การป้องกันการคัดลอกซอฟต์แวร์โดยใช้ฮาร์ดล็อค

SOFTWARE PROTECTION BY USING HARDLOCK

นายกฤษณ์ แซ่จิว
นายศรัณย์ บุญธรรมธนะรุ่ง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

การป้องกันการคัดลอกซอฟต์แวร์โดยใช้ฮาร์ดล็อค

SOFTWARE PROTECTION BY USING HARDLOCK

โดย

นายกฤษณ์ แซ่จิว

นายศรัณย์ บุญธรรมธนะรุ่ง

อาจารย์ที่ปรึกษา

รศ. สมศักดิ์ มิตะถา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การป้องกันการคัดลอกซอฟต์แวร์โดยใช้ฮาร์ดล็อค

SOFTWARE PROTECTION BY USING HARDLOCK

ผู้จัดทำ

1. นาย กฤษณ์ แซ่จิว รหัสประจำตัว 39014014

2. นาย ศรัณย์ บุญธรรมธนะรุ่ง รหัสประจำตัว 39014505

อาจารย์ที่ปรึกษา

(รศ. สมศักดิ์ มิตะถา)

การป้องกันการคัดลอกซอฟต์แวร์โดยใช้ฮาร์ดล็อค

นายกฤษณ์ แซ่จิว 39014014
นายศรัณย์ บุญธรรมธนะรุ่ง 39014505
รศ. สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการนำเสนอการป้องกันการคัดลอกซอฟต์แวร์โดยอุปกรณ์ฮาร์ดล็อค เพื่อใช้ในการป้องกันการคัดลอกซอฟต์แวร์ โดยในส่วนของวงจรที่ใช้ในการป้องกันการคัดลอกซอฟต์แวร์เราจะใช้การออกแบบโดยใช้อุปกรณ์เอฟพีจีเอ (FPGA : Field Programmable Logic Array) ซึ่งเป็นอุปกรณ์ที่สามารถโปรแกรมได้ โดยใช้สำหรับโปรแกรมวงจรที่ได้ออกแบบลงไป เพื่อให้อุปกรณ์เอฟพีจีเอมีฟังก์ชันการทำงานตามแบบที่ต้องการ สำหรับการออกแบบวงจรการป้องกันการคัดลอกซอฟต์แวร์นั้นเราใช้ภาษาวีเอชดีแอล (VHDL : Very High Speed Integrated Circuit Hardware Description Language) ในการออกแบบ ซึ่งภาษาวีเอชดีแอล เป็นภาษาที่บรรยายถึงลักษณะการทำงานของฮาร์ดแวร์ โดยในการออกแบบนั้นเราจะได้ทำการป้องกันการคัดลอกซอฟต์แวร์โดยการใช้ทฤษฎีการเข้ารหัสและถอดรหัสข้อมูลแบบดีอีเอส (DES : Data Encryption Standard) เพื่อป้องกันมิให้ผู้ที่จะทำการคัดลอกซอฟต์แวร์สามารถโจรกรรมข้อมูลสำคัญในซอฟต์แวร์ที่เราได้ป้องกันไว้ และสำหรับในส่วนของการเชื่อมต่อระหว่างวงจรฮาร์ดล็อกที่ได้ออกแบบไว้กับคอมพิวเตอร์นั้น เราได้ใช้การเชื่อมต่อกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม

SOFTWARE PROTECTION BY USING HARDLOCK

Krit Sae-Jew

Srun Boonthamtanarung

Assoc. Prof. Somsak Mitatha Advisor

ABSTRACT

This thesis is present the Software Protection by using Hardlock. In part of Software Protection circuit design we use to design by FPGA (Field Programmable Logic Array), which is device that can program anything that you want it to be by using VHDL (Very High Speed Integrated Circuit Hardware Description Language), that is hardware description language, for design structure in FPGA. In security of data design, we use DES (Data Encryption Standard) algorithm to encryption and decryption data for protect data from Hacker or Cracker. By the way, the method to connect hardlock and computer we use RS-232 serial communication port.

กิตติกรรมประกาศ

ตลอดระยะเวลาที่ได้ดำเนินการสร้างวิทยานิพนธ์ฉบับนี้ เราต้องผ่เนอุปสรรคต่าง ๆ มาอย่างมากมาย ซึ่งการแก้ไขปัญหาและอุปสรรคที่ผ่านมาจะไม่สามารถสำเร็จขึ้นได้ถ้าปราศจากบุคคลที่คอยช่วยเหลือและให้กำลังใจเรา ดังนั้นเราจึงอยากขอบคุณบุคคลดังต่อไปนี้

อาจารย์ สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ช่วยให้คำปรึกษาและคำแนะนำที่เป็นประโยชน์ ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ที่ช่วยสนับสนุนอุปกรณ์ พี่นเรศ พี่ต๋อง พี่หมี ที่ช่วยให้ข้อมูลและโปรแกรมที่จำเป็นต้องใช้ในการดำเนินงาน ตลอดจนผู้ที่มีส่วนให้ความช่วยเหลือในด้านอื่น ๆ ทั้งทางตรงและทางอ้อมที่ไม่ได้กล่าวถึงในที่นี้

และที่สำคัญที่สุดขอขอบคุณ บิดา มารดา ที่คอยอบรมสั่งสอน เลี้ยงดู และให้การศึกษาอย่างเต็มที่จนทำให้เราสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลงได้

กฤษณ์ แซ่จิว

ศรัณย์ บุญธรรมธนะรุ่ง

สารบัญ

หน้าที่

สารบัญ (ต่อ)

หน้าที่

สารบัญ (ต่อ)

หน้าที่

สารบัญ (ต่อ)

หน้าที่

สารบัญ (ต่อ)

หน้าที่

สารบัญตาราง

หน้าที่

สารบัญภาพ

หน้าที่

สารบัญภาพ (ต่อ)

หน้าที่

สารบัญภาพ (ต่อ)

หน้าที่

สารบัญภาพ (ต่อ)

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในงานการผลิตซอฟต์แวร์ (Software) ในปัจจุบันนี้นั้น มีความยากลำบากในการผลิตค่อนข้างมาก จึงทำให้ราคาของซอฟต์แวร์ มีราคาในการขายที่ค่อนข้างสูง ซึ่งทำให้ผู้ที่ต้องการใช้ซอฟต์แวร์ที่มีลิขสิทธิ์ถูกต้องตามกฎหมายต้องเสียค่าใช้จ่ายมาก ด้วยเหตุนี้จึงเกิดมีผู้ไม่ประสงค์ดีได้ทำการละเมิดลิขสิทธิ์ของทางผู้ผลิตซอฟต์แวร์ด้วยการทำการคัดลอกซอฟต์แวร์ต่าง ๆ ออกมาจำหน่ายในราคาถูก ซึ่งทำให้บรรดาผู้ผลิตซอฟต์แวร์เหล่านั้นได้รับความเสียหายเป็นอย่างมาก

ทางผู้ผลิตจึงมีความคิดที่จะทำการป้องกันการคัดลอกซอฟต์แวร์ให้เกิดขึ้นน้อยที่สุด โดยใช้แนวความคิดต่าง ๆ เช่น การใช้โปรแกรมควบคุมอยู่ในโปรแกรมประยุกต์ การใช้รหัสผ่าน การใช้วิธีการเข้ารหัสลับหรือการทำให้ซอฟต์แวร์ เมื่อจะใช้ได้ต้องมีผ่านการลงทะเบียนโดยใช้รหัส หรือที่เรียกว่าซีเรียลนัมเบอร์ (Serial Number) โดยวิธีการเหล่านี้อาจกล่าวได้ว่าเป็นการป้องกันโดยใช้ซอฟต์แวร์ (Software-Lock) ซึ่งก็สามารถป้องกันข้อมูลได้เพียงระดับหนึ่งเท่านั้น คือใช้ได้กับผู้ใช้ที่ไม่ค่อยมีความรู้ทางคอมพิวเตอร์เท่านั้น แต่สำหรับพวกกลุ่มคนที่มีความรู้และเชี่ยวชาญทางด้านคอมพิวเตอร์แล้วนั้น พวกเขาเหล่านั้นก็สามารถที่จะทำการโจรกรรม (Hack) ซอฟต์แวร์นั้น ๆ ได้ และที่สำคัญคือบางแห่งอาจจะมีการเปลี่ยนแปลงแก้ไขซอฟต์แวร์โดยไม่ได้รับอนุญาตด้วย ซึ่งการกระทำในลักษณะนี้สร้างความเสียหายให้กับเจ้าของซอฟต์แวร์เป็นอย่างมาก

จากที่ได้กล่าวมาทั้งหมดนั้น ทำให้เกิดมีแนวความคิดที่จะนำอุปกรณ์ทางฮาร์ดแวร์ (Hardware) มาช่วยเสริมในการป้องกันการคัดลอกซอฟต์แวร์ขึ้น เพื่อให้ความสามารถในการป้องกันการคัดลอกซอฟต์แวร์นั้นมีประสิทธิภาพสูงขึ้น โดยเราเรียกอุปกรณ์ฮาร์ดแวร์ที่นำมาใช้ในการป้องกันการคัดลอกซอฟต์แวร์นี้ว่า ฮาร์ดล็อค (Hardlock หรือ Hardware-Lock) ซึ่งในโครงงานวิจัยนี้ เราจะทำการสร้างอุปกรณ์ฮาร์ดล็อคขึ้นมา เพื่อใช้ในการป้องกันการคัดลอกซอฟต์แวร์ ซึ่งจะได้กล่าวในส่วนของรายละเอียดต่อไป

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 ศึกษาหลักการในการสร้างอุปกรณ์ฮาร์ดล็อค หลังจากนั้นจึงทำการออกแบบและสร้างฮาร์ดล็อคขึ้นมา เพื่อใช้เป็นอุปกรณ์ในการป้องกันการคัดลอกซอฟต์แวร์ ซึ่งฮาร์ดล็อคที่ได้ทำการสร้างขึ้นมานี้จะสามารถนำไปใช้งานได้จริง และบางทียังอาจที่จะสามารถนำไปผลิตออกจำหน่ายในท้องตลาด เพื่อเป็นการหารายได้ได้อีกด้วย

1.2.2 ศึกษาหลักการในการเข้ารหัสและถอดรหัสของข้อมูล เพื่อนำไปใช้ในการเข้ารหัสลับของข้อมูล โดยที่การเข้ารหัสและถอดรหัสข้อมูลนี้จะเป็นการเพิ่มความปลอดภัยในการป้องกันการลักลอบโจรกรรมหรือแอบดูข้อมูล ซึ่งจะเป็นส่วนที่สำคัญในการสร้างอุปกรณ์ฮาร์ดล็อค

1.2.3 เมื่อมีการสร้างอุปกรณ์ที่มีความสามารถในการป้องกันการคัดลอกซอฟต์แวร์ได้ดี จะส่งผลให้อุตสาหกรรมทางด้านซอฟต์แวร์ในประเทศเกิดการแข่งขันและมีการพัฒนาที่สูงขึ้น ซึ่งจะทำให้มีผู้ผลิตซอฟต์แวร์ภายในประเทศมากขึ้น

1.2.4 จากการที่มีการใช้อุปกรณ์ฮาร์ดล็อคในการป้องกันการคัดลอกซอฟต์แวร์อย่างแพร่หลายนั้น จะทำให้เกิดการพัฒนาเทคโนโลยีและวิทยาการทางการเข้ารหัสและถอดรหัสข้อมูลมีมากขึ้นตามไปด้วย

1.3 ขอบเขตของงานวิจัย

โครงงานวิจัยนี้จะเป็นการสร้างอุปกรณ์ฮาร์ดล็อคขึ้นมาเพื่อใช้ในการป้องกันการคัดลอกซอฟต์แวร์ ซึ่งในการสร้างฮาร์ดล็อคนั้นจะนำหลักการของการเข้ารหัสและถอดรหัสของข้อมูลเข้ามาใช้ โดยจะใช้อุปกรณ์ประเภทเอฟพีจีเอ (FPGA : Field Programmable Gate Array) ในการออกแบบวงจรเข้ารหัสและถอดรหัส และจะทำการเชื่อมต่ออุปกรณ์ฮาร์ดล็อคเข้ากับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม (Serial Port) สำหรับฮาร์ดล็อคที่เราจะสร้างในโครงงานนี้ แรกทีเดียวได้ออกแบบไว้ว่าจะให้ฮาร์ดล็อคที่จะสร้างขึ้นนั้นสามารถที่จะนำไปใช้ได้กับทุก ๆ โปรแกรม แต่จากการทดสอบฮาร์ดล็อคแบบนี้จะพบว่ายังมีข้อบกพร่องในเรื่องของความปลอดภัยในการใช้งานให้ปราศจากการถูกโจรกรรมข้อมูล ทางเราจึงได้ทำการออกแบบฮาร์ดล็อคอีกแบบที่มีความปลอดภัยจากการถูกโจรกรรมข้อมูลสูง แต่จะจำกัดการใช้ให้สามารถนำไปใช้ได้กับโปรแกรมที่มีซอสโค้ด (Source Code) ของโปรแกรมมาให้

นอกจากนั้นในโครงงานนี้ยังถือว่า เป็นโครงงานที่ทำการทดลองศึกษาความเป็นไปได้ในการสร้างอุปกรณ์ฮาร์ดล็อคเพื่อนำไปใช้ในการป้องกันการคัดลอกซอฟต์แวร์ ดังนั้นจึงอาจจะยังมีข้อจำกัดของฮาร์ดล็อคที่สร้างขึ้นอยู่ เช่น ขนาดของฮาร์ดล็อคจะยังมีขนาดที่ค่อนข้างใหญ่ ต้องอาศัยไฟเลี้ยงจากภายนอก และต้องทำการดาวน์โหลดโปรแกรมในการเข้ารหัสและถอดรหัสข้อมูลลงในอุปกรณ์เอฟพีจีเอทุกครั้งเมื่อจะใช้งานฮาร์ดล็อค เพราะอุปกรณ์เอฟพีจีเอนั้นต้องมีไฟเลี้ยงอยู่เสมอตลอดการใช้ หากปิดไฟแล้วข้อมูลก็จะหาย แต่ถึงอย่างไรก็ตามอุปกรณ์ฮาร์ดล็อคที่สร้างขึ้นนั้นจะมีความสามารถที่จะนำไปใช้งานได้จริงและมีประสิทธิภาพ

1.4 วิธีการดำเนินงาน

สำหรับในโครงงานวิจัยนี้จะเริ่มต้นด้วยการศึกษาถึงหลักการเบื้องต้นของอุปกรณ์ฮาร์ดล็อค ทั้งในแบบที่มีอยู่แล้วในท้องตลาดและแบบที่เราได้ออกแบบขึ้นมา ซึ่งมีรายละเอียดอยู่ในบทที่ 2 หลังจากนั้นจะทำการศึกษาถึงทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องกับงานโครงงานวิจัยนี้ ซึ่งก็มีเรื่องหลัก ๆ อยู่ 4 เรื่องด้วยกัน คือ ทฤษฎีการเข้ารหัสแบบดีอีเอส (DES : Data Encryption Standard) หลักการเขียนภาษาในการอธิบายฮาร์ดแวร์ ซึ่งในโครงงานวิจัยนี้ได้ใช้ภาษาวีเอชดีแอล (VHDL) หลักการออกแบบวงจรด้วยอุปกรณ์เอฟพีจีเอและการเชื่อมต่อแบบอนุกรมผ่านพอร์ตอนุกรม (Serial / RS-232 Port) ซึ่งจะมีรายละเอียดต่าง ๆ รวมอยู่ในบทที่ 3

จากนั้นจะเริ่มเข้าสู่ขั้นตอนของการออกแบบวงจรของอุปกรณ์ฮาร์ดล็อค โดยในบทที่ 4 จะกล่าวถึงการออกแบบวงจรที่ใช้ในการสร้างฮาร์ดล็อคทั้งหมด ทั้งวงจรฮาร์ดล็อค วงจรการเข้ารหัสแบบดีอีเอส วงจรการควบคุมการทำงาน และวงจรที่ใช้ติดต่อกับคอมพิวเตอร์ และในบทที่ 5 จะเป็นรายละเอียดของการออกแบบโปรแกรมที่ใช้ติดต่อกับผู้ใช้งาน ซึ่งประกอบไปด้วยโปรแกรมของผู้ผลิตซอฟต์แวร์ และโปรแกรมของผู้ที่ใช้ซอฟต์แวร์

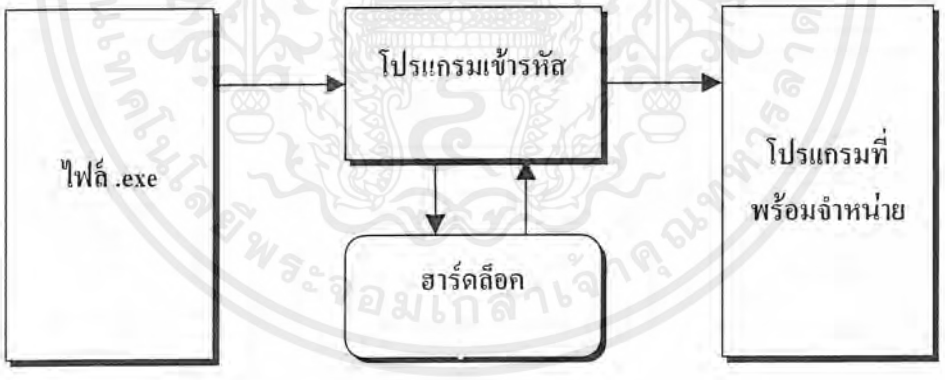
สำหรับในบทที่ 6 จะเป็นการทดสอบระบบรวมทั้งหมด ทั้งการทดสอบวงจรเข้ารหัสแบบดีอีเอส และการทดสอบโปรแกรมที่ใช้ในการติดต่อกับผู้ใช้งาน และในบทที่ 7 ซึ่งเป็นบทสุดท้าย ก็จะเป็นการสรุปการทำงาน ผลที่ได้รับจากโครงงานชิ้นนี้ แนวทางในการนำไปประยุกต์ใช้งาน และแนวทางในการพัฒนาโครงงานนี้เพิ่มเติมในอนาคตต่อไป

บทที่ 2
หลักการเบื้องต้นของฮาร์ดล็อค

การใช้ฮาร์ดล็อคนั้นมีจุดประสงค์เพื่อทำให้สามารถใช้โปรแกรมได้อย่างปกติเมื่อใดก็ตามที่ไม่ต่อฮาร์ดล็อคเข้ากับคอมพิวเตอร์ก็จะไม่สามารถใช้โปรแกรมที่ถูกป้องกันนั้นได้ มีหลายวิธีที่สามารถประยุกต์ใช้ฮาร์ดล็อคเพื่อให้ได้จุดประสงค์ดังกล่าว เช่น ผู้ผลิตซอร์ฟแวร์จะจำหน่ายโปรแกรมเป็นไฟล์ที่ถูกเข้ารหัสไว้แล้วมาโดยยังไม่สามารถที่จะปฏิบัติงาน (execute) ได้ ต้องนำไฟล์ที่ถูกเข้ารหัสนั้นส่งไปถอดรหัสที่ฮาร์ดล็อคก่อนจึงจะสามารถทำงานได้ อีกวิธีการหนึ่งคือการแทรกโค้ดส่วนหนึ่งเข้าไปในตัวโปรแกรมเพื่อให้มีการตรวจหาฮาร์ดล็อคอยู่อย่างสม่ำเสมอโดยถ้าพบว่าไม่มีฮาร์ดล็อคต่ออยู่โปรแกรมนั้นก็จะทำการปิดตัวเองไป ในแต่ละวิธีการนั้นก็จะมีข้อดีและข้อเสียแตกต่างกันไปดังรายละเอียดดังต่อไปนี้

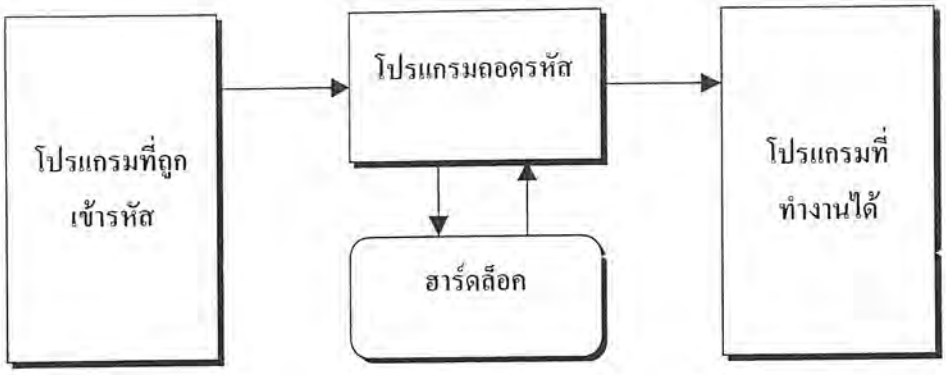
2.1 การป้องกันโดยการนำโปรแกรมไปเข้ารหัส

วิธีการนี้ทำได้โดยผู้ผลิตจะทำการคอมไพล์ (compile) โปรแกรมให้เป็นไฟล์ .exe ก่อนหลังจากนั้นก็จะทำการเรียกโปรแกรมสำหรับเข้ารหัสขึ้นมา โดยโปรแกรมนี้จะทำหน้าที่ในการนำข้อมูลจากไฟล์ต้นฉบับมาบางตำแหน่งแล้วส่งข้อมูลนี้ไปให้ฮาร์ดล็อคเพื่อเข้ารหัส หลังจากนั้นก็จะนำข้อมูลที่เข้ารหัสนี้ไปใส่ไว้ในไฟล์ที่ตำแหน่งเดิม ซึ่งก็จะได้ไฟล์ที่เข้ารหัสและพร้อมที่จะนำไปจำหน่ายดังแสดงในรูปที่ 2-1



รูปที่ 2-1 การป้องกันซอร์ฟแวร์โดยการเข้ารหัส

เมื่อผู้ใช้ต้องการที่จะใช้โปรแกรมก็จะต้องทำการเรียกโปรแกรมสำหรับที่จะถอดรหัสขึ้นมาโดยโปรแกรมนี้จะทำหน้าที่เป็นตัวนำข้อมูลจากไฟล์ ในตำแหน่งที่ถูกเข้ารหัสไว้ส่งไปให้ฮาร์ดล็อคเพื่อที่จะถอดรหัสเมื่อถอดรหัสเสร็จแล้วก็จะนำข้อมูลที่ถอดรหัสเสร็จแล้วเข้าไปใส่ไว้ในตำแหน่งเดิมและทำการปฏิบัติงาน (execute) ไฟล์นั้น ในขณะที่เปิดโปรแกรมสำหรับถอดรหัสอยู่นั้นจะมีการตรวจฮาร์ดล็อคอยู่ตลอดถ้าไม่พบก็จะทำการปิดโปรแกรมที่กำลังใช้งานอยู่ และก็จะทำลายข้อมูลที่ถูกถอดรหัสแล้วทิ้งด้วย ดังแสดงในรูปที่ 2-2

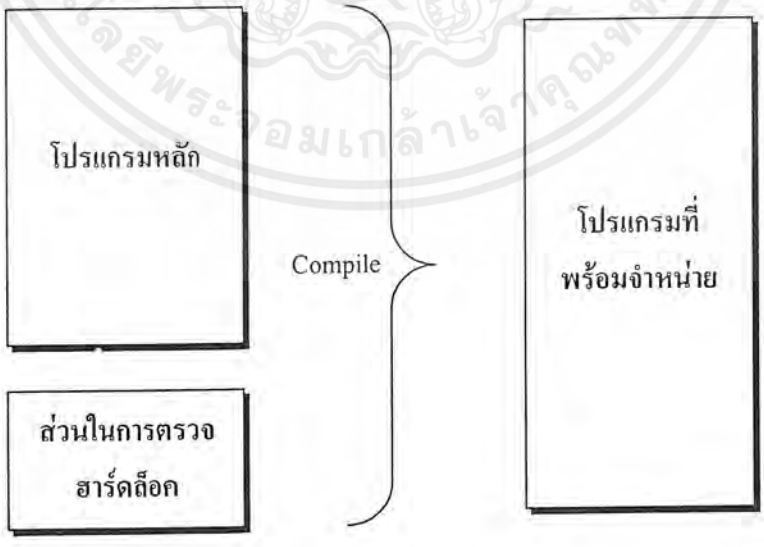


รูปที่ 2-2 วิธีการใช้งานโปรแกรมที่ถูกเข้ารหัส

วิธีการนี้มีข้อดีคือสามารถที่จะใช้ได้กับโปรแกรมบนระบบปฏิบัติการวินโดวส์ที่คอมไพล์เป็น .exe เรียบร้อยแล้วได้ทุกโปรแกรมโดยที่ไม่ต้องมีซอสโค้ด (source code) อยู่ด้วย แต่ก็จะมีข้อเสียคือจะไม่ปลอดภัยตรงที่ถ้าหากมีการดักข้อมูลที่ถูกถอดรหัสไว้แล้วและเก็บไว้ก็จะสามารถนำมาใช้งานได้โดยที่ไม่จำเป็นต้องมีฮาร์ดล็อค แต่ข้อเสียนี้จะสามารถแก้ได้โดยวิธีการแทรกโค้ดเข้าไปในโปรแกรมดังจะกล่าวในหัวข้อต่อไป

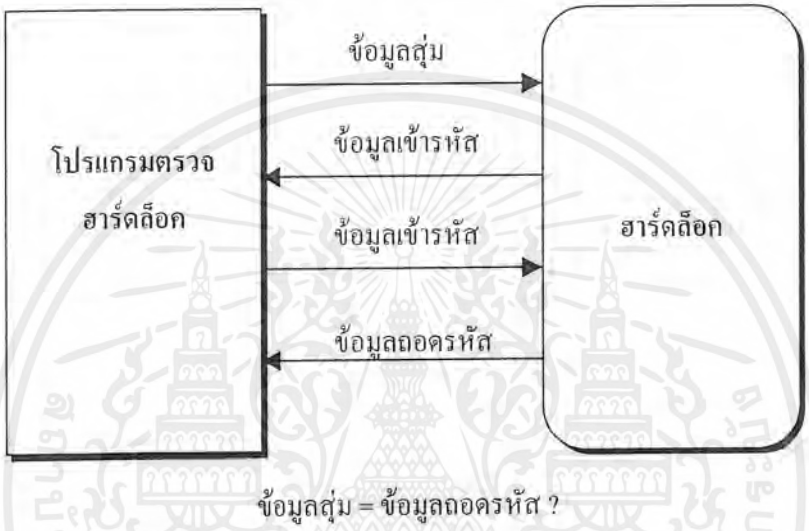
2.2 การป้องกันโดยการแทรกโค้ดเข้าไปในโปรแกรม

วิธีการนี้จะนำซอสโค้ดต้นฉบับมาทำการเพิ่มเติมส่วนที่เป็นตัวในการตรวจสอบฮาร์ดล็อคว่ายังต่ออยู่กับคอมพิวเตอร์หรือไม่ถ้าพบว่าไม่มีการตอบสนองจากฮาร์ดล็อคหรือได้รับการตอบสนองที่ไม่ถูกต้องก็จะปิดตัวเอง รูปที่ 2-3 แสดงขั้นตอนการแทรกโค้ดเข้าไปในโปรแกรมหลัก



รูปที่ 2-3 การแทรกโค้ดเข้าไปในโปรแกรมหลัก

หน้าที่ของโปรแกรมที่เราแทรกเข้าไปในโปรแกรมหลักนั้นคือในทุก ๆ ช่วงเวลาหนึ่งจะทำการสร้างข้อมูลสุ่มขึ้นมาจำนวนหนึ่งหลังจากนั้นก็จะทำการส่งข้อมูลนี้ไปยังฮาร์ดล็อคเพื่อทำการเข้ารหัส หลังจากที่ได้ข้อมูลที่เข้ารหัสมาจากฮาร์ดล็อคแล้วก็จะนำข้อมูลชุดนี้ส่งกลับไปถอดรหัสที่ฮาร์ดล็อคอีกครั้งหนึ่งและก็จะนำข้อมูลที่ฮาร์ดล็อคถอดรหัสให้มานี้มาทำการเปรียบเทียบกับข้อมูลที่สุ่มขึ้นมาในตอนแรก ถ้าข้อมูลตรงกันก็แสดงว่าฮาร์ดล็อคมีการตอบสนองที่ถูกต้อง ถ้าหากไม่ตรงกันหรือไม่มีการตอบสนองจากฮาร์ดล็อคก็แสดงว่าไม่มีฮาร์ดล็อคที่ถูกต้องต่ออยู่กับคอมพิวเตอร์ โปรแกรมก็จะทำการปิดตัวเองดังแสดงในรูปที่ 2-4



รูปที่ 2-4 การทำงานของโปรแกรมตรวจสอบฮาร์ดล็อค

วิธีการนี้จะสามารถแก้ปัญหาในเรื่องของความปลอดภัยที่เป็นจุดบอดของวิธีการแรกได้แต่ก็จะมีข้อเสียคือเราจำเป็นที่จะต้องมีซอสโค้ดโปรแกรมหลักอยู่จึงจะสามารถแทรกโค้ดเพื่อการตรวจสอบฮาร์ดล็อคเข้าไปได้

2.3 ตัวอย่างฮาร์ดล็อคที่มีจำหน่ายอยู่ในปัจจุบันนี้

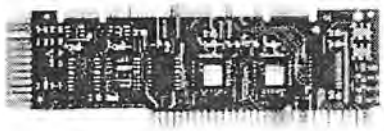
สำหรับตัวอย่างของฮาร์ดล็อคที่มีจำหน่ายอยู่ในปัจจุบันนี้นั้นจะมีอยู่ด้วยกันมากมายหลายรูปแบบ โดยเราจะยกตัวอย่างฮาร์ดล็อคที่มีจำหน่ายของบริษัท ALADDIN ซึ่งเป็นบริษัทที่ผลิตอุปกรณ์ฮาร์ดล็อคหลายรูปแบบมาก โดยเราจะยกตัวอย่างมานำเสนอบางรูปแบบ ดังนี้

2.3.1 Hardlock Internal

สำหรับฮาร์ดล็อครูปแบบนี้มีลักษณะที่สำคัญ ดังเช่น

- เป็นรูปแบบของฮาร์ดล็อคที่อยู่ในรูปแบบของการ์ด (Card) ที่เป็นส่วนเพิ่มเติมเข้าไปภายในคอมพิวเตอร์

- ในส่วนของฮาร์ดแวร์นั้นได้ออกแบบมาให้ใช้ได้กับสล็อต (Slot) แบบ PC/AT (ISA)
- การป้องกันการคัดลอกซอฟต์แวร์นั้นจะเป็นอิสระจากสถาปัตยกรรมของเครื่องคอมพิวเตอร์
- การป้องกันการคัดลอกนั้นจะถูกซ่อนอยู่ภายในคอมพิวเตอร์ เพราะฉะนั้นจึงมองไม่เห็นและไม่สามารถจะเคลื่อนย้ายได้ ดังนั้นจึงเหมาะกับสถานที่ที่เสี่ยงต่อการเกิดการสูญหาย



รูปที่ 2-5 Hardlock Internal

สำหรับข้อดีของฮาร์ดล็อครูปแบบนี้นั้นก็คือ ตัวฮาร์ดล็อคจะต่ออยู่ภายในเครื่องคอมพิวเตอร์ ทำให้สะดวก ไม่มีปัญหาในเรื่องพื้นที่การใช้งาน ปลอดภัยต่อการสูญหาย ส่วนข้อเสียนั้น จะพบว่ามีความไม่สะดวกถ้าเราต้องการนำฮาร์ดล็อคไปใช้กับเครื่องคอมพิวเตอร์เครื่องอื่น เพราะการถอดตัวฮาร์ดล็อคนั้นทำได้ค่อนข้างลำบาก

2.3.2 Hardlock USB

สำหรับฮาร์ดล็อคในรูปแบบนี้จะมีลักษณะที่สำคัญ ดังเช่น

- มีการเชื่อมต่อกับคอมพิวเตอร์ผ่านทางพอร์ต USB (Universal Serial Bus)
- สนับสนุนการทำงานทั้งหมดของมาตรฐาน USB
- มีคุณสมบัติ Plug and Play คือเมื่อต่อฮาร์ดล็อคเข้ากับคอมพิวเตอร์แล้วนั้น จะมีการติดตั้งและปรับแต่งโดยอัตโนมัติของตัวไดรฟ์เวอร์ (Driver) ของฮาร์ดล็อค
- มีความสามารถที่เรียกว่า Hot Plug คือเราสามารถที่จะต่อฮาร์ดล็อคเข้ากับคอมพิวเตอร์ในขณะที่คอมพิวเตอร์นั้นยังใช้งานหรือกำลังรันอยู่ได้เลย โดยไม่ต้องมีการรีสตาร์ท (Restart) คอมพิวเตอร์ใหม่

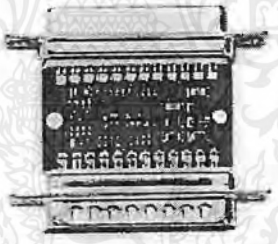
รูปที่ 2-6 Hardlock USB

ข้อดีของฮาร์ดล็อคในรูปแบบนี้นั้น ก็คือมีความสะดวกและง่ายต่อการใช้งาน ซึ่งเราจะสามารถใช้ฮาร์ดล็อคนี้ได้เลยเมื่อเราต้องการเพียงต่อเข้ากับคอมพิวเตอร์ทางพอร์ต USB โดยไม่จำเป็นต้องรีสตาร์ทเครื่องคอมพิวเตอร์ใหม่ด้วย แต่จะมีข้อเสียคือ เทคโนโลยีเกี่ยวกับเรื่องมาตรฐาน USB นั้นเป็นเรื่องที่ยังใหม่ทำให้ค่อนข้างพบได้ยากในการใช้งาน และยังมีราคาที่ค่อนข้างสูงอีกด้วย

2.3.3 ฮาร์ดล็อคที่มีหน่วยความจำภายในตัว

สำหรับฮาร์ดล็อคที่มีหน่วยความจำอยู่ภายในตัวมันนี้ จะมีลักษณะที่สำคัญ ดังเช่น

- มีหน่วยความจำขนาด 128 ไบท์ โดยแบ่งเป็นหน่วยความจำแบบ ROM ขนาด 96 ไบท์ และเป็นหน่วยความจำแบบ RAM ขนาด 32 ไบท์
- สามารถที่จะป้องกันการคัดลอกซอฟต์แวร์ได้หลายโปรแกรมโดยใช้กุญแจการเข้ารหัส 1 กุญแจ
- สำหรับการเข้าถึงหน่วยความจำนั้น จะถูกจัดการและทำให้มีความปลอดภัยโดยการใช้เทคโนโลยีของการทำวงจรรวม (ASIC : Application Specific Integrated Circuit)
- มีการป้องกันการคัดลอก โดยทำในหน่วยความจำแบบ ROM ซึ่งจะผ่านการเข้ารหัสข้อมูลของอุปกรณ์ฮาร์ดแวร์ด้วย



รูปที่ 2-7 ฮาร์ดล็อคที่มีหน่วยความจำภายในตัว

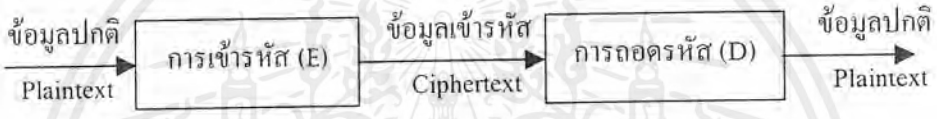
สำหรับข้อดีของฮาร์ดล็อคในรูปแบบนี้ ก็คือตัวฮาร์ดล็อคจะมีหน่วยความจำเป็นของตัวเอง ทำให้ขั้นตอนในการเข้ารหัสและถอดรหัสข้อมูลอยู่ภายในหน่วยความจำ ซึ่งจะมีความปลอดภัยในการถูกโจรกรรมข้อมูล แต่จะมีข้อเสียคือ ฮาร์ดล็อครูปแบบนี้จะใช้กุญแจการเข้ารหัสเพียงอันเดียวในการใช้ป้องกันการคัดลอกกับหลายโปรแกรม ทำให้ถ้าหากกุญแจนั้นถูกถอดรหัสหรือถูกโจรกรรมข้อมูลแล้วก็จะถูกนำไปใช้ได้เลยกับโปรแกรมต่าง ๆ

บทที่ 3
ทฤษฎีพื้นฐานในการสร้างฮาร์ดล็อค

3.1 การเข้ารหัสและถอดรหัสข้อมูล

3.1.1 หลักการถอดรหัสเบื้องต้น

การเข้ารหัส (Encryption) เป็นวิธีการป้องกันข้อมูลแบบหนึ่งที่มีการเปลี่ยนแปลงข้อมูลเดิมไปเป็นข้อมูลชุดใหม่ ซึ่งเป็นชุดข้อมูลที่ใช้ส่งออกจากพอร์ตเพื่อป้องกันผู้แอบดูข้อมูล ดังนั้นผู้เข้ารหัสจึงจำเป็นต้องทำการถอดรหัส (Decryption) เปลี่ยนข้อมูลที่เข้ารหัสลับไปเป็นข้อมูลชุดเดิม ในระบบที่ประกอบด้วยทั้งส่วนของการเข้าและถอดรหัสนั้นจะเรียกว่า ระบบสร้างรหัส (Cryptosystem) ดังรูปที่ 3-1



รูปที่ 3-1 การเข้ารหัสและถอดรหัสเบื้องต้น

ข้อมูลปกติ (Plaintext) เขียนย่อว่า P จะเป็นการเรียงต่อกันของข้อมูลเป็น Bit คือ $P = (p_1, p_2, \ldots, p_n)$ และข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) เขียนได้เป็น $C = (c_1, c_2, \ldots, c_m)$ ขั้นตอนการเข้ารหัสก็คือการแปลงระหว่างข้อมูลปกติไปเป็นข้อมูลที่เข้ารหัส จะเขียนแทนด้วย $C = E(P)$ ขณะที่การถอดรหัสจะเขียนสมการได้เป็น $P = D(C)$ และระบบการสร้างรหัสลับเขียนได้เป็น $P = D(E(P))$

สำหรับการเข้ารหัสและถอดรหัสนั้น ในบางแบบ อาจจะต้องมีการใช้กุญแจ (Key) มารวมเข้ากับข้อมูลปกติเพื่อไปเป็นข้อมูลที่ถูกเข้ารหัส ในกรณีนี้สมการการเข้ารหัส คือ $C = E(K,P)$ โดยกุญแจ K จะเป็นค่าหรือขั้นตอนเฉพาะหนึ่ง ๆ ถ้ากุญแจนี้ใช้ทั้งการเข้ารหัสและถอดรหัส สมการสร้างรหัสจะเป็น $P = D(K,E(K,P))$ แต่ในบางกรณีกุญแจที่ใช้ในการเข้ารหัสและถอดรหัสนั้นเป็นคนละกุญแจกัน จะมีลมการเป็น $P = D(K_D,E(K_E,P))$ โดยที่ K_D คือ กุญแจที่ใช้ในการถอดรหัส และ K_E คือ Key ที่ใช้ในการเข้ารหัส

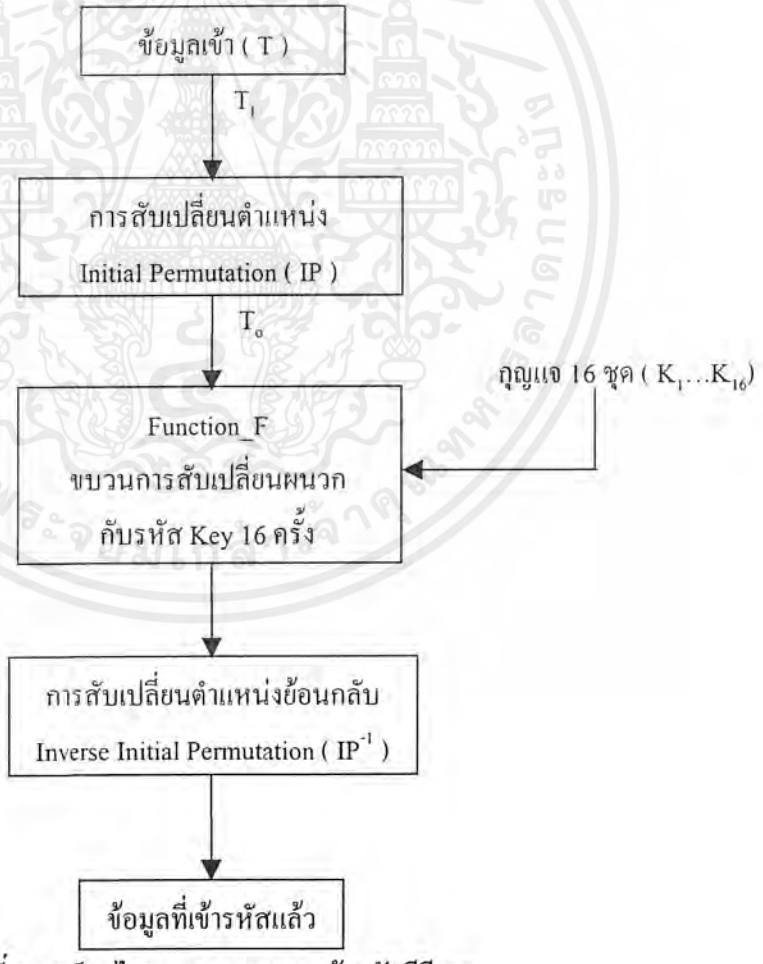
3.1.2 ทฤษฎีการเข้ารหัสและถอดรหัสแบบ DES (Data Encryption Standard)

จากวิธีการเข้ารหัสและถอดรหัสที่กล่าวมาไว้ก่อนหน้านี้เป็นวิธีการเข้ารหัสอย่างง่ายซึ่งยังไม่มีความปลอดภัยมากพอที่จะนำมาทำเป็นฮาร์ดล็อค อย่างไรก็ตามในกรณีที่ต้องการวิธีการเข้ารหัสและถอดรหัสที่ซับซ้อนมากกว่าวิธีที่กล่าวมา สามารถทำได้โดยใช้วิธีของการเข้ารหัสและถอดรหัสแบบดีอีเอส (DES : Data Encryption Standard) ซึ่งเป็นวิธีการที่ใช้วิธีการทางคณิตศาสตร์เข้ามาทำการสับเปลี่ยนข้อมูล โดยข้อมูลที่เข้ามาจะมีการผ่านขบวนการการเข้ารหัส ทำให้ข้อมูลถูกสับเปลี่ยนไปเป็นข้อมูลที่ไม่เป็นรูป

แบบเรียกว่า ไซเฟอร์ (Cipher) ส่วนการถอดรหัสนั้นจะเป็นการนำข้อมูลที่ไม่มีรูปแบบนี้มาเข้าขบวนการการถอดรหัส เพื่อเปลี่ยนข้อมูลให้เป็นข้อมูลชุดเดิม สำหรับวิธีการทางคณิตศาสตร์ที่จะกล่าวถึงนั้น เป็นการเข้ารหัสและถอดรหัสโดยใช้เลขฐานสอง กุญแจประกอบด้วยเลขฐานสอง 64 บิทซึ่งใช้ในขบวนการเข้ารหัสและถอดรหัส โดย 8 บิทใช้ในการตรวจสอบข้อผิดพลาด (Error Detection) ดังนั้นกุญแจที่ใช้จริง ๆ จึงมีขนาดเพียง 56 บิท

3.1.2.1 ขบวนการเข้ารหัส

สำหรับขบวนการเข้ารหัสนั้นได้แสดงไว้ตามรูปที่ 3-2 โดยข้อมูลเข้านั้นจะเป็นบล็อก T_I ซึ่งจะเข้าสู่ขบวนการการสับเปลี่ยนตำแหน่ง IP และจะได้ข้อมูลออกนั้นเป็น $T_0 = IP(T_I)$ ซึ่งข้อมูล T_0 จะเข้าสู่ขบวนการการสับเปลี่ยนผนวกกับรหัสกุญแจ 16 ครั้ง ซึ่งเรียกว่า Function_F เมื่อเสร็จขบวนการแล้วก็จะเข้าสู่การสับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1} ซึ่งผลลัพธ์ที่ได้ก็จะเป็นข้อมูลที่ผ่านขบวนการเข้ารหัสเรียบร้อยแล้ว ดังรูป



