อุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยอัลกอริทึมแบบทริบเปิลเดสโดยใช้เอฟพีจีเอ

Data Encryption and Decryption with Triple DES Algorithm on FPGA

นาย สุรพงษ์ สุฤทธิ์

นาย สุริยา กันทะวัง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

อุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยอัลกอริทึมแบบทริบเปิลเดสโดยใช้เอฟพีจีเอ

Data Encryption and Decryption with Triple DES Algorithm on FPGA

นาย สุรพงษ์ สุฤทธิ์

นาย สุริยา กันทะวัง

อาจารย์ที่ปรึกษา

อ. เจริญ วงษ์ชุ่มเย็น

อ. วัจนพงศ์ เกษมศิริ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง อุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยอัลกอริทึมแบบทริบเปิลเดสโดยใช้เอฟพีจีเอ

Data Encryption and Decryption with Triple DES Algorithm on FPGA

ผู้จัดทำ

1. นายสุรพงษ์ สุฤทธิ์ รหัสประจำตัว 45015393
2. นายสุริยา กันทะวัง รหัสประจำตัว 45015394

_____ อาจารย์ที่ปรึกษา

(อ. เจริญ วงษ์ชุ่มเย็น)

_____ อาจารย์ที่ปรึกษา

(อ. วัจนพงศ์ เกษมศิริ)

อุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยอัลกอริทึมแบบทริบเปิลเดสโดยใช้เอฟพีจีเอ

นาย สุรพงษ์ สุฤทธิ์ 45015393
นาย สุริยา กันทะวัง 45015394
อ. เจริญ วงษ์ชุ่มเย็น อาจารย์ที่ปรึกษา
อ. วัจนพงศ์ เกษมศิริ อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

โครงงานนี้เป็นการสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลโดยใช้ FPGA (Filed Programmable Gate Array) โดยมีจุดมุ่งหมายเพื่อเป็นต้นแบบในการพัฒนาอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยฮาร์ดแวร์ ซึ่งปัจจุบัน การเข้ารหัสและถอดรหัสข้อมูลทำด้วยซอฟท์แวร์เป็นส่วนมาก การทำโครงงานครั้งนี้เป็นการนำความรู้ที่ได้จากการศึกษาทฤษฎีมาสร้างให้เกิดรูปธรรม รายละเอียดของปริญญานิพนธ์จะนำเสนอทฤษฎีที่เกี่ยวข้อง เช่น การอัลกอริทึมของการเข้ารหัสและถอดรหัสข้อมูล การออกแบบวงจร การโปรแกรมด้วยภาษาวีเอชดีแอล การทดลองอุปกรณ์ที่สร้างขึ้น และผลการทดสอบการทำงานของอุปกรณ์ โดยบทสรุปจะชี้ให้เห็นว่า การสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลโดยใช้ FPGA สามารถทำได้ และให้ผลลัพธ์ถูกต้องตามทฤษฎี โดยแสดงผลการทดสอบในรูปแบบของการจำลองการทำงานของอุปกรณ์ และการทดสอบบนอุปกรณ์ที่สร้างขึ้น รวมทั้งจะนำเสนอปัญหาของการทำโครงงานครั้งนี้ซึ่งเป็นส่วนสำคัญส่วนหนึ่งที่น่าสนใจสำหรับผู้ที่ต้องการพัฒนา และสนใจงานด้านการออกแบบวงจร โดยใช้อุปกรณ์ประเภท FPGA

Data Encryption and Decryption with Triple DES Algorithm on FPGA

Mr.Surapong Surit
Mr.Suriya Kanthawang
Mr.Charoen Vongchumyen Advisor
Mr.Watjanapong Kasemsiri Advisor
Acadamic Year 2004

Abstract

This thesis present a design and implement Encrypter and Decrypter on FPGA (Filed Programmable Gate Array) .The goal of the thesis is development Encryption and Decryption by Hardware Device. Encryption and Decryption are most popular implement in software program. This thesis is using acknowledgement in Cryptography such as TDES algorithm . Using in Implementation and Design by VHDL language(VHSIC Hardware Description Language) . Simulation and Testing the project in simulater program tool (Modelsim SE Plus6.0) .

The summary in thesis explain Designing and Implementation Encryption and Decryption can Implement on FPGA and result of it corrected in TDES Algorithm on the simmulater program and implement Testing on FPGA.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความร่วมมือจากหลายๆ ฝ่ายด้วยกัน เพราะบุคคลที่เกี่ยวข้องที่จะกล่าวถึงต่อไปนี้ ล้วนแล้วแต่มีส่วนสำคัญที่ทำให้ปริญญานิพนธ์ฉบับนี้เสร็จลงได้ก็คือ อ.เจริญ วงษ์ชุ่มเย็น และ อ. วัจนพงศ์ เกษมศิริ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่คอยให้คำแนะนำ ให้โอกาสและให้ความรู้ในการทำโครงงาน เข้าใจปัญหาของงานและปัญหาของนักศึกษาเป็นอย่างดี

ต้องขอกราบขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ได้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูข้าพเจ้าเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ ดูแลเอาใจใส่ในทุกๆ ด้านเสมอมา พระคุณของท่านหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณครั้งนี้ไว้ตลอดไป

และสุดท้ายที่ต้องกล่าวถึงคือเพื่อน ๆ พี่ ที่คอยให้ความช่วยเหลือและให้คำปรึกษาแม้ว่าการทำปริญญานิพนธ์ครั้งนี้จะประสบปัญหามากมายเพียงใด เพื่อน และพี่ทุกคนก็ยังคอยให้กำลังใจเสมอมา จึงขอขอบคุณมา ณ ที่นี้

นาย สุรพงษ์ สุฤทธิ์

นาย สุริยา กันทะวัง

สารบัญ

สารบัญ (ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

สารบัญรูปภาพ (ต่อ)

หน้าที่

สารบัญตาราง

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและความเป็นมา

ปัจจุบันความปลอดภัยของข้อมูลถือเป็นสิ่งสำคัญในระบบคอมพิวเตอร์ ผู้ที่เป็นผู้ครอบครองข้อมูลก็หวังที่จะให้ข้อมูลของตนมีความปลอดภัยจากผู้ไม่ประสงค์ดี ในอีกมุมหนึ่ง ผู้ไม่ประสงค์ดีก็มุ่งที่จะก่อความเสียหาย เปลี่ยนแปลง ทำลายข้อมูล หรือล้วงความลับข้อมูลของผู้อื่นเพื่อประโยชน์ของตน ความปลอดภัยของข้อมูล ไม่ว่าจะเป็นการเก็บรักษาข้อมูล หรือการสื่อสารข้อมูลในระบบเครือข่ายก็ต้องมีความปลอดภัยเสมอ โดยเฉพาะในระบบฐานข้อมูลของธนาคารต้องมีความปลอดภัยเป็นอย่างยิ่ง ดังนั้นผู้ครอบครองข้อมูลต้องหาวิธีการใดๆ เพื่อให้ข้อมูลของตนมีความปลอดภัยสูงสุด การเข้ารหัสและถอดรหัสข้อมูล เป็นวิธีการหนึ่งที่นิยมใช้ในการรักษาความปลอดภัยของข้อมูล โดยสามารถทำได้ทั้งซอฟท์แวร์ และฮาร์ดแวร์

โครงงานนี้เป็นการนำความรู้ที่ได้จากการศึกษาทฤษฎีมาสร้างให้เกิดรูปธรรม โดยเป็นการสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลโดยใช้ FPGA และทดสอบให้เห็นว่าสามารถทำงานได้จริง จุดมุ่งหมายเพื่อเป็นต้นแบบในการพัฒนาอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยฮาร์ดแวร์ ซึ่งปัจจุบัน การเข้ารหัสและถอดรหัสข้อมูลทำด้วยซอฟท์แวร์เป็นส่วนมาก จึงจะเป็นสิ่งที่พิสูจน์ให้เห็นว่าการเข้ารหัสและถอดรหัสข้อมูลด้วยฮาดร์แวร์ สามารถทำได้และมีประสิทธิภาพไม่ด้อยไปกว่าการเข้ารหัสและถอดรหัสด้วยซอฟท์แวร์

1.2 วัตถุประสงค์

1.2.1 ศึกษาทฤษฎีการเข้ารหัสและถอดรหัสข้อมูลวิธีต่างๆ ที่ใช้ในปัจจุบัน

1.2.2 เพื่อศึกษาภาษารูปแบบการเขียนโปรแกรมด้วยภาษา VHDL (VHSIC Hardware Description Language) ในการออกแบบระบบฮาร์ดแวร์ดิจิตอล ตั้งแต่การออกแบบ แก้ไขตรวจสอบ จำลองการทำงาน และสังเคราะห์วงจรบนอุปกรณ์ประเภท FPGA

1.2.3 ศึกษาโครงสร้างและคุณสมบัติของอุปกรณ์ประเภท FPGA ในการสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูล รวมทั้งข้อจำกัดของอุปกรณ์ที่มีผลกระทบต่อการออกแบบ

1.2.4 นำทฤษฎีของการเข้ารหัสและถอดรหัสข้อมูลที่เหมาะสมที่สุดจากการศึกษา มาเป็นแนวทางการออกแบบวงจรเข้ารหัสและถอดรหัสข้อมูลโดยใช้ FPGA

1.2.5 เพื่อทดสอบการทำงานของอุปกรณ์ให้ได้ผลลัพธ์ตามทฤษฎี

1.2.6 เพื่อเป็นต้นแบบของการพัฒนาอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลโดยใช้ FPGA

1.3 ขอบเขตของโครงงาน

1.3.1 สร้างวงจรเข้ารหัสและถอดรหัส โดยใช้ FPGA

1.3.2 ทดสอบการทำงานและประสิทธิภาพของวงจรที่ออกแบบด้วยการจำลองการทำงาน

1.3.3 ทดสอบการทำงานของอุปกรณ์ให้ได้ผลลัพธ์ตามทฤษฎีทั้งการจำลองการทำงาน และทดลองบนชิ้นงาน

1.4 ขั้นตอนการดำเนินงาน

1.4.1 ศึกษาทฤษฎีที่เกี่ยวข้อง

- ศึกษาการทฤษฎีการเข้ารหัสและถอดรหัสข้อมูลวิธีต่างๆ ที่มีใช้ในปัจจุบัน และเลือกอัลกอริทึมที่เหมาะสมในการทำโครงงาน
- ศึกษาโครงสร้างและคุณสมบัติของอุปกรณ์ประเภท FPGA ในการสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูล รวมทั้งข้อจำกัดของอุปกรณ์ที่มีผลกระทบต่อการออกแบบ
- ศึกษาภาษารูปแบบการเขียนโปรแกรมด้วยภาษาวีเอชดีแอล ในการออกแบบระบบฮาร์ดแวร์ดิจิตอล ตั้งแต่การออกแบบ แก้ไขตรวจสอบ จำลองการทำงาน และสังเคราะห์วงจรบนอุปกรณ์ประเภท FPGA
- ทดลองเขียนโปรแกรมด้วยภาษาวีเอชดีแอล และสร้างวงจรลงในอุปกรณ์ FPGA

