to_buff_1 : dbus1 <= qit_vector (dbus);
d_to_buff_2 : ts8 PORT MAP (dbus1, dbus_on_buffer, d_buff);
buff_to_d_1 : ts8 PORT MAP (buff_d, buffer_on_dbus, dbus2);
buff_to_d_2 : dbus <= wired_byte (dbus2);
counter : pc PORT MAP (pc1_lines, increment_pc, reset_pc, clk);
pc_to_mar_page : ts8 PORT MAP (pc1_lines(15 DOWNT0 8), pc_on_mar_page_bus,
      pc2_lines(15 DOWNT0 8));
pc_to_mar_offset : ts8 PORT MAP (pc1_lines(7 DOWNT0 0), pc_on_mar_offset_bus,
      pc2_lines(7 DOWNT0 0));
memory_addr : mar PORT MAP (pc2_lines, mar1_lines, load_page_mar, load_offset_mar, clk);
mar_to_abus_1 : ts16 PORT MAP (mar1_lines, mar_on_abus, mar2_lines);
mar_to_abus_2 : abus <= mar2_lines;
compare : cprc PORT MAP (rc_lines, mar2_lines, cp_rc);
logic : l_ctl

  PORT MAP ( clk, interrupt, mod_end, cp_rc, ir_lines,
            load_ir, load_row, load_col, load_buffer,
            increment_pc, reset_pc,
            load_page_mar, load_offset_mar,
            rc_on_rcdata_bus,
            dbus_on_buffer, buffer_on_dbus,
            dbus_on_databus, databus_on_dbus,
            pc_on_mar_page_bus, pc_on_mar_offset_bus,
            mar_on_abus,
            read_mem, write_mem, clear_mem,
            en_sm, en_dm,
```

```

        read_instruction, mod_begin, r_c,
        m1, m2, m3, m4, m5, m6, m7, m8);
END structural;

```

B.3.3 การบรรยายในระดับข้อมูลของส่วนควบคุมภายใน

(The Dataflow Description of Logic of Controller)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY logic_of_controller IS
    GENERIC ( read_delay, write_delay : TIME : 3'NS);
    PORT ( clk : IN qit;
          interrupt, mod_end, cp_rc : IN qit;
          ir_lines : IN byte;
          -- register control signals :
          load_ir, load_row, load_col, load_buffer,
          increment_pc, reset_pc,
          load_page_mar, load_offset_mar,
          -- bus connection control signals :
          rc_on_rcdata_bus,
          dbus_on_buffer, buffer_on_dbus,
          dbus_on_databus, databus_on_dbus,
          pc_on_mar_page_bus, pc_on_mar_offset_bus,
          mar_on_abus,
          -- memory control signals :
          read_mem, write_mem, clear_mem,
          en_sm, en_dm,
          -- other control signals :
          read_instr, mod_begin, row_col,
          m1, m2, m3, m4, m5, m6, m7, m8 : OUT qit);

```

```

END logic_of_controller;

```

```

ARCHITECTURE dataflow OF logic_of_controller IS

```

```

    -- oring is implied in the following signals (oi)

```

```

    SIGNAL load_ir_oi, load_row_oi, load_col_oi, load_buffer_oi,
    increment_pc_oi, reset_pc_oi,
    load_page_mar_oi, load_offset_mar_oi,
    rc_on_rcdata_bus_oi,
    dbus_on_buffer_oi, buffer_on_dbus_oi,
    dbus_on_databus_oi, databus_on_dbus_oi,
    pc_on_mar_page_bus_oi, pc_on_mar_offset_bus_oi,
    mar_on_abus_oi,
    read_mem_oi, write_mem_oi, clear_mem_oi,
    en_sm_oi, en_dm_oi,
    read_instr_oi, mod_begin_oi, row_col_oi,
    m1_oi, m2_oi, m3_oi, m4_oi,
    m5_oi, m6_oi, m7_oi, m8_oi : orod_qit BUS;

```

```

SIGNAL s : ored_qit_vector(4 DOWNT0 1) REGISTER := "0001";
BEGIN
load_ir <= load_ir_oi;
load_row <= load_row_oi;
load_col <= load_col_oi;
load_buffer <= load_buffer_oi;
increment_pc <= increment_pc_oi;
reset_pc <= reset_pc_oi;
load_page_mar <= load_page_mar_oi;
load_offset_mar <= load_offset_mar_oi;
rc_on_rcdata_bus <= rc_on_rcdata_bus_oi;
dbus_on_buffer <= dbus_on_buffer_oi;
buffer_on_dbus <= buffer_on_dbus_oi;
dbus_on_databus <= dbus_on_databus_oi;
databus_on_dbus <= databus_on_dbus_oi;
pc_on_mar_page_bus <= pc_on_mar_page_bus_oi;
pc_on_mar_offset_bus <= pc_on_mar_offset_bus_oi;
mar_on_abus <= mar_on_abus_oi;
read_mem <= read_mem_oi;
write_mem <= write_mem_oi;
clear_mem <= clear_mem_oi;
en_sm <= en_sm_oi;
en_dm <= en_dm_oi;
read_instr <= read_instr_oi;
mod_begin <= mod_begin_oi;
row_col <= row_col_oi;
m1 <= m1_oi;
m2 <= m2_oi;
m3 <= m3_oi;
m4 <= m4_oi;
m5 <= m5_oi;
m6 <= m6_oi;
m7 <= m7_oi;
m8 <= m8_oi;

```

```

s1 : BLOCK (s(1) = '1')

```

```

    BEGIN -- start of fetch

```

```

        -- pc to mar

```

```

        pc_on_mar_page_bus_oi <= GUARDED '1';

```

```

        pc_on_mar_offset_bus_oi <= GUARDED '1';

```

```

        load_page_mar_oi <= GUARDED '1';

```

```

        load_offset_mar_oi <= GUARDED '1';

```

```

        -- reset pc if interrupt

```

```

        reset_pc_oi <= GUARDED '1' WHEN interrupt = '1' ELSE '0';

```

```

        -- gato 2 if interrupt is off

```

```

        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)

```

```

            BEGIN

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

s(1) <= GUARDED '1' WHEN interrupt = '1' ELSE '0';
s(2) <= GUARDED '1' WHEN interrupt /= '1' ELSE '0';
END BLOCK ck;
END BLOCK s1;

```

```

s2 : BLOCK (s(2) = '1')
BEGIN -- fetching continue
-- read instruction into ir
read_instr_oi <= GUARDED '1';
load_ir_oi <= GUARDED '1' AFTER read_delay;
-- goto 3
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
s(3) <= GUARDED '1';
END BLOCK ck;
END BLOCK s2;

```

```

s3 : BLOCK (s(3) = '1')
BEGIN
-- pc to mar
pc_on_mar_page_bus_oi <= GUARDED '1';
pc_on_mar_offset_bus_oi <= GUARDED '1';
load_page_mar_oi <= GUARDED '1';
load_offset_mar_oi <= GUARDED '1';
-- goto 4 if not single byte instruction
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
s(4) <= GUARDED '1' WHEN ir_lines(7 DOWNT0 4) /= "0000" ELSE '0';
END BLOCK ck;
-- perform single byte instruction
sb : BLOCK ((ir_lines(7 DOWNT0 4) = "0000") AND GUARD)
BEGIN
-- clear instruction
clm : BLOCK ((ir_lines(3 DOWNT0 0) = "0001") AND GUARD)
BEGIN
en_sm_oi <= GUARDED '1';
en_dm_oi <= GUARDED '1';
clear_mem_oi <= GUARDED '1' AFTER write_delay;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
s(1) <= GUARDED '1';
END BLOCK ck;
END BLOCK clm;
-- swap memory instruction
swm : BLOCK ((ir_lines(3 DOWNT0 0) = "0010") AND GUARD)
BEGIN

```

```

-- read from destination memory
en_sm_oi <= GUARDED '0';
en_dm_oi <= GUARDED '1';
mar_onabus_oi <= GUARDED '1';
read_mem_oi <= GUARDED '1' AFTER read_delay;
databus_ondbus_oi <= GUARDED '1';
dbus_onbuffer_oi <= GUARDED '1';
load_buffer_oi <= GUARDED '1';
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    -- write to source memory
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    mar_onabus_oi <= GUARDED '1';
    buffer_ondbus_oi <= GUARDED '1';
    dbus_ondatabus_oi <= GUARDED '1';
    write_mem_oi <= GUARDED '1' AFTER write_delay;

    -- continue swapping if not ready
    chk : BLOCK (cp_rc /= '1' AND GUARD)
    BEGIN
        -- increment pc
        increment_pc_oi <= GUARDED '1';
        cck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(3) <= GUARDED '1';
        END BLOCK cck;
    END BLOCK chk;
END BLOCK ck;
END BLOCK swm;
-- module 1 instruction
mod1 : BLOCK ((ir_lines(3 DOWNT0 0) = "1000") AND GUARD)
BEGIN
    m1_oi <= GUARDED '1';
    rc_on_rdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod1;
-- module 2 instruction
mod2 : BLOCK ((ir_lines(3 DOWNT0 0) = "1001") AND GUARD)
BEGIN
    m2_oi <= GUARDED '1';
    rc_on_rdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;

```

```

ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    mod_begin_oi <= GUARDED '1';
END BLOCK ck;
END BLOCK mod2;

-- module 3 instruction
mod3 : BLOCK ((ir_lines(3 DOWNT0 0) = "1010") AND GUARD)
BEGIN
    m3_oi <= GUARDED '1';
    rc_on_rcdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)_
    BEGIN
        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod3;

-- module 4 instruction
mod4 : BLOCK ((ir_lines(3 DOWNT0 0) = "1011") AND GUARD)
BEGIN
    m4_oi <= GUARDED '1';
    rc_on_rcdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod4;

-- module 5 instruction
mod5 : BLOCK ((ir_lines(3 DOWNT0 0) = "1100") AND GUARD)
BEGIN
    m5_oi <= GUARDED '1';
    rc_on_rcdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod5;

-- module 6 instruction
mod6 : BLOCK ((ir_lines(3 DOWNT0 0) = "1101") AND GUARD)
BEGIN
    m6_oi <= GUARDED '1';
    rc_on_rcdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN

```

```

        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod6;
-- module 7 instruction
mod7 : BLOCK ((ir_lines(3 DOWNT0 0) = "1110") AND GUARD)
BEGIN
    m7_oi <= GUARDED '1';
    rc_on_rcdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod7;
-- module 8 instruction
mod8 : BLOCK ((ir_lines(3 DOWNT0 0) = "1111") AND GUARD)
BEGIN
    m8_oi <= GUARDED '1';
    rc_on_rcdata_bus_oi <= GUARDED '1';
    row_col_oi <= GUARDED '1' AFTER write_delay;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_begin_oi <= GUARDED '1';
    END BLOCK ck;
END BLOCK mod8;
-- gato 1
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(1) <= GUARDED '1' WHEN (cp_rc = '1' OR mod_end = '1') ELSE '0';
END BLOCK ck;
END BLOCK sb;
END BLOCK s3;
s4 : BLOCK (s(4) = '1')
BEGIN
    lr : BLOCK ((ir_lines(3 DOWNT0 0) = "0011") AND GUARD)
    BEGIN
        read_instr_oi <= GUARDED '1';
        load_ir_oi <= GUARDED '1' AFTER read_delay;
        load_row_oi <= GUARDED '1' AFTER read_delay;
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(1) <= GUARDED '1';
        END BLOCK ck;
    END BLOCK lr;
    lc : BLOCK ((ir_lines(3 DOWNT0 0) = "1100") AND GUARD)
    BEGIN
        read_instr_oi <= GUARDED '1';