รูปที่ 3-2 บล็อกไดอะแกรมของการเข้ารหัสดีอีเอส

3.1.2.2 ขบวนการถอดรหัส

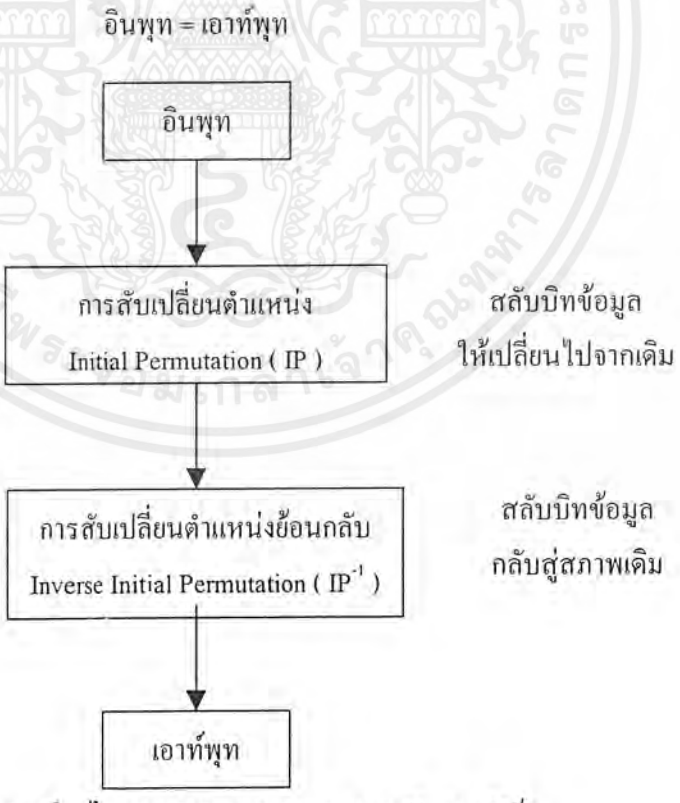
สำหรับขบวนการการถอดรหัสนั้น จะมีวิธีการในการดำเนินการอย่างเดิม แต่ในขบวนการการสับเปลี่ยนผนวกกับรหัสกุญแจ 16 ครั้งนั้นจะย้อนกลับ โดยหมายเลขกุญแจจะใช้ K_{16} , K_{15} , ... , K_1 ส่วนในขบวนการอื่น ๆ นั้นจะมีวิธีการกระทำเหมือนขบวนการเข้ารหัส

3.1.2.3 การสับเปลี่ยนตำแหน่ง

ขบวนการดีอีเอสใช้หลักการสับเปลี่ยนตำแหน่งหลายขั้นตอนทั้งในด้านการเข้ารหัส ถอดรหัส และการคำนวณหมายเลขกุญแจ การสับเปลี่ยนดังกล่าวได้แก่ ตาราง IP, IP^{-1}, E, P, PC_ 1 และ PC_2

จุดประสงค์ของการสับเปลี่ยน IP และ IP^{-1} คือการโยกย้ายบิทให้แตกต่างไปจากข้อมูลเดิม และเมื่อทำการสับเปลี่ยนย้อนกลับแล้วจะได้ข้อมูลเดิม

จากรูปที่ 3-3 จะทำให้การอธิบายดูง่ายขึ้น ถ้าตัดส่วนที่เป็นขบวนการสับเปลี่ยนผนวกกับรหัสกุญแจ 16 ชุดออก จะเห็นว่าข้อมูลเข้าและออกจะมีค่าเดียวกัน นั่นคือขบวนการสับเปลี่ยน IP จะรับอินพุทขนาด 64 บิทมาสับเปลี่ยนตำแหน่งตาม Initial Permutation (IP) และส่ง เอาท์พุท ให้กับ IP^{-1} ซึ่งเป็นการสับเปลี่ยนตำแหน่งย้อนกลับทำให้ได้ข้อมูลกลับสู่สภาพเดิม



รูปที่ 3-3 บล็อกไดอะแกรมแสดงกระบวนการ IP และ IP^{-1}

1	2	3	4	5	6	7	8
58	50	42	34	26	18	10	2
9	10	11	12	13	14	15	16
60	52	44	36	28	20	12	4
17	18	19	20	21	22	23	24
62	54	46	38	30	22	14	6
25	26	27	28	29	30	31	32
64	56	48	40	32	24	16	8
33	34	35	36	37	38	39	40
57	49	41	33	25	17	9	1
41	42	43	44	45	46	47	48
59	51	43	35	27	19	11	3
49	50	51	52	53	54	55	56
61	53	45	37	29	21	13	5
57	58	59	60	61	62	63	64
63	55	47	39	31	23	15	7

รูปที่ 3-4 รายละเอียดตารางสับเปลี่ยนตำแหน่ง IP

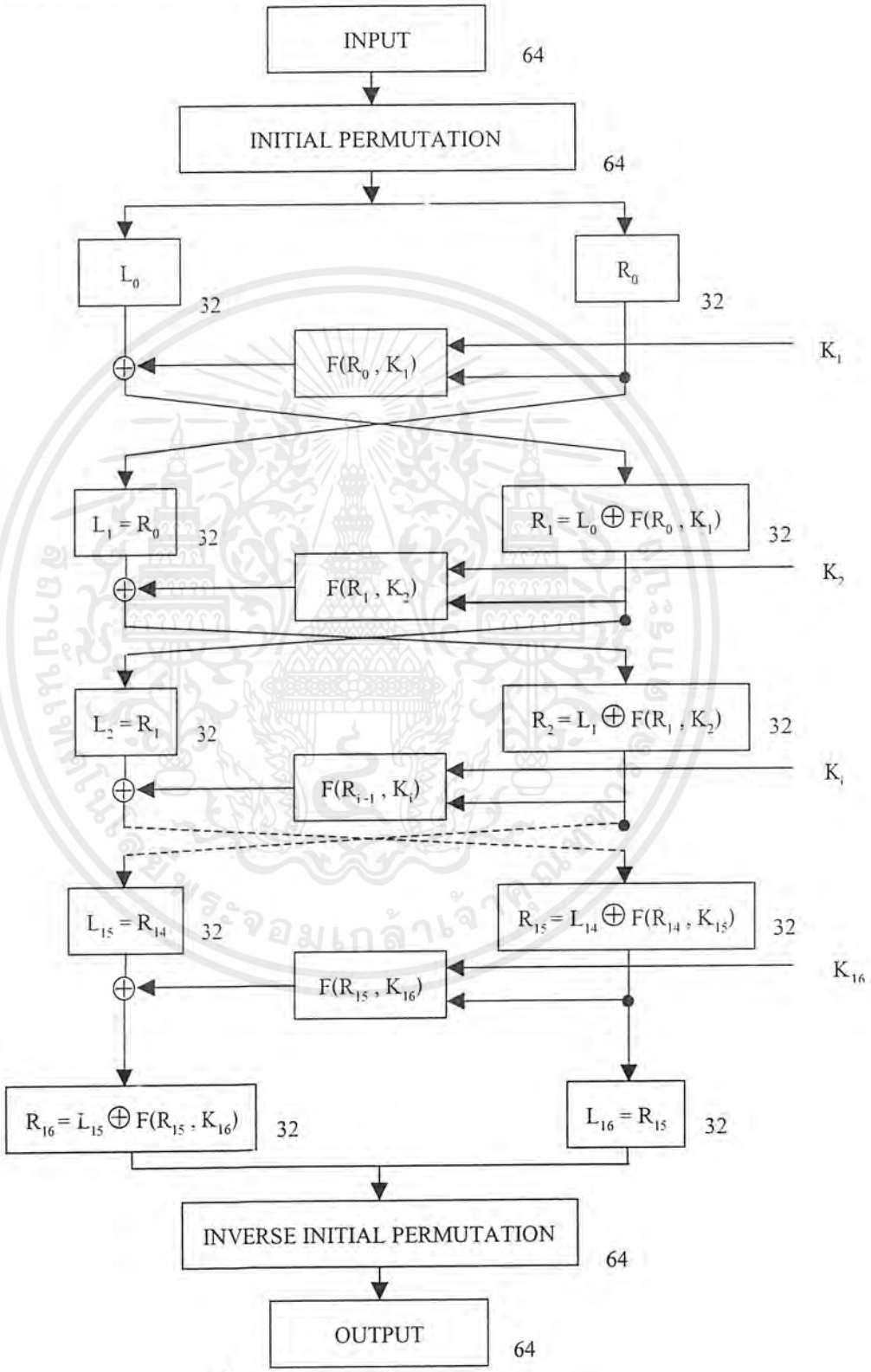
1	2	3	4	5	6	7	8
40	8	48	16	56	24	64	32
9	10	11	12	13	14	15	16
39	7	47	15	55	23	63	31
17	18	19	20	21	22	23	24
38	6	46	14	54	22	62	30
25	26	27	28	29	30	31	32
37	5	45	13	53	21	61	29
33	34	35	36	37	38	39	40
36	4	44	12	52	20	60	28
41	42	43	44	45	46	47	48
35	3	43	11	51	19	59	27
49	50	51	52	53	54	55	56
34	2	42	10	50	18	58	26
57	58	59	60	61	62	63	64
33	1	41	9	49	17	57	25

รูปที่ 3-5 รายละเอียดตารางสับเปลี่ยนตำแหน่ง IP^{-1}

IP ข้อมูลตำแหน่งบิทที่ 1 และ 64 ของรูปที่ 3-4 จะถูกสลับไปอยู่ตำแหน่งบิทที่ 58 และ 7

IP^{-1} ข้อมูลตำแหน่งบิทที่ 58 และ 7 ของรูปที่ 3-5 จะถูกสลับไปอยู่ตำแหน่งบิทที่ 1 และ 64

จุดประสงค์ของขบวนการสับเปลี่ยน IP และ IP^{-1} ก็คือ การโยกย้ายข้อมูลให้แตกต่างไปจากเดิม เพื่อให้สับสนและยากต่อการแกะรอย (Trace)



รูปที่ 3-6 บล็อกแสดงขั้นตอนการเข้ารหัส

จากรูปที่ 3-6 นั้น $F(R_{i-1}, K_i)$ คือ ขบวนการสับเปลี่ยนผนวกกับรหัสกุญแจ 16 ครั้ง แต่ละครั้งเรียกว่า Function_F ซึ่งอยู่ระหว่างการสับเปลี่ยนตำแหน่ง IP กับการสับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1} สำหรับสมการที่ใช้ในขบวนการเข้ารหัสนั้น เป็นดังนี้

$$\left.\begin{array}{l} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \end{array}\right\} \quad (1)$$

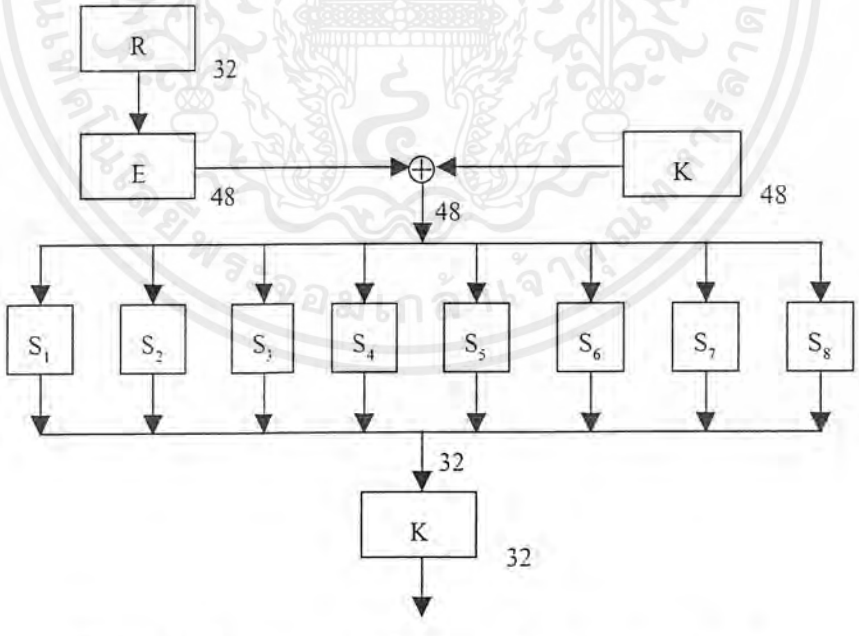
โดยที่ L_i คือ ข้อมูลขนาด 32 บิททางด้านซ้าย

R_i คือ ข้อมูลขนาด 32 บิททางด้านขวา

K_i คือ หมายเลขกุญแจที่มีความยาว 48 บิท

เครื่องหมาย $\oplus$ หมายถึงการทำเอ็กคลูซีฟออ (XOR) และขบวนการสับเปลี่ยนผนวกกับรหัสกุญแจมีทั้งหมด 16 รอบ แสดงในรูปที่ 3-6 รอบที่ 1 ถึง 15 จะทำการสับเปลี่ยนตามสมการ (1) ในรอบที่ 16 ซึ่งเป็นรอบสุดท้าย ด้านซ้าย L และด้านขวา R จะไม่สลับที่กัน ข้อมูล L_{16} R_{16} ถูกใช้เป็นข้อมูลเข้าเพื่อทำการสับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1}

3.1.2.4 Function_F และ S_box



รูปที่ 3-7 บล็อกไดอะแกรมแสดงขั้นตอนการสร้าง Function_F(R , K)

จากรูปที่ 3-7 แสดงการทำ Function_F(R_{i-1} , K_i) การทำงานเริ่มจาก R_{i-1} ถูกสับเปลี่ยนเป็น 48 บิท โดยใช้ตาราง E ในตารางที่ 3-2 จะแสดงถึงตำแหน่งบิท E ซึ่งเป็นลักษณะเดียวกับที่ใช้ในการทำสับเปลี่ยน

ตำแหน่ง IP และสับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1} มีส่วนที่ไม่เหมือนกันก็คือ R_{i-1} ถูกใช้มากกว่า 1 ครั้ง เช่น $R_{i-1} = r_1r_2 \ldots\ldots R_{32}$, $E(R_{i-1}) = r_{32}r_1r_2 \ldots\ldots r_{32}r_1$

ต่อมาจะนำ $E(R_{i-1})$ และ K_i มาทำ XOR ผลลัพธ์ที่ได้ถูกแบ่งเป็นข้อมูลขนาด 6 บิท 8 ชุด คือชุดที่ $B_1 \ldots\ldots\ldots B_8$ ซึ่งสามารถเขียนเป็นสมการได้ ดังนี้

$$E(R_{i-1}) \oplus K_i = B_1B_2 \ldots\ldots B_8 \longrightarrow (2)$$

ข้อมูลในแต่ละชุด B_i มีขนาด 6 บิท จะใช้เป็นข้อมูลเข้าใน Function (S_box) S_j และข้อมูลออกจะมีขนาด 4 บิท เขียนในรูปของฟังก์ชันได้เป็น $S_j(B_j)$

S_box จะมีทั้งหมด 8 ชุด หลังจากผ่านขบวนการ S_box จะได้ผลลัพธ์ขนาด 32 บิท และเข้าขบวนการสับเปลี่ยน P ซึ่งแสดงในตาราง P ตารางที่ 3-1 เป็นข้อมูลที่ได้จากทฤษฎีของดีอีเอส

ผลลัพธ์ของ S_box หลังการเข้าขบวนการสับเปลี่ยน P จะอยู่ในรูปของ $F(R_{i-1}, K_i)$ ซึ่งสามารถเขียนได้เป็น $P(S_1(B_1)\ldots..(S_8(B_8))$

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

ตารางที่ 3-1 ตาราง P

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

ตารางที่ 3-2 ตาราง E

สำหรับวิธีการเข้า S_box นั้นจะสามารถอธิบายได้ ดังนี้คือ ให้ B_j เป็นข้อมูลเข้ามีขนาด 6 บิท ($B_1B_2B_3B_4B_5B_6$) ให้ j เป็นเลขจำนวนเต็มมีค่า 1 ถึง 8 แบ่ง B_j ออกเป็น 2 ส่วน เรียกว่า Row และ Column ให้ในส่วนของ Row = b_1b_6 และส่วนของ Column = $b_2b_3b_4b_5$ นำค่า Row และ Column มาเทียบในตาราง S จะได้ผลลัพธ์เป็นค่า 4 บิท ดังนั้นสามารถจะบอกได้ว่า S(B) ให้ผลลัพธ์ขนาด 4 บิท

<u>ตัวอย่าง</u> ถ้า B_1 = 101010 S_1 จะได้ผลลัพธ์เป็น Row = 10 = 2 Column = 0101 = 5 เทียบในตาราง S_1 จะได้ 6 ซึ่งเท่ากับ 0110

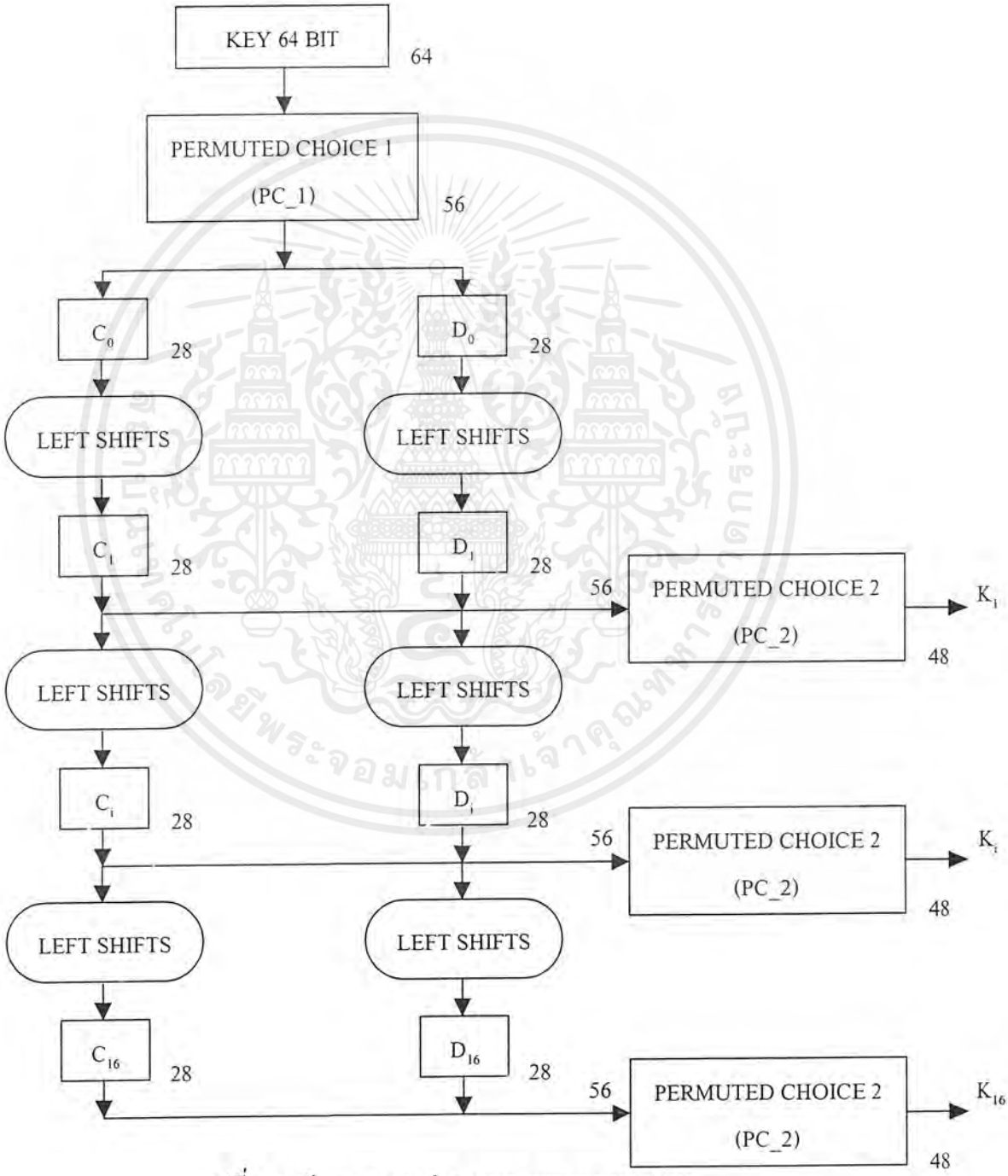
ถ้า B_7 = 100101 S_7 จะได้ผลลัพธ์เป็น Row = 11 = 3 Column = 0010 = 2 เทียบในตาราง S_1 จะได้ 13 ซึ่งเท่ากับ 1101

	Column																
Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Box
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S1
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	S2
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	S3
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S4
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	S5
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	5	8	0	13	3	4	14	7	5	11	
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	S6
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	S7
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	S8
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

ตารางที่ 3-3 ตาราง S_box

3.1.2.5 การคำนวณหมายเลขกุญแจ (Subkey)

ในแต่ละรอบของการสับเปลี่ยนผนวกกับรหัสกุญแจ 16 ชุด จะใช้หมายเลขกุญแจแต่ละชุดมีขนาด 48 บิท หมายเลขกุญแจ 16 ชุดได้มาจากกุญแจ ดังรูปที่ 3-8 กุญแจเป็นข้อมูลเข้ามีขนด 64 บิท โดย 8 บิทในตำแหน่ง 8, 16, ..., 64 ใช้เป็นบิทตรวจสอบข้อผิดพลาด (Parity bit) การทำขบวนการสับเปลี่ยน Permuted PC_1 หรือ Permuted choice 1 จะตัดในส่วนของบิทตรวจสอบข้อผิดพลาดออก และใช้ 56 บิท ที่เหลือเท่านั้น สำหรับตารางสับเปลี่ยนตำแหน่งของ PC_1 แสดงไว้ในตารางที่ 3-4



รูปที่ 3-8 บล็อกแสดงการคำนวณหมายเลขกุญแจ 16 ชุด

ผลลัพธ์หลังจากการทำ PC_1 จะถูกแบ่งเป็น 2 ส่วนที่เท่า ๆ กัน เรียกว่า C และ D ใช้หาค่าหมายเลขกุญแจ K_i

กำหนดให้ C_i และ D_i ซึ่งได้มาจาก C. D ใช้หาค่า K_i มีสมการเป็น

$$C_i = LS_i (C_{i-1})$$

$$D_i = LS_i (D_{i-1})$$

LS_i คือ การทำการหมุนทางซ้ายเท่ากับจำนวนครั้งที่กำหนดในตารางที่ C_0 และ D_0 เป็นข้อมูลเริ่มแรกของ C และ E ดังนั้นหมายเลขกุญแจ K_i เขียนสมการได้เป็น

$$K_i = PC\_2 (C_i D_i)$$

สำหรับในส่วนของขบวนการสับเปลี่ยนตำแหน่ง PC_2 นั้นจะอยู่ในตารางที่ 3-5

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

ตารางที่ 3-4 ตาราง PC_1

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

ตารางที่ 3-5 ตาราง PC_2

กำหนดให้ หมายเลขกุญแจเท่ากับ D465 9CAE 367C D9EB นำหมายเลขกุญแจที่ได้ทำการสับเปลี่ยน PC_1 ได้ผลลัพธ์เท่ากับ CDE3 BA70 983F EC50 ข้อมูลนี้แบ่งออกเป็น 2 ส่วนคือ

C_0 = CDE3 BA70
D_0 = 983F EC50

สำหรับขั้นตอนต่อไปจะนำ C_0 และ D_0 มาหมุนตามจำนวนครั้งในตารางที่ 3-6

ลำดับ	จำนวนครั้ง
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2

ลำดับ	จำนวนครั้ง
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

ตารางที่ 3-6 ตารางแสดงจำนวนครั้งในการหมุนซ้าย (LS)

	C_0 = CDE3BA7 0	D_0 = 983F EC5 0
ลำดับที่ 1 หมุน 1 ครั้ง	C_1 = 9BC7 74F 0	D_1 = 983F EC5 0
ลำดับที่ 2 หมุน 1 ครั้ง	C_2 = CDE3 BA7 0	D_2 = 983F EC5 0
ลำดับที่ 3 หมุน 1 ครั้ง	C_3 = CDE3 BA7 0	D_3 = 983F EC5 0
:	:	:
:	:	:
ลำดับที่ 16 หมุน 1 ครั้ง	C_{16} = CDE3 BA7 0	D_{16} = 983F EC5 0

นำข้อมูลที่ได้หลังจากการหมุนในแต่ละลำดับเข้าขบวนการสับเปลี่ยน PC_2 ได้เป็นกุญแจ K_1 ถึง K_{16} ดังตารางที่ 3-7 ซึ่งกุญแจ 16 ชุดนี้จะนำไปใช้ในขบวนการเข้าและถอดรหัสต่อไป

สำหรับการทำขบวนการย้อนกลับ หรือการถอดรหัส (Deciphering) ทำโดยใช้วิธีการเดิม แต่หมายเลขกุญแจนั้น จะใช้สลับเป็น K_{16} , K_{15} , … , K_1 เป็นจำนวน 16 ครั้ง ซึ่งเขียนเป็นสมการได้ ดังนี้

$R_{i-1} = V_i$
$R_i = L_{i-1} \oplus F(R_{i-1}, K_{17-i})$

ซึ่งในการทำกระบวนการย้อนกลับนั้นจะย้อนกลับในเฉพาะส่วนของกุญแจจาก 16 ถึง 1 เท่านั้น แต่ในส่วนของวิธีจะไม่ย้อนกลับ

ลำดับ	C	D	#Key (K)
1	9BC774F0	307FD8B0	2319 173C 2915 3D31
2	378EE9F0	60FFB160	343E 171A 3F35 2810
3	DE3BA7C0	83FEC590	1F0C 3E33 3414 CD3E
4	78EE9F30	FFB16600	3D3B 240F 251B 2A0C
5	E3BA7CD0	3FEC5980	0A3A 1B17 3C03 1B35
6	8EE9F370	FFB16600	1F15 323E 0E3A 2A2F
7	3BA7CDE0	FEC59830	3D2A 0738 0D27 3613
8	EE9F3780	FB1660F0	272C 3837 0B32 0537
9	DD3E6F10	F62CC1F0	2F1E 3D03 3924 0F33
10	74F9BC70	D8B307F0	1837 1E2F 153A 2D0B
11	D3E6F1D0	62CC1FF0	3E19 1715 3D29 1518
12	4F9BC770	8B307FD0	151C 2B3B 121B 1D2E
13	3E6F1DD0	2CC1FF60	2D3F 0636 1F0F 3228
14	F9BC7740	B307FD80	2F20 3F07 1A05 313F
15	E6F1DD30	CC1FF620	1E37 083D 233F 223A
16	CDE3BA70	983FEC50	1B1F 332D 2216 3C3B

ตารางที่ 3-7 ค่าหมายเลขกุญแจที่คำนวณได้ตามขบวนการดีอีเอส

3.2 ภาษา VHDL

3.2.1 แนะนำ VHDL

ในช่วงฤดูร้อนของปี 1981 สถาบันเพื่อการป้องกัน (The Institute for Defence Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่งเพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า วีเอชดีแอล (VHDL : VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวกับวงจรรวมที่มีความเร็วสูงมาก ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้วีเอชดีแอลกลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของภาษาในด้านของการกำหนดพฤติกรรมการทำงานของวงจร ทำให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งของวงจรดิจิตอลทั่ว ๆ ไปและในระบบที่แตกต่างออกไป ข้อดีหลักที่สำคัญของวีเอชดีแอลก็คือภาษานี้สามารถที่จะถูกใช้ได้ตลอดในทุก ๆ ระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูงจนถึงระดับต่ำสามารถใช้ภาษาเดียวกันได้โดยตลอดทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

3.2.1.1 ข้อกำหนด

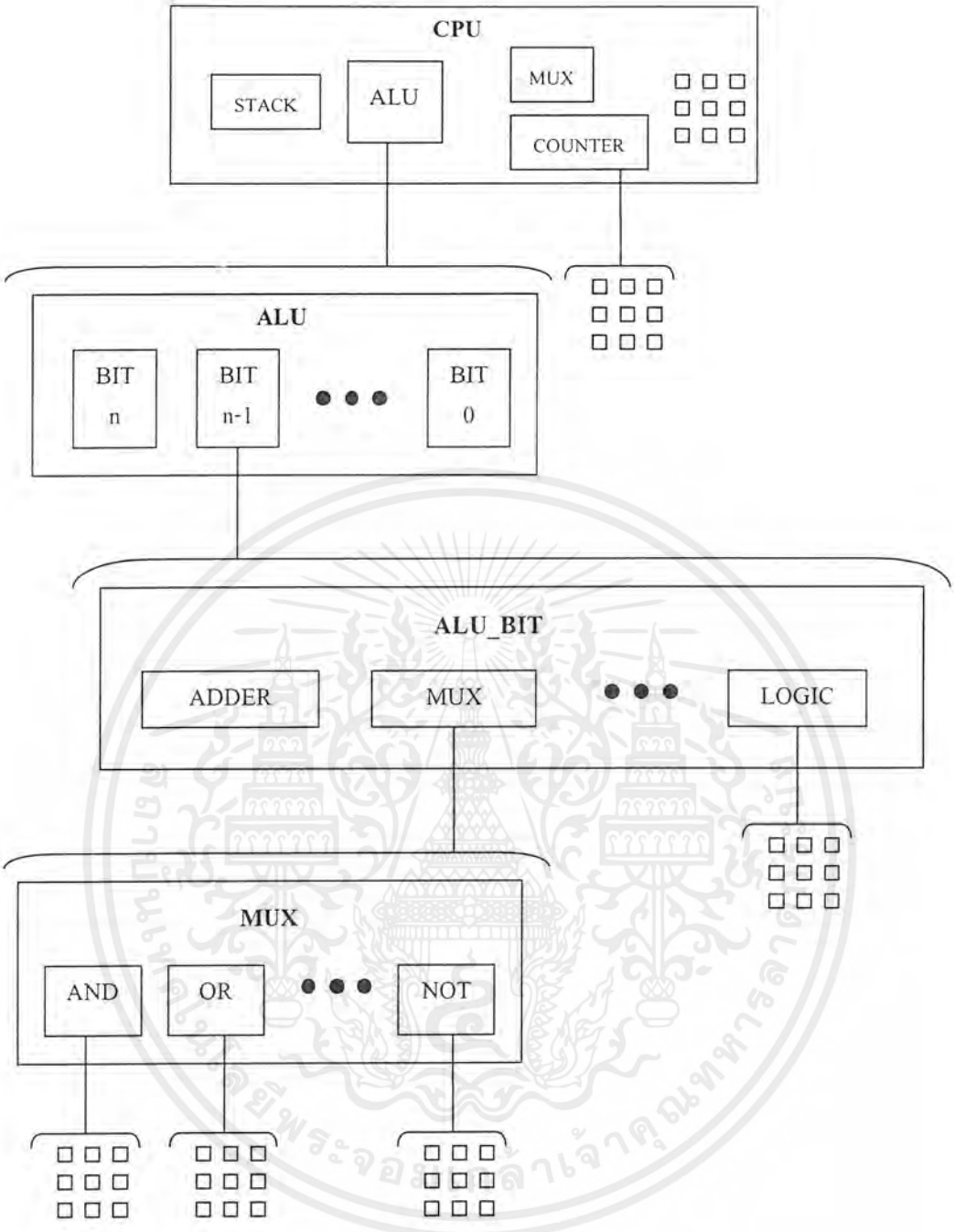
ในเอกสารของ DoD (Department of Defense Requirements for Hardware Description Languages) ซึ่งออกมาในเดือนมกราคมปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษาวีเอชดีแอลไว้ดังนี้

3.2.1.1.1 ลักษณะทั่วไป

วีเอชดีแอลเป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการจำลองแบบ (simulation) การสังเคราะห์ (synthesis) และการทดสอบ นอกจากนั้นยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับบนจนถึงระดับเกท เนื่องจากในการทำงานของระบบดิจิตอลจริง ๆ ทุกองค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมกัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งด้วยเช่นกัน

3.2.1.1.2 สนับสนุนการออกแบบแบบลำดับขั้น

การออกแบบแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลาย ๆ ระดับในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตัวเองหรือถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อย ๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานได้โดยลักษณะแบบโครงสร้างได้ดังรูป 3-9



รูปที่ 3-9 ตัวอย่างการออกแบบแบบลำดับขั้น

จากรูปที่ 3-9 จะเป็นการบรรยายถึงการทำงานของไมโครโพรเซสเซอร์ โดยจะเริ่มจากระดับบนสุดคือจะประกอบไปด้วยสแต็ค (stack) ALU ตัวนับ (counter) ซึ่งตัวนับสามารถที่จะบรรยายลักษณะเชิงพฤติกรรมซึ่งเป็นระดับล่างสุดได้แล้ว ส่วน ALU นั้นยังสามารถที่จะแบ่งเป็นส่วนย่อยได้อีก ซึ่งลักษณะนี้จะใช้การบรรยายเชิงโครงสร้างเพื่อบอกว่าภายใน ALU นั้นมีส่วนประกอบอะไรบ้าง ซึ่งส่วนย่อยภายใน ALU นั้นก็ยังสามารถที่จะแบ่งย่อยออกไปได้อีกขั้นหนึ่ง โดยสรุปแล้วเราจำเป็นจะต้องแบ่งส่วนประกอบให้เป็นส่วนย่อยที่สุดก่อน เพื่อให้มีความสะดวกในการเขียนบรรยายเชิงพฤติกรรม

3.2.1.1.3 ไลบรารี (Library)

วีเอชดีแอลได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารีหรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่น ๆ สามารถนำไปใช้ได้ด้วย

3.2.1.1.4 ลำดับคำสั่ง

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของวีเอชดีแอลก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if-then-else และ loop ได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของวีเอชดีแอลก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

3.2.1.1.5 การกำหนดคุณสมบัติ (Generic Design)

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่น ๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้น ๆ ภาษาสำหรับการออกแบบที่ดีควรจะสามารถให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วยเช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่น ๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษาวีเอชดีแอลด้วยเช่นกัน

3.2.1.1.6 ชนิดของข้อมูล

วีเอชดีแอลสามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด BIT และ BOOLEAN เท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับหรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเองก็ได้

3.2.1.1.7 โปรแกรมย่อย (Subprograms)

ความสามารถในการใช้ฟังก์ชันและโพรซีเยอร์ก็เป็นข้อกำหนดอีกอย่างหนึ่งในวีเอชดีแอลเราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก (logic) การกำหนดตัวกระทำต่าง ๆ ทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

3.2.1.1.8 การควบคุมเวลา

ภาษาวีเอชดีแอลอนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ (event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

3.2.1.1.9 การกำหนดแบบโครงสร้าง (Structural Specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุก ๆ ระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวมที่เกิดจากองค์ประกอบย่อย ๆ ที่แตกต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

3.2.1.2 องค์ประกอบพื้นฐานใน VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบในวีเอชดีแอลประกอบด้วยส่วนกำหนดการเชื่อมต่อ (interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (architecture) ดังแสดงในรูปที่ 3-10 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำว่า ENTITY ตามด้วยชื่อขององค์ประกอบและคำว่า IS ภายในบรรยายถึงพอร์ตการติดต่อ อินพุท-เอาท์พุทพอร์ตขององค์ประกอบ ส่วนลักษณะภายนอกอื่น ๆ เช่น เวลาอุณหภูมิก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท-เอาท์พุทและพารามิเตอร์อื่น ๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูป การบรรยายหน้าที่ขององค์ประกอบเริ่มต้นหลังจากคำว่า BEGIN เป็นต้นไป

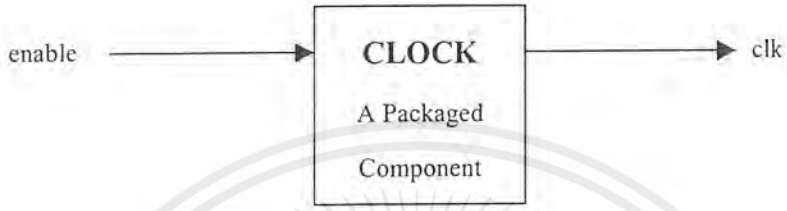
```
ENTITY component_name IS
        input and output ports.
        physical and other parameters.
END component_name;
```

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

รูปที่ 3-10 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

3.2.1.2.1 การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอกอื่น ๆ ดังตัวอย่างในรูปที่ 3-11 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ clock_component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็บ IN และ OUT กำหนดโหมดของสัญญาณเป็นอินพุทหรือเอาท์พุท BIT แสดงชนิดของข้อมูล




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

รูปที่ 3-11 บล็อกไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_component

3.2.1.2.2 การกำหนดรูปแบบการบรรยาย

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในเทอมของอินพุทหรือในรูปขององค์ประกอบอื่น ๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 3-12 ซึ่งเป็นการบรรยายในเชิงพฤติกรรม ภายในโพรเซส (process) กำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น '0' ถ้าสัญญาณ en มีค่าเป็น '1' ค่าของ periodic จะถูกคอมพลีเมนต์(complement) และส่งค่าให้กับ clk ซึ่งเป็นสัญญาณเอาท์พุท คำสั่ง WAIT กำหนดให้สัญญาณมีคาบเป็นเวลา 1 ไมโครวินาที

3.2.1.2.3 โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเยอร์ในวีเอชดีแอลเปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่ว ๆ ไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าเราใช้ฟังก์ชันแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกจริง ๆ ในขณะที่ถ้าเราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 3-13 แสดงการใช้โพรซีเยอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิทเป็นค่าจำนวนเต็ม รูปที่ 3-14 แสดงการใช้ฟังก์ชันโดยกำหนดให้ x เป็นตัวแปรชนิดบิทแทนการกระทำในสมการบูลีน

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

รูปที่ 3-12 การบรรยายเชิงพฤติกรรมของ clock_component

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

รูปที่ 3-13 การใช้โพรซีเยอร์

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

รูปที่ 3-14 การใช้ฟังก์ชัน

3.2.1.2.4 โอเปอเรเตอร์ (VHDL Operators)

การบรรยายเชิงพฤติกรรมในวีเอชดีแอลก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 3-15

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

รูปที่ 3-15 โอเปอเรเตอร์ในวีเอชดีแอล

3.2.1.2.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

ในวงจรอิเล็กทรอนิค อุปกรณ์ทุก ๆ ตัวจะอยู่ในสภาพเตรียมพร้อมเสมอและจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องเสมอในทุก ๆ เหตุการณ์ที่เกิดขึ้นวีเอชดีแอลเป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพร้อมกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของสถาปัตยกรรมบรรยายจะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โพรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลาย ๆ โพรเซสอยู่ภายในโครงสร้างเดียวกันทุกโพรเซสก็จะทำงานไปพร้อม ๆ กันด้วย

3.2.1.2.6 สัญญาณและตัวแปร

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณเช่น o <= i AFTER 50 NS หมายถึงกำหนดค่าของสัญญาณ i ให้กับ o หลังจากเวลาผ่านไป 50 นาโนวินาที ในทางตรงข้าม ตัวแปรมีลักษณะเป็นเสมือนตัวกลางซอฟต์แวร์ที่ใช้ในการส่งผ่านข้อมูลและไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้

กับตัวแปรจะใช้สัญลักษณ์ := ตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่ง เช่นใน ฟังก์ชัน โพรซีเยอร์และโพรเซส

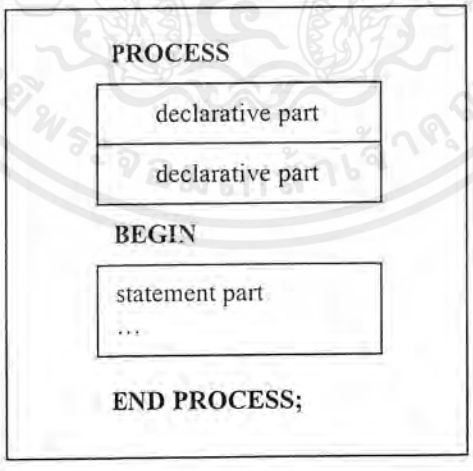
3.2.2 การบรรยายเชิงพฤติกรรม (Behavioral Description of Hardware)

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรมเป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอลิทึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามาโดยไม่คำนึงถึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในจะเป็นอย่างไร ในหัวข้อนี้จะแสดงให้เห็นถึงประโยชน์ในการใช้โปรแกรมย่อยที่จำลองรูปแบบอินพุทและเอาท์พุทในระดับพฤติกรรมแทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่าง ๆ ที่ควรรู้

3.2.2.1 โพรเซส (Process)

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญญาณ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติคำสั่งพร้อม ๆ กันกับโพรเซสอื่น ๆ ที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์การกำหนดค่าให้กับสัญญาณ

การบรรยายโพรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS ในรูปที่ 3-16 แสดงส่วนประกอบของการบรรยายแบบโพรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 3-16 รูปแบบของการบรรยายแบบโพรเซส

3.2.2.1.1 การกำหนดตัวกระทำภายในโพรเซส

ตัวกระทำภายในโพรเซสมี 3 ชนิดคือ ตัวแปร (variable) ไฟล์ (file) และตัวคงที่ (constant) ซึ่งตัวกระทำทั้งสามชนิดนี้หากมีการประกาศไว้ในโพรเซสใดก็จะใช้ได้เฉพาะในโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซสสามารถทำได้โดยใช้สัญญาณหรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE รูปที่ 3-17 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN ค่าเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียวเท่านั้น แต่ค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้โปรแกรมย่อยนั้น ๆ

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

รูปที่ 3-17 ตัวอย่างการประกาศตัวกระทำภายในโพรเซส

3.2.2.1.2 การกำหนดการกระทำภายในโพรเซส

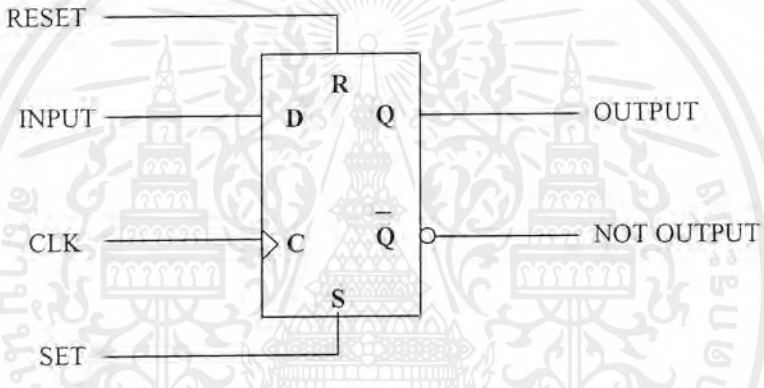
การกระทำใด ๆ ภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับเสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการทำซ้ำได้เช่น IF-THEN-ELSE, CASE-WHEN, FOR LOOP และ WHILE LOOP ดังตัวอย่างในรูปที่ 3-18

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

รูปที่ 3-18 เงื่อนไขการกระทำในโพรเซส

3.2.2.1.3 การกระตุ้นและยับยั้งการกระทำของโพรเซส (Seasitivity List)

การกระทำภายในโพรเซสจะเตรียมพร้อมและมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์เกิดขึ้น อย่างไรก็ตามเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโพรเซสได้โดยการกำหนดรายการของสัญญาณที่ต้องการให้โพรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้น เหตุการณ์ใด ๆ ที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโพรเซสรายการของสัญญาณนี้เรียกว่า Sensitive List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS รูปที่ 3-19 (a) แสดงตัวอย่างโมเดลและ (b) แสดงตัวอย่างการบรรยายการเชื่อมต่อของ D-flipflop รูปที่ 3-20 แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop (a) การใช้ตัวกระทำภายนอกโพรเซส และ (b) การใช้ตัวกระทำภายในโพรเซส โดยมีรายการของสัญญาณ (rst, set, clk) เป็นตัวกระตุ้นการปฏิบัติงานภายในโพรเซส



(a)

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

(b)

รูปที่ 3-19

(a) ตัวอย่างโมเดล D-FlipFlop

(b) การบรรยายการเชื่อมต่อของ D-FLIPFLOP

3.2.2.2 การตรวจสอบการกระทำ (Assertion Statement)

วีเอชดีแอลได้จัดเตรียมรูปแบบคำสั่งเพื่อใช้ในการตรวจดูการกระทำต่าง ๆ ที่เกิดขึ้นภายในอุปกรณ์ไว้ดังนี้

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

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

(a)

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

(b)