1.4.2 ออกแบบและทดลองวงจร

- ออกแบบวงจรการเข้ารหัสและถอดรหัสข้อมูลโดยใช้อัลกอริทึมที่ศึกษา
- เขียนโปรแกรมด้วยภาษาวีเอชดีแอล และสร้างวงจรลงในอุปกรณ์ FPGA
- ทดลองและทดสอบด้วยการจำลองการทำงาน
- ทดลองและทดสอบการทำงานของอุปกรณ์
- สรุปผลการทดลอง และจัดทำเอกสาร

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 เข้าใจถึงอัลกอริทึมของการเข้ารหัสและถอดรหัสข้อมูลด้วยวิธีต่างๆ

1.5.2 สามารถเขียนโปรแกรมภาษา VHDL ในการออกแบบอาร์ดแวร์ระบบดิจิตอล

1.5.3 เข้าใจถึงโครงสร้างและคุณสมบัติของอุปกรณ์ประเภท FPGA ในการสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูล รวมทั้งข้อจำกัดของอุปกรณ์ที่มีผลกระทบต่อการออกแบบ

1.5.4 สามารถสร้างวงจรเข้ารหัสและถอดรหัส โดยใช้ FPGA รวมทั้งทดสอบการทำงานของอุปกรณ์

1.5.5 อุปกรณ์สามารถเป็นต้นแบบของการพัฒนาต่อไปได้

บทที่ 2
การเข้ารหัสและถอดรหัสข้อมูล

การรักษาความมั่นคงปลอดภัยของข้อมูลในระบบคอมพิวเตอร์ หรือระบบเครือข่ายคอมพิวเตอร์ เป็นสิ่งที่ควรตระหนักเป็นอย่างยิ่งในปัจจุบัน เพราะโลกในยุคปัจจุบันเป็นโลกแห่งข้อมูลข่าวสาร การเก็บรักษาข้อมูลให้ปลอดภัยจึงเป็นสิ่งสำคัญกับตัวบุคคลและองค์กร เพราะฉะนั้นการที่จะอนุญาตให้บุคคลใดบุคคลหนึ่งสามารถเข้าถึงข้อมูลจึงเป็นสิ่งที่ควรระมัดระวัง เพราะข้อมูลบางอย่างของบุคคลและองค์กรมีความสำคัญและไม่สามารถเปิดเผยต่อบุคคลภายนอกได้ การปกป้องความมั่นคงปลอดภัยของระบบและข้อมูลภายในองค์กรถือเป็นเรื่องสำคัญในปัจจุบัน ทั้งนี้เนื่องจากการถูกคุกคามโดยผู้ไม่ประสงค์ดีหรือจากโปรแกรมบางประเภทได้เพิ่มมากขึ้นและอาจนำมาซึ่งความเสียหายอย่างมากต่อองค์กร ดังนั้นถ้าภายในระบบมีการรักษาความปลอดภัยที่ดีจะช่วยลดโอกาสเสี่ยงต่อการถูกคุกคามได้

การเข้ารหัสและถอดรหัสข้อมูล เป็นวิธีการหนึ่งที่นิยมใช้ในการรักษาความปลอดภัยของข้อมูล เนื้อหาของบทนี้ จะเสนอรายละเอียดที่เกี่ยวกับอัลกอริทึมของการเข้ารหัสและถอดรหัสข้อมูลเพื่อความปลอดภัยเท่านั้น ซึ่งเป็นเนื้อหาบางส่วนของการรักษาความปลอดภัยของข้อมูลระบบคอมพิวเตอร์ และเป็นเนื้อหาที่เกี่ยวข้องกับแนวทางการทำโครงงานครั้งนี้

2.1 จุดประสงค์ของการเข้ารหัสข้อมูล

2.1.1 การทำให้ข้อมูลเป็นความลับ (Confidentiality) เพื่อป้องกันไม่ให้ผู้ที่ไม่มีสิทธิ์ในการเข้าถึงข้อมูลสามารถเข้าถึงข้อมูลได้

2.1.2 การทำให้ข้อมูลสามารถตรวจสอบความสมบูรณ์ได้ (Integrity) เพื่อป้องกันข้อมูลให้อยู่ในสภาพเดิมอย่างสมบูรณ์ กล่าวคือ ในกระบวนการสื่อสารนั้นผู้รับ (Receiver) ได้รับข้อมูลที่ถูกต้องตามที่ผู้ส่ง (Sender) ส่งมาให้โดยข้อมูลจะต้องไม่มีการสูญหายหรือถูกเปลี่ยนแปลงแก้ไขใดๆ

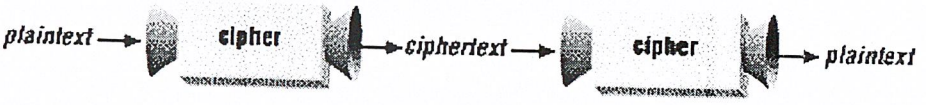
2.1.3 การทำให้สามารถพิสูจน์ตัวตนของผู้ส่งข้อมูลได้ (Authentication/Nonrepudiation) เพื่อให้สามารถตรวจสอบได้ว่าใครคือผู้ส่งข้อมูล หรือในทางตรงกันข้าม ก็คือเพื่อป้องกันการแอบอ้างได้

2.2 CRYPTOGRAPHY

Encryption มีความหมายว่า เป็นกรรมวิธีการข้อรหัสข้อมูลข่าวสาร ทำให้ความหมายของข่าวสารเดิมเปลี่ยนไป โดยไม่สามารถอ่านหรือทำความเข้าใจหรือตีความหมายของข้อความนั้นได้

Decryption มีความหมายว่า เป็นกรรมวิธีที่ตรงกันข้ามกัน ที่ใช้ในการแปลงข่าวสารจารการ Encryption ให้กลับมาเป็นข้อมูลดังเดิม

การเข้ารหัส และการถอดรหัสจะเรียกว่า ระบบสร้างรหัสลับ(Cryptosystem) ข้อมูลที่ถูกเข้ารหัสแล้วจะถูกเรียกว่า ข้อมูลเข้ารหัส หรือ Ciphertext ส่วนข้อมูลเดิมที่เป็นข้อมูลที่แท้จริงนั้นจะถูกเรียกว่า Pliantext

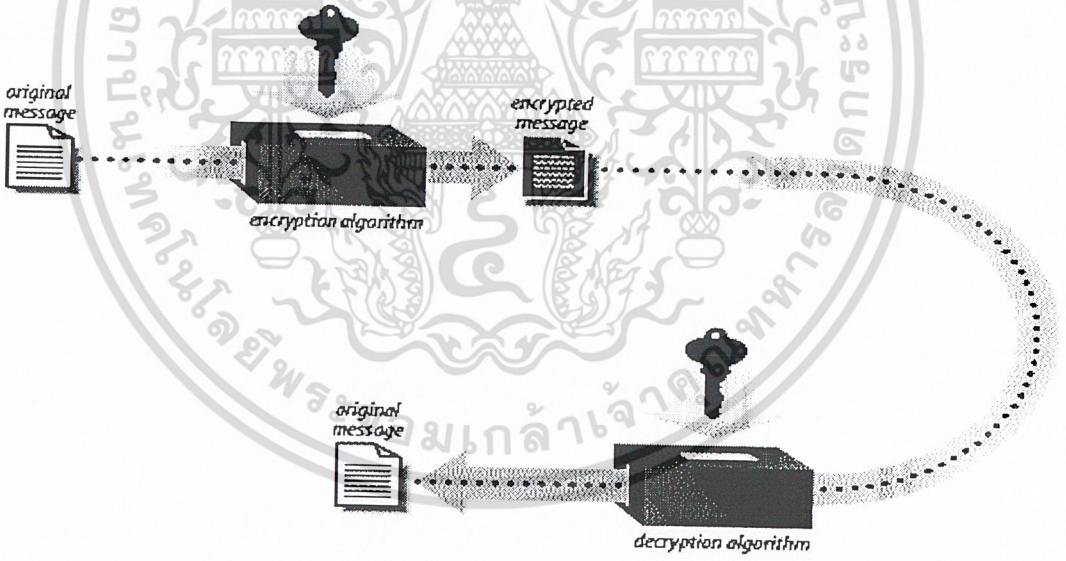


รูปที่ 2.1 Operation of a Cipher (encrypt and decrypt)

2.3 อัลกอริทึมในการเข้ารหัสข้อมูล

2.3.1 อัลกอริทึมแบบสมมาตร (Symmetric key algorithms) หรือ Secret Key Cryptography

อัลกอริทึมแบบนี้จะใช้กุญแจที่เรียกว่า กุญแจลับ (Secret key) ซึ่งมีเพียงหนึ่งเดียวเพื่อใช้ในการเข้ารหัสและถอดรหัสข้อความที่ส่งไป อัลกอริทึมยังสามารถแบ่งย่อยออกเป็น 2 ประเภท ได้แก่ แบบบล็อค (Block Algorithms) ซึ่งจะทำการเข้ารหัสทีละบล็อค (1 บล็อคประกอบด้วยหลายไบต์ เช่น 64 ไบต์ เป็นต้น) และแบบสตรีม (Stream Algorithms) ซึ่งจะทำการเข้ารหัสทีละไบต์อัลกอริทึมแบบนี้จะใช้กุญแจที่ละไบต์



รูปที่ 2.2 Symmetric key Cryptography

อัลกอริทึมสำหรับการเข้ารหัสแบบสมมาตรในปัจจุบันมีเป็นจำนวนมาก ซึ่งจะนำเสนอเพียงจำนวนหนึ่งซึ่งเป็นอัลกอริทึมที่เป็นที่รู้จักกันดีในวงการของการเข้ารหัสข้อมูล

❑ **อัลกอริทึม DES**

DES ย่อมาจาก Data Encryption Standard อัลกอริทึมนี้ได้รับการรับรองโดยรัฐบาลสหรัฐอเมริกาในปี ค.ศ. 1977 ให้เป็นมาตรฐานการเข้ารหัสข้อมูลสำหรับหน่วยงานของรัฐทั้งหมด ในปี 1981 อัลกอริทึมยังได้รับการกำหนดให้เป็นมาตรฐานการเข้ารหัสข้อมูลในระดับนานาชาติตามมาตรฐาน ANSI (American National Standards) อีกด้วย DES เป็นอัลกอริทึมแบบบล็อกซึ่งใช้กุญแจที่มีขนาดความยาว 56 บิตและเป็นอัลกอริทึมที่มีความแข็งแกร่ง แต่เนื่องด้วยขนาดความยาวของกุญแจที่มีขนาดเพียง 56 บิต ซึ่งในปัจจุบันถือได้ว่าสั้นเกินไป ผู้บุกรุกอาจใช้วิธีการลองผิดลองถูกเพื่อค้นหากุญแจที่ถูกต้องสำหรับการถอดรหัสได้ ในปี 1998 ได้มีการสร้างเครื่องคอมพิวเตอร์พิเศษขึ้นมาซึ่งมีมูลค่า 250,000 เหรียญสหรัฐเพื่อใช้ในการค้นหากุญแจที่ถูกต้องของการเข้ารหัสข้อมูลหนึ่งๆ ด้วย DES และพบว่าเครื่องคอมพิวเตอร์นี้สามารถค้นหากุญแจที่ถูกต้องได้ภายในระยะเวลาไม่ถึงหนึ่งวัน