```

```

load_ir_oi <= GUARDED '1' AFTER read_delay;
load_col_oi <= GUARDED '1' AFTER read_delay;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(1) <= GUARDED '1';
END BLOCK ck;
END BLOCK lc;
END BLOCK s4;
-- reset state
ck : BLOCK (clk = '0' AND NOT clk'STABLE)
BEGIN
    s(4 DOWNT0 1) <= GUARDED "0000";
END BLOCK ck;
END dataflow;

```

B.4 การบรรยายของวงจรปรับปรุงคุณภาพของภาพ

(The Description of Histogram Equalization)

B.4.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;

ENTITY histogram_equalization IS
    GENERIC ( read_high_time, read_low_time,
              write_high_time, write_low_time : TIME := 2 US;
              cycle_time : TIME := 4 US);
    PORT ( clk, interrupt, r_c, mod_begin : IN qit;
           r_c_data : IN word;
           dbus : INOUT wired_byte BUS := "ZZZZZZZ";
           read_mem, write_mem : OUT qit;
           en_sm, en_dm, mod_end : OUT qit;
           abus : OUT word;
           dbus_on_databus, databus_on_dbus : OUT qit);
END histogram_equalization;

```

ARCHITECTURE behavioral OF histogram_equalization IS

BEGIN

PROCESS

TYPE qit16_array IS ARRAY (0 TO 255) OF word;

TYPE qit8_array IS ARRAY (0 TO 255) OF byte;

VARIABLE nrow, ncol : INTEGER;

VARIABLE pg, offt : INTEGER := 0;

VARIABLE page, offset, buff : byte;

VARIABLE rc15_8, rc7_0 : byte;

VARIABLE pixel_count : qit16_array;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

VARIABLE new_level : qit8_array;
VARIABLE newj1, pix15_8, newj : INTEGER;
BEGIN
  IF interrupt = '1' THEN
    page := zero_8;
    offset := zero_8;
    buff := zero_8;
    IF r_c = '1' THEN --read number of image rows and cols
      rc15_8 := r_c_data(15 DOWNT0 8);
      rc7_0 := r_c_data(7 DOWNT0 0);
      qit2int(rc15_8, nrow);
      qit2int(rc7_0, ncol);
    ELSIF mod_begin = '1' THEN --start histogram equalization process
      --read the grey level of each pixel
      WHILE pg <= nrow LOOP
        WHILE offt <= ncol LOOP
          en_sm <= '1';
          en_dm <= '0';
          int2qit(pg, page);
          int2qit(offt, offset);
          abus <= page & offset;
          read_mem <= '1'; WAIT FOR read_high_time;
          buff := byte(dbus);
          read_mem <= '0'; WAIT FOR read_low_time;
          --compare and count the number of pixels in each level
          CASE buff IS
            WHEN "00000000" => pixel_count(0) := inc(pixel_count(0));
            WHEN "00000001" => pixel_count(1) := inc(pixel_count(1));
            WHEN "00000010" => pixel_count(2) := inc(pixel_count(2));
            WHEN "00000011" => pixel_count(3) := inc(pixel_count(3));
            WHEN "00000100" => pixel_count(4) := inc(pixel_count(4));
            WHEN "00000101" => pixel_count(5) := inc(pixel_count(5));
            WHEN "00000110" => pixel_count(6) := inc(pixel_count(6));
            WHEN "00000111" => pixel_count(7) := inc(pixel_count(7));
            WHEN "00001000" => pixel_count(8) := inc(pixel_count(8));
            WHEN "00001001" => pixel_count(9) := inc(pixel_count(9));
            WHEN "00001010" => pixel_count(10) := inc(pixel_count(10));
            WHEN "00001011" => pixel_count(11) := inc(pixel_count(11));
            WHEN "00001100" => pixel_count(12) := inc(pixel_count(12));
            WHEN "00001101" => pixel_count(13) := inc(pixel_count(13));
            WHEN "00001110" => pixel_count(14) := inc(pixel_count(14));
            WHEN "00001111" => pixel_count(15) := inc(pixel_count(15));
            WHEN "00010000" => pixel_count(16) := inc(pixel_count(16));
            WHEN "00010001" => pixel_count(17) := inc(pixel_count(17));
            WHEN "00010010" => pixel_count(18) := inc(pixel_count(18));
            WHEN "00010011" => pixel_count(19) := inc(pixel_count(19));
            WHEN "00010100" => pixel_count(20) := inc(pixel_count(20));

```

```

WHEN "00010101" => pixel_count(21) := inc(pixel_count(21));
WHEN "00010110" => pixel_count(22) := inc(pixel_count(22));
WHEN "00010111" => pixel_count(23) := inc(pixel_count(23));
WHEN "00011000" => pixel_count(24) := inc(pixel_count(24));
WHEN "00011001" => pixel_count(25) := inc(pixel_count(25));
WHEN "00011010" => pixel_count(26) := inc(pixel_count(26));
WHEN "00011011" => pixel_count(27) := inc(pixel_count(27));
WHEN "00011100" => pixel_count(28) := inc(pixel_count(28));
WHEN "00011101" => pixel_count(29) := inc(pixel_count(29));
WHEN "00011110" => pixel_count(30) := inc(pixel_count(30));
WHEN "00011111" => pixel_count(31) := inc(pixel_count(31));
WHEN "00100000" => pixel_count(32) := inc(pixel_count(32));
WHEN "00100001" => pixel_count(33) := inc(pixel_count(33));
WHEN "00100010" => pixel_count(34) := inc(pixel_count(34));
WHEN "00100011" => pixel_count(35) := inc(pixel_count(35));
WHEN "00100100" => pixel_count(36) := inc(pixel_count(36));
WHEN "00100101" => pixel_count(37) := inc(pixel_count(37));
WHEN "00100110" => pixel_count(38) := inc(pixel_count(38));
WHEN "00100111" => pixel_count(39) := inc(pixel_count(39));
WHEN "00101000" => pixel_count(40) := inc(pixel_count(40));
WHEN "00101001" => pixel_count(41) := inc(pixel_count(41));
WHEN "00101010" => pixel_count(42) := inc(pixel_count(42));
WHEN "00101011" => pixel_count(43) := inc(pixel_count(43));
WHEN "00101100" => pixel_count(44) := inc(pixel_count(44));
WHEN "00101101" => pixel_count(45) := inc(pixel_count(45));
WHEN "00101110" => pixel_count(46) := inc(pixel_count(46));
WHEN "00101111" => pixel_count(47) := inc(pixel_count(47));
WHEN "00110000" => pixel_count(48) := inc(pixel_count(48));
WHEN "00110001" => pixel_count(49) := inc(pixel_count(49));
WHEN "00110010" => pixel_count(50) := inc(pixel_count(50));
WHEN "00110011" => pixel_count(51) := inc(pixel_count(51));
WHEN "00110100" => pixel_count(52) := inc(pixel_count(52));
WHEN "00110101" => pixel_count(53) := inc(pixel_count(53));
WHEN "00110110" => pixel_count(54) := inc(pixel_count(54));
WHEN "00110111" => pixel_count(55) := inc(pixel_count(55));
WHEN "00111000" => pixel_count(56) := inc(pixel_count(56));
WHEN "00111001" => pixel_count(57) := inc(pixel_count(57));
WHEN "00111010" => pixel_count(58) := inc(pixel_count(58));
WHEN "00111011" => pixel_count(59) := inc(pixel_count(59));
WHEN "00111100" => pixel_count(60) := inc(pixel_count(60));
WHEN "00111101" => pixel_count(61) := inc(pixel_count(61));
WHEN "00111110" => pixel_count(62) := inc(pixel_count(62));
WHEN "00111111" => pixel_count(63) := inc(pixel_count(63));
WHEN "01000000" => pixel_count(64) := inc(pixel_count(64));
WHEN "01000001" => pixel_count(65) := inc(pixel_count(65));
WHEN "01000010" => pixel_count(66) := inc(pixel_count(66));
WHEN "01000011" => pixel_count(67) := inc(pixel_count(67));

```

เอกสารนี้เป็นเอกสารลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN "01000100" => pixel_count(68) := inc(pixel_count(68));
WHEN "01000101" => pixel_count(69) := inc(pixel_count(69));
WHEN "01000110" => pixel_count(70) := inc(pixel_count(70));
WHEN "01000111" => pixel_count(71) := inc(pixel_count(71));
WHEN "01001000" => pixel_count(72) := inc(pixel_count(72));
WHEN "01001001" => pixel_count(73) := inc(pixel_count(73));
WHEN "01001010" => pixel_count(74) := inc(pixel_count(74));
WHEN "01001011" => pixel_count(75) := inc(pixel_count(75));
WHEN "01001100" => pixel_count(76) := inc(pixel_count(76));
WHEN "01001101" => pixel_count(77) := inc(pixel_count(77));
WHEN "01001110" => pixel_count(78) := inc(pixel_count(78));
WHEN "01001111" => pixel_count(79) := inc(pixel_count(79));
WHEN "01010000" => pixel_count(80) := inc(pixel_count(80));
WHEN "01010001" => pixel_count(81) := inc(pixel_count(81));
WHEN "01010010" => pixel_count(82) := inc(pixel_count(82));
WHEN "01010011" => pixel_count(83) := inc(pixel_count(83));
WHEN "01010100" => pixel_count(84) := inc(pixel_count(84));
WHEN "01010101" => pixel_count(85) := inc(pixel_count(85));
WHEN "01010110" => pixel_count(86) := inc(pixel_count(86));
WHEN "01010111" => pixel_count(87) := inc(pixel_count(87));
WHEN "01011000" => pixel_count(88) := inc(pixel_count(88));
WHEN "01011001" => pixel_count(89) := inc(pixel_count(89));
WHEN "01011010" => pixel_count(90) := inc(pixel_count(90));
WHEN "01011011" => pixel_count(91) := inc(pixel_count(91));
WHEN "01011100" => pixel_count(92) := inc(pixel_count(92));
WHEN "01011101" => pixel_count(93) := inc(pixel_count(93));
WHEN "01011110" => pixel_count(94) := inc(pixel_count(94));
WHEN "01011111" => pixel_count(95) := inc(pixel_count(95));
WHEN "01100000" => pixel_count(96) := inc(pixel_count(96));
WHEN "01100001" => pixel_count(97) := inc(pixel_count(97));
WHEN "01100010" => pixel_count(98) := inc(pixel_count(98));
WHEN "01100011" => pixel_count(99) := inc(pixel_count(99));
WHEN "01100100" => pixel_count(100) := inc(pixel_count(100));
WHEN "01100101" => pixel_count(101) := inc(pixel_count(101));
WHEN "01100110" => pixel_count(102) := inc(pixel_count(102));
WHEN "01100111" => pixel_count(103) := inc(pixel_count(103));
WHEN "01101000" => pixel_count(104) := inc(pixel_count(104));
WHEN "01101001" => pixel_count(105) := inc(pixel_count(105));
WHEN "01101010" => pixel_count(106) := inc(pixel_count(106));
WHEN "01101011" => pixel_count(107) := inc(pixel_count(107));
WHEN "01101100" => pixel_count(108) := inc(pixel_count(108));
WHEN "01101101" => pixel_count(109) := inc(pixel_count(109));
WHEN "01101110" => pixel_count(110) := inc(pixel_count(110));
WHEN "01101111" => pixel_count(111) := inc(pixel_count(111));
WHEN "01110000" => pixel_count(112) := inc(pixel_count(112));
WHEN "01110001" => pixel_count(113) := inc(pixel_count(113));
WHEN "01110010" => pixel_count(114) := inc(pixel_count(114));

```

เอกสารนี้เป็นเอกสารสงวนลิขสิทธิ์ของสำนักงานส่งเสริมการค้าในต่างประเทศ ณ นครเชียงใหม่

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN "01110011" => pixel_count(115) := inc(pixel_count(115));
WHEN "01110100" => pixel_count(116) := inc(pixel_count(116));
WHEN "01110101" => pixel_count(117) := inc(pixel_count(117));
WHEN "01110110" => pixel_count(118) := inc(pixel_count(118));
WHEN "01110111" => pixel_count(119) := inc(pixel_count(119));
WHEN "01111000" => pixel_count(120) := inc(pixel_count(120));
WHEN "01111001" => pixel_count(121) := inc(pixel_count(121));
WHEN "01111010" => pixel_count(122) := inc(pixel_count(122));
WHEN "01111011" => pixel_count(123) := inc(pixel_count(123));
WHEN "01111100" => pixel_count(124) := inc(pixel_count(124));
WHEN "01111101" => pixel_count(125) := inc(pixel_count(125));
WHEN "01111110" => pixel_count(126) := inc(pixel_count(126));
WHEN "01111111" => pixel_count(127) := inc(pixel_count(127));
WHEN "10000000" => pixel_count(128) := inc(pixel_count(128));
WHEN "10000001" => pixel_count(129) := inc(pixel_count(129));
WHEN "10000010" => pixel_count(130) := inc(pixel_count(130));
WHEN "10000011" => pixel_count(131) := inc(pixel_count(131));
WHEN "10000100" => pixel_count(132) := inc(pixel_count(132));
WHEN "10000101" => pixel_count(133) := inc(pixel_count(133));
WHEN "10000110" => pixel_count(134) := inc(pixel_count(134));
WHEN "10000111" => pixel_count(135) := inc(pixel_count(135));
WHEN "10001000" => pixel_count(136) := inc(pixel_count(136));
WHEN "10001001" => pixel_count(137) := inc(pixel_count(137));
WHEN "10001010" => pixel_count(138) := inc(pixel_count(138));
WHEN "10001011" => pixel_count(139) := inc(pixel_count(139));
WHEN "10001100" => pixel_count(140) := inc(pixel_count(140));
WHEN "10001101" => pixel_count(141) := inc(pixel_count(141));
WHEN "10001110" => pixel_count(142) := inc(pixel_count(142));
WHEN "10001111" => pixel_count(143) := inc(pixel_count(143));
WHEN "10010000" => pixel_count(144) := inc(pixel_count(144));
WHEN "10010001" => pixel_count(145) := inc(pixel_count(145));
WHEN "10010010" => pixel_count(146) := inc(pixel_count(146));
WHEN "10010011" => pixel_count(147) := inc(pixel_count(147));
WHEN "10010100" => pixel_count(148) := inc(pixel_count(148));
WHEN "10010101" => pixel_count(149) := inc(pixel_count(149));
WHEN "10010110" => pixel_count(150) := inc(pixel_count(150));
WHEN "10010111" => pixel_count(151) := inc(pixel_count(151));
WHEN "10011000" => pixel_count(152) := inc(pixel_count(152));
WHEN "10011001" => pixel_count(153) := inc(pixel_count(153));
WHEN "10011010" => pixel_count(154) := inc(pixel_count(154));
WHEN "10011011" => pixel_count(155) := inc(pixel_count(155));
WHEN "10011100" => pixel_count(156) := inc(pixel_count(156));
WHEN "10011101" => pixel_count(157) := inc(pixel_count(157));
WHEN "10011110" => pixel_count(158) := inc(pixel_count(158));
WHEN "10011111" => pixel_count(159) := inc(pixel_count(159));
WHEN "10100000" => pixel_count(160) := inc(pixel_count(160));
WHEN "10100001" => pixel_count(161) := inc(pixel_count(161));