รูปที่ 3-20 การบรรยายเชิงพฤติกรรมของ D-FlipFlop

(a) การใช้ตัวกระทำภายนอกโพรเซส

(b) การใช้ตัวกระทำภายในโพรเซส

คำสั่งนี้จะถูกปฏิบัติเมื่อเงื่อนไข assertion_condition มีค่าเป็น FALSE และจะแสดงข้อความ "reporting_message" ส่วนพารามิเตอร์ severity_level จะเป็นตัวกำหนดระดับความรุนแรงที่เกิดขึ้น ซึ่งสามารถแบ่งได้เป็น 4 ระดับคือ NOTE, WARNING, ERROR และ FAILURE ระดับความรุนแรง ERROR หรือ FAILURE จะทำให้การเลียนแบบการทำงานหยุดทันทีหลังจากที่ได้แสดงข้อความ reporting_message แล้ว ส่วนระดับ NOTE หรือ WARNING จะแสดงข้อความ reporting_message แต่ยังคงเลียนแบบการทำงานต่อไป สมมติว่าเราต้องการตรวจสอบการทำงานของ D-FlipFlop ในตัวอย่างข้างต้นว่าในขณะที่กำลังทำงานได้เกิดสัญญาณในกรณีที่ไม่พึงประสงค์ขึ้นมาหรือไม่เช่น สัญญาณ set และ rst มีค่าเป็น '1' พร้อมกันก็สามารถตรวจสอบได้โดยแทรกคำสั่ง ASSERT เพื่อตรวจสอบสัญญาณในโพรเซสได้ดังรูปที่ 3-21

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

รูปที่ 3-21 การใช้ ASSERT

3.2.2.3 การหยุดรอ

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณมีอยู่ 4 รูปแบบดังนี้

WAIT FOR waiting_time;

WAIT ON waiting_sensitivity_list;

WAIT UNTIL waiting_condition;

WAIT;

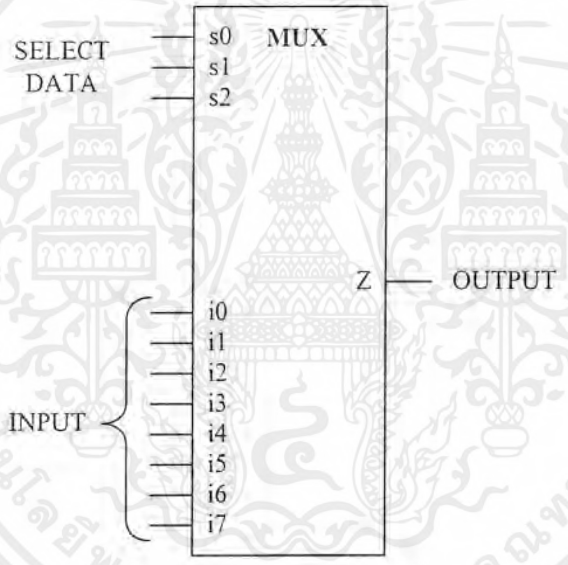
คำสั่งหยุดรอสามารถใช้ได้ภายในโพรเซสและโพรซีเยอร์ที่ไม่ถูกกำหนด Sensitivity List ไว้เท่านั้น WAIT FOR จะหยุดรอเป็นเวลาเท่ากับ waiting_time WAIT ON จะหยุดรอจนกว่าจะมีเหตุการณ์เกิดขึ้นกับสัญญาณ waiting_sensitivity_list WAIT UNTIL จะหยุดรอจนกว่าเงื่อนไข waiting_condition เปลี่ยนจาก FALSE เป็น TRUE WAIT จะหยุดรอตลอดไป

3.2.3 การบรรยายเชิงกระแสข้อมูล

การบรรยายเชิงกระแสข้อมูลนั้นเป็นการบรรยายถึงการเลื่อนไหลของข้อมูลผ่านรีจิสเตอร์และบัสของระบบ เป็นระดับชั้นของการบรรยายที่อยู่ตรงกลางระหว่างการบรรยายเชิงพฤติกรรมและการบรรยายเชิงโครงสร้าง เครื่องมือที่ใช้ในการควบคุมการเคลื่อนไหวของข้อมูลได้แก่ conditional, selected และ guarded ในหัวข้อนี้จะแสดงให้เห็นถึงลักษณะและรูปแบบของการบรรยายเชิงกระแสข้อมูลรวมถึงข้อกำหนดและเงื่อนไขต่าง ๆ พร้อมทั้งตัวอย่างประกอบ

3.2.3.1 การกำหนดเลือกข้อมูล (Multiplexing and Data Selection)

ในระบบดิจิตอล โครงสร้างของอุปกรณ์ฮาร์ดแวร์ส่วนใหญ่จะถูกใช้สำหรับการคัดเลือกและการนำข้อมูลเข้าสู่บัสและรีจิสเตอร์ รูปที่ 3-22 แสดงตัวอย่างโมเดลและการบรรยายเชิงกระแสข้อมูล 8-to-1 มัลติเพลคเซอร์



รูปที่ 3-22 8-to-1 มัลติเพลคเซอร์

ในรูปที่ 3-22 (b) sel_lines เป็นสัญญาณที่กำหนดขึ้นมาเพื่อใช้เป็นสัญญาณในการเลือกอินพุททั้ง 8 โดยเอาท์พุท z จะเปลี่ยนไปตามอินพุทตัวใดนั้นก็ขึ้นอยู่กับสัญญาณที่เข้ามายังขา s0,s1,s2 โดยเอาท์พุทจะเปลี่ยนแปลงตามอินพุทหลังจากเวลาผ่านไป 3 นาโนวินาที

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

ARCHITECTURE  dataflow OF  mux_8_to_1 IS
        SIGNAL sel_lines : BIT_VECTOR (2 DOWNTO 0);
BEGIN
        sel_lines <= s2 & s1 & s0;
        WITH sel_lines SELECT
           z <=  i7  AFTER 3 NS WHEN "111",
                 i6  AFTER 3 NS WHEN "110",
                 i5  AFTER 3 NS WHEN "101",
                 i4  AFTER 3 NS WHEN "100",
                 i3  AFTER 3 NS WHEN "011",
                 i2  AFTER 3 NS WHEN "010",
                 i1  AFTER 3 NS WHEN "001",
                 i0  AFTER 3 NS WHEN "000";
END dataflow;
```

(b)

รูปที่ 3-22 8-to-1 มัลติเพลคเซอร์

(a) โมเดล

(b) การบรรยายเชิงกระแสข้อมูล

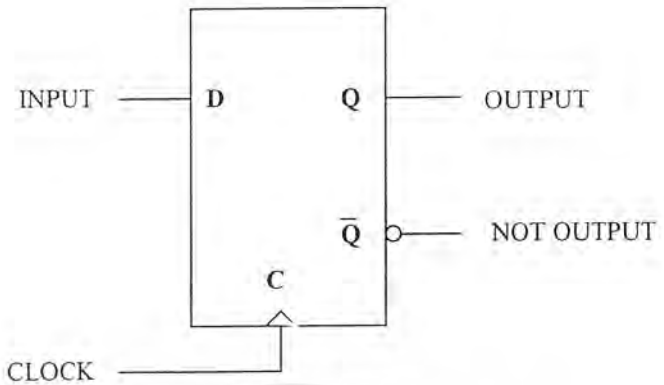
3.2.3.2 การกำหนดการ์ด

ในการบรรยายเชิงกระแสข้อมูลเราสามารถตั้งเงื่อนไขสำหรับการกำหนดค่าสัญญาณใด ๆ ได้ โดยใช้คำสั่ง GUARDED ซึ่งมีรูปแบบในการเขียนดังนี้

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

รูปที่ 3-23 แสดงตัวอย่างการใช้ GUARDED ในการบรรยายการทำงานของ d_flipflop การกำหนด GUARDED ทำได้โดยใช้รูปแบบของคำสั่ง BLOCK ตามด้วยเงื่อนไขภายในวงเล็บ ซึ่งผลลัพธ์ที่ได้จากวงเล็บนี้ก็คือ GUARDED นั่นเอง ในรูปเป็นการบรรยายการทำงานของ d_flipflop ที่มีการทำงานเมื่อสัญญาณนาฬิกามีการเปลี่ยนแปลงจาก '0' เป็น '1' หรือสัญญาณขาขึ้น จะเห็นว่า GUARDED ถูกใช้เป็นเงื่อนไขในการกำหนดค่าให้กับ q และ qb นั่นคือถ้าเงื่อนไขภายในวงเล็บมีค่าเป็น TRUE (GUARDED

เป็น TRUE) ค่า d และ NOT d จะถูกส่งค่าให้กับ q และ qb หากเงื่อนไขภายในวงเล็บมีค่าเป็น FALSE (GUARDED เป็น FALSE) ค่า d และ NOT d ก็จะไม่ถูกส่งค่าให้กับ q และ qb



(a)

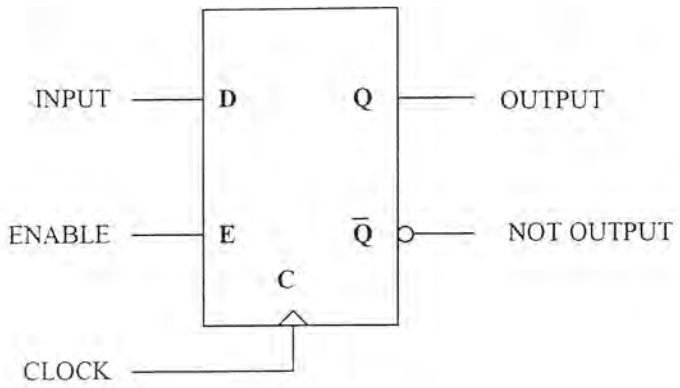
```
ENTITY d_flipflop IS
        GENERIC (delay 1 : TIME := 4 NS; delay2 : TIME := 5 NS;)
        PORT (d.c : IN BIT; q, qb : OUT BIT);
END d_flipflop;

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

(b)

รูปที่ 3-23 ตัวอย่างการใช้ GUARDED

3.2.3.3 การกำหนดโครงข่ายของการ์ด

เราสามารถกำหนดการ์ดเป็นโครงข่ายซ้อน ๆ กันได้โดยการกำหนด BLOCK ซ้อน ๆ กันซึ่งจะทำให้เราสามารถออกแบบอุปกรณ์ที่มีการทำงานที่ซับซ้อนมากกว่าได้ พิจารณา d_flipflop ในรูปที่ 3-24 แสดงตัวอย่างการบรรยายการทำงานที่ซับซ้อนมากกว่า d_flipflop ในตัวอย่างข้างต้น ซึ่งนอกจากสัญญาณอินพุท d จะถูกส่งผ่านไปยัง q โดยมีเงื่อนไขว่า (c = '1' AND NOT c'STABLE) ต้องเป็น TRUE แล้วในกรณีนี้ยังมีเงื่อนไขเพิ่มเติมอีกว่าสัญญาณ enable (e) ต้องเป็น TRUE ด้วย



(a)

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

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

(b)

รูปที่ 3-24 ตัวอย่างการใช้ NESTING GUARDED

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

3.2.3.4 การเลือกสรรข้อมูล

ในทางฮาร์ดแวร์มีหลาย ๆ กรณีที่เรามีความจำเป็นต้องเชื่อมต่อหลาย ๆ เอาท์พุทไปยังอินพุทใดอินพุทหนึ่ง ซึ่งในกรณีเช่นนี้อาจจะทำให้เกิดความสับสนของสัญญาณที่ปะปนกันทำให้ไม่สามารถรู้ค่าที่แน่นอนได้ ในทางวีเอชดีแอลหมายถึงการกำหนดค่าจากหลาย ๆ สัญญาณให้กับสัญญาณเดียวซึ่งก็จะให้

ผลลัพธ์ที่สับสนได้เหมือนกัน ดังตัวอย่างในรูปที่ 3-25 ซึ่งจะไม่สามารถบอกได้เลยว่าค่าของ circuit_node มีค่าเป็นอะไรถ้าสมมุติว่าค่าอินพุท a เป็น '1' และค่าอินพุทอื่น ๆ เป็น '0'

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

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

รูปที่ 3-25 การกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน

กรณีเช่นนี้เราสามารถแก้ไขได้โดยการกำหนดรูปแบบฟังก์ชันขึ้นมาเพื่อใช้เลือกสรรข้อมูล รูปที่ 3-26 แสดงตัวอย่างฟังก์ชันเลือกสรรข้อมูลแบบ AND ซึ่งกำหนดให้สัญญาณที่เข้ามาที่โหนดเดียวกันจะถูกนำมารวมกันในลักษณะของการ AND กันเสียก่อน

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

รูปที่ 3-26 ฟังก์ชันเลือกสรรข้อมูลแบบ anding

นำฟังก์ชันในรูปที่ 3-26 มาใช้ร่วมกับการบรรยายในรูปที่ 3-25 ก็สามารถแก้ปัญหาเรื่องความสับสนของข้อมูลได้ดังรูปที่ 3-27

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

รูปที่ 3-27 การใช้ฟังก์ชัน anding กับการเลือกสรรข้อมูล

3.2.4 การบรรยายเชิงโครงสร้าง

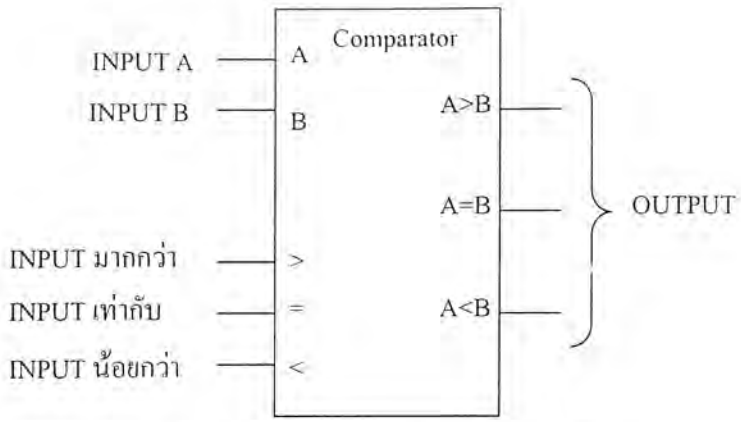
การบรรยายการทำงานของระบบในเชิงโครงสร้างจะต้องแสดงรายการของอุปกรณ์ทั้งหมดที่ใช้ในระบบนั้นและต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ ด้วย เพราะว่าการบรรยายในระดับนี้เป็นการบรรยายที่ใกล้เคียงลักษณะของฮาร์ดแวร์จริงที่สุด วีเอชดีแอลได้จัดเตรียมเครื่องมือและลักษณะโครงสร้างของการบรรยายในระดับนี้ไว้ที่สำคัญ 4 ลักษณะคือ 1) ความสามารถในการเลือกหรือกำหนดอุปกรณ์ที่ต้องการได้จากไลบรารี 2) การสร้างไลบรารีเพื่อเก็บอุปกรณ์ที่ผู้ใช้ออกแบบไว้เองได้ 3) กลไกในการเชื่อมต่อระหว่างอุปกรณ์ 4) โครงสร้างการกำหนดอุปกรณ์ชนิดเดียวกันซ้ำ ๆ กัน

3.2.4.1 ไลบรารี

ภายในไลบรารีนั้นจะมีเกทพื้นฐานทางดิจิตอลไว้ให้ใช้งานอยู่มากมายด้วยกันยกตัวอย่างเช่น อินเวอเตอร์ แอนเกท ออเกท ซึ่งเราไม่จำเป็นที่จะต้องบรรยายถึงพฤติกรรมของอุปกรณ์พวกนี้อีกเราสามรถนำมาใช้ในการสร้างวงจรที่ซับซ้อนต่อไปได้เลย

3.2.4.2 การเชื่อมต่ออุปกรณ์พื้นฐาน

ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานมาทำการเชื่อมต่อเป็นวงจร โดยในหัวข้อนี้จะยกตัวอย่างการออกแบบวงจรเปรียบเทียบ โดยในวงจรจะประกอบไปด้วยสัญญาณข้อมูล 2 อินพุท สัญญาณควบคุม 3 อินพุท และสัญญาณผลเปรียบเทียบ 3 เอาท์พุท ดังรูปที่ 3-28



รูปที่ 3-28 วงจรเปรียบเทียบขนาด 1 บิท

เอาท์พุท A>B มีค่าเป็น '1' เมื่ออินพุท A มีค่ามากกว่า B(AB = 10) หรือถ้า A เท่ากับ B และอินพุท > มีค่าเป็น '1' เอาท์พุท A=B มีค่าเป็น '1' เมื่ออินพุท A เท่ากับ B และอินพุท = มีค่าเป็น '1' ส่วนเอาท์พุท A<B จะตรงข้ามกับเอาท์พุท A>B นั่นคือมีค่าเป็น '1' เมื่ออินพุท A มีค่าน้อยกว่า B(AB = 01) หรือถ้า A เท่ากับ B และอินพุท < มีค่าเป็น '1' ตัวอย่างการบรรยายการเชื่อมต่อของวงจรเปรียบเทียบขนาด 1 บิท ดังแสดงในรูปที่ 3-29 และ 3-30 เครื่องหมาย "- -" ใช้แทนคำอธิบาย (comment)

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

รูปที่ 3-29 การบรรยายการเชื่อมต่อ

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

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

รูปที่ 3-30 การบรรยายการทำงานภายในวงจรเปรียบเทียบขนาด 1 บิท

3.2.5 สรุปวีเอชดีแอล

ในบทนี้ได้กล่าวถึงโครงสร้างพื้นฐานและข้อกำหนดต่าง ๆ ของภาษาวีเอชดีแอลพอสังเขปเพื่อความเข้าใจในการศึกษาโครงสร้าง การออกแบบและการบรรยายของวงจรในบทต่อไป รูปแบบหลักในการบรรยายการทำงานของระบบซึ่งวีเอชดีแอลได้จัดเตรียมไว้แบ่งเป็น 3 ระดับคือ ระดับพฤติกรรม ระดับข้อมูล และระดับโครงสร้าง

3.3 การออกแบบวงจรด้วย FPGA

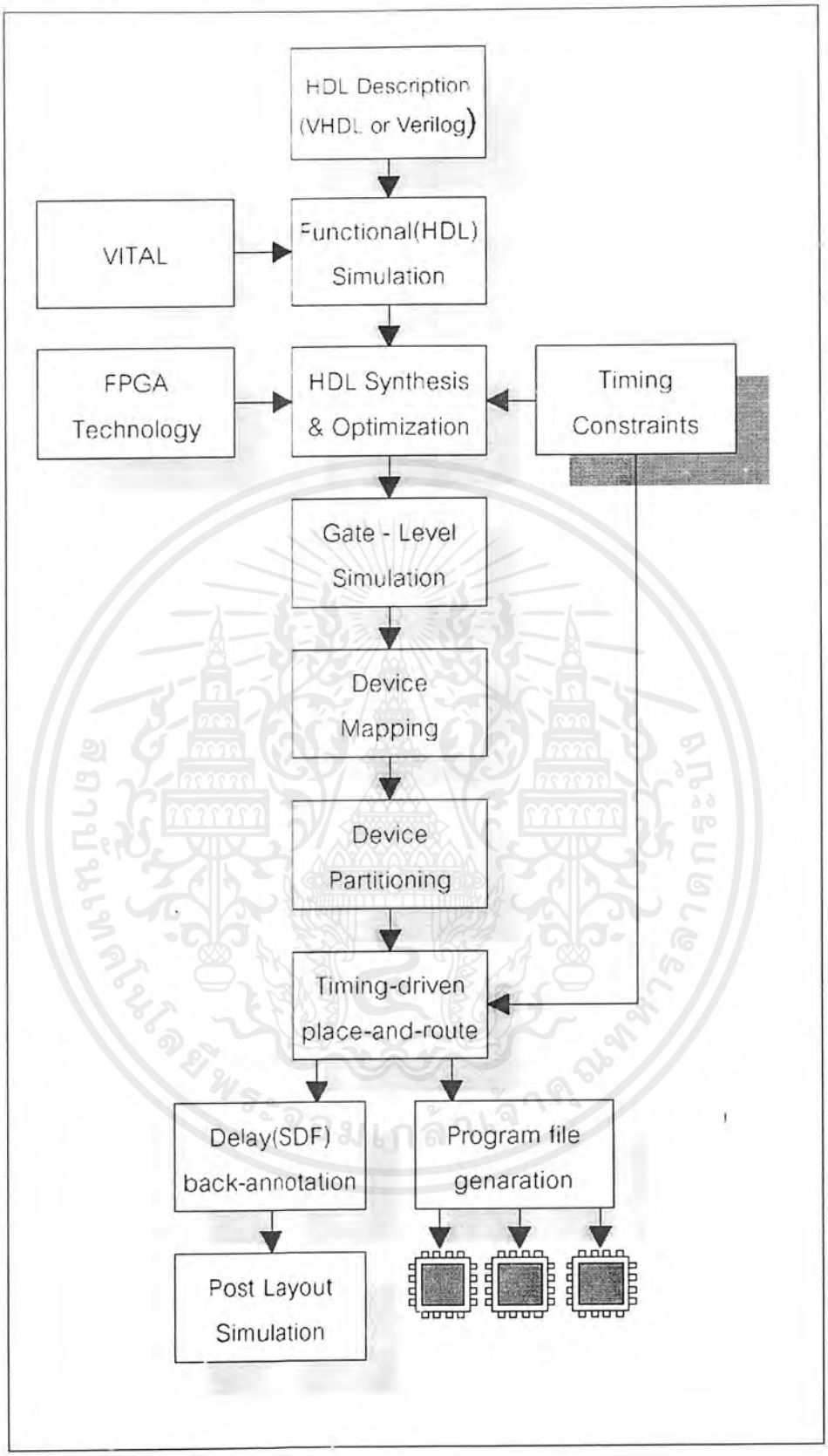
อุปกรณ์เอฟพีจีเอ (FPGA : Field Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์เอฟพีจีเอมีฟังก์ชันการทำงานตามที่ผู้ออกแบบไว้ต้องการ ในการทำเอฟพีจีเอเมื่อเทียบกับการทำการผลิตวงจรรวม (ASIC : Application Specific Integrated Circuit) แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือ การทำเอฟพีจีเอจะมีข้อจำกัดในด้านขนาดของวงจร เนื่องจากภายในอุปกรณ์เอฟพีจีเอจะมีจำนวนของเกท (Gate) ให้ใช้ในจำนวนที่จำกัด และการทำเอฟพีจีเอก็เหมาะกับการทำผลิตภัณฑ์ที่เป็นต้นแบบหรือเพื่อผลิตในปริมาณที่ต่ำ ส่วนข้อดีของการทำเอฟพีจีเอก็คือ ระยะเวลาที่ใช้ในการทำตั้งแต่ขั้นตอนการเขียนโค้ด อธิบายฮาร์ดแวร์ จนกระทั่งการดาวน์โหลด (Download) นั้นจะใช้เวลาที่น้อยกว่าการทำวงจรรวมมาก และการตรวจสอบหรือแก้ไข ออกแบบ ก็ทำได้สะดวก การทำเอฟพีจีเอในปัจจุบันนั้นมีประสิทธิภาพมากขึ้นและสะดวกขึ้น ทั้งนี้ก็เนื่องมาจากทางบริษัทผู้ผลิตอุปกรณ์ เอฟพีจีเอได้ทำการเพิ่มความสามารถของอุปกรณ์เอฟพีจีเอ โดยเพิ่มจำนวนขององค์ประกอบภายใน หรือปรับปรุงโครงสร้างทางสถาปัตยกรรมภายใน และยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ทำการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณ (PPR : Partitioning, Placement and Routing) สำหรับอุปกรณ์นั้น ๆ ด้วย

3.3.1 FPGA คืออะไร

FPGA (Field Programmable Gate Array) นั้นเป็นอุปกรณ์ที่สามารถโปรแกรมได้ โดยใช้สำหรับโปรแกรมวงจรที่ได้ออกแบบลงไป เพื่อให้อุปกรณ์เอฟพีจีเอมีฟังก์ชันการทำงานตามแบบที่ต้องการ ซึ่งเป็นวิธีในการออกแบบไอซี (IC : Integrated Circuit) แบบเซมิคัสตอม (Semicustom) อีกวิธีหนึ่ง สำหรับตัวอุปกรณ์เอฟพีจีเอเองนั้นก็มีโครงสร้างพื้นฐานเทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันไปสำหรับผู้ผลิตแต่ละราย ซึ่งนอกจากนั้นอุปกรณ์เอฟพีจีเอของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกัน

ในการใช้งานนั้นอุปกรณ์เอฟพีจีเอสามารถนำไปประยุกต์ใช้งานได้ เช่น การทำ DSP (Digital Signal Processing) หรือ ไมโครคอนโทรลเลอร์ เป็นต้น สำหรับการออกแบบวงจรด้วยเอฟพีจีเอก็จะคล้ายคลึงกับการออกแบบวงจรรวม ซึ่งมีขั้นตอนต่าง ๆ ดังรูปที่ 3-1

จากรูปจะเห็นว่า ในการออกแบบโดยใช้เอฟพีจีเอนั้นจะเริ่มต้นด้วยการใช้ภาษาในการอธิบายฮาร์ดแวร์เขียนโค้ดอธิบายฟังก์ชันการทำงานที่เราต้องการ ซึ่งจะเป็นไปในลักษณะการจำลองหน้าที่การทำงานต่าง ๆ หลังจากนั้นจะนำโค้ดที่ได้เขียนขึ้นมาทำการสังเคราะห์โค้ด (Synthesis) เพื่อให้ได้เป็นวงจรขึ้นมาโดยจะใช้ซอฟต์แวร์ในการสังเคราะห์ แต่ต้องตรวจดูว่าซอฟต์แวร์นั้น ๆ จะสนับสนุนเทคโนโลยีของเอฟพีจีเอที่เราต้องการใช้หรือไม่ ในขั้นตอนนี้ซอฟต์แวร์จะทำการแปลงโค้ดและทำการออพติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่ได้เลือกใช้ ซึ่งเราสามารถกำหนดข้อบังคับต่าง ๆ สำหรับโมเดลได้ เช่น ข้อบังคับด้านเวลา (Time Constraints) หลังจากนั้นจะเป็นการจำลองการทำงานในระดับเกทและการเทียบ (Mapping) โมเดลให้เข้ากับเทคโนโลยีที่ใช้ จากนั้นจะเป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ออกเป็นส่วนย่อย ๆ สำหรับลงในองค์ประกอบภายในอุปกรณ์เอฟพีจีเอ เมื่อเสร็จแล้วจะเป็นขั้น



รูปที่ 3-31 ขั้นตอนการออกแบบโดยใช้เอฟพีจีเอ

ตอนการวางอุปกรณ์และการค้นหาเส้นทางในการเชื่อมต่อสัญญาณของวงจรแต่ละส่วนเข้าด้วยกัน ซึ่งอาจมีการกำหนดข้อบังคับทางเวลามาช่วยด้วย และหลังจากผ่านขั้นตอนต่าง ๆ แล้วก็จะเป็นขั้นตอนของการตรวจสอบคุณสมบัติของโมเดลในรูปแบบค่าความหน่วงมาตรฐาน (SDF : Standard Delay Format) และทำการจำลองการทำงานของวงจรหลังจากการวางอุปกรณ์และเชื่อมต่อสัญญาณ (Post Layout Simulation)

3.3.2 การออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language)

ในการออกแบบวงจรดิจิตอลนั้น ทำได้ทั้งโดยการวาดวงจร (Schematic Drawing) หรือใช้ภาษาอธิบายฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วยเอฟพีจีเอและการทำวงจรรวม ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการการวาดวงจรจะต่างกันโดยที่วิธีนี้ผู้ทำการออกแบบต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นได้ว่าออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์นั้น ทำได้สะดวกกว่า เพราะการทำวิธีนี้ผู้ออกแบบไม่ต้องมาคำนึงถึงเทคโนโลยีที่จะใช้ และที่สำคัญการออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือ การออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นอยู่กับเทคโนโลยี สำหรับภาษาอธิบายฮาร์ดแวร์ที่ใช้กันมีภาษาวีเอชดีแอลหรือเวริล็อก (Verilog) ในการเขียนโค้ดนั้นสิ่งที่ต้องคำนึงถึงคือ จะต้องเขียนอย่างไรจึงจะสามารถสังเคราะห์ (Synthesis) เป็นวงจรได้ และให้โมเดลที่มีคุณสมบัติถูกต้องตามข้อกำหนด เพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับโมเดลที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้น ซอฟต์แวร์สังเคราะห์วงจร (Synthesis Tools) จะสังเคราะห์วงจรตามโค้ด เช่น ในการเขียนโค้ดวีเอชดีแอลอธิบายการทำงานของโมเดลวงจรอันเดียวกัน แต่เขียนโค้ดในลักษณะที่ต่างกัน เมื่อสังเคราะห์จะได้วงจรที่ต่างกัน และจากวงจรที่ต่างกัน เมื่อนำไปทำต้นแบบด้วยเอฟพีจีเอหรือการทำวงจรรวมแล้ว จะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านของขนาดหรือความเร็ว (Area and Time) ดังนั้นการออกแบบในขั้นนี้ผู้ออกแบบต้องระมัดระวังเป็นพิเศษส่วนการที่จะเขียนโค้ดในลักษณะใดให้ได้ผลลัพธ์ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ของผู้ออกแบบ

3.3.3 การสังเคราะห์วงจร (Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์สังเคราะห์วงจรทำการสังเคราะห์โค้ดวีเอชดีแอลหรือเวริล็อกเพื่อให้ได้เป็นวงจรขึ้นมา แต่ต้องตรวจสอบด้วยว่าซอฟต์แวร์นั้น ๆ สนับสนุนเทคโนโลยี (FPGA Library) ที่ต้องการใช้หรือไม่ ตัวอย่างเอฟพีจีเอที่มีการใช้งานแพร่หลายเช่น เอฟพีจีเอของบริษัท Xilinx และบริษัท Altera ในการทำเอฟพีจีเอนั้นมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Synopsys, Exemplar เป็นต้น ในขั้นตอนนี้ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ดวีเอชดีแอลและทำการออพติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ ในการสังเคราะห์วงจรเพื่อทำเอฟพีจีเอนั้น วงจรระดับเกท (Gate-Level) ไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์เอฟพีจีเอ ดังนั้นในการออพติไมซ์นั้น ซอฟต์แวร์สังเคราะห์วงจรจะต้องทำการออพติไมซ์ให้ได้เป็นวงจรที่ประกอบด้วยกลุ่มของลอจิก (Logic) ที่เหมาะสมกับอุปกรณ์เอฟพีจีเอนั้น ๆ จึงจะทำให้ผลลัพธ์ที่ได้มีประสิทธิภาพ และในขั้นตอนการสังเคราะห์วงจรนี้ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดลได้ เช่น ข้อบังคับในเรื่องของเวลา (Timing Constraints)

หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอนออพติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการออพติไมซ์คือ การเทียบ (Mapping) โมเดลให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์เอฟพีจีเอในกรณีของ Xilinx FPGA จะเทียบโดยใช้วิธี LUT (Look Up Table)

เมื่อทำการสังเคราะห์วงจรเสร็จแล้ว ซอฟต์แวร์สังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วง (Delay) เท่าไร ใช้ทรัพยากรต่าง ๆ ในเอฟพีจีเออะไรบ้าง เมื่อมาถึงขั้นตอนนี้ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็ให้ทำการสังเคราะห์ใหม่จนกว่าจะเป็นไปตามที่กำหนด ในขั้นตอนการสังเคราะห์นี้มีเทคนิคต่าง ๆ ที่ผู้ออกแบบนำมาใช้เพื่อให้โมเดลเป็นไปตามข้อบังคับที่กำหนด

3.3.4 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์เป็นส่วนย่อย ๆ สำหรับลงใน CLB (Configurable Logic Block), IOB (Input/Output Block) หรือองค์ประกอบอื่น ๆ ภายในอุปกรณ์เอฟพีจีเอ สำหรับเกณฑ์ที่ใช้ในการแบ่งคือ ให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (Routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำ โดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจร เช่น เกท ฟลิป-ฟลอบ (Flip-Folp) ลงในทรัพยากรต่าง ๆ ที่มีอยู่ภายในอุปกรณ์เอฟพีจีเอ

หลังจากทำขั้นตอนนี้เสร็จแล้ว ผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์เอฟพีจีเอไปเท่าไร ส่วนข้อมูลทางเวลานั้น ผู้ออกแบบจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่า ความหน่วงลอจิก (Logic Delay) ส่วนซอฟต์แวร์ที่ใช้ในขั้นตอนนี้ก็เช่น M1 ของ Xilinx ซึ่งซอฟต์แวร์ตัวนี้จะรวมเอาซอฟต์แวร์ย่อยอื่น ๆ อีกเพื่อให้การทำการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณเป็นไปอย่างต่อเนื่อง

3.3.5 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจรมาแล้วว่าควรจะอยู่ ณ ตำแหน่งไหนในอุปกรณ์เอฟพีจีเอเพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่น วงจรส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (Route) ได้ง่าย หรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์เอฟพีจีเอนั้นมีความสำคัญ เพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้น หรือตัวที่ทำการค้นหาเส้นทางของสัญญาณ (Router) ทำการค้นเส้นทางของสัญญาณได้อย่างไม่ครบถ้วน

การวางอุปกรณ์ที่ดีควรวางส่วนต่าง ๆ ให้อยู่ใกล้กันโดยเฉพาะส่วนที่มีการเชื่อมต่อสัญญาณด้วยกัน นอกจากนั้นการกำหนดตำแหน่งขาอินพุท/เอาท์พุท (I/O pin) ตามตำแหน่งขาอินพุท/เอาท์พุท ของเอฟพีจีเอบนแผ่น PCB ก็จะมีผลโดยตรงเลยคือ ซอฟต์แวร์จะวางอินพุท/เอาท์พุท ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่งบางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขาอินพุท/เอาท์พุท ควรกำหนดตำแหน่งให้เหมาะสม หรือไม่ก็ให้ซอฟต์แวร์จัดการเอง ในกรณีที่ใช้ซอฟต์แวร์ M1 ผู้ออกแบบสามารถแก้

ไขตำแหน่งใหม่ได้แต่ควรกระทำด้วยความระมัดระวังเป็นอย่างยิ่ง แต่วิธีที่ดีที่สุดก็คือ การให้ซอฟต์แวร์ทำซ้ำหลาย ๆ ครั้งเพื่อหาครั้งที่ดีที่สุด สำหรับเกณฑ์ที่ใช้ในการตัดสินคือ ความหน่วงที่ได้หลังจากทำการค้นหาเส้นทางแล้ว หรือที่เรียกว่า Routing Delay

3.3.6 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่าง ๆ ภายในอุปกรณ์เอฟพีจีเอ เช่น ระหว่าง CLB หรือระหว่าง CLB กับ IOB ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด หรือเกิดความหน่วงเกินค่าที่กำหนดในข้อบังคับ

ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟต์แวร์ เช่น M1 ของบริษัท Xilinx หรือผู้ออกแบบจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (Manual Layout) ก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า โดยให้ทำการค้นหาเส้นทางหลาย ๆ ครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา (Timing Constraints) จะช่วยให้ผลที่ได้จากการทำการเชื่อมต่อสัญญาณดีขึ้นได้

3.3.7 เวลาที่หน่วง (Delay Time)

ในการทำเอฟพีจีเอนั้น ความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (Layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วงที่เกิดขึ้นนั้นแยกได้เป็น 2 ประเภทคือ

1. ความหน่วงลอจิก (Logic Delay) : เป็นความหน่วงภายในองค์ประกอบของตัวอุปกรณ์เอฟพีจีเอเอง เช่น ความหน่วงภายใน CLB
2. ความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing Delay) : เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในของอุปกรณ์เอฟพีจีเอ

โดยปกติแล้วค่าความหน่วงลอจิกไม่ควรเกิน 50 % ของค่าความหน่วงที่ยอมรับได้ เพราะว่าความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลา เพื่อให้ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อให้ได้ผลลัพธ์ที่ดีขึ้น

ค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์และเชื่อมต่อสัญญาณแล้ว เป็นค่าความหน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้นเป็นไปตามข้อกำหนดหรือไม่

3.3.8 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่าง ๆ จนกระทั่งผ่านการทำการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณแล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลดลงในอุปกรณ์เอฟพีจีเอได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (Configuration Data) ซึ่งอยู่ในรูปของ บิทสตรีม (Bit-Stream) ก่อน แล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์เอฟพีจีเอมีฟังก์ชันการทำงานตามโมเดล

ที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับอุปกรณ์เอฟพีจีเอของแต่ละบริษัทผู้ผลิต คือในกรณีที่เป็นอุปกรณ์เอฟพีจีเอชนิดที่ต้องโปรแกรมโดยวิธี SRAM นั้น ในการใช้งานผู้ออกแบบต้องเก็บข้อมูลวงจรไว้ในหน่วยความจำประเภท EPROM หรือ Serial PROM ด้วยเพื่อจะได้ใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปก็ไม่ต้องดาวน์โหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีกเพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่แล้ว แต่ในกรณีที่อุปกรณ์เอฟพีจีเอเป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Antifuse ก็ไม่จำเป็นจ้องมีหน่วงความจำไว้สำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์เอฟพีจีเอชนิดนี้เมื่อดาวน์โหลดข้อมูลวงจรลงไปแล้ว ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์เอฟพีจีเอ และครั้งต่อไปก็ใช้งานโมเดลที่ออกแบบไว้ได้เลย

3.3.9 การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหน เพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น ModelSim ของบริษัท Model Technology ในการจำลองการทำงานของวงจรควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดลเกิดจากขั้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้น ๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร โดยต้องทำทั้งหลังจากเขียนโค้ด การสังเคราะห์วงจร และการทำการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณ การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออกแบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (Functional Test) แต่ยังไม่สามารถตรวจสอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้วเพื่อตรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องอยู่หรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตามข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ ถ้ามีก็จะได้แก้ไขให้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์และเชื่อมต่อสัญญาณ (Post Layout Simulation) แล้วก็มีความสำคัญเช่นกัน เพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้องตรวจสอบคุณสมบัติอื่น ๆ เช่น ความหน่วงที่ได้จากการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณ ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format :SDF) ว่าตรงตามที่กำหนดไว้หรือไม่ หรือตรวจสอบว่าแบบวงจรรวมสามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงานของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นต่าง ๆ

3.3.10 ซอฟต์แวร์ FPGA (FPGA Design Tools)

จะเห็นได้ว่าการออกแบบเพื่อทำเอฟพีจีเอนั้นทำได้สะดวกกว่าการทำวงจรรวมมาก เพราะใช้เวลาน้อยกว่ามากด้วย ส่วนสำคัญที่ใช้ในการทำเอฟพีจีเอคือ ซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงในอุปกรณ์เอฟพีจีเอ ซึ่งซอฟต์แวร์ที่ใช้ต้องเป็นซอฟต์แวร์ที่ใช้ทำงานต่อเนื่องกันได้

คือ ซอฟต์แวร์ที่ใช้ทำการสังเคราะห์วงจรกับซอฟต์แวร์ที่ใช้ทำการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณ เช่น Leonardo กับ M1 หรือ Synnopsys กับ M1 นั่นคือ M1 ต้องสามารถทำการแบ่งวงจร การวางอุปกรณ์ และการเชื่อมต่อสัญญาณวงจรที่สังเคราะห์โดยซอฟต์แวร์ Leonardo หรือ Synopsys ได้ สำหรับซอฟต์แวร์ที่ใช้ทำการจำลองการทำงานของวงจรนั้น ต้องสามารถใช้งานต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบ เพราะโมเดลที่ได้จากการทำขั้นตอนต่าง ๆ (ด้วยซอฟต์แวร์ต่าง ๆ) ต้องเอามาจำลองการทำงานได้ และในการจำลองการทำงานของวงจร ควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดทั้งระบบ เพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ที่ใช้งานบนเครื่องคอมพิวเตอร์ส่วนบุคคลมากขึ้น ซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

3.3.11 สิ่งที่ต้องคำนึงถึงเมื่อใช้ภาษาอธิบายฮาร์ดแวร์สำหรับออกแบบวงจร

ในช่วงที่ผ่านมาการออกแบบวงจรดิจิตอลสำหรับอุปกรณ์แบบเอฟพีจีเอจะเป็นแบบผสม (Hybrid Design Approach) คือจะประกอบด้วยซีเมติกที่ใช้สัญลักษณ์จากไลบรารี่ของผู้ผลิตเอฟพีจีเอเป็นส่วนใหญ่ รวมถึงการใช้โมดูลพิเศษที่อยู่ในรูปของแมโคร (Macro) เช่น XBLOX และ LPM ของ Xilinx เป็นต้น และอาจจะมีการใช้ภาษาระดับสูงร่วมด้วย อาทิเช่น วีเอชดีแอล, เวริล็อค หรือ เอเบล (ABEL) แต่ในปัจจุบันแนวโน้มของการออกแบบจะเน้นไปทางการใช้ภาษาอธิบายฮาร์ดแวร์มากขึ้น เพราะความจุและความสามารถของอุปกรณ์เอฟพีจีเอในยุคปัจจุบันได้เพิ่มมากขึ้น ซึ่งสามารถนำไปใช้ในการออกแบบวงจรดิจิตอลที่มีความซับซ้อนในระดับสูงได้ ดังนั้นเพื่อให้เป็นไปในทิศทางเดียวกันกับการออกแบบวงจรรวมที่ปกติใช้สำหรับวงจรที่มีขนาดใหญ่มาก จึงส่งผลให้การใช้ภาษาอธิบายฮาร์ดแวร์ในการอกกแบบวงจรสำหรับอุปกรณ์เอฟพีจีเอมีความสำคัญมากขึ้นตามลำดับ

แต่อย่างไรก็ตาม การใช้ภาษาอธิบายฮาร์ดแวร์เพียงอย่างเดียว (Completely HDL-based Design Approach) ไม่ว่าจะเป็นวีเอชดีแอลหรือเวริล็อค ก็ตาม ยังมีข้อจำกัดในการนำไปใช้งานจริงอยู่บ้าง ยกตัวอย่างเช่น การเรียกใช้องค์ประกอบ (Component Instantiation) โดยตรงจากไลบรารี่ของผู้ผลิตอุปกรณ์ เอฟพีจีเอซึ่งอาจจะเป็นองค์ประกอบพื้นฐานหรือองค์ประกอบรวม (คล้ายกับการใช้องค์ประกอบจากไลบรารี่ในรูปสัญลักษณ์ทางกราฟิกภายในซีมเมติก) เช่น โมดูลที่เป็นหน่วยความจำแบบแรม (RAM) เป็นต้น ถ้าได้มีการเรียกใช้องค์ประกอบลักษณะนี้ภายในโค้ดภาษาอธิบายฮาร์ดแวร์ องค์ประกอบเหล่านี้นั้นก็จะเป็นเสมือนกล่องดำหรือ Black Box คือมีแต่โครงภายนอกที่มีการกำหนดแค่พอร์ตต่าง ๆ ขององค์ประกอบหรือโมดูล แต่ไม่ทราบฟังก์ชันหรือโครงสร้างภายใน การทำเช่นนี้มีข้อดีคือ องค์ประกอบจากไลบรารี่เหล่านั้นได้ถูกออกแบบให้เหมาะสมกับเทคโนโลยีที่ได้เลือกใช้แล้วโดยผู้ผลิต นักออกแบบวงจรไม่จำเป็นต้องคำนึงถึงเรื่องการที่จะต้องมาออกแบบองค์ประกอบเหล่านั้นเองและอย่างไรบ้าง ซึ่งเป็นการช่วยลดระยะเวลาในการออกแบบ แต่ในขณะเดียวกันวิธีการดังกล่าวก็มีข้อเสีย ดังนี้คือ ดีไซน์จะขึ้นกับเทคโนโลยีเป้าหมายมากขึ้น เพราะได้มีการเลือกใช้องค์ประกอบจากไลบรารี่สำหรับเทคโนโลยีตัวใดตัวหนึ่ง และอีกประการหนึ่งก็คือ เมื่อต้องการจะทำการจำลองการทำงานของวงจรจะไม่สามารถทำได้อย่างสมบูรณ์ถ้าไม่มีไลบรารี่ที่มีองค์ประกอบเหล่านั้นสำหรับการทำการจำลองการทำงานโดยเฉพาะ จะมีก็

