ระบบควบคุมและเฝ้าระวังกล้องผ่านระบบเครือข่ายคอมพิวเตอร์ สำหรับเทคโนโลยีรักษาความปลอดภัย

THE CAMERA IS MONITORING AND CONTROL SYSTEM THROUGH COMPUTER NETWORK FOR A SECURITY TECHNOLOGY

โดย

นายวัณรพ อาจคำพันธ์

นายวีระพงษ์ เกิดความสุข

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ระบบควบคุมและเฝ้าระวังกล้องผ่านระบบเครือข่ายคอมพิวเตอร์ สำหรับเทคโนโลยีรักษาความปลอดภัย

THE CAMERA IS MONITORING AND CONTROL SYSTEM THROUGH COMPUTER NETWORK FOR A SECURITY TECHNOLOGY

โดย

นายวัณรพ อาจคำพันธ์ 48015180

นายวีระพงษ์ เกิดความสุข 48015182

อาจารย์ที่ปรึกษา

รศ.ดร. มนัส สังวรศิลป์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบควบคุมและเฝ้าระวังกล้องผ่านระบบเครือข่ายคอมพิวเตอร์สำหรับเทคโนโลยีรักษาความปลอดภัย

The camera is monitoring and control system through a computer network for a security technology

ผู้จัดทำ 1. นายวัณรพ อาจคำพันธ์ 48015180

2. นายวีระพงษ์ เกิดความสุข 48015182

..... อาจารย์ที่ปรึกษา

(รศ.ดร.มนัส สังวรศิลป์)

ระบบควบคุมและเฝ้าระวังกล้องผ่านระบบเครือข่ายคอมพิวเตอร์สำหรับเทคโนโลยีรักษาความปลอดภัย

นาย วัณรพ อาจคำพันธ์ รหัส 48015180
นาย วีระพงษ์ เกิดความสุข รหัส 48015182
รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

โครงงานนี้ จัดทำขึ้นเพื่อพัฒนาและประยุกต์ใช้ไมโครคอนโทรลเลอร์สำหรับเทคโนโลยีระบบรักษาความปลอดภัย โดยได้ทำการออกแบบให้มีขีดความสามารถในการควบคุมการทำงานและเฝ้าระวังผ่านระบบเครือข่ายคอมพิวเตอร์ด้วยเอฟพีจีเอ ซึ่งทำให้สามารถรับสัญญาณภาพและควบคุมการทำงานของกล้องได้จากระยะไกล ณ สถานที่ต่างได้ในการควบคุมจะเป็นการสั่งให้กล้องหมุนในแนวนอนและแนวตั้ง เพื่อให้กล้องทำงานได้อย่างมีประสิทธิภาพและสะดวกในการควบคุมในงานรักษาความปลอดภัย

The camera is monitoring and control system through computer network for a security technology

Mr. Wanrop Ardkamphan ID. 48015180
Mr. Weeraphong Koetkhwamsuk ID. 48015182
Assoc. Prof. Dr.Manas Sangworasil Advisor
Educational Year 2007

Abstract

The thesis, We present to apply a knowledge base about a microcontroller chip for a security technology. To improve are monitoring and control system through a computer network via our programming on FPGA. This method, we can recciving a video data and sending command to control a camera from a long distance anywhere, Such as control both vertical and horizontal angle. Summarized, We will get a goof performance camera that easy to controlling. That is most importance for security job.

กิตติกรรมประกาศ

ปริญญานิพนธ์เล่มนี้สำเร็จลุล่วงไปได้ด้วยดี เนื่องจากความร่วมมือของสมาชิกภายในกลุ่ม และผู้มีส่วนร่วมทุกท่าน

ขอขอบพระคุณ รศ.ดร.มนัส สังวรศิลป์ ซึ่งเป็นอาจารย์ที่ปรึกษา และอาจารย์ภาควิชาอิเล็กทรอนิกส์ทุกท่านที่คอยให้คำแนะปรึกษา แนวความคิด และแนวทางแก้ไขปัญหา รวมทั้งยังอบรมสั่งสอนให้ได้รับความรู้ในเรื่องต่างๆ ในการจัดทำโครงงาน

ขอขอบคุณพี่สมโชค พี่กิตติ พี่หมู และพี่ๆทุกคน ที่คอยช่วยเหลือให้คำปรึกษา ขอขอบคุณเจ้าหน้าที่ประจำสำนักวิจัยและบริการคอมพิวเตอร์ที่อำนวยความสะดวกในการใช้เครื่องมือและห้องปฏิบัติการ

ขอขอบคุณเพื่อนๆทุกคนที่ให้กำลังใจและให้ความช่วยเหลือพึ่งพาอาศัยกันและกัน

และสุดท้ายนี้ ขอกราบขอบพระคุณบิดามารดาผู้ซึ่งให้การอุปการะในการเรียน ทั้งคอยดูแลและเป็นกำลังใจด้วยดีตลอดมา

ผู้จัดทำ

นายวัณรพ อาจคำพันธ์

นายวีระพงษ์ เกิดความสุข

สารบัญ

สารบัญ (ต่อ)

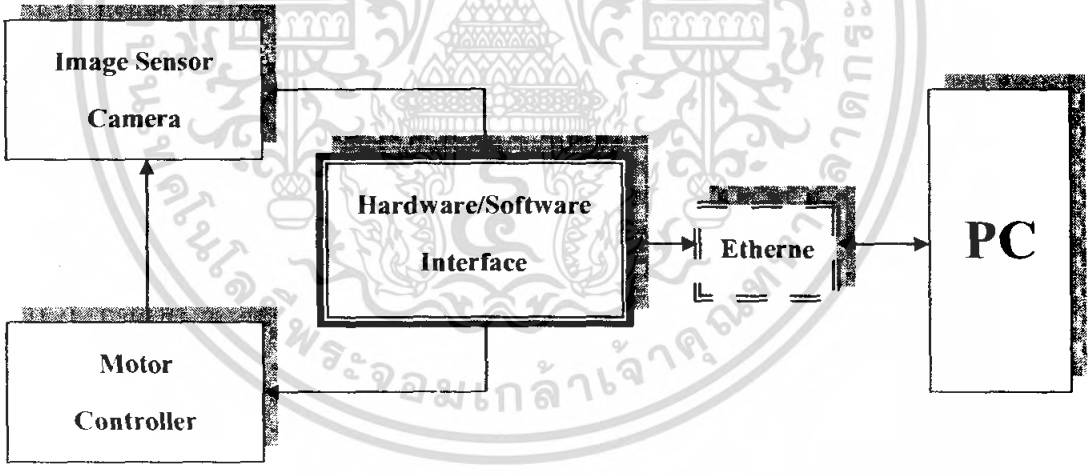
สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 แนวความคิดเริ่มต้น

ในปัจจุบันนี้ เทคโนโลยีด้านระบบรักษาความปลอดภัยเป็นสิ่งที่มีบทบาทสำคัญอย่างยิ่งในชีวิตประจำวัน ไม่ว่าจะเป็นบริษัท โรงงาน หรือสถานที่ที่กว้างใหญ่ ซึ่งในการใช้คนตรวจตราดูแลรักษาความปลอดภัยที่อาจเกิดจากการโจรกรรมหรืออุปกรณ์เครื่องมือที่มีใช้อยู่ภายในนั้นอาจจะไม่ครอบคลุมทั่วทุกพื้นที่หรือมีประสิทธิภาพไม่เพียงพอ ดังนั้น จึงได้มีการนำเทคโนโลยีที่เกี่ยวกับระบบรักษาความปลอดภัยมาประยุกต์ใช้ร่วมกับมนุษย์ เพื่อให้การทำงานมีประสิทธิภาพเพิ่มมากขึ้น ซึ่งก็ได้มีการพัฒนากันระบบดังกล่าวกันมาอย่างต่อเนื่อง โดยเฉพาะกับการประยุกต์ใช้งานร่วมกับระบบอินเตอร์เน็ตและเครือข่ายคอมพิวเตอร์ที่มีใช้กันอย่างแพร่หลาย เพื่อให้มีความสะดวกในการควบคุมหรือสั่งงาน



รูปที่ 1.1 กระบวนการทำงานของระบบการทำงาน

ด้วยเหตุนี้ ผู้จัดทำจึงได้ทำการออกแบบและพัฒนาโครงงานนี้ตามแนวความคิดดังกล่าว โดยการทดลองใช้ FPGA มาทำหน้าที่เป็นตัวกลางในการรับ-ส่งสัญญาณภาพจากกล้องอิมเมจเซ็นต์เซอร์และแปลงข้อมูลสัญญาณภาพเข้าสู่ระบบเครือข่ายคอมพิวเตอร์ โดยโครงงานนี้จะเป็นการนำเสนอรายละเอียดต่างๆของ FPGA ทั้งโครงสร้างทางฮาร์ดแวร์และซอร์ฟแวร์ ร่วมกับอุปกรณ์ต่างๆที่ใช้งานในระบบ

1.2 วัตถุประสงค์

- เพื่อศึกษาและเรียนรู้การใช้งาน FPGA
- เพื่อศึกษาการเขียนโปรแกรมควบคุมการทำงาน FPGA ด้วยภาษา VHDL
- เพื่อเป็นการนำเทคโนโลยีมาประยุกต์ใช้งานร่วมกับระบบรักษาความปลอดภัย

1.3 ขอบเขตของโครงงาน

โครงงานนี้เป็นการใช้ FPGA ทำหน้าที่เป็นตัวกลางในการรับส่งสัญญาณภาพจากกล้อง Image Sensor และสามารถควบคุมการทำงานของมอเตอร์เพื่อหมุนตำแหน่งของกล้อง โดยเชื่อมต่อผ่านระบบเครือข่ายคอมพิวเตอร์เพื่อเพิ่มความสะดวกในการใช้งาน โดยคอมพิวเตอร์สามารถเชื่อมต่อกับชุดควบคุมและสัญญาณภาพที่บอร์ดคอนโทรลของ FPGA ได้โดยตรงผ่านระบบเครือข่าย

1.4 ประโยชน์และผลที่คาดว่าจะได้รับ

- นักศึกษาผู้จัดทำโครงงานมีความรู้ความเข้าใจในระบบการทำงานของ FPGA และเครือข่ายคอมพิวเตอร์เพิ่มมากขึ้น
- ส่งเสริมให้นักศึกษาผู้จัดทำโครงงานมีความคิดริเริ่มสร้างสรรค์ เกิดทักษะ และสามารถแก้ปัญหาต่างๆภายในการทำงานได้
- สามารถประยุกต์ใช้ความก้าวหน้าทางเทคโนโลยีใหม่ๆมาใช้ให้เกิดประโยชน์ และนำโครงงานดังกล่าวนี้ไปใช้งานได้จริง

1.5 เนื้อหาโดยสังเขป

บทที่ 1 กล่าวถึงแนวความคิดเริ่มต้น วัตถุประสงค์ ขอบเขต และประโยชน์ของโครงงาน

บทที่ 2 กล่าวถึงทฤษฎีและหลักการที่นำมาใช้ในการทำโครงงาน ได้แก่ กล้องอิมเมจเซ็นเซอร์ FPGA ระบบเครือข่าย และวิธีการออกแบบระบบอิเล็กทรอนิกส์

บทที่ 3 กล่าวถึงการออกแบบและการสร้างโครงงานทั้งส่วนของฮาร์ดแวร์และซอร์ฟแวร์ ได้แก่ บอร์ดคอนโทรล วงจรขับมอเตอร์ และแผนผังการทำงานของระบบ

บทที่ 4 กล่าวถึงวิธีการทดลองและผลการทดลอง ในการทดสอบการทำงานของโครงงาน

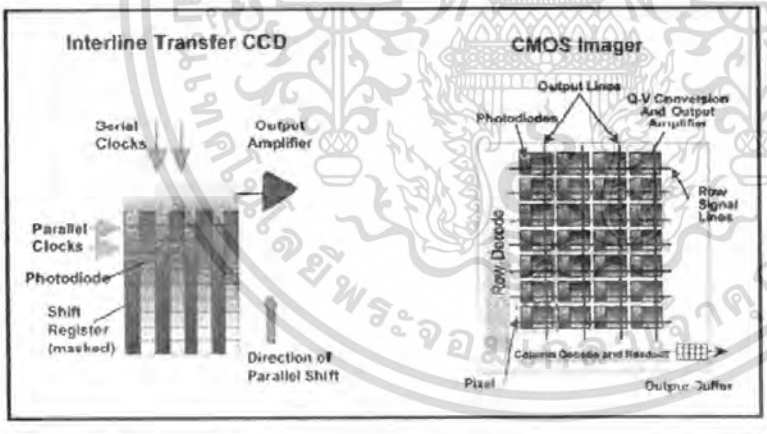
บทที่ 5 กล่าวถึงบทสรุปในการจัดทำโครงงาน และปัญหาหรืออุปสรรคที่เกิดขึ้น รวมทั้งแนวทางในการแก้ไขปัญหา

บทที่ 2
ทฤษฎี และหลักการ

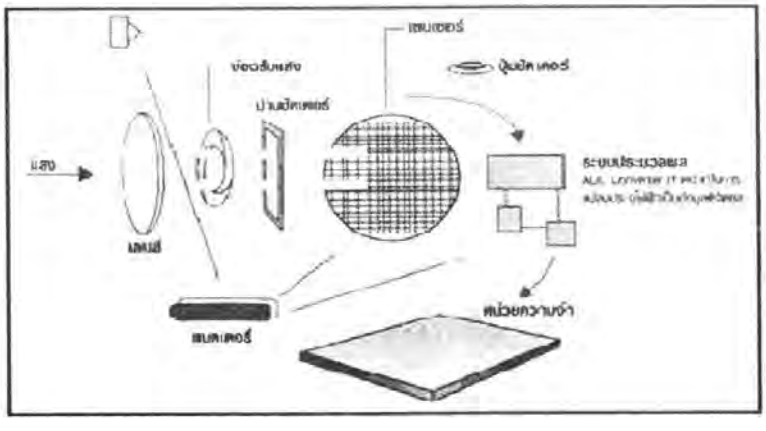
2.1 กล้องอิมเมจ เซ็นเซอร์ (Image Sensor Camera)

กล้องที่มีอยู่จำนวนมากในปัจจุบันจะพบว่า ตัวเซ็นเซอร์สำหรับการรับภาพในกล้องนั้นที่พบอยู่จะมีเป็นสองลักษณะด้วยกัน คือ CCD Sensor และ CMOS Sensor ตัว Image Sensor ทั้งสองชนิดนี้ต่างก็เป็น metal oxide Semiconductors ที่มีคุณสมบัติ คือ สามารถใช้เป็นตัวรับแสงและแปลงอนุภาคของแสงให้เป็นสัญญาณไฟฟ้าได้

ซึ่งทั้ง CCD และ CMOS อาศัยหลักการทำงานเดียวกัน โดยอาศัย Photo site ให้เปลี่ยนแสงที่มาตกกระทบให้กลายเป็นอิเล็กตรอน และบ่งบอกค่าสีของแสงที่มาตกกระทบ โดยทั้ง CMOS และ CCD จะประกอบไปด้วย Photo site ขนาดเล็กนับล้านชิ้น เพื่อทำหน้าที่ในการรับแสงที่มาตกกระทบ สำหรับความแตกต่างของเซนเซอร์รับภาพทั้ง 2 ชนิด คือพื้นฐานการสร้างการออกแบบที่แตกต่างกัน ซึ่งทำให้ความสามารถในการใช้งานแตกต่างกันไปด้วย



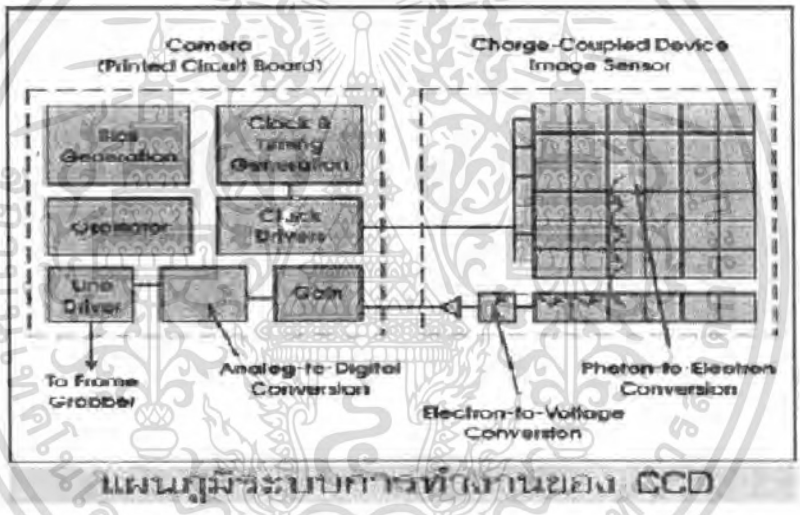
รูปที่ 2.1
เซ็นเซอร์ของ
CCD และ CMOS



รูปที่ 2.2
กระบวนการการ
รับสัญญาณแสงของ Sensor

2.1.1 CCD (Charge Coupled Device)

CCD เป็นเทคโนโลยีที่ใหม่กว่า สามารถถ่ายภาพได้คุณภาพที่ดีกว่า แต่ก็มีต้นทุนที่สูงกว่าเช่นกัน มักนำมาใช้กับ โทรศัพท์มือถือ ราคาแพง ในการคำนวณค่าของแสงที่มาตกกระทบที่แต่ละ Photo site จะมีการประจุค่านั้นโดยตรง เช่นเดียวกับ CMOS และจะแปลงค่าแสงที่เป็น Analog ให้เป็นแบบ Digital ซึ่งกระบวนการเหล่านี้เกิดขึ้นอย่างรวดเร็ว และส่งประจุได้โดยตรงไปยัง Chip โดยไม่เกิดการตัดทอนสัญญาณ หรือสิ่งที่รบกวนสัญญาณภาพ ซึ่งเกิดจากเทคโนโลยีกระบวนการผลิตขั้นสูง เพื่อให้เซนเซอร์มีคุณภาพ และไวต่อแสงที่มาตกกระทบ ซึ่งทำให้คุณภาพของภาพถ่ายที่ได้ดีกว่า CMOS แต่ CCD ก็ยังต้องใช้พลังงานมากกว่า CMOS อยู่ และการผลิตต้องใช้แผ่นซิลิคอนแบบพิเศษที่ผลิตขึ้นมาโดยเฉพาะ จึงทำให้ต้นทุนในการผลิตนั้นสูงกว่า CMOS ไปด้วย



รูปที่ 2.3 แผนภูมิระบบการทำงานของ CCD

ใน CCD Sensor จะทำหน้าที่รับแสงแล้วแปลงอนุภาคแสงเป็นพลังงานไฟฟ้าแบบ Analog ส่งผ่านออกจากแผง CCD ออกมาเข้าสู่แผงวงจรอีกวงจรหนึ่ง (Camera Printed Circuit Board) แผงวงจรนี้จะทำหน้าที่แปลงสัญญาณจากสัญญาณ Analog เป็นสัญญาณ Digital แล้วเข้าสู่ขบวนการจัดการเรื่องสีภาพของกล้องก่อนที่จะบันทึกลงในตัวกลางสำหรับบันทึกภาพต่อไป ดังนั้นด้วยเหตุที่ว่าตัวเซ็นเซอร์รับภาพมีเฉพาะตัวรับแสงและวงจรส่งผ่านไฟฟ้าเพียงอย่างเดียว จึงทำให้สามารถที่จะสร้างให้แต่ละพิกเซลมีพื้นที่ในการรับแสงได้เต็มที่โดยเฉพาะอย่างยิ่ง CCD ในระบบ Full Frame ที่แต่ละพิกเซลมีพื้นที่เกือบเต็มก็สามารถที่จะได้รับอนุภาคของแสงได้มากและมีความสม่ำเสมอของสัญญาณที่ดี แต่กล้องที่ใช้ตัวรับภาพแบบ CCD ก็จะต้องประกอบไปด้วยแผง CCD และแผงวงจรในการแปลงสัญญาณเป็น 2 ชุดอยู่ในกล้อง ซึ่งทำให้การออกแบบกล้องมีความ

ซับซ้อน แต่ก็มีข้อดีที่การออกแบบนั้นสามารถแยกได้ทั้งการออกแบบแผงวงจร และแผง CCD รับภาพ ถ้าต้องการเปลี่ยนแปลงตัวใดตัวหนึ่งก็ย่อมทำได้โดยอิสระทั้งสองส่วน

2.1.2 CMOS (Complementary Metal Oxide Semiconductor)

CMOS นิยมใช้กับ โทรศัพท์มือถือ ราคาประหยัดถึงปานกลาง ซึ่งเป็น ในการคำนวณค่าของแสงที่มาตกกระทบที่แต่ละ Photo site จะมีการประจุค่านั้นโดยตรง เช่นเดียวกับ CCD แต่การส่งผ่านข้อมูลต้องอาศัยสายข้อมูลขนาดเล็ก ไปทำการประมวลผลอีกทอดหนึ่ง อาศัยเทคโนโลยีการผลิตแบบเก่า ซึ่งเป็นแบบเดียวกับการผลิต Microprocessor จึงเต็มไปด้วยสัญญาณรบกวน และมีผลทำให้เกิดการลดทอนคุณภาพของภาพถ่าย ตัวเซนเซอร์เองมีความผิดพลาดในการส่งข้อมูลสูง มีความไวต่อแสงน้อย จึงมีผลทำให้คุณภาพของภาพถ่ายที่ได้ด้อยกว่า CCD ส่วนการผลิตสามารถทำได้บนแผ่นซิลิคอนมาตรฐานทั่วไป จึงทำให้ต้นทุนในการผลิตต่ำกว่า CCD

ใน CMOS Sensor นอกเหนือจากจะประกอบได้ด้วยพิกเซลที่ทำหน้าที่รับแสงอยู่ภายในแผง CMOS เพื่อทำการเปลี่ยนเป็นกระแสไฟฟ้าแบบ Analog แล้ว ยังประกอบไปด้วยวงจรที่ซับซ้อนเพื่อทำหน้าที่แปลงสัญญาณไฟฟ้าแบบ Analog ให้เป็นสัญญาณ Digital อยู่ในแผง CMOS เดียวกันเลย ซึ่งการแปลงสัญญาณนี้จะทำตั้งแต่ในระดับแต่ละพิกเซลเลยทีเดียว หรือจะกล่าวแบบเข้าใจง่ายๆ ก็คือ CMOS Sensor ที่นำมาใช้งานในกล้องดิจิตอลนั้น เมื่อได้รับแสงแล้วมีความสามารถที่จะให้สัญญาณภาพเป็นระบบดิจิตอลในเวลาเดียวกันเมื่อสัญญาณถูกส่งต่อเข้ากับระบบของกล้อง ซึ่งถ้าเทียบกับ CCD Sensor แล้ว CCD จะต้องใช้แผงวงจรต่างหากเพื่อทำหน้าที่แปลงสัญญาณเป็นดิจิตอลอีกครั้งหนึ่ง

รูปที่ 2.4 แผนภูมิระบบการทำงานของ CMOS

แต่ด้วยวงจรไฟฟ้าที่มีความสลับซับซ้อนที่มากกว่าของ CMOS จึงทำให้ไปเบียดพื้นที่ส่วนการรับแสงในแต่ละพิกเซลของ CMOS จึงทำให้พื้นที่ในการรับแสงของ CMOS มีขนาดที่เล็กลงเมื่อเทียบในระดับของขนาดพิกเซลที่เท่าๆ กัน อีกประการหนึ่งการออกแบบสำหรับการทำงานของชั้นวงจรต่างๆ จัดเป็นชั้นๆ ดังนั้นจึงทำให้การออกแบบสำหรับส่วนรับแสงให้เต็มพื้นที่ทำไม่ได้ (ทำไม่ได้ในขณะนี้อนาคตอาจจะทำได้ก็ไม่แน่) และด้วยความซับซ้อนของการทำงานที่มากกว่านี้เองจึงมีผลทำให้ความสม่ำเสมอของสัญญาณที่ส่งออกมาจากแผง CMOS ต่ำลงซึ่งเมื่อเทียบกับ CCD Sensor จะให้ความสม่ำเสมอของสัญญาณที่ดีกว่า

อีกประการหนึ่งการเปลี่ยนแปลงชุดรับแสงนั้น ถ้าต้องการเปลี่ยนแปลงระบบแปลงสัญญาณหรือส่วนในการรับแสงอย่างใดอย่างหนึ่งนั้นไม่สามารถกระทำเพียงอย่างใดอย่างหนึ่งเนื่องจากถูกออกแบบมาในระดับพิกเซลรวมไว้ในยูนิตเดียวกัน ถ้าต้องการเปลี่ยนแปลงนั่นหมายถึงว่าต้องเป็นการออกแบบใหม่ทั้งแผงเลยทีเดียว แต่อย่างไรก็ตามในด้านการออกแบบอย่าง CMOS ที่รวมไว้ในยูนิตเดียวทั้งหมดก็มีข้อดีเบื้องต้นก็คือ กินกำลังไฟหรือสิ้นเปลืองพลังงานน้อยกว่า CCD

2.1.3 ข้อเปรียบเทียบระหว่าง CCD และ CMOS

การตอบสนองการใช้งาน : CMOS Sensor ให้การตอบสนองการใช้งานที่ได้เปรียบกว่าและรวดเร็วกว่า CCD Sensor เพราะเหตุว่าการให้สัญญาณออกจากตัวรับแสงเป็นสัญญานดิจิตอลโดยตรงไม่ต้องแปลงสัญญาณอีกครั้งหนึ่งรวมทั้งภาคขยายสัญญาณอยู่ในตัวทำให้กินพลังงานที่ต่ำกว่า ซึ่งถ้าเป็นระบบ CCD Sensor จำเป็นต้องมีวงจรสำหรับการขยายสัญญาณแปลงสัญญาณอีกชุดหนึ่งทำให้สิ้นเปลืองพลังงานมากกว่า อีกประการหนึ่งก็คือ การออกแบบภายในกล้องซึ่ง CMOS Sensor เพียงแต่มีเลนส์และระบบจัดการก็สามารถออกแบบได้กะทัดรัดแล้วในขณะที่ ระบบ CCD Sensor ต้องการพื้นที่ภายในกล้องสำหรับใส่แผ่นวงจรเพิ่มอีกหนึ่งชุดซึ่งทำให้การออกแบบนั้นมีความยุ่งยากมากกว่า

Dynamic Range : ความสามารถช่วงกว้างในการรับแสงเพื่อเก็บรายละเอียดของบริเวณมืดและสว่างของภาพจะพบว่า การที่ระบบ CCD Sensor ทำหน้าที่เพียงการรับสัญญาณแสงแล้วส่งผ่านออกไปจากตัวรับภาพโดยตรงเพื่อไปเข้าวงจรสำหรับการแปลงสัญญาณอีกครั้งหนึ่ง ทำให้ได้สัญญาณที่มีความสม่ำเสมอสูงกว่า ระบบ CMOS Sensor อีกทั้งไม่มีวงจรการแปลงสัญญาณที่ยุ่งยากภายในตัวรับแสงเองที่ก่อให้เกิดความร้อนที่ Sensor รับภาพน้อยกว่าของ CMOS Sensor ดังนั้น ช่วงกว้างในการเก็บรายละเอียดของ เซ็นเซอร์ที่เป็น CCD จึงทำได้ดีกว่าเซ็นเซอร์ที่เป็น CMOS และ Noise หรือสัญญาณรบกวนของ CCD ก็มีต่ำกว่าของ CMOS

Shuttering : การตอบสนองต่อการเปิดปิดการรับแสง จะพบได้ว่าถ้าตัวรับภาพเป็นแบบ CCD ซึ่งทำหน้าที่รับแสงแปลงเป็นสัญญาณไฟฟ้าส่งผ่านออกมาแต่เพียงอย่างเดียว จะสามารถตอบสนองการเปิดปิดรับแสงแล้วให้สัญญาณที่มีความเป็น Uniformity ได้ดีกว่าตัวรับภาพที่เป็น CMOS ซึ่งมีระบบการแปลงสัญญาณที่ซับซ้อนกว่าเพื่อส่งสัญญาณที่เป็นดิจิตอลออกจากตัว CMOS มีโอกาสที่สัญญาณภาพจะเกิดความเป็น Nonuniformity มากกว่าใน CCD แน่นอนว่าปัญหานี้ทำให้ได้สัญญาณที่เป็นรอง CCD ซึ่งผู้ผลิต CMOS ก็ได้พยายามพัฒนาวงจรอิเลคโทรนิคอื่นๆ มาสนับสนุนให้ดีขึ้น ซึ่งก็สามารถที่จะแก้ปัญหาส่วนนี้ได้ระดับหนึ่งแต่ก็จะมีปัญหาในเรื่องต้นทุนการผลิตที่สูงขึ้นเช่นกัน ดังนั้นสำหรับในกล้องระดับล่างๆ ส่วนใหญ่ก็จะไม่ได้รับการแก้ปัญหาดังกล่าว ยกเว้นแต่จะเป็นกล้องในระดับที่สูงขึ้นเท่านั้น

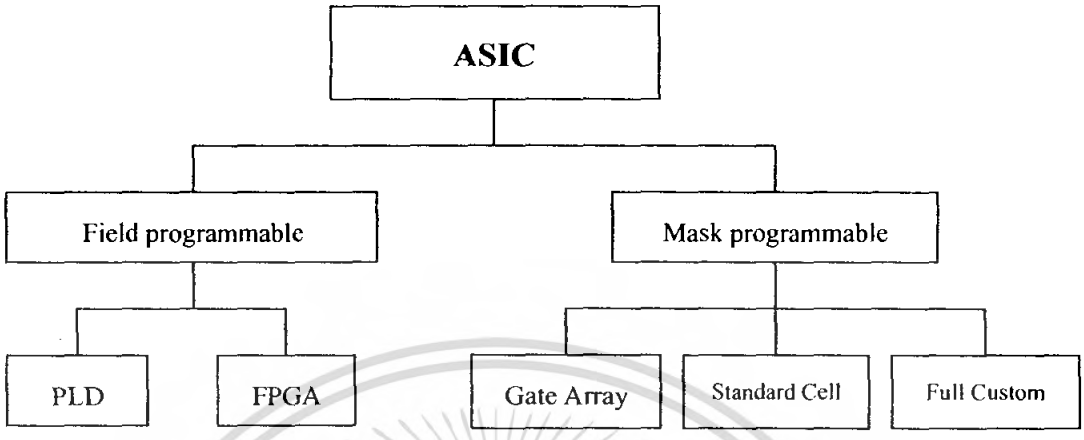
Windowing : จุดเด่นอย่างหนึ่งของ CMOS Sensor ก็คือความสามารถในการส่งผ่านข้อมูลที่รวดเร็วในระบบปฏิบัติการ เนื่องจากเป็นการส่งสัญญาณที่เป็นดิจิตอลโดยตรงออกจากเซ็นเซอร์รับภาพ จึงทำให้ CMOS Sensor มีความสามารถในการจับภาพวัตถุในพื้นที่จำกัดได้น่าสนใจกว่าทั้งยังสามารถที่จะพัฒนาต่อเนื่องได้ง่ายกว่า CCD Sensor ที่มีข้อจำกัดที่มากกว่าในการพัฒนา

Antiblooming : การขยายวงของแสงที่ล้นกรอบ Pixel ที่รับแสงเมื่อได้รับแสงมากเกินไป ส่วนนี้จะทำให้ภาพที่ถ่ายโดยการรับแสงมากเกินไปมีแสงฟุ้งที่ขอบวัตถุ ซึ่งโดยธรรมชาติแล้วตัวรับแสงแบบ CMOS จะปรากฏอาการมากกว่า CCD ซึ่งเช่นเดียวกันจำเป็นต้องการการออกแบบทางวิศวกรรมเพิ่มเติมอีกเพื่อลดปัญหานี้ ซึ่งใน CCD จะมีต่ำกว่า

2.2 Field Programmable Gate Array

2.2.1 การผลิตวงจรรวม

ความก้าวหน้าของอุตสาหกรรมอิเล็คทรอนิกส์ปัจจุบันทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆ มากมายซึ่งทำให้เกิดการลดค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโพรเซสเซอร์และหน่วยความจำปัจจุบัน ทุกๆ ครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวมและไอซีมาตรฐานมากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน ฟังก์ชันลอจิก ที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตให้ขนาดมากๆ และการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวม จะแบ่งตามการสร้างออกเป็น 2 กลุ่ม คือ Field programmable และ Mask programmable



รูปที่ 2.5 ผังแสดงการแบ่งกลุ่มของวงจรรวม ASIC

Programmable Logic สามารถแบ่งได้เป็นหลายชนิด โดยแบ่งตามเทคโนโลยีที่ใช้ผลิต ความซับซ้อนของอุปกรณ์ หรือโครงสร้างภายในอุปกรณ์เอง ซึ่งสามารถแบ่งได้ดังนี้

- Simple Programmable Logic Device (SPLD)
- Complex Programmable Logic Device (CPLD)
- Field Programmable Gate Array (FPGA)
- Field Programmable Interconnects (FPIC)

1) Simple Programmable Logic Device (SPLD)

SPLD เป็นอุปกรณ์ไอซีที่โปรแกรมได้ที่มีโครงสร้างแบบง่ายมีขนาดเล็กโดยมีจำนวนมาโครเซลล์ไม่เกิน 30 มาโครเซลล์

2) Complex Programmable Logic Device (CPLD)

CPLD แบบ EPROM มักจะสามารถโปรแกรมได้เพียงครั้งเดียว (OTP) นอกเสียว่าจะมีช่อง (window) สำหรับใช้ลบข้อมูลด้วยแสงอุลตร้าไวโอเล็ต (UV) ในการโปรแกรม EPLD จะต้องใช้เครื่องโปรแกรมไอซีเฉพาะที่เรียกทั่วไปว่า IC programmer ใน CPLD รุ่นใหม่ๆมักจะเป็นแบบ EEPROM หรือ FLASH เนื่องจากสามารถโปรแกรม ได้ โดยตรงในภาคสนามได้ (in-System Programmable)