```

เอกสารนี้เป็นเอกสารสงวนลิขสิทธิ์สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่ควรเผยแพร่โดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN '10100010' => pixel_count(162) := inc(pixel_count(162));
WHEN '10100011' => pixel_count(163) := inc(pixel_count(163));
WHEN '10100100' => pixel_count(164) := inc(pixel_count(164));
WHEN '10100101' => pixel_count(165) := inc(pixel_count(165));
WHEN '10100110' => pixel_count(166) := inc(pixel_count(166));
WHEN '10100111' => pixel_count(167) := inc(pixel_count(167));
WHEN '10101000' => pixel_count(168) := inc(pixel_count(168));
WHEN '10101001' => pixel_count(169) := inc(pixel_count(169));
WHEN '10101010' => pixel_count(170) := inc(pixel_count(170));
WHEN '10101011' => pixel_count(171) := inc(pixel_count(171));
WHEN '10101100' => pixel_count(172) := inc(pixel_count(172));
WHEN '10101101' => pixel_count(173) := inc(pixel_count(173));
WHEN '10101110' => pixel_count(174) := inc(pixel_count(174));
WHEN '10101111' => pixel_count(175) := inc(pixel_count(175));
WHEN '10110000' => pixel_count(176) := inc(pixel_count(176));
WHEN '10110001' => pixel_count(177) := inc(pixel_count(177));
WHEN '10110010' => pixel_count(178) := inc(pixel_count(178));
WHEN '10110011' => pixel_count(179) := inc(pixel_count(179));
WHEN '10110100' => pixel_count(180) := inc(pixel_count(180));
WHEN '10110101' => pixel_count(181) := inc(pixel_count(181));
WHEN '10110110' => pixel_count(182) := inc(pixel_count(182));
WHEN '10110111' => pixel_count(183) := inc(pixel_count(183));
WHEN '10111000' => pixel_count(184) := inc(pixel_count(184));
WHEN '10111001' => pixel_count(185) := inc(pixel_count(185));
WHEN '10111010' => pixel_count(186) := inc(pixel_count(186));
WHEN '10111011' => pixel_count(187) := inc(pixel_count(187));
WHEN '10111100' => pixel_count(188) := inc(pixel_count(188));
WHEN '10111101' => pixel_count(189) := inc(pixel_count(189));
WHEN '10111110' => pixel_count(190) := inc(pixel_count(190));
WHEN '10111111' => pixel_count(191) := inc(pixel_count(191));
WHEN '11000000' => pixel_count(192) := inc(pixel_count(192));
WHEN '11000001' => pixel_count(193) := inc(pixel_count(193));
WHEN '11000010' => pixel_count(194) := inc(pixel_count(194));
WHEN '11000011' => pixel_count(195) := inc(pixel_count(195));
WHEN '11000100' => pixel_count(196) := inc(pixel_count(196));
WHEN '11000101' => pixel_count(197) := inc(pixel_count(197));
WHEN '11000110' => pixel_count(198) := inc(pixel_count(198));
WHEN '11000111' => pixel_count(199) := inc(pixel_count(199));
WHEN '11001000' => pixel_count(200) := inc(pixel_count(200));
WHEN '11001001' => pixel_count(201) := inc(pixel_count(201));
WHEN '11001010' => pixel_count(202) := inc(pixel_count(202));
WHEN '11001011' => pixel_count(203) := inc(pixel_count(203));
WHEN '11001100' => pixel_count(204) := inc(pixel_count(204));
WHEN '11001101' => pixel_count(205) := inc(pixel_count(205));
WHEN '11001110' => pixel_count(206) := inc(pixel_count(206));
WHEN '11001111' => pixel_count(207) := inc(pixel_count(207));
WHEN '11010000' => pixel_count(208) := inc(pixel_count(208));

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN '11010001' => pixel_count(209) := inc(pixel_count(209));
WHEN '11010010' => pixel_count(210) := inc(pixel_count(210));
WHEN '11010011' => pixel_count(211) := inc(pixel_count(211));
WHEN '11010100' => pixel_count(212) := inc(pixel_count(212));
WHEN '11010101' => pixel_count(213) := inc(pixel_count(213));
WHEN '11010110' => pixel_count(214) := inc(pixel_count(214));
WHEN '11010111' => pixel_count(215) := inc(pixel_count(215));
WHEN '11011000' => pixel_count(216) := inc(pixel_count(216));
WHEN '11011001' => pixel_count(217) := inc(pixel_count(217));
WHEN '11011010' => pixel_count(218) := inc(pixel_count(218));
WHEN '11011011' => pixel_count(219) := inc(pixel_count(219));
WHEN '11011100' => pixel_count(220) := inc(pixel_count(220));
WHEN '11011101' => pixel_count(221) := inc(pixel_count(221));
WHEN '11011110' => pixel_count(222) := inc(pixel_count(222));
WHEN '11011111' => pixel_count(223) := inc(pixel_count(223));
WHEN '11100000' => pixel_count(224) := inc(pixel_count(224));
WHEN '11100001' => pixel_count(225) := inc(pixel_count(225));
WHEN '11100010' => pixel_count(226) := inc(pixel_count(226));
WHEN '11100011' => pixel_count(227) := inc(pixel_count(227));
WHEN '11100100' => pixel_count(228) := inc(pixel_count(228));
WHEN '11100101' => pixel_count(229) := inc(pixel_count(229));
WHEN '11100110' => pixel_count(230) := inc(pixel_count(230));
WHEN '11100111' => pixel_count(231) := inc(pixel_count(231));
WHEN '11101000' => pixel_count(232) := inc(pixel_count(232));
WHEN '11101001' => pixel_count(233) := inc(pixel_count(233));
WHEN '11101010' => pixel_count(234) := inc(pixel_count(234));
WHEN '11101011' => pixel_count(235) := inc(pixel_count(235));
WHEN '11101100' => pixel_count(236) := inc(pixel_count(236));
WHEN '11101101' => pixel_count(237) := inc(pixel_count(237));
WHEN '11101110' => pixel_count(238) := inc(pixel_count(238));
WHEN '11101111' => pixel_count(239) := inc(pixel_count(239));
WHEN '11110000' => pixel_count(240) := inc(pixel_count(240));
WHEN '11110001' => pixel_count(241) := inc(pixel_count(241));
WHEN '11110010' => pixel_count(242) := inc(pixel_count(242));
WHEN '11110011' => pixel_count(243) := inc(pixel_count(243));
WHEN '11110100' => pixel_count(244) := inc(pixel_count(244));
WHEN '11110101' => pixel_count(245) := inc(pixel_count(245));
WHEN '11110110' => pixel_count(246) := inc(pixel_count(246));
WHEN '11110111' => pixel_count(247) := inc(pixel_count(247));
WHEN '11111000' => pixel_count(248) := inc(pixel_count(248));
WHEN '11111001' => pixel_count(249) := inc(pixel_count(249));
WHEN '11111010' => pixel_count(250) := inc(pixel_count(250));
WHEN '11111011' => pixel_count(251) := inc(pixel_count(251));
WHEN '11111100' => pixel_count(252) := inc(pixel_count(252));
WHEN '11111101' => pixel_count(253) := inc(pixel_count(253));
WHEN '11111110' => pixel_count(254) := inc(pixel_count(254));
WHEN OTHERS => pixel_count(255) := inc(pixel_count(255));

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น มิได้อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