เพียงแต่ไลบรารี่สำหรับการสังเคราะห์วงจรเท่านั้น ในกรณีดังกล่าวการทำการจำลองการทำงานของวงจรในระดับพฤติกรรมของวงจรจึงต้องข้ามไป ซึ่งเป็นการเสี่ยงอย่างยิ่ง เพราะเราไม่อาจจะแน่ใจได้ว่าโมดูลที่ได้เขียนขึ้นจะทำงานได้ถูกต้องหรือไม่ และถ้าโมดูลไม่ถูกต้องตั้งแต่เริ่มแรกแล้ว การสังเคราะห์และการทำขั้นตอนอื่นต่อ ๆ ไปก็ไม่มีประโยชน์

ต่อมาภายหลังผู้ผลิตเอฟพีจีเอ ซึ่งเป็นผู้สร้างไลบรารี่สำหรับเทคโนโลยีของตนเองนั้นได้เริ่มหันมาสนใจกับปัญหานี้มากขึ้น อย่างเช่น Xilinx ก็ได้สร้างไลบรารี่สำหรับองค์ประกอบพื้นฐาน (SIMPRIM) เพื่อให้สามารถทำการจำลองการทำงานของวงจรในระดับพฤติกรรมหรือระดับเกตได้อย่างสมบูรณ์แบบ นอกจากนั้นก็ได้พยายามผลิตซอฟต์แวร์ตัวใหม่ออกมา เพื่อการออกแบบวงจรสำหรับอุปกรณ์ของตนเองโดยเฉพาะ โดยครอบคลุมแผนขั้นตอนการออกแบบเกือบทั้งหมด และนักออกแบบก็ไม่จำเป็นต้องใช้ซอฟต์แวร์ต่างผู้ผลิตเหมือนอย่างในอดีตที่ผ่านมา ซึ่งเมื่อหลายปีก่อน Xilinx จะทำซอฟต์แวร์เฉพาะใช้สำหรับขั้นตอนในลำดับท้ายตามแผนขั้นตอนการออกแบบเท่านั้น

อย่างไรก็ตาม การใช้ภาษาอธิบายฮาร์ดแวร์ในการออกแบบวงจรดิจิตอลสำหรับอุปกรณ์เอฟพีจีเอจะส่งผลให้ดีไซน์ที่ถูกสร้างขึ้นไม่ขึ้นอยู่กับเทคโนโลยีเป้าหมายมากเกินไป แต่ในขณะเดียวกันนักออกแบบก็ต้องคำนึงถึงการใช้งานอุปกรณ์ (Device Utilization) ให้ได้ประสิทธิภาพมากที่สุด ดังนั้นในระหว่างการออกแบบวงจรก็มีความจำเป็นต้องคำนึงถึงสถาปัตกรรมภายในและเทคโนโลยีของอุปกรณ์เอฟพีจีเอที่เลือกใช้ด้วยเพื่อที่จะได้ใช้อุปกรณ์เอฟพีจีเออย่างถูกต้องและมีประสิทธิภาพ ซึ่งมีผลโดยตรงต่อประสิทธิภาพในการทำงานของวงจรที่นำไปใช้ด้วย

3.3.12 อุปกรณ์เอฟพีจีเอตระกูล XC4000

สำหรับในโครงงานนี้นั้นเราได้นำอุปกรณ์เอฟพีจีเอตระกูล XC4000 มาใช้ในส่วนของการเข้ารหัสและถอดรหัสข้อมูล โดยที่ชิปเอฟพีจีเอตระกูล XC4000 นี้เป็นชิปที่มีความสามารถและความจุในการโปรแกรมที่สูง ในขณะที่ยังช่วยหลีกเลี่ยงค่าใช้จ่ายในการพัฒนาและดูแลรักษาอีกด้วย สำหรับชิปเอฟพีจีเอเบอร์ต่าง ๆ ในตระกูล XC4000 นั้นเราได้แสดงไว้ในตารางที่ 3-8 และเบอร์ของชิปเอฟพีจีเอตระกูล XC4000 ที่เราใช้ในโครงงานนี้นั้น เราได้ใช้ชิปเบอร์ XC4010E ซึ่งมีจำนวนของเกทที่สามารถใช้ได้ทั้งสิ้น 10000 เกท ดังแสดงไว้ในรูปที่ 3-32

รูปที่ 3-32 ชิปเอฟพีจีเอเบอร์ XC4010E

Device	Logic Cells	Max Logic Gates (No RAM)	Max. RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total CLBs	Number of Flip-Flops	Max. User I/O
XC4002XL	152	1,600	2,048	1,000 - 3,000	8 x 8	64	256	64
XC4003E	238	3,000	3,200	2,000 - 5,000	10 x 10	100	360	80
XC4005E/XL	466	5,000	6,272	3,000 - 9,000	14 x 14	196	616	112
XC4006E	608	6,000	8,192	4,000 - 12,000	16 x 16	256	768	128
XC4008E	770	8,000	10,368	6,000 - 15,000	18 x 18	324	936	144
XC4010E/XL	950	10,000	12,800	7,000 - 20,000	20 x 20	400	1,120	160
XC4013E/XL	1368	13,000	18,432	10,000 - 30,000	24 x 24	576	1,536	192
XC4020E/XL	1862	20,000	25,088	13,000 - 40,000	28 x 28	784	2,016	224
XC4025E	2432	25,000	32,768	15,000 - 45,000	32 x 32	1,024	2,560	256
XC4028EX/XL	2432	28,000	32,768	18,000 - 50,000	32 x 32	1,024	2,560	256
XC4036EX/XL	3078	36,000	41,472	22,000 - 65,000	36 x 36	1,296	3,168	288
XC4044XL	3800	44,000	51,200	27,000 - 80,000	40 x 40	1,600	3,840	320
XC4052XL	4598	52,000	61,952	33,000 - 100,000	44 x 44	1,936	4,576	352
XC4062XL	5472	62,000	73,728	40,000 - 130,000	48 x 48	2,304	5,376	384
XC4085XL	7448	85,000	100,352	55,000 - 180,000	56 x 56	3,136	7,168	448

ตารางที่ 3-8 รายละเอียดของชิปเอฟพีจีเอตระกูล XC4000

ในส่วนของคุณสมบัติโดยทั่วไปของเอฟพีจีเอตระกูล XC4000 นั้นจะขอแสดงโดยแยกไว้เป็นหัวข้อได้ดังต่อไปนี้

- เป็นอุปกรณ์ที่มีฟลิปฟลอป (Flip Flop) จำนวนมาก
- มีความยืดหยุ่นสูงในการผลิตฟังก์ชันการทำงาน
- มีค่าแฟนเอาต์ (Fan-Out) สูง
- มีบัสภายใน 3 สถานะ
- ทำงานกับสัญญาณ TTL และ CMOS
- มีหน่วยความจำ RAM ภายในที่มีความเร็วสูง (< 25 นาโนวินาที)
- เส้นทางการเชื่อมต่อ (Interconnect Line) เป็นแบบลำดับชั้น
- มีการกระจายกำลังงานของสัญญาณที่ต่ำ
- มีสถาปัตยกรรมของลอจิกบล็อค (Logic Block) และไอโอบล็อค (I/O Block) ที่สามารถโปรแกรมได้
- รองรับมาตรฐาน IEEE 1149.1 ในการทำ Boundary-Scan Logic
- สามารถโปรแกรมค่า Output Slew Rate ได้
- สามารถโปรแกรมให้อินพุทมีลักษณะพูลอัพ (Pull-Up) หรือพูลดาวน์ (Pull-Down) รีจิสเตอร์ได้
- ให้กระแสเอาท์พุทได้ตั้งแต่ 12-24 มิลลิแอมป์
- ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
- มีโหมดในการโปรแกรมให้เลือก 6 โหมด
- มีโปรแกรมการวางและเชื่อมโยงอุปกรณ์ภายในแบบอัตโนมัติ (Automatic Place and Routing) ที่ครบสมบูรณ์

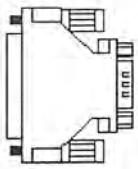
3.4 การเชื่อมต่อแบบอนุกรมผ่านพอร์ตอนุกรม (Serial / RS-232 Port)

การสื่อสารข้อมูลอนุกรมเป็นการรับหรือส่งข้อมูลในลักษณะของบิทหรือกลุ่มของบิทคราวละ 1 บิทเป็นลำดับเรื่อยไปจนสิ้นสุด การสื่อสารแบบนี้จะมีข้อแตกต่างจากการสื่อสารแบบขนานเป็นอย่างมาก เนื่องจากข้อมูลมีการโอนย้ายมาพร้อมกันจึงมีความจำเป็นต้องใช้จำนวนเส้นสัญญาณมากขึ้นตามจำนวนบิทของข้อมูลด้วย ในขณะที่การสื่อสารแบบอนุกรมนั้นต้องการเส้นสัญญาณเพียง 2 หรือ 3 เส้นเท่านั้น ดังนั้นการสื่อสารแบบขนานจึงไม่เหมาะสมในการสื่อสารกับอุปกรณ์ภายนอกเป็นระยะทางไกล ๆ เพราะจะทำให้สิ้นเปลืองค่าใช้จ่ายมาก

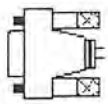
ในส่วนของการเชื่อมต่อแบบอนุกรมผ่านทางพอร์ตอนุกรม (Serial Port) นั้นจะมีความยุ่งยากมากกว่าการติดต่อผ่านพอร์ตขนาน (Parallel Port) โดยในส่วนใหญ่แล้ว อุปกรณ์ต่าง ๆ ที่จะทำการเชื่อมต่อผ่านทางพอร์ตอนุกรมนั้นมีความจำเป็นที่จะต้องแปลงการส่งข้อมูลแบบอนุกรมให้เป็นแบบขนานก่อนจึงจะสามารถนำไปใช้ได้ ซึ่งจะกระทำโดยผ่านมาตรฐาน UART ในส่วนของทางด้านซอฟต์แวร์นั้นมาตรฐานการส่งข้อมูลแบบอนุกรมจะมีรีจิสเตอร์ (Register) ในการใช้งานอยู่มากกว่ามาตรฐานของพอร์ตขนาน (SPP : Standard Parallel Port) และต่อไปนี้เป็นข้อดีของการส่งข้อมูลแบบอนุกรมเมื่อเทียบกับแบบขนาน

- สายเคเบิลที่ใช้ในการส่งข้อมูลแบบอนุกรมนั้นจะสามารถใช้ได้ยาวกว่าการส่งข้อมูลแบบขนาน กล่าวคือ ในการส่งข้อมูลแบบอนุกรมจะส่งสัญญาณ –3 ถึง –25 โวลต์แทนข้อมูล '1' และจะส่งสัญญาณ +3 ถึง +25 โวลต์แทนข้อมูล '0' แต่ในการส่งข้อมูลแบบขนานนั้นจะส่งสัญญาณ 5 โวลต์แทนข้อมูล '1' และสัญญาณ 0 โวลต์แทนข้อมูล '0' ซึ่งจะเห็นได้ว่าพอร์ตอนุกรมนั้นจะมีช่วงของการส่งข้อมูลสูงสุด 50 โวลต์ แต่พอร์ตขนานจะมีช่วงของการส่งข้อมูลได้สูงสุดแค่ 5 โวลต์เท่านั้น นั่นทำให้การผิดเพี้ยนของสัญญาณที่ส่งผ่านสายเคเบิลในการส่งข้อมูลแบบอนุกรมนั้นจะเป็นปัญหาน้อยกว่าการส่งข้อมูลแบบขนาน
- การส่งข้อมูลแบบอนุกรมไม่มีความจำเป็นที่จะต้องใช้สายสัญญาณมากเท่ากับการส่งข้อมูลแบบขนานเมื่อต้องการเชื่อมต่อกับคอมพิวเตอร์ในระยะทางไกล ๆ
- ในปัจจุบันมีการนำไมโครโปรเซสเซอร์มาใช้งานกันอย่างแพร่หลายและเป็นที่นิยมมาก ซึ่งในส่วนของตัวไมโครโปรเซสเซอร์เหล่านั้นจะมี SCI (Serial Communications Interfaces) เพื่อใช้ในการติดต่อสื่อสารกับส่วนของภายนอก

สำหรับพอร์ตอนุกรมนั้นจะมีอยู่ 2 ประเภทด้วยกันคือ ประเภทคอนเนคเตอร์ D-Type 25 ขา (D25) และประเภทคอนเนคเตอร์ D-Type 9 ขา (D9) ดังแสดงในรูปที่ 3-32 โดยทั้ง 2 ประเภทนั้นจะเป็นลักษณะของคอนเนคเตอร์ตัวผู้อยู่ทางด้านหลังของคอมพิวเตอร์ ดังนั้นในการเชื่อมต่ออุปกรณ์เราต้องนำคอนเนคเตอร์ตัวเมียมาทำการเชื่อมต่อ ซึ่งขาต่าง ๆ ของพอร์ตอนุกรมทั้ง 2 ประเภทนั้นเราได้แสดงรายละเอียดไว้ในตารางที่ 3-8



D-Type 25 Pin Connector



D-Type 9 Pin Connector

รูปที่ 3-33พอร์ตอนุกรมประเภท D25 และ D9

D-Type 25 Pin No.	D-Type 9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request to Send
Pin 5	Pin 8	CTS	Clear to Send
Pin 6	Pin 6	DSR	Data Set Raeady
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

ตารางที่ 3-9 ขาต่าง ๆ ของพอร์ตอนุกรมประเภท D25 และ D9

3.4.1 รูปแบบของข้อมูลอนุกรม

วิธีการที่จะทำให้ข้อมูลสื่อสารอนุกรมมีความถูกต้องมากยิ่งขึ้น จะใช้การเพิ่มเติมบิทข้อมูลบางอย่างร่วมไปกับการส่งข้อมูลจริง ได้แก่

3.4.1.1 บิทเริ่มต้น (Start Bit)

บิทเริ่มต้นมีหน้าที่สำหรับการบ่งบอกให้วงจรฮาร์ดแวร์ทางด้านรับทราบถึงตำแหน่งจุดเริ่มต้นของบิทข้อมูลกลุ่มใหม่ เพื่อที่จะทำการปรับจังหวะของสัญญาณการรับข้อมูลให้ตรงกัน ดังนั้นบิทเริ่มต้นนี้จึงจะถูกเพิ่มเข้าไปก่อนที่จะมีการส่งข้อมูลจริง ตามปกติแล้วค่าของบิทเริ่มต้นมักจะเป็นระดับสถานะทางลอจิกที่ตรงข้ามกับระดับสถานะลอจิกของสภาวะของสายสื่อสารขณะเมื่อไม่มีการส่งข้มูล (Idle State) เช่น หากสภาวะของสายสัญญาณเมื่อไม่มีข้อมูลเป็นระดับสูง (High) บิทเริ่มต้นก็จะเป็นระดับต่ำ (Low)

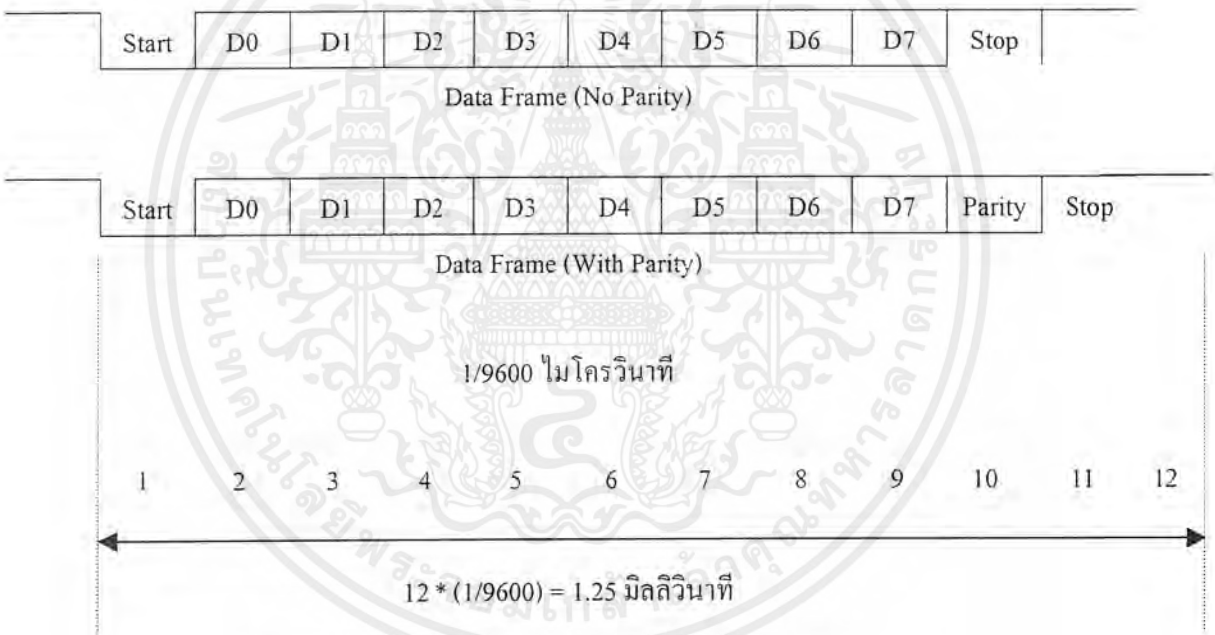
3.4.1.2 บิทพาริตี้ (Parity Bit)

สำหรับบิทนี้จะมีหน้าที่เพื่อการตรวจสอบความถูกต้องของข้อมูล โดยทั่วไปจะนำไปแทรกต่อท้ายบิทข้อมูล ค่าของบิทนี้ขึ้นอยู่กับจำนวนค่าของบิทข้อมูลที่เป็น 1 ซึ่งจะเป็นได้ใน 2 ลักษณะคือ พาริตี้คู่ (Even Parity) หรือพาริตี้คี่ (Odd Parity)

3.4.1.3 บิทสุดท้าย (Stop Bit)

บิทสุดท้ายเป็นบิทที่เพิ่มเติมขึ้นเพื่อระบุถึงขอบเขตการสิ้นสุดของกลุ่มบิทข้อมูล บิทสุดท้ายนี้อาจจะมีจำนวนมากกว่า 1 บิทก็ได้ คือ 1 บิท 1½ บิท และ 2 บิท

ดังนั้นในกรณีของการส่งข้อมูล 8 บิทพร้อมบิทที่เพิ่มเติมเข้าไปโดยสมบูรณ์ คือ บิทเริ่มต้น บิทพาริตี้ และบิทสุดท้าย (2 บิท) รวมทั้งสิ้น 12 บิท ตามแผนภาพสัญญาณเวลาในรูปที่ 3-32 หากข้อมูลถูกส่งออกไปด้วยอัตราเร็ว 9600 บอด (Baud) เพราะฉะนั้นเวลาโดยรวมในการส่งข้อมูล 1 ไบท์จะมีค่าเป็น 12* (1/9600) ไมโครวินาที หรือ 1.25 มิลลิวินาที



รูปที่ 3-34 เฟรมข้อมูลอนุกรม 8 บิทพร้อมด้วยบิทต่าง ๆ ที่เพิ่มเข้าไป

3.4.2 การส่งข้อมูล

ในการส่งข้อมูลแบบอนุกรมนั้นจะกระทำโดยการกระตุ้นการส่งผ่านทางบิท Transmitter Enable (TE) ในรีจิสเตอร์ควบคุม UART โดยเมื่อพร้อมที่จะส่งข้อมูลแล้วนั้น ข้อมูลจะถูกส่งจาก Transmitter Holding Register ไปที่ Transmitter Shift Register และถูกแปลงให้เป็นชุดข้อมูลแบบอนุกรมที่ขา Transmitter Serial Output Pin (TXD) ซึ่งมันจะส่งบิทเริ่มต้นและตามด้วยบิทข้อมูลอีก 8 บิทโดยอัตโนมัติ ซึ่งอาจจะมีบิทพาริตี้ (Parity Bit) เพื่อคอยตรวจสอบความถูกต้องของการส่งข้อมูลก็ได้ ดังรูปที่ 3-34 โดยบิทที่มีความสำคัญน้อยที่สุด (Least Significant Bit) ของข้อมูลนั้นจะถูกส่งไปเป็นอันดับแรก

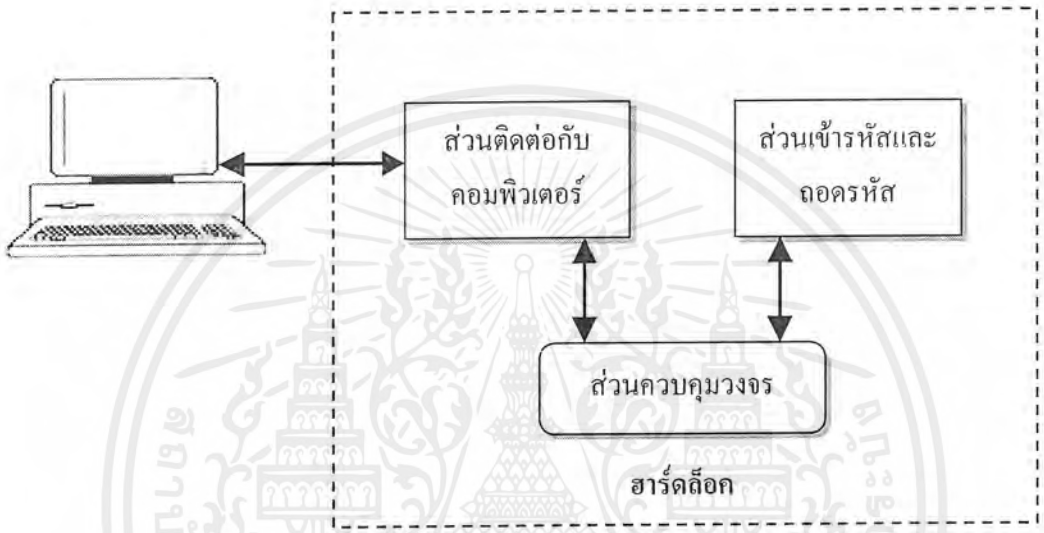
3.4.3 การรับข้อมูล

สำหรับการรับข้อมูลแบบอนุกรมนั้นจะกระทำโดยการกระตุ้นการส่งผ่านทางบิท Receiver Enable (RE) ในรีจิสเตอร์ควบคุม UART โดยตัวที่ทำการรับนั้นจะมองหาการเปลี่ยนแปลงของสัญญาณจากสูงไปเป็นต่ำของบิทเริ่มต้นที่ขา Receiver Serial Data Input ถ้าพบการเปลี่ยนแปลงสถานะของข้อมูลเข้าแบบอนุกรมจะถูกทำการสุ่มที่ตรงกลางของบิทต่อมา ถ้าข้อมูลเข้านั้นที่ถูกสุ่มมีสถานะเป็นสูงแล้วแสดงว่าบิทเริ่มต้นนั้นไม่ถูกต้องและตัวที่ทำการรับก็จะคอยมองหาบิทเริ่มต้นที่ถูกต้องต่อไป แต่ถ้าข้อมูลเข้าที่ถูกสุ่มนั้นยังคงมีสถานะเป็นต่ำแล้วแสดงว่าบิทเริ่มต้นนั้นเป็นบิทที่ถูกต้อง และตัวรับข้อมูลจะทำการสุ่มรับข้อมูลในคราวละ 1 บิท จนกระทั่งได้รวบรวมจำนวนบิทของข้อมูลและบิทพาริตี้ที่เหมาะสม และบิทสิ้นสุด (Stop Bit) ถูกตรวจพบ สำหรับข้อมูลเข้านั้นจะถูกสุ่มเป็นจำนวน 3 ครั้งด้วยกันในแต่ละบิทเพื่อที่จะกรองสัญญาณรบกวนออก

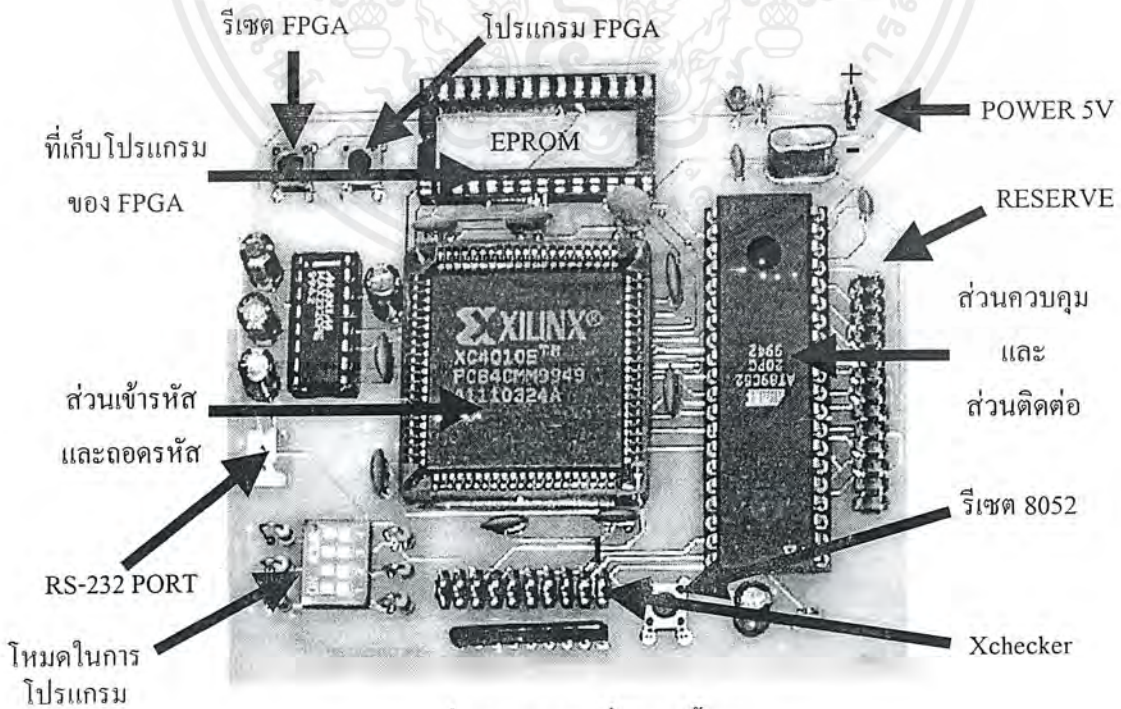
บทที่ 4
การออกแบบวงจรฮาร์ดล็อคของโครงงานนี้

4.1 วงจรฮาร์ดล็อค

วงจรฮาร์ดล็อคจะประกอบไปด้วยส่วนประกอบที่สำคัญ 3 ส่วนคือ 1.ส่วนที่ทำการเข้ารหัสและถอดรหัส 2.ส่วนในการควบคุมวงจร 3.ส่วนในการติดต่อกับคอมพิวเตอร์ โดยส่วนประกอบทั้งสามนี้จะเชื่อมต่อกันดังรูปที่ 4-1 (a) และรูปที่ 4-1 (b) เป็นวงจรของจริงที่ใช้

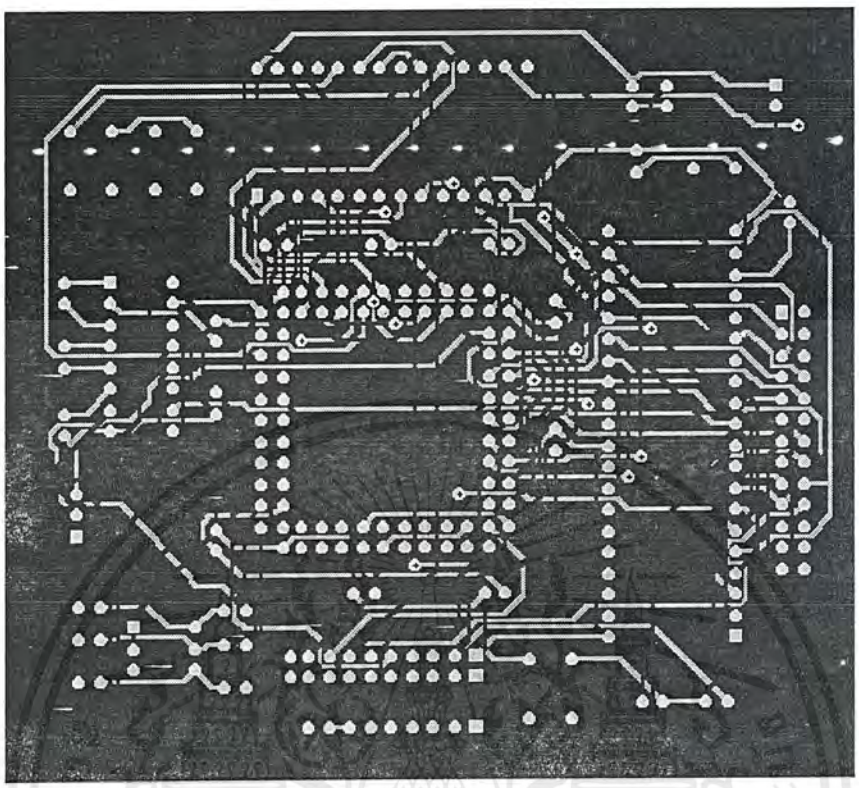


รูปที่ 4-1 (a) การเชื่อมต่อส่วนต่าง ๆ ของวงจรฮาร์ดล็อค



รูปที่ 4-1 (b) วงจรที่พัฒนาขึ้นมา

ลายวงจรของรูปที่ 4-1 (b) แสดงในรูปที่ 4-1 (c)



รูปที่ 4-1 (c) ลายวงจรของฮาร์ดล็อค

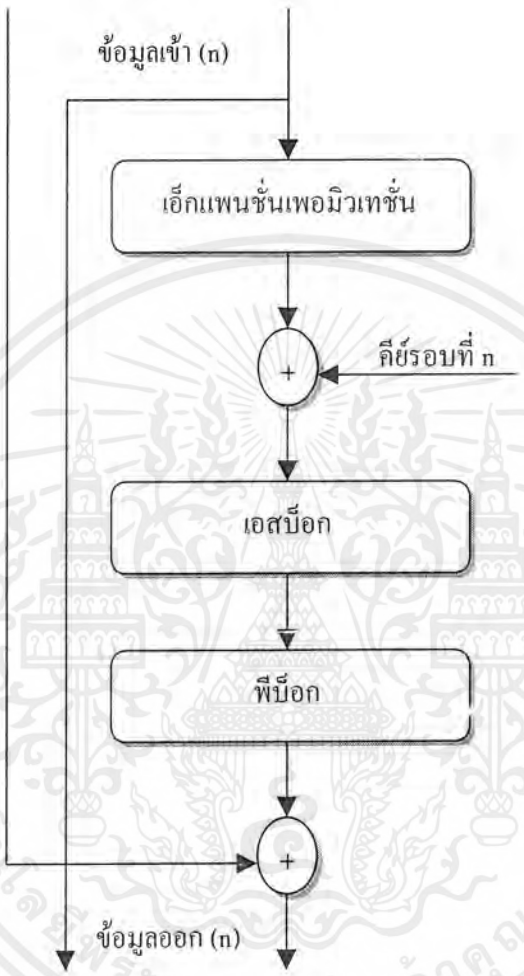
สำหรับฮาร์ดแวร์ที่เลือกใช้ในการทำเป็นฮาร์ดล็อคนี้จะใช้ชิปตระกูลเอฟพีจีเอ (FPGA) และใช้ภาษาวีเอชดีแอล (VHDL) ในการอธิบายรูปแบบการทำงานของวงจร สาเหตุที่เลือกใช้ชิปตระกูลเอฟพีจีเอ เนื่องจากชิปตระกูลนี้ถูกออกแบบมาให้ใช้เพื่อการพัฒนาต้นแบบวงจรรวมและมีความจุของเกทและจำนวนอินพุทเอาท์พุทให้เลือกใช้ในขนาดต่าง ๆ กันมากมาย นอกจากนี้ยังสามารถที่จะโปรแกรมให้เป็นวงจรตามที่เราต้องการได้หลายครั้ง และสามารถใช้ภาษาวีเอชดีแอลในการอธิบายการทำงานของวงจรได้

4.2 ส่วนการเข้ารหัสและถอดรหัส

ในส่วนนี้มีหน้าที่ในการรับข้อมูลมาทำการเข้ารหัสและถอดรหัส โดยจะใช้วิธีการเข้ารหัสและถอดรหัสแบบดีอีเอส (DES) โดยสามารถที่จะเข้ารหัสข้อมูลได้ครั้งละ 64 บิท โดยจะมีอินพุทเป็นข้อมูลขนาด 64 บิทและคีย์ขนาด 64 บิท ส่วนเอาท์พุทจะเป็นข้อมูลขนาด 64 บิทที่ผ่านขบวนการเข้ารหัสแล้ว สำหรับการอธิบายโดยใช้ภาษาวีเอชดีแอลนั้นเราจะต้องแบ่งวงจรออกเป็นส่วนย่อย ๆ (Module) ก่อนเพื่อความสะดวกในการที่จะพัฒนาและทดสอบ

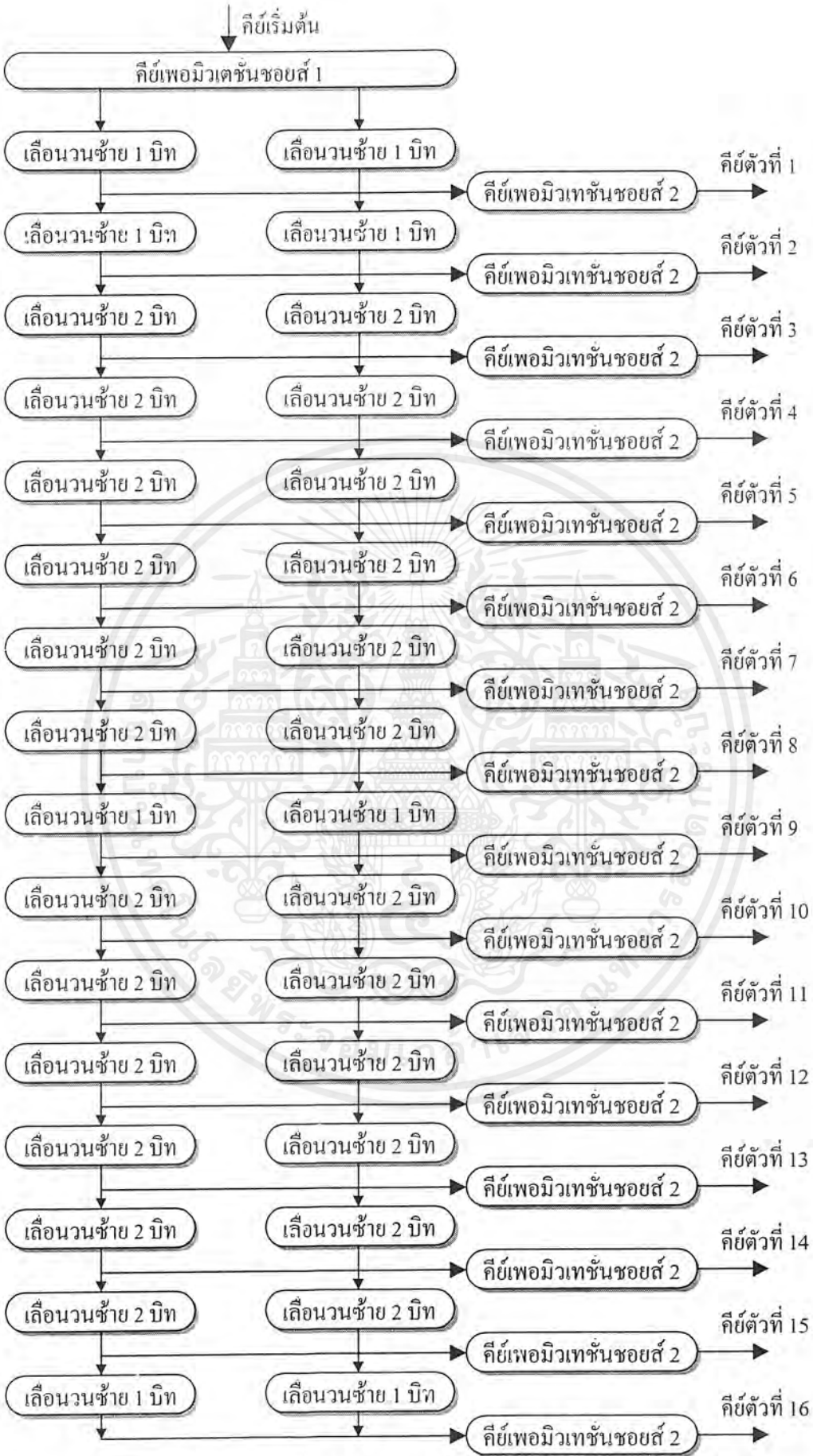
4.2.1 การแบ่งวงจรออกเป็นส่วนย่อย ๆ

ในขั้นแรกเราจะต้องทำการมองวิธีการเข้ารหัสแบบดีอีเอสเป็นการไหลของข้อมูลผ่านขบวนการต่างก่อนซึ่งจะสังเกตุได้ว่าจะมีขั้นตอนที่ต้องทำซ้ำเหมือน ๆ กันทั้งหมด 16 ครั้งเพื่อความประหยัดในทรัพยากรเราสามารถที่จะออกแบบโดยให้มีการทำซ้ำในส่วนที่เหมือนกันจำนวน 16 ครั้งได้ดังแสดงในรูปที่ 4-2



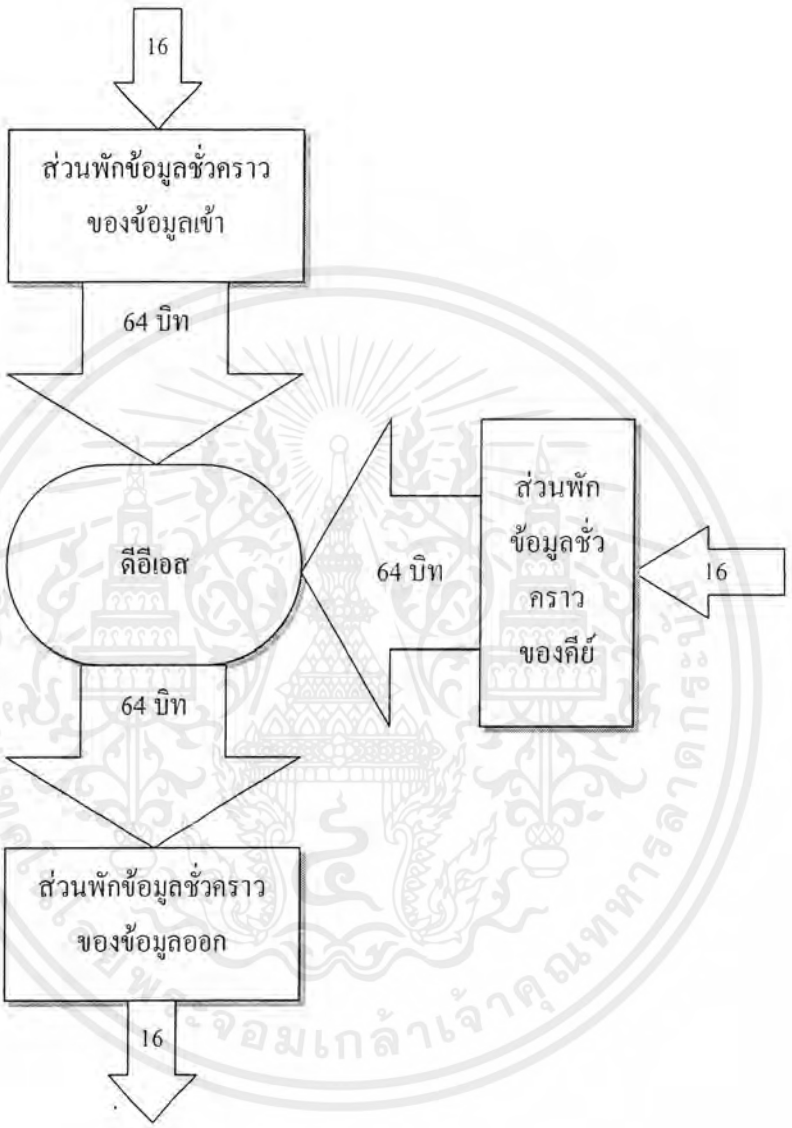
รูปที่ 4-2 ส่วนที่ทำซ้ำกันจำนวน 16 ครั้งของการเข้ารหัสแบบดีอีเอส

ส่วนคีย์ในแต่ละรอบของการเข้ารหัสก็จะใช้คีย์ที่แตกต่างกันซึ่งส่วนจัดการคีย์จะทำหน้าที่เตรียมคีย์ไว้ 16 ตัวสำหรับการเข้ารหัสในแต่ละรอบโดยใช้ข้อมูลเริ่มต้นจากคีย์ที่ป้อนเข้ามา เมื่อเราได้คีย์เริ่มต้นเข้ามาแล้วเราก็จัดการแปลงให้เป็นคีย์สำหรับในแต่ละรอบเลยและเก็บไว้ในที่พักข้อมูลชั่วคราว และเมื่อนำไปต่อกับส่วนอื่นก็จะนำข้อมูลจากส่วนที่พักข้อมูลชั่วคราวออกไปตามลำดับของคีย์ โดยถ้าเป็นการเข้ารหัสก็จะนำคีย์ตัวที่ 1 ออกไปก่อนจนถึงคีย์ตัวที่ 16 ส่วนถ้าเป็นการถอดรหัสก็จะนำคีย์ตัวที่ 16 ออกไปก่อนจนถึงคีย์ตัวที่ 1 วิธีการแปลงให้เป็นคีย์สำหรับในแต่ละรอบนั้นมีลักษณะดังรูปที่ 4-3



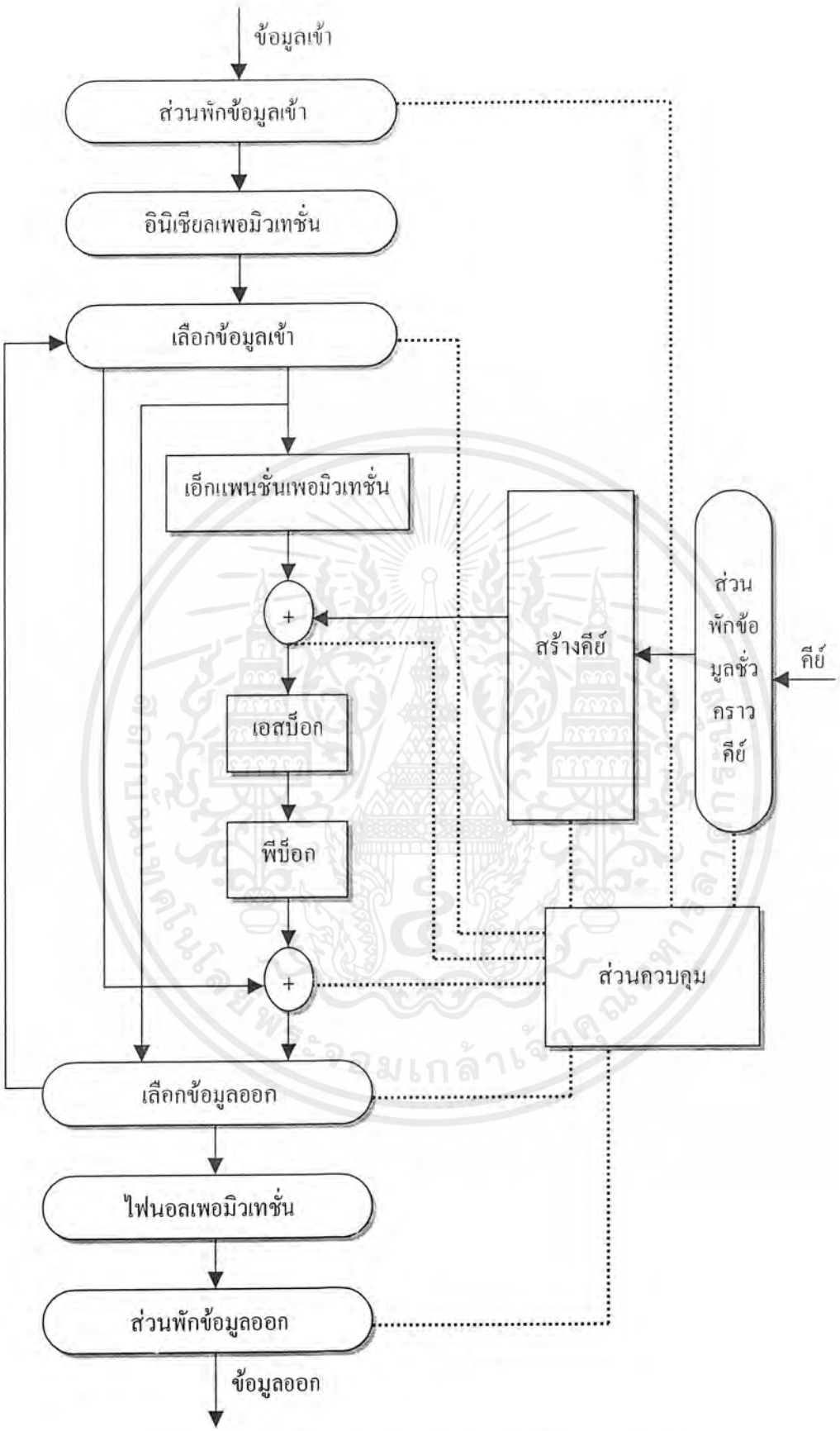
รูปที่ 4-3 การคำนวณคีย์ในแต่ละรอบ

เมื่อมองภาพรวมของส่วนที่ทำหน้าที่เข้ารหัสและถอดรหัสแล้วจะเห็นได้ว่าจะมีส่วนที่เป็นอินพุตและเอาท์พุตจำนวน 64 + 64 + 64 = 192 บิทซึ่งจะเป็นจำนวนสัญญาณที่ค่อนข้างมากเราสามารถที่จะลดขาสัญญาณนี้ลงได้โดยการใช้ส่วนพักข้อมูลชั่วคราว (buffer) เข้ามาช่วยโดยจากการที่จะต้องนำข้อมูลเข้าและออกทีละ 64 บิทเราก็จะนำข้อมูลเข้าและออกทีละ 16 บิทเป็นจำนวน 4 ครั้งด้วยกันดังรูป 4-4