❑ **อัลกอริทึม Triple-DES**

Triple-DES เป็นอัลกอริทึมที่เสริมความปลอดภัยของ DES ให้มีความแข็งแกร่งมากขึ้นโดยใช้อัลกอริทึม DES เป็นจำนวนสามครั้งเพื่อทำการเข้ารหัส แต่ละครั้งจะใช้กุญแจในการเข้ารหัสที่แตกต่างกัน ดังนั้นจึงเปรียบเสมือนการใช้กุญแจเข้ารหัสที่มีความยาวเท่ากับ 56*3 = 168 บิต Triple-DES ได้ถูก ใช้งานกับสถาบันทางการเงินอย่างแพร่หลาย รวมทั้งใช้งานกับโปรแกรม Secure Shell (ssh) ด้วยการใช้อัลกอริทึม DES เพื่อเข้ารหัสเป็นจำนวนสองครั้งด้วยกุญแจสองตัว (56*2=112 บิต) ยังถือได้ว่าไม่ปลอดภัยอย่างพอเพียง

❑ **อัลกอริทึม Blowfish**

Blowfish เป็นอัลกอริทึมที่มีความรวดเร็วในการทำงาน มีขนาดเล็กกระทัดรัด และใช้การเข้ารหัสแบบบล็อค ผู้พัฒนาคือ Bruce Schneier อัลกอริทึมสามารถใช้กุญแจที่มีขนาดความยาวตั้งแต่ไม่มากนักไปจนถึงขนาด 448 บิต ซึ่งทำให้เกิดความยืดหยุ่นสูงในการเลือกใช้กุญแจ รวมทั้งอัลกอริทึมยังได้รับการออกแบบมาให้ทำงานอย่างเหมาะสมกับหน่วยประมวลผลขนาด 32 หรือ 64 บิต Blowfish ได้เปิดเผยสู่สาธารณะและไม่ได้มีการจดสิทธิบัตรใดๆ นอกจากนั้นยังใช้ในโปรแกรม SSH และอื่นๆ

❑ **อัลกอริทึม IDEA**

IDEA ย่อมาจาก International Data Encryption Algorithm อัลกอริทึมนี้ได้รับการพัฒนาในประเทศสวิสเซอร์แลนด์ที่เมือง Zarich โดย James L. Massey และ Xuejia Lai และได้รับการตีพิมพ์เผยแพร่ในปี ค.ศ. 1990 อัลกอริทึมใช้กุญแจที่มีขนาด 128 บิต และได้รับการใช้งานกับโปรแกรมยอดฮิตสำหรับการเข้ารหัสและลงลายมือชื่ออิเล็กทรอนิกส์ในระบบอีเมล์ที่มีชื่อว่า PGP ต่อมา IDEA ได้รับการจดสิทธิบัตรทางด้านซอฟต์แวร์โดยบริษัท Ascom-Tech AG ในประเทศสวิสเซอร์แลนด์ ซึ่งทำให้การนำไปใช้ในงานต่างๆ เริ่มลดลง ทั้งนี้เนื่องจากติดปัญหาเรื่องลิขสิทธิ์นั่นเอง

❑ **อัลกอริทึม RC4**

อัลกอริทึมนี้เป็นอัลกอริทึมแบบสตรีม (ทำงานกับข้อมูลทีละไบต์) ซึ่งได้รับการพัฒนาขึ้นมาโดย Ronald Riverst และถูกเก็บเป็นความลับทางการค้าโดยบริษัท RSA Data Security ในภายหลังอัลกอริทึมนี้

ได้รับการเปิดเผยใน Usenet เมื่อปี ค.ศ. 1994 และเป็นที่ทราบกันว่าเป็นอัลกอริทึมที่มีความแข็งแกร่งโดย สามารถใช้ขนาดความยาวของกุญแจที่มีขนาดตั้งแต่ 1 บิตไปจนกระทั่งถึงขนาด 2048 บิต

- **อัลกอริทึม Rijndael (หรืออัลกอริทึม AES)**

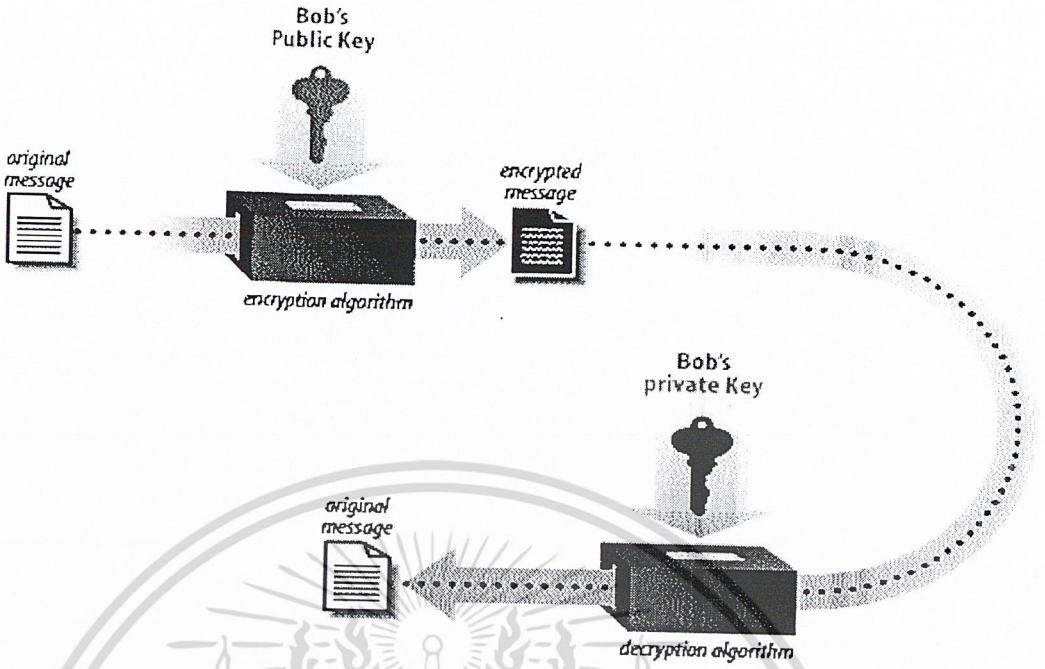
อัลกอริทึมนี้ได้รับการพัฒนาโดย Joan Daemen และ Vincent Rijmen ในปี 2000 อัลกอริทึมได้รับการคัดเลือกโดยหน่วยงาน National Institute of Standard and Technology (NIST) ของสหรัฐอเมริกา ให้เป็นมาตรฐานในการเข้ารหัสชั้นสูงของประเทศ อัลกอริทึมมีความเร็วสูงและมีขนาดกะทัดรัดโดย สามารถใช้กุญแจที่มีความยาวขนาด 128, 192 และ 256 บิต

- **อัลกอริทึม One-time Pads**

อัลกอริทึมนี้ได้รับการยอมรับว่าเป็นอัลกอริทึมที่ไม่มีใครสามารถเจาะความแข็งแกร่งของอัลกอริทึมได้ อัลกอริทึมใช้กุญแจที่มีขนาดความยาวซึ่งอาจจะมากกว่าขนาดความยาวของข้อความที่ต้องการเข้ารหัส กุญแจจะถูกสร้างออกมาแบบสุ่มและโดยปกติจะถูกใช้งานแค่เพียงครั้งเดียวแล้วทิ้งไป แต่ละไบต์ของข้อความที่ต้องการส่งไปจะถูกเข้าและถอดรหัสโดยหนึ่งไบต์ (ชนิดไบต์ต่อไบต์) ของกุญแจที่ถูกสร้างขึ้นมาใช้งาน เนื่องจากกุญแจที่ถูกใช้งานแต่ละครั้งจะไม่ซ้ำกันและถูกสร้างขึ้นมาแบบสุ่ม จึงเป็นการยากที่จะค้นหากุญแจที่ถูกต้องได้ข้อจำกัดของอัลกอริทึมนี้ คือขนาดของกุญแจที่อาจมีขนาดยาวกว่าข้อความที่ต้องการส่ง ซึ่งส่งผลให้การส่งมอบกุญแจที่มีขนาดใหญ่ทำได้ไม่สะดวกนัก รวมทั้งการสร้างกุญแจให้มีความสุ่มสูงไม่ใช่เป็นสิ่งที่ทำได้ง่ายนัก อย่างไรก็ตามอัลกอริทึมนี้ก็ยังมีการใช้งานในระบบเครือข่ายที่ต้องการความปลอดภัยสูง

2.3.2 อัลกอริทึมแบบอสมมาตร (Asymmetric key algorithms) หรือ Public Key Cryptogryphy

อัลกอริทึมนี้จะใช้กุญแจสองตัวเพื่อทำงาน ตัวหนึ่งใช้ในการเข้ารหัสและอีกตัวหนึ่งใช้ในการถอดรหัสข้อมูลที่เข้ารหัสมาโดยกุญแจตัวแรก อัลกอริทึมกลุ่มสำคัญในแบบอสมมาตรนี้คือ อัลกอริทึมแบบกุญแจสาธารณะ (Public keys Algorithms) ซึ่งใช้กุญแจที่เรียกกันว่า กุญแจสาธารณะ (Public keys) ในการเข้ารหัสและใช้กุญแจที่เรียกกันว่า กุญแจส่วนตัว (Private keys) ในการถอดรหัสข้อมูลนั้น กุญแจสาธารณะนี้สามารถส่งมอบให้กับผู้อื่นได้ เช่น เพื่อนร่วมงานที่เราต้องการติดต่อด้วย หรือแม้กระทั่งวางไว้บนเว็บไซต์เพื่อให้ผู้อื่นสามารถดาวน์โหลดไปใช้งานได้ สำหรับกุญแจส่วนตัวนั้นต้องเก็บไว้กับผู้เป็นเจ้าของกุญแจส่วนตัวเท่านั้นและห้ามเปิดเผยให้ผู้อื่นทราบโดยเด็ดขาด



รูปที่ 2.3 Public key cryptography

อัลกอริทึมแบบกุญแจสาธารณะยังสามารถประยุกต์ใช้ได้กับการลงลายมือชื่ออิเล็กทรอนิกส์ (ซึ่งเปรียบเสมือนการลงลายมือชื่อของเราที่ใช้กับเอกสารสำนักงานทั่วไป) การลงลายมือชื่อนี้จะเป็นการพิสูจน์ความเป็นเจ้าของและสามารถใช้ได้กับการทำธุรกรรมต่างๆ บนอินเทอร์เน็ต เช่น การซื้อสินค้า เป็นต้น วิธีการใช้งานคือ ผู้เป็นเจ้าของกุญแจส่วนตัวลงลายมือชื่อของตนกับข้อความที่ต้องการส่งไปด้วยกุญแจส่วนตัว แล้วจึงส่งข้อความนั้นไปให้กับผู้รับ เมื่อได้รับข้อความที่ลงลายมือชื่อมา ผู้รับสามารถใช้กุญแจสาธารณะ (ที่เป็นคู่ของกุญแจส่วนตัวนั้น) เพื่อตรวจสอบว่าเป็นข้อความที่มาจากผู้ส่งนั้นหรือไม่ อัลกอริทึมแบบกุญแจสาธารณะ แบ่งตามลักษณะการใช้งาน ได้เป็น 2 ประเภท คือ ใช้สำหรับการเข้ารหัส และใช้สำหรับการลงลายมือชื่ออิเล็กทรอนิกส์ อัลกอริทึมที่เป็นที่รู้จักกันดีมีดังนี้