3) Field Programmable Gate Array (FPGA)

FPGA มีโครงสร้างที่แตกต่างไปจาก SPLD และ CPLD และมักจะมีความหนาแน่นของเกตมากกว่า โครงสร้างของ FPGA ประกอบด้วย แผงของบล็อกลอจิก ล้อมลอบด้วยส่วน I/Oที่สามารถโปรแกรมได้ และมีการเชื่อมต่อแบบโปรแกรมได้ เช่นกัน FPGA มีขนาดตั้งแต่ระดับ1000 เกต จนถึงหลายล้านเกต

4) Field Programmable Interconnects (FPIC)

FPIC จริงๆ แล้วไม่ได้ไอซีทางด้านลอจิก แต่จะเป็นอุปกรณ์ที่สามารถโปรแกรมส่วนที่เชื่อมต่อ (wiring) เช่นการเชื่อมตัวให้ไอซีหนึ่งตัวสามารถเลือกได้ว่าให้เชื่อมต่อเข้ากับไอซีตัวไหนได้ บริษัทที่ผลิตอุปกรณ์ประเภทนี้ได้แก่ ไอซี Digital Cross paint Switch ของบริษัท I-Cube Digital cross point device ของบริษัท Lattice หรือ อุปกรณ์ Hardware emulator ของบริษัท Aptix

2.2.2 Field Programmable Gate Array (FPGA)

FPGA จัดเป็น อุปกรณ์สารกึ่งตัวนำชนิดโปรแกรมได้ที่มีโครงข่ายการเชื่อมต่อภายในแบบแมตริกซ์ โครงสร้างภายในของ FPGA นั้นสามารถโปรแกรมให้มีหน้าที่การทำงานเหมือนลอจิกเกตพื้นฐาน เช่น AND, OR, XOR, NOT หรือรวมกันหลายๆ ชนิด (combinational logic) เพื่อให้ทำหน้าที่ที่มีความซับซ้อนเพิ่มขึ้น เช่น decoders หรือฟังก์ชันทางคณิตศาสตร์ ใน FPGA ทั่วไป นอกจากจะประกอบด้วยส่วนของวงจรลอจิกแบบโปรแกรมได้แล้ว จะยังมีบล็อกของหน่วยความจำ ซึ่งอาจจะสร้างด้วยฟลิบฟลอปอย่างง่าย หรือใช้พื้นที่ของสารกึ่งตัวนำสร้างเป็นหน่วยความจริงๆอยู่ภายในก็ได้

ในการออกแบบวงจรดิจิตอลอิเล็กทรอนิกส์ ที่มี FPGA อยู่บนแผงวงจรด้วยนั้น จะช่วยให้ผู้ออกแบบสามารถลดขนาดของแผงวงจร รวมทั้งสามารถออกแบบได้รวดเร็ว ไม่ต้องทดสอบรายละเอียดภายในให้เสร็จสมบูรณ์ 100 % ก็สามารถออกแบบแผงวงจรได้ เมื่อได้รับแผงวงจรและประกอบอุปกรณ์ต่างๆ เสร็จแล้ว จึงค่อยกำหนดหน้าที่การทำงานของ FPGA ได้ในภายหลัง ต่างจากการออกแบบด้วยลอจิกเกตขนาดเล็ก ที่ต้องออกแบบทางเดินของลายทองแดงให้เสร็จสมบูรณ์ก่อน และไม่สามารถแก้ไขได้ในภายหลัง นอกจากนี้ การใช้งาน FPGA สามารถโปรแกรมการทำงานได้ในทุกขณะแม้แต่ขณะที่ส่งมอบงานแล้ว ก็ยังสามารถเข้าไปแก้ไขวงจรได้โดยง่ายดาย จึงเป็นที่มาของคำว่า "field programmable" ซึ่งก็หมายถึงโปรแกรมได้ในภาคสนามหรือที่หน้างานนั่นเอง อย่างไรก็ตามข้อกำหนด (Configuration) ของ FPGA จะหายไปหลังจากปิดไฟเลี้ยง ดังนั้นจะต้องมีหน่วยความจำภายนอก (Flash) มาคอยรักษาข้อกำหนดของ FPGA ไว้ ซึ่ง FPGA จะมีกระบวนการอ่านข้อกำหนดนั้นโดยอัตโนมัติหลังจากได้รับไฟเลี้ยง

รูปที่ 2.6 ไอซี FPGA เบอร์ XC3S250E-4PQ208C

การทำงานของ FPGA จะยังมีความเร็วที่ด้อยกว่า application-specific integrated circuit (ASIC) และเมื่อเปรียบเทียบขนาดทางกายภาพ พบว่าจะมีความหนาแน่นของวงจรที่น้อยกว่า รวมทั้งใช้กำลังงานมากกว่า ASIC อย่างไรก็ตาม FPGA มีข้อได้เปรียบตรงที่ใช้เวลาในการพัฒนาผลิตภัณฑ์ (time to market) ที่น้อยกว่า สามารถแก้ไขวงจรได้หลังจากที่ใช้งานจริงในภาคสนาม และมีค่าแรงในการดำเนินการที่ต่ำกว่า (non-recurring engineering) นอกจากนี้ ยังมี FPGA ชนิดที่โปรแกรมได้ครั้งเดียว (OTP) ซึ่งมีราคาที่ต่ำกว่าโดย FPGA ชนิดนี้เมื่อโปรแกรมแล้วจะคล้ายกับ ASIC นอกจากนี้ยังมีการรวมหน่วยความจำ config เข้าไว้ในอุปกรณ์ FPGA ซึ่งจะยังคงอยู่แม้ปิดไฟเลี้ยง เรียกว่า Complex programmable logic devices (CPLD)

2.2.3 โครงสร้างสถาปัตยกรรมภายใน FPGA

การแบ่งหมวดของ FPGA นั้นสามารถแบ่งแยกตามคุณลักษณะ ได้ดังนี้

- โครงสร้างสถาปัตยกรรมภายใน
- เทคโนโลยีการโปรแกรม
- I/O
- Interconnect

FPGA สามารถแบ่งตามโครงสร้างสถาปัตยกรรมภายในของอุปกรณ์ ได้ดังนี้

- LCA (Logic Cell Array)
- PASIC (Programmable ASIC)
- FLEX, APEX
- ACT (Actel)

การแบ่งโครงสร้างสถาปัตยกรรมของ FPGA แบ่งได้เป็น 2 ลักษณะ คือ

1. Coarse-grained โครงสร้างสถาปัตยกรรมแบบ Coarse-grained จะประกอบด้วยลอจิกบล็อกขนาดใหญ่ เช่นมักประกอบด้วย LUT และ Flip-Flops 2 อัน หรือมากกว่า ตัวอย่าง FPGA แบบนี้ได้แก่ Xilinx ตระกูล XC4K, Spartan, Virtex หรือ Altera ตระกูล FLEX, APEX

2. Fine-grained ประกอบด้วย ลอจิกบล็อกแบบง่าย จำนวนมาก ลอจิกบล็อกประกอบด้วยลอจิกฟังก์ชัน หรือ 2 อินพุท หรือมัลติเพล็กเซอร์แบบ 4-1 และฟลิปฟล็อป ตัวอย่าง FPGA แบบ fine-grained เช่น ตระกูล ACT ของบริษัท ACTEL

**ข้อแตกต่างอื่นๆ ของโครสร้างสถาปัตยกรรมของ FPGA คือ เทคโนโลยีที่ใช้ในการผลิตตัวชิป **

2.2.4 การเขียนภาษาอธิบายลักษณะพฤติกรรมของฮาร์ดแวร์

การออกแบบวงจรดิจิตอลด้วย FPGA โดยทั่วไปมีองค์ประกอบ 3 ส่วน

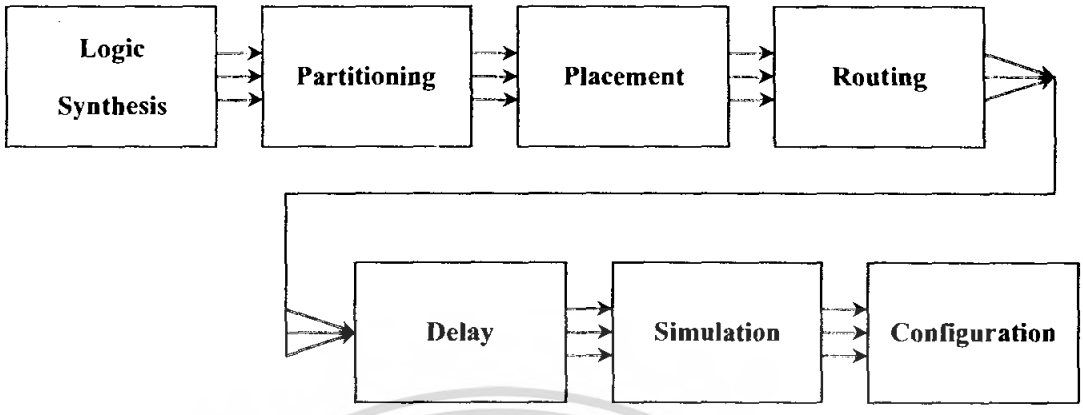
1) ซอฟต์แวร์ Design Entry

- โดยใช้ Schematic Design Entry ใช้ไลบารีของ FPGA
- ใช้ภาษา HDL เช่น VHDL, AHDL, Verilog

2) Design Implementation เป็นการแปลงแบบที่ได้ออกแบบจาก Schematic หรือ HDL ให้เป็นลอจิก ซึ่งอาจใช้ซอร์ฟแวร์สำหรับสังเคราะห์วงจร (Logic synthesis)

3) Device Programming เป็นการโปรแกรมอุปกรณ์หรือชิป FPGA

สำหรับการออกแบบวงจรด้วย FPGA การใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำ ได้สะดวกและเข้าใจได้ง่ายกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญเหมือนกับ ASIC การออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้



รูปที่ 2.7 ขั้นตอนการออกแบบภาษาอธิบายฮาร์ดแวร์

1) การสังเคราะห์วงจร (Logic Synthesis) ขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรม ของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ และในขั้นตอนนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดล แต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำ การสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้น เป็นอย่างไร เช่นมีค่าความหน่วง (Delay) หรือการใช้ทรัพยากรต่างๆ ใน FPGA

2) การแบ่งวงจร (Partitioning) ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์เป็นส่วนย่อยๆ สำหรับลงใน CLB, IOB หรือองค์ ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำโดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิป-ฟลอบ (flip-flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ หลังจากทำขั้นตอนนี้เสร็จแล้วจะทราบว่าการใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ส่วนข้อมูลทางเวลานั้นจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความ หน่วงลอจิก(logic delay) ส่วนซอฟต์แวร์จะรวมเอาซอฟต์แวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement & Routing) เป็นไปอย่างต่อเนื่อง

3) การวางอุปกรณ์ (Placement) ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควรจะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุดและช่วยลดความหน่วงของเวลา ซึ่งการวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กันโดยเฉพาะส่วนที่มีการเชื่อมต่อสัญญาณด้วยกัน นอกจากนั้นการกำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ควรกำหนดตำแหน่งให้เหมาะสม หรือ ไม่ก็ให้ซอฟต์แวร์จัดการเอง

4) การเชื่อมต่อสัญญาณ (Routing) ขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ต่อเนื่องจากการวางอุปกรณ์ ซึ่งสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือจะทำการเชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับทางเวลา จะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

5) ความหน่วงด้านเวลา (Delay) ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์โดยไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วงที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ

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

โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก

6) การจำลองการทำงานของวงจร (Simulation) ขั้นตอนนี้เป็นขั้นตอนที่สำคัญ เพราะเป็นขั้นตอนตรวจสอบฟังก์ชันการทำงานของโมเดลขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรอยู่ ในการจำลองการทำงานของวงจรควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จ นั่นคือต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR

ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้ในตอนนี้จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้ว ยังต้องตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวม สามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

7) การโปรแกรมอุปกรณ์ FPGA (Configuration) หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้ ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับ อุปกรณ์ FPGA ของแต่ละบริษัทผู้ผลิต

2.2.5 เทคโนโลยีการโปรแกรม

เทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบ คือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และ การโปรแกรมโดยการใช้หน่วยความจำ

สำหรับเทคนิคการโปรแกรมไอซี FPGA นั้นมีดังนี้

1) Fuse : เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียว ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อม ต่อจะขาดจากกัน

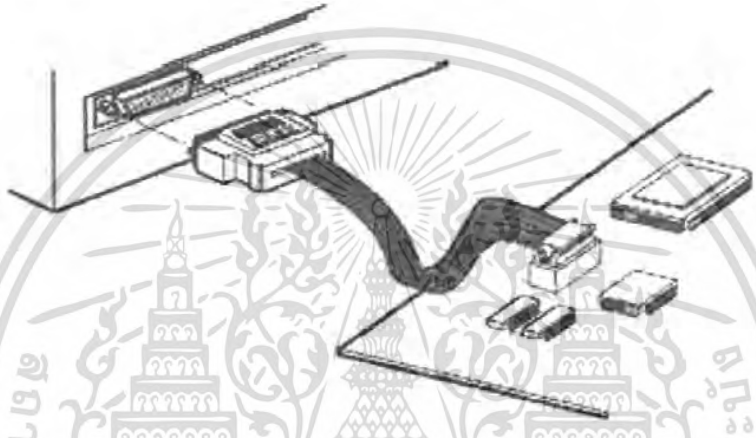
2) Anti Fuse : เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรม แล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

3) EEPROM : FPGA ที่ใช้การโปรแกรมแบบนี้มักเรียกว่า CPLD ซึ่งเทคโนโลยีที่ใช้จะเหมือนกับ EEPROM ทำให้มีความจุของเกทต่ำ โดยทั่วไปจะน้อยกว่า 20,000 เกท แต่ข้อดีของ EEPROM Based FPGA คือสามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง และในการโปรแกรมจะใช้ทรานซิสเตอร์ 1 ตัวต่อ 1 บิท ซึ่งการโปรแกรมสามารถทำได้ประมาณ 10,000 ครั้ง

4) SRAM : FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM) ทำให้สามารถ โปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 – 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการโปรแกรมน้อย (ระดับ nsec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และเหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมในภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

การโปรแกรมอุปกรณ์หรือชิป FPGA นั้น มีวิธีใหญ่ๆ 3 ลักษณะ ทั้งนี้ตัวชิปจะต้องสนับสนุนการทำงานในโหมดของการโปรแกรมเหล่านี้ด้วย

- การโปรแกรมโดยผ่านสายดาวน์โหลด หรือผ่าน JTAG หรือผ่าน ISP
- การโปรแกรมโดยใช้ ตัวเก็บข้อมูลไฟล์บิต
- การโปรแกรมโดยใช้เครื่องโปรแกรมไอซี



รูปที่ 2.8 การโปรแกรมลงในชิพ

2.2.6 ภาษา VDHL

VHDL ย่อมาจาก VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง(High Level Language) ที่ใช้สำหรับการออกแบบและบรรยายลักษณะพฤติกรรมของฮาร์ดแวร์ทางดิจิตอล ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง การจำลอง (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายลักษณะพฤติกรรมของฮาร์ดแวร์ตั้งแต่ระดับบนซึ่งก็คือระบบจนถึงระดับเกทอีกด้วย โดยที่ตัวของภาษานั้นสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบ ภาษา VHDL เป็นมาตรฐาน IEEE ซึ่งหมายความว่าเป็นภาษามาตรฐานที่สามารถใช้กับผู้ผลิตชิพ CPLD/FPGA ได้ทุกราย

การพัฒนาของภาษา VHDL

ภาษา VHDL หรือ VHSIC (Very High Speed Integrated Circuit) Hardware Description Language ซึ่งเป็นภาษาที่พัฒนาขึ้นโดยกระทรวงกลาโหมของสหรัฐอเมริกาในช่วงปี ค.ศ. 1980 โดยเป้าหมายของโครงการนี้ก็เพื่อพัฒนา ขีดความสามารถในการออกแบบวงจรรวมให้สูงขึ้นและสามารถทำได้ง่ายมากยิ่งขึ้น เป้าหมายของหลักของพัฒนาภาษา VHDL มี 2 ประการคือ

- เนื่องจากนักออกแบบวงจรรวมมีความต้องการภาษาที่สามารถรองรับการออกแบบวงจรที่มีความซับซ้อน

- นักออกแบบต้องการภาษาที่เป็นมาตรฐานหรือเป็นภาษากลางที่ทำให้สามารถเผยแพร่ผลงานการออกแบบกันภายใน กลุ่มนักออกแบบด้วยกันได้

ในปี 1986 ภาษา VHDL ได้เริ่มมีการปรับปรุงภาษา VHDL เพื่อให้สามารถกำหนดเป็นมาตรฐานของ IEEE โดยสามารถประกาศเป็นมาตรฐานได้ในเดือนธันวาคมปี 1987 โดยอยู่ในหมวด IEEE 1076 – 1987 หลังจากนั้นก็ได้มีการพัฒนาปรับปรุงอย่างต่อเนื่องโดยได้มีประกาศปรับปรุงอีกครั้งในปี 1993 ซึ่งเรียกว่า IEEE 1076 – 1993 โดยได้มีการเพิ่มเติม Syntax พิเศษเพื่อให้ผู้ใช้สามารถใช้งานได้สะดวกมากยิ่งขึ้น สำหรับขีดความสามารถในการออกแบบโดยใช้ภาษา VHDL นั้นสามารถออกแบบได้เฉพาะวงจรที่มีลักษณะเป็น Digital เท่านั้นส่วนวงจรที่เป็น Analog ในขณะนี้ยังไม่สามารถออกแบบได้โดยการใช้ภาษา VHDL แต่จะทำได้ในอนาคต

ข้อดีของภาษา VDHL มีดังนี้

- **Standard :** เป็นมาตรฐานของ IEEE ทำให้มี Tools และบริษัทที่สนับสนุนการทำงานมากมาย นอกจากนี้วงจรที่ออกแบบโดย VHDL ก็จะใช้งานได้นานเนื่องจากมีความเข้ากันได้ของภาษากับวงจรที่ได้รับการออกแบบใหม่

- **Government Support :** เนื่องจาก VHDL ได้รับการพัฒนาโดยกระทรวงกลาโหมของสหรัฐอเมริกา ดังนั้นการออกแบบวงจรโดยใช้ภาษา VHDL จึงได้รับการสนับสนุนจากรัฐบาลสหรัฐอเมริกา

- **Industrial Support :** เนื่องจากภาษา VHDL เป็นภาษาที่เป็นมาตรฐานของ IEEE จึงมีอุตสาหกรรมจำนวนมากที่รองรับการออกแบบที่ใช้ภาษา VHDL

- **Portability :** การออกแบบโดยใช้ภาษา VHDL สามารถนำไปจำลองการทำงานหรือสังเคราะห์ด้วยซอฟท์แวร์ตัวใดก็ได้ที่รองรับภาษา VHDL จึงทำให้การออกแบบด้วยภาษา VHDL จึงเป็นการออกแบบที่ไม่ยึดติดกับซอฟท์แวร์ที่ใช้ในการออกแบบ

- **Modeling capability :** ผู้ออกแบบวงจรสามารถออกแบบวงจรโดยใช้ภาษา VHDL ได้หลายระดับตั้งแต่ระดับ โครบล็อก (Macro block) จนถึงระดับเกท (Gate) และสามารถออกแบบวงจรที่มีความซับซ้อนสูงและมีขนาดใหญ่มากได้

- **Reusability :** วงจรที่ออกแบบโดยภาษา VHDL สามารถนำกลับมาใช้ใหม่ได้ง่าย เนื่องจากสามารถเปลี่ยนแปลงแก้ไขวงจรได้ง่าย

- **Documentation :** VHDL เป็นภาษาในรูปแบบบรรยายพฤติกรรม ทำให้เราสามารถอธิบายการทำงานของวงจรภายในการออกแบบได้ทันที

ลักษณะการใช้งานภาษา VDHL อาจจำแนกได้ 5 ประเภท คือ

1) Document Language : ใช้สำหรับบรรยายรายละเอียดการทำงานของวงจรที่ออก

2) Design Language : ใช้สำหรับออกแบบวงจรที่มีความซับซ้อนสูงเพื่อใช้สำหรับการจำลองการ

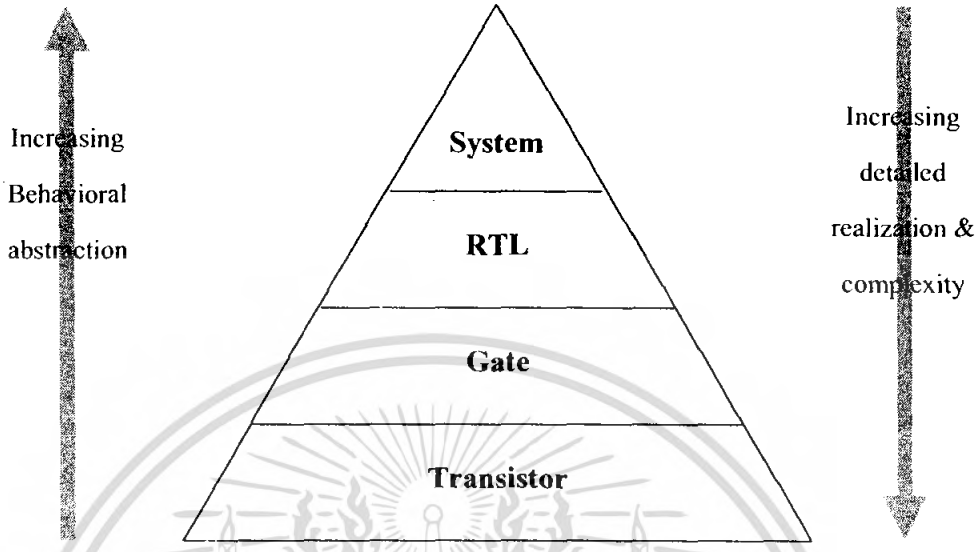
3) Verification Language : ใช้ตรวจสอบการทำงานของวงจรที่ออกแบบว่ามีความถูกต้อง

4) Test Language : ใช้สำหรับสร้าง Test Vector เพื่อใช้สำหรับเป็นข้อมูลที่ใช้สำหรับทดสอบการทำงานของวงจรที่ออกแบบ

5) Synthesis Language : ใช้สำหรับสร้างวงจรจริงเพื่อนำไป Implement เป็น Hardware ต่อไป

ระดับของการออกแบบวงจรดิจิตอลใน VHDL

ในวงจรดิจิตอลที่มีฟังก์ชันการทำงานที่เหมือนกัน สามารถออกแบบได้หลายๆระดับที่แตกต่างกัน ตั้งแต่ระดับสูงสุดไปจนถึงระดับล่างสุด ขึ้นอยู่กับการออกแบบตามความเหมาะสมและทักษะของผู้ออกแบบ



รูปที่ 2.9 ระดับของการออกแบบวงจรดิจิตอลในภาษา VHDL

1) System Level เป็นระดับบนสุดของการเขียน VHDL โดยจะบรรยายพฤติกรรมการทำงานของระบบหรือแบบอัลกอริทึมของระบบ โดยคำนึงถึงเรื่องข้อมูลของเวลา

2) RTL (Register Transfer Level) เป็นการบรรยายลงลึกในแต่ละส่วนของวงจรในระดับ Architecture โดยระดับนี้อาจเห็นรายละเอียดลึกลงไปในวงจร ตลอดจนการไหลเข้าออกของข้อมูลในแต่ละบล็อกภายในวงจร

3) Gate Level เป็นการบรรยายระดับลึกกว่าระดับ RTL โดยจะมีรายละเอียดวงจรระดับของลอจิกเกต ฟลิปฟลอปต่างๆ ที่เชื่อมต่อกันเป็นวงจร

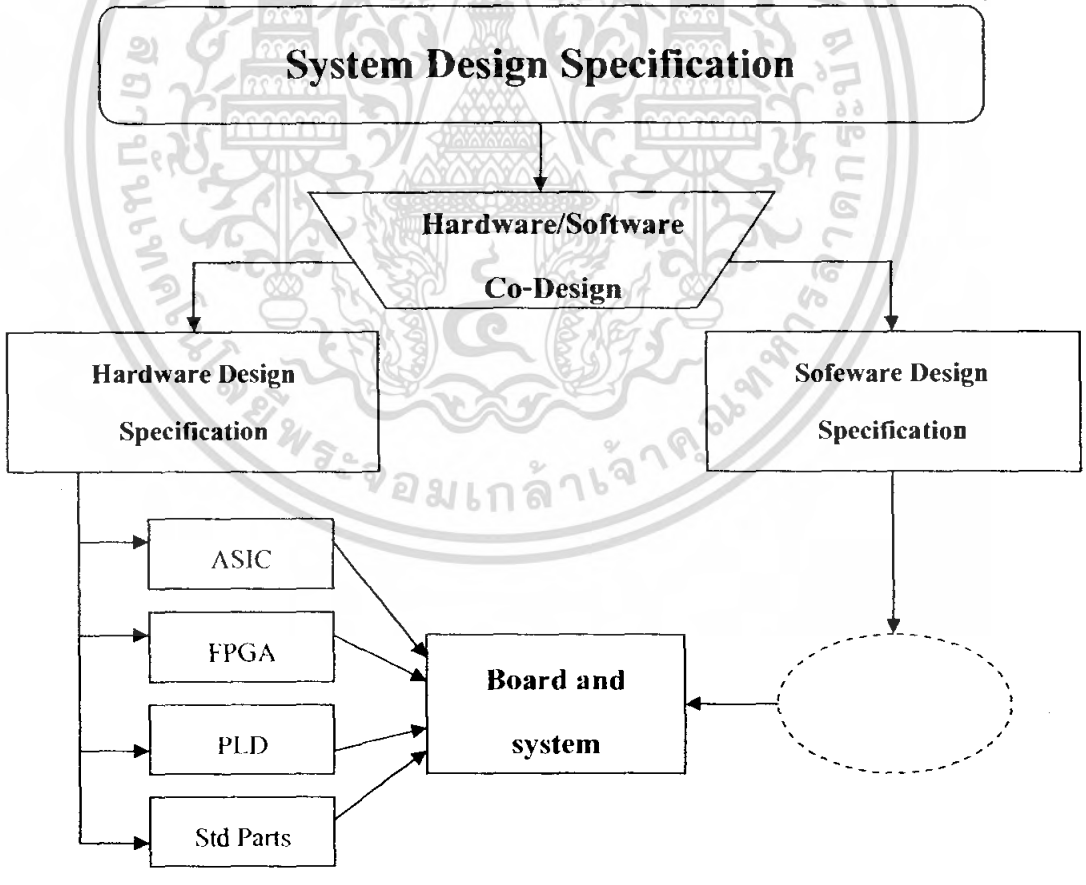
4) Transistor Level เป็นระดับล่างสุดของการออกแบบวงจรหรือระบบดิจิตอล บางครั้งอาจเรียกระดับนี้ว่า Layer Level ในระดับนี้ภาษา VHDL ไม่สามารถใช้เขียนบรรยายวงจรได้ จะต้องาศัยซอร์ฟแวร์ช่วยในการสังเคราะห์วงจรในระดับทรานซิสเตอร์ หรือโดยทั่วไปมักเรียกว่าซอร์ฟแวร์ช่วยในการวางและเชื่อมต่อทรานซิสเตอร์ (Place and Route)

ภาษาโปรแกรมฮาร์ดแวร์ที่ใช้ในการสร้างวงจรดิจิตอลสำหรับ FPGA นั้น นอกจากภาษา VHDL แล้วก็ยังมีภาษาอื่นๆ เช่น ADHL, Verilog หรือภาษาของแต่ละบริษัทที่ผลิตชิพขึ้นมา แต่ VHDL นั้นจะเป็นภาษากลางที่สุดที่สามารถใช้งานได้กับ FPGA และ Synthesis Tool ของทุกบริษัท

2.2.7 วิธีการออกแบบระบบอิเล็กทรอนิกส์

การออกแบบระบบอิเล็กทรอนิกส์ จะเริ่มต้นจากการกำหนดหน้าที่การทำงานของระบบอิเล็กทรอนิกส์ที่ต้องการซึ่งโดยส่วนใหญ่แล้ว ระบบอิเล็กทรอนิกส์มักจะประกอบด้วย 2 ส่วนคือ ส่วนที่เป็น Hardware และส่วนที่เป็น Software โดยทั่วไปแล้วการออกแบบทั้ง 2 ส่วนดังกล่าวนี้จะต้องทำไปพร้อมๆ กัน หลังจากนั้นจะเป็นขั้นตอนการกำหนดหน้าที่การทำงานของ Hardware และ Software โดยสำหรับการออกแบบทางด้าน Hardware สามารถเลือกได้ว่าต้องการออกแบบเป็น ASIC, FPGA, PLD หรือจาก Standard Component ที่มีขายอยู่ทั่วไปตามท้องตลาด สำหรับการออกแบบทาง Software ผู้ออกแบบสามารถเลือกใช้ภาษา Programming ต่างๆ เช่น ภาษา C/C++ ภาษา Assembly เป็นต้น สำหรับในการออกแบบ Firmware และ Application

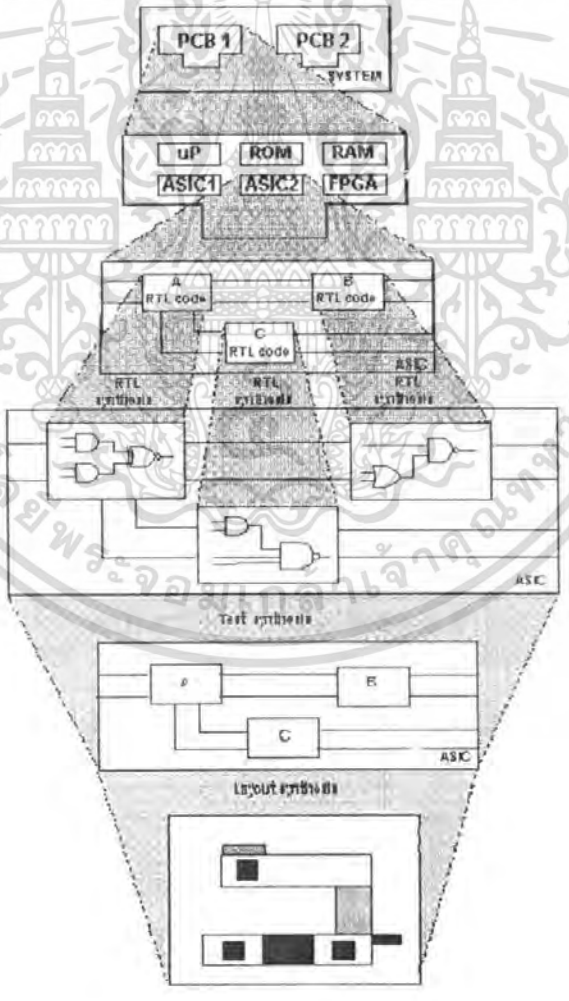
โดยในระหว่างการออกแบบนั้น ผู้ออกแบบทั้งทางด้าน Hardware และ Software จะต้องมีการทดสอบการทำงานร่วมกันเพื่อทำให้สามารถแก้ไขความผิดพลาดในการออกแบบที่เกิดขึ้นได้



รูปที่ 2.10 การออกแบบระบบอิเล็กทรอนิกส์

2.2.8 การออกแบบ แบบบนลงล่าง (Top-Down Design)

เป็นการออกแบบที่เน้นความถูกต้องและประสิทธิภาพในการทำงานของวงจรที่ออกแบบ โดยจะใช้การออกแบบ ในลักษณะที่เรียกว่า High-level behavioral description จะมีการแบ่งวงจรทั้งหมดที่ต้องการออกแบบออกเป็นส่วน แล้วทำการออกแบบในแต่ละส่วนแล้วจำลองการทำงานของวงจร ในแต่ละส่วนแยกอิสระออกจากกันเพื่อตรวจสอบความถูกต้อง ของหน้าที่การทำงานของวงจรและประสิทธิภาพการทำงานของวงจร ขั้นต่อไปก็จะนำการออกแบบไปสังเคราะห์ เพื่อให้ได้เป็นวงจรในระดับ Gate Level แล้วทำการจำลองการทำงาน เพื่อตรวจสอบความล่าช้าของการทำงานภายในวงจร และทดสอบประสิทธิภาพการทำงานของวงจรก่อนที่จะนำไป Implement เป็น Hardware



รูปที่ 2.11 การออกแบบบนลงล่าง (Top-down Design)

2.3 ระบบเครือข่ายคอมพิวเตอร์

การสื่อสารข้อมูล เป็การแลกเปลี่ยนข่าวสารและข้อมูลกันระหว่างอุปกรณ์ที่ใช้ในการสื่อสาร โดยต้องมีสื่อในการโอนถ่ายข้อมูลกัน และการสื่อสารข้อมูลกันได้นั้นจะต้องอาศัยทั้งฮาร์ดแวร์และซอร์พแวร์ โดยที่คุณสมบัติพื้นฐานของการสื่อสารข้อมูล ประกอบด้วย ความถูกต้องของการส่ง (Delivery) ความถูกต้องของข้อมูล (Accuracy) และเวลาที่เหมาะสม (Timeliness)

2.3.1 รูปแบบของการเชื่อมโยงเครือข่ายหรือโทโลโปยี

สถาปัตยกรรมของระบบเครือข่าย (Network Architecture) หรือโทโปโลยี (Topology) แต่ละแบบมีความเหมาะสมในการใช้งานแตกต่างกัน