END CASE;
offt := offt + 1;
END LOOP;
offt := 0;
pg := pg + 1;
END LOOP;
--find new grey level
new_level(0) := pixel_count(0)(15 DOWNT0 8);
FOR j IN 1 TO 255 LOOP
  qit2int (new_level(j-1), newj1);
  qit2int (pixel_count(j)(15 DOWNT0 8), pix15_8);
  newj := newj1 + pix15_8;
  int2qit (newj, new_level(j));
END LOOP;
--change to new grey level
pg := 0;
offt := 0;
WHILE pg <= nrow LOOP
  WHILE offt <= ncol LOOP
    en_sm <= '1';
    en_dm <= '0';
    int2qit(pg, page);
    int2qit(offt, offset);
    abus <= page & offset;
    read_mem <= '1'; WAIT FOR read_high_time;
    buff := byte(dbus);
    read_mem <= '0'; WAIT FOR read_low_time;
    en_sm <= '0';
    en_dm <= '1';
    --compare and send the new grey level of each pixels to memory
    CASE buff IS
      WHEN "00000000" => dbus <= wired_byte(new_level(0));
      WHEN "00000001" => dbus <= wired_byte(new_level(1));
      WHEN "00000010" => dbus <= wired_byte(new_level(2));
      WHEN "00000011" => dbus <= wired_byte(new_level(3));
      WHEN "00000100" => dbus <= wired_byte(new_level(4));
      WHEN "00000101" => dbus <= wired_byte(new_level(5));
      WHEN "00000110" => dbus <= wired_byte(new_level(6));
      WHEN "00000111" => dbus <= wired_byte(new_level(7));
      WHEN "00001000" => dbus <= wired_byte(new_level(8));
      WHEN "00001001" => dbus <= wired_byte(new_level(9));
      WHEN "00001010" => dbus <= wired_byte(new_level(10));
      WHEN "00001011" => dbus <= wired_byte(new_level(11));
      WHEN "00001100" => dbus <= wired_byte(new_level(12));
      WHEN "00001101" => dbus <= wired_byte(new_level(13));
      WHEN "00001110" => dbus <= wired_byte(new_level(14));
      WHEN "00001111" => dbus <= wired_byte(new_level(15));

```

```

WHEN "00010000" => dbus <= wired_byte(new_level(16));
WHEN "00010001" => dbus <= wired_byte(new_level(17));
WHEN "00010010" => dbus <= wired_byte(new_level(18));
WHEN "00010011" => dbus <= wired_byte(new_level(19));
WHEN "00010100" => dbus <= wired_byte(new_level(20));
WHEN "00010101" => dbus <= wired_byte(new_level(21));
WHEN "00010110" => dbus <= wired_byte(new_level(22));
WHEN "00010111" => dbus <= wired_byte(new_level(23));
WHEN "00011000" => dbus <= wired_byte(new_level(24));
WHEN "00011001" => dbus <= wired_byte(new_level(25));
WHEN "00011010" => dbus <= wired_byte(new_level(26));
WHEN "00011011" => dbus <= wired_byte(new_level(27));
WHEN "00011100" => dbus <= wired_byte(new_level(28));
WHEN "00011101" => dbus <= wired_byte(new_level(29));
WHEN "00011110" => dbus <= wired_byte(new_level(30));
WHEN "00011111" => dbus <= wired_byte(new_level(31));
WHEN "00100000" => dbus <= wired_byte(new_level(32));
WHEN "00100001" => dbus <= wired_byte(new_level(33));
WHEN "00100010" => dbus <= wired_byte(new_level(34));
WHEN "00100011" => dbus <= wired_byte(new_level(35));
WHEN "00100100" => dbus <= wired_byte(new_level(36));
WHEN "00100101" => dbus <= wired_byte(new_level(37));
WHEN "00100110" => dbus <= wired_byte(new_level(38));
WHEN "00100111" => dbus <= wired_byte(new_level(39));
WHEN "00101000" => dbus <= wired_byte(new_level(40));
WHEN "00101001" => dbus <= wired_byte(new_level(41));
WHEN "00101010" => dbus <= wired_byte(new_level(42));
WHEN "00101011" => dbus <= wired_byte(new_level(43));
WHEN "00101100" => dbus <= wired_byte(new_level(44));
WHEN "00101101" => dbus <= wired_byte(new_level(45));
WHEN "00101110" => dbus <= wired_byte(new_level(46));
WHEN "00101111" => dbus <= wired_byte(new_level(47));
WHEN "00110000" => dbus <= wired_byte(new_level(48));
WHEN "00110001" => dbus <= wired_byte(new_level(49));
WHEN "00110010" => dbus <= wired_byte(new_level(50));
WHEN "00110011" => dbus <= wired_byte(new_level(51));
WHEN "00110100" => dbus <= wired_byte(new_level(52));
WHEN "00110101" => dbus <= wired_byte(new_level(53));
WHEN "00110110" => dbus <= wired_byte(new_level(54));
WHEN "00110111" => dbus <= wired_byte(new_level(55));
WHEN "00111000" => dbus <= wired_byte(new_level(56));
WHEN "00111001" => dbus <= wired_byte(new_level(57));
WHEN "00111010" => dbus <= wired_byte(new_level(58));
WHEN "00111011" => dbus <= wired_byte(new_level(59));
WHEN "00111100" => dbus <= wired_byte(new_level(60));
WHEN "00111101" => dbus <= wired_byte(new_level(61));
WHEN "00111110" => dbus <= wired_byte(new_level(62));

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN "00111111" => dbus <= wired_byte(new_level(63));
WHEN "01000000" => dbus <= wired_byte(new_level(64));
WHEN "01000001" => dbus <= wired_byte(new_level(65));
WHEN "01000010" => dbus <= wired_byte(new_level(66));
WHEN "01000011" => dbus <= wired_byte(new_level(67));
WHEN "01000100" => dbus <= wired_byte(new_level(68));
WHEN "01000101" => dbus <= wired_byte(new_level(69));
WHEN "01000110" => dbus <= wired_byte(new_level(70));
WHEN "01000111" => dbus <= wired_byte(new_level(71));
WHEN "01001000" => dbus <= wired_byte(new_level(72));
WHEN "01001001" => dbus <= wired_byte(new_level(73));
WHEN "01001010" => dbus <= wired_byte(new_level(74));
WHEN "01001011" => dbus <= wired_byte(new_level(75));
WHEN "01001100" => dbus <= wired_byte(new_level(76));
WHEN "01001101" => dbus <= wired_byte(new_level(77));
WHEN "01001110" => dbus <= wired_byte(new_level(78));
WHEN "01001111" => dbus <= wired_byte(new_level(79));
WHEN "01010000" => dbus <= wired_byte(new_level(80));
WHEN "01010001" => dbus <= wired_byte(new_level(81));
WHEN "01010010" => dbus <= wired_byte(new_level(82));
WHEN "01010011" => dbus <= wired_byte(new_level(83));
WHEN "01010100" => dbus <= wired_byte(new_level(84));
WHEN "01010101" => dbus <= wired_byte(new_level(85));
WHEN "01010110" => dbus <= wired_byte(new_level(86));
WHEN "01010111" => dbus <= wired_byte(new_level(87));
WHEN "01011000" => dbus <= wired_byte(new_level(88));
WHEN "01011001" => dbus <= wired_byte(new_level(89));
WHEN "01011010" => dbus <= wired_byte(new_level(90));
WHEN "01011011" => dbus <= wired_byte(new_level(91));
WHEN "01011100" => dbus <= wired_byte(new_level(92));
WHEN "01011101" => dbus <= wired_byte(new_level(93));
WHEN "01011110" => dbus <= wired_byte(new_level(94));
WHEN "01011111" => dbus <= wired_byte(new_level(95));
WHEN "01100000" => dbus <= wired_byte(new_level(96));
WHEN "01100001" => dbus <= wired_byte(new_level(97));
WHEN "01100010" => dbus <= wired_byte(new_level(98));
WHEN "01100011" => dbus <= wired_byte(new_level(99));
WHEN "01100100" => dbus <= wired_byte(new_level(100));
WHEN "01100101" => dbus <= wired_byte(new_level(101));
WHEN "01100110" => dbus <= wired_byte(new_level(102));
WHEN "01100111" => dbus <= wired_byte(new_level(103));
WHEN "01101000" => dbus <= wired_byte(new_level(104));
WHEN "01101001" => dbus <= wired_byte(new_level(105));
WHEN "01101010" => dbus <= wired_byte(new_level(106));
WHEN "01101011" => dbus <= wired_byte(new_level(107));
WHEN "01101100" => dbus <= wired_byte(new_level(108));
WHEN "01101101" => dbus <= wired_byte(new_level(109));

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN "01101110" => dbus <= wired_byte(new_level(110));
WHEN "01101111" => dbus <= wired_byte(new_level(111));
WHEN "01110000" => dbus <= wired_byte(new_level(112));
WHEN "01110001" => dbus <= wired_byte(new_level(113));
WHEN "01110010" => dbus <= wired_byte(new_level(114));
WHEN "01110011" => dbus <= wired_byte(new_level(115));
WHEN "01110100" => dbus <= wired_byte(new_level(116));
WHEN "01110101" => dbus <= wired_byte(new_level(117));
WHEN "01110110" => dbus <= wired_byte(new_level(118));
WHEN "01110111" => dbus <= wired_byte(new_level(119));
WHEN "01111000" => dbus <= wired_byte(new_level(120));
WHEN "01111001" => dbus <= wired_byte(new_level(121));
WHEN "01111010" => dbus <= wired_byte(new_level(122));
WHEN "01111011" => dbus <= wired_byte(new_level(123));
WHEN "01111100" => dbus <= wired_byte(new_level(124));
WHEN "01111101" => dbus <= wired_byte(new_level(125));
WHEN "01111110" => dbus <= wired_byte(new_level(126));
WHEN "01111111" => dbus <= wired_byte(new_level(127));
WHEN "10000000" => dbus <= wired_byte(new_level(128));
WHEN "10000001" => dbus <= wired_byte(new_level(129));
WHEN "10000010" => dbus <= wired_byte(new_level(130));
WHEN "10000011" => dbus <= wired_byte(new_level(131));
WHEN "10000100" => dbus <= wired_byte(new_level(132));
WHEN "10000101" => dbus <= wired_byte(new_level(133));
WHEN "10000110" => dbus <= wired_byte(new_level(134));
WHEN "10000111" => dbus <= wired_byte(new_level(135));
WHEN "10001000" => dbus <= wired_byte(new_level(136));
WHEN "10001001" => dbus <= wired_byte(new_level(137));
WHEN "10001010" => dbus <= wired_byte(new_level(138));
WHEN "10001011" => dbus <= wired_byte(new_level(139));
WHEN "10001100" => dbus <= wired_byte(new_level(140));
WHEN "10001101" => dbus <= wired_byte(new_level(141));
WHEN "10001110" => dbus <= wired_byte(new_level(142));
WHEN "10001111" => dbus <= wired_byte(new_level(143));
WHEN "10010000" => dbus <= wired_byte(new_level(144));
WHEN "10010001" => dbus <= wired_byte(new_level(145));
WHEN "10010010" => dbus <= wired_byte(new_level(146));
WHEN "10010011" => dbus <= wired_byte(new_level(147));
WHEN "10010100" => dbus <= wired_byte(new_level(148));
WHEN "10010101" => dbus <= wired_byte(new_level(149));
WHEN "10010110" => dbus <= wired_byte(new_level(150));
WHEN "10010111" => dbus <= wired_byte(new_level(151));
WHEN "10011000" => dbus <= wired_byte(new_level(152));
WHEN "10011001" => dbus <= wired_byte(new_level(153));
WHEN "10011010" => dbus <= wired_byte(new_level(154));
WHEN "10011011" => dbus <= wired_byte(new_level(155));
WHEN "10011100" => dbus <= wired_byte(new_level(156));