รูปที่ 4-4 การทำงานเมื่อนำส่วนพักข้อมูลชั่วคราวเข้ามาเชื่อมต่อ

เมื่อเราสามารถที่จะแปลงอัลกอลิทึม (algorithm) ของดีอีเอสให้สามารถสร้างเป็นวงจรได้แล้วเราก็สามารถที่นำส่วนต่าง ๆ มาเชื่อมกันเพื่อจะได้วงจรที่สมบูรณ์ ดังรูปที่ 4-5



รูปที่ 4-5 ส่วนประกอบทั้งหมดของการเข้ารหัสและถอดรหัสดีอีเอส

จากรูปข้อมูลจะถูกรับเข้ามาทีละ 16 บิท จำนวน 4 รอบเพื่อให้ได้ข้อมูลเข้าขนาด 64 บิทก่อนที่จะส่งต่อไปให้อินิเชียลเพอมิวเทชั่นเพื่อสลับตำแหน่งข้อมูล หลังจากนั้นก็จะเข้าสู่ส่วนเลือกอินพุทข้อมูล ถ้าเป็นข้อมูลรอบแรกก็จะรับข้อมูลจากอินิเชียลเพอมิวเทชั่นถ้าเป็นรอบอื่นก็จะรับข้อมูลจากส่วนเลือกข้อมูลออก หลังจากผ่านส่วนเลือกข้อมูลเข้ามาแล้วข้อมูลบิทที่ 31-0 จะถูกส่งไปยังเอกแพนชั่นเพอมิวเทชั่นเพื่อเพิ่มขนาดข้อมูลจาก 32 บิทให้เป็น 48 บิทเพื่อจะได้มีขนาดของข้อมูลเท่ากับขนาดของคีย์ จากนั้นจะนำไปเอ็กคลูซีฟออกับคีย์ ข้อมูลผลลัพธ์ที่ได้จะส่งไปยังเอสบ็อกเพื่อทำการเปรียบเทียบค่ากับตารางทั้ง 8 เมื่อผ่านเอสบ็อกออกมาแล้วข้อมูลจะมีขนาด 32 บิท จากนั้นข้อมูลก็จะผ่านพีบ็อกเพื่อสลับตำแหน่งข้อมูล เมื่อผ่านพีบ็อกแล้วข้อมูลจะถูกนำไปเอ็กคลูซีฟออกับข้อมูลอีกส่วนหนึ่งซึ่งเป็นบิทที่ 63-32 ของส่วนเลือกข้อมูลเข้า ผลลัพธ์ก็จะเข้าไปยังส่วนเลือกข้อมูลออกบิทที่ 31-0 ส่วนเลือกข้อมูลออกบิทที่ 63-32 จะรับมาจากส่วนรับข้อมูลเข้าบิทที่ 31-0 สำหรับการทำงานถ้าเป็นการทำงานในรอบที่ 1-15 ข้อมูลที่อยู่ในส่วนเลือกข้อมูลออกจะถูกส่งกลับไปที่ส่วนเลือกข้อมูลเข้าเพื่อทำกระบวนการที่กล่าวมาแล้วใหม่อีกรอบ ถ้าเป็นการทำงานรอบที่ 16 ข้อมูลจะผ่านมายังส่วนไฟนอลเพอมิวเทชั่นเพื่อสลับข้อมูลและจะมาเก็บไว้ในส่วนพักข้อมูลออก เพื่อนำข้อมูลออกไปสู่ภายนอกทีละ 16 บิทจำนวน 4 ครั้ง

4.2.1.1 ส่วนพักข้อมูลเข้า (Input Buffer)

ในส่วนนี้จะรับข้อมูลเข้ามาจำนวน 16 บิททั้งหมด 4 ครั้งเพื่อที่จะได้ข้อมูลขนาด 64 บิทและส่งต่อไปยังส่วนอินนิเชียลเพอมิวเทชั่นต่อไป สำหรับข้อมูลที่ส่งเข้ามานี้จะส่งจากบิทต่ำก่อนแล้วจึงส่งบิทสูงมา การทำงานจะเริ่มเมื่อมีสัญญาณนาฬิกาเข้ามาโดยจะทำในขอบขาขึ้น

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1 จะนำข้อมูลเข้ามาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 15-0
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 2 จะนำข้อมูลเข้ามาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 31-16
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 3 จะนำข้อมูลเข้ามาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 47-32
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 4 จะนำข้อมูลเข้ามาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 63-48
- เมื่อได้รับสัญญาณนาฬิกาตั้งแต่ครั้งที่ 5 ขึ้นไปจะคงค่าเดิมไว้จนกว่าจะมีสัญญาณรีเซต (Reset)

4.2.1.2 ส่วนพักคีย์ (Key Buffer)

ในส่วนนี้จะมีการทำงานคล้ายกับส่วนพักข้อมูลเข้าโดยจะรับคีย์เข้ามาจำนวน 16 บิททั้งหมด 4 ครั้งเพื่อที่จะได้คีย์ขนาด 64 บิทและส่งต่อไปยังส่วนสร้างคีย์ต่อไป สำหรับคีย์ที่ส่งเข้ามานี้จะส่งจากบิทต่ำก่อนแล้วจึงส่งบิทสูงมา การทำงานจะเริ่มเมื่อมีสัญญาณนาฬิกาเข้ามาโดยจะทำในขอบขาขึ้น

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1 จะนำคีย์มาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 15-0
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 2 จะนำคีย์มาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 31-16
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 3 จะนำคีย์มาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 47-32
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 4 จะนำคีย์มาเก็บไว้ในส่วนพักข้อมูลชั่วคราวบิทที่ 63-48
- เมื่อได้รับสัญญาณนาฬิกาตั้งแต่ครั้งที่ 5 ขึ้นไปจะคงค่าเดิมไว้จนกว่าจะมีสัญญาณรีเซต

4.2.1.3 ส่วนพักข้อมูลออก (Output Buffer)

ส่วนนี้จะทำหน้าที่นำข้อมูลจากที่พักข้อมูลชั่วคราวขนาด 64 บิทออกไปยังภายนอกโดยผ่านทางสัญญาณ 16 บิทเป็นจำนวน 4 ครั้งโดยจะมีการทำงานในแต่ละครั้งของสัญญาณนาฬิกาดังนี้

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1 จะนำข้อมูลบิทที่ 15-0 ออกจากที่พักข้อมูล
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 2 จะนำข้อมูลบิทที่ 31-16 ออกจากที่พักข้อมูล
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 3 จะนำข้อมูลบิทที่ 47-32 ออกจากที่พักข้อมูล
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 4 จะนำข้อมูลบิทที่ 63-48 ออกจากที่พักข้อมูล
- เมื่อได้รับสัญญาณนาฬิกาตั้งแต่ครั้งที่ 5 ขึ้นไปจะไม่เกิดอะไรขึ้นจนกว่าจะมีสัญญาณรีเซต

4.2.1.4 อินิเชียลเพอมิวเทชั่น (Initial Permutation)

ส่วนนี้จะทำการสลับบิทตามอัลกอลิทึมดีอีเอส โดยจะมีข้อมูลเข้าขนาด 64 บิทและข้อมูลออกขนาด 64 บิทในส่วนนี้จะทำงานโดยไม่ขึ้นกับสัญญาณนาฬิกา โดยจะเชื่อมต่อระหว่างส่วนพักข้อมูลเข้ากับส่วนเลือกข้อมูลเข้า

4.2.1.5 ไฟนอลเพอมิวเทชั่น (Final Permutation)

จะมีหน้าที่ในการสลับตำแหน่งของบิทข้อมูล โดยตำแหน่งในการสลับจะตรงข้ามกับอินนิเชียลเพอมิวเทชั่น ข้อมูลเข้ามีขนาด 64 บิทและข้อมูลออกก็มีขนาด 64 บิท การทำงานของส่วนนี้จะไม่ขึ้นกับสัญญาณนาฬิกาล่าวคือการทำงานของส่วนนี้จะทำงานอยู่ตลอดเวลาเมื่อใดที่ข้อมูลเข้าเปลี่ยนข้อมูลออกก็จะเปลี่ยนแปลงด้วยในทันทีทันใด

4.2.1.6 ส่วนสร้างคีย์ (Key Generator)

ในส่วนนี้จะมีหน้าที่ในการสร้างคีย์ให้กับวงจรในแต่ละรอบโดยเมื่อได้รับคีย์เข้ามาจะทำการนำคีย์นั้นไปคำนวณเพื่อหาคีย์ทั้ง 16 ตัวและเก็บไว้ในที่พักคีย์ชั่วคราวสำหรับคีย์ตัวที่ 1 ถึงตัวที่ 16 เลยในทันทีโดยไม่ต้องรอสัญญาณนาฬิกา และเมื่อมีสัญญาณนาฬิกามาก็จะทำการเลือกข้อมูลจากที่พักคีย์ชั่วคราวออกมาเป็นเอาท์พุทดังนี้

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 1 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 16 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 2 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 2 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 15 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 3 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 3 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 14 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 4 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 4 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 13 ออกมา

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 5 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 5 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 12 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 6 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 6 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 11 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 7 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 7 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 10 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 8 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 8 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 9 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 9 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 9 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 8 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 10 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 10 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 7 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 11 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 11 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 6 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 12 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 12 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 5 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 13 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 13 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 4 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 14 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 14 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 3 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 15 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 15 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 2 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 16 ถ้าเป็นการเข้ารหัสจะนำข้อมูลจากที่พักคีย์อันที่ 16 ออกมาถ้าเป็นการถอดรหัสจะนำข้อมูลจากที่พักคีย์อันที่ 1 ออกมา
- เมื่อได้รับสัญญาณนาฬิกาตั้งแต่ครั้งที่ 17 ขึ้นไปจะไม่เกิดอะไรขึ้นจนกว่าจะมีสัญญาณรีเซต

4.2.1.7 ส่วนเลือกข้อมูลเข้า (Input Multiplexor)

ส่วนนี้จะทำการเลือกข้อมูลเพื่อนำเข้าไปทำตามอัลกอลิทึมดีอีเอส โดยจะเลือกระหว่างข้อมูลจากภายนอกและข้อมูลที่ได้จากการกระทำในรอบก่อนหน้านี้โดยอาศัยสัญญาณนาฬิกาเป็นตัวควบคุมดังนี้

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1 จะนำข้อมูลจากที่พักข้อมูลเข้า ส่งไปทำตามอัลกอลิทึมดีอีเอส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 2-16 จะนำข้อมูลจากส่วนเลือกข้อมูลออกที่ผ่านการกระทำในรอบที่แล้วส่งไปกระทำตามอัลกอลิทึมดีอีเอส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 17 ขึ้นไปจะไม่เกิดอะไรขึ้นจนกว่าจะมีสัญญาณรีเซต

4.2.1.8 ส่วนเลือกข้อมูลออก (Output Multiplexor)

ส่วนนี้จะทำการเลือกข้อมูลหลังจากที่ผ่านการกระทำมาแล้วว่าจะส่งไปกระทำยังรอบต่อไป (กรณีที่เป็นรอบที่ 1-15) หรือส่งออกไปเป็นผลลัพธ์เมื่อทำครบ 16 รอบแล้วโดยจะทำตามจังหวะสัญญาณนาฬิกาขอบขาขึ้นดังนี้

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1-15 จะนำข้อมูลที่ได้มาจากการกระทำในรอบที่แล้ว ส่งไปกระทำตามอัลกอลิทึมดีอีเอสในรอบถัดไป
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 16 จะนำข้อมูลที่ได้มาจากการกระทำในรอบที่แล้ว ส่งออกไปยังไฟนอลเพอมิวเทชั่นเพื่อเป็นผลลัพธ์ต่อไป
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 17 ขึ้นไปจะไม่เกิดอะไรขึ้นจนกว่าจะมีสัญญาณรีเซต

4.2.1.9 เอ็กแพนชั่นเพอมิวเทชั่น (Expansion Permutation)

ส่วนนี้จะทำการขยายบิทจาก 32 บิทไปเป็น 48 บิทโดยมีการสลับตำแหน่งตามอัลกอลิทึมดีอีเอสด้วย โดยจะเชื่อมต่อระหว่างส่วนเลือกข้อมูลเข้าบิทที่ 31-0 กับเอ็กคลูซีฟออ 48 บิทส่วนนี้จะทำงานโดยไม่ขึ้นอยู่กับสัญญาณนาฬิกา

4.2.1.10 เอสบ็อก (S-box)

ส่วนนี้จะมีอินพุทจำนวน 48 บิทซึ่งรับมาจากเอ็กคลูซีฟออ 48 บิทและส่งเอาท์พุทขนาด 32 บิทไปยังพีบ็อก ซึ่งจะมีการกำหนดค่าให้ตามอัลกอลิทึมดีอีเอส ในส่วนนี้จะมีการใช้ตารางทั้งหมด 8 ตารางเพื่อกำหนดค่าให้กับแต่ละบิทซึ่งจะเป็นส่วนที่ทำให้เปลืองหน่วยความจำของวงจรมากที่สุด ซึ่งเราอาจจะลดตารางในส่วนนี้ลงได้หากมีหน่วยความจำที่ไม่เพียงพอ การทำงานในส่วนนี้จะเริ่มทำงานเมื่อได้รับสัญญาณนาฬิกาขอบขาขึ้นกล่าวคือเอาท์พุทจะเปลี่ยนแปลงตามอินพุทเมื่อได้รับสัญญาณนาฬิกา แต่เนื่องจากเราไม่จำเป็นที่จะต้องควบคุมการไหลของข้อมูลในส่วนนี้เราอาจจะตัดสัญญาณนาฬิกาที่ควบคุมการทำงานในส่วนนี้ออกไปได้เพื่อลดความซับซ้อนของวงจร

4.2.1.11 พีบ็อก (P-box)

ส่วนนี้จะเชื่อมต่อระหว่างเอสบ็อกและเอ็กคลูซีฟออ 32 บิทโดยจะมีอินพุทและเอาท์พุทขนาด 32 บิท ทำหน้าที่ในการสลับตำแหน่งของบิทให้เป็นไปตามอัลกอลิทึมดีอีเอส การทำงานในส่วนนี้จะไม่ขึ้นกับสัญญาณนาฬิกากล่าวคือเอาท์พุทจะเปลี่ยนแปลงทันทีเมื่ออินพุทมีการเปลี่ยนแปลง

4.2.1.12 เอ็กคลูซีฟออ 48 บิท (Exclusive OR 48)

ส่วนนี้จะทำหน้าที่ในการเอ็กคลูซีฟอออข้อมูลขนาด 48 บิทโดยจะรับข้อมูลเข้าขนาด 48 บิทจากเอ็กแพนชั่นเพอมิวเทชั่นและส่วนสร้างคีย์ ผลลัพท์จำนวน 48 บิทก็จะถูกส่งต่อไปยังเอสบ็อก โดยจะทำงานเมื่อได้รับสัญญาณนาฬิกาขอบขาขึ้น เอาท์พุทก็จะเปลี่ยนแปลงตามอินพุทและจะค้างค่านั้นไว้จนกว่าจะมีสัญญาณนาฬิกาขอบขาขึ้นเข้ามาอีก เอาท์พุทจึงจะเปลี่ยนแปลงตามอินพุทอีกครั้งหนึ่ง

4.2.1.13 เอ็กคลูซีฟออ 32 บิท (Exclusive OR 32)

ส่วนนี้จะทำหน้าที่ในการเอ็กคลูซีฟออข้อมูลขนาด 32 บิทโดยจะรับข้อมูลเข้าขนาด 32 บิทจากพีบ็อกและจากส่วนเลือกอินพุทบิทที่ 63-32 ผลลัพท์จำนวน 32 บิทก็จะถูกส่งต่อไปยังส่วนเลือกเอาท์พุทบิทที่ 31-0 ซึ่งจะทำงานเมื่อได้รับสัญญาณนาฬิกาเปลี่ยนจากระดับต่ำเป็นระดับสูง เอาท์พุทก็จะเปลี่ยนแปลงตามอินพุทและจะคงค่านั้นไว้จนกว่าจะมีสัญญาณนาฬิกาเข้ามาอีกครั้ง เอาท์พุทจึงจะเปลี่ยนแปลงตามอินพุทอีกที

4.2.1.14 ส่วนควบคุมวงจร (Control)

ส่วนนี้เป็นส่วนที่สำคัญที่สุดในวงจร ซึ่งจะมีหน้าที่ในการควบคุมการไหลของข้อมูลให้เป็นไปตามอันดับของอัลกอริทึมดีอีเอส และยังเป็นตัวที่ทำการรีเซตส่วนอื่น ๆ ทั้งหมดอีกด้วย สำหรับการควบคุมอุปกรณ์ส่วนอื่น ๆ นั้นจะใช้สัญญาณอยู่สองอย่างคือสัญญาณนาฬิกาเพื่อบอกจังหวะการทำงานและสัญญาณรีเซตเพื่อให้ในส่วนนั้นเริ่มต้นทำงานใหม่ สำหรับส่วนควบคุมจะมีอินพุทที่ใช้ควบคุมอยู่สามอย่างคือ สัญญาณนาฬิกาจากภายนอกเพื่อกำหนดจังหวะการทำงาน สัญญาณรีเซตและสัญญาณเลือกว่าจะทำการเข้ารหัสหรือถอดรหัส โดยในแต่ละสัญญาณนาฬิกาที่รับเข้ามาจะมีการทำงานดังต่อไปนี้

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 1-5 จะเป็นการรีเซตส่วนประกอบทุกส่วนในวงจรและพร้อมที่จะรับข้อมูลเข้ามาเพื่อทำการเข้ารหัสหรือถอดรหัส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 6-7 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลเข้ากับส่วนพักคีย์เพื่อให้รับข้อมูลที่จะนำมาเข้ารหัสหรือถอดรหัสและคีย์บิทที่ 15-0
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 8-9 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลเข้ากับส่วนพักคีย์เพื่อให้รับข้อมูลที่จะนำมาเข้ารหัสหรือถอดรหัสพร้อมกับคีย์บิทที่ 31-16
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 10-11 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลเข้ากับส่วนพักคีย์เพื่อให้รับข้อมูลที่จะนำมาเข้ารหัสหรือถอดรหัสพร้อมคีย์บิทที่ 47-32
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 12-13 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลเข้ากับส่วนพักคีย์เพื่อให้รับข้อมูลที่จะนำมาเข้ารหัสหรือถอดรหัสพร้อมคีย์บิทที่ 63-48
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 14-15 จะส่งสัญญาณนาฬิกาไปที่ส่วนเลือกข้อมูลเข้าและส่วนสร้างคีย์ เพื่อที่จะนำข้อมูลบิทที่ 31-0 จากส่วนเลือกข้อมูลเข้าผ่านเอ็กแพนชั่นเพอมิวเทชั่นทำให้เป็นข้อมูลขนาด 48 บิทและคีย์ของในแต่ละรอบส่งไปยังเอ็กคลูซีฟออ 48 บิท
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 16-17 จะส่งสัญญาณนาฬิกาไปที่เอ็กคลูซีฟออ 48 บิทเพื่อให้เริ่มทำงานและส่งข้อมูลต่อไปยังเอสบ็อก
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 18-19 จะส่งสัญญาณนาฬิกาไปที่เอสบ็อกเพื่อให้เริ่มทำงานและนำข้อมูลที่ผ่านขบวนการส่งต่อไปยังเอ็กคลูซีฟออ 32 บิท
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 20-21 จะส่งสัญญาณนาฬิกาไปที่เอ็กคลูซีฟออ 32 บิทเพื่อให้เริ่มทำงาน

- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 22-23 จะส่งสัญญาณนาฬิกาไปที่ส่วนเลือกข้อมูลออกซึ่งจะสิ้นสุดการทำงานในรอบที่หนึ่งของอัลกอลิทึมดีอีเอส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 24-173 จะทำการส่งสัญญาณนาฬิกาเหมือนสัญญาณนาฬิกาครั้งที่ 14-23 แต่จะเป็นรอบที่ 2-16 ของอัลกอลิทึมดีอีเอส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 174-175 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลออกเพื่อที่จะนำข้อมูลบิทที่ 15-0 จากที่พักข้อมูลออกไปเป็นเอาท์พุทของส่วนการเข้ารหัสและถอดรหัส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 176-177 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลออกเพื่อที่จะนำข้อมูลบิทที่ 31-16 จากที่พักข้อมูลออกไปเป็นเอาท์พุทของส่วนการเข้ารหัสและถอดรหัส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 178-179 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลออกเพื่อที่จะนำข้อมูลบิทที่ 47-32 จากที่พักข้อมูลออกไปเป็นเอาท์พุทของส่วนการเข้ารหัสและถอดรหัส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 180-181 จะส่งสัญญาณนาฬิกาไปที่ส่วนพักข้อมูลออกเพื่อที่จะนำข้อมูลบิทที่ 63-48 จากที่พักข้อมูลออกไปเป็นเอาท์พุทของส่วนการเข้ารหัสและถอดรหัส
- เมื่อได้รับสัญญาณนาฬิกาครั้งที่ 182 ขึ้นไปจะไม่มีการทำงานใด ๆ จนกว่าจะได้รับสัญญาณรีเซต

4.2.1.15 วงจรเข้ารหัสและถอดรหัสดีอีเอส (Main DES)

ในส่วนนี้จะเป็นระดับบนสุดที่เชื่อมต่อทุกส่วนประกอบเข้าด้วยกัน โดยจะมีอินพุทข้อมูลขนาด 16 บิท อินพุทคีย์ที่ใช้เข้ารหัสขนาด 16 บิท มีเอาท์พุทข้อมูลขนาด 16 บิท สัญญาณที่ใช้ควบคุมจะมีทั้งหมด 3 สัญญาณคือ สัญญาณนาฬิกาใช้เพื่อกำหนดจังหวะในการทำงานของวงจร สัญญาณเลือกว่าจะเข้ารหัสหรือถอดรหัส และสัญญาณรีเซตเพื่อเริ่มต้นระบบใหม่อีกครั้ง

4.2.2 การแปลงแต่ละส่วนให้เป็นโค้ดวีเอชดีแอล

เมื่อเราได้ทำการแยกวงจรดีอีเอสออกเป็นส่วน ๆ เพื่อให้ง่ายต่อการเขียนโค้ดวีเอชดีแอลแล้วต่อไปเราก็จะนำในแต่ละส่วนนี้มาเขียนอธิบายการทำงานด้วยภาษาวีเอชดีแอลได้ดังนี้

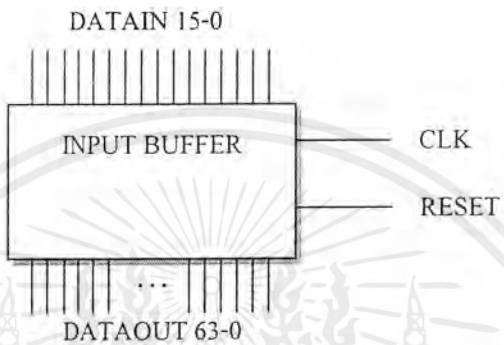
4.2.2.1 ส่วนพักข้อมูลเข้า

ส่วนพักข้อมูลเข้ามีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-6 และอินพุทเอาท์พุทดังรูปที่ 4-7 ขาสัญญาณต่าง ๆ ของส่วนพักข้อมูลเข้ามีดังนี้

DATAIN (15-0)	อินพุทข้อมูลจำนวน 16 บิท
DATAOUT (63-0)	เอาท์พุทข้อมูลจำนวน 64 บิท
CLK	กำหนดจังหวะในการรับข้อมูลและเปลี่ยนสถานะ
RESET	เริ่มต้นการรับข้อมูลใหม่

```
entity inbuffer is
        port (datain : in std_logic_vector(15 downto 0);
            dataout : out std_logic_vector(63 downto 0);
            clk,reset : in std_logic);
end  inbuffer;
```

รูปที่ 4-6 เอนติตี้ของส่วนพักข้อมูลเข้า



รูปที่ 4-7 อินพุทและเอาท์พุทของส่วนพักข้อมูลเข้า

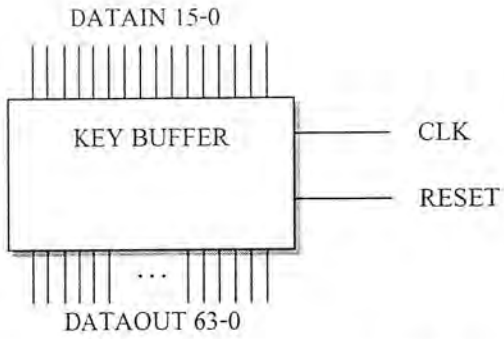
4.2.2.2 ส่วนพักคีย์

ส่วนพักคีย์มีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-8 และอินพุทเอาท์พุทดังรูปที่ 4-9 ขาสัญญาณต่าง ๆ ของส่วนพักคีย์มีดังนี้

DATAIN (15-0)	อินพุทข้อมูลจำนวน 16 บิท
DATAOUT (63-0)	เอาท์พุทข้อมูลจำนวน 64 บิท
CLK	กำหนดจังหวะในการรับข้อมูลและเปลี่ยนสถานะ
RESET	เริ่มต้นการรับข้อมูลใหม่

```
entity keybuffer is
        port (datain : in std_logic_vector(15 downto 0);
            dataout : out std_logic_vector(63 downto 0);
            clk,reset : in std_logic);
end  keybuffer;
```

รูปที่ 4-8 เอนติตี้ของส่วนพักคีย์



รูปที่ 4-9 อินพุทและเอาท์พุทของส่วนพักคีย์

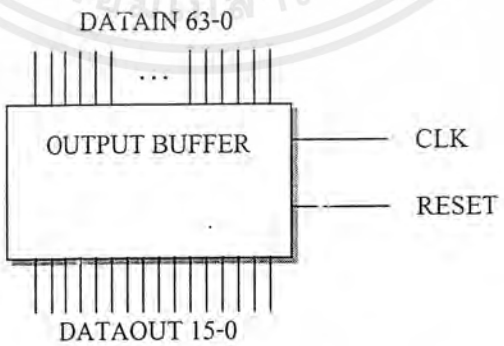
4.2.2.3 ส่วนพักข้อมูลออก

ส่วนพักข้อมูลออกมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-10 และอินพุทเอาท์พุทดังรูปที่ 4-11 ขาสัญญาณต่าง ๆ ของส่วนพักข้อมูลออกมีดังนี้

DATAIN (63-0)	อินพุทข้อมูลจำนวน 64 บิท
DATAOUT (15-0)	เอาท์พุทข้อมูลจำนวน 16 บิท
CLK	กำหนดจังหวะในการส่งข้อมูลและเปลี่ยนสถานะ
RESET	เริ่มต้นการทำงานใหม่

```
entity outbuffer is
        port (datain : in std_logic_vector(63 downto 0);
              dataout : out std_logic_vector(15 downto 0);
              clk,reset : in std_logic);
end  outbuffer;
```

รูปที่ 4-10 เอนติตี้ของส่วนพักข้อมูลออก



รูปที่ 4-11 อินพุทและเอาท์พุทของส่วนพักข้อมูลออก

4.2.2.4 อินิเชียลเพอมิวเทชั่น (Initial Permutation)

อินิเชียลเพอมิวเทชั่นมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-12 และอินพุทเอาท์พุทดังรูปที่ 4-13 ขาสัญญาณต่าง ๆ ของอินิเชียลเพอมิวเทชั่นมีดังนี้

I (64-1) อินพุทข้อมูลจำนวน 64 บิท

O (64-1) เอาท์พุทข้อมูลจำนวน 64 บิท

```
entity permute is
        port (i : in std_logic_vector(64 downto 1);
              o : out std_logic_vector(64 downto 1));
end  permute;
```

รูปที่ 4-12 เอนติตี้ของอินิเชียลเพอมิวเทชั่น

รูปที่ 4-13 อินพุทและเอาท์พุทของอินิเชียลเพอมิวเทชั่น

4.2.2.5 ไฟนอลเพอมิวเทชั่น (Final Permutation)

ไฟนอลเพอมิวเทชั่นมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-14 และอินพุทเอาท์พุทดังรูปที่ 4-15 ขาสัญญาณต่าง ๆ ของไฟนอลเพอมิวเทชั่นมีดังนี้

I (64-1) อินพุทข้อมูลจำนวน 64 บิท

O (64-1) เอาท์พุทข้อมูลจำนวน 64 บิท

```
entity depermute is
        port (i : in std_logic_vector(64 downto 1);
              o : out std_logic_vector(64 downto 1));
end  depermute;
```

รูปที่ 4-14 เอนติตี้ของไฟนอลเพอมิวเทชั่น

รูปที่ 4-15 อินพุทและเอาท์พุทของไฟนอลเพอมิวเทชั่น

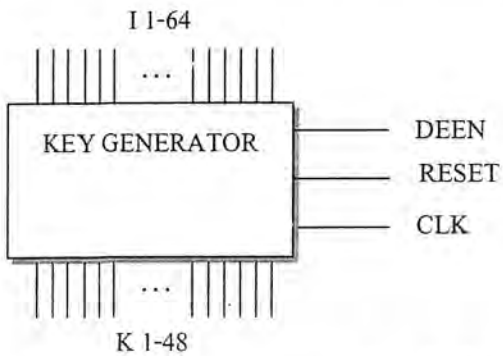
4.2.2.6 ส่วนสร้างคีย์ (Key Generator)

ส่วนสร้างคีย์มีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-16 และอินพุทเอาท์พุทดังรูปที่ 4-17 ขาสัญญาณต่าง ๆ ของส่วนสร้างคีย์มีดังนี้

I (1-64)	รับคีย์เข้ามา
K (1-48)	เอาท์พุทของคีย์ในแต่ละรอบของการเข้ารหัส
DEEN	เลือกว่าจะทำการเข้ารหัสหรือถอดรหัส
CLK	กำหนดจังหวะในการทำงาน
RESET	เริ่มต้นการทำงานใหม่

```
entity keycom is
        port (i : in std_logic_vector(1 to 64);
              k : out std_logic_vector(1 to 48);
              clk,reset,deen : in std_logic);
end  keycom;
```

รูปที่ 4-16 เอนติตี้ของส่วนสร้างคีย์



รูปที่ 4-17 อินพุทและเอาท์พุทของส่วนสร้างคีย์

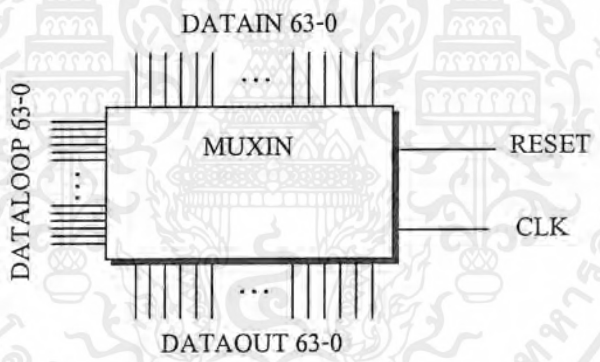
4.2.2.7 ส่วนเลือกข้อมูลเข้า (Input Multiplexor)

ส่วนเลือกข้อมูลเข้ามีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-18 และอินพุทเอาท์พุทดังรูปที่ 4-19 ขาสัญญาณต่าง ๆ ของส่วนเลือกข้อมูลเข้ามีดังนี้

DATAIN (63-0)	อินพุทข้อมูลจากภายนอก
DATALOOP (63-0)	อินพุทข้อมูลที่กลับมาเข้ารหัสอีกรอบ
DATAOUT (63-0)	เอาท์พุทข้อมูลไปยังส่วนอื่น
CLK	กำหนดจังหวะในการทำงานและเปลี่ยนสถานะ
RESET	เริ่มต้นการทำงานใหม่

```
entity muxin is
        port (datain,dataloop : in std_logic_vector(63 downto 0);
            dataout : out std_logic_vector(63 downto 0);
            clk,reset : in std_logic);
end  muxin;
```

รูปที่ 4-18 เอนติตี้ของส่วนเลือกข้อมูลเข้า



รูปที่ 4-19 อินพุทและเอาท์พุทของส่วนเลือกข้อมูลเข้า

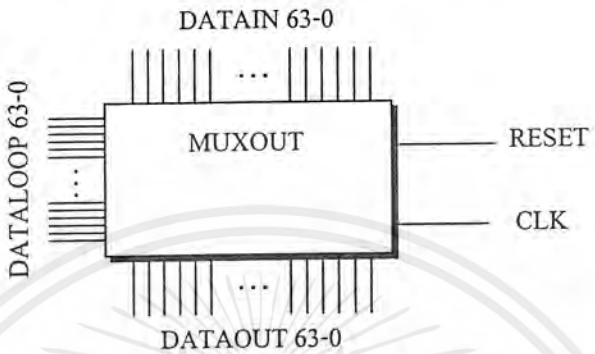
4.2.2.8 ส่วนเลือกข้อมูลออก (Output Multiplexor)

ส่วนเลือกข้อมูลออกมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-20 และอินพุทเอาท์พุทดังรูปที่ 4-21 ขาสัญญาณต่าง ๆ ของส่วนเลือกข้อมูลออกมีดังนี้

DATAIN (63-0)	อินพุทข้อมูลที่ผ่านขบวนการมาแล้วหนึ่งรอบ
DATALOOP (63-0)	เอาท์พุทข้อมูลที่ส่งไปเข้ารหัสอีกรอบ
DATAOUT (63-0)	เอาท์พุทข้อมูลที่ส่งไปยังที่พักข้อมูลออก
CLK	กำหนดจังหวะในการทำงานและเปลี่ยนสถานะ
RESET	เริ่มต้นการทำงานใหม่

```
entity muxout is
        port (datain : in std_logic_vector(63 downto 0);
            dataout,dataloop : out std_logic_vector(63 downto 0);
            clk,reset : in std_logic);
end  muxout;
```

รูปที่ 4-20 เอนติตี้ของส่วนเลือกข้อมูลออก



รูปที่ 4-21 อินพุทและเอาท์พุทของส่วนเลือกข้อมูลออก

4.2.2.9 เอ็กแพนชั่นเพอมิวเทชั่น (Expansion Permutation)

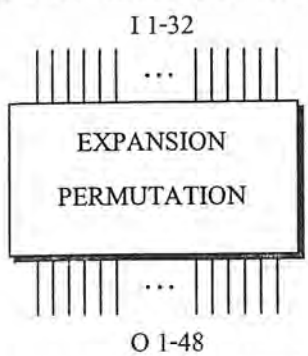
เอ็กแพนชั่นเพอมิวเทชั่นมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-22 และอินพุทเอาท์พุทดังรูปที่ 4-23 ขาสัญญาณต่าง ๆ ของเอ็กแพนชั่นเพอมิวเทชั่นมีดังนี้

I (1-32) อินพุทข้อมูลจำนวน 32 บิท

O (1-48) เอาท์พุทข้อมูลจำนวน 48 บิท

```
entity exp is
        port (i : in std_logic_vector(1 to 32);
            o : out std_logic_vector(1 to 48));
end  exp;
```

รูปที่ 4-22 เอนติตี้ของเอ็กแพนชั่นเพอมิวเทชั่น



รูปที่ 4-23 อินพุทและเอาท์พุทของเอ็กแพนชั่นเพอมิวเทชั่น

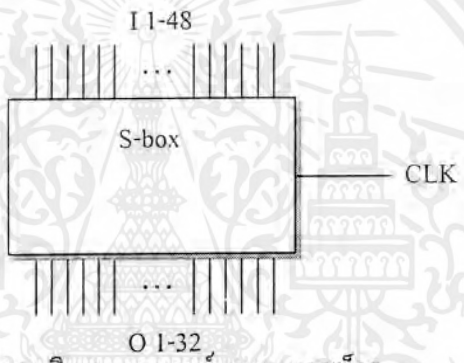
4.2.2.10 เอสบ็อก

เอสบ็อกมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-24 และอินพุทเอาท์พุทดังรูปที่ 4-25 ขาสัญญาณต่าง ๆ ของเอสบ็อกมีดังนี้

I (1-48)	อินพุทข้อมูลจำนวน 32 บิท
O (1-32)	เอาท์พุทข้อมูลจำนวน 48 บิท
CLK	กำหนดจังหวะในการทำงานและเปลี่ยนสถานะ

```
entity sbox is
        port (i : in std_logic_vector(1 to 48);
              clk : in std_logic;
              o : out std_logic_vector(1 to 32));
end sbox;
```

รูปที่ 4-24 เอนติตี้ของเอสบ็อก



รูปที่ 4-25 อินพุทและเอาท์พุทของเอสบ็อก

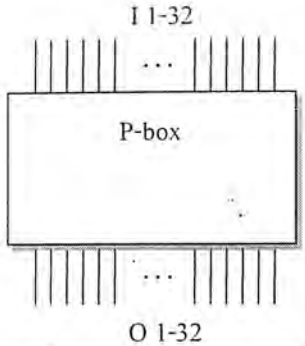
4.2.2.11 พีบ็อก

พีบ็อกมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-26 และอินพุทเอาท์พุทดังรูปที่ 4-27 ขาสัญญาณต่าง ๆ ของพีบ็อกมีดังนี้

I (1-32)	อินพุทข้อมูลจำนวน 32 บิท
O (1-32)	เอาท์พุทข้อมูลจำนวน 48 บิท

```
entity pp is
        port (i : in std_logic_vector(1 to 32);
              o : out std_logic_vector(1 to 32));
end pp;
```

รูปที่ 4-26 เอนติตี้ของพีบ็อก



รูปที่ 4-27 อินพุทและเอาท์พุทของพีบ็อก

4.2.2.12 เอ็กคลูซีฟออ 32 บิทและ 48 บิท

เอ็กคลูซีฟออ 32 บิทและ 48 บิทมีการทำงานเหมือนกันทุกประการแตกต่างกันที่จำนวนบิทที่เป็นอินพุทและเอาท์พุทเท่านั้นดังนั้นจึงจะอธิบายในส่วนของเอ็กคลูซีฟออ 32 บิทเพียงอย่างเดียว เอ็กคลูซีฟออ 32 บิทมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-28 และอินพุทเอาท์พุทดังรูปที่ 4-29 ขาสัญญาณต่าง ๆ ของเอ็กคลูซีฟออ 32 บิทมีดังนี้

DATAINA (31-0) อินพุทข้อมูลขนาด 32 บิท

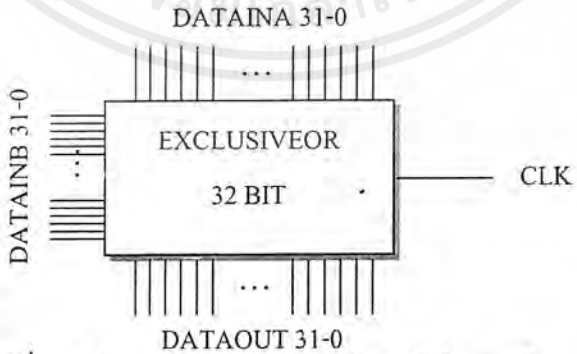
DATAINB (31-0) อินพุทข้อมูลขนาด 32 บิท

DATAOUT (31-0) เอาท์พุทผลจากการทำเอ็กคลูซีฟออ

CLK สัญญาณควบคุมจังหวะการทำงาน

```
entity xor32 is
        port (datainа,datainb : in std_logic_vector(31 downto 0);
              dataout : out std_logic_vector(31 downto 0);
              clk : in std_logic);
end  xor32;
```

รูปที่ 4-28 เอนติตี้ของเอ็กคลูซีฟออ 32 บิท



รูปที่ 4-29 อินพุทและเอาท์พุทของเอ็กคลูซีฟออ 32 บิท

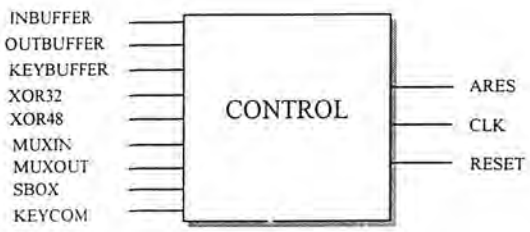
4.2.5.13 ส่วนควบคุมวงจร

ส่วนควบคุมวงจรมีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-30 และอินพุทเอาท์พุทดังรูปที่ 4-31 ขาสัญญาณต่าง ๆ ของส่วนควบคุมวงจรมีดังนี้

ARES	ขาสัญญาณที่จะทำการรีเซตทุกส่วนย่อย
INBUFFER	สัญญาณนาฬิกาที่ส่งไปยังส่วนพักข้อมูลเข้า
OUTBUFFER	สัญญาณนาฬิกาที่ส่งไปยังส่วนพักข้อมูลออก
KEYBUFFER	สัญญาณนาฬิกาที่ส่งไปยังส่วนพักคีย์
XOR32	สัญญาณนาฬิกาที่ส่งไปยังส่วนเอ็กคลูซีฟออ 32 บิท
XOR48	สัญญาณนาฬิกาที่ส่งไปยังส่วนเอ็กคลูซีฟออ 48 บิท
MUXIN	สัญญาณนาฬิกาที่ส่งไปยังส่วนเลือกข้อมูลเข้า
MUXOUT	สัญญาณนาฬิกาที่ส่งไปยังส่วนเลือกข้อมูลออก
SBOX	สัญญาณนาฬิกาที่ส่งไปยังส่วนเอสบ็อก
KEYCOM	สัญญาณนาฬิกาที่ส่งไปยังส่วนสร้างคีย์
CLK	สัญญาณควบคุมจังหวะการทำงาน
RESET	เริ่มต้นการทำงานใหม่

```
entity des is
        port    (clk,reset : in std_logic;
                 ares      : out std_logic;
                 inbuffer  : out std_logic;
                 outbuffer : out std_logic;
                 keybuffer : out std_logic;
                 xor32     : out std_logic;
                 xor48     : out std_logic;
                 muxin     : out std_logic;
                 muxout    : out std_logic;
                 sbox      : out std_logic;
                 keycom    : out std_logic);

end  des;
```

รูปที่ 4-30 เอนติตี้ของส่วนควบคุมวงจร



รูปที่ 4-31 อินพุทและเอาท์พุทของส่วนควบคุมวงจร

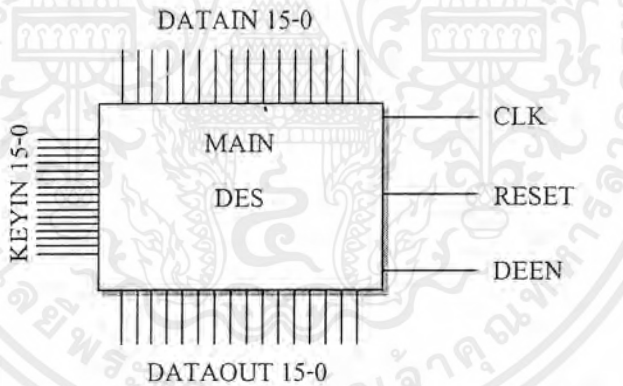
4.2.2.14 วงจรเข้ารหัสและถอดรหัสดีอีเอส

วงจรเข้ารหัสและถอดรหัสดีอีเอสเป็นส่วนบนสุดของทุกส่วนย่อยซึ่งสัญญาณจะต่อกับอุปกรณ์ภายนอกโดยตรง มีการประกาศเอนติตี้ (entity) ดังรูปที่ 4-32 และอินพุทเอาท์พุทดังรูปที่ 4-33 ขาสัญญาณต่าง ๆ ของวงจรเข้ารหัสและถอดรหัสดีอีเอสมีดังนี้