1) โทโปโลยีแบบเมช (Mesh Topology) เป็นรูปแบบการเชื่อมโยงที่ถือว่า สามารถป้องกันการผิดพลาดที่อาจจะเกิดขึ้นกับระบบได้ดีที่สุด เป็นรูปแบบที่ใช้วิธีการเดินสายของแต่เครื่อง ไปเชื่อมการติดต่อกับทุกเครื่องในระบบเครือข่าย คือเครื่องทุกเครื่องในระบบเครือข่ายนี้ ต้องมีสายไปเชื่อมกับทุก ๆ เครื่อง

ข้อดี
- ช่วยลดปัญหาการจราจรภายในเครือข่าย เนื่องจากไม่ต้องใช้สื่อร่วมกัน
- ถ้าสื่อเส้นใดเสียหาย จะไม่ส่งผลกระทบต่อระบบ
- มีความปลอดภัยมาก เนื่องจากมีการเชื่อมโยงแบบจุดต่อจุด
- สามารถตรวจสอบความบกพร่องของระบบได้ง่าย

ข้อเสีย
- สิ้นเปลืองค่าใช้จ่ายในส่วนของสื่อที่ใช้ในการเชื่อมต่อ
- มีข้อจำกัดในการนำไปเชื่อมโยงกับโทโปโลยีแบบอื่นๆ

รูปที่ 2.12 โทโปโลยีแบบเมช (Mesh Topology)

2) โทโปโลยีแบบวงแหวน (Ring Topology) เป็นการเชื่อมต่ออุปกรณ์ต่างๆ เข้ากันเป็นวงกลม ข้อมูลข่าวสารจะถูกส่งจากโหนดหนึ่งไปยังอีกโหนดหนึ่ง วนอยู่ในเครือข่ายไปในทิศทางเดียวเหมือนวงแหวน (ในระบบเครือข่ายรูปวงแหวนบางระบบสามารถส่งข้อมูลได้สองทิศทาง) ในแต่ละโหนดหรือสถานี จะมีรีพีตเตอร์ประจำโหนด 1 ตัว ซึ่งจะทำหน้าที่เพิ่มเติมข่าวสารที่จำเป็นต่อการสื่อสาร ในส่วนหัวของแพ็กเกจข้อมูล สำหรับการส่งข้อมูลออกจากโหนด และมีหน้าที่รับแพ็กเกจข้อมูลที่ไหลผ่านมาจากสายสื่อสาร เพื่อตรวจสอบว่าเป็นข้อมูลที่ส่งมาให้โหนดตนหรือไม่ ถ้าใช่ก็จะคัดลอกข้อมูลทั้งหมดนั้นส่งต่อไปให้กับโหนดของตน แต่ถ้าไม่ใช่ก็จะปล่อยข้อมูลนั้นไปยังรีพีตเตอร์ของโหนดถัดไป

ข้อดี

- การส่งข้อมูลสามารถส่งไปยังผู้รับหลาย ๆ โหนดพร้อมกันได้ โดยกำหนดตำแหน่งปลายทางเหล่านั้นลงในส่วนหัวของแพ็กเกจข้อมูล รีพีตเตอร์ของแต่ละโหนดจะตรวจสอบเองว่ามีข้อมูลส่งมาให้ที่โหนดตนเองหรือไม่
- การส่งข้อมูลเป็นไปในทิศทางเดียวกัน จึงไม่มีการชนกันของสัญญาณข้อมูล

ข้อเสีย

- ถ้ามีโหนดใดโหนดหนึ่งเกิดเสียหาย ข้อมูลจะไม่สามารถส่งผ่านไปยังโหนดต่อไปได้ และจะทำให้เครือข่ายทั้ง เครือข่ายขาดการติดต่อสื่อสาร
- เมื่อโหนดหนึ่งต้องการส่งข้อมูล โหนดอื่น ๆ ต้องมีส่วนร่วมด้วย ซึ่งจะทำให้เสียเวลา

รูปที่ 2.13 โทโปโลยีแบบวงแหวน (Ring Topology)

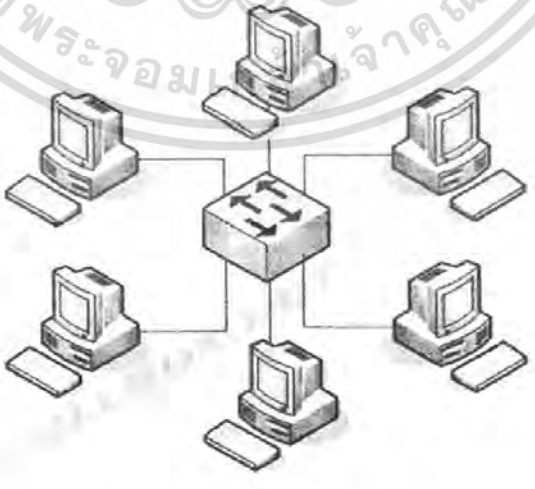
3) โทโปโลยีรูปดาว (Star Topology) เป็นการเชื่อมโยงการติดต่อสื่อสารที่มีลักษณะคล้ายรูปดาว หลายแฉก โดยมีสถานีกลาง หรือฮับ เป็นจุดผ่านการติดต่อกันระหว่างทุกโหนดในเครือข่าย สถานีกลางจึงมีหน้าที่เป็นศูนย์ควบคุมเส้นทางการสื่อสาร ทั้งหมด นอกจากนี้สถานีกลางยังทำหน้าที่เป็นศูนย์กลางคอยจัดส่งข้อมูลให้กับโหนดปลายทางอีกด้วย การสื่อสารภายใน เครือข่ายแบบดาว จะเป็นแบบ 2 ทิศทาง โดยจะอนุญาตให้มีเพียงโหนดเดียวเท่านั้นที่สามารถส่งข้อมูลเข้าสู่เครือข่ายได้ จึงไม่มีโอกาสที่หลายๆ โหนดจะส่งข้อมูลเข้าสู่เครือข่ายในเวลาเดียวกัน เพื่อป้องกันการชนกันของสัญญาณข้อมูล เครือข่ายแบบดาว เป็นโทโปโลยีอีกแบบหนึ่งที่เป็นที่นิยมใช้กันในปัจจุบัน

ข้อดี

- การติดตั้งเครือข่ายและการดูแลรักษาทำได้ง่าย
- หากมีโหนดใดเกิดความเสียหายก็สามารถตรวจสอบได้ง่าย และเนื่องจากใช้อุปกรณ์ 1 ตัวต่อสายส่งข้อมูล 1 เส้น ทำให้การเสียหายของอุปกรณ์ใดในระบบไม่กระทบต่อการทำงานของจุดอื่นๆ ในระบบ
- ง่ายในการให้บริการเพราะ โทโปโลยีแบบดาวมีศูนย์กลางทำหน้าที่ควบคุม

ข้อเสีย

- ถ้าสถานีกลางเกิดเสียขึ้นมาจะทำให้ทั้งระบบทำงานไม่ได้2. ต้องใช้สายส่งข้อมูลจำนวนมากกว่าโทโปโลยีแบบบัส และ แบบวงแหวน



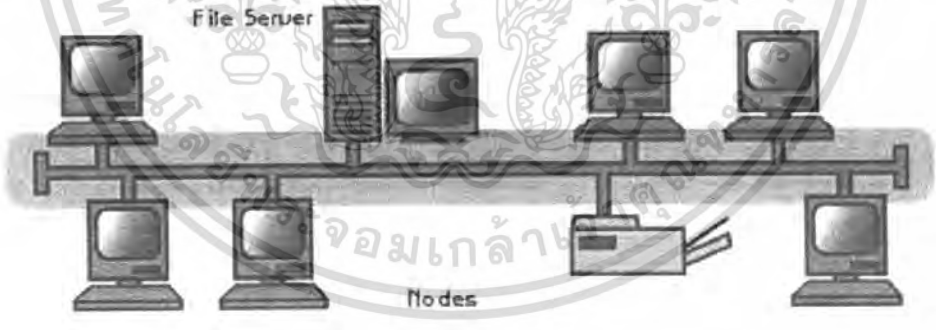
รูปที่ 2.14 โทโปโลยีรูปดาว (Star Topology)

4) โทโปโลยีแบบบัส (Bus Topology) เป็นโทโปโลยีที่ได้รับความนิยมใช้กันมากที่สุดมาตั้งแต่อดีตจนถึงปัจจุบัน ลักษณะการทำงานของเครือข่าย โทโปโลยีแบบบัส คืออุปกรณ์ทุกชิ้นหรือโหนดทุกโหนด ในเครือข่ายจะต้องเชื่อมโยงเข้ากับสายสื่อสารหลักที่เรียกว่า"บัส" (BUS) เมื่อโหนดหนึ่งต้องการจะส่งข้อมูลไปให้ยังอีกโหนด หนึ่งภายในเครือข่าย จะต้องตรวจสอบให้แน่ใจก่อนว่าบัสว่างหรือไม่ ถ้าหากไม่ว่างก็ไม่สามารถจะส่งข้อมูลออกไปได้ ทั้งนี้เพราะสายสื่อสารหลักมีเพียงสายเดียว ในกรณีที่มีข้อมูลวิ่งมาในบัส ข้อมูลนี้จะวิ่งผ่านโหนดต่างๆ ไปเรื่อยๆ ในขณะที่แต่ละโหนดจะคอยตรวจสอบข้อมูลที่ผ่านมาว่าเป็นของตนเองหรือไม่ หากไม่ใช่ ก็จะปล่อยให้ข้อมูลวิ่งผ่านไป แต่หากเลขที่อยู่ปลายทาง ซึ่งกำกับมากับข้อมูลตรงกับเลขที่อยู่ของของตน โหนดนั้นก็จะรับข้อมูลเข้าไป

ข้อดี - ใช้สายส่งข้อมูลน้อยและมีรูปแบบที่ง่ายในการติดตั้ง ทำให้ลดค่าใช้จ่ายในการติดตั้งและบำรุงรักษา

- สามารถเพิ่มอุปกรณ์ชิ้นใหม่เข้าไปในเครือข่ายได้ง่าย

ข้อเสีย - ในกรณีที่เกิดการเสียหายของสายส่งข้อมูลหลัก จะทำให้ทั้งระบบทำงานไม่ได้2. การตรวจสอบข้อผิดพลาดทำได้ยาก ต้องทำจากหลาย ๆจุด



รูปที่ 2.15 โทโปโลยีแบบบัส (Bus Topology)

5) โทโปโลยีแบบผสม (Hybridge Topology) เป็นเครือข่ายการสื่อสารข้อมูลแบบผสมระหว่างเครือข่ายแบบใดแบบหนึ่งหรือมากกว่า เพื่อความถูกต้องแน่นอน ทั้งนี้ขึ้นอยู่กับความต้องการและภาพรวมขององค์กร

2.3.2 ประเภทของเครือข่าย

1) ระบบเครือข่ายท้องถิ่น (Local Area Network : LAN) หมายถึง การเชื่อมต่อคอมพิวเตอร์ และอุปกรณ์ต่าง ๆ ในระยะใกล้ภายในสำนักงาน หรืออาคารเดียวกัน หรืออาคารที่อยู่ใกล้กันโดยใช้ สายสัญญาณ ได้แก่ สายโทรศัพท์ สายโคแอกเชียล หรือ สายใยแก้วนำแสง ตัวอย่างเช่น เครือข่ายภายในมหาวิทยาลัย ภายในอาคารหรือบริษัทเดียวกัน ระบบเครือข่ายท้องถิ่นสามารถเพิ่มประสิทธิภาพ การปฏิบัติงาน ในด้านการใช้ทรัพยากร ของระบบร่วมกัน หรือสามารถเชื่อมต่อกับเครือข่ายอื่นได้

2) ระบบเครือข่ายระดับเมือง (Metropolitan Area Network : MAN) หมายถึง การเชื่อมต่อ เครือข่ายคอมพิวเตอร์ ที่มีระยะทางการเชื่อมต่อไกลกว่า ระบบเครือข่ายท้องถิ่น (LAN) แต่ระยะทางยังคงใกล้กว่าระบบ WAN (Wide Area Network) ได้แก่เครือข่ายคอมพิวเตอร์ ที่เชื่อมต่อกันภายในเมืองเดียวกันหรือจังหวัดเดียวกัน ในเขตเดียวกัน เป็นต้น

3) ระบบเครือข่ายระยะไกล (Wide Area Network : WAN) หมายถึง การเชื่อมต่อคอมพิวเตอร์ ระยะไกล เช่น ระหว่างประเทศ การเชื่อมต่อเครือข่ายทั่วโลก เนื่องจากเป็นการติดต่อสื่อสารระยะไกล อัตราการรับส่งข้อมูลจึงต่ำ และมีโอกาสผิดพลาดได้สูง การสื่อสารระยะไกล จำเป็นต้องมีอุปกรณ์แปลงสัญญาณ คือ โมเด็ม ช่วยในการติดต่อสื่อสาร และสามารถนำเครือข่าย LAN มาเชื่อมต่อกัน เป็นเครือข่ายระยะไกลได้ ตัวอย่างของเครือข่ายระยะไกล เช่น อินเทอร์เน็ต เครือข่ายระบบงานธนาคารทั่วโลก เครือข่ายของสายการบิน เป็นต้น

2.3.3 รูปแบบการใช้งานของระบบเครือข่าย

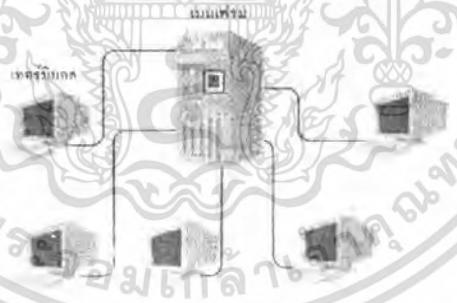
1) ระบบเครือข่ายแบบรวมศูนย์กลาง (Centralized Networks) เป็นระบบที่มีเครื่องหลักเพียงเครื่องเดียวที่ใช้ในการประมวลผล ซึ่งจะตังอยู่ที่ศูนย์กลางและมีการเชื่อมต่อไปยังเครื่องเทอร์มินอลที่อยู่รอบๆ โดยการเดินสายเคเบิลเชื่อมต่อกันโดยตรง เพื่อให้เครื่องเครื่องเทอร์มินอลสามารถเข้าใช้งาน โดยส่งคำสั่งต่างๆ มาประมวลผลที่เครื่องกลาง ซึ่งมักเป็นเครื่องคอมพิวเตอร์เมนเฟรมประสิทธิภาพสูง ระบบเครือข่ายแบบรวมศูนย์กลางจะมีราคาสูง และและไม่สามารถสนับสนุนระบบการประมวลผลแบบ Multiprocessor ได้ดีเท่ากับระบบเครือข่ายแบบ Client/Server

2) ระบบเครือข่ายแบบ Peer-to Peer แต่ละสถานีงานบนระบบเครือข่าย Peer-to-Peer จะมีความเท่าเทียมกันสามารถที่จะแบ่งปันทรัพยากรให้แก่กันและกันได้ เช่นการใช้เครื่องพิมพ์หรือแฟ้มข้อมูลร่วมกันในเครือข่าย ในขณะเดียวกันเครื่องแต่ละสถานีงานก็จะมีขีดความสามารถในการทำงานได้ด้วยตัวเอง (Stand Alone) คือจะต้องมีทรัพยากรภายในของตัวเองเช่น ดิสก์สำหรับเก็บข้อมูล หน่วยความจำที่เพียงพอ และมีความสามารถในการประมวลผลข้อมูลได้

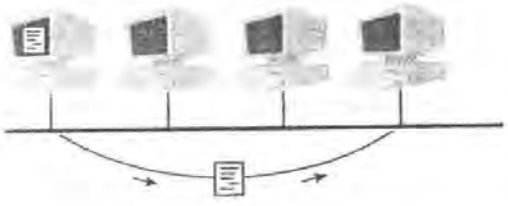
ข้อดีของระบบนี้คือ ความง่ายในการจัดตั้งระบบ มีราคาถูก และสะดวกต่อการบริหารจัดการ ซึ่งมักจะมอบเป็นภาระหน้าที่ของผู้ใช้ในแต่ละสถานีงานให้ รับผิดชอบในการดูแลพิจารณาการแบ่งปันทรัพยากรของตนเองให้กับสมาชิกผู้อื่นในกลุ่ม ดังนั้นระบบนี้จึงเหมาะสมสำหรับสำนักงานขนาดเล็กที่มีสถานีงานประมาณ 5-10เครื่องที่วางอยู่ในพื้นที่เดียวกัน

ข้อเสียของระบบนี้คือ เรื่องการรักษาความปลอดภัยของข้อมูล เนื่องจากไม่มีระบบการป้องกันในรูปแบบของ บัญชีผู้ใช้ และรหัสผ่าน ในการเข้าถึงทรัพยากรต่างๆ ของระบบ

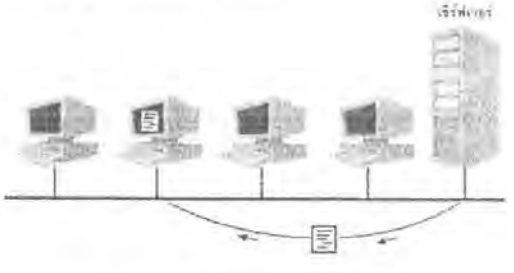
3) ระบบเครือข่ายแบบ Client/Server เป็นระบบเครือข่ายที่มีประสิทธิภาพสูง และมีการใช้งานกันอย่างกว้างขวางมากกว่าระบบเครือข่ายแบบอื่นที่มีในปัจจุบัน ระบบ Client/Server สามารถสนับสนุนให้มีเครื่องลูกข่ายได้เป็นจำนวนมาก และสามารถเชื่อมต่อกับเครื่องคอมพิวเตอร์ได้หลายแพลตฟอร์ม ระบบนี้จะทำงานโดยมีเครื่อง Server ที่ให้บริการ เป็นศูนย์กลางอย่างน้อย 1 เครื่อง และมีการบริหารจัดการทรัพยากรต่างๆ จากส่วนกลาง ระบบเครือข่ายแบบ Client/Server เป็นระบบที่มีความยืดหยุ่นสูง สนับสนุนการทำงานแบบ Multiprocessor สามารถเพิ่มขยายขนาดของจำนวนผู้ใช้ได้ตามต้องการ นอกจากนี้ยังสามารถเพิ่มจำนวนเครื่อง Servers สำหรับให้บริการต่างๆ เพื่อช่วยกระจายภาระของระบบได้ ส่วนข้อเสียของระบบนี้ก็คือ มีความยุ่งยากในการติดตั้งมากกว่าระบบ Peer-to-Peer รวมทั้งต้องการบุคลากรเพื่อการบริหารจัดการระบบโดยเฉพาะอีกด้วย



(ก) ระบบเครือข่ายแบบรวมศูนย์กลาง (Centrallized Networks)



(ข) ระบบเครือข่ายแบบ Peer-to Peer



(ค) ระบบเครือข่ายแบบ Client/Server

รูปที่ 2.16 ระบบเครือข่าย

2.3.4 อุปกรณ์เชื่อมต่อเครือข่าย

1) รีพีตเตอร์ (Repeater) หรือเรียกว่า ฮับ (HUB) เป็นอุปกรณ์ที่ทำงานอยู่ในระดับฟิสิคัลเลเยอร์ (Physical Layer) ใน OSI Model มีหน้าที่เชื่อมต่อสำหรับขยายสัญญาณให้กับเครือข่าย เพื่อเพิ่มระยะทางในการรับส่งข้อมูลให้กับเครือข่ายให้ไกลออกไปได้กว่าปกติ และมีหน้าที่รับส่งเฟรมข้อมูลทุกเฟรมที่ได้รับจากพอร์ตใดพอร์ตหนึ่ง ไปยังพอร์ตที่เหลือ คอมพิวเตอร์ที่เชื่อมต่อเข้ากับฮับจะแชร์แบนด์วิธหรืออัตราข้อมูลของเครือข่าย ข้อจำกัด คือทำหน้าที่ในการส่งต่อสัญญาณที่ได้มาเท่านั้น จะไม่มีการเชื่อมต่อกับระบบเครือข่ายซึ่งอาศัยวิธีการ access ที่แตกต่างกัน เช่น Ethernet กับ Token Ring และไม่รู้จักลักษณะของข้อมูลที่แฝงมากับสัญญาณเลย

2) บริดจ์ (Bridge) บริดจ์ มักใช้ในการเชื่อมต่อวงแลน (LAN Segment) 2 วงเข้าด้วยกัน ทำให้สามารถขยายขอบเขตของ เครือข่ายออกไปเรื่อยๆ โดยที่ประสิทธิภาพรวมของระบบไม่ลดลงมากนัก โดยบริดจ์อาจเป็นได้ทั้งฮาร์ดแวร์เฉพาะ หรือ ซอฟแวร์บนเครื่องคอมพิวเตอร์ บริดจ์จะมีการทำงานที่ดาต้าลิ้งค์เลเยอร์ (Data Link Layer) ทำการกรองสัญญาณและส่งผ่านแพ็กเก็ตข้อมูลไปยังส่วนต่างๆ ของระบบเครือข่าย ซึ่งอาจจะเป็นส่วนของระบบเครือข่ายที่มีโครงสร้างสถาปัตยกรรมที่แตกต่างกันได้ เช่น บริดจ์สามารถเชื่อมโยงส่วนของ Ethernet เข้ากับส่วนของ Token Ring ได้ และถึงแม้ว่าระบบเครือข่ายทั้งคู่จะใช้โปรโตคอลที่แตกต่างกัน บริดจ์ก็ยังคงสามารถโยกย้ายแพ็กเก็ตข้อมูลระหว่างระบบเครือข่ายทั้งสองได้อยู่ดี

3) สวิตช์ (Switch) หรือที่นิยมเรียกว่า อีเธอร์เนตสวิตช์ (Ethernet Switch) จะเป็น บริดจ์แบบหลายช่องทาง (Multiport Bridge) ที่นิยมใช้ในระบบเครือข่าย LAN แบบ Ethernet เพื่อใช้เชื่อมต่อเครือข่ายหลายๆ เครือข่าย (Segment) เข้าด้วยกัน สวิตช์จะช่วยลดการจราจรระหว่างเครือข่ายที่ไม่จำเป็น และเนื่องจากการเชื่อมต่อแต่ละช่องทางกระทำอยู่ภายในตัวสวิตช์เอง ทำให้สามารถทำการแลกเปลี่ยนข้อมูลในแต่ละเครือข่าย (Switching) ได้อย่างรวดเร็วกว่าการใช้บริดจ์จำนวนหลายๆ ตัวเชื่อมต่อกัน

4) เราท์เตอร์ (Router) เป็นอุปกรณ์ที่ทำงานอยู่ในระดับที่สูงกว่าบริดจ์ทำให้สามารถใช้ในการเชื่อมต่อระหว่างเครือข่ายที่ใช้โปรโตคอลต่างกันได้ และสามารถทำการกรอง (Filter) เลือกเฉพาะชนิดของข้อมูลที่ระบุไว้ว่าให้ผ่านไปได้ทำให้ช่วยลดปัญหาการจราจรที่คับคั่งของข้อมูล และเพิ่มระดับความปลอดภัยของเครือข่าย นอกจากนี้ เราท์เตอร์ยังสามารถหาเส้นทางการส่งข้อมูลที่เหมาะสมให้โดยอัตโนมัติด้วย อย่างไรก็ดีเราท์เตอร์ จะเป็นอุปกรณ์ที่ขึ้นอยู่กับโปรโตคอล นั่นคือในการใช้งานจะต้องเลือกซื้อเราท์เตอร์ที่สนับสนุนโปรโตคอลเครือข่ายที่ต้องการจะเชื่อมต่อเข้าด้วยกัน

5) เกทเวย์ (Gateway) เป็นอุปกรณ์ที่มีหน้าที่ในการเชื่อมต่อและแปลงข้อมูลระหว่างเครือข่ายที่แตกต่างกันทั้งในส่วนของโปรโตคอล และสถาปัตยกรรมเครือข่าย เช่น เชื่อมต่อและแปลงข้อมูลระหว่างระบบเครือข่าย LAN และระบบ Mainframe หรือเชื่อมระหว่างเครือข่าย SNA ของ IBM กับ DECNet ของ DEC เป็นต้น โดยปกติ เกทเวย์มักเป็น Software Package ที่ใช้งานบนเครื่องคอมพิวเตอร์เครื่องใดเครื่องหนึ่ง (ซึ่งทำให้เครื่องนั้นมีสถานะเกทเวย์) และมักใช้สำหรับเชื่อม Workstation เข้าสู่เครื่องที่เป็นเครื่องหลัก (Host) ทำให้เครื่องเป็น Workstation สามารถทำงาน ติดต่อกับเครื่องหลักได้ โดยไม่ต้องกังวลเกี่ยวกับข้อแตกต่างของระบบเลย

2.3.5 แบบจำลองของเครือข่าย

หลักการออกแบบเลเยอร์

- แต่ละเลเยอร์จะมีการกำหนดการทำงานอย่างละเอียดโดยมีการทำงานเป็นอิสระไม่ขึ้นต่อกัน
- ฟังก์ชันภายในเลเยอร์จะพยายามมุ่งไปสู่ข้อกำหนดมาตรฐาน (standard protocol)
- ขอบเขตของเลเยอร์จะถูกเลือกและจำกัดให้มีปริมาณการเชื่อมต่อระหว่างเลเยอร์ให้น้อยที่สุด
- จำนวนของเลเยอร์ต้องมากพอที่จะแยกฟังก์ชั่นที่จำเป็นและแตกต่างกันไม่ให้อยู่ในเลเยอร์เดียวกัน

มาตรฐานการสื่อสารข้อมูล

การกำหนดมาตรฐานของการสื่อสารข้อมูลนั้น นับว่ามีความจำเป็นอย่างมากสำหรับระบบเครือข่ายที่มี องค์ประกอบของอุปกรณ์ต่างๆ หลากหลายผู้ผลิต ซึ่งอุปกรณ์ทั้งหมดเหล่านั้นจะต้องทำงานเข้ากันได้อย่างราบรื่น การกำหนดมาตรฐานต่างๆ นั้นจะเริ่มตั้งแต่โครงสร้างพื้นฐานของฮาร์ดแวร์ระบบเครือข่าย ได้แก่ ระบบสายเคเบิล อุปกรณ์ในการส่งสัญญาณข้อมูล ตลอดจนถึงเครื่องเซิร์ฟเวอร์ และซอฟต์แวร์ในการสื่อสารบนระบบเครือข่าย เพื่อเป็นการรับประกันว่าส่วนประกอบต่างๆ จะสามารถทำงานร่วมกันได้ ผู้ผลิตฮาร์ดแวร์และซอฟต์แวร์ระบบเครือข่ายจะต้องทำตามคำแนะนำตามมาตรฐานการออกแบบและสร้างผลิตภัณฑ์ ซึ่งกำหนดขึ้นโดย องค์กรมาตรฐานสากล (International Organization for Standardization - ISO) โดยมาตรฐานที่กำหนดขึ้นและได้ประกาศใช้ตั้งแต่ปี คศ.1984 เรียกว่า Open Systems Interconnection Reference Model เรียกสั้นๆ ว่า OSI Reference Model หรือ ISO/OSI Model

แบบจำลอง OSI (OSI Model)

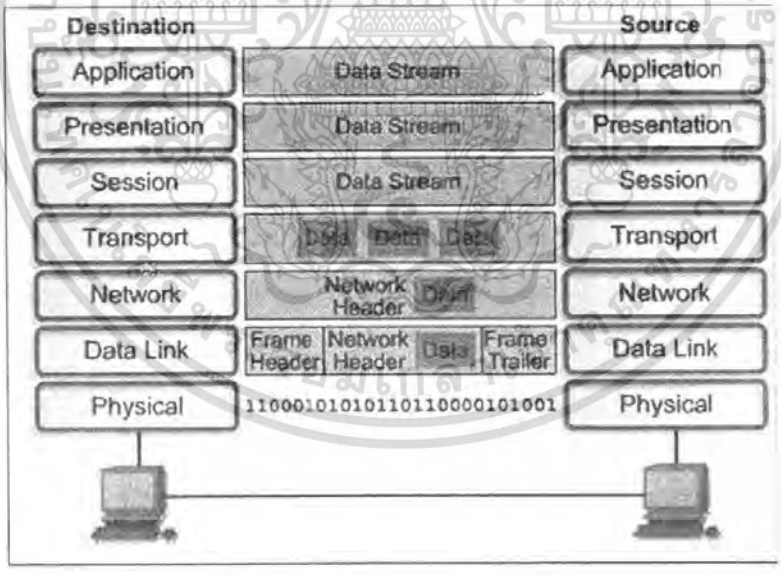
OSI Reference Model เป็นการกำหนดชุดของคุณลักษณะเฉพาะที่ใช้อธิบายโครงสร้างของระบบเครือข่าย โดยมีวัตถุประสงค์ เพื่อให้ผู้ผลิตฮาร์ดแวร์หรือซอฟต์แวร์ใดๆ ใช้เป็นโครงสร้าง

อ้างอิงในการสร้างอุปกรณ์ให้สามารถทำงานร่วมกันได้อย่างดีบนระบบเครือข่าย โดยมีการจัดแบ่งเลเยอร์ของ OSI ออกเป็น 7 เลเยอร์ แต่ละเลเยอร์จะมีการโต้ตอบหรือรับส่งข้อมูลกับเลเยอร์ที่อยู่ข้างเคียงเท่านั้น โดยเลเยอร์ที่อยู่ชั้นล่างจะกำหนดลักษณะของอินเตอร์เฟซ เพื่อให้บริการกับเลเยอร์ที่อยู่เหนือขึ้นไปตามลำดับขั้น เริ่มตั้งแต่ส่วนล่างสุดซึ่งเป็นการจัดการลักษณะทางกายภาพของฮาร์ดแวร์และการส่งกระแสของข้อมูลในระดับบิต ไปสิ้นสุดที่แอพพลิเคชั่นเลเยอร์ในส่วนบนสุด

OSI Model ได้แบ่ง ตามลักษณะของออกเป็น 2 กลุ่มใหญ่ ได้แก่

- Application-oriented Layers เป็น 4 Layer ด้านบนคือ Layer ที่ 7,6,5,4 ทำหน้าที่เชื่อมต่อรับส่งข้อมูลระหว่างผู้ใช้กับโปรแกรมประยุกต์ เพื่อให้รับส่งข้อมูลกับฮาร์ดแวร์ที่อยู่ชั้นล่างได้อย่างถูกต้อง ซึ่งจะเกี่ยวข้องกับซอฟแวร์

- Network-dependent Layers เป็น 3 Layers ด้านล่าง ทำหน้าที่เกี่ยวกับการรับส่งข้อมูลผ่านสายส่ง และควบคุมการรับส่งข้อมูล ตรวจสอบข้อผิดพลาด รวมทั้งเลือกเส้นทางที่ใช้ในการรับส่ง ซึ่งจะเกี่ยวข้องกับฮาร์ดแวร์เป็นหลัก ทำให้ใช้ผลิตภัณฑ์ต่างบริษัทกันได้



รูปที่ 2.17 แบบจำลอง OSI และรูปแบบข้อมูลในแต่ละเลเยอร์

เลเยอร์ 1: Physical Layer

Layer นี้เป็นการกล่าวถึงข้อกำหนดมาตรฐานคุณสมบัติทางกายภาพของฮาร์ดแวร์ที่ใช้เชื่อมต่อระหว่างคอมพิวเตอร์ทั้ง2ระบบ สัญญาณทางไฟฟ้าและการเชื่อมต่อต่างๆของสาย

เคเบิล,Connectorต่างๆ เช่นสายที่ใช้รับส่งข้อมูลเป็นแบบไหน ข้อต่อหรือปลั๊กที่ใช้มีมาตรฐานอย่างไร ใช้ไฟกี่โวลต์ ความเร็วในการรับส่งเป็นเท่าไร สัญญาณที่ใช้รับส่งข้อมูลมีมาตรฐานอย่างไร Layer1 นี้จะมองเห็นข้อมูลเป็นการรับ-ส่งที่ละ bit เรียงต่อกันไปโดยไม่มีการพิจารณาเรื่องความหมายของข้อมูลเลย การรับส่งจะเป็นในรูป 0 หรือ 1 หากการรับส่งข้อมูลมีปัญหาเนื่องจากฮาร์ดแวร์ เช่นสายขาดก็จะเป็นหน้าที่ของ Layer1 นี้ที่จะตรวจสอบและแจ้งข้อผิดพลาดนั้นให้ชั้นอื่นๆที่อยู่เหนือขึ้นไปทราบ

เลเยอร์ 2: Data Link Layer

เลเยอร์นี้มีจุดประสงค์หลักคือพยายามควบคุมการส่งข้อมูลให้เสมือนกับว่าไม่มีความผิดพลาดเกิดขึ้น เพื่อให้เลเยอร์สูงขึ้นไปสามารถนำข้อมูลไปใช้ได้อย่างถูกต้อง วิธีการคือฝ่ายผู้ส่งจะทำการแตกข้อมูลออกเป็นเฟรมข้อมูล (data-frame) โดยจะต้องมีการกำหนดขอบเขตของเฟรม (frame boundary) โดยการเติมบิทเข้าไปยังจุดเริ่มต้นและจุดสิ้นสุดของเฟรม จากนั้นทำการส่งเฟรมข้อมูลออกไปทีละชุดและรอรับการตอบรับ (acknowledge frame) จากผู้รับ ถ้าหากมีการสูญหายของเฟรมข้อมูล ซึ่งอาจเนื่องมาจากสัญญาณรบกวนจากภายนอกหรือข้อผิดพลาดอื่นๆ ในกรณีนี้ฝ่ายผู้ส่งจะต้องส่งเฟรมข้อมูลเดิมออกมาใหม่