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น เมื่ออนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้คัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN "10011101" => dbus <= wired_byte(new_level(157));
WHEN "10011110" => dbus <= wired_byte(new_level(158));
WHEN "10011111" => dbus <= wired_byte(new_level(159));
WHEN "10100000" => dbus <= wired_byte(new_level(160));
WHEN "10100001" => dbus <= wired_byte(new_level(161));
WHEN "10100010" => dbus <= wired_byte(new_level(162));
WHEN "10100011" => dbus <= wired_byte(new_level(163));
WHEN "10100100" => dbus <= wired_byte(new_level(164));
WHEN "10100101" => dbus <= wired_byte(new_level(165));
WHEN "10100110" => dbus <= wired_byte(new_level(166));
WHEN "10100111" => dbus <= wired_byte(new_level(167));
WHEN "10101000" => dbus <= wired_byte(new_level(168));
WHEN "10101001" => dbus <= wired_byte(new_level(169));
WHEN "10101010" => dbus <= wired_byte(new_level(170));
WHEN "10101011" => dbus <= wired_byte(new_level(171));
WHEN "10101100" => dbus <= wired_byte(new_level(172));
WHEN "10101101" => dbus <= wired_byte(new_level(173));
WHEN "10101110" => dbus <= wired_byte(new_level(174));
WHEN "10101111" => dbus <= wired_byte(new_level(175));
WHEN "10110000" => dbus <= wired_byte(new_level(176));
WHEN "10110001" => dbus <= wired_byte(new_level(177));
WHEN "10110010" => dbus <= wired_byte(new_level(178));
WHEN "10110011" => dbus <= wired_byte(new_level(179));
WHEN "10110100" => dbus <= wired_byte(new_level(180));
WHEN "10110101" => dbus <= wired_byte(new_level(181));
WHEN "10110110" => dbus <= wired_byte(new_level(182));
WHEN "10110111" => dbus <= wired_byte(new_level(183));
WHEN "10111000" => dbus <= wired_byte(new_level(184));
WHEN "10111001" => dbus <= wired_byte(new_level(185));
WHEN "10111010" => dbus <= wired_byte(new_level(186));
WHEN "10111011" => dbus <= wired_byte(new_level(187));
WHEN "10111100" => dbus <= wired_byte(new_level(188));
WHEN "10111101" => dbus <= wired_byte(new_level(189));
WHEN "10111110" => dbus <= wired_byte(new_level(190));
WHEN "10111111" => dbus <= wired_byte(new_level(191));
WHEN "11000000" => dbus <= wired_byte(new_level(192));
WHEN "11000001" => dbus <= wired_byte(new_level(193));
WHEN "11000010" => dbus <= wired_byte(new_level(194));
WHEN "11000011" => dbus <= wired_byte(new_level(195));
WHEN "11000100" => dbus <= wired_byte(new_level(196));
WHEN "11000101" => dbus <= wired_byte(new_level(197));
WHEN "11000110" => dbus <= wired_byte(new_level(198));
WHEN "11000111" => dbus <= wired_byte(new_level(199));
WHEN "11001000" => dbus <= wired_byte(new_level(200));
WHEN "11001001" => dbus <= wired_byte(new_level(201));
WHEN "11001010" => dbus <= wired_byte(new_level(202));
WHEN "11001011" => dbus <= wired_byte(new_level(203));

```

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

WHEN '11001100' => dbus <= wired_byte(new_level(204));
WHEN '11001101' => dbus <= wired_byte(new_level(205));
WHEN '11001110' => dbus <= wired_byte(new_level(206));
WHEN '11001111' => dbus <= wired_byte(new_level(207));
WHEN '11010000' => dbus <= wired_byte(new_level(208));
WHEN '11010001' => dbus <= wired_byte(new_level(209));
WHEN '11010010' => dbus <= wired_byte(new_level(210));
WHEN '11010011' => dbus <= wired_byte(new_level(211));
WHEN '11010100' => dbus <= wired_byte(new_level(212));
WHEN '11010101' => dbus <= wired_byte(new_level(213));
WHEN '11010110' => dbus <= wired_byte(new_level(214));
WHEN '11010111' => dbus <= wired_byte(new_level(215));
WHEN '11011000' => dbus <= wired_byte(new_level(216));
WHEN '11011001' => dbus <= wired_byte(new_level(217));
WHEN '11011010' => dbus <= wired_byte(new_level(218));
WHEN '11011011' => dbus <= wired_byte(new_level(219));
WHEN '11011100' => dbus <= wired_byte(new_level(220));
WHEN '11011101' => dbus <= wired_byte(new_level(221));
WHEN '11011110' => dbus <= wired_byte(new_level(222));
WHEN '11011111' => dbus <= wired_byte(new_level(223));
WHEN '11100000' => dbus <= wired_byte(new_level(224));
WHEN '11100001' => dbus <= wired_byte(new_level(225));
WHEN '11100010' => dbus <= wired_byte(new_level(226));
WHEN '11100011' => dbus <= wired_byte(new_level(227));
WHEN '11100100' => dbus <= wired_byte(new_level(228));
WHEN '11100101' => dbus <= wired_byte(new_level(229));
WHEN '11100110' => dbus <= wired_byte(new_level(230));
WHEN '11100111' => dbus <= wired_byte(new_level(231));
WHEN '11101000' => dbus <= wired_byte(new_level(232));
WHEN '11101001' => dbus <= wired_byte(new_level(233));
WHEN '11101010' => dbus <= wired_byte(new_level(234));
WHEN '11101011' => dbus <= wired_byte(new_level(235));
WHEN '11101100' => dbus <= wired_byte(new_level(236));
WHEN '11101101' => dbus <= wired_byte(new_level(237));
WHEN '11101110' => dbus <= wired_byte(new_level(238));
WHEN '11101111' => dbus <= wired_byte(new_level(239));
WHEN '11110000' => dbus <= wired_byte(new_level(240));
WHEN '11110001' => dbus <= wired_byte(new_level(241));
WHEN '11110010' => dbus <= wired_byte(new_level(242));
WHEN '11110011' => dbus <= wired_byte(new_level(243));
WHEN '11110100' => dbus <= wired_byte(new_level(244));
WHEN '11110101' => dbus <= wired_byte(new_level(245));
WHEN '11110110' => dbus <= wired_byte(new_level(246));
WHEN '11110111' => dbus <= wired_byte(new_level(247));
WHEN '11111000' => dbus <= wired_byte(new_level(248));
WHEN '11111001' => dbus <= wired_byte(new_level(249));
WHEN '11111010' => dbus <= wired_byte(new_level(250));

```

```

    WHEN '11111011' => dbus <= wired_byte(new_level(251));
    WHEN '11111100' => dbus <= wired_byte(new_level(252));
    WHEN '11111101' => dbus <= wired_byte(new_level(253));
    WHEN '11111110' => dbus <= wired_byte(new_level(254));
    WHEN OTHERS      => dbus <= wired_byte(new_level(255));
END CASE;
offt := offt + 1;
END LOOP;
offt := 0;
pg := pg + 1;
END LOOP;
mod_end <= '1'; WAIT FOR cycle_time;
mod_end <= '0';
END IF;
END IF;
END PROCESS;
END behavioral;

```

B.4.2 การบรรยายในระดับโครงสร้าง (The Structural Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY hgram;

ENTITY histogram_equalization IS
    PORT ( clk, interrupt, r_c, mod_begin : IN qit;
          r_c_data : IN word;
          dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
          read_mem, write_mem : OUT qit;
          en_sm, en_dm, mod_end : OUT qit;
          abus : OUT word;
          dbus_on_databus, databus_on_dbus : OUT qit);
END histogram_equalization;

```

ARCHITECTURE structural OF histogram_equalization IS

COMPONENT ctrl_unit

```

    PORT ( clk : IN qit;
          interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
          load_row, load_col, load_buffer,
          increment_pc_page, increment_pc_offset,
          reset_pc_page, reset_pc_offset, reset_pxc,
          load_page_mar, load_offset_mar,
          databus_on_dbus, dbus_on_databus,
          newbus_on_dbus, dec_on_newbus, dec_on_counter,
          pc_on_mar_page_bus, pc_on_mar_offset_bus,

```

```

        mar_on_abus,
        read_mem, write_mem,
        en_sm, en_dm,
        dec_enable, mod_end : OUT qit);
END COMPONENT;

```

```

COMPONENT i256
  PORT ( i : IN qit_vector (0 TO 255);
        c : IN qit;
        o : OUT qit_vector (0 TO 255));
END COMPONENT;

```

```

COMPONENT t8_256
  PORT ( i : IN byte256;
        c : IN qit_vector (0 TO 255);
        o : OUT byte256);
END COMPONENT;

```

```

COMPONENT decoder_unit
  PORT ( i8 : IN byte; en : IN qit;
        o : OUT qit_vector(0 TO 255));
END COMPONENT;

```

```

COMPONENT pxc
  PORT ( o8 : OUT byte256;
        inc : IN qit_vector(0 TO 255);
        reset, ck : IN qit);
END COMPONENT;

```

```

COMPONENT new_gray_level
  PORT ( i8 : IN byte256; o8 : OUT byte256);
END COMPONENT;

```

```

FOR control : ctrl_unit USE ENTITY hgram.histogram_controller(dataflow);
FOR pixel : pxc USE ENTITY hgram.pixel_counter(structural);
FOR dec : decoder_unit USE ENTITY hgram.decoder_8_to_256(dataflow);
FOR new_data : t8_256 USE ENTITY hgram.tri_state_8_256(structural);
FOR ALL : i256 USE ENTITY hgram.tri_state_256(structural);
FOR new_pixel : new_gray_level USE ENTITY hgram.new_level(structural);
FOR ALL : reg USE ENTITY reg.register_unit(dataflow);
FOR row_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);
FOR col_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);
FOR memory_addr : mar USE ENTITY reg.memory_address_register_unit(behavioral);
FOR compare_rc : cprc USE ENTITY reg.compare_row_col(dataflow);
FOR compare_c : cpc USE ENTITY reg.compare_col(dataflow);
FOR ALL : ts8 USE ENTITY reg.tri_state_8(structural);
FOR ALL : ts16 USE ENTITY reg.tri_state_16(structural);

```

```

SIGNAL cp_rc, cp_c : qit;
SIGNAL load_row, load_col, load_buffer,
    increment_pc_page, increment_pc_offset,
    reset_pc_page, reset_pc_offset, reset_pxc,
    load_page_mar, load_offset_mar : qit;
SIGNAL newbus_on_dbus, new_level_on_newbus,
    dec_enable, dec_on_newbus, dec_on_counter,
    pc_on_mar_page_bus, pc_on_mar_offset_bus,
    mar_on_abus : qit;
SIGNAL pc1_lines, pc2_lines, mar1_lines, mar2_lines : word;
SIGNAL rc_lines : word;
SIGNAL buff_lines, dbuff : byte;
SIGNAL dec1_lines, dec2_lines, dec3_lines : qit_vector(0 TO 255);
SIGNAL count_newlevel_lines : byte256;
SIGNAL new1_lines, new2_lines : byte256;
SIGNAL new_bus : wired_byte BUS;
SIGNAL newbus1_lines, newbus2_lines : byte;
BEGIN
    row : reg PORT MAP (r_c_data(15 DOWNT0 8), rc_lines(15 DOWNT0 8), load_row, clk);
    col : reg PORT MAP (r_c_data(7 DOWNT0 0), rc_lines(7 DOWNT0 0), load_col, clk);
    compare_rc : cprc PORT MAP (rc_lines, mar2_lines, cp_rc);
    compare_c : cpc PORT MAP (rc_lines(7 DOWNT0 0), mar2_lines(7 DOWNT0 0), cp_c);
    row_counter : rcpc PORT MAP (pc1_lines(15 DOWNT0 8), increment_pc_page,
        reset_pc_page, clk);
    col_counter : rcpc PORT MAP (pc1_lines(7 DOWNT0 0), increment_pc_offset,
        reset_pc_offset, clk);
    pc_to_mar_page : ts8 PORT MAP (pc1_lines(15 DOWNT0 8), pc_on_mar_page_bus,
        pc2_lines(15 DOWNT0 8));
    pc_to_mar_offset : ts8 PORT MAP (pc1_lines(7 DOWNT0 0), pc_on_mar_offset_bus,
        pc2_lines(7 DOWNT0 0));
    memory_addr : mar PORT MAP (pc2_lines, mar1_lines, load_page_mar, load_offset_mar, clk);
    mar_to_abus_1 : ts16 PORT MAP (mar1_lines, mar_on_abus, mar2_lines);
    mar_to_abus_2 : abus <= mar2_lines;
    buffer_1 : dbuff <= qit_vector(dbus);
    buffer_2 : reg PORT MAP (dbuff, buff_lines, load_buffer, clk);
    dec : decoder_unit PORT MAP (buff_lines, dec_enable, dec1_lines);
    dec_to_count : t256 PORT MAP (dec1_lines, dec_on_counter, dec2_lines);
    pixel : pxc PORT MAP (count_newlevel_lines, dec2_lines, reset_pxc, clk);
    new_pixel : new_gray_level PORT MAP (count_newlevel_lines, new1_lines);
    dec_to_newbus : t256 PORT MAP (dec1_lines, dec_on_newbus, dec3_lines);
    new_data_1 : t8_256 PORT MAP (new1_lines, dec3_lines, new2_lines);
    new_data_2 : newbus <= wired_byte (new2_lines);
    newbus_to_dbus_1 : newbus1_lines <= qit_vector (newbus);
    newbus_to_dbus_2 : ts8 PORT MAP (newbus1_lines, newbus_on_dbus, newbus2_lines);
    newbus_to_dbus_3 : dbus <= wired_byte (newbus2_lines);
    control : ctrl_unit

```