DATAIN (15-0)	ข้อมูลที่จะนำมาเข้ารหัสหรือถอดรหัส
KEYIN (15-0)	คีย์ที่ใช้ในการเข้ารหัสและถอดรหัส
DATAOUT (15-0)	ผลที่ได้จากการเข้ารหัสหรือถอดรหัส
DEEN	เลือกว่าจะทำการเข้ารหัสหรือถอดรหัส
CLK	สัญญาณควบคุมจังหวะการทำงาน
RESET	เริ่มต้นการทำงานใหม่

```
entity maindes is
        port (datain,keyin : in std_logic_vector(15 downto 0);
            dataout : out std_logic_vector(15 downto 0);
            clk,reset,deen : in std_logic);
end maindes;
```

รูปที่ 4-32 เอนติตี้ของวงจรเข้ารหัสและถอดรหัสดีอีเอส



รูปที่ 4-33 อินพุทและเอาท์พุทของวงจรเข้ารหัสและถอดรหัสดีอีเอส

4.2.3 นำโค้ดที่เขียนได้ไปทำการสังเคราะห์ (synthesis)

เมื่อเราได้เขียนบรรยายพฤติกรรมของวงจรเข้ารหัสและถอดรหัสเสร็จแล้วเราก็นำโค้ดที่ได้ไปทำการสังเคราะห์เพื่อที่จะได้รู้จำนวนเกท (logic gate) ที่จำเป็นต้องใช้ ซึ่งจะช่วยให้สามารถเลือกเบอร์ของชิปเอฟพีจีเอที่จะนำมาใช้ได้อย่างเหมาะสม จากโค้ดที่เราเขียนได้ในตอนแรกสุดซึ่งเป็นการเข้ารหัสของข้อมูล 64 บิทเมื่อนำไปสังเคราะห์ (การสังเคราะห์ได้ใช้โปรแกรม Leonardo Spectrum ในการสังเคราะห์) จะได้ผลลัพธ์ดังรูปที่ 4-34

```
Total accumulated area :
 Number of FG Function Generators :  1530
 Number of H Function Generators :   503
 Number of Packed CLBs :             798
 Number of CLB Flip Flops :          464
 Number of IBUF :                    32
 Number of IOB Output Flip Flops :   16

 Number of ports :                   51
 Number of nets :                    2586
 Number of instances :               2553
 Number of references to this view : 0

************************************************
Device Utilization for 4020eHQ208
************************************************
Resource                Used  Avail  Utilization
-------------------------------------------------
IOs                     51    160    31.87%
FG Function Generators  1530  1568   97.58%
H Function Generators   503   784    64.16%
CLB Flip Flops          464   2016   23.02%

-------------------------------------------------
Using wire table: 4020e-3
                Clock Frequency Report
     Clock          : Frequency
    ---------------------------------
      clk           : 20.2 MHz
```

รูปที่ 4-34 ผลที่ได้รับจากการทำการสังเคราะห์วงจรเข้ารหัสแบบ 64 บิท

ผลที่ได้จากการสังเคราะห์ได้แนะนำว่าควรจะใช้เอฟพีจีเอเบอร์ XC4020E จึงจะพอเพียงต่อความต้องการของวงจรแต่เนื่องจากเราไม่สามารถที่จะหาอุปกรณ์และชิปเบอร์นี้ได้อย่างสะดวก เราจึงทำการลดความซับซ้อนของวงจรลงโดยการเปลี่ยนจากวงจรเข้ารหัสและถอดรหัส 64 บิทเป็นวงจรเข้ารหัสและถอดรหัสขนาด 32 บิทแทนและในส่วนของเอสบ็อกซึ่งแต่เดิมใช้ตารางในการเปรียบเทียบค่าถึง 8 ตารางก็ได้ลดลงมาเหลือเพียง 4 ตาราง ซึ่งก็จะทำให้ลดความต้องการทรัพยากรของวงจรลงจนสามารถใช้ชิปเบอร์ XC4010E ซึ่งเป็นเบอร์ที่สามารถหาอุปกรณ์ได้ไม่ยากนัก ผลจากการสังเคราะห์ (การสังเคราะห์ครั้งนี้ใช้โปรแกรม Xilinx Foundation F2.1i ในการสังเคราะห์) โค้ดอันใหม่ดังรูปที่ 4-35

```
Design Information
------------------
Command Line   : m1map -p xc4010e-1-pc84 -o map.ncd -pr b
Target Device  : x4010e
Target Package : pc84
Target Speed   : -1
Mapper Version : xc4000e -- C.16

Design Summary
--------------
  Number of errors:              0
  Number of warnings:           11
  Number of CLBs:              347 out of   400   86%
    CLB Flip Flops:            253
    4 input LUTs:              523
    3 input LUTs:              119 (72 used as route-throughs)
  Number of bonded IOBs:        15 out of    61   24%
    IOB Flops:                 12
    IOB Latches:               0
  Number of clock IOB pads:     1 out of     8   12%
  Number of primary CLKs:       1 out of     4   25%
  Number of secondary CLKs:     3 out of     4   75%

Total equivalent gate count for design: 4939
Additional JTAG gate count for IOBs:   720

Design statistics:
  Minimum period: 48.143ns (Maximum frequency: 20.771MHz)
  Maximum net delay: 19.848ns
```

รูปที่ 4-35 ผลที่ได้รับจากการทำการสังเคราะห์วงจรเข้ารหัสแบบ 32 บิท

4.3 ส่วนการควบคุมวงจร

ส่วนนี้จะเป็นส่วนที่ทำการแปลคำสั่งที่ได้รับมาจากโปรแกรมของผู้ใช้งานและเป็นตัวกลางในการส่งข้อมูลระหว่างโปรแกรมผู้ใช้งานกับวงจรการเข้ารหัสและถอดรหัส ซึ่งเราได้นำไมโครคอนโทรลเลอร์เข้ามาใช้ในส่วนนี้ ซึ่งมีความสะดวกในการใช้งานมากเนื่องจากมีพอร์ตให้ใช้งานพอเพียงและยังมีส่วนในการติดต่อกับคอมพิวเตอร์ทางพอร์ตอนุกรมไว้ให้อยู่แล้ว พอร์ตของไมโครคอนโทรลเลอร์ที่เรานำมาใช้งานนั้นเราจะใช้พอร์ต 1 และพอร์ต 2 ซึ่งจะมีขาสัญญาณให้ใช้ทั้งหมด 16 ขา แต่ในวงจรเข้ารหัสและถอดรหัสเราได้ทำการเปลี่ยนแปลงจนเหลือสัญญาณที่ใช้เพียง 15 ขาดังนี้

- ขาอินพุทข้อมูล 4 ขา
- ขาอินพุทคีย์ 4 ขา
- ขาเอาท์พุทข้อมูล 4 ขา
- ขาสัญญาณนาฬิกา
- ขารีเซต
- ขาเลือกว่าจะทำการเข้ารหัสหรือถอดรหัส

ความจริงแล้วไมโครคอนโทรลเลอร์ยังมีพอร์ต 0 ให้ใช้งานอีกจำนวน 8 ขาแต่ที่เราไม่นำมาใช้เนื่องจากพอร์ต 0 นี้ไม่มีตัวต้านทานภายใน ซึ่งถ้าเราต้องการจะใช้งานพอร์ตนี้เราต้องนำตัวต้านทานมาต่อภายนอกเองซึ่งจะทำให้ไม่สะดวกในการใช้เราจึงทำการลดขาสัญญาณที่ตัววงจรเข้ารหัสดีอีเอสแทน

4.4 ส่วนติดต่อกับคอมพิวเตอร์

ส่วนนี้จะเป็นส่วนที่ทำการติดต่อกับคอมพิวเตอร์ทางพอร์ตอนุกรม เนื่องจากเราได้ใช้ไมโครคอนโทรลเลอร์เป็นส่วนในการควบคุมวงจรอยู่แล้ว ซึ่งไมโครคอนโทรลเลอร์ก็มีความสามารถในการติดต่อกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมโดยผ่านทางพอร์ต 3.0 (RxD) และพอร์ต 3.1 (TxD) เราเพียงแต่นำชิปที่เปลี่ยนระดับสัญญาณจาก RS-232 (+20V - -20V) เป็น TTL/CMOS (0V - 5V) และจาก TTL/CMOS เป็น RS-232 มาต่อทางขาสัญญาณ RxD และ TxD ก็จะสามารถติดต่อกับคอมพิวเตอร์ได้แล้ว สำหรับชิปที่เราใช้เปลี่ยนสัญญาณนั้นคือ MAX232

บทที่ 5
การออกแบบโปรแกรมติดต่อกับผู้ใช้งาน

สำหรับในส่วนของการออกแบบส่วนที่จะทำการติดต่อกับผู้ใช้งานนั้น เราจะแบ่งออกเป็น 3 ส่วน ที่สำคัญ คือ ส่วนที่ใช้ในการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ส่วนโปรแกรมของผู้ผลิตซอฟต์แวร์ และส่วนของโปรแกรมของผู้ที่ซื้อซอฟต์แวร์ไปใช้งาน โดยจะมีรายละเอียดในแต่ละส่วน ดังต่อไปนี้

5.1 ส่วนการติดต่อกับฮาร์ดแวร์

สำหรับในส่วนนี้ จะเป็นส่วนที่คอมพิวเตอร์ใช้ทำการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ผ่านทางส่วนติดต่อกับคอมพิวเตอร์ของฮาร์ดล็อค โดยที่ในส่วนของส่วนติดต่อกับคอมพิวเตอร์นั้นจะมีการนำไมโครคอนโทรลเลอร์ (Microcontroller) มาใช้งาน เนื่องจากความจุของอุปกรณ์เอฟพีจีเอ (FPGA) รุ่นที่เรานำมาใช้นั้นมีความจุไม่เพียงพอ จึงต้องมีการนำไมโครคอนโทรลเลอร์มาช่วย ซึ่งในโครงงานนี้เราใช้ไมโครคอนโทรลเลอร์ เบอร์ 89C52 ในการออกแบบ ซึ่งในการติดต่อนั้นจะติดต่อผ่านทางพอร์ตอนุกรม (Serial Port)

ในส่วนของรูปแบบของการติดต่อระหว่างคอมพิวเตอร์กับส่วนติดต่อนั้น จะใช้รูปแบบของคำสั่งที่มีขนาด 4 ไบท์ โดยใน 3 ไบท์แรกของคำสั่งนั้นจะเป็นคำสั่งที่ใช้เป็นตัวกระตุ้นให้คอนโทรลเลอร์ (Controller) นั้นพร้อมที่จะรับคำสั่ง โดยจะทำหน้าที่เป็นเสมือนส่วนหัวของคำสั่ง (Header) ส่วนไบท์สุดท้ายของคำสั่งนั้นจะเป็นรูปแบบคำสั่งที่จะสั่งให้ตัวคอนโทรลเลอร์นั้นทำงานตามหน้าที่ที่กำหนด โดยจะสามารถกำหนดคำสั่งให้ตัวคอนโทรลเลอร์ได้ทั้งหมดถึง 256 รูปแบบที่แตกต่างกัน (00000000 ถึง 11111111) แต่ในโครงงานนี้เรานำมาใช้งานแค่ 5 รูปแบบ ดังนี้

- **แบบที่ 1** :ไบท์ที่ 4 เป็น 00110001 – สั่งให้คอนโทรลเลอร์พร้อมรับข้อมูล 32 บิทจากโปรแกรม
- **แบบที่ 2** : ไบท์ที่ 4 เป็น 00110010 – สั่งให้คอนโทรลเลอร์พร้อมส่งข้อมูล 32 บิทมาให้โปรแกรม
- **แบบที่ 3** :ไบท์ที่ 4 เป็น 00110011 – สั่งให้คอนโทรลเลอร์ส่งข้อมูล 32 บิทไปทำการเข้ารหัสในส่วนของการเข้ารหัสและถอดรหัสแบบดีอีเอส (DES)
- **แบบที่ 4** : ไบท์ที่ 4 เป็น 00110100 – สั่งให้คอนโทรลเลอร์ส่งข้อมูล 32 บิทไปทำการถอดรหัสในส่วนของการเข้ารหัสและถอดรหัสแบบดีอีเอส
- **แบบที่ 5** :ไบท์ที่ 4 เป็น 00110101 – สั่งให้คอนโทรลเลอร์ส่งข้อความตอบสนองมายังคอมพิวเตอร์ เพื่อเป็นการตรวจสอบว่ามีการต่อฮาร์ดล็อคอยู่

สำหรับการติดต่อระหว่างคอมพิวเตอร์กับตัวส่วนติดต่อนั้น ในขั้นตอนการส่งคำสั่งจากคอมพิวเตอร์ไปยังส่วนติดต่อนั้นจะเป็นการส่งคำสั่งขนาด 4 ไบท์แบบธรรมดา แต่ในขั้นตอนของการที่

ส่วนติดต่อจะส่งข้อมูลกลับมาให้กับคอมพิวเตอร์นั้น ตัวส่วนติดต่อจะส่งข้อมูลกลับมาให้คอมพิวเตอร์คราวละ 1 ไบท์ (โดยจะส่งมากี่ครั้งนั้นจะขึ้นอยู่กับคำสั่งในไบท์ที่ 4 ที่คอมพิวเตอร์ส่งไปให้กับส่วนติดต่อ) โดยข้อมูลที่จะส่งกลับมาให้คอมพิวเตอร์นั้น ส่วนติดต่อจะนำข้อมูลมาทำการเอ็กคลูซีฟออ (XOR) กับกุญแจ (Key) พิเศษที่มีขนาด 1 ไบท์ก่อนที่จะทำการส่งทุกครั้ง และเมื่อคอมพิวเตอร์ได้รับข้อมูลมาแล้วก็จะนำข้อมูลนั้นมาทำการเอ็กคลูซีฟออกับกุญแจอันเดิมอีกครั้งก่อนที่จะนำข้อมูลไปใช้ต่อไป ซึ่งการที่เรานำข้อมูลมาทำการเอ็กคลูซีฟออก่อนที่จะทำการส่งนั้นก็เพื่อเป็นการเพิ่มความปลอดภัยของข้อมูลจากการถูกโจรกรรมข้อมูล สำหรับรูปแสดงการติดต่อระหว่างคอมพิวเตอร์กับส่วนติดต่อกับคอมพิวเตอร์นั้นได้แสดงไว้ในรูปที่ 5-1

รูปที่ 5-1 การติดต่อระหว่างคอมพิวเตอร์กับส่วนที่ทำการติดต่อกับคอมพิวเตอร์ของฮาร์ดล็อค

5.2 โปรแกรมของผู้ผลิตซอฟต์แวร์

สำหรับในส่วนนี้จะเป็นการกล่าวถึงโปรแกรมของผู้ผลิตซอฟต์แวร์ ซึ่งในการออกแบบในโครงงานนี้ได้แบ่งออกเป็น 2 รูปแบบด้วยกัน ได้แก่

5.2.1 โปรแกรมของผู้ผลิตซอฟต์แวร์รูปแบบที่ 1

สำหรับในส่วนนี้จะเป็นส่วนของโปรแกรมของผู้ผลิตที่ใช้ในการเข้ารหัสโปรแกรมก่อนที่จะนำโปรแกรมนั้นไปจำหน่ายต่อไป ซึ่งในโปรแกรมนี้นั้นได้ออกแบบไว้ให้ตัวผู้ผลิตสามารถที่จะทำอะไรกับไฟล์ก็ได้ไม่ว่าจะเป็นการเข้ารหัสหรือถอดรหัสไฟล์ โดยในส่วนของไฟล์ที่จะนำมาทำการเข้ารหัสนั้น เราจะนำไฟล์ .exe มาทำการเข้ารหัส ซึ่งหลังจากการเข้ารหัสแล้วจะได้ไฟล์ใหม่ออกมา (แล้วแต่เราจะตั้งชื่อและนามสกุลของมัน) ซึ่งเป็นไฟล์ที่ไม่สามารถรันได้ สำหรับส่วนอินเตอร์เฟส (Interface) ของโปรแกรมของผู้ผลิตซอฟต์แวร์ในรูปแบบที่ 1 นั้นได้แสดงไว้ในรูปที่ 5-2

ในส่วนของการทำงานของโปรแกรมนั้น ในโปรแกรมเราจะมีตัวไทเมอร์ (Timer) ไว้คอยตรวจสอบว่ามีฮาร์ดล็อคต่ออยู่กับคอมพิวเตอร์หรือเปล่า (ซึ่งตัวไทเมอร์จะเป็นรูปแบบของลูปการทำงาน โดยเราสามารถที่จะกำหนดเวลาให้กับมันได้ว่าจะให้มันทำงานในทุก ๆ เวลาเท่าใด) ถ้าไม่มีฮาร์ดล็อคต่ออยู่แล้ว ปุ่ม Encode และปุ่ม Decode จะไม่สามารถใช้การได้ แต่ถ้าตรวจแล้วมีฮาร์ดล็อคต่ออยู่ ปุ่ม Encode และปุ่ม Decode ก็จะใช้การได้ โดยปุ่ม Encode นั้นจะมีไว้ใช้ในการเข้ารหัสไฟล์ .exe ส่วนปุ่ม Decode นั้นจะมีไว้ในการถอดรหัสไฟล์ที่ผ่านการเข้ารหัสมา ซึ่งในการตรวจสอบฮาร์ดล็อคของตัวไทเมอร์นั้น เมื่อโปรแกรมของผู้ผลิตเริ่มทำงาน คอมพิวเตอร์จะทำการติดต่อกับฮาร์ดล็อคโดยผ่านส่วนติดต่อกับ

คอมพิวเตอร์ของฮาร์ดล็อค โดยคอมพิวเตอร์จะใช้รูปแบบของคำสั่งขนาด 4 ไบท์ในการติดต่อ ดังที่ได้กล่าวมาแล้วในหัวข้อ 5.1 คือคอมพิวเตอร์จะส่งคำสั่งในไบท์ที่ 4 เป็นแบบที่ 5 ไปเพื่อให้ฮาร์ดล็อคส่งสัญญาณตอบสนองกลับมา

รูปที่ 5-2 โปรแกรมของผู้ผลิตซอฟต์แวร์ในรูปแบบที่ 1

สำหรับการทำงานหลัก ๆ ของโปรแกรมของผู้ผลิตนี้ ก็คือการเข้ารหัสไฟล์ .exe ก่อนที่จะนำโปรแกรมออกไปจำหน่าย ซึ่งในส่วนของการทำงานของการเข้ารหัสนั้น ในขั้นตอนแรกเราจะต้องทำการกำหนดและเลือกไฟล์ที่จะนำมาเข้ารหัส และกำหนดไฟล์ใหม่ที่จะได้มาหลังจากการเข้ารหัสแล้ว หลังจากนั้นเมื่อเรากดปุ่ม Encode แล้วโปรแกรมจะเริ่มทำการเข้ารหัสโดยจะเริ่มโหลดไฟล์ .exe เข้าไปไว้ในหน่วยความจำ เสร็จแล้วจะนำไฟล์มาเข้ารหัสจำนวน 64 บิท หรือ 8 ไบท์ (โดยจะทำครั้งละ 4 ไบท์) โดยในครั้งแรกโปรแกรมจะนำ 4 ไบท์แรกของไฟล์ .exe ไปทำการเข้ารหัสตามขบวนการเข้ารหัสแบบดีอีเอสก่อน หลังจากนั้นจะนำอีก 4 ไบท์ (ซึ่งได้จากการคำนวณตำแหน่งของไบท์ตามขนาดของไฟล์ตามอัลกอลิทึม) มาเข้ารหัสต่อไป โดยในส่วนของขบวนการเข้ารหัสนั้น คอมพิวเตอร์จะใช้รูปแบบของคำสั่งขนาด 4 ไบท์ในการติดต่อกับส่วนที่ใช้ติดต่อกับคอมพิวเตอร์ของฮาร์ดล็อค ซึ่งในขบวนการเข้ารหัสไฟล์ 4 ไบท์นั้น จะมีการส่งคำสั่งไปทั้งหมด 3 ครั้งด้วยกันในการเข้ารหัสแต่ละครั้ง โดยจะมีลำดับของคำสั่งในไบท์ที่ 4 ในแต่ละครั้งคือ แบบที่ 1 แบบที่ 3 และแบบที่ 2 ตามลำดับ เมื่อครบทั้ง 3 รอบก็จะถือว่าเป็นการเสร็จการเข้ารหัสไฟล์ขนาด 4 ไบท์

สำหรับในขบวนการถอดรหัสนั้นจะมีลักษณะการทำงานคล้ายกับขบวนการเข้ารหัส คือจะทำการถอดรหัสคราวละ 4 ไบท์เช่นเดียวกัน แต่จะต่างกันในส่วนของคำสั่งในไบท์ที่ 4 ในการติดต่อกันของคอมพิวเตอร์กับส่วนติดต่อกับคอมพิวเตอร์ของฮาร์ดล็อค โดยในขบวนการถอดรหัสนั้นจะทำการส่งคำสั่งไป 3 ครั้งเหมือนกัน แต่ลำดับของคำสั่งในไบท์ที่ 4 จะเป็น แบบที่ 1 แบบที่ 4 และแบบที่ 2 ตามลำดับ

5.2.2 โปรแกรมของผู้ผลิตซอฟต์แวร์รูปแบบที่ 2

สำหรับในส่วนนี้นั้น จริง ๆ แล้วมันจะไม่เชิงเป็นในรูปแบบโปรแกรมซะทีเดียว แต่มันจะเป็นรูปแบบของการที่ทางผู้ผลิตจะทำการเพิ่มโค้ดเข้าไปในซอสโค้ดของโปรแกรมที่จะนำไปจำหน่าย โดยโค้ดที่เราได้ทำการเพิ่มเข้าไปนั้น จะเป็นโค้ดที่ใช้ในการตรวจสอบว่ามีฮาร์ดล็อคต่ออยู่กับคอมพิวเตอร์หรือไม่ ซึ่งโค้ดที่เราเพิ่มเข้าไปนั้นจะไม่ไปยุ่งเกี่ยวกับการทำงานของโปรแกรม โดยในรายละเอียดของการเพิ่มโค้ดนั้นได้กล่าวไว้แล้วในบทที่ 2

5.3 โปรแกรมของผู้ใช้ซอฟต์แวร์

สำหรับในส่วนนี้นั้น จะเป็นส่วนของโปรแกรมของผู้ที่ซื้อซอฟต์แวร์ไปใช้งาน ซึ่งก็จะมีอยู่ 2 รูปแบบเหมือนกับในส่วนของผู้ผลิต ได้แก่

5.3.1 โปรแกรมของผู้ใช้ซอฟต์แวร์รูปแบบที่ 1

สำหรับในส่วนนี้จะเป็นโปรแกรมที่ผู้ซื้อซอฟต์แวร์ไปใช้งาน จะต้องทำการรันโปรแกรมนี้ก่อนที่เพื่อเรียกโปรแกรมที่ต้องการใช้งานขึ้นมา ซึ่งส่วนอินเตอร์เฟสของโปรแกรมของผู้ใช้งานซอฟต์แวร์ในรูปแบบที่ 1 นี้นั้นได้แสดงไว้ในรูปที่ 5-3

รูปที่ 5-3 โปรแกรมของผู้ใช้งานในรูปแบบที่ 1

ในส่วนของการทำงานของโปรแกรมนั้น โปรแกรมจะมีการตรวจสอบฮาร์ดล็อคโดยตัวไทเมอร์ว่ามีฮาร์ดล็อคต่ออยู่กับคอมพิวเตอร์หรือไม่ ถ้าไม่มีฮาร์ดล็อคต่ออยู่ก็จะไม่สามารถที่จะใช้งานโปรแกรมได้ ถ้ามีฮาร์ดล็อคต่ออยู่โปรแกรมจึงจะสามารถทำงานได้ โดยในระหว่างการใช้งานโปรแกรมอยู่นั้นเราจะไม่สามารถที่จะถอดฮาร์ดล็อคออกได้เลย สำหรับในการทำงานของโปรแกรมนั้น ในขั้นแรกเราจะต้องไปเรียกไฟล์ของโปรแกรมที่เราต้องการจะใช้งานมาก่อน (ซึ่งไฟล์นั้นจะไม่สามารถที่จะรันได้ เนื่องจากเป็น

ไฟล์ที่ถูกเข้ารหัสมาจากทางผู้ผลิต) จากนั้นเมื่อเราจะทำการใช้งานโปรแกรม เราจะต้องกดปุ่มรันจากโปรแกรมของผู้ใช้ซอฟต์แวร์ โดยเมื่อเราทำการกดปุ่มรันแล้ว โปรแกรมของผู้ใช้ซอฟต์แวร์จะเริ่มขบวนการถอดรหัสไฟล์ที่อยู่ในรูปไฟล์ที่ถูกเข้ารหัสให้เป็นไฟล์ที่สามารถรันได้ (โดยไฟล์นี้จะถูกสร้างขึ้นมาในรูปแบบที่ชั่วคราว คือเมื่อเลิกใช้งานโปรแกรมแล้ว ไฟล์นี้มันก็จะถูกลบไปโดยอัตโนมัติ) ซึ่งหลักการของขบวนการถอดรหัสนั้นได้กล่าวไว้แล้วในหัวข้อ 5.2.1 ในส่วนของโปรแกรมของผู้ผลิตรูปแบบที่ 1

5.3.2 โปรแกรมของผู้ใช้ซอฟต์แวร์รูปแบบที่ 2

สำหรับในส่วนนี้จะเป็นโปรแกรมที่จะนำมาใช้งานจริง ๆ โดยเป็นโปรแกรมที่ได้รับการเพิ่มโค้ดในการวนตรวจสอบฮาร์ดล็อคเข้าไปในโปรแกรมเรียบร้อยแล้ว ตามที่ได้กล่าวไว้ในหัวข้อ 5.2.2 ในส่วนของโปรแกรมของผู้ผลิตรูปแบบที่ 2 ซึ่งเราจะสามารถใช้งานโปรแกรมได้อย่างปกติ โดยที่ระหว่างการใช้งานอยู่นั้นก็จะมีการตรวจสอบด้วยว่ามีฮาร์ดล็อคต่ออยู่กับคอมพิวเตอร์หรือไม่ ถ้าเราถอดฮาร์ดล็อคออกโปรแกรมก็จะหยุดทำงานทันที ซึ่งหลักการทำงานในรูปแบบที่ 2 นี้จะมีความปลอดภัยในการใช้งานมากกว่าในรูปแบบที่ 1 ที่จะต้องมีการสร้างไฟล์ชั่วคราวขึ้นมา ซึ่งอาจจะเสี่ยงต่อการถูกโจรกรรมข้อมูลได้

บทที่ 6
การทดลองและผลการทดสอบ

ในบทนี้จะเป็นการทดลองในส่วนต่าง ๆ ของโครงงานที่ได้ออกแบบและทำไปแล้วได้แก่ ส่วนของฮาร์ดล็อคและโปรแกรมติดต่อกับผู้ใช้งานซึ่งในขั้นตอนนี้เพื่อให้สะดวกในการหาจุดบกพร่องเราจึงเริ่มทดสอบจากส่วนย่อยของชิ้นงานทีละส่วนก่อนแล้วจึงค่อยทดสอบในส่วนรวมต่อไป

6.1 การทดสอบโค้ดวีเอชดีแอล

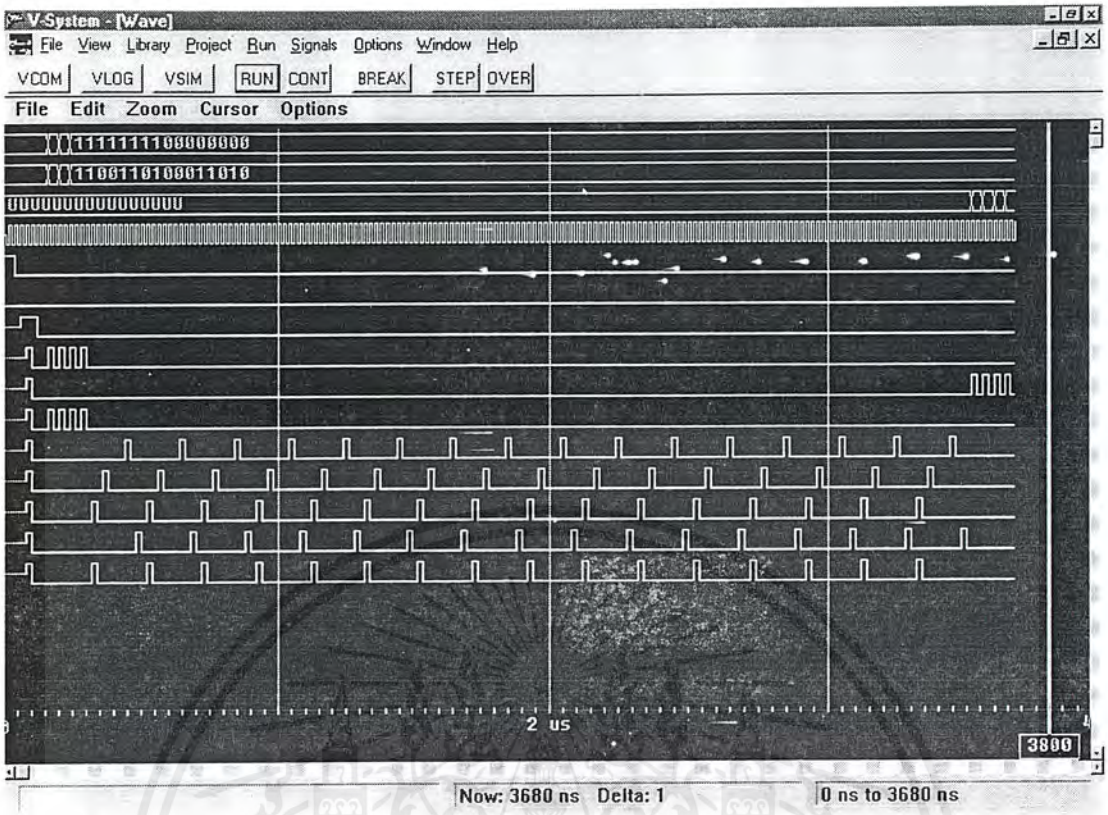
ในการทดสอบโค้ดวีเอชดีแอลที่บรรยายถึงการทำงานของส่วนเข้ารหัสและถอดรหัสแบบดีอีเอล เราได้ใช้โปรแกรมวีซิสเต็ม (V-System) ในการจำลองการทำงาน โดยการป้อนอินพุทข้อมูลเข้าไปดังนี้

- ข้อมูลเข้าบิทที่ 15-0 "1010101010101010"
- ข้อมูลเข้าบิทที่ 31-16 "1100110011001100"
- ข้อมูลเข้าบิทที่ 47-32 "1111000011110000"
- ข้อมูลเข้าบิทที่ 63-48 "1111111100000000"
- คีย์บิทที่ 15-0 "1100001100111100"
- คีย์บิทที่ 31-16 "1110011100011000"
- คีย์บิทที่ 47-32 "1111001111001111"
- คีย์บิทที่ 63-48 "1100110100011010"

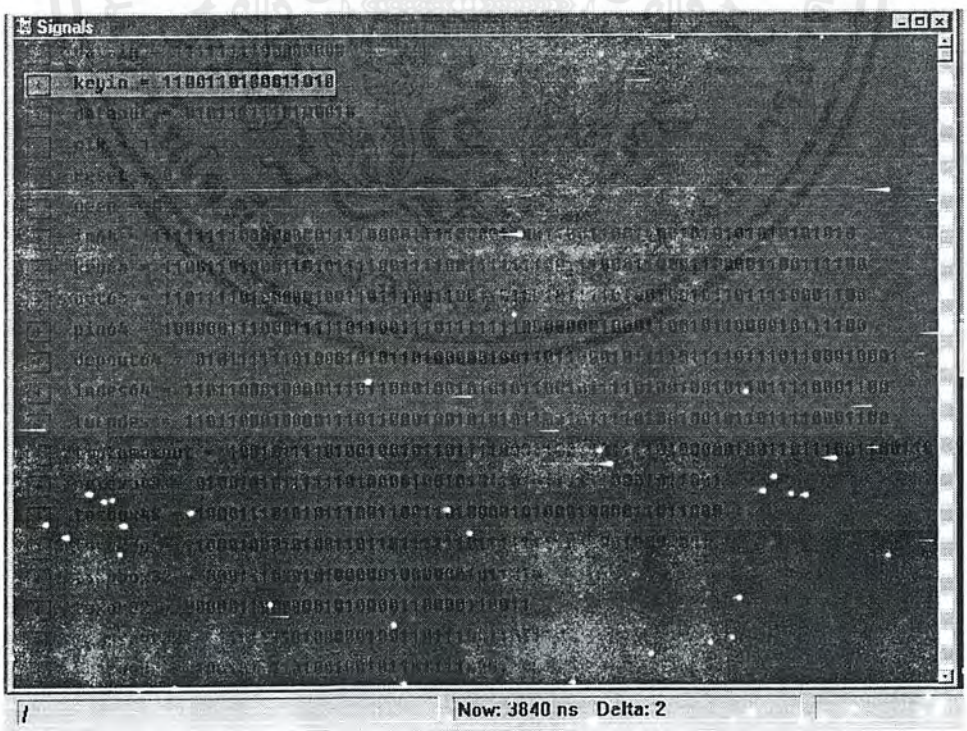
แล้วทำการเข้ารหัสโดยป้อนสัญญาณนาฬิกาให้เป็นจำนวน 181 ครั้งก็จะทำการเข้ารหัสเสร็จ การจำลองการทำงานของส่วนเข้ารหัสและถอดรหัสแสดงในรูปที่ 6-1 สัญญาณภายในส่วนต่าง ๆ ของฮาร์ดล็อคแสดงไว้ในรูปที่ 6-2 ผลลัพธ์ที่ได้ออกมามีดังนี้

- ผลลัพธ์บิทที่ 15-0 "1011101100010001"
- ผลลัพธ์บิทที่ 31-16 "0110001011110111"
- ผลลัพธ์บิทที่ 47-32 "1011010000010011"
- ผลลัพธ์บิทที่ 63-48 "0101111110100010"

และเมื่อเรานำผลที่ได้ไปถอดรหัสอีกครั้งหนึ่งโดยใช้คีย์เหมือนเดิมเราก็จะได้ข้อมูลที่ถอดรหัสออกมา จากนั้นก็เปรียบเทียบข้อมูลที่เราใส่ให้ในตอนต้นว่าตรงกับข้อมูลที่ถอดรหัสมาหรือไม่ ถ้าตรงกันแสดงว่าโค้ดวีเอชดีแอลที่เขียนนั้นทำงานได้ถูกต้องแล้วสามารถนำไปใช้งานได้



รูปที่ 6-1 การจำลองการทำงานของส่วนเข้ารหัสและถอดรหัส



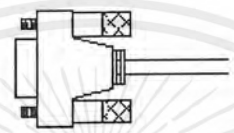
รูปที่ 6-2 สัญญาณภายในส่วนต่าง ๆ ของวงจรฮาร์ดล็อค

6.2 การทดสอบฮาร์ดล็อก

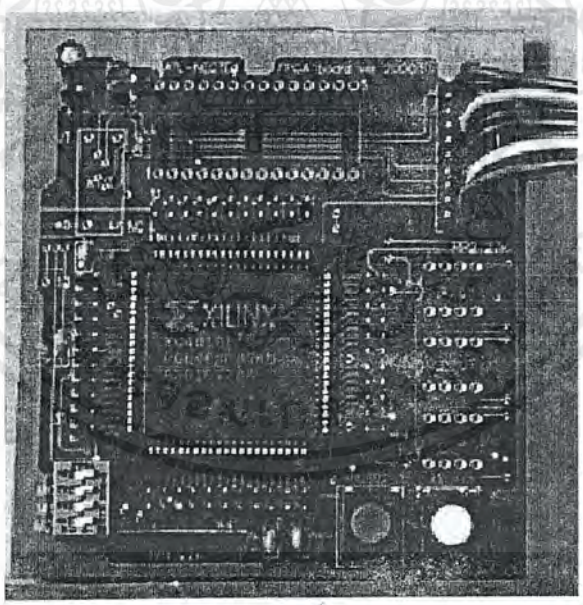
ในขั้นตอนนี้เราจะได้ฮาร์ดล็อกที่พร้อมใช้งานออกมาแล้วแต่ยังไม่มีโปรแกรมติดต่อกับผู้ใช้งาน ดังนั้นเราจำเป็นที่จะต้องใช้โปรแกรมไฮเปอร์เทอมินอล (Hyper Terminal) ในการส่งชุดคำสั่งและข้อมูลไปยังฮาร์ดล็อคแล้วดูว่ามีการตอบสนองเป็นไปตามที่ต้องการหรือไม่

6.2.1 การต่อฮาร์ดล็อคเข้ากับคอมพิวเตอร์

ในการเชื่อมต่อตัวฮาร์ดล็อคเข้ากับคอมพิวเตอร์นั้นเราใช้สายจำนวน 3 เส้นต่อออกจากชิปแมก 232 (MAX232) คือสายสัญญาณ GND, RxD และ TxD ต่อเข้ากับพอร์ตอนุกรมของคอมพิวเตอร์ ตัวเชื่อมต่อพอร์ตอนุกรมแสดงในรูป 6-3 และบอร์ดชิปเอฟพีจีเอที่ใช้พัฒนาแสดงในรูปที่ 6-4



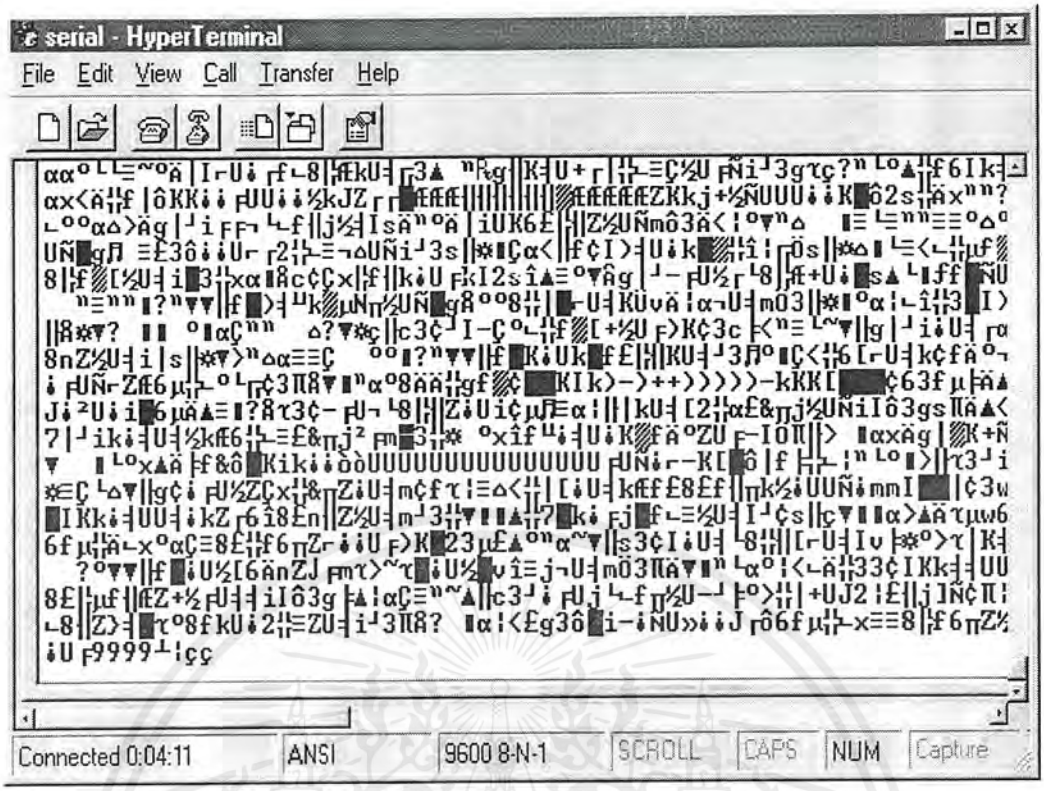
รูปที่ 6-3 ตัวเชื่อมต่อทางพอร์ตอนุกรม



รูปที่ 6-4 บอร์ดเอฟพีจีเอที่ใช้พัฒนาต้นแบบฮาร์ดล็อค

6.2.2 การทดสอบคำสั่งของฮาร์ดล็อก

ดังที่ได้กล่าวไปแล้วในบทที่ 5 ฮาร์ดล็อกที่ออกแบบนั้นมีคำสั่งอยู่ 5 คำสั่งซึ่งเราสามารถป้อนให้ได้โดยใช้โปรแกรมไฮเปอร์เทอมินอลดังแสดงในรูปที่ 6-5



รูปที่ 6-5 การใช้ไฮเปอร์เทอมินอลทดสอบคำสั่งของฮาร์ดล็อค

ในการทดลองเราควรจะออกแบบให้อินพุทข้อมูลอยู่ในช่วงของรหัสแอสกี้ที่สามารถแสดงผลออกมาได้เนื่องจากฮาร์ดล็อกสามารถตอบสนองข้อมูลได้ตั้งแต่ 0 จนถึง 255 ดังนั้นในการตอบสนองบางอย่างอาจไม่แสดงผลออกมาทางหน้าจอของโปรแกรม ดังนั้นเราจึงควรดัดแปลงโค้ดของตัวไมโครคอนโทรเลอร์ให้แสดงข้อมูลออกมาอยู่ในช่วงที่สามารถมองเห็นเป็นตัวอักษรได้ เมื่อเราได้ทำการทดลองส่งคำสั่งไป โดยในคำสั่งที่หนึ่งเป็นคำสั่งที่ให้ฮาร์ดล็อครับข้อมูลเข้าขนาด 32 บิทจากนั้นก็พิมพ์คีย์บอร์ดเป็นจำนวน 4 ครั้ง เพื่อเป็นการจำลองส่งข้อมูลไปให้ตัวฮาร์ดล็อค ต่อจากนั้นก็จึงส่งคำสั่งที่ 3 ไปเพื่อให้ ฮาร์ดล็อคทำการเข้ารหัสแต่ในการทดสอบนี้เราเพียงแต่ให้ฮาร์ดล็อคทำการสลับไบท์ของข้อมูลเท่านั้น จากนั้นก็ส่งคำสั่งที่ 2 ไปซึ่งเป็นคำสั่งที่ให้ฮาร์ดล็อคส่งค่า 32 บิทกลับมาเราจะเห็นอักษรที่เราพิมพ์ส่งไปให้ในตอนแรก 4 ตัวปรากฏในหน้าต่างโปรแกรม แต่จะมีการเรียงลำดับต่างกัน ซึ่งผลการทดลองปรากฏว่ามีการทำงานและตอบสองตามที่เราต้องการ ก็แสดงว่าฮาร์ดล็อคสามารถรับและจำแนกคำสั่งได้อย่างถูกต้องขั้นตอนต่อไปก็จะเป็นการนำโปรแกรมติดต่อกับผู้ใช้งานมาทดสอบกับฮาร์ดล็อก ซึ่งในตอนแรกก็ลองให้รับคำสั่งและปฏิบัติตามคำสั่ง โดยยังไม่มีการเข้ารหัสที่แท้จริงก่อนจากนั้นเมื่อแน่ใจว่ามีการรับคำสั่งและปฏิบัติคำสั่งได้แน่นอนไม่ผิดพลาดแล้วจึงค่อยให้มีการเข้ารหัสที่แท้จริง ซึ่งจะเป็นการทดสอบทุกส่วนประกอบร่วมกันอย่างสมบูรณ์