เลเยอร์ 3: Network Layer

เป็นเลเยอร์ที่ทำหน้าที่หลักเกี่ยวข้องกับการหาเส้นทาง (routing) ในการส่งแพคเก็ตจากต้นทางไปยังปลายทาง ซึ่งจะมีการสลับช่องทางในการส่งข้อมูลหรือที่เรียกว่า แพ็กเกตสวิตชิ่ง (packet switching) มีการสร้างวงจรเสมือน (virtual circuit) ซึ่งคล้ายกับว่ามีเส้นทางเชื่อมโยงกันระหว่างคอมพิวเตอร์ 2 เครื่องให้ติดต่อสื่อสารถึงกันได้โดยตรง การกำหนดเส้นทางการส่งข้อมูลนั้นคอมพิวเตอร์ฝ่ายผู้ส่งอาจทำหน้าที่พิจารณาหาเส้นทางที่เหมาะสมในการส่งข้อมูล ตั้งแต่ต้น หรืออาจใช้วิธีแบบไดนามิกส์ (dynamic) คือแต่ละแพคเก็ตสามารถเปลี่ยนแปลงเส้นทางได้ตลอดเวลา นอกจากนี้เครื่องคอมพิวเตอร์ฝ่ายผู้ส่งยังมีหน้าที่ในการจัดการเรื่องที่อยู่ของเครือข่ายปลายทางโดยจะมีการแปลงที่อยู่แบบตรรกะ (logical address) ให้เป็นที่อยู่แบบกายภาพ (physical address) ซึ่งถูกกำหนดโดยการ์ดเชื่อมต่อระบบเครือข่าย

เลเยอร์ 4: Transport Layer

Transport Layer ทำหน้าที่เสมือนบริษัทขนส่งที่รับผิดชอบการจัดส่งข้อมูลโดยปราศจากความผิดพลาด ซึ่งมีหน้าที่หลักคือ การตรวจสอบและแก้ไขความผิดพลาดที่เกิดขึ้นในข้อมูล คอย

แยกแยะและจัดระเบียบของแพ็กเก็ต ข้อมูลให้จัดเรียงลำดับอย่างถูกต้อง และมีขนาดที่เหมาะสม นอกจากนี้ยังทำการผนวกข้อมูลทั้งหลายให้อยู่ในรูปของ วงจรเดียวหรือเรียกว่าการมัลติเพล็กซ์ (multiplex) และมีกลไกสำหรับควบคุมการไหลของข้อมูลให้มีความสม่ำเสมอ

เลเยอร์ 5: Session Layer

จากเลเยอร์ที่ผ่านมาจะเห็นว่าการทำงานต่างๆ จะเกี่ยวพันอยู่เฉพาะกับบิตและข้อมูลเท่านั้น โดยไม่ได้สนใจเกี่ยวกับสถานะภาพการใช้งานจริงของผู้ใช้แต่อย่างใด ซึ่งหน้าที่ดังกล่าวนี้จะเกิดขึ้นที่ Session Layer ในเลเยอร์นี้จะมีการให้บริการสำหรับการใช้งานเครื่องที่อยู่ห่างไกลออกไป (remote login) การถ่ายโอนไฟล์ระหว่างเครื่อง โดยจะมีการจัดตั้งการสื่อสารระหว่าง 2 ฝ่าย เรียกว่า Application Entities หรือ AE ซึ่งเทียบได้กับบุคคล 2 คนที่ต้องการสนทนากันทางโทรศัพท์ โดย Session Layer จะมีหน้าที่จัดการให้การสนทนาเป็นไปอย่างราบรื่น โดยการเฝ้า ตรวจสอบการไหลของข้อมูลอย่างเป็นจังหวะ ดูแลเรื่องความปลอดภัยเช่น ตรวจสอบอายุการใช้งานของรหัสผ่าน จำกัดช่วงระเวลาในการติดต่อ ควบคุมการถ่ายเทข้อมูลรวมถึงการกู้ข้อมูลที่เสียหายอันเกิดมาจากเครือข่ายทำงานผิดปกติ นอกจากนี้ยังสามารถตรวจดูการใช้งานของระบบและจัดทำบัญชีรายงานช่วงเวลาการใช้งานของผู้ใช้ได้

เลเยอร์ 6: Presentation Layer

หน้าที่หลักคือการแปลงรหัสข้อมูลที่ส่งระหว่างเครื่องคอมพิวเตอร์ 2 เครื่องให้เป็นอักขระแบบเดียวกัน เครื่องคอมพิเตอร์ส่วนใหญ่จะใช้รหัส ASCII (American Standard Code for Information Interchange) แต่ในบางกรณีเครื่องที่ใช้รหัส ASCII อาจจะต้องสื่อสารกับเครื่องเมนเฟรมของ IBM ที่ใช้รหัส EBCDIC (Extended Binary Coded Decimal Interchange Code) ดังนั้น Presentation Layer จะทำหน้าที่แปลงรหัสเหล่านี้ให้เครื่องคอมพิวเตอร์เข้าใจได้ตรงกัน นอกจากนี้ยังสามารถทำการลดขนาดของข้อมูล (data compression) เพื่อเป็นการประหยัดเวลาในการรับส่ง และสามารถเข้ารหัสเพื่อเป็นการป้องกันการโจรกรรมข้อมูลได้อีกด้วย

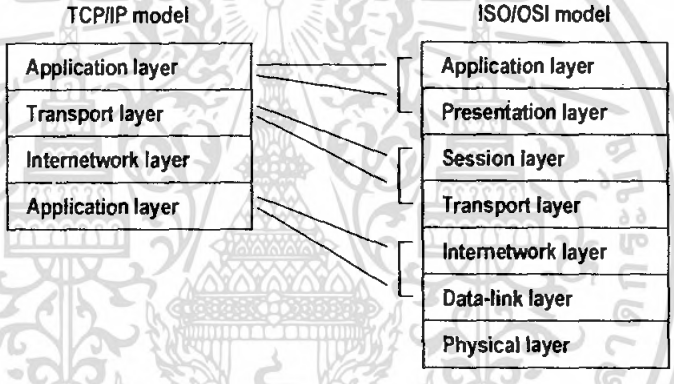
เลเยอร์ 7: Application Layer

เป็นเลเยอร์บนสุดที่ทำงานใกล้ชิดกับผู้ใช้ การทำงานของเลเยอร์นี้จะเกี่ยวข้องกับโปรโตคอลต่างๆ มากมาย ซึ่งจะมีการใช้งานที่เฉพาะตัวแตกต่างกันออกไป มีบริการทางด้านโปรแกรมประยุกต์ต่างๆ ได้แก่ email, file transfer, remote job entry, directory services นอกจากนี้

ยังมีการจัดเตรียมฟังก์ชั่นในการเข้าถึงไฟล์และเครื่องพิมพ์ ซึ่งเป็นการแบ่งปันการใช้ทรัพยากรบนระบบเครือข่าย

2.3.6 แบบจำลอง TCP/IP

TCP/IP Model มีแนวคิดพื้นฐานแตกต่างจาก OSI Model คือไม่ได้มีพื้นฐานของการสื่อสารแบบการสนทนา TCP/IP Model เป็นภาพแสดงถึงโลกของระบบเครือข่ายสากล (Internetworking) ที่ทำการเคลื่อนย้ายและกำหนดเส้นทางให้กับข้อมูลระหว่างเครือข่ายและระหว่างเครื่องคอมพิวเตอร์ต่างๆ เมื่อเปรียบเทียบความสัมพันธ์ระหว่างทั้ง 2 โมเดล จะพบว่ามีบางเลเยอร์ที่มีการกำหนดคุณสมบัติที่เทียบได้ใกล้เคียงกัน แต่บางเลเยอร์ก็ไม่สามารถเทียบหาความสัมพันธ์กันได้เลย



รูปที่ 2.18 เลเยอร์ต่างๆ ใน TCP/IP และ ISO/OSI Model

Network Access Layer

ประกอบด้วยโปรโตคอลที่ใช้ในการจัดส่งเฟรมข้อมูล โดยจะพิจารณาว่าจะมีการส่งเฟรมข้อมูลไปบนระบบเครือข่ายทางกายภาพอย่างไร ซึ่งจะใช้การกำหนดที่อยู่อย่างถาวรให้กับการ์ดเชื่อมต่อระบบเครือข่าย

Inter network Layer

เลเยอร์นี้จะไม่สามารถเทียบได้กับ OSI Model เนื่องจากเป็นส่วนที่ประกอบด้วยโปรโตคอลที่ทำหน้าที่กำหนดเส้นทางให้กับข้อมูลจากผู้ส่งไปยังผู้รับ เป็นกระบวนการส่งแพ็กเก็ตข้อมูลผ่านสื่อกลางของระบบเครือข่าย โดยแพ็กเก็ตข้อมูลจะถูกเรียกเป็น Datagram ซึ่งหมายถึงเป็นแพ็กเก็ตข้อมูลที่มีข่าวสารในส่วนหัว (Header) และส่วนท้าย (Trailer) ประกอบอยู่ด้วย และยังรวมถึงการใช้เราเตอร์และเกตเวย์ในการส่ง Datagram ไปมาระหว่างโหนดต่างๆ ด้วย

Transport layer

จะทำหน้าที่เช่นเดียวกับใน OSI Reference Model คือมีหน้าที่สร้างความน่าเชื่อถือในการจัดส่ง Datagram และช่วยในการสื่อสารระหว่างเครื่องคอมพิวเตอร์ โดยจัดตั้งการเชื่อมต่อหรือสร้างวงจรเสมือน (Virtual Circuit) ซึ่งจะคล้ายกับการสนทนาใน OSI Model โดยเริ่มด้วยคำสั่งในการเปิด และสิ้นสุดด้วยคำสั่งปิด สำหรับในโลกของ TCP/IP นั้น แพ็กเก็ตข้อมูลจะถูกกำหนดเส้นทางการส่งจากแหล่งกำเนิดไปยังปลายทางผ่านเส้นทางที่ดีที่สุดในขณะนั้น ซึ่งการแลกเปลี่ยนข้อมูลลักษณะนี้เรียกว่า Connectionless

Application Layer

เลเยอร์นี้สามารถเทียบได้กับ Application Layer และ Presentation Layer ใน OSI Model โดยจะบรรจุโปรโตคอลหลายแบบที่ทำให้แอบพลิเคชั่นสามารถเข้าถึงระบบเครือข่ายและบริการบนระบบเครือข่ายได้

2.3.7 โพรโตคอล

เป็นมาตรฐานในการสื่อสารข้อมูลของคอมพิวเตอร์ หรือภาษาที่ใช้ในการสื่อสารของคอมพิวเตอร์ และต้องเป็นภาษาเดียวกัน ปัจจุบันที่นิยมใช้ คือ TCP/IP (Transmission Control Protocol/internet Protocol) เป็น โปรโตคอลที่ใหญ่ที่สุดให้ระบบเครือข่ายของโลก นอกจากนี้ยังมี IPX/SPX (Internet Packet Exchange/Scquenced Packet Exchange) ของบริษัทโนเวลล์

TCP/IP (Transmission Control Protocol/internet Protocol)

หลายเทคโนโลยีที่เราท่านใช้อยู่ทั่วไปมีจุดกำเนิดจากเทคโนโลยีการสงคราม IP เน็ตเวิร์กก็เป็นหนึ่งในนั้น เมื่อครั้งสงครามเย็นระหว่างสหรัฐอเมริกาและสหภาพโซเวียต กระทรวงกลาโหมภายใต้รัฐบาลกลางสหรัฐฯ จ้างมหาวิทยาลัยต่าง ๆ ทำวิจัยเพื่อสร้างเครือข่ายที่ทนต่อความล้มเหลว (ด้วยระเบิดนิวเคลียร์) สิ่งที่ได้คือโพรโตคอล TCP/IP เครือข่ายคอมพิวเตอร์ที่ใช้โพรโตคอลนี้เรียกสั้น ๆ ว่า TCP/IP (Transmission Control Protocol/Internet Protocol) คือชุดของโพรโตคอลที่รวมกันเป็นกลุ่มให้ใช้งานเช่น Internet Protocol (IP) , Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), User Datagram Protocol (UDP) ฯลฯ แต่โพรโตคอลที่มีบทบาทสำคัญคือ Internet Protocol (IP) โดยมีหลักการทำงาน คือแบ่งเนื้อข้อมูลที่ต้องการส่งเป็นชิ้นเล็ก ๆ เรียกว่าแพ็กเก็ตส่งแพ็กเก็ตไปยังเส้นทางที่เหมาะสมเป็นทอดจนกว่าจะถึงปลายทาง แต่ละแพ็กเก็ตอาจใช้เส้นทางคนละทิศขึ้นกับการพิจารณาของเราเตอร์ในช่วงต่าง ๆ หากเกิดข้อผิดพลาด ณ ช่วงการส่งใด เราเตอร์ที่รับผิดชอบการส่งช่วงนั้นจะจัดส่งแพ็กเก็ตชิ้นนั้นใหม่โดย

อัตโนมัติ เมื่อถึงจุดหมายระบบปลายทางจะรวบรวมแพ็กเก็ตกลับให้เป็นเนื้อข้อมูลดังเดิม ซึ่งถ้าจะว่ากันตามทฤษฎีแล้ว TCP/IP นั้นจะประกอบด้วยส่วนสำคัญอยู่ 2 ส่วนด้วยกันก็คือ TCP หรือ Transmission Control Protocol และอีกส่วนก็คือ IP หรือ Internet Protocol นั่นเอง การแบ่งลักษณะในการทำงานก็จะแบ่งเป็น TCP มีหน้าที่ในการตรวจสอบการรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ผู้รับ และเครื่องคอมพิวเตอร์ผู้ส่ง ให้ได้รับข้อมูลที่ถูกต้องและครบถ้วนหรือว่าหากมีการสูญหายของข้อมูลก็จะมีการแจ้งให้ต้นทางที่ส่งข้อมูลมารับทราบแล้วให้ทำการส่งข้อมูลมาให้ใหม่

ลักษณะการทำงาน

ลักษณะการทำงานของ IP นั้น จะทำหน้าที่ในการเลือกเส้นทางที่จะใช้ในการรับส่งข้อมูลในระบบเครือข่าย และทำการตรวจสอบที่อยู่ของผู้รับโดยการใช้ข้อมูลขนาด 4 Byte เป็นตัวกำหนดแอดเดรสหรือที่เราเรียกกันว่า IP Address ซึ่งโพรโตคอล TCP จะทำงานอยู่ในชั้น Transport Layer ตัวแพ็กเก็ต TCP จะประกอบด้วย ส่วนหัว (Header) และส่วนข้อมูล (Data) และโพรโตคอล IP จะทำงานอยู่ในชั้น Network Layer ตัวแพ็กเก็ต IP ประกอบด้วย 2 ส่วนใหญ่ ๆ คือ ส่วนหัว (IP Header) จะประกอบด้วย IP แอดเดรสของเครื่องต้นทางและปลายทาง และส่วนข้อมูล (IP Data) จะเป็นที่เก็บโพรโตคอล TCP เนื่องจากโพรโตคอล TCP/IP จะถูก Encapsulate ให้มาอยู่ในส่วนของแพ็กเก็ต IP

จุดเด่นของโพรโตคอล TCP/IP คือ

1) สามารถนำส่งข้อมูลไปถึงจุดหมายได้แม้เส้นทางบางที่เสียหาย : เป็นจุดประสงค์หลักที่ช่วยให้ทนต่อความล้มเหลว โดยหากระหว่างการสื่อสารข้อมูลและมีเส้นทางใดเสียหายหรือล้มเหลว IP เน็ตเวิร์กจะปรับใช้เส้นทางอื่นที่ทดแทนได้เพื่อนำส่งข้อมูลให้ไปถึงปลายทางอย่างอัตโนมัติ ผู้ส่งและผู้รับข้อมูลไม่จำเป็นต้องรับรู้หรือปรับตัวแต่ประการใด

2) ไม่ขึ้นกับแพลตฟอร์มใด ๆ : ไม่ว่าเครือข่ายนั้นเป็นเครือข่ายท้องถิ่นหรือเครื่อข่ายระหว่างภูมิภาค เป็นไฟล์/พรินต์เซิร์ฟเวอร์หรือไคลเอ็นต์/เซิร์ฟเวอร์ เป็นระบบปฏิบัติการใด เน็ตเวิร์กอินเทอร์เฟซเป็นแบบใดก็ตาม ในมุมมองของโพรโตคอล TCP/IP ก็คือ IP เน็ตเวิร์ก

จุดอ่อนของ IP มี 2 ประเด็น คือ

1) รับส่งโดยไม่มีการรักษาความปลอดภัยเนื้อข้อมูล : การรับส่งข้อมูลด้วย IP แพ็กเก็ตไม่มีทั้งการเข้ารหัสข้อมูลและป้องกันการปลอมแปลงใด ๆ การไม่เข้ารหัสข้อมูลอาจทำให้ผู้ไม่ประสงค์

ดีระหว่างเส้นทางที่ IP แพ็กเก็ตผ่านดักลอบดูเนื้อข้อมูลอย่างง่ายดาย แม้ว่าเราอาจสามารถบังคับเส้นทางของ IP แพ็กเก็ตได้ก็ไม่อาจมั่นใจได้ว่าระหว่างทางมีการดักลอบดูหรือไม่

ในเรื่องปัญหาการปลอมแปลงแบ่งออกเป็นสองกรณีคือ การปลอมแปลงหรือดัดแปลงเนื้อข้อมูล และการปลอมแปลงส่วนหัวของ IP แพ็กเก็ต ทั้งสองกรณีให้ผลเหมือนกันคือผู้รับได้ข้อมูลที่ผิดจากความเป็นจริง ทว่าจุดประสงค์ต่างกัน หากเป็นกรณีแรกนั้น ผู้ไม่หวังดีต้องการหลอกหรือกลั่นแกล้งให้ได้ข้อมูลผิด ๆ หากเป็นกรณีหลัง ผู้ไม่หวังดีต้องการแอบอ้างว่าข้อมูลนั้นมากจากแหล่งที่ผู้รับไว้ใจหรือแหล่งอื่นที่กลายเป็นเหยื่อของการแอบอ้างโดยไม่รู้ตัว

2) รับส่งโดยไม่คำนึงถึงคุณภาพการให้บริการ : การรับส่งต่อ IP แพ็กเก็ตระหว่างเครือข่ายย่อยไปเป็นทอดนั้นใช้หลักการใครมาก่อนได้ก่อน ฉะนั้นจึงคาดเดาไม่ได้ว่าข้อมูลที่นำส่งไปจะไปถึงปลายทางเมื่อใด แม้ว่า IP เน็ตเวิร์กใช้หลักการเลือกเส้นทางที่เหมาะสมที่สุดในขณะนั้นก็ตาม หากแต่ความเหมาะสมนั้นผู้ส่งและผู้รับไม่อาจคาดการณ์หรือมีส่วนร่วมตัดสินใจได้เลยว่าจะช้าเร็วหรือมีโอกาสที่ข้อมูลผิดพลาดมากน้อยเพียงไร

สำหรับเรื่องการรับส่งโดยไม่มีการรักษาความปลอดภัยเนื้อข้อมูลนั้น องค์กรกลางของอินเทอร์เน็ตได้ออกมาตรฐานที่ช่วยแก้ไขปัญหานี้คือ IPSec โดยมีทั้งการเข้าและถอดรหัสเนื้อข้อมูลในระดับ IP แพ็กเก็ตการตรวจสอบความถูกต้องเนื้อข้อมูลและการพิสูจน์ตนของ IP แพ็กเก็ตเพื่อป้องกันการปลอมแปลง

2.4 การประมวลผลภาพ (Image Processing)

การประมวลผลภาพ (Image Processing) คือ เป็นการประยุกต์ใช้งานการประมวลผลสัญญาณบนสัญญาณ 2 มิติ เช่น ภาพนิ่ง (ภาพถ่าย) หรือภาพวิดิทัศน์(วีดิโอ) และยังรวมถึงสัญญาณ 2 มิติอื่นๆ ที่ไม่ใช่ภาพด้วย

แนวความคิดและเทคนิค ในการประมวลผลสัญญาณ สำหรับสัญญาณ 1 มิตินั้น สามารถปรับมาใช้กับภาพได้ไม่ยาก แต่นอกเหนือจาก เทคนิคจากการประมวลผลสัญญาณแล้ว การประมวลผลภาพก็มีเทคนิคและแนวความคิดที่เฉพาะ (เช่น connectivity และ rotation invariance) ซึ่งจะมีความหมายกับสัญญาณ 2 มิติเท่านั้น แต่อย่างไรก็ตามเทคนิคบางอย่าง จากการประมวลผลสัญญาณใน 1 มิติ จะค่อนข้างซับซ้อนเมื่อนำมาใช้กับ 2 มิติ

เมื่อหลายสิบปีมาแล้ว การประมวลผลภาพนั้น จะอยู่ในรูปของการประมวลผลสัญญาณแอนะล็อก (analog) โดยใช้อุปกรณ์ปรับแต่งแสง (optics) ซึ่งวิธีเหล่านั้นก็ไม่ได้หายสาบสูญ หรือเลิก

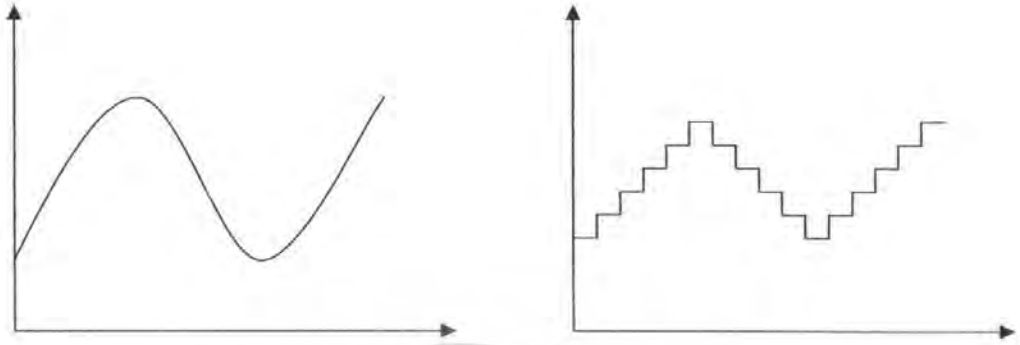
ใช้ไป ยังมีใช้เป็นส่วนสำคัญ สำหรับการประยุกต์ใช้งานบางอย่าง เช่น ฮอโลกราฟี (holography) แต่เนื่องจากอุปกรณ์คอมพิวเตอร์ในปัจจุบัน ราคาถูกลง และเร็วขึ้นมาก การประมวลผลภาพดิจิทัล (digital image processing) จึงได้รับความนิยมมากกว่า เพราะการประมวลผลที่ทำได้ซับซ้อนขึ้น แม่นยำ และง่ายใน

การประมวลผลภาพดิจิทัล (digital image processing) เป็นสาขาที่กล่าวถึงเทคนิคและอัลกอริทึมต่างๆ ที่ใช้การประมวลผลภาพที่อยู่ในรูปแบบดิจิทัล(ภาพดิจิทัล) ภาพในที่นี้ รวมความหมายถึงสัญญาณดิจิทัลใน 2 มิติอื่นๆ โดยทั่วไปคำนี้เมื่อใช้อย่างกว้างๆ จะครอบคลุมถึงสัญญาณวีดิโอ (video) หรือภาพเคลื่อนไหว ซึ่งจะเป็นชุดของภาพนิ่ง เรียกว่า เฟรม (frame) หลายๆภาพต่อกันไปตามเวลา ซึ่งก็คือสัญญาณ 3 มิติ เมื่อนับเวลาเป็นมิติที่ 3 หรือ อาจจะครอบคลุมถึงสัญญาณ 3 มิติอื่นๆ เช่น ภาพ 3 มิติทางการแพทย์ หรือ อาจจะมากกว่านั้น เช่น ภาพ 3 มิติ และ หลายชนิด (multimodal image)

2.4.1 พื้นฐานเกี่ยวกับ Digital Image

Digital Image คือ ภาพที่เก็บอยู่ในรูปแบบดิจิทัลนั่นเอง ภาพที่เรามองเห็นโดยทั่วไปนั้นเป็นภาพในลักษณะ 3 มิติ คือ มีลิมิตของความกว้าง ความยาว และความสูงหรือความลึก และภาพถ่ายที่เราเห็นกันอยู่ในทีวีหรือรูปภาพที่อยู่ในคอมพิวเตอร์นั้น เป็นการแปลงภาพจาก 3 มิติ มาเป็น 2 มิติเรียบร้อยแล้ว โดยการแปลงสัญญาณไฟฟ้าในรูปแบบอะนาลอก ยกตัวอย่างเช่น ในกล้องวิดีโอเซนเซอร์ที่อยู่ในกล้องจะทำการแสกนหรือวัดผลรวมความเข้มแสงที่จุดต่างๆ ไปตามแนวแสกนที่เรียกว่า Raster Scan การแสกนแบบนี้จะมีทิศทางจากบนลงล่างและจากซ้ายไปขวา ภาพที่ได้จากการแสกนนั้นจะเป็นภาพแบบต่อเนื่องด้วยความเร็วทั่วไปที่ 24 ภาพต่อวินาที เช่นเดียวกันในเครื่องรับภาพวิดีโอก็จะรับภาพที่ได้มาจากเครื่องถ่ายวิดีโอ และแสดงผลโดยเริ่มจากบนลงล่างและจากซ้ายไปขวาเช่นเดียวกัน

แต่ภาพที่ได้มาจากระบบอะนาลอกนั้นยังเป็นภาพต่อเนื่องที่ยังไม่สามารถนำมาใช้เป็นการประมวลผลได้ ต้องมาทำการแปลงให้เป็นภาพเชิงตัวเลขเสียก่อนด้วยวิธีการ Digitization ซึ่งเป็การแปลงฟังก์ชันต่อเนื่อง $f(x,y)$ ให้เป็นฟังก์ชันไม่ต่อเนื่อง $g(x,y)$ เพื่อให้สามารถนำมาประมวลผลด้วยคอมพิวเตอร์ได้

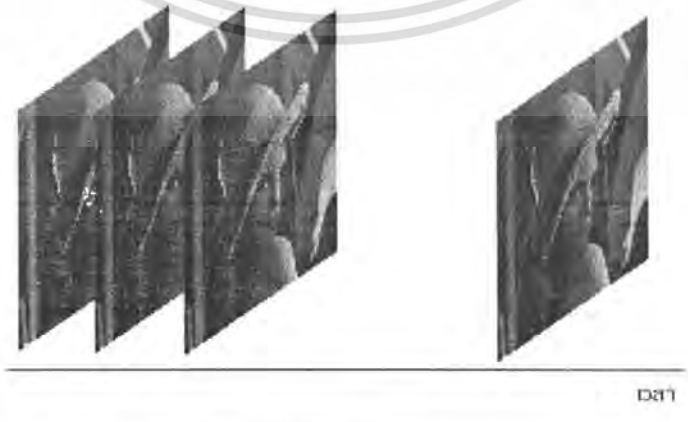


ภาพแบบต่อเนื่อง ภาพเชิงตัวเลข

รูปที่ 2.19 สัญญาณภาพแบบต่อเนื่องและเชิงตัวเลข

ถ้าพูดกันง่ายๆ การแปลงสัญญาณภาพจากภาพแบบต่อเนื่องมาเป็นภาพเชิงตัวเลขนั้นก็เหมือนกับวงจรแปลสัญญาณ Analog to Digital Converter หรือ A/D โดยการแบ่งฟังก์ชันต่อเนื่องออกเป็นช่วง ซึ่งค่าที่ได้นั้นจะเป็นจำนวนเต็มบวก โดยในแต่ละจุดภาพจะเป็นสมาชิกของแมทริกที่มีขนาด M แถว และ N หลัก ระดับของสีในภาพ 8 บิต จะมีค่าความเข้มเท่ากับ 2^8 หรือ 256 (0-255) โดยใช้ 8 บิตสำหรับเก็บข้อมูล

ภาพในแต่ละจุด หากต้องการให้ภาพที่ได้มีความละเอียดสูงๆนั้น จะต้องมีจำนวนบิตในการเก็บข้อมูลมากขึ้น เช่น 16, 24, 32 บิต แสดงว่าภาพจะมีความเข้มเท่ากับ 2^{16}, 2^{24}, 2^{32} ตามลำดับ หรือที่เราเซ็ทขนาดหน้าจอของวินโดวส์ว่าเป็น 24 บิต หรือ 32 บิต เป็นต้น



รูปที่ 2.20 วีดิโอสตรีม (video stream)

2.4.2 ความหมายของการประมวลผลภาพดิจิทัล

การประมวลผลภาพ เรียกอีกอย่างหนึ่งว่า Image Processing หมายถึง การเรียกใช้ขั้นตอนหรือกรรมวิธีใดๆมากระทำกับภาพ โดยมีวัตถุประสงค์เพื่อปรับปรุงคุณภาพของภาพให้ได้ภาพใหม่ที่มีคุณสมบัติตามที่ต้องการ เช่น ความคมชัด หรือ การประหยัดพื้นที่ในการเก็บข้อมูล หรือใช้สำหรับการประมวลระดับสูง เช่น การจดจำรูปร่างลักษณะให้ได้อย่างแม่นยำ แต่โดยทั่วไปแล้ววตถุประสงค์ของ Image Processing ก็คือ

- Image Processing : Image In ⟶ Image Out

วิธีการนี้จะใช้กระบวนการทาง DIP เพื่อทำให้ได้ภาพออกมา เช่น การตกแต่งภาพด้วยโปรแกรม Photoshop เป็นต้น

- Image Analysis : Image In ⟶ Measurements Out

วิธีการนี้จะชะกระบวนการทาง DIP เพื่อทำให้ได้ค่าการวัดออกมา เช่น การวัดขนาดในงานอุตสาหกรรม เป็นต้น

- Image Understanding : Image In ⟶ High-level Description Out

วิธีการนี้จะชะกระบวนการทาง DIP เพื่อให้ได้ผลลัพธ์ออกมาเป็นความหมาย ตัวอย่างของ High-level Description เช่น การจดจำตัวอักษร (Optical Character Recognition : OCR) เป็นต้น

ในส่วนของการประยุกต์ใช้งาน Digital Image Processing กับงานด้านอุตสาหกรรมนั้น นับได้ว่ามีความจำเป็นและมีความสำคัญเป็นอย่างมาก เนื่องจากในวงการอุตสาหกรรมโดยเฉพาะในสายการผลิตจะมีอัตราการผลิตที่สูงมาก เร็ว และต้องการความละเอียดแม่นยำสูง ตัวอย่างเช่น การผลิต Chip IC ซึ่งแต่ละตัวมีขนาดที่เล็กมาก เรียกว่าต้องใช้กล้อง Electron Microscope กันเลยทีเดียว และแน่นอนว่า ท่ความเล็กขนาดนี้คงไม่สามารถใช้สายตาของคนมาทำการตรวจสอบคุณภาพและขั้นตอนการผลิตได้ ในที่นี้จึงจำเป็นต้องใช้กล้องดิจิทัลเข้ามาช่วยในการตรวจจับภาพและนำภาพที่ได้มาทำการประมวลผล จากนั้นจึงนำผลที่ได้มาทำการวิเคราะห์ต่อไป โดยหลักแล้ว ก็จะสรุปได้ว่า งานที่ใช้ Digital Image Processing เข้ามาใช้ ได้แก่ งานที่มีจำนวนมาก งานที่มีขนาดเล็ก งานที่ใช้สำหรับส่งผ่านภาพ และงานอื่นตามแต่การประยุกต์ใช้งาน

การประมวลภาพด้วยคอมพิวเตอร์ สามารถทำได้โดยนำภาพที่ได้มาจากกล้องหรือ Image Source ต่างๆ ซึ่งเป็นสัญญาณอะนาลอก และนำมาแปลงเป็นสัญญาณดิจิทัสที่มีลักษณะที่เป็นรหัสเชิงตัวเลข 0, 1 ที่สามารถใช้รูปแบบทางคณิตศาสตร์เข้ามาช่วยในการคำนวณและการประมวนผลข้อมูลด้วยคอมพิวเตอร์ได้ต่อไป

2.4.3 ระดับของการประมวลผลภาพ

1. การประมวลผลภาพระดับต่ำ (Low Level Image Processing) เป็นการประมวลผลขั้นแรกสุดก่อนจะนำไปสู่การประมวลภาพระดับสูงต่อไป นั่นคือ หลังจากที่เราได้ภาพมา ภาพที่ได้จะประกอบไปด้วยองค์ประกอบต่างๆ มากมาย รวมถึงสิ่งที่เราไม่ต้องการด้วย ในที่นี้เราจะเรียกว่า Noise ซึ่งทำให้ภาพที่ได้มีคุณภาพที่ไม่ดี ยังไม่สามารถนำไปใช้ประมวลผลได้ ดังนั้น การประมวลผลภาพระดับต่ำจึงประกอบไปด้วยกากำจัดสัญญาณรบกวน การทำภาพให้ชัด (High Pass Filter) การหาขอบภาพ (Edge Detection) การแปลง Binary Image การแบ่งแยกรูปร่างวัตถุ (Image Segmentation) เป็นต้น เพื่อหาค่าตัวแปรต่างๆ มาอธิบายข้อมูลรูปภาพ และมีวัตถุประสงค์ที่จะนำตัวแปรเหล่านี้มาใช้ประมวลผลภาพระดับสูงต่อไป

2. การประมวลผลภาพระดับสูง (High Level Image Processing) เป็นการทำให้คอมพิวเตอร์รู้จักและเข้าใจภาพนั้นได้ เช่น การจดจำใบหน้าคน หรือ การจดจำตัวอักษร เป็นต้น ความแตกต่างของการประมวลผลภาพระดับต่ำและระดับสูง คือ ข้อมูลที่นำมาใช้ในการประมวลผล โดยการประมวลผลระดับต่ำจะใช้ค่าความสว่างหรือความเข้มแสงโดยตรง