```

load_row, load_col, load_buffer,
increment_pc_page, increment_pc_offset,
reset_pc_page, reset_pc_offset, reset_pxc,
load_page_mar, load_offset_mar,
databus_on_dbus, dbus_on_databus,
newbus_on_dbus, dec_on_newbus, dec_on_counter,
pc_on_mar_page_bus, pc_on_mar_offset_bus,
mar_on_abus,
read_mem, write_mem,
en_sm, en_dm,
dec_enable, mod_end);

```

END structural;

B.4.3 การบรรยายในระดับข้อมูลของส่วนควบคุมภายใน

(The Dataflow Description of Histogram Equalization Controller)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

ENTITY histogram_controller IS
    GENERIC ( read_delay, write_delay : TIME := 3 NS);
    PORT ( clk : IN qit;
          interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
          -- register control signals:
          load_row, load_col, load_buffer,
          increment_pc_page, increment_pc_offset,
          reset_pc_page, reset_pc_offset, reset_pxc,
          load_page_mar, load_offset_mar,
          -- bus connection control signals:
          databus_on_dbus, dbus_on_databus,
          newbus_on_dbus, dec_on_newbus, dec_on_counter,
          pc_on_mar_page_bus, pc_on_mar_offset_bus,
          mar_on_abus,
          -- memory control signals:
          read_mem, write_mem,
          en_sm, en_dm,
          -- other control signal:
          dec_enable, mod_end : OUT qit);
END histogram_controller;

```

ARCHITECTURE dataflow OF histogram_controller IS

```

-- oring is implied in the following signals (oi)
SIGNAL load_row_oi, load_col_oi, load_buffer_oi,
        increment_pc_page_oi, increment_pc_offset_oi,
        reset_pc_page_oi, reset_pc_offset_oi, reset_pxc_oi,
        load_page_mar_oi, load_offset_mar_oi,
        databus_on_dbus_oi, dbus_on_databus_oi,

```

```

newbus_on_dbus_oi, dec_on_newbus_oi, dec_on_counter_oi,
pc_on_mar_page_bus_oi, pc_on_mar_offset_bus_oi,
mar_on_abus_oi,
read_mem_oi, write_mem_oi,
en_sm_oi, en_dm_oi,
dec_enable_oi, mod_end_oi : ored_qit BUS;
SIGNAL s : ored_qit_vector (3 DOWNT0 1) REGISTER := '001';
BEGIN -- start of histogram
load_row <= load_row_oi;
load_col <= load_col_oi;
load_buffer <= load_buffer_oi;
increment_pc_page <= increment_pc_page_oi;
increment_pc_offset <= increment_pc_offset_oi;
reset_pc_page <= reset_pc_page_oi;
reset_pc_offset <= reset_pc_offset_oi;
reset_pxc <= reset_pxc_oi;
load_page_mar <= load_page_mar_oi;
load_offset_mar <= load_offset_mar_oi;
databus_on_dbus <= databus_on_dbus_oi;
dbus_on_databus <= dbus_on_databus_oi;
newbus_on_dbus <= newbus_on_dbus_oi;
dec_on_newbus <= dec_on_newbus_oi;
dec_on_counter <= dec_on_counter_oi;
pc_on_mar_page_bus <= pc_on_mar_page_bus_oi;
pc_on_mar_offset_bus <= pc_on_mar_offset_bus_oi;
mar_on_abus <= mar_on_abus_oi;
read_mem <= read_mem_oi;
write_mem <= write_mem_oi;
en_sm <= en_sm_oi;
en_dm <= en_dm_oi;
dec_enable <= dec_enable_oi;
mod_end <= mod_end_oi;

s1 : BLOCK (s(1) = '1')
BEGIN
-- pc to mar
pc_on_mar_page_bus_oi <= GUARDED '1';
pc_on_mar_offset_bus_oi <= GUARDED '1';
load_page_mar_oi <= GUARDED '1';
load_offset_mar_oi <= GUARDED '1';
-- reset pc and pxc if no interrupt
reset_pc_page_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
reset_pc_offset_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
reset_pxc_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
int : BLOCK (interrupt = '1' AND GUARD)
BEGIN -- load rows and cols
load_row_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';

```

```

load_col_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
-- goto 2 if module begin
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(1) <= GUARDED '1' WHEN mod_begin = '0' ELSE '0';
    s(2) <= GUARDED '1' WHEN mod_begin /= '0' ELSE '0';
END BLOCK ck;
END BLOCK int;
END BLOCK s1;
s2 : BLOCK (s(2) = '1')
BEGIN -- count the number of pixels in each level
    -- pc to mar
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';
    load_page_mar_oi <= GUARDED '1';
    load_offset_mar_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    mar_on_abus_oi <= GUARDED '1';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    load_buffer_oi <= GUARDED '1';
    dec_enable_oi <= GUARDED '1';
    dec_on_counter_oi <= GUARDED '1';
    chk : BLOCK (cp_rc /= '1' AND GUARD)
    BEGIN
        -- increment pc offset
        increment_pc_offset_oi <= GUARDED '1' WHEN cp_c /= '1';
        -- increment pc page and reset pc offset
        increment_pc_page_oi <= GUARDED '1' WHEN cp_c = '1';
        reset_pc_offset_oi <= GUARDED '1' WHEN cp_c = '1';
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(2) <= GUARDED '1';
        END BLOCK ck;
    END BLOCK chk;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        s(3) <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
        reset_pc_page_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
        reset_pc_offset_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
    END BLOCK ck;
END BLOCK s2;
s3 : BLOCK (s(3) = '1')
BEGIN -- change to new level
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';

```

```

load_page_mar_oi <= GUARDED '1';
load_offset_mar_oi <= GUARDED '1';
-- read from source memory
en_sm_oi <= GUARDED '1';
en_dm_oi <= GUARDED '0';
mar_onabus_oi <= GUARDED '1';
read_mem_oi <= GUARDED '1' AFTER read_delay;
databus_on_dbus_oi <= GUARDED '1';
load_buffer_oi <= GUARDED '1';
dec_enable_oi <= GUARDED '1';
dec_on_newbus_oi <= GUARDED '1';
wck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    -- write to destination memory
    en_sm_oi <= GUARDED '0';
    en_dm_oi <= GUARDED '1';
    mar_onabus_oi <= GUARDED '1';
    newbus_on_dbus_oi <= GUARDED '1';
    dbus_on_databus_oi <= GUARDED '1';
    write_mem_oi <= GUARDED '1' AFTER write_delay;
    chk : BLOCK (cp_rc /= '1' AND GUARD)
    BEGIN
        -- increment pc offset
        increment_pc_offset_oi <= GUARDED '1' WHEN cp_c /= '1';
        -- increment pc page and reset pc offset
        increment_pc_page_oi <= GUARDED '1' WHEN cp_c = '1';
        reset_pc_offset_oi <= GUARDED '1' WHEN cp_c = '1';
        cck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(3) <= GUARDED '1';
        END BLOCK cck;
    END BLOCK chk;
END BLOCK wck;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    mod_end_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
END BLOCK ck;
END BLOCK s3;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(3 DOWNT0 1) <= GUARDED '000';
END BLOCK ck;
END dataflow;

```

B.5 การบรรยายของวงจรตรวจหาขอบของวัตถุ

(The Description of Edge Detection)

B.5.1 การบรรยายในระดับพฤติกรรม (The Behavioral Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
USE util.image_parameters.ALL;

ENTITY edge_detection IS
    GENERIC ( read_high_time, read_low_time,
              write_high_time, write_low_time : TIME := 2 US;
              cycle_time : TIME := 4 US);
    PORT ( clk, interrupt, r_c, mod_begin : IN qit;
           r_c_data : IN word;
           dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
           read_mem, write_mem : OUT qit;
           en_sm, en_dm, mod_end : OUT qit;
           abus : OUT word;
           dbus_on_databus, databus_on_dbus : OUT qit);
END edge_detection;

```

ARCHITECTURE behavioral OF edge_detection IS

BEGIN

PROCESS

TYPE sobel IS ARRAY (INTEGER RANGE <>) OF INTEGER;

VARIABLE nrow, ncol : INTEGER;

VARIABLE page, offset, buff : byte;

VARIABLE i, j, gx, gy : INTEGER := 0;

VARIABLE z : sobel (1 TO 9);

VARIABLE grd : qit_vector(9 DOWNT0 0);

VARIABLE rc15_8, rc7_0 : byte;

BEGIN

IF interrupt = '1' THEN

page := zero_8;

offset := zero_8;

-- read the number of rows and cols

IF r_c = '1' THEN

rc15_8 := r_c_data (15 DOWNT0 8);

rc7_0 := r_c_data (7 DOWNT0 0);

qit2int (rc15_8, nrow);

qit2int (rc7_0, ncol);

-- find the gradient that use the sobel operator

ELSIF mod_begin = '1' THEN

-- read 3X3 matrix

WHILE (i < nrow - 2) LOOP

WHILE (j < ncol - 2) LOOP

```

en_sm <= '1';
en_dm <= '0';
FOR k IN 1 TO 9 LOOP
CASE k IS
WHEN 1 =>
    int2qit (i, page);
    int2qit (j, offset);
WHEN 2 =>
    int2qit (i, page);
    int2qit (j+1, offset);
WHEN 3 =>
    int2qit (i, page);
    int2qit (j+2, offset);
WHEN 4 =>
    int2qit (i+1, page);
    int2qit (j, offset);
WHEN 5 =>
    int2qit (i+1, page);
    int2qit (j+1, offset);
WHEN 6 =>
    int2qit (i+1, page);
    int2qit (j+2, offset);
WHEN 7 =>
    int2qit (i+2, page);
    int2qit (j, offset);
WHEN 8 =>
    int2qit (i+2, page);
    int2qit (j+1, offset);
WHEN 9 =>
    int2qit (i+2, page);
    int2qit (j+2, offset);
END CASE;
abus <= page & offset;
read_mem <= '1'; WAIT FOR read_high_time;
buff := byte(dbus);
qit2int (buff, z(k));
read_mem <= '0'; WAIT FOR read_low_time;
END LOOP;
gx := (z(7) + 2*z(8) + z(9)) - (z(1) + 2*z(2) + z(3));
gy := (z(3) + 2*z(6) + z(9)) - (z(1) + 2*z(4) + z(7));
-- absolute gradient
IF gx < 0 THEN
    gx := -gx;
END IF;
IF gy < 0 THEN
    gy := -gy;
END IF;

```