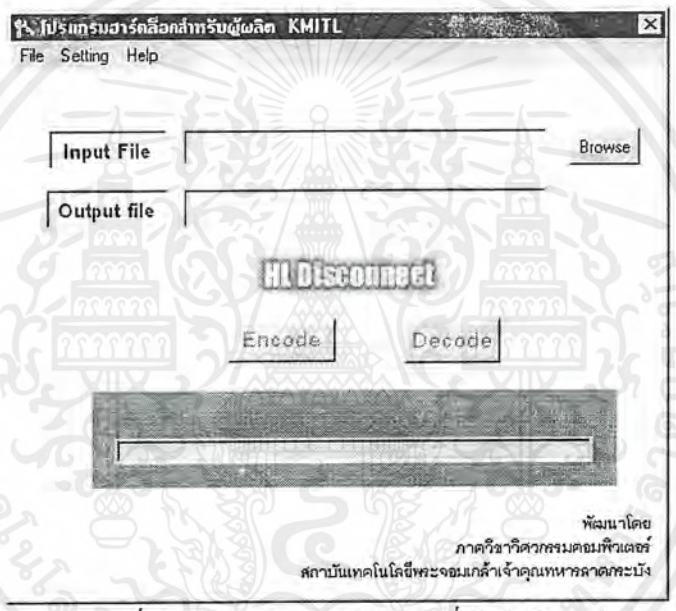
6.3 การทดสอบโปรแกรมติดต่อกับผู้ใช้งาน

สำหรับในส่วนของการทดสอบโปรแกรมที่ใช้ติดต่อกับผู้ใช้งานนั้น เราจะแบ่งการทดสอบเป็นทั้งโปรแกรมของผู้ผลิตและโปรแกรมของผู้ใช้งานซอฟต์แวร์ โดยเราจะทำการทดสอบการทำงานทั้ง 2 รูปแบบ ซึ่งจะได้ผล ดังต่อไปนี้

6.3.1 โปรแกรมของผู้ผลิต

6.3.1.1 โปรแกรมสำหรับผู้ผลิตรูปแบบที่ 1

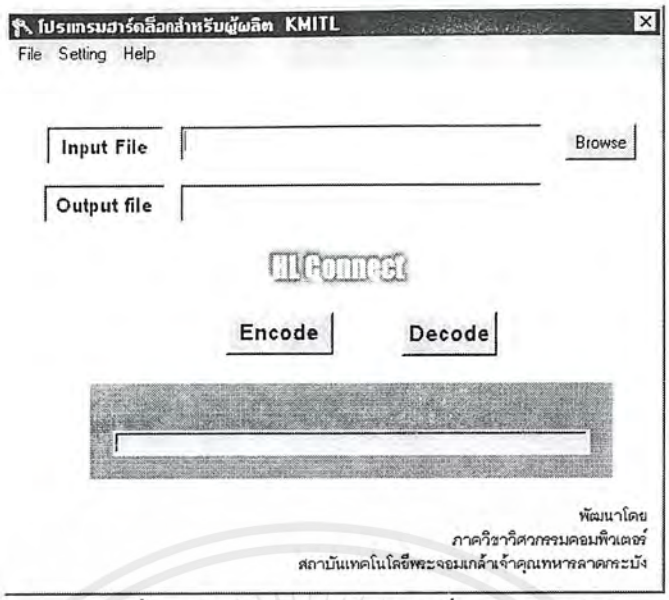
สำหรับการทดสอบโปรแกรมสำหรับผู้ผลิตในรูปแบบที่ 1 นั้นเราจะเริ่มจากการที่เรียกโปรแกรมขึ้นมาก่อนโดยที่ยังไม่ได้ต่อฮาร์ดล็อค ซึ่งจะทำให้ปุ่ม Encode และปุ่ม Decode ไม่สามารถใช้งานได้ ดังรูปที่ 6-6



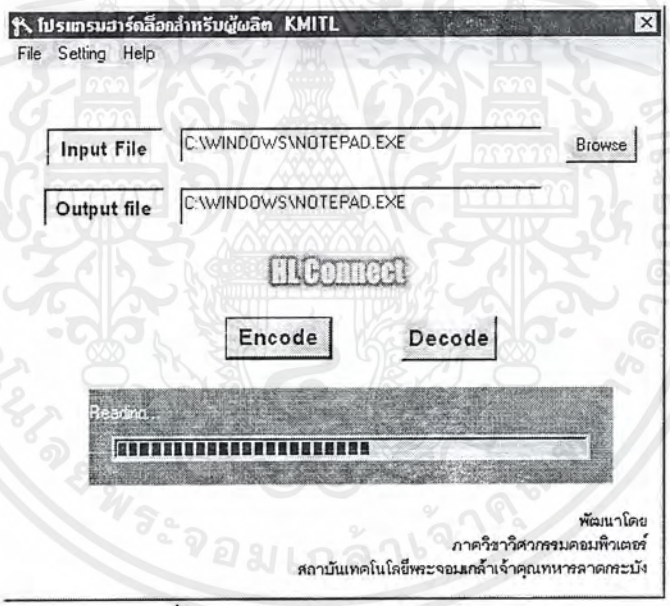
รูปที่ 6-6 โปรแกรมสำหรับผู้ผลิตเมื่อไม่ได้ต่อฮาร์ดล็อค

โดยที่เมื่อเราต้องการใช้งานโปรแกรม ก็ต้องทำการต่อฮาร์ดล็อคเข้าไป จึงจะทำให้ปุ่ม Encode และปุ่ม Decode สามารถใช้งานได้ ดังรูปที่ 6-7

หลังจากนั้นเราจะทดสอบโดยการนำไฟล์ .exe ของโปรแกรมมาทำการเข้ารหัสก่อนที่จะนำโปรแกรมนั้นออกจำหน่าย ซึ่งในโครงงานนี้เราได้ทดลองกับไฟล์ .exe หลายไฟล์ แต่ที่เรายกตัวอย่างมาแสดงเป็นโปรแกรม Notepad โดยเราจะนำไฟล์ Notepad.exe มาทำการเข้ารหัสและบันทึกทับไฟล์เดิมให้เป็นไฟล์ที่ผ่านการเข้ารหัสแล้ว ซึ่งจะไม่สามารถรันได้ ตามรูปที่ 6-8 และ 6-9



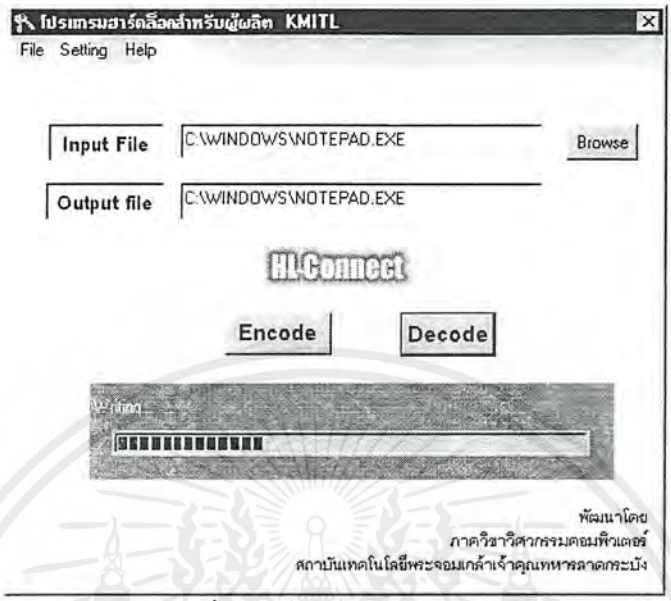
รูปที่ 6-7 โปรแกรมสำหรับผู้ผลิตที่มีการต่อฮาร์ดล็อค



รูปที่ 6-8 ขบวนการเข้ารหัสไฟล์ .exe

รูปที่ 6-9 ไฟล์ .exe ที่ผ่านขบวนการเข้ารหัสแล้วจะไม่สามารถรันได้

นอกจากขบวนการเข้ารหัสแล้ว โปรแกรมสำหรับผู้ผลิตก็สามารถจะทำการถอดรหัสเพื่อเรียกไฟล์ขึ้นมาทำงานได้ด้วย ซึ่งเราได้ทดสอบถอดรหัสไฟล์ Notepad.exe ที่เราได้ทำการเข้ารหัสไว้ ซึ่งเมื่อทำการถอดรหัสแล้วก็จะสามารถเรียกไฟล์ Notepad.exe ขึ้นมาใช้งานได้ ดังรูปที่ 6-10 และ 6-11



รูปที่ 6-10 ขบวนการถอดรหัสไฟล์ .exe

รูปที่ 6-11 ไฟล์ .exe ที่ผ่านขบวนการถอดรหัสแล้วจะสามารถรันได้

6.3.1.2 โปรแกรมสำหรับผู้ผลิตรูปแบบที่ 2

สำหรับโปรแกรมสำหรับผู้ผลิตในแบบที่ 2 นี้ จริง ๆ แล้วมันไม่ใช่โปรแกรมแต่จะเป็นการเพิ่มโค้ดเข้าไปในซอสโค้ดหลักของโปรแกรมที่เราจะทำการนำออกจำหน่าย เพื่อคอยทำการตรวจสอบว่ามีฮาร์ดล็อคต่ออยู่กับคอมพิวเตอร์หรือไม่ ซึ่งถ้าพบว่าไม่ได้มีการต่อฮาร์ดล็อคอยู่แล้ว โปรแกรมก็จะทำการปิดตัวเองลงทันที โดยโค้ดที่เพิ่มเข้าไปนั้นจะไม่ไปกระทบกับการทำงานหลักของโปรแกรมนั้น ๆ

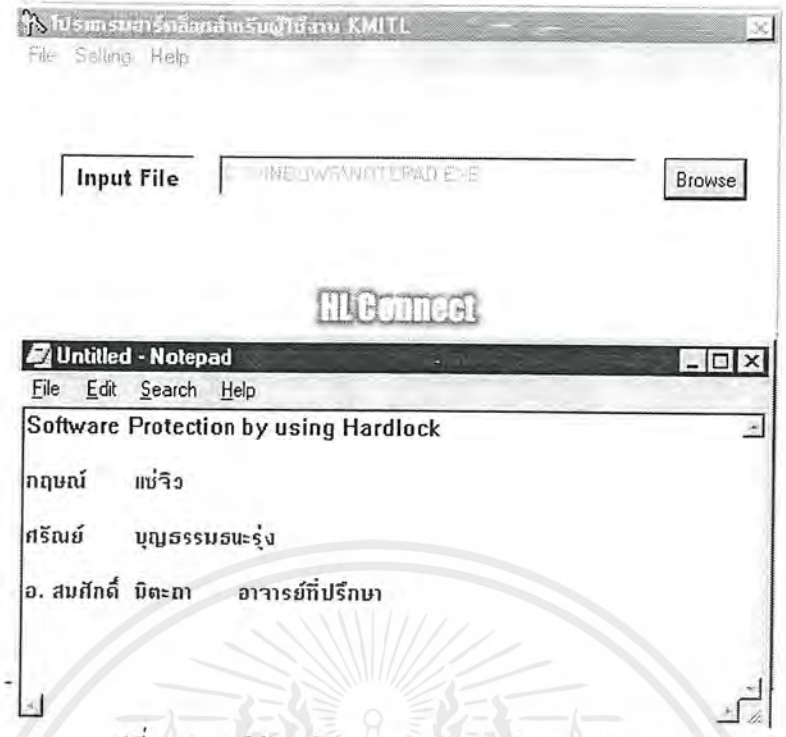
6.3.2 โปรแกรมสำหรับผู้ใช้ซอฟต์แวร์

6.3.2.1 โปรแกรมสำหรับผู้ใช้ซอฟต์แวร์รูปแบบที่ 1

สำหรับการทดสอบโปรแกรมสำหรับผู้ใช้ซอฟต์แวร์ในรูปแบบที่ 1 นี้ จะเริ่มโดยการที่เราจะต้องเรียกโปรแกรมสำหรับผู้ใช้ซอฟต์แวร์ขึ้นมาก่อน เพื่อที่จะใช้ในการรันซอฟต์แวร์ที่เราต้องการใช้งานต่อไป หลังจากนั้นจะต้องเรียกโปรแกรมที่จะใช้งานขึ้นมา โดยจะเรียกไฟล์ขึ้นมา ซึ่งเป็นไฟล์ที่อยู่ในรูปของไฟล์ที่เข้ารหัสและไม่สามารถใช้งานได้ (ในที่นี้จะนำไฟล์ Notepad.exe ที่ผ่านการเข้ารหัสมาทดสอบ) โดยเมื่อเรากดปุ่ม Run แล้วโปรแกรมของผู้ใช้ซอฟต์แวร์จะทำการถอดรหัสไฟล์ที่อยู่ในรูปไฟล์ที่เข้ารหัสให้เป็นไฟล์ที่สามารถรันได้ ดังรูปที่ 6-12

ซึ่งหลังจากที่โปรแกรมได้ผ่านขบวนการถอดรหัสไฟล์เสร็จแล้ว โปรแกรมของผู้ใช้ซอฟต์แวร์ก็จะทำการสร้างไฟล์ที่สามารถรันได้ขึ้นมา แล้วจึงนำไฟล์นั้นรันขึ้นมาใช้งานเลย ดังรูปที่ 6-13 โดยไฟล์ที่ถูกนำมาใช้งานนั้นจะอยู่ในรูปของไฟล์ชั่วคราว ซึ่งเมื่อเราเลิกใช้งานโปรแกรมแล้วไฟล์ที่ใช้งานได้นี้ก็จะถูกลบออกไป และในระหว่างที่ใช้งานโปรแกรมอยู่นั้น ถ้าหากมีการถอดฮาร์ดล็อคออกไปแล้วโปรแกรมก็จะไม่สามารถใช้งานได้

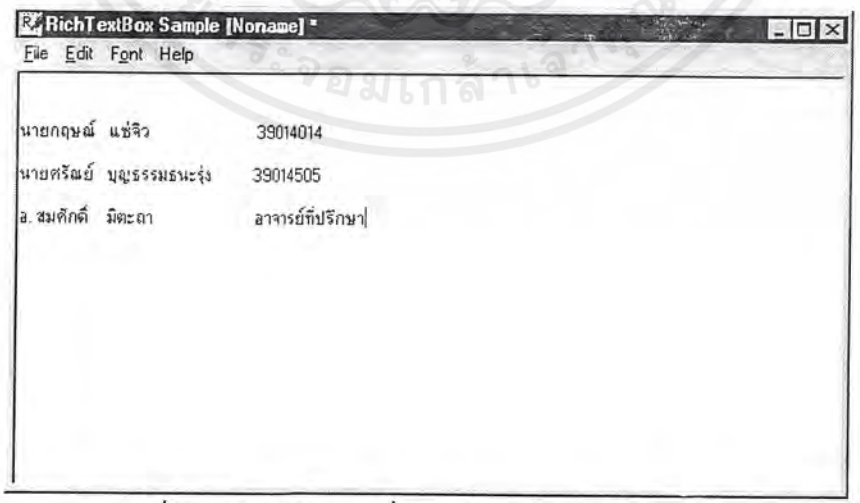
รูปที่ 6-12 การรันไฟล์ที่อยู่ในรูปไฟล์ที่เข้ารหัส โดยโปรแกรมจะทำการถอดรหัสไฟล์นั้นก่อน



รูปที่ 6-13 การใช้งานโปรแกรมหลังจากผ่านขบวนการถอดรหัส

6.3.2.2 โปรแกรมสำหรับผู้ใช้ซอฟต์แวร์รูปแบบที่ 2

สำหรับโปรแกรมสำหรับผู้ใช้งานซอฟต์แวร์ในรูปแบบที่ 2 นี้ จะเป็นโปรแกรมที่สามารถใช้งานได้เลย เพียงแต่ว่าในตัวซอสโค้ดจริง ๆ ของโปรแกรมนั้นจะมีการถูกเพิ่มโค้ดในการตรวจสอบฮาร์ดล็อคเข้าไปด้วย โดยถ้าตรวจพบว่าไม่ได้มีการต่อฮาร์ดล็อคอยู่แล้ว โปรแกรมก็จะไม่สามารถใช้งานได้ ซึ่งในโครงงานนี้เราได้ทดลองวิธีการเพิ่มโค้ดให้โปรแกรมหลายโปรแกรมด้วยกัน แต่จะนำโปรแกรมด้าน Text Editor ที่ชื่อ RichTextBox Sample มาเป็นตัวอย่าง ตามรูปที่ 6-14 จะเป็นการเรียกโปรแกรมขึ้นมาใช้งานธรรมดา แต่เมื่อเราถอดฮาร์ดล็อคออกแล้ว โปรแกรมก็จะไม่สามารถทำงานได้ ดังรูปที่ 6-15



รูปที่ 6-14 เรียกโปรแกรมที่มีการต่อฮาร์ดล็อคมาใช้งานได้ตามปกติ

รูปที่ 6-15 โปรแกรมจะไม่สามารถทำงานต่อไปได้ถ้าฮอดฮาร์ดล็อคออก

บทที่ 7
บทสรุปและวิจารณ์

สำหรับในปริญญานิพนธ์ฉบับนี้มีวัตถุประสงค์ในการสร้างอุปกรณ์ทางฮาร์ดแวร์ที่ใช้ในการป้องกันการคัดลอกซอฟต์แวร์ ซึ่งเรียกอุปกรณ์นี้ว่า ฮาร์ดล็อค โดยอุปกรณ์ฮาร์ดล็อคที่ได้พัฒนาขึ้นมาเป็นต้นแบบนั้น ใช้เทคโนโลยีเอฟพีจีเอ (FPGA : Field Programmable Gate Array) มาทำในส่วนของ วงจรการเข้ารหัสและถอดรหัสข้อมูล ซึ่งหลักการของการเข้ารหัสและถอดรหัสข้อมูลนั้นจะใช้ทฤษฎีการเข้ารหัสแบบดีอีเอส (DES : Data Encryption Standard) และการใช้งานฮาร์ดล็อคนั้นเราจะทำการเชื่อมต่อเข้ากับคอมพิวเตอร์ทางพอร์ตอนุกรม (Serial Port) ของคอมพิวเตอร์

อุปกรณ์ฮาร์ดล็อคที่ได้ออกแบบไว้ในปริญญานิพนธ์ฉบับนี้ เราได้ออกแบบไว้ 2 รูปแบบด้วยกัน โดยรูปแบบแรกจะสามารถนำไปประยุกต์ใช้ได้กับทุกโปรแกรม แต่อาจจะยังมีข้อบกพร่องในเรื่องของความปลอดภัยในการถูกโจรกรรมข้อมูลอยู่ ส่วนในแบบที่ 2 จะมีความปลอดภัยจากการถูกโจรกรรมข้อมูลค่อนข้างมาก แต่จะนำไปประยุกต์ใช้ได้เฉพาะกับโปรแกรมที่มีซอสโค้ดของโปรแกรมมาให้เท่านั้น

อุปกรณ์ฮาร์ดล็อคที่ได้ออกแบบไว้ทั้ง 2 รูปแบบนั้น จากการทดสอบการทำงานพบว่าสามารถที่จะทำงานได้อย่างถูกต้องและครบถ้วนตามหน้าที่ที่เราได้ออกแบบเอาไว้ในตอนต้น แต่ในการสร้างฮาร์ดล็อคในปริญญานิพนธ์ฉบับนี้ ยังไม่บรรลุวัตถุประสงค์ที่ได้ตั้งไว้ในตอนแรกอย่างสมบูรณ์ เนื่องจากในตอนแรก เราคิดว่าจะใช้การเชื่อมต่อฮาร์ดล็อคกับคอมพิวเตอร์ทางพอร์ต USB (Universal Serial Bus) แต่เนื่องจากเรื่องเกี่ยวกับมาตรฐาน USB นั้นยังเป็นเรื่องที่ใหม่อยู่ ทำให้เกิดปัญหาในการจัดหาและนำอุปกรณ์เกี่ยวกับพอร์ต USB มาใช้งาน จึงได้เปลี่ยนการทดลองมาใช้พอร์ตอนุกรมแทน และในการออกแบบวงจรการเข้ารหัสนั้น ซึ่งมาตรฐานการเข้ารหัสควรจะเป็นการเข้ารหัสข้อมูลจำนวน 64 บิท แต่ในปริญญานิพนธ์นี้ได้ใช้การเข้ารหัสข้อมูลจำนวน 32 บิทแทน เนื่องจากอุปกรณ์เอฟพีจีเอที่นำมาใช้นั้นมีความจุในการใช้งานไม่เพียงพอ ซึ่งจะเป็นผลให้มีความปลอดภัยลดลงได้

มีข้อควรคำนึงถึงอีกประการหนึ่งคือ ฮาร์ดล็อคที่เราทำขึ้นมานี้ไม่ได้เน้นเรื่องความปลอดภัยในการส่งข้อมูลระหว่างฮาร์ดล็อคกับคอมพิวเตอร์เท่าที่ควร เราใช้การป้องกันโดยมีคีย์ในการส่งและคีย์ในการรับเท่านั้นซึ่งยังสามารถแฮคได้ง่ายอยู่ ดังนั้นอาจจะเป็นจุดอ่อนของฮาร์ดล็อคได้ แต่สามารถจะแก้ไขได้โดยหาวิธีส่งข้อมูลระหว่างฮาร์ดล็อคและคอมพิวเตอร์แบบใหม่ที่มีความปลอดภัยมากกว่าเช่น อาจจะประยุกต์ใช้โปรโตคอลที่ใช้ส่งข้อมูลกันระหว่างเครือข่าย (network) อยู่ในขณะนี้ซึ่งจะมีความปลอดภัยสูง

สำหรับอุปกรณ์ฮาร์ดล็อคในปริญญานิพนธ์ฉบับนี้ยังเป็นเพียงอุปกรณ์ต้นแบบอยู่ โดยผู้ที่มีความสนใจเทคโนโลยีทางด้านนี้ ก็สามารถที่จะนำฮาร์ดล็อคที่ทางเราได้ออกแบบไว้ไปพัฒนาต่อไปได้ให้สามารถใช้งานได้สะดวกและมีประสิทธิภาพมากยิ่งขึ้น เช่น ลดขนาดของตัวฮาร์ดล็อคให้เล็กลง พัฒนาให้ฮาร์ดล็อคสามารถใช้งานได้โดยไม่ต้องอาศัยไฟเลี้ยงจากภายนอก พัฒนาการเชื่อมต่อฮาร์ดล็อคกับคอมพิวเตอร์ให้สามารถเชื่อมต่อผ่านทางพอร์ต USB พัฒนาวงจรการเข้ารหัสข้อมูลให้เป็นการเข้ารหัสข้อมูลจำนวน 64 บิท เพื่อเพิ่มความปลอดภัยมากยิ่งขึ้น เป็นต้น

บรรณานุกรม

[1] Jennifer Seberry and Josef Pieprzyk : *"Cryptography An Introduction to Computer Security"*,Prentice Hall

[2] Compaq, Intel, Microsoft, NEC : *"Universal Serial Bus Specification"*,Compaq, Intel, Microsoft, NEC Revision 1.1 September 23, 1998

[3] Craig Peacock : *"Interfacing the Serial/RS232 Port V5.0"*,Craig Peacock http://www.senet.com.au/~cpeacock 1998

[4] XILINX : *"ONLINE HDL Synthesis for FPGAs Design Guide"*,Xilinx Inc 1995

[5] Ayala, K.J. : *"The 8051 Microcontroller Architecture, Programing and Applications"*,West Publishing Company 1991

[6] วัชรากร หนูทอง : *"คู่มือการใช้โปรแกรม Xilinx Foundation Series 2.1i ในการ download วงจรลงสู่อุปกรณ์ FPGA และการใช้งาน XILINX84 Demo Board"*,ฝ่ายออกแบบวงจรรวม ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ (TMEC)

[7] โอภาส สุวิกะปกรณ์กุล : *"การออกแบบวงจรป้องกันการคัดลอกซอฟต์แวร์โดยใช้เอฟพีจีเอ"*, ภาควิชาวิศวกรรมคอมพิวเตอร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 1998

[8] ธีรศักดิ์ชัย อั่งสกุล : *"การออกแบบวงจรรวมเฉพาะกิจเพื่อป้องกันการเข้าถึงข้อมูลในหน่วยความจำชนิด EEPROM"*, ภาควิชาวิศวกรรมคอมพิวเตอร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 1995

[9] สุนทร วิทูสุรพจน์ : *"ไมโครคอนโทรลเลอร์ตระกูล 8051"*,บริษัท ซีเอ็ดยูเคชั่น จำกัด(มหาชน)

ภาคผนวก ก.

ข้อมูลอุปกรณ์ FPGA ตระกูล XC4000

XILINX®

XC4000E and XC4000X Series Field Programmable Gate Arrays

May 14, 1999 (Version 1.6)

Product Specification

XC4000E and XC4000X Series Features

Note: Information in this data sheet covers the XC4000E, XC4000EX, and XC4000XL families. A separate data sheet covers the XC4000XLA and XC4000XV families. Electrical Specifications and package/pin information are covered in separate sections for each family to make the information easier to access, review, and print. For access to these sections, see the Xilinx WEBLINX web site at

http://www.xilinx.com/partinfo/databook.htm#xc4000.

- System featured Field-Programmable Gate Arrays
  - Select-RAM™ memory: on-chip ultra-fast RAM with
    - synchronous write option
    - dual-port RAM option
  - Fully PCI compliant (speed grades -2 and faster)
  - Abundant flip-flops
  - Flexible function generators
  - Dedicated high-speed carry logic
  - Wide edge decoders on each edge
  - Hierarchy of interconnect lines
  - Internal 3-state bus capability
  - Eight global low-skew clock or signal distribution networks
- System Performance beyond 80 MHz
- Flexible Array Architecture
- Low Power Segmented Routing Architecture
- Systems-Oriented Features
  - IEEE 1149.1-compatible boundary scan logic support
  - Individually programmable output slew rate
  - Programmable input pull-up or pull-down resistors
  - 12 mA sink current per XC4000E output
- Configured by Loading Binary File
  - Unlimited re-programmability
- Read Back Capability
  - Program verification
  - Internal node observability
- Backward Compatible with XC4000 Devices
- Development System runs on most common computer platforms
  - Interfaces to popular design environments
  - Fully automatic mapping, placement and routing
  - Interactive design editor for design optimization

Low-Voltage Versions Available

- Low-Voltage Devices Function at 3.0 - 3.6 Volts
- XC4000XL: High Performance Low-Voltage Versions of XC4000EX devices

Additional XC4000X Series Features

- Highest Performance — 3.3 V XC4000XL
- Highest Capacity — Over 180,000 Usable Gates
- 5 V tolerant I/Os on XC4000XL
- 0.35 µm SRAM process for XC4000XL
- Additional Routing Over XC4000E
  - almost twice the routing capacity for high-density designs
- Buffered Interconnect for Maximum Speed Blocks
- Improved VersaRing™ I/O Interconnect for Better Fixed Pinout Flexibility
- 12 mA Sink Current Per XC4000X Output
- Flexible New High-Speed Clock Network
  - Eight additional Early Buffers for shorter clock delays
  - Virtually unlimited number of clock signals
- Optional Multiplexer or 2-input Function Generator on Device Outputs
- Four Additional Address Bits in Master Parallel Configuration Mode
- XC4000XV Family offers the highest density with 0.25 µm 2.5 V technology

Introduction

XC4000 Series high-performance, high-capacity Field Programmable Gate Arrays (FPGAs) provide the benefits of custom CMOS VLSI, while avoiding the initial cost, long development cycle, and inherent risk of a conventional masked gate array.

The result of thirteen years of FPGA design experience and feedback from thousands of customers, these FPGAs combine architectural versatility, on-chip Select-RAM memory with edge-triggered and dual-port modes, increased speed, abundant routing resources, and new, sophisticated software to achieve fully automated implementation of complex, high-density, high-performance designs.

The XC4000E and XC4000X Series currently have 20 members, as shown in Table 1.

Table 1: XC4000E and XC4000X Series Field Programmable Gate Arrays

Device	Logic Cells	Max Logic Gates (No RAM)	Max. RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total CLBs	Number of Flip-Flops	Max. User I/O
XC4002XL	152	1,600	2,048	1,000 - 3,000	8 x 8	64	256	64
XC4003E	238	3,000	3,200	2,000 - 5,000	10 x 10	100	360	80
XC4005E/XL	466	5,000	6,272	3,000 - 9,000	14 x 14	196	616	112
XC4006E	608	6,000	8,192	4,000 - 12,000	16 x 16	256	768	128
XC4008E	770	8,000	10,368	6,000 - 15,000	18 x 18	324	936	144
XC4010E/XL	950	10,000	12,800	7,000 - 20,000	20 x 20	400	1,120	160
XC4013E/XL	1368	13,000	18,432	10,000 - 30,000	24 x 24	576	1,536	192
XC4020E/XL	1862	20,000	25,088	13,000 - 40,000	28 x 28	784	2,016	224
XC4025E	2432	25,000	32,768	15,000 - 45,000	32 x 32	1,024	2,560	256
XC4028EX/XL	2432	28,000	32,768	18,000 - 50,000	32 x 32	1,024	2,560	256
XC4036EX/XL	3078	36,000	41,472	22,000 - 65,000	36 x 36	1,296	3,168	288
XC4044XL	3800	44,000	51,200	27,000 - 80,000	40 x 40	1,600	3,840	320
XC4052XL	4598	52,000	61,952	33,000 - 100,000	44 x 44	1,936	4,576	352
XC4062XL	5472	62,000	73,728	40,000 - 130,000	48 x 48	2,304	5,376	384
XC4085XL	7448	85,000	100,352	55,000 - 180,000	56 x 56	3,136	7,168	448

* Max values of Typical Gate Range include 20-30% of CLBs used as RAM.

Note: *All functionality in low-voltage families is the same as in the corresponding 5-Volt family, except where numerical references are made to timing or power.*

Description

XC4000 Series devices are implemented with a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), interconnected by a powerful hierarchy of versatile routing resources, and surrounded by a perimeter of programmable Input/Output Blocks (IOBs). They have generous routing resources to accommodate the most complex interconnect patterns.

The devices are customized by loading configuration data into internal memory cells. The FPGA can either actively read its configuration data from an external serial or byte-parallel PROM (master modes), or the configuration data can be written into the FPGA from an external device (slave and peripheral modes).

XC4000 Series FPGAs are supported by powerful and sophisticated software, covering every aspect of design from schematic or behavioral entry, floor planning, simulation, automatic block placement and routing of interconnects, to the creation, downloading, and readback of the configuration bit stream.

Because Xilinx FPGAs can be reprogrammed an unlimited number of times, they can be used in innovative designs where hardware is changed dynamically, or where hardware must be adapted to different user applications. FPGAs are ideal for shortening design and development cycles, and also offer a cost-effective solution for production rates well beyond 5,000 systems per month. For lowest high-volume unit cost, a design can first be implemented in the XC4000E or XC4000X, then migrated to one of Xilinx' compatible HardWire mask-programmed devices.

Taking Advantage of Re-configuration

FPGA devices can be re-configured to change logic function while resident in the system. This capability gives the system designer a new degree of freedom not available with any other type of logic.

Hardware can be changed as easily as software. Design updates or modifications are easy, and can be made to products already in the field. An FPGA can even be re-configured dynamically to perform different functions at different times.

Re-configurable logic can be used to implement system self-diagnostics, create systems capable of being re-configured for different environments or operations, or implement multi-purpose hardware for a given application. As an added benefit, using re-configurable FPGA devices simplifies hardware design and debugging and shortens product time-to-market.

Table 16: Pin Descriptions

Pin Name	I/O During Config.	I/O After Config.	Pin Description
Permanently Dedicated Pins			
VCC	I	I	Eight or more (depending on package) connections to the nominal +5 V supply voltage (+3.3 V for low-voltage devices). All must be connected, and each must be decoupled with a 0.01 - 0.1 µF capacitor to Ground.
GND	I	I	Eight or more (depending on package type) connections to Ground. All must be connected.
CCLK	I or O	I	During configuration, Configuration Clock (CCLK) is an output in Master modes or Asynchronous Peripheral mode, but is an input in Slave mode and Synchronous Peripheral mode. After configuration, CCLK has a weak pull-up resistor and can be selected as the Readback Clock. There is no CCLK High or Low time restriction on XC4000 Series devices, except during Readback. See "Violating the Maximum High and Low Time Specification for the Readback Clock" on page 56 for an explanation of this exception.
DONE	I/O	O	DONE is a bidirectional signal with an optional internal pull-up resistor. As an output, it indicates the completion of the configuration process. As an input, a Low level on DONE can be configured to delay the global logic initialization and the enabling of outputs. The optional pull-up resistor is selected as an option in the XACT*step* program that creates the configuration bitstream. The resistor is included by default.
$\overline{\text{PROGRAM}}$	I	I	$\overline{\text{PROGRAM}}$ is an active Low input that forces the FPGA to clear its configuration memory. It is used to initiate a configuration cycle. When $\overline{\text{PROGRAM}}$ goes High, the FPGA finishes the current clear cycle and executes another complete clear cycle, before it goes into a WAIT state and releases $\overline{\text{INIT}}$. The $\overline{\text{PROGRAM}}$ pin has a permanent weak pull-up, so it need not be externally pulled up to Vcc.
User I/O Pins That Can Have Special Functions			
RDY/$\overline{\text{BUSY}}$	O	I/O	During Peripheral mode configuration, this pin indicates when it is appropriate to write another byte of data into the FPGA. The same status is also available on D7 in Asynchronous Peripheral mode, if a read operation is performed when the device is selected. After configuration, RDY/$\overline{\text{BUSY}}$ is a user-programmable I/O pin. RDY/$\overline{\text{BUSY}}$ is pulled High with a high-impedance pull-up prior to $\overline{\text{INIT}}$ going High.
$\overline{\text{RCLK}}$	O	I/O	During Master Parallel configuration, each change on the A0-A17 outputs (A0 - A21 for XC4000X) is preceded by a rising edge on $\overline{\text{RCLK}}$, a redundant output signal. $\overline{\text{RCLK}}$ is useful for clocked PROMs. It is rarely used during configuration. After configuration, $\overline{\text{RCLK}}$ is a user-programmable I/O pin.
M0, M1, M2	I	I (M0), O (M1), I (M2)	As Mode inputs, these pins are sampled after $\overline{\text{INIT}}$ goes High to determine the configuration mode to be used. After configuration, M0 and M2 can be used as inputs, and M1 can be used as a 3-state output. These three pins have no associated input or output registers. During configuration, these pins have weak pull-up resistors. For the most popular configuration mode, Slave Serial, the mode pins can thus be left unconnected. The three mode inputs can be individually configured with or without weak pull-up or pull-down resistors. A pull-down resistor value of 4.7 kΩ is recommended. These pins can only be used as inputs or outputs when called out by special schematic definitions. To use these pins, place the library components MD0, MD1, and MD2 instead of the usual pad symbols. Input or output buffers must still be used.
TDO	O	O	If boundary scan is used, this pin is the Test Data Output. If boundary scan is not used, this pin is a 3-state output without a register, after configuration is completed. This pin can be user output only when called out by special schematic definitions. To use this pin, place the library component TDO instead of the usual pad symbol. An output buffer must still be used.

Table 16: Pin Descriptions (Continued)

Pin Name	I/O During Config.	I/O After Config.	Pin Description
TDI, TCK, TMS	I	I/O or I (JTAG)	If boundary scan is used, these pins are Test Data In, Test Clock, and Test Mode Select inputs respectively. They come directly from the pads, bypassing the IOBs. These pins can also be used as inputs to the CLB logic after configuration is completed. If the BSCAN symbol is not placed in the design, all boundary scan functions are inhibited once configuration is completed, and these pins become user-programmable I/O. In this case, they must be called out by special schematic definitions. To use these pins, place the library components TDI, TCK, and TMS instead of the usual pad symbols. Input or output buffers must still be used.
HDC	O	I/O	High During Configuration (HDC) is driven High until the I/O go active. It is available as a control output indicating that configuration is not yet completed. After configuration, HDC is a user-programmable I/O pin.
$\overline{\text{LDC}}$	O	I/O	Low During Configuration ($\overline{\text{LDC}}$) is driven Low until the I/O go active. It is available as a control output indicating that configuration is not yet completed. After configuration, $\overline{\text{LDC}}$ is a user-programmable I/O pin.
$\overline{\text{INIT}}$	I/O	I/O	Before and during configuration, $\overline{\text{INIT}}$ is a bidirectional signal. A 1 kΩ - 10 kΩ external pull-up resistor is recommended. As an active-Low open-drain output, $\overline{\text{INIT}}$ is held Low during the power stabilization and internal clearing of the configuration memory. As an active-Low input, it can be used to hold the FPGA in the internal WAIT state before the start of configuration. Master mode devices stay in a WAIT state an additional 30 to 300 µs after $\overline{\text{INIT}}$ has gone High. During configuration, a Low on this output indicates that a configuration data error has occurred. After the I/O go active, $\overline{\text{INIT}}$ is a user-programmable I/O pin.
PGCK1 - PGCK4 (XC4000E only)	Weak Pull-up	I or I/O	Four Primary Global inputs each drive a dedicated internal global net with short delay and minimal skew. If not used to drive a global buffer, any of these pins is a user-programmable I/O. The PGCK1-PGCK4 pins drive the four Primary Global Buffers. Any input pad symbol connected directly to the input of a BUFGP symbol is automatically placed on one of these pins.
SGCK1 - SGCK4 (XC4000E only)	Weak Pull-up	I or I/O	Four Secondary Global inputs each drive a dedicated internal global net with short delay and minimal skew. These internal global nets can also be driven from internal logic. If not used to drive a global net, any of these pins is a user-programmable I/O pin. The SGCK1-SGCK4 pins provide the shortest path to the four Secondary Global Buffers. Any input pad symbol connected directly to the input of a BUFGS symbol is automatically placed on one of these pins.
GCK1 - GCK8 (XC4000X only)	Weak Pull-up	I or I/O	Eight inputs can each drive a Global Low-Skew buffer. In addition, each can drive a Global Early buffer. Each pair of global buffers can also be driven from internal logic, but must share an input signal. If not used to drive a global buffer, any of these pins is a user-programmable I/O. Any input pad symbol connected directly to the input of a BUFGLS or BUFGE symbol is automatically placed on one of these pins.
FCLK1 - FCLK4 (XC4000XLA and XC4000XV only)	Weak Pull-up	I or I/O	Four inputs can each drive a Fast Clock (FCLK) buffer which can deliver a clock signal to any IOB clock input in the octant of the die served by the Fast Clock buffer. Two Fast Clock buffers serve the two IOB octants on the left side of the die and the other two Fast Clock buffers serve the two IOB octants on the right side of the die. On each side of the die, one Fast Clock buffer serves the upper octant and the other serves the lower octant. If not used to drive a Fast Clock buffer, any of these pins is a user-programmable I/O.

Table 16: Pin Descriptions (Continued)

Pin Name	I/O During Config.	I/O After Config.	Pin Description
$\overline{CS0}$, CS1, $\overline{WS}$, $\overline{RS}$	I	I/O	These four inputs are used in Asynchronous Peripheral mode. The chip is selected when $\overline{CS0}$ is Low and CS1 is High. While the chip is selected, a Low on Write Strobe ($\overline{WS}$) loads the data present on the D0 - D7 inputs into the internal data buffer. A Low on Read Strobe ($\overline{RS}$) changes D7 into a status output — High if Ready, Low if Busy — and drives D0 - D6 High. In Express mode, CS1 is used as a serial-enable signal for daisy-chaining. $\overline{WS}$ and $\overline{RS}$ should be mutually exclusive, but if both are Low simultaneously, the Write Strobe overrides. After configuration, these are user-programmable I/O pins.
A0 - A17	O	I/O	During Master Parallel configuration, these 18 output pins address the configuration EPROM. After configuration, they are user-programmable I/O pins.
A18 - A21 (XC4003XL to XC4085XL)	O	I/O	During Master Parallel configuration with an XC4000X master, these 4 output pins add 4 more bits to address the configuration EPROM. After configuration, they are user-programmable I/O pins. (See Master Parallel Configuration section for additional details.)
D0 - D7	I	I/O	During Master Parallel and Peripheral configuration, these eight input pins receive configuration data. After configuration, they are user-programmable I/O pins.
DIN	I	I/O	During Slave Serial or Master Serial configuration, DIN is the serial configuration data input receiving data on the rising edge of CCLK. During Parallel configuration, DIN is the D0 input. After configuration, DIN is a user-programmable I/O pin.
DOUT	O	I/O	During configuration in any mode but Express mode, DOUT is the serial configuration data output that can drive the DIN of daisy-chained slave FPGAs. DOUT data changes on the falling edge of CCLK, one-and-a-half CCLK periods after it was received at the DIN input. In Express modefor XC4000E and XC4000X only, DOUT is the status output that can drive the CS1 of daisy-chained FPGAs, to enable and disable downstream devices. After configuration, DOUT is a user-programmable I/O pin.
Unrestricted User-Programmable I/O Pins			
I/O	Weak Pull-up	I/O	These pins can be configured to be input and/or output after configuration is completed. Before configuration is completed, these pins have an internal high-value pull-up resistor (25 kΩ - 100 kΩ) that defines the logic level as High.

Boundary Scan

The 'bed of nails' has been the traditional method of testing electronic assemblies. This approach has become less appropriate, due to closer pin spacing and more sophisticated assembly methods like surface-mount technology and multi-layer boards. The IEEE Boundary Scan Standard 1149.1 was developed to facilitate board-level testing of electronic assemblies. Design and test engineers can imbed a standard test logic structure in their device to achieve high fault coverage for I/O and internal logic. This structure is easily implemented with a four-pin interface on any boundary scan-compatible IC. IEEE 1149.1-compatible devices may be serial daisy-chained together, connected in parallel, or a combination of the two.

The XC4000 Series implements IEEE 1149.1-compatible BYPASS, PRELOAD/SAMPLE and EXTEST boundary scan instructions. When the boundary scan configuration option is selected, three normal user I/O pins become dedicated inputs for these functions. Another user output pin becomes the dedicated boundary scan output. The details of how to enable this circuitry are covered later in this section.

By exercising these input signals, the user can serially load commands and data into these devices to control the driving of their outputs and to examine their inputs. This method is an improvement over bed-of-nails testing. It avoids the need to over-drive device outputs, and it reduces the user interface to four pins. An optional fifth pin, a reset for the control logic, is described in the standard but is not implemented in Xilinx devices.

The dedicated on-chip logic implementing the IEEE 1149.1 functions includes a 16-state machine, an instruction register and a number of data registers. The functional details can be found in the IEEE 1149.1 specification and are also discussed in the Xilinx application note XAPP 017: "*Boundary Scan in XC4000 Devices*."

Figure 40 on page 43 shows a simplified block diagram of the XC4000E Input/Output Block with boundary scan implemented. XC4000X boundary scan logic is identical.

XC4000 Series Electrical Characteristics and Device-Specific Pinout Table

For the latest Electrical Characteristics and package/pinout information for each XC4000 Family, see the Xilinx web site at http://www.xilinx.com/partinfo/databook.htm#xc4000

Ordering Information

Example: XC4013E-3HQ240C

- Device Type
- Speed Grade
  - -6
  - -5
  - -4
  - -3
  - -2
  - -1
- Temperature Range
  - C = Commercial (T_J = 0 to +85°C)
  - I = Industrial (T_J = -40 to +100°C)
  - M = Military (T_C = -55 to+125°C)
- Number of Pins
- Package Type
  - PC = Plastic Lead Chip Carrier
  - PQ = Plastic Quad Flat Pack
  - VQ = Very Thin Quad Flat Pack
  - TQ = Thin Quad Flat Pack
  - BG = Ball Grid Array
  - PG = Ceramic Pin Grid Array
  - HQ = High Heat Dissipation Quad Flat Pack
  - MQ = Metal Quad Flat Pack
  - CB = Top Brazed Ceramic Quad Flat Pack

X9020

Revision Control

Version	Description
3/30/98 (1.5)	Updated XC4000XL timing and added XC4002XL
1/29/99 (1.5)	Updated pin diagrams
5/14/99 (1.6)	Replaced Electrical Specification and pinout pages for E, EX, and XL families with separate updates and added URL link for electrical specifications/pinouts for WebLINX users

XC4010E/XL Device Pinout Tables

The following table may contain pinout information for unsupported device/package combinations. Please see the availability charts elsewhere in the XC4000 Series data sheet for availability information.