ส่วนการประมวลผลระดับสูง ข้อมูลที่นำมาใช้ในการประมวลผลจะถูกแสดงในรูปของสัญลักษณ์ โดยสัญลักษณ์เหล่านี้จะแสดงถึงสิ่งต่างๆ ที่อยู่ในภาพ และการใช้ตัวแปรที่ได้จากการประมวลผลภาพระดับต่ำ มาอธิบายถึงสัญลักษณ์เหล่านี้ การประมวลผลภาพระดับสูงนั้น ส่วนใหญ่มักจะใช้ทฤษฎีต่างๆเข้ามาเป็นตัวช่วยในการทำงาน หรือเป็นหัวใจของโปรแกรม เช่น Fuzzy, Logic, Neural Network

2.4.4 การแทนภาพด้วยข้อมูลแบบดิจิทัล

จากที่ได้กล่าวมาแล้วว่า ข้อมูลภาพแบบดิจิตอลเป็นภาพที่ถูกดัดแปลงมาจากภาพต่อเนื่อง หรือ Analog Image ให้อยู่ในรูปเชิงตัวเลขหรือ Digital Image ด้วยวิธีการ Digitization โดยภาพอะนาลอกจะถูกแบ่งพื้นที่เป็นสี่เหลี่ยมเล็กที่เรียกว่า พิกเซล (Pixels) โดยในแต่ละพิกเซลจะใช้ *(x,y)* ในการระบุตำแหน่ง การแสดงข้อมูลภาพดิจิทัลสามารถอธิบายได้ด้วยแมทริก *(M*N)* และให้จุดต่างๆ ที่อยู่ในแมทริกเป็นจุดพิกัด *(x,y)* ใดๆ เป็นส่วนประกอบของภาพ

ค่าของพิกเซล ณ จุดใดๆ จะแสดงได้ด้วยค่าของความเข้มแสง ซึ่งอาจแบ่งได้หลายระดับ ถ้ามี 2 ระดับ ก็จะเป็นแค่ 0 กับ 1 จุดต่างๆที่อยู่ในพิกัดนี้ก็คือ พิกเซล หรือ Picture Element หรือก็คือ ความสว่าง (Luminance :L) ของภาพ ถ้าภาพนั้นเป็นภาพขาวดำ มีขนาด 8 บิต จะมี $L = 2^8$ or 256 คือตั้งแต่ระดับ 0-255 บางครั้งค่าความสว่างก็อาจจะหมายถึงความละเอียดของภาพ

2.4.5 ภาพที่นำมาใช้ในการประมวลผล

1. ภาพเคลื่อนไหว ซึ่งความจริงแล้วภาพเคลื่อนไหวก็คือภาพนิ่งที่นำมาแสดงต่อกันแบบต่อเนื่อง จะต้องใช้รูปภาพอย่างน้อย 24 รูปต่อวินาที เนื่องจากสายตาของคนเราเมื่อนำภาพนิ่งมาฉายติดต่อกันมากกว่า 24 รูปต่อวินาทีแล้วก็มองเห็นภาพนั้นเป็นภาพเคลื่อนไหว เพราะสายตาเราแยกไม่ออกเนื่องจากมีความเร็วมากเกินไป แต่หากนำภาพนิ่งมาฉายติดต่อกันน้อยกว่า 24 รูปต่อวินาทีแล้ว เราจะมองเห็นนั้นไม่ต่อเนื่อง

2. ภาพสีหรือภาพนิ่ง ภาพที่นำมาประมวลผลในคอมพิวเตอร์นั้น ภถ้าในระบบ RGB ก็จะใช้ความเข้มสีแดง เขียว และน้ำเงิน ความหมายของภาพนิ่งก็คือมีอยู่ภาพเดียว ภาพนิ่งที่นำมาใช้ก็มีอยู่หลาย Format เช่น

- .BMP เป็นไฟล์มาตรฐานที่ระบบปฏิบัติการ Windows สร้างขึ้นมา เป็นไฟล์ที่สามารถรักษาความละเอียดของภาพได้เป็นอย่างดี แต่มีข้อจำกัดคือ ไฟล์นั้นจะมีขนาดใหญ่นำมาใช้งานไม่สะดวก

- .JPG เป็นไฟล์ที่มีการบันทึกข้อมูลแบบสูญเสียข้อมูล ภาพที่ได้นำมาใช้งานทั่ว ๆ ไป ไฟล์ประเภทนี้จะตัดรายละเอียดของภาพบางส่วนออก ซึ่งเป็นรายละเอียดที่ไม่สามารถมองเห็นสีได้มากนักเหมาะสำหรับเก็บไว้ดูหรือนำไปลงอินเทอร์เน็ต

- .GIF เป็นไฟล์ที่มีการบีบอัดข้อมูลสูง แต่จะให้ความละเอียดของภาพมากกว่า ทำให้ไฟล์มีขนาดเล็กมาก มักนำมาใช้งานบนอินเทอร์เน็ตมากที่สุด เพราะไฟล์ที่มีขนาดเล็กทำให้ไม่เสียเวลาในการเปิดหน้าเว็บไซต์ที่มีรูปภาพประกอบได้ในเวลาอันรวดเร็ว

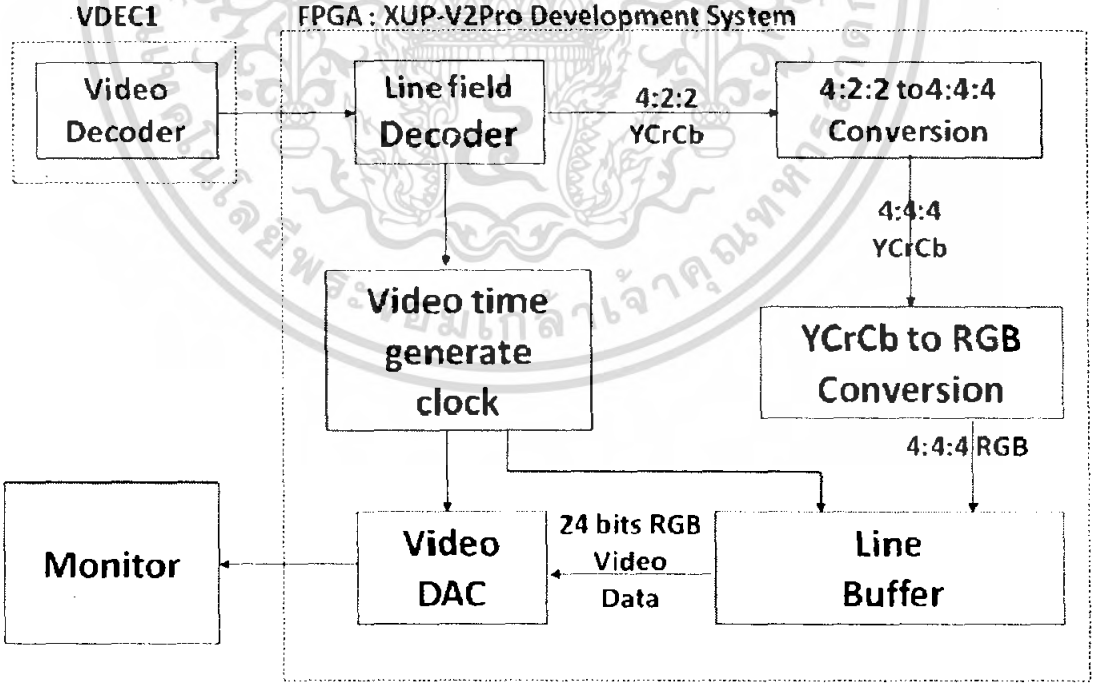
แต่ไม่ว่าจะเป็น .bmp .jpg หรือ .gif ในการเลือกใช้นั้นก็แล้วตามความเหมาะสม ส่วนใหญ่จะเป็น .bmp เพราะไม่ต้องมาถอดรหัสก่อน เนื่องจากภาพที่เป็น .jpg หรือ .gif นั้นมีการบีบอัดภาพให้มีขาดเล็ก ดังนั้น หากนำมาใช้ก็ต้องคลายข้อมูลออกมาก่อนที่จะนำภาพไปประมวลผลต่อไป

บทที่ 3
การออกแบบ และ การสร้าง

3.1 กล่าวนำ

ในการออกแบบและการสร้าง ***"ระบบควบคุมและเฝ้าระวังกล้องผ่านระบบเครือข่ายคอมพิวเตอร์ สำหรับเทคโนโลยีรักษาความปลอดภัย"*** จะประกอบไปด้วยส่วนของฮาร์ดแวร์และซอร์ฟแวร์ที่ใช้ในการควบคุมการทำงานของกล้องและอุปกรณ์ที่ใช้เชื่อมต่อกับระบบของโครงงานนี้ ที่ประกอบไปด้วย

- กล้องวิดีโอ
- ชุดควบคุมการหมุนตำแหน่งกล้อง
- บอร์ดควบคุม (FPGA)
- ระบบเครือข่ายคอมพิวเตอร์
- คอมพิวเตอร์

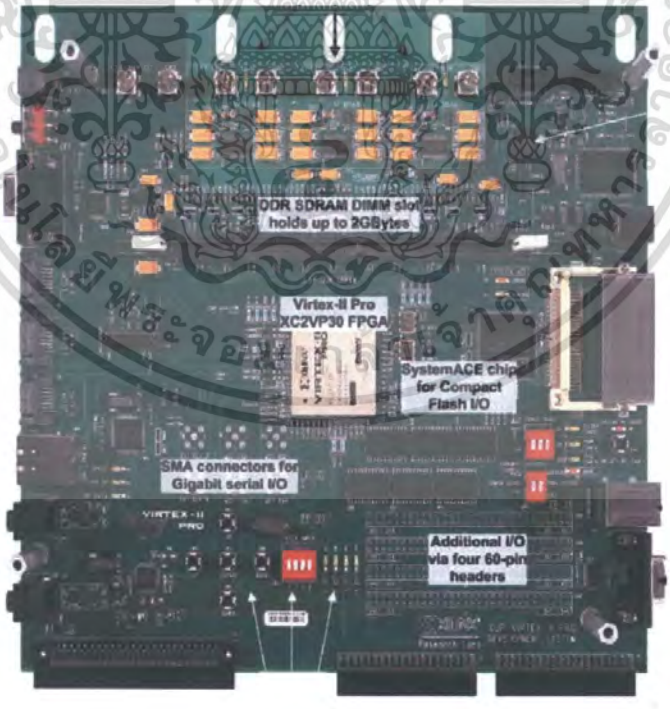


รูปที่ 3.1 บล็อกไดอะแกรมสำหรับการออกแบบ

3.2 บอร์ดควบคุมการรับ-ส่งสัญญาณภาพและควบคุมมอเตอร์

บอร์ดควบคุมการรับ-ส่งสัญญาณภาพและควบคุมมอเตอร์ จะใช้ทำหน้าที่ในการรับสัญญาณภาพจากกล้องวิดีโอและส่งภาพเข้าสู่ระบบเครือข่ายคอมพิวเตอร์ไปแสดงผลทางจอมอนิเตอร์ของเครื่องคอมพิวเตอร์ นอกจากนั้นยังใช้สั่งการทำงานของมอเตอร์ด้วยจากเครื่องคอมพิวเตอร์ไปยังฐานหมุนกล้อง ซึ่งจะเปรียบเสมือนตัวกลางในการสื่อสารกันระหว่างส่วนควบคุมและอุปกรณ์

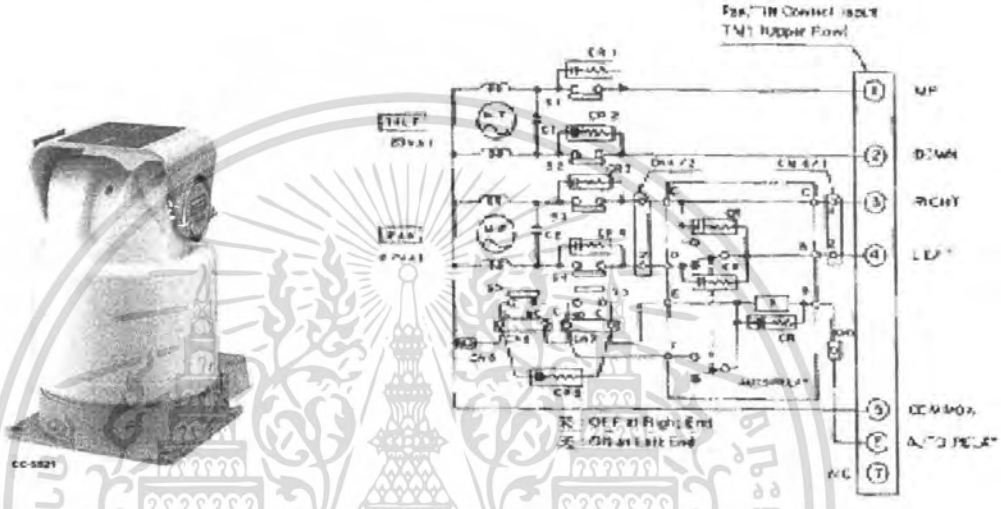
รูปที่ 3.2 บล็อกไดอะแกรมการเชื่อมต่อของบอร์ดควบคุม



รูปที่ 3.3 บอร์ดควบคุมที่ใช้ FPGA

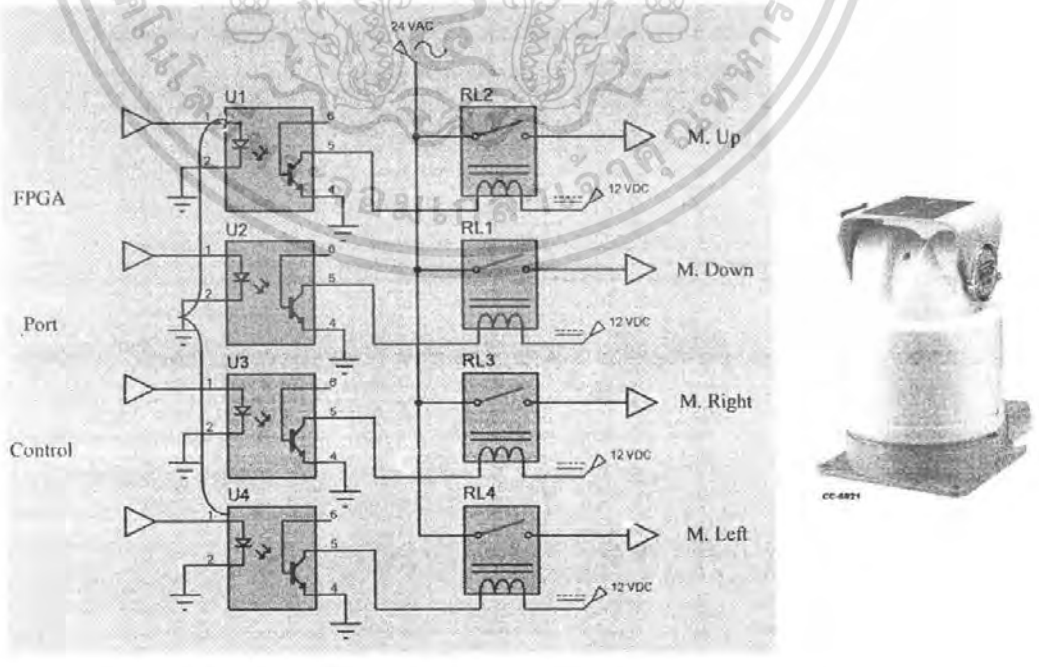
3.3 อุปกรณ์หมุนตำแหน่งกล้อง

อุปกรณ์ที่ใช้สำหรับหมุนตำแหน่งกล้อง เป็นตัว pan tilt head model CC-5521 (อาจเลือกใช้อุปกรณ์อื่นแทนได้) ที่สามารถหมุนได้ซ้าย-ขวา และขึ้น-ลง และมีฐานสำหรับใช้ติดตั้งกล้อง แรงดันไฟฟ้าที่ใช้ขับมอเตอร์ ใช้แรงดันไฟฟ้าขนาด 24 VAC



(ก) ลักษณะโครงสร้างของ CC-5521 (ข) วงจรขับมอเตอร์และจุดต่ออินพุต

รูปที่ 3.4 pan tilt head model CC-5521

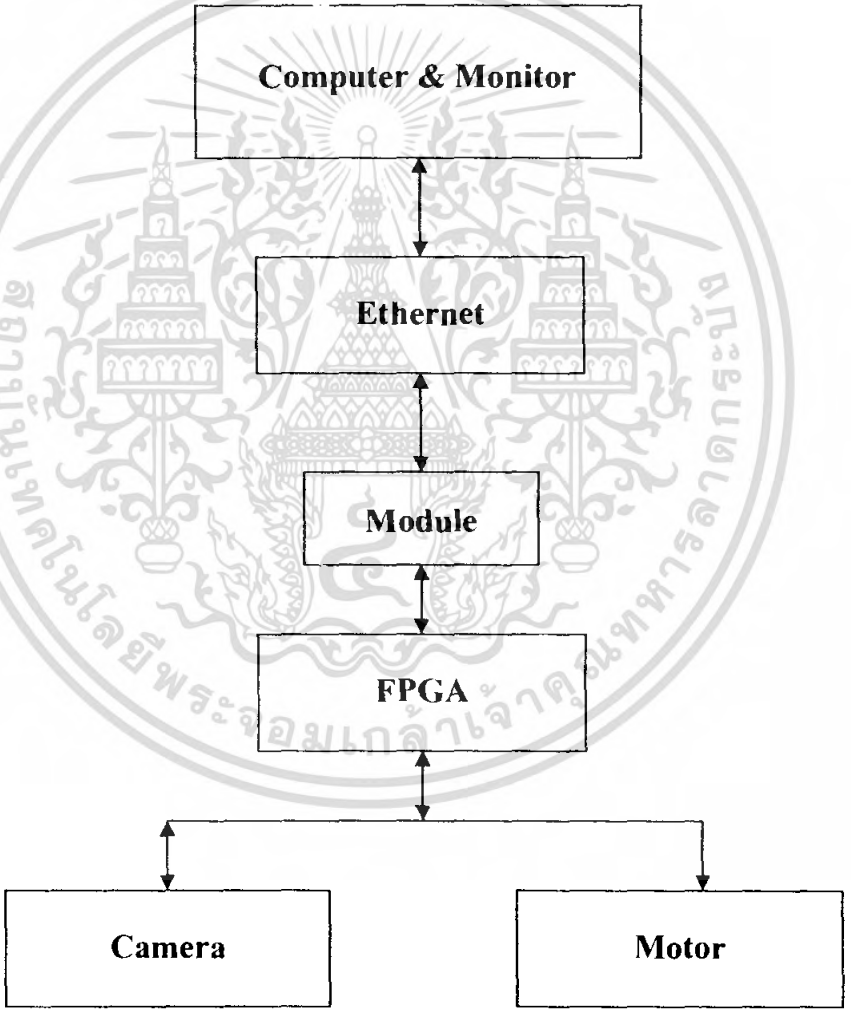


รูปที่ 3.5 วงจรเชื่อมต่อระหว่าง FPGA กับชุดควบคุมมอเตอร์

3.4 โปรแกรมการทำงานของระบบ

ในส่วนนี้จะเป็นการเขียนแปรแกรมด้วยภาษา VHDL สำหรับโปรแปรมให้กับ FPGA เพื่อให้ FPGA สามารถทำงานได้ 3 ส่วน คือ

1. ส่วนของการรับส่งภาพ
2. ส่วนของการควบคุมกล้อง
3. ส่วนของการเชื่อมต่ออีเทอร์เน็ต



รูปที่ 3.6 แผนผังการทำงานของระบบ

บทที่ 4

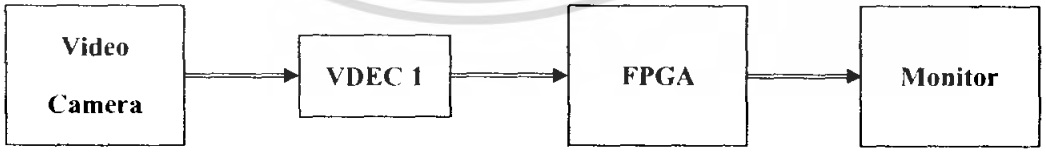
การทดลอง และ ผลการทดลอง

กล่าวนำ

จากที่ได้กล่าวมาแล้วในบทที่ 3 เกี่ยวกับการสร้างและการออกแบบ ทั้งทางด้านฮาร์ดแวร์และซอฟต์แวร์ ในบทนี้จะเป็นการทดลองและผลการทดลอง ตามที่ได้ออกแบบไว้ในแต่ละวงจรของการทดลองนั้นมีผลการทดลองที่ถูกต้องหรือไม่ และสามารถนำเอาอุปกร์หรือชุดวงจรนั้นๆ มาประยุกต์กับการทำโครงงานได้มากน้อยเพียงใด ดังนั้นในการทดลองจึงแยกย่อยออกเป็นการทดลองในส่วนต่างๆ

อุปกร์และเครื่องมือที่ใช้สำหรับการทดลอง

1. เครื่องคอมพิวเตอร์ สำหรับเขียนและทดสอบโปรแกรม
2. โปรแกรมที่ใช้สำหรับการจำลองการทำงานของแปรแกรมที่เขียนด้วยภาษา VHDL
   - โปรแกรม Xilinx ISE 9.2i
   - โปรแกรม ModelSim SE 6.2b
   - โปรแกรม Xilinx system generator ควบคู่กับโปรแกรม Matlab
3. อุปกรณ์ที่ใช้ในระบบ
   - กล้องวิดีโอ
   - คอมพิวเตอร์และจอแสดงผล
   - XUP Virtex-II Pro Development Board (FPGA)
   - Video Decoder 1 board (VDEC1)



รูปที่ 4.1 บล็อกไดอะแกรมแสดงการต่ออุปกรณ์สำหรับทดลอง

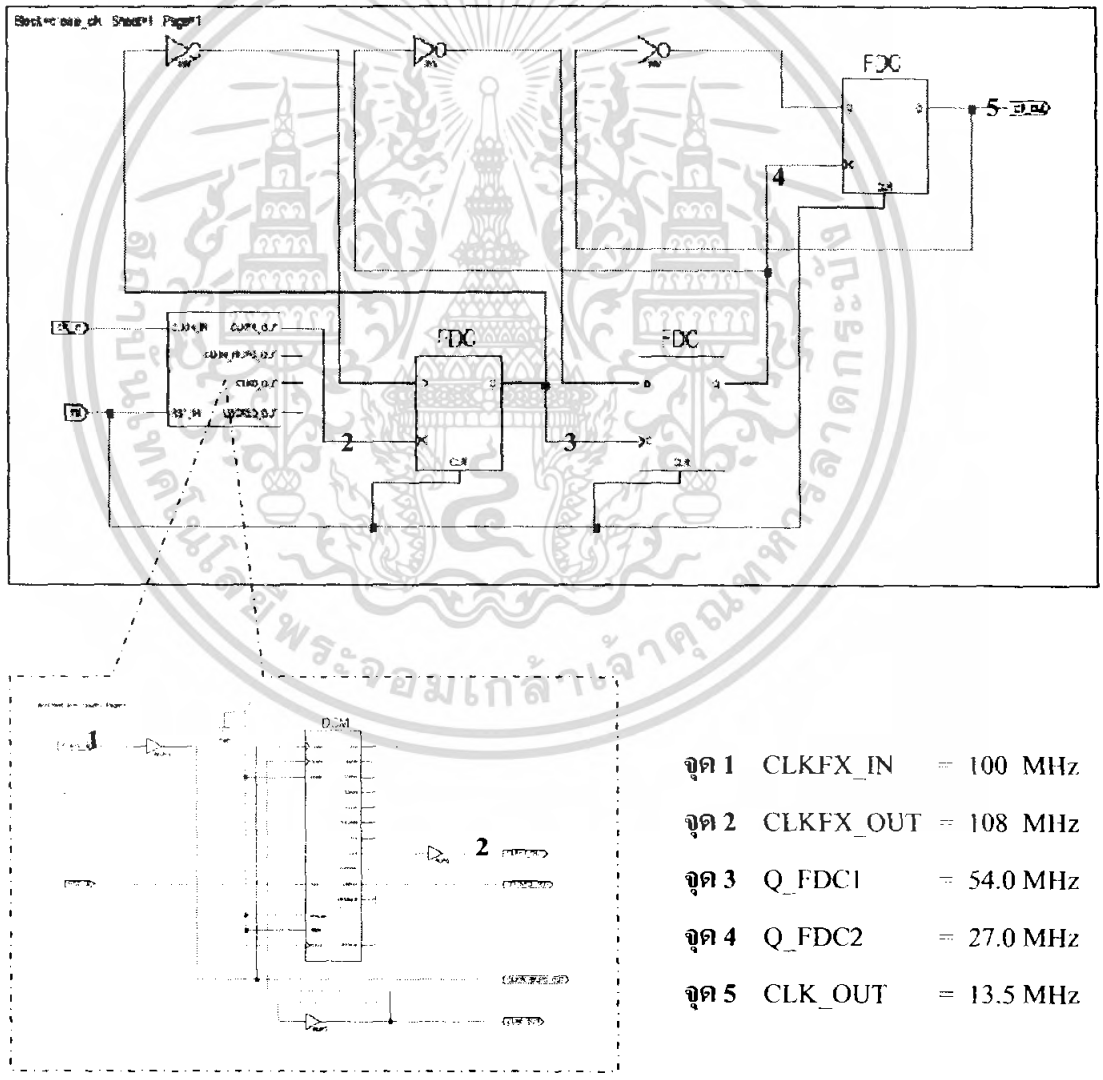
(* หมายเหตุ การทดลองดังต่อไปนี้เป็นส่วนเฉพาะการทดสอบโปรแกรมในคอมพิวเตอร์เท่านั้น*)

4.1 การสร้างสัญญาณนาฬิกาที่ความถี่ 13.5 MHz

การทดลองนี้ เป็นการจำลองการทำงานของโปรแกรมในส่วนของการสร้างสัญญาณ Clock 13.5 MHz จาก IP Core ของ FPGA สร้างชุด Digital Clock Management (DCM)

โปรแกรมที่ใช้ - โปรแกรม Xilinx ISE 9.2i

- โปรแกรม ModelSim SE 6.2b
- Source code ของโปรแกรมแสดงใน<u>ภาคผนวก B</u>



รูปที่ 4.2 กระบวนการทำงานของวงจร Digital Clock Management (DCM)

4.2 การแปลงสัญญาณจาก YCrCb เป็น RGB

การทดลองนี้ เป็นการจำลองการทำงานของโปรแกรมในส่วนของการแปลงสัญญาณจากสัญญาณ YCrCb เป็นสัญญาณ RGB ตามสมการ

$$R = Y + 1.40252 * (Cr - 0.5)$$

$$G = Y - 0.24642 * (Cr - 0.5) - 0.11840 * (Cb - 0.5)$$

$$B = Y + 1.77305 * (Cb - 0.5)$$

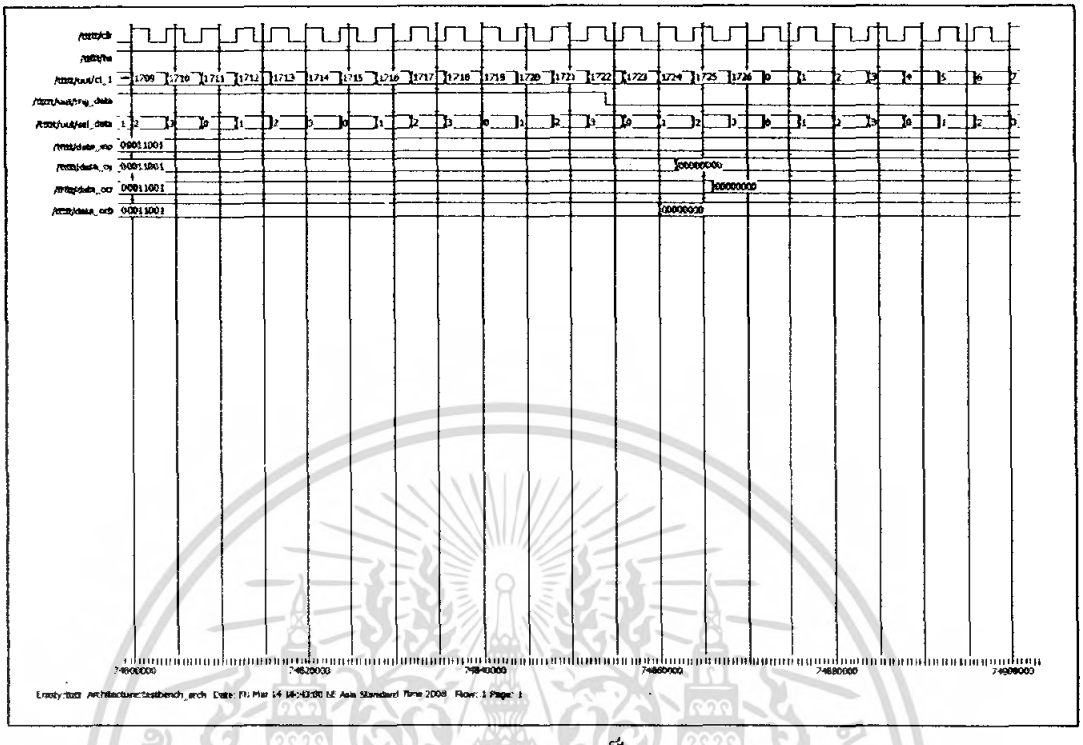
โดยการทดสอบโปรแกรม จะแสดงให้เห็นจำนวนสัญญาณ Clock ที่ใช้ คือ 1726 ลูก และการเริ่มต้นเก็บข้อมูลเมื่อสัญญาณ rng_data เป็น 1 โดยเว้น 2 clock ซึ่งเป็นส่วนของ Startbit และการหยุดเก็บข้อมูลเมื่อสัญญาณ rng_data เป็น 0 โดยเว้น 2 clock ก่อนถึงลูกสุดท้ายซึ่งเป็น Stopbit

โปรแกรมที่ใช้ - โปรแกรม Xilinx ISE 9.2i

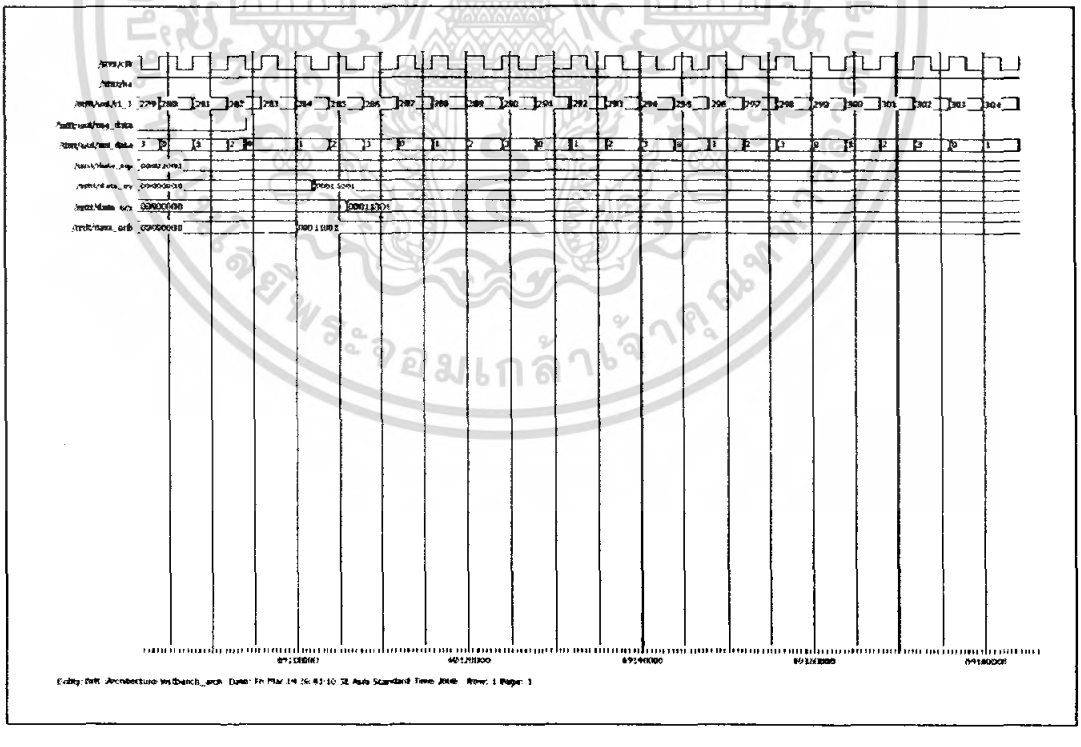
- โปรแกรม ModelSim SE 6.2b
- Source code ของโปรแกรมแสดงใน<u>ภาคผนวก D</u>