```

-- send to destination memory
en_sm <= '0';
en_dm <= '1';
int2qit ((i+1), page);
int2qit ((j+1), offset);
int2qit ((gx + gy), grd);
abus <= page & offset;
dbus <= wired_byte (grd(9 DOWNT0 2));
write_mem <= '1'; WAIT FOR write_high_time;
write_mem <= '0'; WAIT FOR write_low_time;
dbus <= "ZZZZZZZZ";
j := j + 1;
END LOOP;
j := 0;
i := i + 1;
END LOOP;
mod_end <= '1'; WAIT FOR cycle_time;
mod_end <= '0';
END IF;
END IF;
END PROCESS;
END behavioral;

```

B.5.2 การบรรยายในระดับโครงสร้าง (The Structural Description)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;
LIBRARY reg;
USE reg.register_logic.ALL;
LIBRARY edge;

ENTITY edge_detection IS
  PORT ( clk, interrupt, r_c, mod_begin : IN qit;
         r_c_data : IN word;
         dbus : INOUT wired_byte BUS := "ZZZZZZZZ";
         read_mem, write_mem : OUT qit;
         en_sm, en_dm, mod_end : OUT qit;
         abus : OUT word;
         dbus_on_databus, databus_on_dbus : OUT qit);
END edge_detection;

```

ARCHITECTURE dataflow OF edge_detection IS

COMPONENT ctrl_unit

```

  PORT ( clk : IN qit;
         interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;
         load_row, load_col,
         load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,

```

```

load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
increment_pc_page; increment_pc_offset,
reset_pc_page, reset_pc_offset,
load_page_mar, load_offset_mar,
load_z_sar,
pass_z1_sar, pass_z2_sar, pass_z3_sar,
pass_z4_sar, pass_z4_sar, pass_z6_sar,
pass_z7_sar, pass_z8_sar, pass_z9_sar,
dbus_on_databus, databus_on_dbus,
grd_on_dbus, dbus_on_sbus,
pc_on_mar_page_bus, pc_on_mar_offset_bus,
mar_on_sar_bus, sar_onabus,
read_mem, write_mem,
en_sm, en_dm,
mod_end : OUT qit);
END COMPONENT;
COMPONENT sar
  PORT ( i16 : IN word; o16 : OUT wired_word;
         load_z, ck : IN qit;
         pass_z : IN qit_vector (1 TO 9));
END COMPONENT;
COMPONENT sdr
  PORT ( din : IN byte;
         o1, o2, o3, o4, o6, o7, o8, o9 : OUT byte;
         i1, i2, i3, i4, i6, i7, i8, i9 : IN qit;
         ck : IN qit);
END COMPONENT;
COMPONENT grd
  PORT ( i1, i2, i3, i4, i6, i7, i8, i9 : IN byte;
         o1 : OUT byte);
END COMPONENT;
FOR control : ctrl_unit USE ENTITY edge.edge_controller(dataflow);
FOR sobel_address : sar USE ENTITY edge.sobel_address_register_unit(behavioral);
FOR sobel_matrix : sdr USE ENTITY edge.sobel_data_register_unit(dataflow);
FOR gradient : grd USE ENTITY edge.gradient(behavioral);
FOR ALL : reg USE ENTITY reg.register_unit(dataflow);
FOR row_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);
FOR col_counter : rcpc USE ENTITY reg.row_col_counter_unit(behavioral);
FOR memory_addr : mar USE ENTITY reg.memory_address_register_unit(behavioral);
FOR compare_rc : cprc USE ENTITY reg.compare_row_col(dataflow);
FOR compare_c : cpc USE ENTITY reg.compare_col(dataflow);
FOR ALL : ts8 USE ENTITY reg.tri_state_8(structural);
FOR ALL : ts16 USE ENTITY reg.tri_state_16(structural);

SIGNAL cp_rc, cp_c : qit;
SIGNAL load_row, load_col,

```

```

load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
increment_pc_page, increment_pc_offset,
reset_pc_page, reset_pc_offset,
load_page_mar, load_offset_mar,
load_z_sar,
pass_z1_sar, pass_z2_sar, pass_z3_sar,
pass_z4_sar, pass_z5_sar, pass_z6_sar,
pass_z7_sar, pass_z8_sar, pass_z9_sar : qit;
SIGNAL dbus_on_databus, databus_on_dbus,
grd_on_dbus, dbus_on_sbus,
pc_on_mar_page_bus, pc_on_mar_offset_bus,
mar_on_sar_bus, sar_on_abus : qit;
SIGNAL pc1_lines, pc2_lines, mar1_lines, mar2_lines : word;
SIGNAL rc_lines : word;
SIGNAL sar1_lines, sar2_lines : word;
SIGNAL sbus1_lines, sbus2_lines : byte;
SIGNAL gbus1_lines, gbus2_lines : byte;
SIGNAL sdr1_lines, sdr2_lines, sdr3_lines, sdr4_lines,
sdr6_lines, sdr7_lines, sdr8_lines, sdr9_lines : byte;
SIGNAL pass_z_sar : qit_vector (1 TO 9);
BEGIN
row : reg PORT MAP (r_c_data(15 DOWNT0 8), rc_lines(15 DOWNT0 8), load_row, clk);
col : reg PORT MAP (r_c_data(7 DOWNT0 0), rc_lines(7 DOWNT0 0), load_col, clk);
compare_rc : cpc PORT MAP (rc_lines, sar2_lines, cp_rc);
compare_c : cpc PORT MAP (rc_lines(7 DOWNT0 0), sar2_lines(7 DOWNT0 0), cp_c);
row_counter : rcpc PORT MAP (pc1_lines(15 DOWNT0 8), increment_pc_page,
reset_pc_page, clk);
col_counter : rcpc PORT MAP (pc1_lines(7 DOWNT0 0), increment_pc_offset,
reset_pc_offset, clk);
pc_to_mar_page : ts8 PORT MAP (pc1_lines(15 DOWNT0 8), pc_on_mar_page_bus,
pc2_lines(15 DOWNT0 8));
pc_to_mar_offset : ts8 PORT MAP (pc1_lines(7 DOWNT0 0), pc_on_mar_offset_bus,
pc2_lines(7 DOWNT0 0));
memory_addr : mar PORT MAP (pc2_lines, mar1_lines, load_page_mar, load_offset_mar, clk);
mar_to_sar : ts16 PORT MAP (mar1_lines, mar_on_sar_bus, mar2_lines);
pass_z : pass_z_sar <= pass_z1_sar & pass_z2_sar & pass_z3_sar & pass_z4_sar &
pass_z5_sar & pass_z6_sar & pass_z7_sar & pass_z8_sar &
pass_z9_sar;
sobel_address : sar PORT MAP (mar2_lines, sar1_lines, load_z_sar, clk, pass_z_sar);
sar_to_abus : ts16 PORT MAP (sar1_lines, sar_on_abus, sar2_lines);
dbus_to_sbus_1 : sbus1_lines <= qit_vector (dbus);
dbus_to_sbus_2 : ts8 PORT MAP (sbus1_lines, dbus_on_sbus, sbus2_lines);
sobel_matrix : sdr PORT MAP (sbus2_lines, sdr1_lines, sdr2_lines, sdr3_lines,
sdr4_lines, sdr6_lines, sdr7_lines, sdr8_lines,
sdr9_lines, load_z1_sdr, load_z2_sdr, load_z3_sdr,
load_z4_sdr, load_z6_sdr, load_z7_sdr, load_z8_sdr,
load_z9_sdr, clk);

```

```

gradient : grd PORT MAP (sdr1_lines, sdr2_lines, sdr3_lines, sdr4_lines,
                          sdr6_lines, sdr7_lines, sdr8_lines, sdr9_lines,
                          gbus1_lines);
gbus_to_dbus_1 : ts8 PORT MAP (gbus1_lines, grd_on_dbus, gbus2_lines);
gbus_to_dbus_2 : dbus <= wired_byte (gbus2_lines);
control : ctrl_unit
    PORT MAP (clk, interrupt, mod_begin, cp_rc, cp_c, r_c,
              load_row, load_col,
              load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,
              load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,
              increment_pc_page, increment_pc_offset,
              reset_pc_page, reset_pc_offset,
              load_page_mar, load_offset_mar,
              load_z_sar,
              pass_z1_sar, pass_z2_sar, pass_z3_sar,
              pass_z4_sar, pass_z4_sar, pass_z6_sar,
              pass_z7_sar, pass_z8_sar, pass_z9_sar,
              dbus_on_databus, databus_on_dbus,
              grd_on_dbus, dbus_on_sbus,
              pc_on_mar_page_bus, pc_on_mar_offset_bus,
              mar_on_sar_bus, sar_on_abus,
              read_mem, write_mem,
              en_sm, en_dm,
              mod_end);

```

END structural;

B.5.3 การบรรยายในระดับข้อมูลของส่วนควบคุมภายใน

(The Dataflow Description of Edge Detection Controller)

```

LIBRARY util;
USE util.basic_utilities.ALL;
USE util.image_utilities.ALL;

```

ENTITY edge_controller IS

GENERIC (read_delay, write_delay : TIME := 3 NS; delay : TIME := 3 NS);

PORT (clk : IN qit;

interrupt, mod_begin, cp_rc, cp_c, r_c : IN qit;

-- register control signals:

load_row, load_col,

load_z1_sdr, load_z2_sdr, load_z3_sdr, load_z4_sdr,

load_z6_sdr, load_z7_sdr, load_z8_sdr, load_z9_sdr,

increment_pc_page, increment_pc_offset,

reset_pc_page, reset_pc_offset,

load_page_mar, load_offset_mar,

load_z_sar,

pass_z1_sar, pass_z2_sar, pass_z3_sar,

pass_z4_sar, pass_z5_sar, pass_z6_sar,

pass_z7_sar, pass_z8_sar, pass_z9_sar,

```

-- bus connection control signals:
dbus_on_databus, databus_on_dbus,
grd_on_dbus, dbus_on_sbus,
pc_on_mar_page_bus, pc_on_mar_offset_bus,
mar_on_sar_bus, sar_on_abus,
-- memory control signals:
read_mem, write_mem,
en_sm, en_dm,
-- other control signal:
mod_end : OUT qit);
END edge_controller;

```

ARCHITECTURE dataflow OF edge_controller IS

-- oring is implied in the following signals (oi)

```

SIGNAL load_row_oi, load_col_oi,
load_z1_sdr_oi, load_z2_sdr_oi, load_z3_sdr_oi, load_z4_sdr_oi,
load_z6_sdr_oi, load_z7_sdr_oi, load_z8_sdr_oi, load_z9_sdr_oi,
increment_pc_page_oi, increment_pc_offset_oi,
reset_pc_page_oi, reset_pc_offset_oi,
load_page_mar_oi, load_offset_mar_oi,
load_z_sar_oi,
pass_z1_sar_oi, pass_z2_sar_oi, pass_z3_sar_oi,
pass_z4_sar_oi, pass_z5_sar_oi, pass_z6_sar_oi,
pass_z7_sar_oi, pass_z8_sar_oi, pass_z9_sar_oi,
dbus_on_databus_oi, databus_on_dbus_oi,
grd_on_dbus_oi, dbus_on_sbus_oi,
pc_on_mar_page_bus_oi, pc_on_mar_offset_bus_oi,
mar_on_sar_bus_oi, sar_on_abus_oi,
read_mem_oi, write_mem_oi,
en_sm_oi, en_dm_oi,
mod_end_oi : ored_qit BUS;

SIGNAL s : ored_qit_vector (5 DOWNT0 1) REGISTER := "00001";
SIGNAL z : ored_qit_vector (8 DOWNT0 1) REGISTER := "00000001";