- **อัลกอริทึม RSA**

อัลกอริทึม RSA ได้รับการพัฒนาขึ้นที่มหาวิทยาลัย MIT ในปี 1977 โดยศาสตราจารย์ 3 คน ซึ่งประกอบด้วย Ronald Rivest, Adi Shamir และ Leonard Adleman ชื่อของอัลกอริทึมได้รับการตั้งชื่อตามตัวอักษรตัวแรกของนามสกุลของศาสตราจารย์ทั้งสามคน อัลกอริทึมนี้สามารถใช้ในการเข้ารหัสข้อมูลรวมทั้งการลงลายมือชื่ออิเล็กทรอนิกส์ด้วย

- **อัลกอริทึม DSS**

DSS ย่อมาจาก Digital Signature Standard อัลกอริทึมนี้ได้รับการพัฒนาขึ้นมาโดย National Security Agency ในประเทศสหรัฐอเมริกาและได้รับการรับรองโดย NIST ให้เป็นมาตรฐานกลางสำหรับการลงลายมือชื่ออิเล็กทรอนิกส์ในประเทศสหรัฐอเมริกา

2.3.3 อัลกอริทึมแบบ HASH (เมสเซสไดเจสต์)

เมสเซสไดเจสต์ (Message Digest) หรือเรียกสั้นๆ ว่าไดเจสต์ แปลว่าข้อความสรุปจากเนื้อหาข้อความตั้งต้น โดยปกติข้อความสรุปจะมีความยาวน้อยกว่าความยาวของข้อความตั้งต้นมาก จุดประสงค์สำคัญของอัลกอริทึมนี้คือ การสร้างข้อความสรุปที่สามารถใช้เป็นตัวแทนของข้อความตั้งต้นได้ โดยทั่วไปข้อความสรุปจะมีความยาวอยู่ระหว่าง 128 ถึง 256 บิต และจะไม่ขึ้นกับขนาดความยาวของข้อความตั้งต้น คุณสมบัติที่สำคัญของอัลกอริทึมสำหรับสร้างไดเจสต์มีดังนี้

- ทุกๆ บิตของไดเจสต์จะขึ้นอยู่กับทุกบิตของข้อความตั้งต้น
- ถ้าบิตใดบิตหนึ่งของข้อความตั้งต้นเกิดการเปลี่ยนแปลง เช่น ถูกแก้ไข ทุกๆ บิตของไดเจสต์จะมีโอกาสร้อยละ 50 ที่จะแปรเปลี่ยนค่าไปด้วย ซึ่งหมายถึงว่า 0 เปลี่ยนค่าเป็น 1 และ 1 เปลี่ยนเป็น 0 คุณสมบัติข้อนี้สามารถอธิบายได้ว่าการเปลี่ยนแปลงแก้ไขข้อความตั้งต้นโดยผู้ไม่ประสงค์ดีแม้ว่าอาจแก้ไขเพียงเล็กน้อยก็ตาม เช่น เพียง 1 บิตเท่านั้น ก็จะส่งผลให้ผู้รับข้อความทราบว่าข้อความที่ตนได้รับไม่ใช่ข้อความตั้งต้น (โดยการนำข้อความที่ตนได้รับเข้าอัลกอริทึมเพื่อทำการคำนวณหาไดเจสต์ออกมา แล้วจึงเปรียบเทียบไดเจสต์ที่คำนวณได้กับไดเจสต์ที่ส่งมาให้ด้วย ถ้าต่างกัน แสดงว่าข้อความที่ได้รับนั้นถูกเปลี่ยนแปลงแก้ไข) โอกาสที่ข้อความตั้งต้น 2 ข้อความใดๆ ที่มีความแตกต่างกัน จะสามารถคำนวณได้ค่าไดเจสต์เดียวกันมีโอกาสน้อยมากคุณสมบัติข้อนี้ทำให้แน่ใจได้ว่า เมื่อผู้ไม่ประสงค์ดีทำการแก้ไขข้อความตั้งต้น ผู้รับข้อความที่ถูกแก้ไขไปแล้วนั้น จะสามารถตรวจพบได้ถึงความผิดปกติที่เกิดขึ้นอย่างแน่นอนอัลกอริทึมสำหรับสร้างไดเจสต์ยอดนิยมมีดังนี้

❑ **อัลกอริทึม MD2**

ผู้พัฒนาคือ Ronald Rivest อัลกอริทึมนี้เชื่อกันว่ามีความแข็งแกร่งที่สุดในบรรดาอัลกอริทึมต่างๆ ที่ Rivest พัฒนาขึ้นมา (ความแข็งแกร่งพิจารณาได้จากคุณสมบัติสามประการข้างต้น) ข้อเสียของอัลกอริทึมนี้คือใช้เวลามากในการคำนวณไดเจสต์หนึ่งๆ MD2 จึงไม่ค่อยได้มีการใช้งานกันมากนัก MD2 สร้างไดเจสต์ที่มีความยาว 128 บิต

❑ **อัลกอริทึม MD4**

ผู้พัฒนาคือ Rivest เช่นเดียวกับ MD2 อัลกอริทึมนี้พัฒนาขึ้นมาเพื่อแก้ปัญหาความล่าช้าในการคำนวณของ MD2 อย่างไรก็ตามในภายหลังได้พบว่าอัลกอริทึมมีข้อบกพร่องที่เกี่ยวข้องกับคุณสมบัติข้อที่สามโดยตรง กล่าวคือปัญหาการชนกันของไดเจสต์มีโอกาสเกิดขึ้นได้ไม่น้อย ซึ่งผู้บุกรุกอาจใช้ประโยชน์จากจุดอ่อนนี้เพื่อทำการแก้ไขข้อความตั้งต้นที่ส่งมาให้ได้ MD4 ผลิตไดเจสต์ที่มีขนาด 128 บิต

❑ **อัลกอริทึม MD5**

Rivest เป็นผู้พัฒนาเช่นกันโดยพัฒนาต่อจาก MD4 เพื่อให้มีความปลอดภัยที่สูงขึ้น ถึงแม้จะเป็นที่นิยมใช้งานกันอย่างแพร่หลาย ทว่าในปี 1996 ก็มีผู้พบจุดบกพร่องของ MD5 (เช่นเดียวกับ MD4) จึงทำให้ความนิยมเริ่มลดลง MD5 ผลิตไดเจสต์ที่มีขนาด 128 บิต

❑ **อัลกอริทึม SHA**

SHA ย่อจาก Secure Hash Algorithm อัลกอริทึม SHA ได้รับแนวคิดในการพัฒนามาจาก MD4 และได้รับการพัฒนาขึ้นมาเพื่อใช้งานร่วมกับอัลกอริทึม DSS (ซึ่งใช้ในการลงลายมือชื่ออิเล็กทรอนิกส์) หลังจากที่ได้มีการตีพิมพ์เผยแพร่อัลกอริทึมนี้ได้ไม่นาน NIST ก็ประกาศตามมาว่าอัลกอริทึมจำเป็นต้องได้รับการแก้ไขเพิ่มเติมเล็กน้อยเพื่อให้สามารถใช้งานได้อย่างเหมาะสม SHA สร้างไดเจสต์ที่มีขนาด 160 บิต

❑ **อัลกอริทึม SHA-1**

SHA-1 เป็นอัลกอริทึมที่แก้ไขเพิ่มเติมเล็กน้อยจาก SHA การแก้ไขเพิ่มเติมนี้เป็นที่เชื่อกันว่าทำให้อัลกอริทึม SHA-1 มีความปลอดภัยที่สูงขึ้น SHA-1 สร้างไดเจสต์ที่มีขนาด 160 บิต

❑ **อัลกอริทึม SHA-256, SHA-384 และ SHA-512**

NIST เป็นผู้นำเสนออัลกอริทึมทั้งสามนี้ในปี 2001 เพื่อใช้งานร่วมกับอัลกอริทึม AES (ซึ่งเป็นอัลกอริทึมในการเข้ารหัสแบบสมมาตร) อัลกอริทึมเหล่านี้สร้างไดเจสต์ที่มีขนาด 256, 384 และ 512 บิต ตามลำดับ

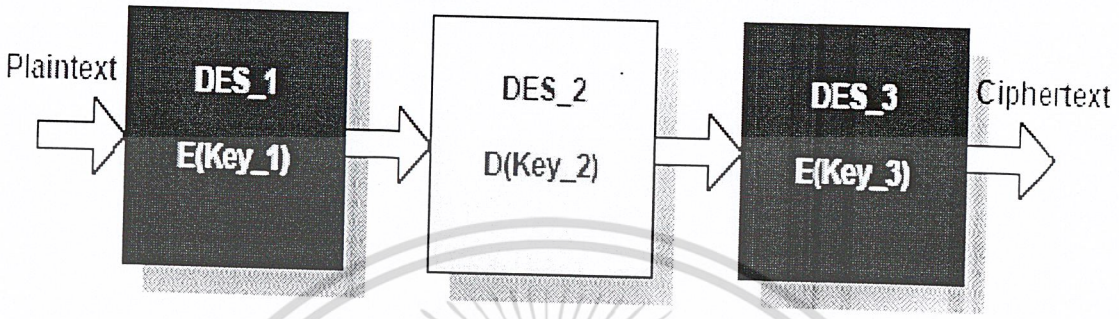
นอกจากอัลกอริทึมสำหรับการสร้างไดเจสต์ที่กล่าวถึงไปแล้วนั้น อัลกอริทึมสำหรับการเข้ารหัสแบบสมมาตร เช่น DES สามารถใช้ในการสร้างไดเจสต์เช่นกัน วิธีการใช้งานอัลกอริทึมแบบสมมาตรเพื่อสร้างไดเจสต์คือ ให้เลือกกุญแจลับสำหรับการเข้ารหัสขึ้นมา 1 กุญแจโดยวิธีการเลือกแบบสุ่ม และต่อมาใช้กุญแจนี้เพื่อเข้ารหัสข้อความตั้งต้น แล้วใช้เฉพาะบล็อกสุดท้ายที่เข้ารหัสแล้วเพื่อเป็นไดเจสต์ของข้อความทั้งหมด (ไม่รวมบล็อคอื่นๆ ที่เข้ารหัสแล้ว) อัลกอริทึมแบบสมมาตรสามารถสร้างไดเจสต์ที่มีคุณภาพดี แต่ข้อเสียคือต้องใช้เวลาในการคำนวณไดเจสต์มาก ไดเจสต์เป็นเครื่องมือที่สำคัญที่สามารถใช้ในการตรวจสอบว่า ไฟล์ในระบบที่ใช้งานมีการเปลี่ยนแปลงแก้ไขหรือไม่ (ไม่ว่าจะโดยเจตนาหรือไม่ก็ตาม) บางครั้งการเปลี่ยนแปลงแก้ไขอาจถูกกระทำโดยผู้ที่ไม่มีสิทธิ์ เช่น ผู้บุกรุก เป็นต้น วิธีการใช้ไดเจสต์เพื่อตรวจสอบไฟล์ในระบบคือให้เลือกใช้อัลกอริทึมหนึ่ง เช่น MD5 เพื่อสร้างไดเจสต์ของไฟล์ในระบบและเก็บไดเจสต์นั้นไว้อีกที่หนึ่งนอกระบบ ภายหลังจากระยะเวลาหนึ่งที่กำหนดไว้ เช่น 1 เดือน ก็มาคำนวณไดเจสต์ของไฟล์เดิมอีกครั้งหนึ่ง แล้วเปรียบเทียบไดเจสต์ใหม่นี้กับไดเจสต์ที่เก็บไว้นอกระบบว่าตรงกันหรือไม่ ถ้าตรงกัน ก็แสดงว่าไฟล์ในระบบยังเป็นปกติเช่นเดิม ไดเจสต์ยังเป็นส่วนหนึ่งของการลงลายมือชื่ออิเล็กทรอนิกส์ กล่าวคือการลงลายมือชื่ออิเล็กทรอนิกส์ในปัจจุบันจะใช้การลงลายมือชื่อกับไดเจสต์ของข้อความตั้งต้นแทนการลงลายมือชื่อกับข้อความตั้งต้นทั้งข้อความ