ผลการทดลอง

รูปที่ 4.3 สัญญาณ YCrCb แปลงเป็น สัญญาณ RGB



ก) rng_data เป็น 1



ข) rng_data เป็น 0

รูปที่ 4.4 สัญญาณ RGB OUT สัมพันธ์กับสัญญาณ Clock

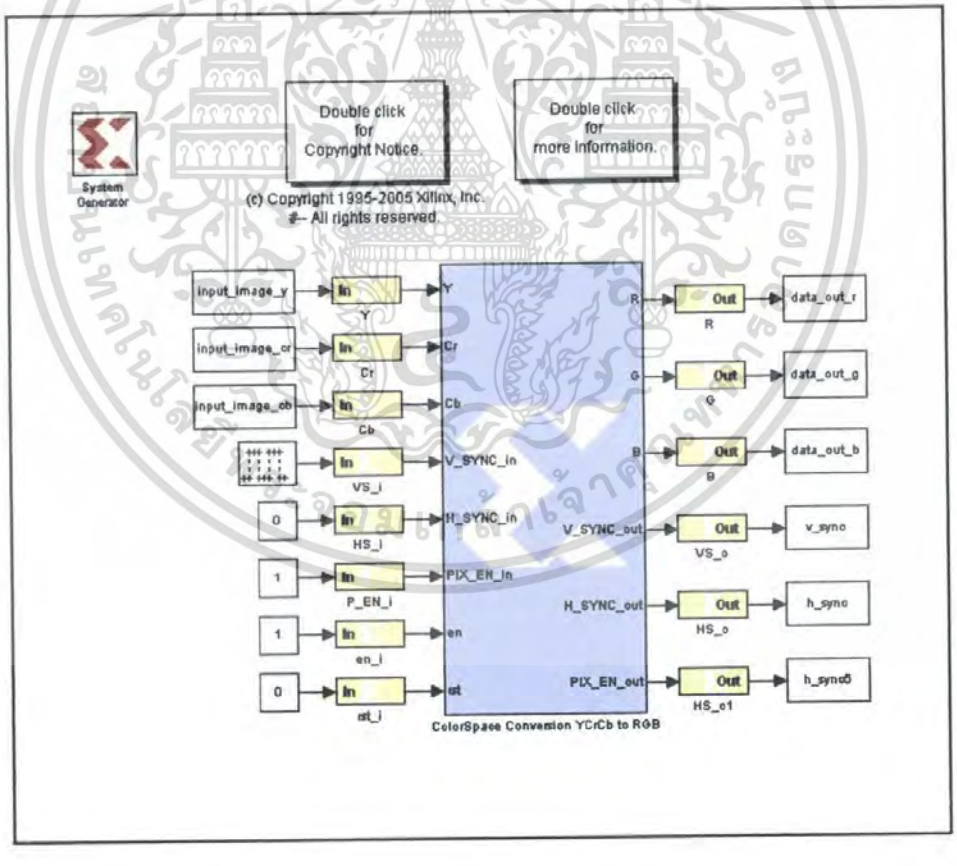
4.3 การแสดงผลลักษณะของสัญญาณภาพภาพ

การทดลองนี้ เป็นการจำลองการทำงานของโปรแกรมในส่วนของรูปสัญญาณด้วยโปรแกรม Xilinx System Generator ควบคู่กับโปรแกรม Matlab ซึ่งจะแสดงลักษณะสีของภาพตามโปรแกรมที่เขียน

ซึ่งจะเห็นได้ว่า สัญญาณสี RGB ที่ได้จาก Source code ของโปรแปรมที่เขียนขึ้น มีความเพี้ยนของสัญญาณสีเกิดขึ้น ซึ่งต้องแก้ไขในส่วนของ Source code

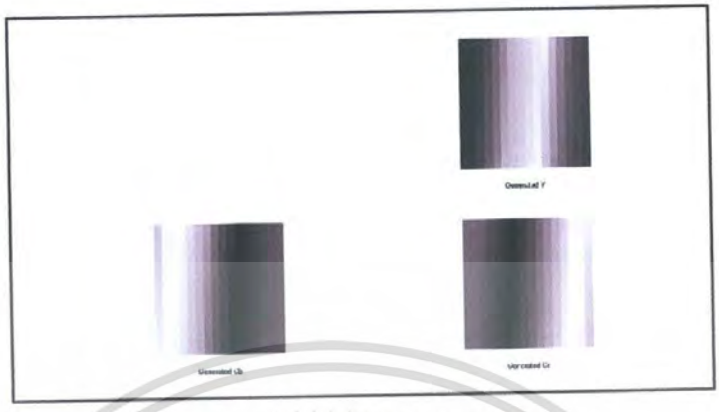
โปรแรมที่ใช้ - โปรแกรม Xilinx ISE 9.2i

- โปรแกรม Xilinx system generator
- โปรแกรม Matlab
- Source code ของโปรแกรมแสดงใน<u>ภาคผนวก F</u>

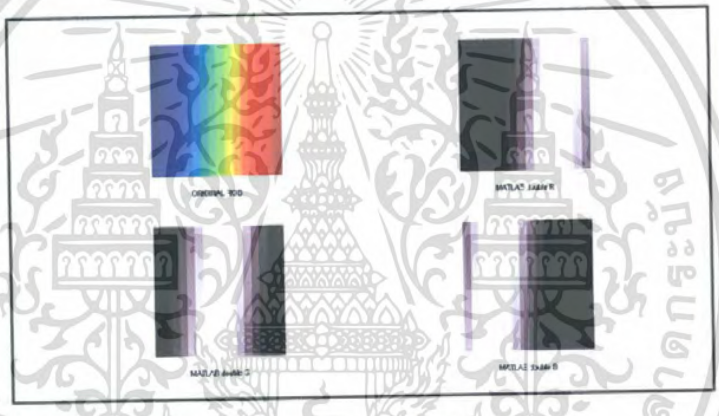


รูปที่ 4.5 การจำลองการทำงานด้วย Matlab

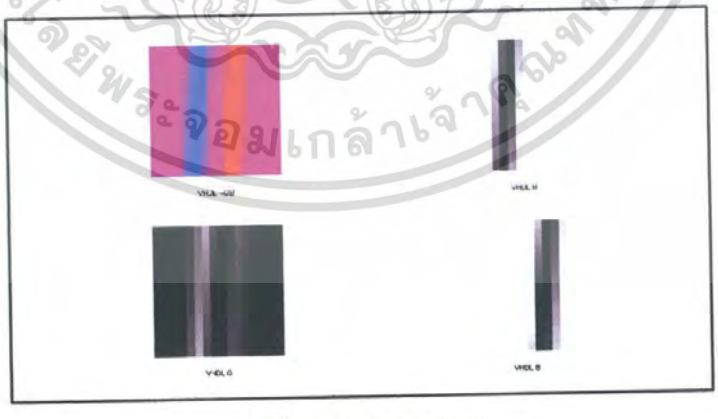
ผลการทดลอง



ก) สัญญาณ YCrCb Original



ข) สัญญาณ RGB Original



ค) สัญญาณ RGB OUT

รูปที่ 4.6 ลักษณะสีของสัญญาณภาพ

บทที่ 5
บทสรุป

สรุปและวิจารณ์ผลการทดลอง

การศึกษาการออกแบบและสร้างระบบรักษาความปลอดภัยด้วยกล้องที่สามารถทำงานในระบบเครือข่ายคอมพิวเตอร์ เพื่อสร้างชุดควบคุมการทำงานของกล้อง และชุดเชื่อมต่อสำหรับส่งสัญญาณข้อมูลภาพผ่านระบบเครือข่าย เพื่ออำนวยความสะดวกในการรักษาความปลอดภัย

จากการศึกษาและลงมือปฏิบัติโครงงานนี้ ทำให้เข้าใจรายละเอียดถึงคุณสมบัติและระบบการทำงานของกล้อง FPGA หลักการเขียนโปรแกรม การส่งผ่านข้อมูลในระบบเครือข่าย วงจรขับมอเตอร์ การเชื่อมต่อในระบบเครือข่าย และอุปกรณ์ต่าง ๆที่นำมาใช้ในการทำโครงงาน และในการศึกษาโครงงานดังกล่าวนี้มักประสบกับปัญหาและอุปสรรคต่างๆ จึงทำให้เกิดความรู้ ทักษะประสบการณ์ และแนวทางแก้ไขที่ทำให้สามารถแก้ปัญหานั้น ๆ ได้เป็นอย่างดี

นอกจากนี้ ในการทำงานกลุ่มยังเป็นการสร้างความสามัคคีภายในกลุ่มในการร่วมกันทำโครงงานและแก้ปัญหาหรืออุปสรรคต่างๆที่เกิดขึ้น ทำให้สามารถแก้ปัญหาได้รวดเร็วมากขึ้น และจากความสามัคคีในการทำงานร่วมกันทำให้การทำงานสำเร็จลุล่วงได้ตามวัตถุประสงค์ที่ตั้งเอาไว้

และจากการทำงานสร้างโครงงานนี้ ทำให้สามารถดูภาพจากกล้องและควบคุมการทำงานของกล้องผ่านระบบเครือข่ายได้

ปัญหาและแนวทางแก้ไข

ปัญหาที่เกิดขึ้น

1. ปัญหาที่เกิดจากสัญญาณนาฬิกาของอุปกรณ์ที่นำมาต่อกับ FPGA มีค่าคาบของเวลาสัญญาณไม่เท่ากัน อุปกรณ์แต่ละตัวจะใช้สัญญาณนาฬิกาคนละความถี่

2. การตราจสอบการเขยนโปรแกรมในส่วนของชุดแปลงสัญญาณ YCrCb เป็น RGB ไม่สามารถสร้างสัญญาณต้นแบบ และไม่ทราบสัญญาณเอาท์พุทเมื่อทำการแปลงสัญญาณแล้ว

แนวทางแก้ไขปัญหา

1. ปัญหาที่เกิดจากสัญญาณนาฬิกาของอุปกรณ์แต่ละชนิดที่นำมาใช้มีค่าไม่เท่ากัน สามารถแก้ปัญหาได้โดยการใช้ IP Core ของ FPGA สร้างชุด DCM (Digital clock management) เป็นตัวช่วยในการสร้างและใช้รูปแบบการเขียนโปรแกรมที่กำหนดที่ขอบขาการทำงานของสัญญาณนาฬิกาเพื่อเป็นการทำให้ได้สัญญาณนาฬกาที่ต้องการ

2. ปัญหาในการสร้างสัญญาณต้นแบบสำหรับจำลองการทำงานในส่วนของการเขียนโปรแกรมแปลงสัญญาณ YCrCb เป็น RGB สามารถทำได้โดยการใช้โปรแกรม Xilink system generator ควบคู่กับโปรแกรม Matlab เพื่อเป็นการสร้างสัญญาณและตรวจสอบสัญญาณ และสามารถใช้โปรแกรม Modelsim ในการมาตรวจสอบสัญญาณนาฬิกาของการทำงานในแต่ละส่วนของอิพุทและเอาท์พุทได้อีกด้วย

บรรณานุกรม

ชำนาญ ปัญญาใส และ วัชรากร หนูทอง. **ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น, 2547.

สิทธิโชค ยอดระยับ. **การเขียนโปรแกรม Digital Image Processing ด้วย Visual Basic.** กรุงเทพฯ : สถาบันส่งเสริมการสอนวิทยาศาสตร์และเทคโนโลยี, 2547.

Behrouz A. Forouzan. **การสื่อสารข้อมูลและเครือข่ายคอมพิวเตอร์.** แปลโดย จักกริช พฤษการ. กรุงเทพฯ : ท้อป, 2548.

Brown, Stephen. **Fundamental of digital logic with VHDL design.** Boston. McGraw Hill : c2000.

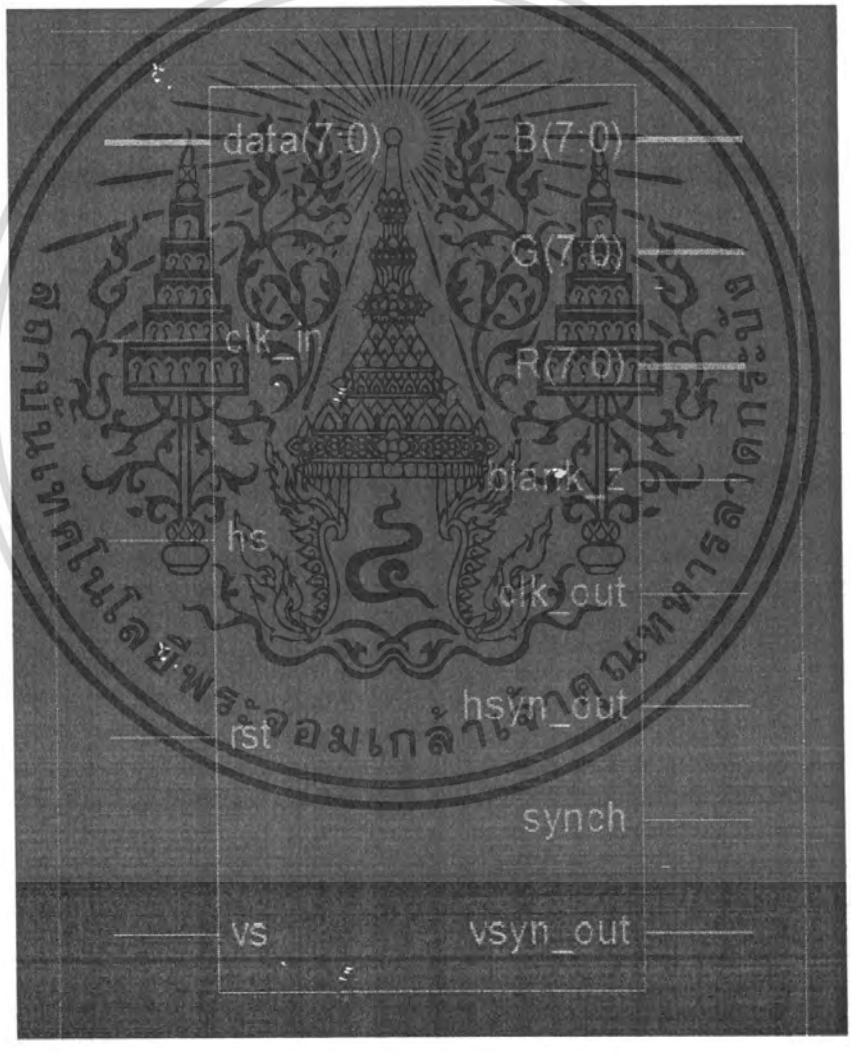
FPGA. (2550). Available URL : http://www.astronlogic.com/fpga.html.

CCD ปะทะ CMOS. (2550). Available URL : http://www.camerartmagazine.com/index.php?option=com_content&task=view&id=74&Itemid=39

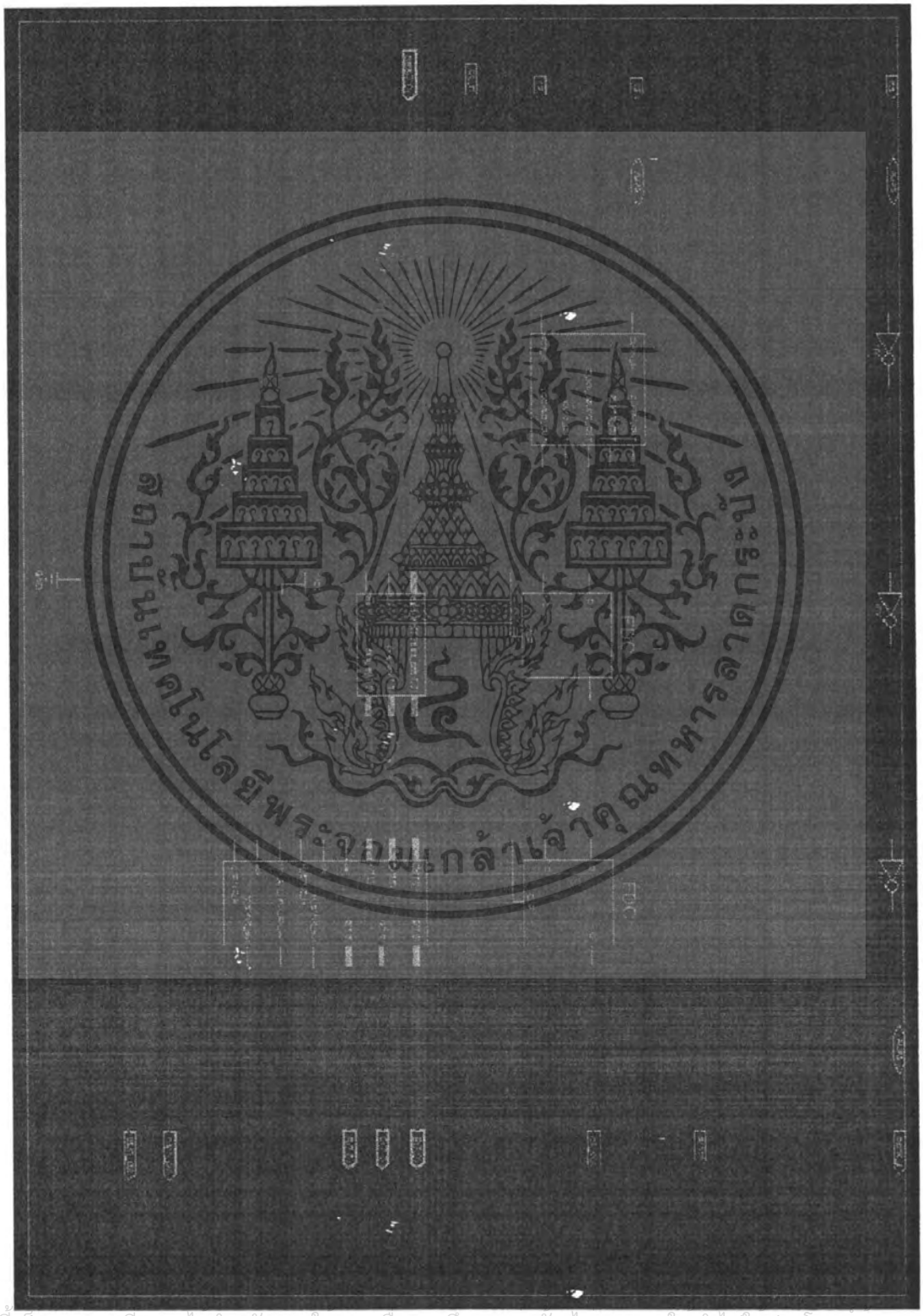
ภาคผนวก

ภาคผนวก A ===: รูปไดอะแกรมส่วนแปลงสัญญาณภาพ YCrCb เป็น RGB สำหรับเขียนโปรแกรมออกแบบ FPGA

Block diagram Outside…



Block diagram Inside…



ภาคผนวก B ===: Source code ของ dcm_clock.vhdl

(Digital Clock Management (DCM) สร้างสัญญาณนาฬิกาที่ความถี่ 13.5 Hz)

```
-- Filename : dcm_clock.vhd
--------------------------------------------------------------------------------
-- Module dcm_clock
-- Generated by Xilinx Architecture Wizard
-- Written for synthesis tool: XST
-- Period Jitter (unit interval) for block DCM_INST = 0.11 UI
-- Period Jitter (Peak-to-Peak) for block DCM_INST = 0.99 ns

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity dcm_clock is
  port ( CLKIN_IN       : in   std_logic;
         RST_IN         : in   std_logic;
         CLKDV_OUT      : out  std_logic;
         CLKFX_OUT      : out  std_logic;
         CLKIN_IBUFG_OUT : out  std_logic;
         CLK0_OUT       : out  std_logic;
         LOCKED_OUT     : out  std_logic);
end dcm_clock;

architecture BEHAVIORAL of dcm_clock is
  signal CLKDV_BUF      : std_logic;
  signal CLKFB_IN       : std_logic;
```

```
  signal CLKFX_BUF      : std_logic;
  signal CLKIN_IBUFG    : std_logic;
  signal CLK0_BUF       : std_logic;
  signal GND_BIT        : std_logic;

begin
  GND_BIT <= '0';
  CLKIN_IBUFG_OUT <= CLKIN_IBUFG;
  CLK0_OUT <= CLKFB_IN;
  CLKDV_BUFG_INST : BUFG
    port map (I=>CLKDV_BUF,
          O=>CLKDV_OUT);

  CLKFX_BUFG_INST : BUFG
    port map (I=>CLKFX_BUF,
          O=>CLKFX_OUT);

  CLKIN_IBUFG_INST : IBUFG
    port map (I=>CLKIN_IN,
          O=>CLKIN_IBUFG);

  CLK0_BUFG_INST : BUFG
    port map (I=>CLK0_BUF,
          O=>CLKFB_IN);

  DCM_INST : DCM
  generic map( CLK_FEEDBACK => "1X",
        CLKDV_DIVIDE => 2.0,
        CLKFX_DIVIDE => 25,
        CLKFX_MULTIPLY => 27,
```

```
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => 10.000,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=>CLKFB_IN,
          CLKIN=>CLKIN_IBUFG,
          DSSEN=>GND_BIT,
          PSCLK=>GND_BIT,
          PSEN=>GND_BIT,
          PSINCDEC=>GND_BIT,
          RST=>RST_IN,
          CLKDV=>CLKDV_BUF,
          CLKFX=>CLKFX_BUF,
          CLKFX180=>open,
          CLK0=>CLK0_BUF,
          CLK2X=>open,
          CLK2X180=>open,
          CLK90=>open,
          CLK180=>open,
          CLK270=>open,
          LOCKED=>LOCKED_OUT,
          PSDONE=>open,
          STATUS=>open);
end BEHAVIORAL;
```

ภาคผนวก C ===: Source code ของ Counter_hs.vhd

```
----------------------------------------------------------------------------------
-- Module Name:    counter_hs - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter_hs is
    Port ( hs : in  STD_LOGIC;
           clk : in  STD_LOGIC;
                  data_inp :in std_logic_vector(7 downto 0);
                     data_oy : out std_logic_vector(7 downto 0);
                     data_ocr : out std_logic_vector(7 downto 0);
                     data_ocb : out std_logic_vector(7 downto 0)
                     );
end counter_hs;

architecture Behavioral of counter_hs is

signal ct : integer range 0 to 1726:= 0;       -- clock cycle 1 frame 1728 - 1 hs clock cycle
signal inv_hs : std_logic;
signal st_en : std_logic;
```

```
signal rst : std_logic;
signal ct_int : integer range 0 to 1726 ;
signal en_int : std_logic;
signal rng_data : std_logic;
signal data_int : std_logic_vector(7 downto 0);
signal sel_data : integer range 0 to 3 ;
signal ct_1 : integer range 0 to 1726 ;

begin
  rst <= not hs ;
        process (clk)       -- loop 1 frame = 1728 clock cycle
        begin
          if (rst='1')then
          ct <= 0;
          elsif (rising_edge(clk))then
              if (ct = 1726) then
                    ct <= 0;
              else
                    ct <= ct + 1;
              end if;
          end if;
        end process;
        ct_1 <= ct;

        process (clk)       -- select range data clock cycle at 282-1722
          begin
            if (rst='1')then
                  rng_data <='0';
            elsif (falling_edge(clk))then
                  case ct is
```

```
                when 282 =>
                        rng_data <='1';
                when 1722=>
                        rng_data <= '0';
                when others =>
                        null;
            end case;
        end if;
end process;
st_en <= rng_data;

process (clk)          -- select data in range data at confix
    begin
     if (rst ='1') then
        data_int <= "00000000";
     elsif (falling_edge(clk))then
        if (st_en ='1')then
                data_int <= data_inp;
        else
                data_int <="00000000";
        end if;
    end if;
end process;

process (clk)
    begin
        if (ct_1 = 282) then
                sel_data <= 0;
        elsif (rising_edge(clk))then
                if (sel_data = 3)then
```

```
                    sel_data <= 0;
                else
                    sel_data <= sel_data + 1;
                end if;
            end if;
        end process;

        process (clk)
            begin
                case sel_data is
                    when 0 =>
                        data_ocb <= data_int;
                    when 1 =>
                        data_oy  <= data_int;
                    when 2 =>
                        data_ocr  <= data_int;
                    when others =>
                        null;
                end case;
        end process;

end Behavioral;
```