```

BEGIN

```

load_row <= load_row_oi;
load_col <= load_col_oi;
load_z1_sdr <= load_z1_sdr_oi;
load_z2_sdr <= load_z2_sdr_oi;
load_z3_sdr <= load_z3_sdr_oi;
load_z4_sdr <= load_z4_sdr_oi;
load_z6_sdr <= load_z6_sdr_oi;
load_z7_sdr <= load_z7_sdr_oi;
load_z8_sdr <= load_z8_sdr_oi;
load_z9_sdr <= load_z9_sdr_oi;
increment_pc_page <= increment_pc_page_oi;
increment_pc_offset <= increment_pc_offset_oi;
reset_pc_page <= reset_pc_page_oi;

```

```

reset_pc_offset <= reset_pc_offset_oi;
load_page_mar <= load_page_mar_oi;
load_offset_mar <= load_offset_mar_oi;
load_z_sar <= load_z_sar_oi;
pass_z1_sar <= pass_z1_sar_oi;
pass_z2_sar <= pass_z2_sar_oi;
pass_z3_sar <= pass_z3_sar_oi;
pass_z4_sar <= pass_z4_sar_oi;
pass_z5_sar <= pass_z5_sar_oi;
pass_z6_sar <= pass_z6_sar_oi;
pass_z7_sar <= pass_z7_sar_oi;
pass_z8_sar <= pass_z8_sar_oi;
pass_z9_sar <= pass_z9_sar_oi;
dbus_on_databus <= dbus_on_databus_oi;
databus_on_dbus <= databus_on_dbus_oi;
grd_on_dbus <= grd_on_dbus_oi;
dbus_on_sbus <= dbus_on_sbus_oi;
pc_on_mar_page_bus <= pc_on_mar_page_bus_oi;
pc_on_mar_offset_bus <= pc_on_mar_offset_bus_oi;
mar_on_sar_bus <= mar_on_sar_bus_oi;
sar_on_abus <= sar_on_abus_oi;
read_mem <= read_mem_oi;
write_mem <= write_mem_oi;
en_sm <= en_sm_oi;
en_dm <= en_dm_oi;
mod_end <= mod_end_oi;

s1 : BLOCK (s(1) = '1')
BEGIN
    -- pc to mar
    pc_on_mar_page_bus_oi <= GUARDED '1';
    pc_on_mar_offset_bus_oi <= GUARDED '1';
    load_page_mar_oi <= GUARDED '1';
    load_offset_mar_oi <= GUARDED '1';
    -- reset pc if no interrupt
    reset_pc_page_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
    reset_pc_offset_oi <= GUARDED '1' WHEN interrupt = '0' ELSE '0';
    int : BLOCK (interrupt = '1' AND GUARD)
        BEGIN -- load rows and cols
            load_row_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
            load_col_oi <= GUARDED '1' WHEN r_c = '1' ELSE '0';
            -- goto 2 if module begin
            ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
                BEGIN
                    s(1) <= GUARDED '1' WHEN mod_begin = '0' ELSE '0';
                    s(2) <= GUARDED '1' WHEN mod_begin /= '0' ELSE '0';
                END BLOCK ck;
        END BLOCK int;
END BLOCK s1;

```

```

        END BLOCK int;
    END BLOCK s1;
s2 : BLOCK (s(2) = '1')
    BEGIN -- set sobel address register
        -- pc to mar to sar
        pc_on_mar_page_bus_oi <= GUARDED '1';
        pc_on_mar_offset_bus_oi <= GUARDED '1';
        load_page_mar_oi <= GUARDED '1';
        load_offset_mar_oi <= GUARDED '1';
        mar_on_sar_bus_oi <= GUARDED '1';
        load_z_sar_oi <= GUARDED '1' AFTER delay;
        -- goto 3
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
            BEGIN
                s(3) <= GUARDED '1';
            END BLOCK ck;
    END BLOCK s2;
s3 : BLOCK (s(3) = '1')
    BEGIN -- read matrix data into sdr
        z1 : BLOCK (z(1) = '1')
            BEGIN
                pass_z1_sar_oi <= GUARDED '1';
                sar_onabus_oi <= GUARDED '1';
                en_sm_oi <= GUARDED '1';
                en_dm_oi <= GUARDED '0';
                read_mem_oi <= GUARDED '1' AFTER read_delay;
                databus_on_dbus_oi <= GUARDED '1';
                dbus_on_sbus_oi <= GUARDED '1';
                load_z1_sdr_oi <= GUARDED '1';
                -- goto z2
                ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
                    BEGIN
                        z(2) <= GUARDED '1';
                    END BLOCK ck;
            END BLOCK z1;
        z2 : BLOCK (z(2) = '1')
            BEGIN
                pass_z2_sar_oi <= GUARDED '1';
                sar_onabus_oi <= GUARDED '1';
                en_sm_oi <= GUARDED '1';
                en_dm_oi <= GUARDED '0';
                read_mem_oi <= GUARDED '1' AFTER read_delay;
                databus_on_dbus_oi <= GUARDED '1';
                dbus_on_sbus_oi <= GUARDED '1';
                load_z2_sdr_oi <= GUARDED '1';
                -- goto z3
                ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)

```

```

BEGIN
    z(3) <= GUARDED '1';
END BLOCK ck;
END BLOCK z2;
z3 : BLOCK (z(3) = '1')
BEGIN
    pass_z3_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z3_sdr_oi <= GUARDED '1';
    -- goto z4
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        z(4) <= GUARDED '1';
    END BLOCK ck;
END BLOCK z3;
z4 : BLOCK (z(4) = '1')
BEGIN
    pass_z4_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z4_sdr_oi <= GUARDED '1';
    -- goto z5
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        z(5) <= GUARDED '1';
    END BLOCK ck;
END BLOCK z4;
z5 : BLOCK (z(5) = '1')
BEGIN
    pass_z6_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z6_sdr_oi <= GUARDED '1';

```

```

-- goto z6
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    z(6) <= GUARDED '1';
END BLOCK ck;
END BLOCK z5;
z6 : BLOCK (z(6) = '1')
BEGIN
    pass_z7_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z7_sdr_oi <= GUARDED '1';
    -- goto z7
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        z(7) <= GUARDED '1';
    END BLOCK ck;
END BLOCK z6;
z7 : BLOCK (z(7) = '1')
BEGIN
    pass_z8_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';
    load_z8_sdr_oi <= GUARDED '1';
    -- goto z8
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        z(8) <= GUARDED '1';
    END BLOCK ck;
END BLOCK z7;
z8 : BLOCK (z(8) = '1')
BEGIN
    pass_z9_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '1';
    en_dm_oi <= GUARDED '0';
    read_mem_oi <= GUARDED '1' AFTER read_delay;
    databus_on_dbus_oi <= GUARDED '1';
    dbus_on_sbus_oi <= GUARDED '1';

```

```

load_z9_sdr_oi <= GUARDED '1';
-- goto 4
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(4) <= GUARDED '1';
END BLOCK ck;
END BLOCK z8;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    z(8 DOWNT0 1) <= GUARDED "00000000";
END BLOCK ck;
END BLOCK s3;
s4 : BLOCK (s(4) = '1')
BEGIN -- write gradient data
    pass_z5_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    grd_on_dbus_oi <= GUARDED '1';
    dbus_on_databus_oi <= GUARDED '1';
    en_sm_oi <= GUARDED '0';
    en_dm_oi <= GUARDED '1';
    write_mem_oi <= GUARDED '1' AFTER write_delay;
-- goto 5
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(5) <= GUARDED '1';
END BLOCK ck;
END BLOCK s4;
s5 : BLOCK (s(5) = '1')
BEGIN -- check end of image
    pass_z9_sar_oi <= GUARDED '1';
    sar_on_abus_oi <= GUARDED '1';
    chk : BLOCK (cp_rc /= '1' AND GUARD)
    BEGIN
        -- increment pc offset
        increment_pc_offset_oi <= GUARDED '1' WHEN cp_c /= '1';
        -- increment pc page and reset pc offset
        increment_pc_page_oi <= GUARDED '1' WHEN cp_c = '1';
        reset_pc_offset_oi <= GUARDED '1' WHEN cp_c = '1';
        ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
        BEGIN
            s(2) <= GUARDED '1';
        END BLOCK ck;
    END BLOCK chk;
    ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
    BEGIN
        mod_end_oi <= GUARDED '1' WHEN cp_rc = '1' ELSE '0';
    END BLOCK ck;

```

```

END BLOCK s5;
ck : BLOCK ((clk = '0' AND NOT clk'STABLE) AND GUARD)
BEGIN
    s(5 DOWNT0 1) <= GUARDED "00000";
END BLOCK ck;
END dataflow;

```



เอกสารอ้างอิง

- [1] A D&T Roundtable, "**Behavioral Description Language**," IEEE Design & Test of computers, Feb. 1990, pp 56-62.
- [2] J.R. Armstrong, "**Tuning VHDL for Multivalued Logic Modeling**," IEEE Design & Test of computers, June 1990, pp 8-10.
- [3] M. Shahdad, R. Lipsett, E. Marschner, K. Sheehan, H. Cohen, R. Waxman, D. Ackley, "**VHSIC Hardware Description Language**," Computer, Feb. 1985, pp 94-103.
- [4] A.S. Gilman, "**VHDL : The Designer Environment**," IEEE Design & Test of computers, April 1986, pp 42-47.
- [5] Zainalabedin Navabi, "**VHDL : Analysis and Modeling of Digital Systems**," McGraw-Hill Inc, 1993.
- [6] Douglas L. Perry, "**VHDL**," McGraw-Hill, Inc, 1991.
- [7] Rafael C. Gonzalez, Richard E. Woods, "**Digital Image Processing**," Addison - Wesley, 1992.
- [8] A. Rosenfeld, A.C. Kak, "**Digital Picture Processing**," 2nd, vol. 1, Academic Press, 1982.
- [9] Paul M. Embree, Bruce Kimble, "**C Language Algorithms for Digital Signal Processing**," Prentice-Hall, 1991.
- [10] B. Wilkinson, R. Makki, "**Digital System Design**," 2ed., Prentice-Hall, 1992.

ประวัติผู้เขียน

ชื่อผู้เขียน
วันเดือนปีเกิด
สถานที่เกิด
วุฒิการศึกษาระดับปริญญาตรี
สถานที่สำเร็จการศึกษา
ปีที่สำเร็จการศึกษา
ผลงานทางวิชาการที่ได้รับการตีพิมพ์

นายวิรัช เลิศบุษตราชาม
วันที่ 23 มกราคม พ.ศ. 2509
จังหวัดสงขลา
วิทยาศาสตร์บัณฑิต สาขาวิชาฟิสิกส์
มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่
ปีการศึกษา 2529
เรื่องการวิเคราะห์ตัวอักษรลายมือเขียนภาษาไทย
เรื่องการรู้จำตัวอักษรลายมือเขียนภาษาไทยโดยพิจารณา
ลักษณะเฉพาะของรูปแบบลายมือเขียน
เรื่องการออกแบบวงจรรวมเฉพาะกิจสำหรับการหา
Histogram Equalization โดยใช้ AHDL
ทุนมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร
ทุนสนับสนุนการทำวิทยานิพนธ์จากบัณฑิตวิทยาลัย
ขณะศึกษาในระดับปริญญาโท

รางวัลหรือทุนที่เคยได้รับ