2.4 อัลกอริทึมที่เลือกในการ Implement บน FPGA

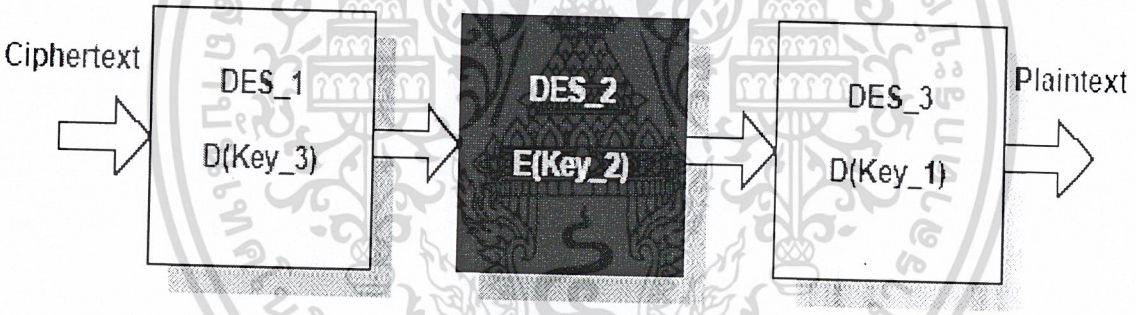
Triple-DES (TDES หรือ 3DES)

Triple - DES เป็นอัลกอริทึมที่เสริมความปลอดภัยของ DES ให้มีความแข็งแกร่งมากขึ้นโดยใช้อัลกอริทึม DES เป็นจำนวนสามครั้งเพื่อทำการเข้ารหัส แต่ละครั้งจะใช้กุญแจในการเข้ารหัสที่แตกต่างกัน ดังนั้นจึงเปรียบเสมือนการใช้กุญแจเข้ารหัสที่มีความยาวเท่ากับ 56*3 = 168 บิต Triple-DES ได้ถูก ใช้งานกับสถาบันทางการเงินอย่างแพร่หลาย รวมทั้งใช้งานกับโปรแกรม Secure Shell (ssh) ด้วย

การใช้อัลกอริทึม DES เพื่อเข้ารหัสเป็นจำนวนสองครั้งด้วยกุญแจสองตัว (56*2=112 บิต) ยังถือได้ว่าไม่ปลอดภัยอย่างพอเพียง โดยทั่วไปแล้วจะใช้งาน Triple-DES ลักษณะดังนี้



รูปที่ 2.4 Triple DES Encryption Flow Diagram



รูปที่ 2.5 Triple DES Decryption Flow Diagram

รายละเอียดของ Triple DES ในส่วนของการทำงานของแต่ละบล็อกไดอะแกรม จะได้กล่าวถึงในคราวเดียวในบทที่ 5 เป็นเรื่องของการออกแบบการเข้ารัหสและถอดรหัสข้อมูล โดยจะกล่าวถึงภาพรวมของ Triple DES จากนั้นจะแยก DES ออกมาเพื่ออธิบายว่า Component ภายในที่ประกอบอยู่ภายในมีอะไรบ้าง ทำหน้าที่อย่างไร ซึ่งจะได้ทราบโดยละเอียด

บทที่ 3
ภาษาวีเอชดีแอล

ความซับซ้อนในระบบดิจิตอลในปัจจุบันได้เพิ่มมากขึ้นทุกขณะ ส่งผลให้มีการนำคอมพิวเตอร์เพื่อช่วยใน การออกแบบหรือ CAD มาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มขึ้นเช่นกัน อีกทั้งอุปกรณ์และวิธีการ ออกแบบใหม่ๆ ก็ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบมากขึ้นด้วย สำหรับภาษาบรรยายอุปกรณ์ฮาร์ดแวร์ (HDL : Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่อง เพื่อช่วยให้การปรับปรุงขบวนการออกแบบระบบดิจิตอลเป็นไปอย่างมีประสิทธิภาพ

3.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และ สามารถเขียนได้หลายรูปแบบ ด้วยเหตุผลนี้จึงทำให้ภาษา VHDL เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกร ได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชันการทำงานของวงจรอย่างสังเขป โดยยังไม่ต้องคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้น VHDL ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มี ความซับซ้อนได้ทั้งหมด ดังนั้น VHDL จึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง วิวัฒนาการของภาษา VHDL เริ่มต้นประมาณปี ค.ศ. 1981 เมื่อกระทรวงกลาโหมสหรัฐอเมริกา หรือ DoD (Department of Defense) ได้พยายามปรับปรุงอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร ให้มีความทันสมัยมากขึ้น ประกอบกับเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์มีการพัฒนาไปอย่างรวดเร็วดังจะเห็นได้ จากการนำวงจรดิจิตอลหลายๆ วงจรมาทำการผลิตอยู่บนแผ่นซิลิกอนที่มีพื้นที่เพียง 1 - 2 ตารางเซนติเมตรเท่านั้น ซึ่ง เป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้นตลอดจนความน่าเชื่อถือ ในการทำงานและความคงทนต่อสภาพ แวดล้อมสูง แต่เนื่องจากในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ ต้องใช้วิศวกร และเวลาในดำเนินการมาก ฉะนั้นทาง DoD จึงจัดตั้งโครงการขึ้นมาเพื่อศึกษาวิธีการที่ช่วยในการพัฒนา วงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น ซึ่งโครงการดังกล่าวมีชื่อว่า "Very High Speed Integrated Circuits" หรือ VHSIC โดยในระยะแรกนั้นโครงการนี้ถือเป็นความลับทาง ด้านความมั่งคงของประเทศ และอยู่ภายใต้ความควบคุมดูแลของ United States International Traffic and Arms Regulations (ITAR) สำหรับมาตรฐานของภาษาที่ใช้บรรยาย พฤติกรรมวงจรหรือฮาร์ดแวร์ของระบบ สำหรับโครงการ VHSIC ที่ DoDได้ให้ไว้สามารถสรุปได้ดังนี้

1. ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่อง คอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก

2. สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้

3. ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่นเดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรม ภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า "Hardware Description Language" หรือ HDL

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

3.2 ข้อกำหนดของภาษา VHDL

DoD ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ในเดือนมกราคมปี ค.ศ.1983 ไว้ดังนี้

3.2.1 ลักษณะทั่วไป

DoD ได้กำหนดให้ VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถ ในการอธิบายและออกแบบในระดับสูง การจำลอง (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับบนซึ่งก็คือระบบจนถึง ระดับเกทอีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลนั้น ทุกๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่ จะทำงานไปพร้อมๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งของ VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์นั้นความพร้อมเพรียงจะหมายถึงทุกๆ คำสั่ง องค์ประกอบ เกทหรือวงจรลอจิกจะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในที่สุดแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไป พร้อมๆ กัน)

3.2.2 สนับสนุนการออกแบบแบบลำดับขั้น

การออกแบบแบบลำดับขั้น เป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบระบบที่มีหลายๆ ระดับ โดยในการ ออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อ และส่วนการบรรยายหน้าที่การทำงาน ซึ่งหน้าที่การทำงาน ของระบบสามารถกำหนดได้ด้วยตัวเอง หรืออาจถูกกำหนดโดยโครงสร้างที่

ประกอบด้วยองค์ประกอบย่อยๆ ลง ไปได้เช่นกัน แต่ที่ระดับล่างสุด องค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเอง และไม่สามารถกำหนด การทำงานโดยลักษณะแบบโครงสร้างได้

3.2.3 ไลบรารี

VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของ อุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูก ต้องควรจัดเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้ว เพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไป ใช้ได้ด้วย

3.2.4 ลำดับคำสั่ง

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษา เองก็ยังมีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบ ที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายใน ของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if - then - else และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้ สะดวกและง่ายขึ้น อย่างไรก็ตาม โครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

3.2.5 การกำหนดคุณสมบัติ

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

3.2.6 ชนิดของข้อมูล

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

3.2.7 โปรแกรมย่อย

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

3.2.8 การควบคุมเวลา

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลา

ที่แน่นอนหรือกำหนดให้มีการรอคอย เหตุการณ์ (Event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

3.2.9 การกำหนดแบบโครงสร้าง

การกำหนดโครงสร้างขององค์ประกอบต่างๆ สามารถกระทำได้ในทุกระดับของการออกแบบ โดยการกำหนดโครง สร้างขององค์ประกอบร่วมที่เกิดจากองค์ประกอบย่อยซึ่งแตกต่างกันหรือ เหมือนกันก็เป็นข้อกำหนดอย่างหนึ่งของ VHDL เช่นกัน

3.3 ส่วนประกอบต่างๆ ของภาษา VHDL

ในการเขียนรูปแบบบรรยายระบบดิจิตอลในมุมมองของการออกแบบลักษณะบนลงล่าง จะต้องทำความเข้าใจในเรื่องของโครงสร้างและส่วนประกอบต่างๆ ของรูปแบบภาษาวีเอชดีแอลเสียก่อน ซึ่งส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียนมี 4 หน่วยคือ

- หน่วยการออกแบบเอนทิตี (Entity Design Unit)
- หน่วยการออกแบบสถาปัตยกรรม (Architecture Design Unit)
- หน่วยการออกแบบแพ็กเก็ต (Package Design Unit)
- หน่วยการออกแบบโครงแบบ (Configuration Design Unit)

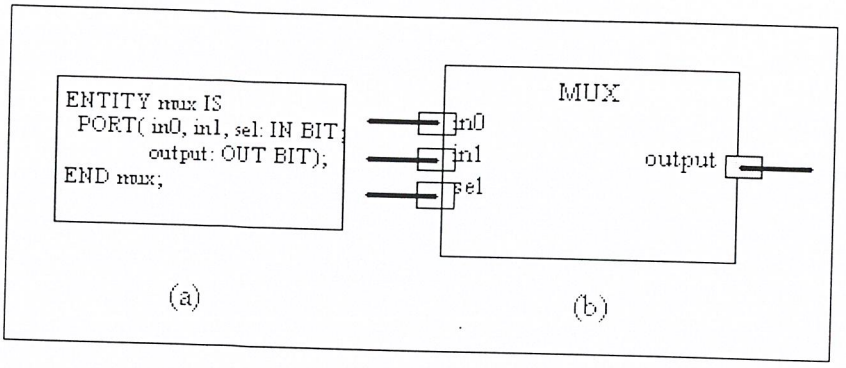
3.3.1 หน่วยการออกแบบเอนทิตี

หน่วยการออกแบบนี้เป็นส่วนที่ใช้สำหรับติดต่อระหว่างโลกภายนอกกับรูปแบบที่เขียนขึ้น ที่เรียกว่า หน่วยการออกแบบเอนทิตี ในส่วนนี้ใช้กำหนดจุดเชื่อมต่อ ของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆ ของข้อมูลที่ไหล่ผ่านจุดต่อเหล่านั้น รูปที่ 3.1 แสดงให้เห็นโครงสร้างอย่างง่ายๆ ของ หน่วยการออกแบบเอนทิตี