ภาคผนวก D ===: Source code ของ Xil_YCrCb2RGB.vhd

(Line Filed Detector สำหรับแปลง สัญญาณ YCrCb เป็น RGB)

```
-- ****************************************************************
-- *007*  YCrCb2RGB Macro from imagexlib
-- Description: Color Space Converter (YCrCb to RGB)
-- ****************************************************************
--LIBRARY genxlib;
--USE genxlib.genxlib_utils.ALL;
--LIBRARY mathxlib;
--USE mathxlib.mathxlib_utils.ALL;
--LIBRARY imagexlib;
--USE imagexlib.imagexlib_utils.ALL;

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

LIBRARY work;
USE work.color_space_pkg.all;

LIBRARY work;
USE work.genxlib_utils.ALL;

-- ****************************************************************
-- *009*  YCrCb2RGB Macro
-- Description: Color Space Converter (RGB to YCrCb)
-- Generalized conversion:
-- R = (Y - Yoffset) + ACoeff' * (Cr - Coffset)
```

```
-- G = (Y - Yoffset) + BCoeff' * (Cr - 0.5) + CCoeff' * (Cb - 0.5)
-- B = (Y - Yoffset) + DCoeff' * (Cb - 0.5)
-- R = Y + ACoeff' * Cr              - Roffset
-- G = Y + BCoeff' * Cr + CCoeff' * Cb - Goffset
-- B = Y + DCoeff' * Cb              - Boffset
-- In order to complement RGB2YCrCb:
-- ACoeff' = 1/CCOEFF
-- BCoeff' = ACOEFF/CCOEFF * (1-ACOEFF-BCOEFF)
-- CCoeff' = BCOEFF/DCOEFF * (1-ACOEFF-BCOEFF)
-- DCoeff' = 1/DCOEFF
-- Roffset = Yoffset + Acoeff' * Coffset
-- Goffset = Yoffset + (Bcoeff' + Ccoeff') * Coffset
-- Boffset = Yoffset + Dcoeff' * Coffset
-- ITU 601 (SDTV) standard:
-- if RGB data is between 0 and 255
-- R = Y + 1.40252 * (Cr - 0.5)
-- G = Y - 0.24642 * (Cr - 0.5) - 0.11840 * (Cb - 0.5)
-- B = Y + 1.77305 * (Cb - 0.5)
--
-- In order to better match the RGB2YCbCr module:
-- For R: ACoeff' = (1/CCOEF) value 2048/1460 is approximated instead of 1.40252
-- For B: DCoeff' = (1/DCOEF) value 2048/1155 is approximated instead of 1.77305

entity Xil_YCrCb2RGB is
  generic (
    FAMILY_HAS_MAC: integer:= 1;
    FABRIC_ADDS   : integer:= 1;      -- Adders are implemented using logic fabric based adders
    IWIDTH        : integer:= 8;
    CWIDTH        : integer:= 13;     -- Coefficients are signed, CWIDTH.CWIDTH-2 format
    MWIDTH        : integer:= 23;     -- ONLY FOR NON-V4: Controls bits witheld after mults.
```

```
  OWIDTH      : integer:= 8;          -- OTHERWISE (default) IWIDTH+CWIDTH+1;
  RGBMAX      : integer:= 255;
  RGBMIN      : integer:= 0;
  ACOEF       : integer:= 2872;       --  1.4023  *pow2(CWIDTH-2)
  BCOEF       : integer:= -1461;      --  -0.7133  *pow2(CWIDTH-2)
  CCOEF       : integer:= -703;       --  -0.3434  *pow2(CWIDTH-2)
  DCOEF       : integer:= 3630;       --  1.7724  *pow2(CWIDTH-2)
  ROFFSET     : integer:= -366592;    -- Should be MWIDTH bits wide
  GOFFSET     : integer:= 278016;
  BOFFSET     : integer:= -463616;
  HAS_CLIP    : integer:= 1;
  HAS_CLAMP   : integer:= 1);
 port (
  Y           : in std_logic_vector(IWIDTH-1 downto 0);      -- Y = a(R-G) + G + b(B-G)
  Cr          : in std_logic_vector(IWIDTH-1 downto 0);      -- Cr = d(R-Y)
  Cb          : in std_logic_vector(IWIDTH-1 downto 0);      -- Cb = c(B-Y)
  R           : out std_logic_vector(OWIDTH-1 downto 0);
  G           : out std_logic_vector(OWIDTH-1 downto 0);
  B           : out std_logic_vector(OWIDTH-1 downto 0);
  V_SYNC_in   : in std_logic := '0';
  H_SYNC_in   : in std_logic := '0';
  PIX_EN_in   : in std_logic := '1';
  V_SYNC_out  : out std_logic;
  H_SYNC_out  : out std_logic;
  PIX_EN_out  : out std_logic;
  clk         : in std_logic;
  ce          : in std_logic := '1';
  sclr        : in std_logic := '0');
end Xil_YCrCb2RGB;
```

```
architecture rtl of Xil_YCrCb2RGB is

-- High level constants
 constant MODULE_LATENCY : integer := YCrCb2RGB_LATENCY(FAMILY_HAS_MAC,
                                        FABRIC_ADDS, HAS_CLIP, HAS_CLAMP);
-- ADDER_DELAY is set to 1, MULT_DELAY is 2
 constant ACOEFvec      : std_logic_vector(CWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(ACOEF, CWIDTH);
--    ACOEF SRL ACOEFF_RANGE, CWIDTH); -- ACOEFF constant is normalized
 constant BCOEFvec      : std_logic_vector(CWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(BCOEF, CWIDTH);
--    BCOEF SRL BCOEFF_RANGE, CWIDTH); -- BCOEFF constant is normalized
 constant CCOEFvec      : std_logic_vector(CWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(CCOEF,CWIDTH);
 constant DCOEFvec      : std_logic_vector(CWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(DCOEF,CWIDTH);
 constant Goffsetvec    : std_logic_vector(MWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(GOFFSET,MWIDTH);
 constant Roffsetvec    : std_logic_vector(MWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(ROFFSET,MWIDTH);
 constant Boffsetvec    : std_logic_vector(MWIDTH-1 downto 0) :=
                                        conv_std_logic_vector(BOFFSET,MWIDTH);
 constant MAXvec        : std_logic_vector(OWIDTH+1 downto 0) :=
                                        conv_std_logic_vector(RGBMAX,OWIDTH+2);
 constant MINvec        : std_logic_vector(OWIDTH+1 downto 0) :=
                                        conv_std_logic_vector(RGBMIN,OWIDTH+2);

-- This is the delay of a virtex4 multiplier followed by a rounder. In order to facilitate
-- grouping the rounder with the mult into the same DSP48, overall latency must be 2
```

```
-- Low level constants
  constant logic0       : std_logic := '0';
  constant logic1       : std_logic := '1';

-- signal declarations
  signal Cr_delay       : std_logic_vector(IWIDTH downto 0);
  signal Cb_delay       : std_logic_vector(IWIDTH downto 0);
  signal Cb_unsign      : std_logic_vector(IWIDTH downto 0);
  signal Y_delay        : std_logic_vector(IWIDTH-1 downto 0);
  signal Y_padded       : std_logic_vector(OWIDTH downto 0);
  signal Acoef_by_Cr    : std_logic_vector(IWIDTH+CWIDTH  downto 0);
  signal Bcoef_by_Cr    : std_logic_vector(IWIDTH+CWIDTH downto 0);
  signal Ccoef_by_Cb    : std_logic_vector(IWIDTH+CWIDTH downto 0);
  signal Dcoef_by_Cb    : std_logic_vector(IWIDTH+CWIDTH downto 0);
  signal Acoef_by_Cr_rnd : std_logic_vector(MWIDTH downto 0);
  signal Bcoef_by_Cr_rnd : std_logic_vector(MWIDTH downto 0);
  signal Ccoef_by_Cb_rnd : std_logic_vector(MWIDTH downto 0);
  signal Dcoef_by_Cb_rnd : std_logic_vector(MWIDTH downto 0);
  signal G_int          : std_logic_vector(OWIDTH+1 downto 0);
  signal B_int          : std_logic_vector(OWIDTH+1 downto 0);
  signal R_int          : std_logic_vector(OWIDTH+1 downto 0);
  signal G_postmax      : std_logic_vector(OWIDTH+1 downto 0);
  signal B_postmax      : std_logic_vector(OWIDTH+1 downto 0);
  signal R_postmax      : std_logic_vector(OWIDTH+1 downto 0);
  signal G_postmin      : std_logic_vector(OWIDTH+1 downto 0);
  signal B_postmin      : std_logic_vector(OWIDTH+1 downto 0);
  signal R_postmin      : std_logic_vector(OWIDTH+1 downto 0);
  signal sync_in        : std_logic_vector(2 downto 0);
  signal sync_out       : std_logic_vector(2 downto 0);
```

```
begin
-------------------------------------------------------------------
-- Generate the output sync signals
-------------------------------------------------------------------
 SYNC_in(2) <= PIX_EN_in;
 SYNC_in(1) <= V_SYNC_in;
 SYNC_in(0) <= H_SYNC_in;

 del_SYNC : entity work.delay(rtl)
  generic map (
   width => 3,
   delay => MODULE_LATENCY )
  port map (
   clk  => clk,
   d    => SYNC_in,
   q    => SYNC_out,
   ce   => ce);

 PIX_EN_out <= SYNC_out(2);
 V_SYNC_out <= SYNC_out(1);
 H_SYNC_out <= SYNC_out(0);
-------------------------------------------------------------------
-- Create and round Cb*BCOEFF, Cb*DCOEFF, Cr*ACOEFF, Cr*CCOEFF
-------------------------------------------------------------------
 del_Cr : entity work.delay(rtl)          -- Delay Cr, so Acoef_by_Cr arrives in sync
  generic map (                           -- with Ccoef_by_Cb to the adder/rounder
   width => IWIDTH,
   delay => 1)
  port map (
   clk  => clk,
```

```
  d    => Cr,
  q    => Cr_delay(IWIDTH-1 downto 0),
  ce   => ce);

del_Cb : entity work.delay(rtl)          -- Delay Cb, so Dcoef_by_Cb arrives in sync
  generic map (                          -- with B_int and G_int are in sync.
    width => IWIDTH,
    delay => 1)
  port map (
    clk  => clk,
    d    => Cb,
    q    => Cb_delay(IWIDTH-1 downto 0),
    ce   => ce);

Cb_unsign(IWIDTH-1 downto 0) <= Cb;
Cb_unsign(IWIDTH) <= '0';                -- Making sure that Cb and Cr signals are
Cb_delay(IWIDTH) <= '0';                 -- interpreted as unsigned
Cr_delay(IWIDTH) <= '0';                 -- at the mutlipliers
sp3_v2_v2p: if (FAMILY_HAS_MAC=0) generate
  mult_aCr: entity work.mult(rtl)        -- ACOEFF * Cr
    generic map (
      IWIDTHA => IWIDTH+1,
      IWIDTHB => CWIDTH)
    port map (
      clk    => clk,
      ce     => ce,
      sclr   => sclr,
      a      => Cr_delay,
      b      => ACOEFvec,
      p      => Acoef_by_Cr);
```

```
mult_bCr: entity work.mult(rtl)          -- BCOEFF * Cr
 generic map (
  IWIDTHA => IWIDTH+1,
  IWIDTHB => CWIDTH)
 port map (
  clk    => clk,
  ce     => ce,
  sclr   => sclr,
  a      => Cr_delay,
  b      => BCOEFvec,
  p      => Bcoef_by_Cr);
mult_cCb: entity work.mult(rtl)          -- CCOEFF * Cb
 generic map (
  IWIDTHA => IWIDTH+1,
  IWIDTHB => CWIDTH)
 port map (
  clk    => clk,
  ce     => ce,
  sclr   => sclr,
  a      => Cb_unsign,
  b      => CCOEFvec,
  p      => Ccoef_by_Cb);
mult_dCb: entity work.mult(rtl)          -- DCOEFF * Cb
 generic map (
  IWIDTHA => IWIDTH+1,
  IWIDTHB => CWIDTH)
 port map (
  clk    => clk,
  ce     => ce,
  sclr   => sclr,
```

```
    a     => Cb_delay,
    b     => DCOEFvec,
    p     => Dcoef_by_Cb);

-- Rounding and offset compensating R with one adder
  round_aCr : entity work.radd_sub_sclr(rtl)
    generic map (
     width => MWIDTH,
     add   => true,
     fabric=> 1)
    port map (
     a     => Acoef_by_Cr(IWIDTH+CWIDTH-1 downto IWIDTH+CWIDTH-MWIDTH),
     b     => Roffsetvec,
     s     => Acoef_by_Cr_rnd,
     c_in  => logic0,
     clk   => clk,
     ce    => ce,
     sclr  => sclr);

 -- Adding and Rounding of Ccoef_by_Cr and Bcoef_by_Cb_rnd using one adder
round_bCr : entity work.radd_sub_sclr(rtl)
    generic map (
     width => MWIDTH,
     add   => true,
     fabric=> 1)
    port map (
     a     => Bcoef_by_Cr(IWIDTH+CWIDTH-1 downto IWIDTH+CWIDTH-MWIDTH),
     b     => Ccoef_by_Cb_rnd(MWIDTH-1 downto 0),
     s     => Bcoef_by_Cr_rnd,
     c_in  => logic0,
```

```
    clk   => clk,
    cc    => cc,
    sclr  => sclr);

-- Rounding and offset compensating G with one adder
round_cCb : entity work.radd_sub_sclr(rtl)
    generic map (
     width => MWIDTH,
     add   => true,
     fabric=> 1)
    port map (
     a    => Ccoef_by_Cb(IWIDTH+CWIDTH-1 downto IWIDTH+CWIDTH-MWIDTH),
     b    => Goffsetvec,
     s    => Ccoef_by_Cb_rnd,
     c_in => logic0,
     clk  => clk,
     ce   => ce,
     sclr => sclr);

-- Rounding and offset compensating G with one adder
   round_dCb : entity work.radd_sub_sclr(rtl)
     generic map (
      width => MWIDTH,
      add   => true,
      fabric=> 1)
     port map (
      a    => Dcoef_by_Cb(IWIDTH+CWIDTH-1 downto IWIDTH+CWIDTH-MWIDTH),
      b    => Boffsetvec,
      s    => Dcoef_by_Cb_rnd,
      c_in => logic0,
```

```
    clk  => clk,
    cc   => ce,
    sclr => sclr);
end generate;

v4: if (FAMILY_HAS_MAC = 1) generate        -- DSP48 based implementation
 mult_aCr: entity work.mac(rtl)             -- ACOEFF * Cr + Roffsetvec
  generic map (                             -- offset contains rounding const
   IWIDTHA  => IWIDTH+1,
   IWIDTHB  => CWIDTH,
   OWIDTH   => MWIDTH,
   ROUND_MODE=> 0,
   HAS_C    => 1)
  port map (
   clk      => clk,
   ce       => ce,
   sclr     => sclr,
   a        => Cr_delay,
   b        => ACOEFvec,
   c        => Roffsetvec,
   p        => Acoef_by_Cr_rnd(MWIDTH downto 1));
 -- sign extension for simulation (v4 results are the same as s3_v2_v2p)
 --Acoef_by_Cr_rnd(IWIDTH+CWIDTH) <= Acoef_by_Cr_rnd(IWIDTH+CWIDTH-1);

 mult_BCr: entity work.mac(rtl)             -- BCOEFF * Cr + CCOEFF * Cb + Goffsetvec
  generic map (
   IWIDTHA  => IWIDTH+1,
   IWIDTHB  => CWIDTH,
   OWIDTH   => MWIDTH,
   ROUND_MODE=> 0,
```

```
 CREG     => 0,                          -- use Pcascade chain
 HAS_C    => 1)
port map (
 clk      => clk,
 ce       => ce,
 sclr     => sclr,
 a        => Cr_delay,
 b        => BCOEFvec,
 c        => Ccocf_by_Cb_rnd(MWIDTH downto 1),
 p        => Bcoef_by_Cr_rnd(MWIDTH downto 1));
-- sign extension for simulation (v4 results are the same as s3_v2_v2p)
--Bcoef_by_Cr_rnd(IWIDTH+CWIDTH) <= Bcoef_by_C._rnd(IWIDTH+CWIDTH-1);

mult_cCb: entity work.mac(rtl)              -- CCOEFF * Cb + Goffsetvec
 generic map (                              -- offset contains rounding const
  IWIDTHA  => IWIDTH+1,
  IWIDTHB  => CWIDTH,
  OWIDTH   => MWIDTH,
  ROUND_MODE=> 0,
  HAS_C    => 1)
 port map (
  clk      => clk,
  ce       => ce,
  sclr     => sclr,
  a        => Cb_unsign,
  b        => CCOEFvec,
  c        => Goffsetvec,
  p        => Ccoef_by_Cb_rnd(MWIDTH downto 1));
-- sign extension for simulation (v4 results are the same as s3_v2_v2p)
--  Ccoef_by_Cb_rnd(IWIDTH+CWIDTH) <= Ccoef_by_Cb_rnd(IWIDTH+CWIDTH-1);
```

```
mult_DCb: entity work.mac(rtl)                -- DCOEFF * Cb + Boffsetvec
 generic map (                                -- offset contains rounding const
  IWIDTHA  => IWIDTH+1,
  IWIDTHB  => CWIDTH,
  OWIDTH   => MWIDTH,
  ROUND_MODE=> 0,
  HAS_C    => 1)
 port map (
  clk      => clk,
  ce       => ce,
  sclr     => sclr,
  a        => Cb_delay,
  b        => DCOEFvec,
  c        => Boffsetvec,
  p        => Dcoef_by_Cb_rnd(MWIDTH downto 1));
-- sign extension for simulation (v4 results are the same as s3_v2_v2p)
--Dcoef_by_Cb_rnd(IWIDTH+CWIDTH) <= Dcoef_by_Cb_rnd(IWIDTH+CWIDTH-1);

--  Acoef_by_Cr_rnd(MWIDTH) <= Acoef_by_Cr_rnd(MWIDTH-1);
--  Bcoef_by_Cr_rnd(MWIDTH) <= Bcoef_by_Cr_rnd(MWIDTH-1);
--  Ccoef_by_Cb_rnd(MWIDTH) <= Ccoef_by_Cb_rnd(MWIDTH-1);
--  Dcoef_by_Cb_rnd(MWIDTH) <= Dcoef_by_Cb_rnd(MWIDTH-1);

 Acoef_by_Cr_rnd(0) <= '0';
 Bcoef_by_Cr_rnd(0) <= '0';
 Ccoef_by_Cb_rnd(0) <= '0';
 Dcoef_by_Cb_rnd(0) <= '0';
end generate;
```

```
-- Add Y component
---------------------------------------------------------------------
 del_Y : entity work.delay(rtl)  -- Delay matching: y is delayed so it can be combined with
rounded signals
   generic map (
    width => IWIDTH,
    delay => 4) -- 3+FAMILY_HAS_MAC -- ADD_DELAY(FAMILY_HAS_MAC,
FABRIC_ADDS)+MULT_DELAY(FAMILY_HAS_MAC)
   port map (
    clk   => clk,
    d     => Y,
    q     => y_delay,
    ce    => ce);
 connect_Y: if (IWIDTH=OWIDTH) generate
  Y_padded(IWIDTH-1 downto 0) <= y_delay;
 end generate;
 padd_Y: if (IWIDTH<OWIDTH) generate
  Y_padded(OWIDTH-1 downto OWIDTH-IWIDTH) <= y_delay;
  Y_padded(OWIDTH-IWIDTH-1 downto 0) <= (others => '0');
 end generate;
 truncate_Y: if (IWIDTH>OWIDTH) generate
  Y_padded(OWIDTH-1 downto 0) <= y_delay(IWIDTH-1 downto IWIDTH-OWIDTH);
 end generate;
 Y_padded(OWIDTH) <= '0'; -- Makes sure Y_padded is unsigned positive


add_R : entity work.radd_sub_sclr(rtl)
   generic map (
    width     => OWIDTH+1,
    add       => true,
    a_signed  => true,
```

```
  b_signed  => false,
  delay     => 2-FABRIC_ADDS,
  fabric    => FABRIC_ADDS)
 port map (
  clk   => clk,
  a     => Acoef_by_Cr_rnd(MWIDTH-2 downto MWIDTH-OWIDTH-2),
  b     => Y_padded,
  s     => R_int,
  c_in  => logic0,
  ce    => ce,
  sclr  => sclr);

add_G : entity work.radd_sub_sclr(rtl)
 generic map (
  width     => OWIDTH+1,
  add       => true,
  a_signed  => true,
  b_signed  => false,
  delay     => 2-FABRIC_ADDS,
  fabric    => FABRIC_ADDS)
 port map (
  clk   => clk,
  a     => Bcoef_by_Cr_rnd(MWIDTH-2 downto MWIDTH-OWIDTH-2),
  b     => Y_padded,
  s     => G_int,
  c_in  => logic0,
  ce    => ce,
  sclr  => sclr);

add_B : entity work.radd_sub_sclr(rtl)
```

```
  generic map (
   width    => OWIDTH+1,
   add      => true,
   a_signed => true,
   b_signed => false,
   delay    => 2-FABRIC_ADDS,
   fabric   => FABRIC_ADDS)
  port map (
   clk  => clk,
   a    => Dcoef_by_Cb_rnd(MWIDTH-2 downto MWIDTH-OWIDTH-2),
   b    => Y_padded,
   s    => B_int,
   c_in => logic0,
   ce   => ce,
   sclr => sclr);

-- clipping and clamping of R,G,B
-----------------------------------------------------
  clip: if (HAS_CLIP=1) generate
   max_R : entity work.max_sat(rtl)        -- Add the logic to catch overflow saturation (max)
    generic map (width => OWIDTH+2)
    port map (
     a    => R_int,
     max  => MAXvec,
     ma   => R_postmax,
     clk  => clk,
     ce   => ce,
     sclr => sclr);
   max_G : entity work.max_sat(rtl)        -- Add the logic to catch overflow saturation (max)
    generic map (width => OWIDTH+2)
```

```
    port map (
    a    => G_int,
    max  => MAXvec,
    ma   => G_postmax,
    clk  => clk,
    ce   => ce,
    sclr => sclr);
  max_B : entity work.max_sat(rtl)        -- Add the logic to catch overflow saturation (max)
   generic map (width => OWIDTH+2)
   port map (
    a    => B_int,
    max  => MAXvec,
    ma   => B_postmax,
    clk  => clk,
    ce   => ce,
    sclr => sclr);
end generate;
no_clip: if (HAS_CLIP/=1) generate
  R_postmax <= R_int;
  G_postmax <= G_int;
  B_postmax <= B_int;
end generate;
clamp: if (HAS_CLAMP=1) generate
  min_R : entity work.min_sat(rtl)        -- Add the logic to catch underflow saturation (min)
   generic map (width => OWIDTH+2)
   port map (
    a    => R_postmax,
    min  => MINvec,
    ma   => R_postmin,
    clk  => clk,
```

```
    ce   => ce,
    sclr  => sclr);
  min_G : entity work.min_sat(rtl)          -- Add the logic to catch underflow saturation (min)
   generic map (width => OWIDTH+2)
   port map (
    a    => G_postmax,
    min   => MINvec,
    ma    => G_postmin,
    clk   => clk,
    ce    => ce,
    sclr  => sclr);
  min_B : entity work.min_sat(rtl)          -- Add the logic to catch underflow saturation (min)
   generic map (width => OWIDTH+2)
   port map (
    a    => B_postmax,
    min   => MINvec,
    ma    => B_postmin,
    clk   => clk,
    ce    => ce,
    sclr  => sclr);
 end generate;
 no_clamp: if (HAS_CLAMP/=1) generate
  R_postmin <= R_postmax;
  G_postmin <= G_postmax;
  B_postmin <= B_postmax;
 end generate;
 R <= R_postmin(OWIDTH-1 downto 0);
 G <= G_postmin(OWIDTH-1 downto 0);
 B <= B_postmin(OWIDTH-1 downto 0);
end rtl;
```

ภาคผนวก E ===: Source code ของ GenXlib_arch.vhd

```
-- Filename - GenXlib_arch.vhd
--------------------------------------------------------------------------------
-- This delay module does not have SCLR input pin: suitable for delay matching data-path components
-- where SCLR is not critical. Having no SCLR allows SRL16 based implementation in S, V, V2, V2P and V4.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity delay is
 generic (
  width : integer :=16;
  delay : integer :=8;
  vector: integer :=1);
 port (
  clk  : in std_logic;
  ce   : in std_logic;
  d1   : in std_logic := '0';
  q1   : out std_logic;
  d    : in std_logic_vector(width-1 downto 0) := (others => '0');
  q    : out std_logic_vector(width-1 downto 0));
end delay;

architecture RTL of delay is
 constant zeros : std_logic_vector(width-1 downto 0) := (others => '0');
 signal d_i: std_logic_vector(width-1 downto 0);
 signal q_i: std_logic_vector(width-1 downto 0);
 begin
  vect: if (vector=1) generate
```

```
  d_i <= d;
  q   <= q_i;
 end generate;

 sgnal: if (vector/=1) generate
  d_i(0) <= d1;
  q1     <= q_i(0);
 end generate;

 connect: if (delay<1) generate
  q_i <= d_i;
 end generate;

 needs_delay: if (delay>0) generate
  clk_process: process(clk)
   type delay_array is array (delay downto 1) of std_logic_vector(width-1 downto 0);
   variable shift_register : delay_array := (others => zeros);
   begin
    if (clk'event and clk = '1') then
     if (cc = '1') then
      for i in delay-1 downto 1 loop
       shift_register(i+1) := shift_register(i);
      end loop;
     shift_register(1) := d_i;
     end if;
    end if;
   q_i <= shift_register(delay);
   end process;
 end generate;
end RTL;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

DSP48.

entity delay_sclr is
 generic (
  width : integer :=16;
  delay : integer :=1 );
 port (
  clk   : in std_logic;
  ce    : in std_logic;
  sclr  : in std_logic;
  d     : in std_logic_vector(width-1 downto 0);
  q     : out std_logic_vector(width-1 downto 0));
end delay_sclr;

architecture RTL of delay_sclr is
constant zeros : std_logic_vector(width-1 downto 0) := (others => '0');
begin

 connect: if (delay<1) generate
  q <= d;
 end generate;

 needs_delay: if (delay>0) generate
  clk_process: process(clk)
   type delay_array is array (delay downto 1) of std_logic_vector(width-1 downto 0);
   variable shift_register : delay_array := (others => zeros);
   begin
```

```
    if (clk'event and clk = '1') then
      if (sclr = '1') then shift_register(delay downto 1) := (others => zeros);
       elsif (ce = '1') then
       for i in delay-1 downto 1 loop
        shift_register(i+1) := shift_register(i);
       end loop;
      shift_register(1) := d;
     end if;
   end if;

  q <= shift_register(delay);
  end process;
 end generate;
end RTL;

-- ***********************************************
-- Module can be implemented in DSP48
-- ***********************************************
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

-- This radd_sub_sclr module will be implemented in DSP48.
entity radd_sub_sclr_yes is
 generic (
  width   : integer := 16;
  delay   : integer := 1;   -- This parameter allows the synthesser to break the
                            -- carry chain if delay is >1. In DSP48s, it allows
  add     : boolean :=true; -- registering A|B and C input ports.
```

```
  a_signed: boolean :=true; -- 1 when operand 'a' is signed
  b_signed: boolean :=true);-- 1 when operand 'b' is signed
 port (
  clk   : in std_logic;
  ce    : in std_logic;
  c_in  : in std_logic;
  a     : in std_logic_vector(width-1 downto 0);
  b     : in std_logic_vector(width-1 downto 0);
  s     : out std_logic_vector(width downto 0);
  sclr  : in std_logic);

 attribute register_balancing: string;
 attribute register_balancing of radd_sub_sclr_yes: entity is "yes";
 attribute use_dsp48: string;
 attribute use_dsp48 of radd_sub_sclr_yes: entity is "yes";

end radd_sub_sclr_yes;

architecture rtl of radd_sub_sclr_yes is
 signal c    : std_logic_vector(width downto 0);
 signal a_ext: std_logic := '0';
 signal b_ext: std_logic := '0';
begin

 sgn_a: if a_signed generate a_ext <= a(width-1); end generate;
 sgn_b: if b_signed generate b_ext <= b(width-1); end generate;

 adder: if add     generate c <= (a_ext & a) + (b_ext & b) + c_in; end generate;
 subtr: if not add generate c <= (b_ext & b) - (a_ext & a) - c_in; end generate;
```

```
 reg : entity work.delay_sclr(rtl)
   generic map ( width => width+1, delay => delay)
   port map ( clk => clk, ce => ce, sclr => sclr, d => c, q => s);

end rtl;

-- This radd_sub_sclr module will be implemented in fabric.
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity radd_sub_sclr_no is
  generic (
    width   : integer :=16;
    delay   : integer := 1;   -- This parameter allows the synthesser to break the
                              -- carry chain if delay is >1. In DSP48s, it allows
    add     : boolean :=true; -- registering A|B and C input ports.
    a_signed: boolean :=true; -- 1 when operand 'a' is signed
    b_signed: boolean :=true);-- 1 when operand 'b' is signed
  port (
    clk   : in std_logic;
    ce    : in std_logic;
    c_in  : in std_logic;
    a     : in std_logic_vector(width-1 downto 0);
    b     : in std_logic_vector(width-1 downto 0);
    s     : out std_logic_vector(width downto 0);
    sclr  : in std_logic);

  attribute register_balancing: string;
```

```
 attribute register_balancing of radd_sub_sclr_no: entity is "yes";
 attribute use_dsp48: string;
 attribute use_dsp48 of radd_sub_sclr_no: entity is "no";
end radd_sub_sclr_no;

architecture rtl of radd_sub_sclr_no is
 signal c    : std_logic_vector(width downto 0);
 signal a_ext: std_logic := '0';
 signal b_ext: std_logic := '0';
begin
 sgn_a: if a_signed generate a_ext <= a(width-1); end generate;
 sgn_b: if b_signed generate b_ext <= b(width-1); end generate;
 adder: if add      generate c <= (a_ext & a) + (b_ext & b) + c_in; end generate;
 subtr: if not add  generate c <= (b_ext & b) - (a_ext & a) - c_in; end generate;
 reg : entity work.delay_sclr(rtl)
  generic map ( width => width+1, delay => delay)
  port map ( clk => clk, ce => ce, sclr => sclr, d => c, q => s);
end rtl;

-- This radd_sub_sclr module will be implemented in either fabric or DSP48.
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity radd_sub_sclr is
 generic (
  width   : integer :=16;
  delay   : integer := 1;   -- This parameter allows the syntheser to break the
  fabric  : integer := 1;   -- carry chain if delay is >1. In DSP48s, it allows
```

```
  add     : boolean :=true; -- registering A|B and C input ports.
  a_signed: boolean :=true; -- 1 when operand 'a' is signed
  b_signed: boolean :=true);-- 1 when operand 'b' is signed
 port (
  clk   : in std_logic;
  ce    : in std_logic;
  c_in  : in std_logic;
  a     : in std_logic_vector(width-1 downto 0);
  b     : in std_logic_vector(width-1 downto 0);
  s     : out std_logic_vector(width downto 0);
  sclr  : in std_logic);
end radd_sub_sclr;

architecture rtl of radd_sub_sclr is

begin
 use_dsp48 : if( fabric = 0 ) generate
  adder : entity work.radd_sub_sclr_yes(rtl)
  generic map ( width => width, delay => delay, add => add, a_signed => a_signed, b_signed =>
b_signed )
  port map ( clk => clk, ce => ce, c_in => c_in, a => a, b => b, s => s, sclr => sclr );
 end generate;

 use_fabric : if( fabric /= 0 ) generate
  adder : entity work.radd_sub_sclr_no(rtl)
  generic map ( width => width, delay => delay, add => add, a_signed => a_signed, b_signed =>
b_signed )
  port map ( clk => clk, ce => ce, c_in => c_in, a => a, b => b, s => s, sclr => sclr );
 end generate;
 end rtl;
```

```
-- ********************************************
-- *0003* NON-SATURATING, unbiased round macro
-- ********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

library work;
use work.genxlib_utils.max;

entity get_round_addend is
  generic (
    IWIDTH    : integer := 32;
    OWIDTH    : integer := 16;
    has_offset: integer := 0;
    mode      : integer := 0); -- 0: biased, 1: towards zero, 2: towards inf
  port (
    msb       : in std_logic; -- sign bit of input
    offset    : in  std_logic_vector(OWIDTH-1 downto 0) := (others => '0');
    r_add     : out std_logic_vector(IWIDTH-1 downto 0);
    r_cy      : out std_logic);
end get_round_addend;

architecture rtl of get_round_addend is
  constant zeros: std_logic_vector(OWIDTH downto 0) := (others => '0');
-- integer part + first fractional bit = '0'

  constant ones:  std_logic_vector(max(IWIDTH-OWIDTH-2,0) downto 0):= (others => '1');
-- rest of the fractional bits are '1'
```

```
begin
 general: if IWIDTH>OWIDTH+1 generate
  biased : if (MODE=0) generate r_cy <= '1'; end generate;
  tw_zero: if (MODE=1) generate r_cy <= msb; end generate;
  tw_inf : if (MODE=2) generate r_cy <= NOT msb; end generate;

  no_offset : if (has_offset = 0) generate r_add <= zeros & ones; end generate;
  hs_offset : if (has_offset = 1) generate r_add <= offset & '0' & ones; end generate;
 end generate;

 one_bit: if IWIDTH=OWIDTH+1 generate
  biased : if (MODE=0) generate r_cy <= '0'; end generate;
  tw_zero: if (MODE=1) generate r_cy <= msb; end generate;
  tw_inf : if (MODE=2) generate r_cy <= NOT msb; end generate;

  no_offset : if (has_offset = 0) generate r_add <= zeros; end generate;
  hs_offset : if (has_offset = 1) generate r_add <= offset & '0'; end generate;
 end generate;

 no_rnd: if IWIDTH<OWIDTH+1 generate
  biased : if (MODE=0) generate r_cy <= '0'; end generate;
  tw_zero: if (MODE=1) generate r_cy <= msb; end generate;
  tw_inf : if (MODE=2) generate r_cy <= NOT msb; end generate;

  no_offset : if (has_offset = 0) generate r_add <= zeros(IWIDTH-1 downto 0) ; end generate;
  hs_offset : if (has_offset = 1) generate r_add <= offset(IWIDTH-1 downto 0); end generate;
 end generate;
end rtl;
```

```
-- ********************************************
-- *0004* round Macro
-- ********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity round is
 generic (
  iwidth   : integer := 16;
  owidth   : integer := 15;
  has_offset: integer := 0;
  mode     : integer := 0; -- 0: biased, 1: towards zero, 2: towards inf
  delay    : integer := 1);
 port (
  a      : in std_logic_vector(iwidth-1 downto 0);
  offset : in std_logic_vector(OWIDTH-1 downto 0) := (others => '0');
  ra     : out std_logic_vector(OWIDTH-1 downto 0);
  clk    : in std_logic;
  ce     : in std_logic;
  sclr   : in std_logic);
end round;

architecture rtl of round is
 signal rounding_const : std_logic_vector(iwidth-1 downto 0);
 signal rounding_carry : std_logic;
 signal q              : std_logic_vector(iwidth downto 0);

begin
```

```
rnd_add : entity work.get_round_addend(rtl)
  generic map(
    IWIDTH     => IWIDTH,
    OWIDTH     => OWIDTH,
    has_offset => has_offset,
    mode       => mode)
  port map(
    msb        => a(IWIDTH-1),
    offset     => offset,
    r_add      => rounding_const,
    r_cy       => rounding_carry);

 adder : entity work.radd_sub_sclr(rtl)
   generic map ( width => iwidth, delay => delay)
   port map ( clk => clk, ce => ce, sclr => sclr,
          c_in => rounding_carry, a => a, b => rounding_const, s => q);

 ra <= q(iwidth-1 downto iwidth-OWIDTH);
end rtl;

-- ***************************************************
-- *0005* mult Macro
-- ***************************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity mult is
  generic (
```

```
  IWIDTHA : integer:=18;
  IWIDTHB : integer:=18;
  delay   : integer:=2);
 port (
  a       : in std_logic_vector(IWIDTHA-1 downto 0);
  b       : in std_logic_vector(IWIDTHB-1 downto 0);
  p       : out std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0);
  clk     : in std_logic;
  ce      : in std_logic;
  sclr    : in std_logic);

 attribute register_balancing: string;
 attribute register_balancing of mult: entity is "yes";
 attribute mult_style: string;
 attribute mult_style of mult: entity is "pipe_block";
end mult;

architecture rtl of mult is

 signal c : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0);

begin

 c <= a * b;

 reg : entity work.delay_sclr(rtl)
  generic map ( width => IWIDTHA+IWIDTHB, delay => delay)
  port map ( clk => clk, ce => ce, sclr => sclr, d => c, q => p);

end rtl;
```

```
-- ***********************************************
-- *0006* mac Macro
-- ***********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity mac is          -- this module calculates p = round(a*b)+c
 generic (             -- p = reg( reg(reg(a) * reg(b)) + reg(c))
  IWIDTHA    : integer:=16;
  IWIDTHB    : integer:=16;
  OWIDTH     : integer:=16;
  ROUND_MODE : integer:= 0; -- 0: biased, 1: towards zero, 2: towards inf
  HAS_C      : integer:= 0;
  CREG       : integer:= 0);
 port (
  a          : in std_logic_vector(IWIDTHA-1 downto 0);
  b          : in std_logic_vector(IWIDTHB-1 downto 0);
  c          : in std_logic_vector(OWIDTH-1 downto 0);
  p          : out std_logic_vector(OWIDTH-1 downto 0);
  clk        : in std_logic;
  ce         : in std_logic;
  sclr       : in std_logic);

 attribute register_balancing: string;
 attribute register_balancing of mac: entity is "yes";
 attribute mult_style: string;
 attribute mult_style of mac: entity is "pipe_block";
```

```
 attribute use_dsp48: string;
 attribute use_dsp48 of mac: entity is "yes";
end mac;

architecture rtl of mac is
 signal rounding_const : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0);
                        -- rounding constant (optional) combined with input c (optional)
 signal add_c_rnd_const: std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0);
                        -- optionally registered version of rounding_const
 signal rounding_carry : std_logic;
 signal r_cy_in        : std_logic := '0';
 signal ar    : std_logic_vector(IWIDTHA-1 downto 0) := (others => '0');
 signal br    : std_logic_vector(IWIDTHB-1 downto 0) := (others => '0');
 signal cr    : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');
                        -- registered version of rounding_const
 signal pr    : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');
 signal mac   : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');
 signal ones  : std_logic_vector(OWIDTH-1 downto 0) := (others => '1');

 attribute keep_hierarchy : string;
 attribute keep_hierarchy of rtl: architecture is "yes";

begin
 yes_creg: if (CREG /= 0) generate add_c_rnd_const <= cr; end generate;
 no_creg : if (CREG = 0)  generate add_c_rnd_const <= rounding_const; end generate;

 rnd_add : entity work.get_round_addend(rtl)
  generic map(
   IWIDTH     => IWIDTHA+IWIDTHB,
   OWIDTH     => OWIDTH,
```

```
  has_offset  => HAS_C,
  mode        => ROUND_MODE)
 port map(
  msb         => r_cy_in,
  offset      => c,
  r_add       => rounding_const,
  r_cy        => rounding_carry);

clk_process: process(clk)
begin
 if (clk'event and clk = '1') then
  if (sclr = '1') then
   ar     <= (others => '0');
   br     <= (others => '0');
   cr     <= (others => '0');
   pr     <= (others => '0');
   r_cy_in <= '0';
   mac    <= (others => '0');
  elsif (cc = '1') then
   r_cy_in <= ar(IWIDTHA-1) xor br(IWIDTHB-1);
   mac <= pr +  add_c_rnd_const + rounding_carry;
   pr <= ar*br;
   ar <= a;
   br <= b;
   cr <= rounding_const;
  end if;
 end if;
end process;
p <= mac(IWIDTHA+IWIDTHB-2 downto IWIDTHA+IWIDTHB-OWIDTH);
end;
```

```
-- ********************************************
-- simple mac Macro with C input
-- ********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity mac_c is            -- this module calculates mac = a*b+c
  generic (                -- p = reg( reg(reg(a) * reg(b)) + reg(c))
    IWIDTHA     : integer:=16;
    IWIDTHB     : integer:=16;
    OWIDTH      : integer:=16);
  port (
    a         : in std_logic_vector(IWIDTHA-1 downto 0);
    b         : in std_logic_vector(IWIDTHB-1 downto 0);
    c         : in std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0);
    p         : out std_logic_vector(OWIDTH-1 downto 0);
    clk       : in std_logic;
    ce        : in std_logic;
    sclr      : in std_logic);

  attribute register_balancing: string;
  attribute register_balancing of mac_c: entity is "yes";
  attribute mult_style: string;
  attribute mult_style of mac_c: entity is "pipe_block";
  attribute use_dsp48: string;
  attribute use_dsp48 of mac_c: entity is "yes";

end mac_c;
```

```
architecture rtl of mac_c is
 signal ar    : std_logic_vector(IWIDTHA-1 downto 0) := (others => '0');
 signal br    : std_logic_vector(IWIDTHB-1 downto 0) := (others => '0');
 signal cr    : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');
                        -- registered version of rounding_const
 signal pr    : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');
 signal mac   : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');

begin
 clk_process: process(clk)
 begin
  if (clk'event and clk = '1') then
   if (sclr = '1') then
    ar     <= (others => '0');
    br     <= (others => '0');
    cr     <= (others => '0');
    pr     <= (others => '0');
    mac    <= (others => '0');
   elsif (cc = '1') then
    mac <= pr +  cr;
    pr <= ar*br;
    ar <= a;
    br <= b;
    cr <= c;
   end if;
  end if;
 end process;

 p <= mac(IWIDTHA+IWIDTHB-1 downto IWIDTHA+IWIDTHB-OWIDTH);
end;
```

```
-- ***********************************************
-- simple mac Macro with P cascade input
-- ***********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity mac_pc is                          -- this module calculates mac = a*b+c
 generic (                                -- p = reg( reg(reg(a) * reg(b)) + reg(c))
  IWIDTHA    : integer:=16;
  IWIDTHB    : integer:=16;
  OWIDTH     : integer:=16);
 port (
  a         : in std_logic_vector(IWIDTHA-1 downto 0);
  b         : in std_logic_vector(IWIDTHB-1 downto 0);
  pcin      : in std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0);
  p         : out std_logic_vector(OWIDTH-1 downto 0);
  clk       : in std_logic;
  ce        : in std_logic;
  sclr      : in std_logic);

 attribute register_balancing: string;
 attribute register_balancing of mac_pc : entity is "yes";
 attribute mult_style: string;
 attribute mult_style of mac_pc : entity is "pipe_block";
 attribute use_dsp48: string;
 attribute use_dsp48 of mac_pc : entity is "yes";

end mac_pc;
```

```
architecture rtl of mac_pc is

 signal ar   : std_logic_vector(IWIDTHA-1 downto 0) := (others => '0');
 signal br   : std_logic_vector(IWIDTHB-1 downto 0) := (others => '0');
 signal pr   : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');
 signal mac  : std_logic_vector(IWIDTHA+IWIDTHB-1 downto 0) := (others => '0');

begin

 clk_process: process(clk)
 begin
  if (clk'event and clk = '1') then
   if (sclr = '1') then
    ar    <= (others => '0');
    br    <= (others => '0');
    pr    <= (others => '0');
    mac   <= (others => '0');
   elsif (ce = '1') then
    mac <= pr + pcin;
    pr <= ar*br;
    ar <= a;
    br <= b;
   end if;
  end if;
 end process;

 p <= mac(IWIDTHA+IWIDTHB-1 downto IWIDTHA+IWIDTHB-OWIDTH);

end;
```

```
-- ********************************************
-- *0007* max_sat Macro
-- ********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity max_sat is
 generic (
  width  : integer:= 16;
  delay  : integer:= 1 );
 port (
  a      : in std_logic_vector(width-1 downto 0);
  max    : in std_logic_vector(width-1 downto 0);
  ma     : out std_logic_vector(width-1 downto 0);
  clk    : in std_logic;
  ce     : in std_logic;
  sclr   : in std_logic);
 attribute register_balancing: string;
 attribute register_balancing of max_sat: entity is "yes";
end max_sat;

architecture rtl of max_sat is
 signal c : std_logic_vector(width-1 downto 0);
begin
 c <= max when (a > max) else a;
 reg : entity work.delay_sclr(rtl)
  generic map ( width => width, delay => delay)
  port map ( clk => clk, ce => ce, sclr => sclr, d => c, q => ma);
end rtl;
```

```
-- ********************************************
--  *0008*  min_sat Macro
-- ********************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity min_sat is
 generic (
   width   : integer:=16;
   delay   : integer:=1);
 port (
   a       : in std_logic_vector(width-1 downto 0);
   min     : in std_logic_vector(width-1 downto 0);
   ma      : out std_logic_vector(width-1 downto 0);
   clk     : in std_logic;
   ce      : in std_logic;
   sclr    : in std_logic );
 attribute register_balancing: string;
 attribute register_balancing of min_sat: entity is "yes";
end min_sat;

architecture rtl of min_sat is
 signal c : std_logic_vector(width-1 downto 0);
begin
 c <=  min when (a < min) else a;
 reg : entity work.delay_sclr(rtl)
   generic map ( width => width, delay => delay)
   port map ( clk => clk, ce => ce, sclr => sclr, d => c, q => ma);
end rtl;
```