XC4010E/XL Pad Name	PC 84	PQ 100††	TQ 144††	PQ 160	TQ 176††	PG 191†	PQ/HQ 208	BG 225†	BG 256††	Bndry Scan
VCC	P2	P99	P128	P142	P155	/CC*	P183	VCC*	VCC*	-
I/O (A8)	P3	P93	P129	P143	P156	J3	P184	E8	C10	62
I/O (A9)	P4	P94	P130	P144	P157	J2	P185	B7	D10	65
I/O (19)	-	P95	P131	P145	P158	J1	P186	A7	A9	68
I/O (18)	-	P96	P132	P146	P159	H1	P187	C7	B9	71
I/O	-	-	-	-	P160	H2	P188	D7	C9	74
I/O	-	-	-	-	P161	H3	P189	E7	D9	77
I/O (A10)	P5	P97	P133	P147	P162	G1	P190	A6	A8	80
I/O (A11)	P6	P98	P134	P148	P163	G2	P191	B6	B8	83
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O	-	-	P135	P149	P164	F1	P192	A5	B6	86
I/O	-	-	P136	P150	P165	E1	P193	B5	A5	89
GND	-	-	P137	P151	P166	GND*	P194	GND*	GND*	-
I/O	-	-	-	-	-	F2	P195	D6	C6	92
I/O	-	-	-	-	P167	D1	P196	C5	B5	95
I/O	-	-	-	P152	P168	C1	P197	A4	A4	98
I/O	-	-	-	P153	P169	E2	P198	E6	C5	101
I/O (A12)	P7	P99	P138	P154	P170	F3	P199	B4	B4	104
I/O (A13)	P8	P100	P139	P155	P171	D2	P200	D5	A3	107
I/O	-	-	P140	P156	P172	B1	P201	B3	B3	110
I/O	-	-	P141	P157	P173	E3	P202	F6	B2	113
I/O (A14)	P9	P1	P142	P158	P174	C2	P203	A2	A2	116
I/O, SGCK1 †, GCK8 †† (A15)	P10	P2	P143	P159	P175	B2	P204	C3	C3	119
VCC	P11	P3	P144	P160	P176	VCC*	P205	VCC*	VCC*	-
GND	P12	P4	P1	P1	P1	GND*	P2	GND*	GND*	-
I/O, PGCK1†, GCK1†† (A16)	P13	P5	P2	P2	P2	C3	P4	D4	B1	122
I/O (A17)	P14	P6	P3	P3	P3	C4	P5	B1	C2	125
I/O	-	-	P4	P4	P4	B3	P6	C2	D2	128
I/O	-	-	P5	P5	P5	C5	P7	E5	D3	131
I/O, TDI	P15	P7	P6	P6	P6	A2	P8	D3	E4	134
I/O, TCK	P16	P8	P7	P7	P7	B4	P9	C1	C1	137
I/O	-	-	-	P8	P8	C6	P10	D2	D1	140
I/O	-	-	-	P9	P9	A3	P11	G6	E3	143
I/O	-	-	-	-	-	B5	P12	E4	E2	146
I/O	-	-	-	-	-	B6	P13	D1	E1	149
GND	-	-	P8	P10	P10	GND*	P14	GND*	GND*	-
I/O	-	-	P9	P11	P11	A4	P15	F5	G3	152
I/O	-	-	P10	P12	P12	A5	P16	E1	G2	155
I/O, TMS	P17	P9	P11	P13	P13	B7	P17	F4	G1	158
I/O	P18	P10	P12	P14	P14	A6	P18	F3	H3	161
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O	-	-	-	-	P15	C8	P19	G4	J2	164
I/O	-	-	-	-	P16	A7	P20	G3	J1	167
I/O	-	-	P13	P15	P17	B8	P21	G2	K2	170
I/O	-	P11	P14	P16	P18	A8	P22	G1	K3	173
I/O	P19	P12	P15	P17	P19	B9	P23	G5	K1	176
I/O	P20	P13	P16	P18	P20	C9	P24	H3	L1	179
GND	P21	P14	P17	P19	P21	GND*	P25	GND*	GND*	-
VCC	P22	P15	P18	P20	P22	VCC*	P26	VCC*	VCC*	-
I/O	P23	P16	P19	P21	P23	C10	P27	H4	L2	182
I/O	P24	P17	P20	P22	P24	B10	P28	H5	L3	185
I/O	-	P18	P21	P23	P25	A9	P29	J2	L4	188
I/O	-	-	P22	P24	P26	A10	P30	J1	M1	191
I/O	-	-	-	-	P27	A11	P31	J3	M2	194
I/O	-	-	-	-	P28	C11	P32	J4	M3	197
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O	P25	P19	P23	P25	P29	B11	P33	K2	P1	200
I/O	P26	P20	P24	P26	P30	A12	P34	K3	P2	203
I/O	-	-	P25	P27	P31	B12	P35	J6	R1	206
I/O	-	-	P26	P28	P32	A13	P36	L1	P3	209
GND	-	-	P27	P29	P33	GND*	P37	GND*	GND*	-
I/O	-	-	-	-	-	B13	P38	L3	T2	212
I/O	-	-	-	-	-	A14	P39	M1	U1	215
I/O	-	-	-	P30	P34	A15	P40	K5	T3	218
I/O	-	-	-	P31	P35	C13	P41	M2	U2	221

XC4010E/XL Pad Name	PC 84	PQ 100††	TQ 144††	PQ 160	TQ 176††	PG 191†	PQ/HQ 208	BG 225†	BG 256††	Bndry Scan
I/O	P27	P21	P28	P32	P36	B14	P42	L4	V1	224
I/O	-	P22	P29	P33	P37	A16	P43	N1	T4	227
I/O	-	-	P30	P34	P38	B15	P44	M3	U3	230
I/O	-	-	P31	P35	P39	C14	P45	N2	V2	233
I/O	P28	P23	P32	P36	P40	A17	P46	K6	W1	236
I/O, SGCK2 †, GCK2 ††	P29	P24	P33	P37	P41	B16	P47	P1	V3	239
O (M1)	P30	P25	P34	P38	P42	C15	P48	N3	W2	242
GND	P31	P26	P35	P39	P43	GND*	P49	GND*	GND*	-
I (M0)	P32	P27	P36	P40	P44	A18	P50	P2	Y1	245
VCC	P33	P28	P37	P41	P45	VCC*	P55	VCC*	VCC*	-
I (M2)	P34	P29	P38	P42	P46	C16	P56	M4	W3	246
I/O, PGCK2 †, GCK3 ††	P35	P30	P39	P43	P47	B17	P57	R2	Y2	247
I/O (HDC)	P36	P31	P40	P44	P48	E16	P58	P3	W4	250
I/O	-	-	P41	P45	P49	C17	P59	L5	V4	253
I/O	-	-	P42	P46	P50	D17	P60	N4	U5	256
I/O	-	P32	P43	P47	P51	B18	P61	R3	Y3	259
I/O (LDC)	P37	P33	P44	P48	P52	E17	P62	P4	Y4	262
I/O	-	-	-	P49	P53	F16	P63	K7	V5	265
I/O	-	-	-	P50	P54	C18	P64	M5	W5	268
I/O	-	-	-	-	-	D18	P65	R4	Y5	271
I/O	-	-	-	-	-	F17	P66	N5	V6	274
GND	-	-	P45	P51	P55	GND*	P67	GND*	GND*	-
I/O	-	-	P46	P52	P56	E18	P68	R5	W7	277
I/O	-	-	P47	P53	P57	F18	P69	M6	Y7	280
I/O	P38	P34	P48	P54	P58	G17	P70	N6	V8	283
I/O	P39	P35	P49	P55	P59	G18	P71	P6	W8	286
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O	-	-	-	-	P60	H16	P72	R6	Y8	289
I/O	-	-	-	-	P61	H17	P73	M7	U9	292
I/O	-	P36	P50	P56	P62	H18	P74	R7	V10	295
I/O	-	P37	P51	P57	P63	J18	P75	L7	Y10	298
I/O	P40	P38	P52	P58	P64	J17	P76	N8	Y11	301
I/O (INIT)	P41	P39	P53	P59	P65	J16	P77	P8	W11	304
VCC	P42	P40	P54	P60	P66	VCC*	P78	VCC*	VCC*	-
GND	P43	P41	P55	P61	P67	GND*	P79	GND*	GND*	-
I/O	P44	P42	P56	P62	P68	K16	P80	L8	V11	307
I/O	P45	P43	P57	P63	P69	K17	P81	P9	U11	310
I/O	-	P44	P58	P64	P70	K18	P82	R9	Y12	313
I/O	-	P45	P59	P65	P71	L18	P83	N9	W12	316
I/O	-	-	-	-	P72	L17	P84	M9	V12	319
I/O	-	-	-	-	P73	L16	P85	L9	U12	322
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O	P46	P46	P60	P66	P74	M18	P86	N10	Y15	325
I/O	P47	P47	P61	P67	P75	M17	P87	K9	V14	328
I/O	-	-	P62	P68	P76	N18	P88	R11	W15	331
I/O	-	-	P63	P69	P77	P18	P89	P11	Y16	334
GND	-	-	P64	P70	P78	GND*	P90	GND*	GND*	-
I/O	-	-	-	-	-	N17	P91	R12	Y17	337
I/O	-	-	-	-	-	R18	P92	L10	V16	340
I/O	-	-	-	P71	P79	T18	P93	P12	W17	343
I/O	-	-	-	P72	P80	P17	P94	M11	Y18	346
I/O	P48	P48	P65	P73	P81	N16	P95	R13	U16	349
I/O	P49	P49	P66	P74	P82	T17	P96	N12	V17	352
I/O	-	-	P67	P75	P83	R17	P97	P13	W18	355
I/O	-	-	P68	P76	P84	P16	P98	K10	Y19	358
I/O	P50	P50	P69	P77	P85	U18	P99	R14	V18	361
I/O, SGCK3 †, GCK4 ††	P51	P51	P70	P78	P86	T16	P100	N13	W19	364
GND	P52	P52	P71	P79	P87	GND*	P101	GND*	GND*	-
DONE	P53	P53	P72	P80	P88	U17	P103	P14	Y20	-
VCC	P54	P54	P73	P81	P89	VCC*	P106	VCC*	VCC*	-
PROGRAM	P55	P55	P74	P82	P90	V18	P108	M12	V19	-
I/O (D7)	P56	P56	P75	P83	P91	T15	P109	P15	U19	367
I/O, PGCK3 †, GCK5 ††	P57	P57	P76	P84	P92	U16	P110	N14	U18	370
I/O	-	-	P77	P85	P93	T14	P111	L11	T17	373
I/O	-	-	P78	P86	P94	U15	P112	M13	V20	376

XC4010E/XL Pad Name	PC 84	PQ 100††	TQ 144††	PQ 160	TQ 176††	PG 191†	PQ/HQ 208	BG 225†	BG 256††	Bndry Scan
I/O (D6)	P58	P58	P79	P87	P95	V17	P113	J10	T19	379
I/O	-	P59	P80	P88	P96	V16	P114	L12	T20	382
I/O	-	-	-	P89	P97	T13	P115	M15	R18	385
I/O	-	-	-	P90	P98	U14	P116	L13	R19	388
I/O	-	-	-	-	-	V15	P117	L14	R20	391
I/O	-	-	-	-	-	V14	P118	K11	P18	394
GND	-	-	P81	P91	P99	GND*	P119	GND*	GND*	-
I/O	-	-	P82	P92	P100	U13	P120	K13	N19	397
I/O	-	-	P83	P93	P101	V13	P121	K14	N20	400
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O (D5)	P59	P60	P84	P94	P102	U12	P122	K15	M17	403
I/O (CS0)	P60	P61	P85	P95	P103	V12	P123	J12	M18	406
I/O	-	-	-	-	P104	T11	P124	J13	M20	409
I/O	-	-	-	-	P105	U11	P125	J14	L19	412
I/O	-	P62	P86	P96	P106	V11	P126	J15	L18	415
I/O	-	P63	P87	P97	P107	V10	P127	J11	L20	418
I/O (D4)	P61	P64	P88	P98	P108	U10	P128	H13	K20	421
I/O	P62	P65	P89	P99	P109	T10	P129	H14	K19	424
VCC	P63	P66	P90	P100	P110	VCC*	P130	VCC*	VCC*	-
GND	P64	P67	P91	P101	P111	GND*	P131	GND*	GND*	-
I/O (D3)	P65	P68	P92	P102	P112	T9	P132	H12	K18	427
I/O (RS)	P66	P69	P93	P103	P113	U9	P133	H11	K17	430
I/O	-	P70	P94	P104	P114	V9	P134	G14	J20	433
I/O	-	-	P95	P105	P115	V8	P135	G15	J19	436
I/O	-	-	-	-	P116	U8	P136	G13	J18	439
I/O	-	-	-	-	P117	T8	P137	G12	J17	442
I/O (D2)	P67	P71	P96	P106	P118	V7	P138	G11	H19	445
I/O	P68	P72	P97	P107	P119	U7	P139	F15	H18	448
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O	-	-	P98	P108	P120	V6	P140	F14	G19	451
I/O	-	-	P99	P109	P121	U6	P141	F13	F20	454
GND	-	-	P100	P110	P122	GND*	P142	GND*	GND*	-
I/O	-	-	-	-	-	V5	P143	E13	D20	457
I/O	-	-	-	-	-	V4	P144	D15	E18	460
I/O	-	-	-	P111	P123	U5	P145	F11	D19	463
I/O	-	-	-	P112	P124	T6	P146	D14	C20	466
I/O (D1)	P69	P73	P101	P113	P125	V3	P147	E12	E17	469
I/O (RCLK, RDY/BUSY)	P70	P74	P102	P114	P126	V2	P148	C15	D18	472
I/O	-	-	P103	P115	P127	U4	P149	D13	C19	475
I/O	-	-	P104	P116	P128	T5	P150	C14	B20	478
I/O (D0, DIN)	P71	P75	P105	P117	P129	U3	P151	F10	C18	481
I/O, SGCK4 †, GCK6 †† (DOUT)	P72	P76	P106	P118	P130	T4	P152	B15	B19	484
CCLK	P73	P77	P107	P119	P131	V1	P153	C13	A20	-
VCC	P74	P78	P108	P120	P132	VCC*	P154	VCC*	VCC*	-
O, TDO	P75	P79	P109	P121	P133	U2	P159	A15	A19	0
GND	P76	P80	P110	P122	P134	GND*	P160	GND*	GND*	-
I/O (A0, WS)	P77	P81	P111	P123	P135	T3	P161	A14	B18	2
I/O, PGCK4 †, GCK7 †† (A1)	P78	P82	P112	P124	P136	U1	P162	B13	B17	5
I/O	-	-	P113	P125	P137	P3	P163	E11	C17	8
I/O	-	-	P114	P126	P138	R2	P164	C12	D16	11
I/O (CS1, A2)	P79	P83	P115	P127	P139	T2	P165	A13	A18	14
I/O (A3)	P80	P84	P116	P128	P140	N3	P166	B12	A17	17
I/O	-	-	P117	P129	P141	P2	P167	A12	A16	20
I/O	-	-	-	P130	P142	T1	P168	C11	C15	23
I/O	-	-	-	-	-	R1	P169	B11	B15	26
I/O	-	-	-	-	-	N2	P170	E10	A15	29
GND	-	-	P118	P131	P143	GND*	P171	GND*	GND*	-
I/O	-	-	P119	P132	P144	P1	P172	A11	B14	32
I/O	-	-	P120	P133	P145	N1	P173	D10	A14	35
VCC	-	-	-	-	-	VCC*	-	VCC*	VCC*	-
I/O (A4)	P81	P85	P121	P134	P146	M2	P174	A10	C12	38
I/O (A5)	P82	P86	P122	P135	P147	M1	P175	D9	B12	41
I/O	-	-	-	-	P148	L3	P176	C9	A12	44
I/O	-	-	-	P136	P149	L2	P177	B9	B11	47
I/O (A21)††	-	P87	P123	P137	P150	L1	P178	A9	C11	50
I/O (A20)††	-	P88	P124	P138	P151	K1	P179	E9	A11	53
I/O (A6)	P83	P89	P125	P139	P152	K2	P180	C8	A10	56
I/O (A7)	P84	P90	P126	P140	P153	K3	P181	B8	B10	59
GND	P1	P91	P127	P141	P154	GND*	P182	GND*	GND*	-

6/19/97

* Pads labelled GND* or VCC* are internally bonded to Ground or VCC planes within the package. They have no direct connection to any specific package pin.
† = E only
†† = XL only

Additional XC4010E/XL Package Pins

PQ/HQ208

Not Connected Pins						
P1	P3	P51	P52	P53	P54	P102
P104	P105	P107	P155	P156	P157	P158
P206	P207	P208	-	-	-	-

5/27/97

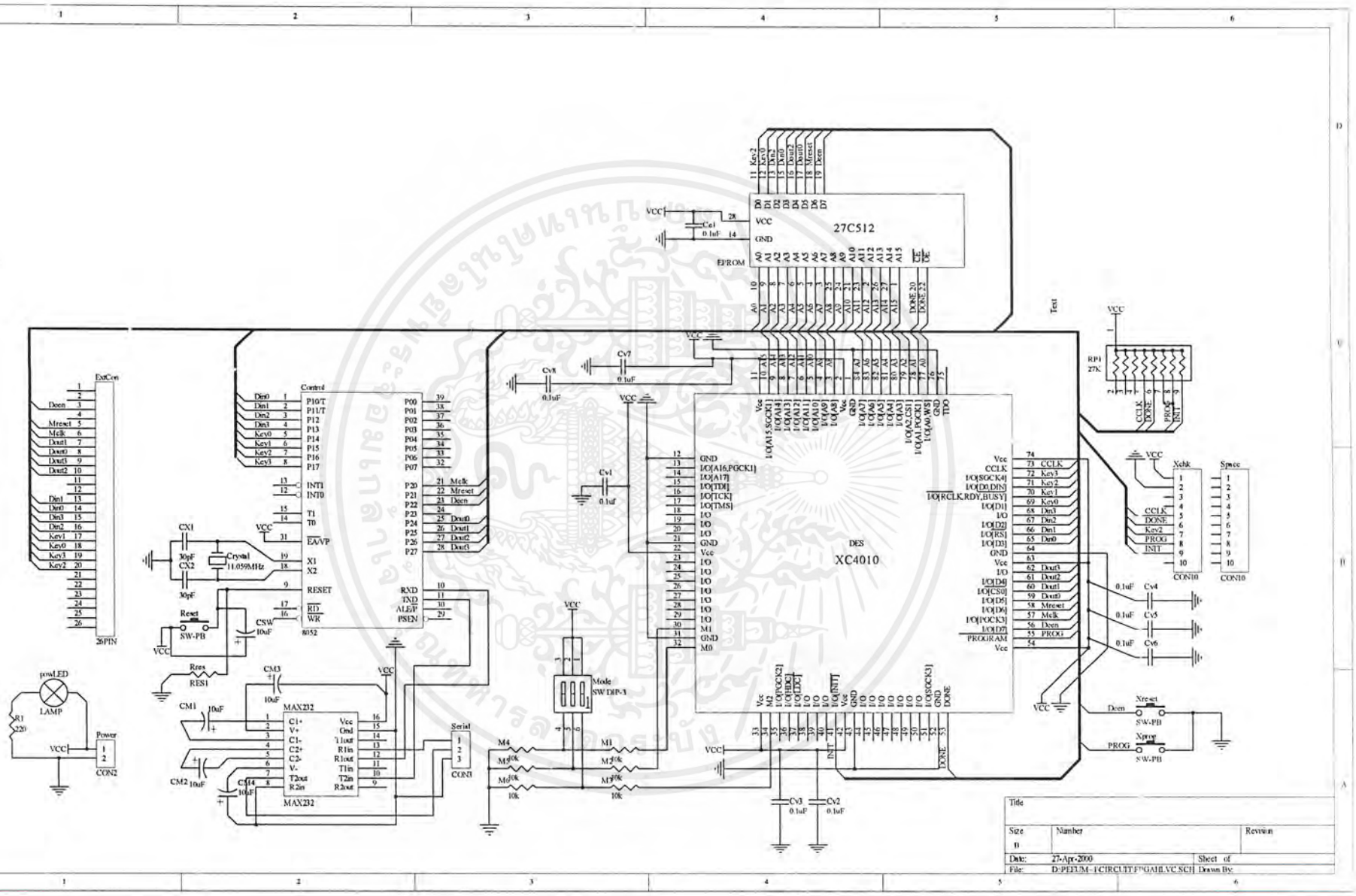
PG191

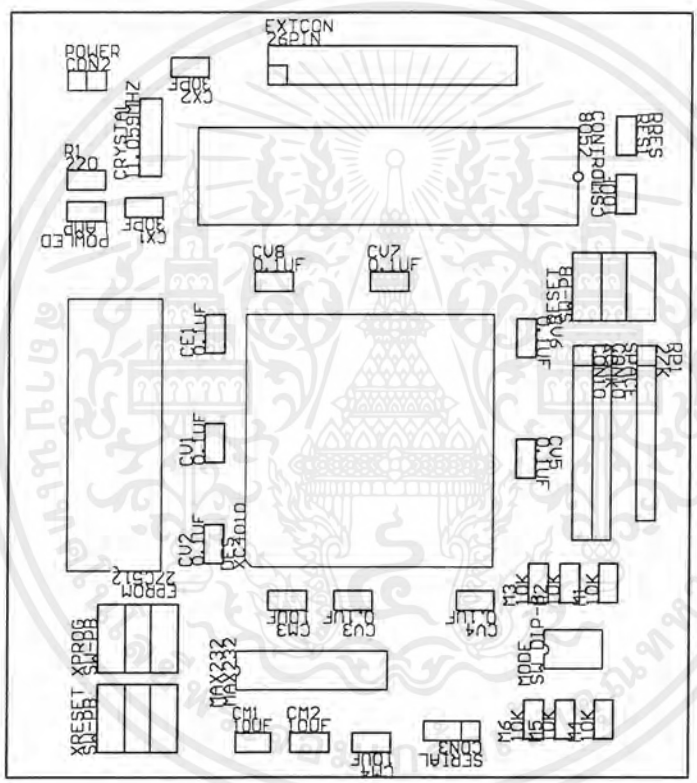
VCC Pins						
D3	D10	D16	J4	J15	R4	R10
R15	-	-	-	-	-	-
GND Pins						
C7	C12	D4	D9	D15	G3	G16
K4	K15	M3	M16	R3	R9	R16
T7	T12	-	-	-	-	-

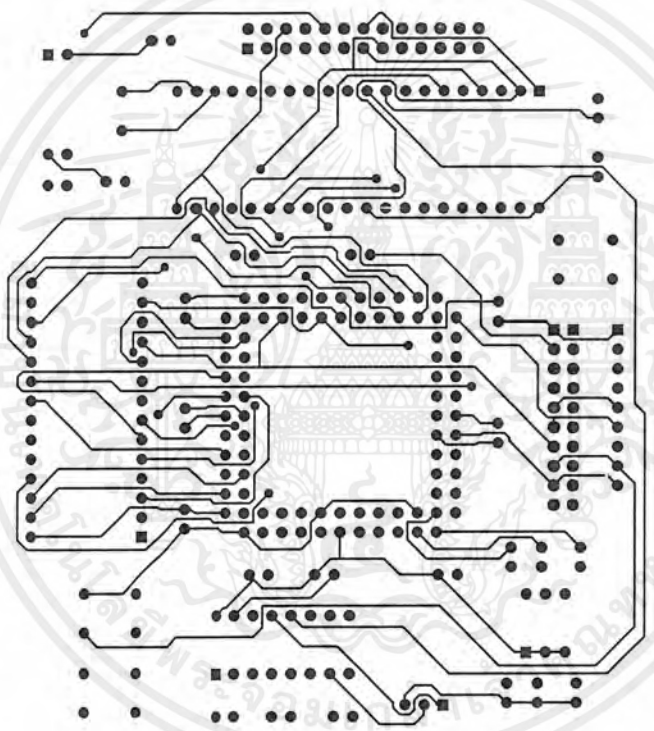
5/27/97

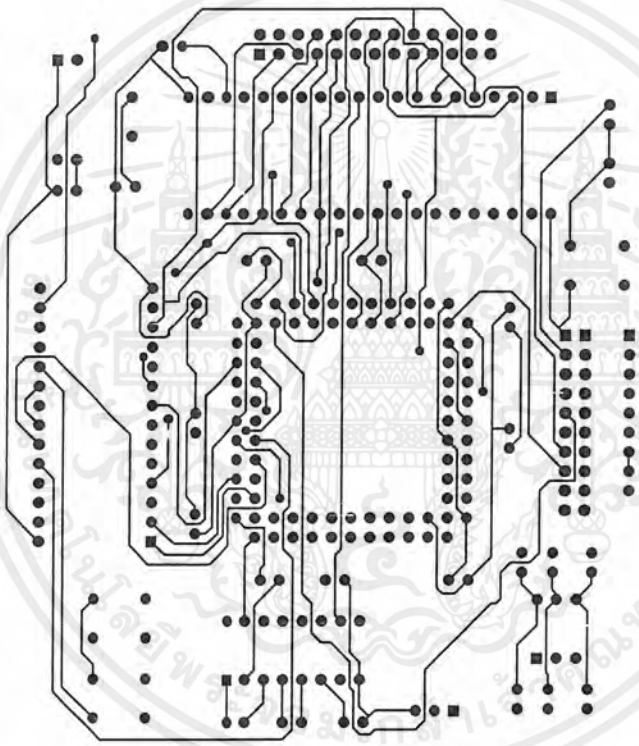
ภาคผนวก ข.

วงจรฮาร์ดล็อคต้นแบบ


27C512
EPROM
DES
XC4010
8052
MAX232
Control
ExCon
26PIN
Crystal
11.059MHz
SW-PB
LAMP
Mode
SW DIP-3
Test
Xchk
Spece
CON10
Xreset
Xprog
Title
Size
B
Number
Revision
Date: 27-Apr-2000
Sheet of
File: D:\PEDUM~1\CIRCUIT\FPGA\LVC.SCH
Drawn By:







ภาคผนวก ค.

ความรู้เบื้องต้นเกี่ยวกับมาตรฐาน USB (Universal Serial Bus)

Universal Serial Bus (USB)

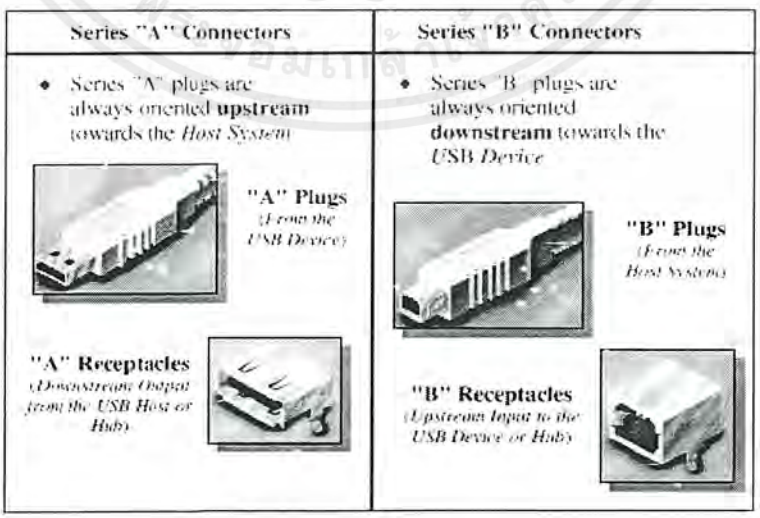
สำหรับในโครงงานนี้ ในตอนแรกนั้นทางเราได้ตั้งใจไว้ว่าจะใช้มาตรฐานยูเอสบี (USB : Universal Serial Bus) ในการเชื่อมต่อระหว่างอุปกรณ์ฮาร์ดล็อคเข้ากับเครื่องคอมพิวเตอร์ แต่เนื่องจากเทคโนโลยีทางด้านมาตรฐานยูเอสบีนั้นยังเป็นเรื่องที่ค่อนข้างใหม่ ทำให้เกิดปัญหาในการจัดหาอุปกรณ์เกี่ยวกับยูเอสบีมาใช้งาน แต่อย่างไรก็ตามทางเราก็ได้ทำการศึกษาความรู้พื้นฐานเกี่ยวกับมาตรฐานยูเอสบีไว้แล้ว สำหรับผู้ที่สนใจก็สามารถที่จะนำความรู้พื้นฐานเหล่านี้ไปพัฒนาได้ต่อไป

Universal Serial Bus (USB)

มาตรฐาน USB (Universal Serial Bus) เป็นมาตรฐานที่ใช้ในการรับ – ส่งข้อมูลระหว่างอุปกรณ์ต่าง ๆ ของเครื่องคอมพิวเตอร์ ซึ่งจะมุ่งเน้นไปที่ผู้ใช้และการใช้งานให้เป็นประโยชน์ (Application) ต่าง ๆ โดยมีการใช้เทคโนโลยีของ Computer Telephony Integration (CTI) สำหรับ USB แล้วนั้นจะมีลักษณะเดียวกันกับบัส (Bus) ที่เชื่อมต่อระหว่างสล็อต (Slot) ต่าง ๆ บนเมนบอร์ด (Mainboard) ของเครื่องคอมพิวเตอร์ บัสสำหรับ USB นั้นสามารถเชื่อมต่อกับอุปกรณ์ได้สูงสุดถึง 127 ตัวโดยที่ไม่มีปัญหาเพราะ USB จะมีกลไกที่ทำให้อุปกรณ์ทั้ง 127 ตัวสามารถใช้ทรัพยากรร่วมกันได้โดยไม่มีการยึดครองทรัพยากรที่มีจำกัดไปไว้ที่อุปกรณ์ตัวใดตัวหนึ่ง

สำหรับการลดปัญหาต่าง ๆ ที่จะเกิดกับผู้ใช้งานนั้น USB มีการใช้โปรโตคอล (Protocol) ที่เรียกว่า Keyed Connector Protocol ซึ่งคอนเนคเตอร์ (Connector) ใน USB นั้นจะมีอยู่ด้วยกัน 2 แบบคือแบบ A (Series A) กับ แบบ B (Series B) และความแตกต่าง ๆ ระหว่างแบบ A กับแบบ B นั้นจะทำให้ผู้ใช้งานสามารถเลือกใช้ในการเชื่อมต่อได้อย่างเหมาะสม

ในคอนเนคเตอร์ของแบบ A นั้นจะเป็นหลักในการเชื่อมต่อของอุปกรณ์ต่าง ๆ ในมาตรฐาน USB ซึ่งสำหรับอุปกรณ์ทาง USB ทั้งหมดนั้นจะมีคอนเนคเตอร์แบบ A ประกอบอยู่ด้วย สำหรับในส่วนของคอนเนคเตอร์แบบ B นั้นจะเป็นมาตรฐานที่ทางผู้ผลิตสามารถที่จะจัดแยกออกมาได้



รูปที่ 1 Keyed Connector Protocol

ลักษณะที่สำคัญของ USB

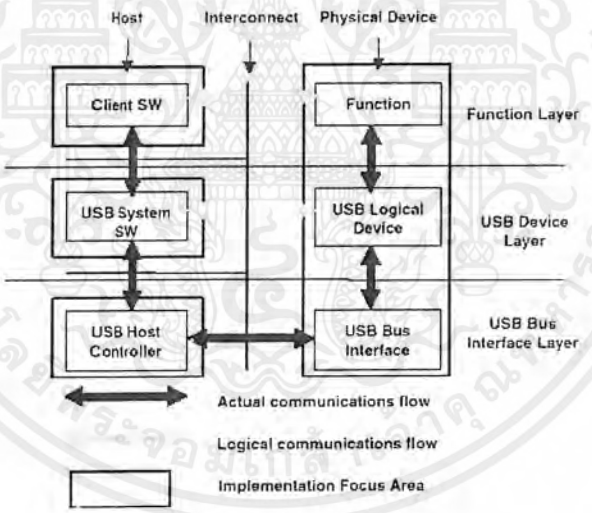
สำหรับลักษณะที่ของมาตรฐาน USB นั้นจะเป็นการรวบรวมคุณสมบัติความสามารถในการทำงานได้ในหลาย ๆ รูปแบบ และยังมีความสามารถในการทำหน้าที่การทำงานได้ในระบบและส่วนประกอบที่แตกต่างกัน ซึ่งลักษณะต่าง ๆ ที่สำคัญของมาตรฐาน USB นั้นมีดังนี้

- ง่ายต่อการใช้งานเพราะมีความสามารถที่เรียกว่า Plug and Play และนอกจากนี้ยังเป็นการง่ายต่อการขยายส่วนต่าง ๆ ภายนอกของเครื่องคอมพิวเตอร์
- อุปกรณ์แบบ USB ทั้งหมดจะมีความสามารถที่เรียกว่า Hot Swapping ซึ่งทำให้สามารถที่จะต่ออุปกรณ์ต่าง ๆ เข้ากับพอร์ท USB (USB Port) ได้ทันทีโดยไม่ต้องจำเป็นต้องปิดเครื่องคอมพิวเตอร์ก่อน และเครื่องก็จะรับรู้ทันทีว่ามีอุปกรณ์ชิ้นใหม่ต่อเข้ามาแล้ว โดยไม่ต้องทำการรีสตาร์ทวินโดวส์ใหม่ และจะทำการติดตั้งโปรแกรมต่าง ๆ ที่มีความจำเป็นให้โดยอัตโนมัติ สำหรับการถอดอุปกรณ์ออกจากระบบนั้นก็สามารถที่จะทำได้ตลอดเวลาเช่นเดียวกันโดยจะไม่ทำให้เครื่องเสียหายหรือโปรแกรมเกิดการหยุดทำงานด้วย
- มาตรฐาน USB นั้นสามารถที่จะเชื่อมต่อกับอุปกรณ์ได้สูงสุดถึง 127 ตัว โดยใช้อุปกรณ์ขยายพอร์ทที่เรียกว่า HUB มาต่อเข้ากับบัส
- USB นั้นจะสนับสนุนการทำงานของอุปกรณ์หลาย ๆ ตัวที่ทำงานพร้อมกัน หรือที่เรียกว่าการทำงานแบบ Concurrency
- USB นั้นจะเหมาะกับอุปกรณ์ที่มีช่วงของอัตราในการส่งสัญญาณ (Bandwidths) ตั้งแต่ไม่กี่กิโลบิทต่อวินาที (kb/s) จนถึงหลาย ๆ เมกกะบิทต่อวินาที (Mb/s) ซึ่งถือว่ามีช่วงของอัตราการส่งสัญญาณที่กว้างพอสมควร
- มาตรฐาน USB นั้นจะสนับสนุนการรับ-ส่งข้อมูลข่าวสารที่มีปริมาณมาก ๆ ในระหว่างตัวคอมพิวเตอร์หลัก (Host) และตัวอุปกรณ์ที่ต่ออยู่กับมัน
- มีความสามารถที่จะใช้ได้กับอุปกรณ์ในหลาย ๆ แบบ
- มีโอเวอร์เฮด (Overhead) ของข้อมูลในการติดต่อสื่อสารต่ำ ทำให้ Bus มีการใช้ประโยชน์อย่างเต็มที่
- มีความเหมาะสมในการใช้งานในด้านระบบโทรศัพท์ และระบบเสียง
- USB จะสนับสนุนขนาดของชุดข้อมูลหลาย ๆ ขนาดในการส่งข้อมูล
- มีกลไกความสามารถในการจัดการกับข้อผิดพลาดต่าง ๆ ที่เกิดขึ้น (Error Handling) และยังมีความสามารถในการกู้ข้อมูลคืนจากความผิดพลาดต่าง ๆ ที่เกิดขึ้น (Fault Recovery) ด้วย
- USB จะมีความเร็วในการส่งข้อมูล 2 ระดับคือ ระดับต่ำ (Low – Speed) ที่ 1.5 เมกกะบิทต่อวินาที และระดับสูง (Full – Speed) ที่ 12 เมกกะบิทต่อวินาที
- บัสของ USB นั้นสามารถที่จะจ่ายกระแสไฟฟ้าให้กับอุปกรณ์ต่าง ๆ ที่ต่ออยู่ได้ โดยจะจ่ายกระแส 100 มิลลิแอมป์ (mA) สำหรับอุปกรณ์ที่มีความเร็วต่ำ (1.5 เมกกะบิทต่อวินาที) และจะจ่าย 500 มิลลิแอมป์สำหรับอุปกรณ์ที่มีความเร็วสูง (12 เมกกะบิทต่อวินาที)

- USB นั้นจะสนับสนุนการส่งข้อมูลทั้งแบบโหมดไอโซโครนัส (Isochronous Mode) และโหมดอะซิงโครนัส (Asynchronous Mode)
- USB มีความสามารถในการส่งข้อมูลแบบ 2 ทิศทาง (Full Duplex)
- USB จะสนับสนุนข้อมูลแบบตอบสนอง (Real – Time) อย่างเต็มที่

โครงสร้างโดยรวมของ USB

สำหรับบัสของ USB นั้นจะเป็นสายเคเบิล (Cable Bus) ที่สนับสนุนการติดต่อสื่อสารและแลกเปลี่ยนข้อมูลกันระหว่างคอมพิวเตอร์กับอุปกรณ์ภายนอกต่าง ๆ ที่ต่ออยู่ซึ่งสามารถเข้าถึงข้อมูลได้ในเวลาเดียวกัน สำหรับการต่ออุปกรณ์นั้น อุปกรณ์ภายนอกจะทำการแบ่งช่องสัญญาณของ USB ผ่านทางโปรโตคอลที่เรียกว่า Host – Scheduled กับ Token – Based Protocol โดยบัสของ USB นั้นจะยินยอมให้อุปกรณ์ภายนอกทำการต่อเข้า การตั้งค่าโดยรวม การเรียกใช้งาน และการถอดอุปกรณ์ออกได้แม้ว่าตัวคอมพิวเตอร์หลักจะทำงานกับอุปกรณ์ตัวอื่นอยู่ก็ตาม สำหรับในโครงงานนี้นั้นเราจะสนใจเพียงแค่การติดต่อกันของตัวคอมพิวเตอร์หลักกับอุปกรณ์ USB เพียงตัวเดียวเท่านั้น ซึ่งก็คือตัว Hardlock นั่นเอง โดยจะมีลักษณะการเชื่อมต่อกัน ดังภาพ



รูปที่ 2 แสดงการเชื่อมต่อกันของ USB Host กับ USB Device

จากภาพนั้น เราจะมุ่งประเด็นไปที่ส่วนของการสนับสนุน 4 ส่วน ดังนี้

1. อุปกรณ์ทางกายภาพของ USB (USB Physical Device) : เป็นส่วนของอุปกรณ์ฮาร์ดแวร์ที่อยู่ที่ปลายสายเคเบิลของ USB
2. ซอฟต์แวร์ของไคลเอนต์ (Client Software) : เป็นซอฟต์แวร์ที่จะทำการปฏิบัติงานที่เครื่องคอมพิวเตอร์ ซึ่งจะสอดคล้องกับอุปกรณ์ USB สำหรับ Client Software นี้จะถูกจัดไว้ให้โดยระบบปฏิบัติการหรือถูกจัดให้มาพร้อมด้วยอุปกรณ์ USB

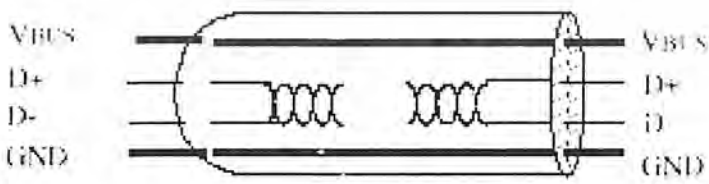
3. ซอฟต์แวร์ของระบบ USB (USB System Software) : เป็นซอฟต์แวร์ที่จะสนับสนุนกับมาตรฐาน USB ในระบบปฏิบัติการที่เฉพาะ โดยจะถูกจัดไว้โดยระบบปฏิบัติการ ซึ่งจะเป็นอิสระกับอุปกรณ์ USB หรือ Client Software
4. USB Host Controller (Host Side Bus Interface) : เป็นส่วนของทั้งฮาร์ดแวร์และซอฟต์แวร์ที่จะทำการยินยอมให้มีการติดต่อกันระหว่างอุปกรณ์ USB กับเครื่องคอมพิวเตอร์หลัก

จากรูปที่ 2 นั้นได้แสดงให้เห็นว่าการเชื่อมต่อกันระหว่างเครื่องคอมพิวเตอร์หลักกับอุปกรณ์ USB นั้นมีความต้องการการติดต่อระหว่างกันของลำดับชั้น (Layer) และเอนติตี้ (Entities) ต่าง ๆ ซึ่งจะแบ่งเป็นลำดับชั้นต่าง ๆ ดังนี้

1. ลำดับชั้นการเชื่อมต่อบัสของ USB (USB Bus Interface Layer) : จะเป็นลำดับชั้นที่ทำการเชื่อมต่อทางด้านกายภาพ (Physical) / การส่งสัญญาณ (Signaling) / ชุดข้อมูล (Packet) ระหว่างเครื่องคอมพิวเตอร์หลักกับอุปกรณ์USB
2. ลำดับชั้นของอุปกรณ์ USB (USB Device Layer) : เป็นลำดับชั้นที่เป็นการมองว่าซอฟต์แวร์ของระบบ USB นั้นมีการปฏิบัติงานทั่ว ๆ ไปของ USB กับอุปกรณ์
3. ลำดับชั้นที่เกี่ยวกับหน้าที่การทำงาน (Function Layer) : จะเป็นลำดับชั้นที่จัดหาความสามารถที่เพิ่มเติมให้กับเครื่องคอมพิวเตอร์หลักโดยผ่านทางชั้นของซอฟต์แวร์ของไคลเอนต์ที่เหมาะสม

สำหรับในลำดับชั้นของอุปกรณ์ USB และลำดับชั้นที่เกี่ยวกับหน้าที่การทำงานนั้น จะเป็นมุมมองของการติดต่อในระดับตรรกะ (Logical) ภายในแต่ละลำดับชั้นของมัน ซึ่งที่จริงเป็นการใช้ ลำดับชั้นการเชื่อมต่อบัสของ USB ในการส่งข้อมูล

สำหรับในส่วนของโครงสร้างของ USB นั้นจะขออธิบายในส่วนของการเชื่อมต่อทางกายภาพ (Physical Interface) ของ USB โดย USB จะทำการส่งสัญญาณและไฟเลี้ยงไปตามสายเคเบิล ซึ่งภายในสายเคเบิลของ USB นั้นจะมีสายไฟทั้งหมด 4 เส้น โดย 2 เส้นนั้นจะใช้ในการส่งข้อมูลและอีก 2 เส้นจะใช้ในการจ่ายไฟเลี้ยงให้กับอุปกรณ์ต่าง ๆ ที่ต่ออยู่กับบัส ดังภาพ



รูปที่ 3 แสดงโครงสร้างภายในของสายเคเบิลของ USB

จากรูปจะสามารถอธิบายได้ ดังนี้

- เส้น V_{BUS} และเส้น GND จะเป็นเส้นที่ใช้ในการจ่ายไฟเลี้ยงให้กับอุปกรณ์ต่าง ๆ ที่ต่ออยู่กับบัส ซึ่งโดยปกติแล้วเส้น V_{BUS} นั้นจะใช้ไฟ +5 โวลต์ (Volt)

- เส้น D+ และเส้น D- นั้นจะเป็นเส้นที่ใช้ในการรับ–ส่งข้อมูล โดยถ้าเราต้องการส่งข้อมูลที่ระดับความเร็วสูง (12 เมกกะบิทต่อวินาที) เราจะใช้เส้น D+ ในการส่งข้อมูล แต่ถ้าต้องการส่งข้อมูลที่ระดับความเร็วต่ำ (1.5 เมกกะบิทต่อวินาที) เราจะใช้เส้น D- ในการส่งข้อมูล

USB นั้นถูกทำให้มีประสิทธิภาพสำหรับการง่ายต่อการใช้งาน ซึ่งมีความคาดหวังว่า ถ้านำอุปกรณ์มาเสียบเมื่อไรก็สามารถใช้งานได้ทันที ซึ่งโดยรายละเอียดแล้ว สภาพที่จะทำให้อุปกรณ์ USB ไม่ประสบความสำเร็จในการใช้ประโยชน์นั้น ก็คือ การขาดไฟเลี้ยง การขาดแคลนช่องสัญญาณ และการมีรูปแบบการเชื่อมต่อ (Topology) ที่มีความลึก (Depth) มากเกินไป ซึ่งสภาพเหล่านี้จะถูกเข้าใจได้ดีโดยซอฟต์แวร์ของระบบ