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

รูปที่ 3.1 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี

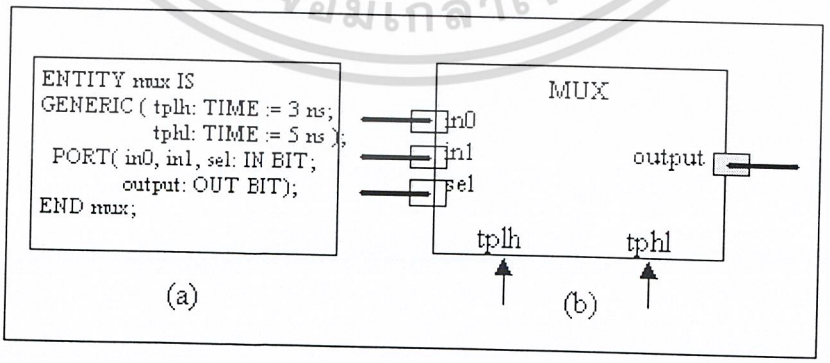
ส่วนนี้จะขึ้นต้นด้วยคำ ENTITY และ IS ระหว่างคำทั้งสองเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component_name) สำหรับการตั้งชื่อนั้นต้องเป็นไปตามกฎเกณฑ์ของภาษาหลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและออกของข้อมูล (input-output) รวมทั้งพารามิเตอร์อื่นๆ ส่วนนี้เรียกว่าส่วนหัว (entity header) และที่สำคัญคือ หน่วยการออกแบบเอนทิตีจะต้องปิดท้ายด้วยคำว่า END และเครื่องหมายอัฒภาคเสมอ (;)



รูปที่ 3.2 แสดงรูปแบบของมัลติเพลกซ์
(a) หน่วยการออกแบบเอนทิตีในรูปของวีเอชดีแอล
(b) มุมมองของตัวเชื่อมประสาน (Interfacing)

ในรูปที่ 3.2 เป็นหน่วยการออกแบบเอนทิตี ที่บรรยายอุปกรณ์ที่มีชื่อว่ามัลติเพลกซ์ หรือ MUX ในส่วนหัวของเอนทิตี มีการกำหนดจุดต่อ 4 จุดภายใต้ชุดคำสั่ง PORT โดยที่ 3 จุดแรกเป็นจุดให้ข้อมูลไหลผ่านเข้า ได้แก่ in0, in1, sel ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลเข้าของข้อมูล (IN) ที่แสดงด้วยรูปสี่เหลี่ยมโปร่งในรูปที่ 3.2 (b) ส่วนจุดเอาต์พุตเป็นจุดให้ข้อมูลไหลออก ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลออก (OUT) ที่แสดงด้วยรูปสี่เหลี่ยมทึบในรูปที่ 3.2 (b) ส่วนประเภทของข้อมูลที่จะไหล เข้าและออก นั้นเป็นประเภท BIT ที่สามารถมีค่าได้เพียงสองค่าคือ '0' และ '1' เท่านั้น

นอกจากนั้นผู้ออกแบบยังสามารถกำหนดค่าพารามิเตอร์ทางฟิสิกส์ที่เป็นข้อมูลเพิ่มเติมอื่นๆ ลงในส่วนหัวของเอนทิตีได้อีก เช่น ข้อมูลเกี่ยวกับความเร็วในการทำงานของอุปกรณ์ อันได้แก่ ค่าเวลาหน่วงแพร่กระจาย (Propagation delay time) พารามิเตอร์เหล่านี้ เรียกว่า เจนเนริก (Generic) ที่กำหนดด้วยคำสั่ง GENERIC จากตัวอย่างในรูปที่ 3.3



รูปที่ 3.3 รูปแบบมัลติเพลกซ์ที่ประกอบด้วยข้อมูลค่าเวลาหน่วงแพร่กระจาย
(a) หน่วยการออกแบบเอนทิตีในรูปของวีเอชดีแอล
(b) มุมมองของตัวเชื่อมประสาน

ในบางกรณีสามารถที่จะใช้ภาษาวีเอชดีแอล สร้างรูปแบบที่ปราศจากช่องทางไหล เข้าและออกของข้อมูล ได้ ซึ่งส่วนใหญ่จะพบในการสร้างรูปแบบ สำหรับตรวจสอบการทำงานของอีกรูปแบบหนึ่ง คือ วีเอชดีแอลสำหรับการทดสอบเปรียบเทียบ (Test bench)

```
ENTITY test_bench IS
END test_bench;
```

รูปที่ 3.4 หน่วยการออกแบบเอนทิตีที่ไม่มีการกำหนดช่องทางที่ต่อกับภายนอก

3.3.2 หน่วยการออกแบบสถาปัตยกรรม

คือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบ ในมุมมองของการจำลองการทำงาน พฤติกรรมต่างๆ ที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออก ตรงช่องทางตลอดจนพารามิเตอร์ต่างๆ ที่กำหนดใน หน่วยการออกแบบเอนทิตี รูปที่ 3.5 แสดงให้เห็นถึงโครงสร้างอย่างง่ายๆ ของหน่วยการออกแบบสถาปัตยกรรม

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

รูปที่ 3.5 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม

ส่วนของหน่วยการออกแบบสถาปัตยกรรม เริ่มต้นด้วยคำ ARCHITECTURE และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า architecture นั้นใช้บรรยายหน่วยการออกแบบเอนทิตีใดๆ (OF <entity design unit> IS) ส่วนที่อยู่ระหว่าง ARCHITECTURE และ BEGIN เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (architecture declarative area) ที่เป็นเพียงส่วนเผื่อเลือก (option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าต่างๆ ที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ อาทิเช่นประเภท (type) ต่างๆ (ตัวอย่างเช่น bit, bit_vector), สัญญาณ (signal), ตัวคงที่ (constant), โปรแกรมย่อย (ได้แก่ function และ procedure) และอุปกรณ์ (component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า และไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง PORT) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า BEGIN กับ END ของหน่วยการออกแบบสถาปัตยกรรม และนอก

จากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (concurrent statement) เท่านั้น หน่วยการออกแบบสถาปัตยกรรม จะต้องปิดท้ายด้วยคำสั่ง END และชื่อของสถาปัตยกรรมนั้นๆ ที่เป็นส่วนเผื่อเลือกโดยทั่วไปการเขียนรูปแบบระบบดิจิตอลด้วยภาษาวีเอชดีแอล สามารถเขียนได้ในลักษณะต่างๆ ดังนี้

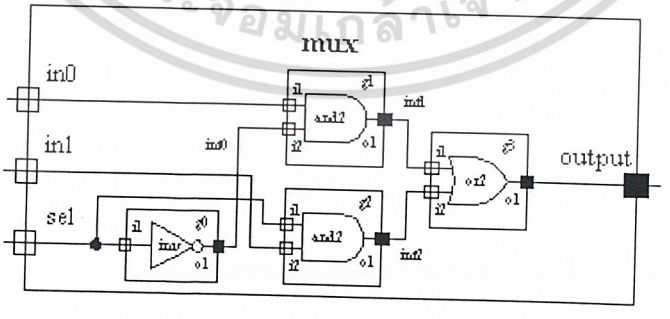
- ประเภทการไหลของข้อมูล (Dataflow description)
- ประเภทพฤติกรรม (Behavioral description)
- ประเภทโครงสร้าง (Structure description)
- ประเภทผสม (Mixed model description)

```
ARCHITECTURE data_flow  OF mux IS
BEGIN
    output <= ((NOT sel) AND in0) OR (sel AND in1);
END data_flow;
```

รูปที่ 3.6 แสดงหน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ตามฟังก์ชันบูลีน

$$output = (\overline{sel}.in0) + (sel.in1)$$

รูปที่ 3.6 ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (*in0, in1*) กับข้อมูลที่ไหลออก (*output*) ประกอบด้วยชุดคำสั่งแบบแข่งขนานเพียงชุดเดียว ซึ่งเขียนเป็นประเภทการไหลของข้อมูลของมัลติเพลกซ์ หรือ ระดับการถ่ายโอนข้อมูลระหว่างเรจิสเตอร์ (RTL: Register Transfer Level)



รูปที่ 3.7 แสดงโครงสร้างภายในสถาปัตยกรรมของมัลติเพลกซ์

รูปที่ 3.7 เป็นหน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทโครงสร้าง โดยใช้ อินเวอร์เตอร์ (inv ที่ตำแหน่ง g0), แอนด์เกต 2 อินพุตจำนวน 2 ตัว (and2 ที่ตำแหน่ง g1 และ g2) และออร์เกต 2 อินพุต (or2 ที่ตำแหน่ง g3) มาสร้างตามฟังก์ชันบูลีนของรูปที่ 3.6

```
ARCHITECTURE struc OF mux IS
  COMPONENT inv
  PORT ( i1 : IN BIT ; o1 : OUT BIT );
  COMPONENT and2
  PORT ( i1, i2 : IN BIT ; o3 : OUT BIT );
  COMPONENT or2
  PORT ( i1, i2 : IN BIT ; o1 : OUT BIT );
END COMPONENT;
  SIGNAL int0, int1, int2 : BIT;
BEGIN
  g0 : inv  PORT MAP (i1 => sel, o1 => int0);
  g1 : and2 PORT MAP (i1 => in0, i2 => int0, o1 => int1);
  g3 : or2  PORT MAP (i1 => int1, i2 => int2, o1 => ouput);
END struc;
```

รูปที่ 3.8 หน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทโครงสร้าง

```
ARCHITECTURE behav OF mux IS
BEGIN
    PROCESS (in0, in1, sel)
    BEGIN
        IF (sel = '0') THEN output <= in0;
        ELSE output <= in1;
        END IF;
    END PROCESS;
END behav;
```

รูปที่ 3.9 หน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทพฤติกรรม

ไม่ว่าเขียนบรรยายส่วนของสถาปัตยกรรมของมัลติเพลกซ์ในลักษณะของ ประเภทพฤติกรรม ประเภทการไหลของข้อมูล ประเภทโครงสร้างหรือประเภทผสมที่นำเอาแต่ละประเภทมาเขียนไว้ในส่วนของสถาปัตยกรรม ก็ตามต่างก็มีพฤติกรรมเดียวกัน และจะให้ผลลัพธ์จากการจำลองการทำงานที่เหมือนกัน ซึ่งนี้ก็เป็นข้อดีของภาษาวีเอชดีแอล

3.3.3 หน่วยการออกแบบแพ็กเก็ต

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิตอล สามารถเก็บไว้ในส่วนของแพ็กเก็ตได้ และข้อมูลเหล่านี้สามารถเรียกไปใช้ได้โดย หน่วยการออกแบบเอนทิตี หน่วยการออกแบบสถาปัตยกรรม หรือ จากหน่วยการออกแบบแพ็กเก็ตอื่นๆ นอกจากนั้นสิ่งที่นิยมทำกันมากคือรูปแบบมาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐาน (เช่น ไอซีตระกูล 74XX เป็นต้น) จะถูกเก็บไว้ในแพ็กเก็ต ที่ทุกคนสามารถเข้าถึง โดยปกติแล้ว แพ็กเก็ตจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็กเก็ต (Package declaration) และ ส่วนของบอดี้แพ็กเก็ต (Package body) เนื่องจาก แพ็กเก็ตถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้นการที่นำแพ็กเก็ตไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษาวีเอชดีแอล สามารถกระทำได้ด้วยชุดคำสั่ง USE