ภาคผนวก F ===: Source code ของ top.vhd

(การแสดงสัญญาณภาพ ด้วยโปรแกรม Xilinx System Generator ควบคู่กับโปรแกรม Matlab)

```
----------------------------------------------------------------------------------
-- Module Name:    top - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
        port(
                data :in std_logic_vector ( 7 downto 0);
                clk_in : in std_logic; -- 100 Mhz
                rst : in std_logic;
                hs : in std_logic;
                vs : in std_logic;
                R : out std_logic_vector (7 downto 0);
                G : out std_logic_vector (7 downto 0);
                B : out std_logic_vector (7 downto 0);
                vsyn_out : out std_logic;
                hsyn_out : out std_logic;
                clk_out : out std_logic;
                synch : out std_logic;
```

```
                blank_z : out std_logic
                );
end top;

architecture Behavioral of top is

 signal data_ip : std_logic_vector(7 downto 0);
 signal data_iy : std_logic_vector(7 downto 0);
 signal data_icr : std_logic_vector(7 downto 0);
 signal data_icb : std_logic_vector(7 downto 0);
 signal hs_i : std_logic;
 signal ce : std_logic:='1';
 signal PIX_EN_in1 :  std_logic := '1';
 signal sclr  :  std_logic := '0';
 signal clk_signal_1 : std_logic;
 signal clk_signal_2 : std_logic;
 signal clk_dcm : std_logic;
 signal clk : std_logic;

 component dcm_clock
  port ( CLKIN_IN       : in   std_logic;
         RST_IN         : in   std_logic;
         CLKDV_OUT      : out  std_logic;
         CLKFX_OUT      : out  std_logic;
         CLKIN_IBUFG_OUT : out  std_logic;
         CLK0_OUT       : out  std_logic;
         LOCKED_OUT     : out  std_logic);
  end component;
```

```
component counter_hs
Port ( hs : in  STD_LOGIC;
       clk : in  STD_LOGIC;
            data_inp :in std_logic_vector(7 downto 0);
                data_oy : out std_logic_vector(7 downto 0);
                data_ecr : out std_logic_vector(7 downto 0);
                data_ocb : out std_logic_vector(7 downto 0)
                );
end component;

component  Xil_YCrCb2RGB
 generic (
  FAMILY_HAS_MAC: integer:= 1;
  FABRIC_ADDS  : integer:= 1;    -- Adders are implemented using logic fabric based adders
  IWIDTH       : integer:= 8;
  CWIDTH       : integer:= 13;      -- Coefficients are signed, CWIDTH.CWIDTH-2 format
  MWIDTH       : integer:= 23;      -- ONLY FOR NON-V4: Controls bits witheld after mults.
  OWIDTH       : integer:= 8;       -- OTHERWISE (default) IWIDTH+CWIDTH+1;
  RGBMAX       : integer:= 255;
  RGBMIN       : integer:= 0;
  ACOEF        : integer:= 2872;                -- 1.4023 *pow2(CWIDTH-2)
  BCOEF        : integer:= -1461;               -- -0.7133 *pow2(CWIDTH-2)
  CCOEF        : integer:= -703;                -- -0.3434 *pow2(CWIDTH-2)
  DCOEF        : integer:= 3630;                -- 1.7724 *pow2(CWIDTH-2)
  ROFFSET      : integer:= -366592;             -- Should be MWIDTH bits wide
  GOFFSET      : integer:= 278016;
  BOFFSET      : integer:= -463616;
  HAS_CLIP     : integer:= 1;
  HAS_CLAMP    : integer:= 1);
 port (
```

```
  Y          : in std_logic_vector(IWIDTH-1 downto 0);          -- Y = a(R-G) + G + b(B-G)
  Cr         : in std_logic_vector(IWIDTH-1 downto 0);          -- Cr = d(R-Y)
  Cb         : in std_logic_vector(IWIDTH-1 downto 0);          -- Cb = c(B-Y)
  R          : out std_logic_vector(OWIDTH-1 downto 0);
  G          : out std_logic_vector(OWIDTH-1 downto 0);
  B          : out std_logic_vector(OWIDTH-1 downto 0);
  V_SYNC_in    : in std_logic := '0';
  H_SYNC_in    : in std_logic := '0';
  PIX_EN_in    : in std_logic := '1';
  V_SYNC_out   : out std_logic;
  H_SYNC_out   : out std_logic;
  PIX_EN_out   : out std_logic;
  clk          : in std_logic;
  ce           : in std_logic := '1';
  sclr         : in std_logic := '0');
end component;

begin

u2: dcm_clock
  port map ( CLKIN_IN      => clk_in,
             RST_IN        => rst,
             CLKDV_OUT     => open,
             CLKFX_OUT     => clk,-- 108 Mhz
             CLKIN_IBUFG_OUT => open,
             CLK0_OUT      => open,
             LOCKED_OUT    => open  );

 clk_div1 : process(clk,rst)-- 54 Mhz
begin
```

```
        if(rst='1')then
            clk_signal_1 <= '0';
        elsif(rising_edge(clk))then
            clk_signal_1 <= not clk_signal_1;
        end if;
end process;

clk_div2 : process(clk_signal_1,rst)-- 27 Mhz
begin
        if(rst='1')then
            clk_signal_2 <= '0';
        elsif(rising_edge(clk_signal_1))then
            clk_signal_2 <= not clk_signal_2;
        end if;
end process;

clk_dcm <= clk_signal_2;-- 27 clk_dcm = 27 Mhz

 data_ip <= data;
 u0 : counter_hs
        port map(       clk => clk_dcm,
                        data_inp => data_ip,
                        data_oy => data_iy,
                        data_ocr => data_icr,
                        data_ocb => data_icb,
                        hs => hs
                        );
 u1:Xil_YCrCb2RGB
 port map(
   Y    => data_iy,
```

```
    Cr   => data_icr,
    Cb   => data_icb,
    R    => R,
    G    => G,
    B    => B,
    V_SYNC_in => vs,
    H_SYNC_in  => hs,
    PIX_EN_in    => PIX_EN_in1,
    V_SYNC_out   => open,
    H_SYNC_out   => open,
    PIX_EN_out   => open,
    clk          => clk_dcm,
    ce           => ce,
    sclr         => sclr );
        clk_out <= clk_dcm;

process (clk_in)
        begin
            if (rst='1')then
                synch <= '0';
                blank_z <= '0';
            else
                synch <= '1';
                blank_z <='1';
            end if;
end process;

  vsyn_out <= vs;
  hsyn_out <= hs;
end Behavioral;
```

ภาคผนวก G ===: รูปแบบสัญญาณในการสร้าง จำนวน Clock ที่ใช้ และขนาดของข้อมูล สำหรับการเขียนโปรแกรม

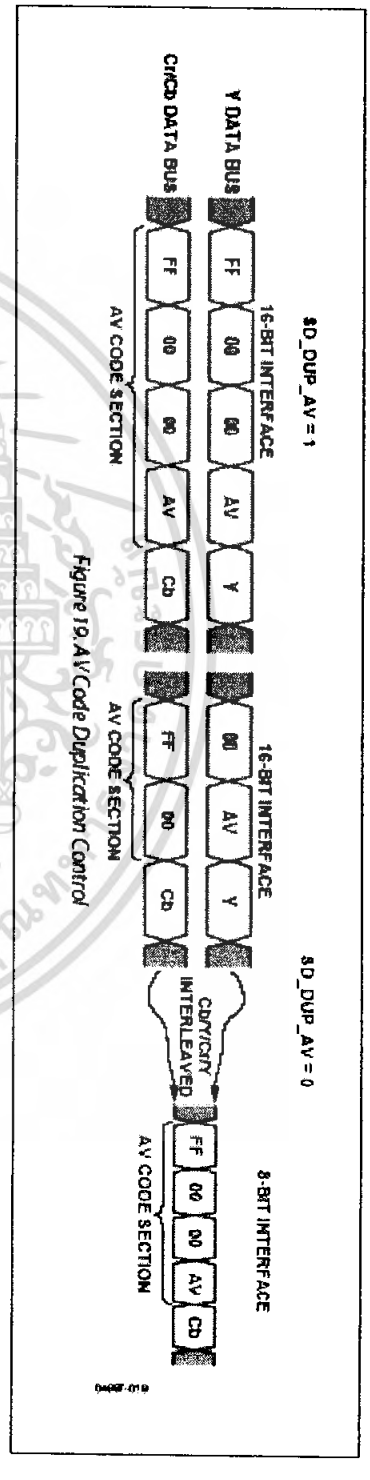


Figure 19. AV Code Duplication Control

Table 55. HS Timing Parameters (see Figure 20)

Standard	HS Begin Adjust (HSB[10:0]) (Default)	HS End Adjust (HSE[10:0]) (Default)	Characteristic: HS to Active Video (LLC1 Clock Cycles) (C in Figure 20) (Default)	Active Video Samples/Line (D in Figure 20)	Total LLC1 Clock Cycles (E in Figure 20)
NTSC	00000000010b	00000000000b	272	720Y + 720C = 1440	1716
NTSC Square Pixel	00000000010b	00000000000b	276	640Y + 640C = 1280	1560
PAL	00000000010b	00000000000b	284	720Y + 720C = 1440	1728

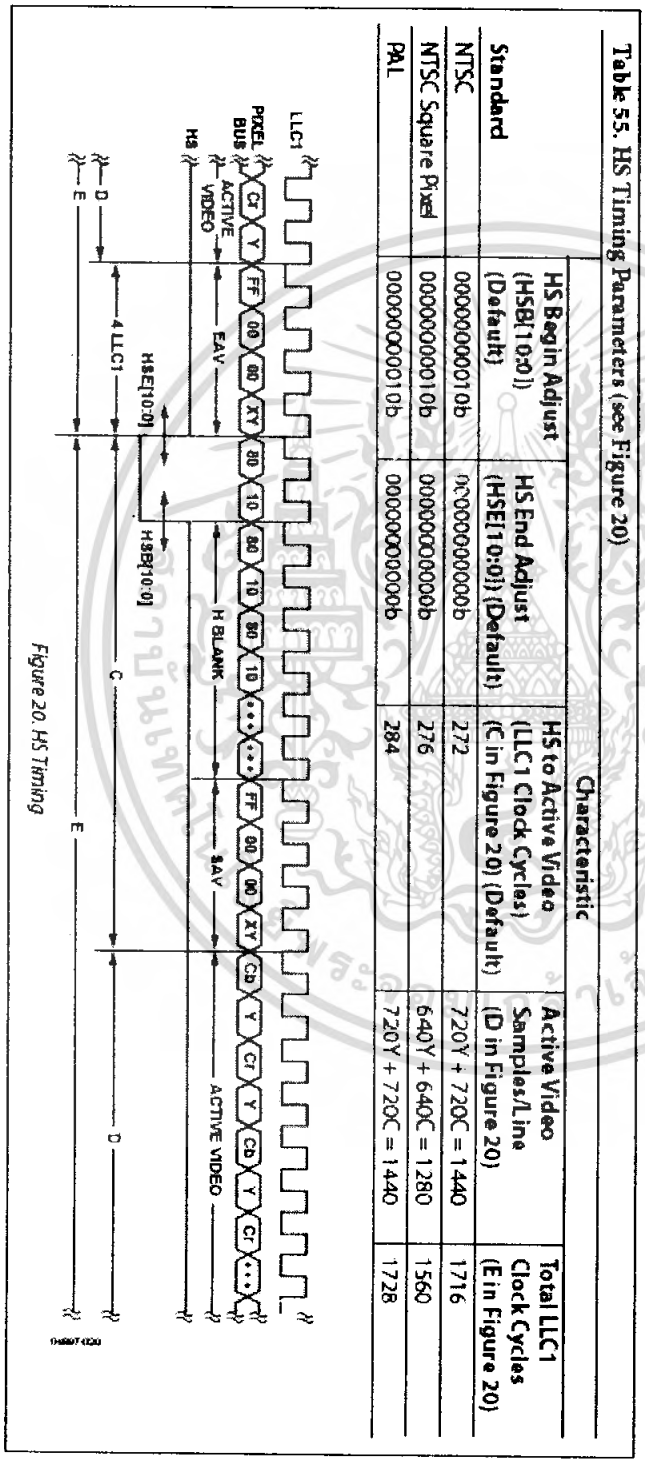


Figure 20. HS Timing

(อ้างอิง จากอุปกรณ์ Analog Device เบอร์ ADV 7183B)

ภาคผนวก H ===: Data sheet ของ Video Decoder 1 board (VDEC1)...

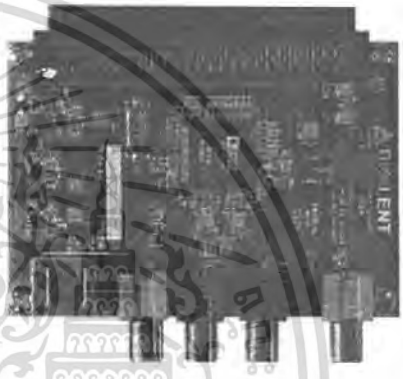
Digilent Video Decoder Board (VDEC1) Reference Manual

Revision: 4/12/05

DIGILENT™
www.digilentinc.com
246 East Main | Pullman, WA 99163
(509) 334 6306 Voice and Fax

Overview

The Video Decoder 1 board (VDEC1), centered on the ADV7183B Video Decoder chip from Analog Devices, can digitize NTSC, PAL, and SECAM video signals. The ADV7183B automatically detects standard analog baseband television signals, and digitizes them with three 54MHz 10-bit ADCs. Output data can be sent to an attached system board in 8-bit or 16-bit YCrCb 4:2:2 format.



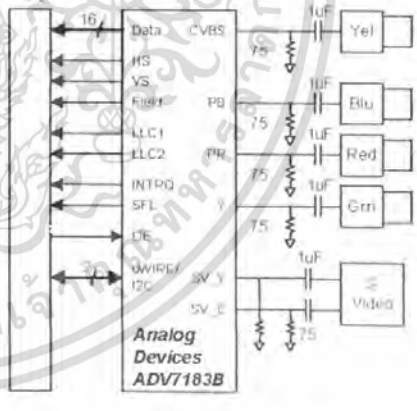
Features include:

- component, composite, and S-video inputs;
- I²C® compatible control bus;
- high-speed Hirose FX2 data connector
- supports NTSC, PAL, and SECAM inputs
- 8-bit or 16-bit YCrCb 4:2:2 outputs plus HS, VS and Field signals
- programmable controls include peak white, hue, brightness, saturation and contrast

Functional Description

The VDEC1 board is essentially a "carrier" for Analog Devices' ADV7183B Video Decoder chip. It contains all required support circuitry, including well-filtered and stable power supplies, properly terminated75-ohm inputs, a stable 27-MHz clock source, various video input connectors, and input protection networks. Refer to the Analog Devices data sheet for more information about the ADV7138B.

The VDEC1 can be used with any Digilent board that uses the Hirose FX2 connector. Catalog entries in the Digilent products webpage contain a "connector" field that



VEDC1 Circuit Diagram

clearly shows which boards have the Hirose connector.

Operation

In operation, the VDEC1 should not be attached to a system board until the signals driving the Hirose connector from the system board have been properly defined. If the

VDEC1 board is attached to a system board, and the system board is driving as outputs the same signals the VDEC1 is driving, damage to the VDEC1 and system board will result.

Before attaching the VDEC1 to a system board, ensure that any power-on auto-loaded configuration drives the Hirose pins correctly. Otherwise, ensure the system board powers on in a reset mode, not driving the Hirose pins as outputs.

Once the VDEC1 board is attached to a system board, the ADV7183B chip must be programmed (via its $I^2C^®$ compatible port) for a specific operating mode before output video data is available. Please refer to the ADV7138B data sheet for information on programming various operation modes.

After an operating mode has been selected, a video source can be attached to the appropriate video input connector, and output digital video data will be available.

Hirose Connector Pinout

The VDEC1 contains a 100-pin Hirose FX2 socket connector that mates with a corresponding Hirose plug connector on a system board. Pin1 of the socket connector attaches to pin1 of the plug connector. Thus, to generate a pin connection list for a given system board, the signal definitions in the following table can be directly mapped to the signal definitions on the system board (e.g., the signal name on VDEC1 pin5 maps directly to the signal on the system board pin5).

A Pin #	Signal	B Pin #	Signal
1	VCC33	1	Shield
2	VCC33	2	GND
3	NC	3	NC
4	NC	4	NC
5	NC	5	GND
6	RESET	6	GND
7	SDA	7	GND
8	SCLK	8	GND
9	P15	9	GND
10	P14	10	GND
11	P13	11	GND
12	P12	12	GND
13	OE	13	GND
14	FIELD	14	GND
15	VS	15	GND
16	HS	16	GND
17	P11	17	GND
18	P10	18	GND
19	P9	19	GND
20	P8	20	GND
21	INTRQ	21	GND
22	SFL	22	GND
23	P7	23	GND
24	P6	24	GND
25	P5	25	GND
26	P4	26	GND
27	P3	27	GND
28	P2	28	GND
29	LLC2	29	GND
30	P1	30	GND
31	P0	31	GND
32	PWRDN	32	GND
33	NC	33	GND
34	NC	34	GND
35	NC	35	GND
36	NC	36	GND
37	NC	37	GND
38	NC	38	GND
39	NC	39	GND
40	NC	40	GND
41	NC	41	GND
42	NC	42	GND
43	NC	43	GND
44	NC	44	GND
45	NC	45	GND
46	GND	46	GND
47	NC	47	GND
48	GND	48	NC
49	VCC5	49	VCC5
50	VCC5	50	Shield

ภาคผนวก I ===: Data sheet ของ XUP Virtex-II Pro Development System Board...

(Data sheet ที่แสดงเป็นบางส่วนเท่านั้น สามารถดูพิมพ์ได้ที่ www.xilink.com)

Xilinx University Program Virtex-II Pro Development System

Hardware Reference Manual

UG069 (v1.0) March 8, 2005

XILINX®

XUP Virtex-II Pro Development System
UG069 (v1.0) March 8, 2005

The following table shows the revision history for this document.

	Version	Revision
03/08/05	1.0	Initial Xilinx release. (DRAFT)

t

XILINX®

Chapter 1

XUP Virtex-II Pro Development System

Features

- Virtex™-II Pro FPGA with PowerPC™ 405 cores
- Up to 2 GB of Double Data Rate (DDR) SDRAM
- System ACE™ controller and Type II CompactFlash™ connector for FPGA configuration and data storage
- Embedded Platform Cable USB configuration port
- High speed SelectMAP FPGA configuration from Platform Flash In-System Programmable Configuration PROM
- Support for "Golden" and "User" FPGA configuration bitstreams
- On-board 10/100 Ethernet PHY device
- Silicon Serial Number for unique board identification
- RS-232 DB9 serial port
- Two PS-2 serial ports
- Four LEDs connected to Virtex-II Pro I/O pins
- Four switches connected to Virtex-II Pro I/O pins
- Five push buttons connected to Virtex-II Pro I/O pins
- Six expansion connectors joined to 80 Virtex-II Pro I/O pins with over-voltage protection
- High-speed expansion connector joined to 40 Virtex-II Pro I/O pins that can be used differentially or single ended
- AC-97 audio CODEC with audio amplifier and speaker/headphone output and line level output
- Microphone and line level audio input
- On-board XSGA output, up to 1200 x 1600 at 70 Hz refresh
- Three Serial ATA ports, two Host ports and one Target port
- Off board expansion MGT link, with user supplied clock
- 100 MHz system clock, 75 MHz SATA clock
- Provision for user-supplied clock
- On-board power supplies
- Power-on reset circuitry
- PowerPC 405 reset circuitry

General Description

The XUP Virtex-II Pro Development System provides an advanced hardware platform th consists of a high performance Virtex-II Pro Platform FPGA surrounded by a comprehensive collection of peripheral components that can be used to create a comple system and to demonstrate the capability of the Virtex-II Pro Platform FPGA.

Block Diagram

Figure 1-1 shows a block diagram of the XUP Virtex-II Pro Development System.

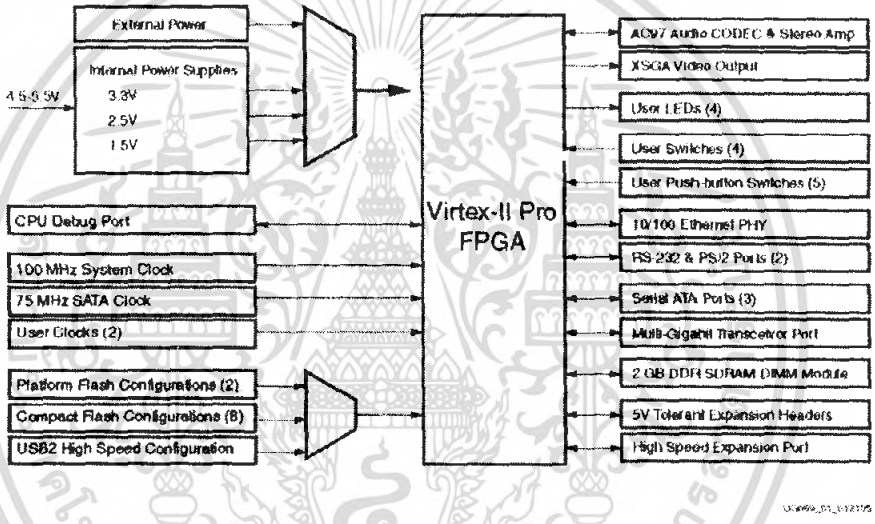


Figure 1-1: **XUP Virtex-II Pro Development System Block Diagram**

Board Components

This section contains a concise overview of several important components on the XUP Virtex-II Pro Develop ment System (see Figure 1-2). The most recent documentation for system can be obtained from the XUP Virtex-II Pro Development System support webs at: http://www.xilinx.com/univ/xup2vp.html

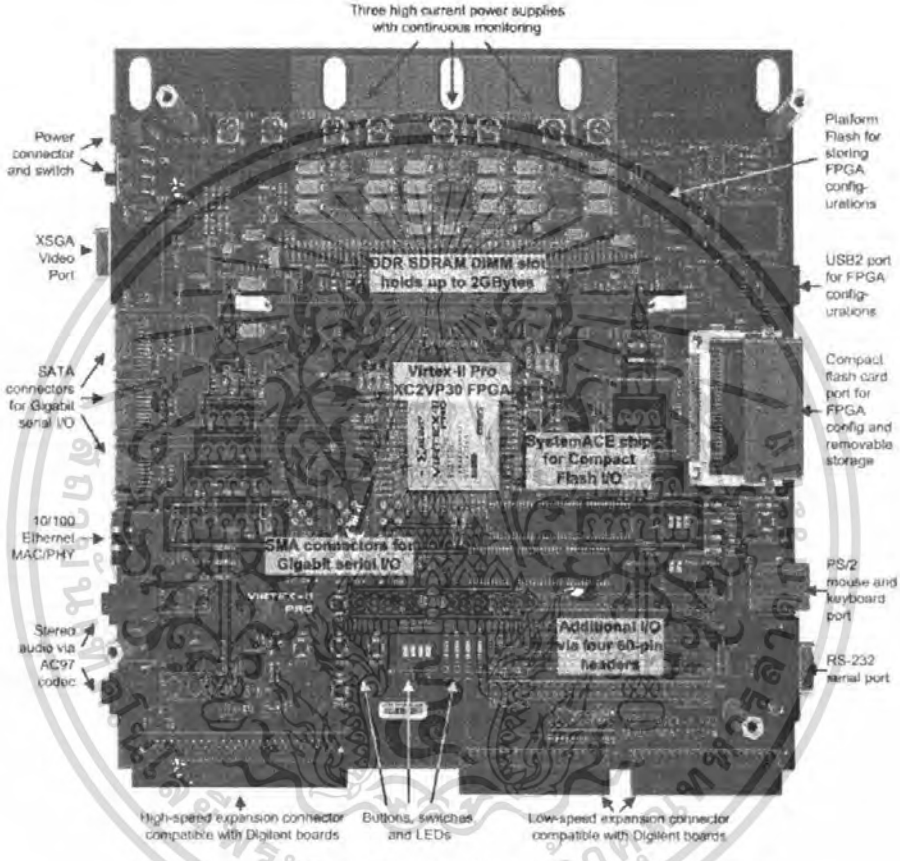


Figure 1-2: **XUP Virtex-II Pro Development System Board Photo**

Virtex-II Pro FPGA

U1 is a Virtex-II Pro FPGA device packaged in a flip-chip-fine-pitch FF896 BGA package. Two different capacity FPGAs can be used on the XUP Virtex-II Pro Development System with no change in functionality. Table 1-1 lists the Virtex-II Pro device features.

Table 1-1: **XC2VP20 and XC2VP30 Device Features**

Features	XC2VP20	XC2VP30
Slices	9280	13969
Array Size	56 x 46	80 x 46
Distributed RAM	290 Kb	428 Kb
Multiplier Blocks	88	136

Table 1-1: **XC2VP20 and XC2VP30 Device Features *(Continued)***

Features	XC2VP20	XC2VP30
Block RAMs	1584 Kb	2448 Kb
DCMs	8	8
PowerPC RISC Cores	2	2
Multi-Gigabit Transceivers	8	8

Figure 1-3 identifies the I/O banks that are used to connect the various peripheral devices to the FPGA.

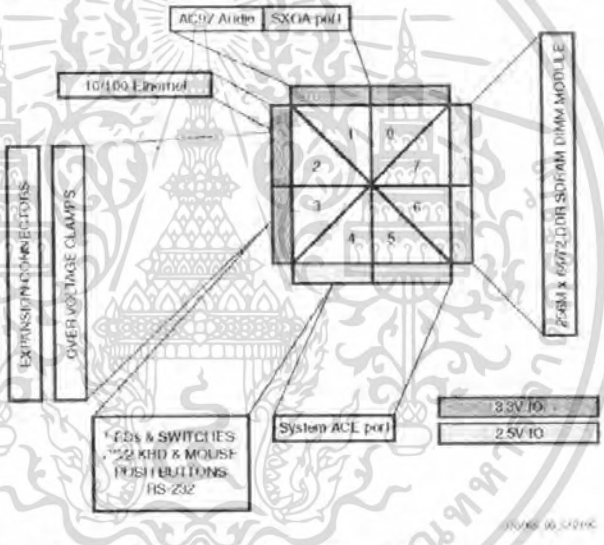


Figure 1-3: **I/O Bank Connections to Peripheral Devices**

Power Supplies and FPGA Configuration

The XUP Virtex-II Pro Development System is powered from a 5V regulated power supply. On-board switching power supplies generate 3.3V, 2.5V, and 1.5V for the FPGA, and peripheral components and linear regulators power the MGTs.

The board has provisioning for current measurement for all of the FPGA digital power supplies, as well as application of external power if the capacity of the on-board switching power supplies is exceeded.

The XUP Virtex-II Pro Development System provides several methods for the configuration of the Virtex-II Pro FPGA. The configuration data can originate from the internal Platform Flash PROM (two potential configurations), the internal CompactFlash storage media (eight potential configurations), and external configurations delivered from the embedded Platform Cable USB or parallel port interface.

ceivers

ght Multi-Gigabit Transceivers (MGTs) that are present in the Virtex-II Pro
ught out to connectors and can be utilized by the user. Three of the
MGT channels are terminated at Serial Advanced Technology Attachment
ctors and the fourth channel terminates at user-supplied Sub-Miniature A
tors. The MGT transceivers are equipped with a 75 MHz clock source that is
or the system clock to support standard SATA communication. An
GT clock source is available through a differential user-supplied (SMA)
r. Two of the ports with SATA connectors are configured as Host ports and
A port is configured as a Target port to allow for simple board-to-board

ex-II Pro Development System has provision for the installation of user-
EC-standard 184-pin dual in-line Double Data Rate Synchronous Dynamic
y module. The board supports buffered and unbuffered memory modules
ty of 2 GB or less in either 64-bit or 72-bit organizations. The 72-bit
should be used if ECC error detection and correction is required.

act Flash Controller

dvanced Configuration Environment (System ACE™) Controller manages
ration data. The controller provides an intelligent interface between an
hain and various supported configuration sources. The controller has several
mpact Flash port, the Configuration JTAG port, the Microprocessor (MPU)
Test JTAG port. The XUP Virtex-II Pro Development System supports a single
Controller. The Configuration JTAG ports connect to the FPGA and front
nnectors. The Test JTAG port connects to the JTAG port header and USB2
D, and the MPU ports connect directly to the FPGA.

ace

ex-II Pro Development System provides an IEEE-compliant Fast Ethernet
at supports both 100BASE-TX and 10BASE-T applications. It supports full
tion at 10 Mb/s and 100 Mb/s, with auto negotiation and parallel detection.
vides a Media Independent Interface (MII) for attachment to the 10/100
Controller (MAC) implemented in the FPGA. Each board is equipped with a
Number that uniquely identifies each board with a 48-bit serial number. This
r is retrieved using "1-Wire" protocol. This serial number can be used as the
address.

ex-II Pro Development System provides three serial ports: a single RS-232
PS/2 ports. The RS-232 port is configured as a DCE with hardware
sing a standard DB-9 serial connector. This connector is typically used for
ons with a host computer using a standard 9-pin serial cable connected to a
he two PS/2 ports could be used to attach a keyboard and mouse to the XUP
Development System. All of the serial ports are equipped with level-shifting
use the Virtex-II Pro FPGAs cannot interface directly to the voltage levels
RS-232 or PS/2.

User LEDs, Switches, and Push Buttons

A total of four LEDs are provided for user-defined purposes. When the FPGA drives a logic 0, the corresponding LED turns on. A single four-position DIP switch and five push buttons are provided for user input. If the DIP switch is *up*, *closed*, or *on*, or the push button is pressed, a logic 0 is seen by the FPGA, otherwise a logic 1 is indicated.

Expansion Connectors

A total of 80 Virtex-II Pro I/O pins are brought out to four user-supplied 60-pin headers and two 40-pin right angle connectors for user-defined use. The 60-pin headers are designed to accept ribbon-cable connectors, with every second signal a ground for signal integrity. Some of these signals are shared with the front-mounted right-angle connectors. The front-mounted connectors support Digilent expansion modules. In addition, a high-speed connector is provided to support Digilent high-speed expansion modules. This connector provides 40 single-ended or differential I/O signals in addition to three clocks. Consult the Digilent website at www.diglentinc.com for a list of expansion boards that are compatible with the XUP Virtex-II Pro Development System.

XSGA Output

The XUP Virtex-II Pro Development System includes a video DAC and 15-pin high-density D-sub connector to support XSGA output. The video DAC can operate with a pixel clock of up to 180 MHz. This allows for a VESA-compatible output of 1280 x 1024 at 75 Hz refresh and a maximum resolution of 1600 x 1200 at 70 Hz refresh.

AC97 Audio CODEC

An audio CODEC and stereo power amplifier are included on the XUP Virtex-II Pro Development System to provide a high-quality audio path and provide all of the analog functionality in a PC audio system. It features a full-duplex stereo ADC and DAC, with an analog mixer, combining the line-level inputs, microphone input, and PCM data.

CPU Trace and Debug Port

The FPGA is equipped with a CPU debugging interface and a 16-pin header. This connector can be used in conjunction with third party tools, the Xilinx Parallel Cable IV, or the Xilinx Platform Cable USB to debug software as it runs on either PowerPC 405 processor core.

ChipScope Pro™ can also be used to perform real-time debug and verification of the FPGA design. ChipScope Pro inserts logic analyzer, bus analyzer, and Virtual I/O low-profile software cores into the FPGA design. These cores allow the designer to view all the internal signals and nodes within the FPGA including the Processor Local Bus (PLB) or On-Chip Peripheral Bus (OPB) supporting the PowerPC 405 cores. Signals are captured and brought out through the embedded Platform Cable USB programming interface for analysis using the ChipScope Pro Logic Analyzer tool.

USB 2 Programming Interface

The XUP Virtex-II Pro Development System includes an embedded USB 2.0 microcontroller capable of communications with either high-speed (480 Mb/s) or full speed (12 Mb/s) USB hosts. This interface is used for programming or configuring the Virtex-II Pro FPGA in Boundary-Scan (IEEE 1149.1/IEEE 1532) mode. Target clock speeds are selectable from 750 kHz to 24 MHz. The USB 2.0 microcontroller attaches to a desktop or laptop PC with an off-the-shelf high-speed A-B USB cable.