❑ **Package declaration**

ส่วนที่มีความสำคัญที่สุดของแพ็กเก็ต (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ ส่วนการประกาศแพ็กเก็ต เพราะจะเป็นส่วนที่กำหนดชื่อ ของสิ่งที่ประกาศอยู่ภายในแพ็กเก็ต สำหรับนำไปใช้ภายนอกตัวของแพ็กเก็ตเอง สิ่งใดๆ ถูกประกาศในส่วนของ ส่วนบอดี้แพ็กเก็ต แต่ไม่ถูกประกาศในส่วนการประกาศแพ็กเก็ต จะไม่สามารถถูกนำค่า และพฤติกรรมไปใช้ส่วนนอกได้ ซึ่งสามารถเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศเอนทิตีคือ จุดเชื่อมต่อ หรือ พอร์ต ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้ว แพ็กเก็ต สามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้ และยังสามารถถูกนำไปใช้จากรูปแบบภายนอกได้เช่น ใช้สำหรับประกาศ ชนิด (Type) หรือ สัญญาณ เช่นเดียวกันกับ ส่วนบอดี้แพ็กเก็ต ที่ไม่จำเป็นต้องมี ส่วนของการประกาศแพ็กเก็ต แต่แพ็กเก็ต นั้นจะไม่สามารถถูกนำไปใช้จากรูปแบบอื่นได้

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

รูปที่ 3.10 แสดงโครงสร้างโดยทั่วไปของส่วนการประกาศแพ็กเก็ต

❑ **Package body**

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (Subprogram) ทั้งหลายที่ชื่อของโปรแกรมย่อยนั้นๆ ที่ถูกประกาศไปในส่วนของการประกาศแพ็กเก็ต แล้วจะถูกเก็บไว้ในส่วนบอดี้แพ็กเก็ต ทั้งนี้รวมทั้ง การกำหนดค่าคงที่ต่างๆ อันได้แก่

ตังคงที่ที่ถูกประกาศชื่อก่อนในส่วนของการประกาศแพ็กแก็ต แต่ถูกกำหนดค่าในส่วนของบอดี้แพ็กแก็ต) ฉะนั้นส่วนบอดี้แพ็กแก็ต จึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็กแก็ต ไม่มีการประกาศชื่อ ที่เป็นโปรแกรมย่อย หรือ ค่าคงที่ การเขียนบอดี้แพ็กแก็ตนั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 3.11

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

รูปที่ 3.11 โครงสร้างของบอดี้แพ็กแก็ต

3.3.4 หน่วยการออกแบบโครงแบบ

ดังที่ทราบกันแล้วว่ารูปแบบหนึ่งของระบบดิจิตอลไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนทิตีได้ เพียงหนึ่งเดียวเท่านั้น แต่ในขณะที่หน่วยการออกแบบเอนทิตี หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรม ที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมี หน่วยการออกแบบโครงแบบมาเพื่อกำหนดการใช้โครงแบบ (Configuration) ประกอบเอนทิตีกับหน่วยการออกแบบสถาปัตยกรรมหน่วยไหนเข้าด้วยกัน

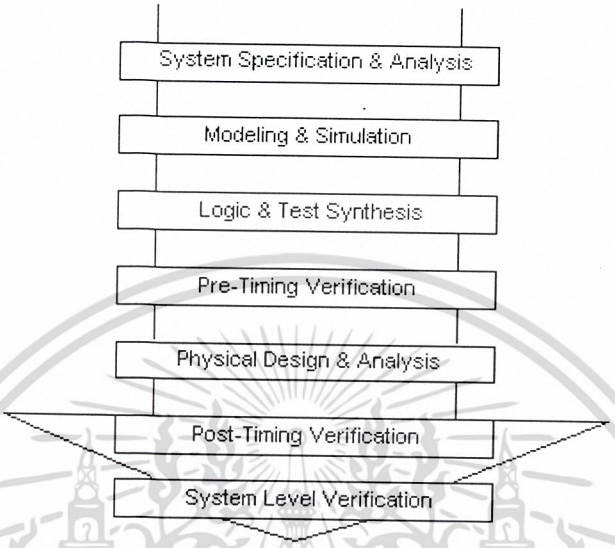
```
CONFIGURATION identifier OF entity_name IS
    Configuration_declarative_part
END ;
```

รูปที่ 3.12 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

3.4 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน เช่น วงจรรวม (ASIC: Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของของ บล็อกไดอะแกรมเสียก่อน ก่อนที่จะวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอลนั้นอนุญาติให้อธิบายการทำงานของแต่ละบล็อก และวิเคราะห์การทำงาน แก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ และเพิ่มเติมในรายละเอียดที่ละขั้นนี้คือ หลักการออกแบบจากบนลงล่าง (Top-Down Design) ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Botom-Up Design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เพราะเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ จำลองการทำงาน ตรวจสอบความถูกต้อง ซึ่งใช้เวลามาก และถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากให้การ

ออกแบบในลักษณะนี้ ดังนั้นการใช้ภาษาวีเอชดีแอลกับหลักการออกแบบจากบนลงล่าง จึงเป็นทางออกให้กับวิศวกรออกแบบที่จะสามารถออกแบบและพัฒนา วงจรที่มีซับซ้อนได้มากขึ้น และช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ



รูปที่ 3.13 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 3.13 แสดงให้ถึงขั้นตอนของการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย ก็เนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายๆ เทคโนโลยี เช่น พีแอลดี (PLD: Programmable Logic Device) อันได้แก่ พีแอลเอ (PLA: Programmable Logic Array), เอฟพีจีเอ (FPGA: Filed Programmable Gate Array), ซีพีแอลดี (CPLD: Cell Programmable Logic Device) เป็นต้น นอกนั้นยังมี เซมิคัลตัมไอซี (Semi-Custom IC) ได้แก่ เกตอะเรย์ (Gate array), เซลล์มาตรฐาน (Standard Cell) ขั้นตอนของการออกแบบจากบนลงล่างมีรายละเอียดดังนี้

1. ขั้นตอนการสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2. ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษาวีเอชดีแอล หรือ ภาษาเอชดีแอลอื่นๆ สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3. หลังจากที่ได้หลักการขั้นต้นพร้อมกับแนวความคิดที่ผ่านการตรวจสอบแล้ว หลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมาเป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจร หรือสังเคราะห์ ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้น ให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์หรือวงจรในระดับเกต และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้น หรือไม่ก็อยู่ในรูปของเน็ทลิสท์ (Netlist) ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้

4. หลังจากการสังเคราะห์วงจรให้อยู่ในรูประดับเกตหรือเน็ทลิสท์แล้ว ข้อมูลที่ได้จากผู้ผลิตอุปกรณ์วงจรนั้น นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงาน ในเรื่องของความถูกต้องของฟังก์ชันแล้ว ยังมีข้อมูลที่เกี่ยวกับเวลาด้วย ซึ่งเป็นความจริงที่ว่า อุปกรณ์ทางอิเล็กทรอนิกส์ทุกชิ้นจะมี เวลาหน่วงของการแพร่กระจาย (Propagation delay time) เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับ นาโนวินาที (10^{-9} นาที) แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกตของฟังก์ชันต่างๆ จำนวน 10,000 เกต ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจจะทำให้การทำงานของวงจรรวมทั้งหมดผิดไป หรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

5. ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้น หรืออยู่ในรูปของวงจรรวม (ASIC)

6. หลังจากที่ได้วงจรจริงมาแล้ว ยังต้องมีความจำเป็นที่ต้องตรวจสอบการทำงานที่คำนึงถึงเวลาด้วย เพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เพราะในขั้นตอนนี้วงจรที่ออกแบบ จะประกอบด้วยอินพุตและเอาต์พุตแพด (Pad) ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก

7. หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอลแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆ อีกครั้ง เป็นการควบคุมคุณภาพของผลิตภัณฑ์

บทที่ 4
เอฟพีจีเอ

4.1 วิวัฒนาการของการประดิษฐ์วงจรรวม

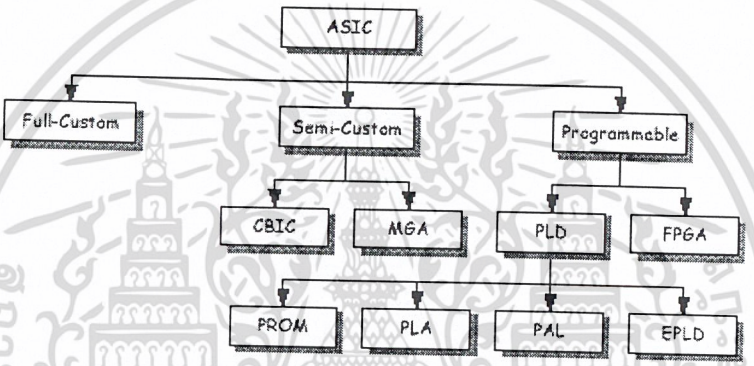
ในช่วงก่อนทศวรรษ 1970 อุตสาหกรรมเซมิคอนดัคเตอร์ได้ถูกปลุกให้ตื่นตัวขึ้นหลังจากที่มีการประดิษฐ์วงจรรวมหรือไอซี ตัวแรกสำเร็จ ในยุคแรกนั้นไอซีขนาดเล็กหรือ SSI (Small-Scale Integration) ประกอบไปด้วยเกทดิจิตอลจำนวนไม่มาก นัก(ประมาณ 1 ถึง 10 ตัว) ต่อมาได้มีการเพิ่มปริมาณของเกทดิจิตอลและฟังก์ชันทางลอจิกให้มากขึ้นจนเป็น MSI (Medium- Scale Integration) การพัฒนาไอซีเป็นไปอย่างต่อเนื่องจนมาถึงยุคของ LSI (Large-Scale Integration) ซึ่งเป็นยุคที่มีการ สร้างไมโครโปรเซสเซอร์ตัวแรกขึ้นและในปัจจุบันเป็นยุคของ VLSI (Very Large-Scale Integration) ซึ่งเทคโนโลยีในการ สร้างไอซีรุดหน้าจนสามารถสร้างไมโครโปรเซสเซอร์ขนาด 64 บิต ที่มีหน่วยความจำแคชกับหน่วยคำนวณทางคณิตศาสตร์ ของโฟล์ทติงพอยน์ (Floating-Point Arithmetic Units) รวมอยู่ภายในตัวมันและเนื่องจากการปรับปรุงเทคโนโลยี ของกระบวนการสร้างซีมอส ที่มีมาอย่างต่อเนื่องทำให้ขนาดของทรานซิสเตอร์ที่บรรจุอยู่ภายในไอซีมีขนาดเล็กลงเรื่อยๆ จนบางคนโดยเฉพาะ ในญี่ปุ่นใช้คำว่า ULSI (Ultralarge Scale Integration) เพื่อใช้เรียกระดับของไอซีในปัจจุบันแต่คนส่วนมาก ยังมักนิยมเรียกเพียงแค่ VLSI

จากการปรากฏตัวของ VLSI ในช่วงทศวรรษ 1980 ทำให้วิศวกรเริ่มมีการออกแบบไอซีตามความต้องการของลูกค้าซึ่งใช้ใน ระบบที่เจาะจงนอกเหนือจากการใช้ไอซีมาตรฐานเพียงอย่างเดียว โดยไอซีเหล่านี้มีชื่อเรียกว่า ASIC :Application-Specific Integrated Circuit (ออกเสียงว่า เอ-ซิก) ซึ่งตัวอย่างของ ASIC ได้แก่ชิพไอซีที่ใช้ สำหรับตุ๊กตาของเล่นพูดได้ ดาวเทียม และ ชิพที่ภายในบรรจุด้วยไมโครโปรเซสเซอร์กับอุปกรณ์ทางลอจิกอื่นๆ

อุปกรณ์ FPGA (Field Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชันการทำงานตามที่ผู้ออกแบบต้องการ ในการทำ FPGA เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuits) แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือ การทำ FPGA จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในอุปกรณ์ FPGA จะมีจำนวน gate ให้ใช้จำนวนจำกัด และการทำ FPGA ก็เหมาะกับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียน code อธิบายฮาร์ดแวร์จนกระทั่ง download นั้นน้อยกว่าการทำ ASIC มาก และการตรวจสอบหรือแก้ไข design ก็ทำได้สะดวก การทำ FPGA ในปัจจุบันมีประสิทธิภาพมากขึ้น และสะดวกขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตอุปกรณ์ FPGA ได้เพิ่มความสามารถของอุปกรณ์ FPGA โดยเพิ่มจำนวนองค์ประกอบภายใน หรือ ปรับปรุงโครงสร้างสถาปัตยกรรมภายใน และยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ ที่ใช้ทำ PPR (Partitioning, Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย

4.2 ASIC

ความก้าวหน้าของอุตสาหกรรมอิเล็คทรอนิกส์ ปัจจุบันทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆ มากมายซึ่งทำให้เกิดการลดค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจาก เทคโนโลยีไมโครโพรเซสเซอร์และหน่วยความจำปัจจุบัน ทุกๆ ครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวมและไอซีมาตรฐานมากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน ฟังก์ชันลอจิก ที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตให้ขนาดมากๆ และการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวม จะแบ่งเป็น 3 ประเภทใหญ่ๆ คือ Full-custom, Semi-custom และ Programmable ดังรูปที่ 4.1



รูปที่ 4.1 ประเภทของ ASIC

4.2.1 Full-custom

ASIC ประเภทนี้ลูกค้าจะเป็นผู้ออกแบบเซลล์ลอจิก (เช่น แอนด์เกท ออร์เกท มัลติเพล็กเซอร์ และฟลิบฟล็อบ) และลักษณะ การจัดวางอุปกรณ์บนตัวไอซีรวมถึงหน้ากากสำหรับควบคุมการเจือและสร้างชั้นสาร (Mask) ต่างๆ ที่ ใช้ในการทำไอซีเอง ดังนั้นค่าใช้จ่ายในการออกแบบและการผลิตจะสูงมาก

4.2.2 Semi-custom

ASIC ประเภทนี้เซลล์ลอจิกจะถูกออกแบบเอาไว้ก่อนแล้วในรูปแบบของไลบรารีและลูกค้าจะเป็นผู้ออกแบบ Mask ต่างๆเอง ตัวอย่างของไอซีประเภทนี้ได้แก่ Standard-Cell-Based ASIC และ Masked Gate-Array-Based ASIC

4.2.2.1 Standard-Cell-Based ASIC

ไอซีประเภทนี้จะมีพื้นที่สำหรับจัดวางเซลล์ลอจิกมาตรฐาน ซึ่งถูกออกแบบเอาไว้แล้วในบางครั้งเซลล์ มาตรฐานเหล่านี้ จะถูกนำมาประกอบกันเป็นเซลที่มีขนาดใหญ่ขึ้นเรียกว่า Megacell สำหรับการออกแบบนั้น ผู้ออกแบบจะทำเพียงแค่กำหนดตำแหน่งของเซลล์มาตรฐานและการเชื่อมต่อภายในของแต่ละเซลล์เท่านั้นแต่อย่างไรก็ดีเซลล์ต่างๆ เหล่านี้สามารถวางที่ตำแหน่งใดๆ ก็ได้บนแผ่นเวเฟอร์ซิลิกอน นั่นก็หมายความว่าชั้น Mask จะถูกจัดวางตามความต้องการของผู้ออกแบบ

4.2.2.2 Masked Gate-Array-Based ASIC

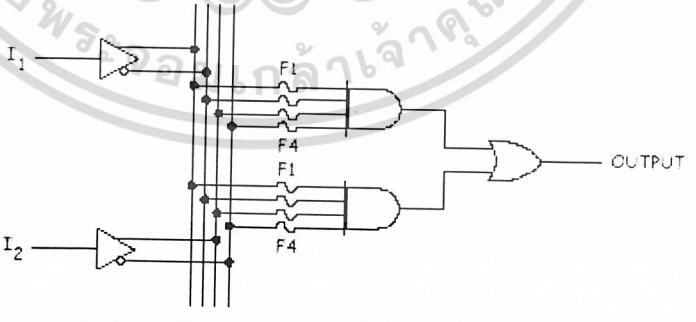
ไอซีชนิดนี้จะมีทรานซิสเตอร์หรือเกทถูกสร้างมาในลักษณะของอะเรย์สองมิติบนแผ่นเวเฟอร์ซิลิกอน และผู้ออกแบบจะทำการออกแบบ Mask เพื่อใช้สำหรับกำหนดการต่อเชื่อมของทรานซิสเตอร์แต่ละตัว

4.2.3 Programmable

ASIC ประเภทนี้เซลล์ลอจิกจะถูกออกแบบไว้ก่อนเช่นเดียวกับ Semi-Custom แต่ชั้นของ Mask จะไม่สามารถเปลี่ยนแปลงได้ ตามความต้องการของผู้ออกแบบ ไอซีประเภทนี้ยังแบ่งออกเป็น 2 ชนิดคือ Programmable Logic Device (PLD) และ Field Programmable Gate Array (FPGA)

4.2.3.1 Programmable Logic Device (PLD)

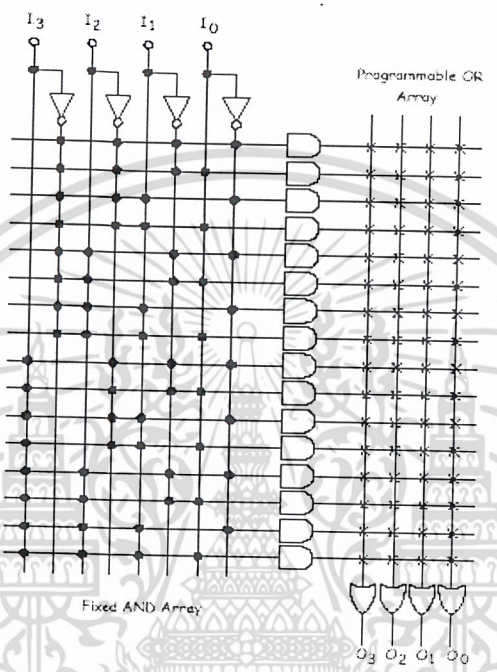
มีโครงสร้างภายในเป็นวงจรพื้นฐานทางด้านลอจิกต่อกันอยู่เป็นกลุ่มซึ่งมีทั้งวงจรคอมบิเนชัน(Combination) และซีเควนเชียล (Sequential) สำหรับเทคโนโลยีของวงจรที่ใช้สร้าง PLD จะมีทั้ง TTL, ECL และ CMOS ตามความเหมาะสมของแต่ละระบบ ไอซี PLD ทุกชนิดมีหลักการพื้นฐานของวงจรภายในที่เหมือนกันโดยมี วงจรคอมบิเนชันที่เป็นผลคูณร่วม บวก (Sum of product)ประกอบไปด้วยชุดของแอนด์เกทต่อร่วมกับออร์เกทและในการ โปรแกรมจะเป็นการเลือกว่าอินพุทภายในของแอนด์เกทกับสัญญาณอินพุทใดบ้างที่จะต้องต่อถึงกันซึ่งมีทั้งจากภายนอกและสัญญาณป้อนกลับจากเอาท์พุทภายในเอง เช่น การติดต่ออินพุทของออร์เกทกับเอาท์พุทของแอนด์เกทตัวต่างๆ สำหรับการโปรแกรมทางกายภาพนั้นอินพุท ต่างๆ ของอุปกรณ์ทุกตัว จะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใดก็จะตัดฟิวส์ตัวนั้นทิ้งทำให้สามารถ โปรแกรมได้เพียงครั้งเดียว ไอซี PLD บางชนิดใช้ มอสทรานซิสเตอร์แทนฟิวส์ทำให้สามารถโปรแกรมโดย ใช้กระแสไฟฟ้าและสามารถลบแล้ว โปรแกรมเข้าไปใหม่ได้อีก สำหรับไอซีในตระกูล PLD ได้แก่ PROM, PAL, PLA และ EPLD



รูปที่ 4.2 วงจรพื้นฐานของ PLD ซึ่งอยู่ในรูปผลคูณร่วมบวก

❑ **PROM (Programmable Read Only Memory)**

PROM คือหน่วยความจำประเภท ROM ซึ่งนับว่าเป็นไอซี PLD ชนิดหนึ่งซึ่งวงจรภายใน ของ PROM ประกอบไปด้วยอะเรย์ของแอนด์และออร์เกท (And - Or Array) ผลลัพธ์ที่ขา ดาต้าเอาท์พุทสามารถแสดงได้ในสมการของฟังก์ชันผลคูณร่วมบวก (Sum of product) ของสัญญาณอินพุทที่ขาแอดเดรส



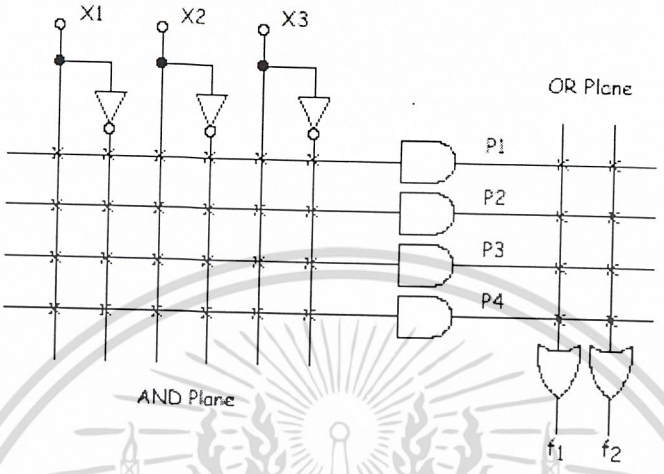
รูปที่ 4.3 ลักษณะของ PROM เมื่อเปรียบเทียบเป็นวงจรในรูปผลคูณร่วมบวก

รูปที่ 4.3 แสดงถึงลักษณะการเชื่อมต่อแอนด์เกทและออร์เกทของ PROM ขนาด 16x4 บิต วงจรทางด้านซ้ายบนสุดเป็นแอนด์เกทจะให้ผลคูณ (Product) ของกรณีที่อินพุทเป็น0000 แอนด์เกทที่อยู่ถัดลงมาเป็นผลคูณของกรณีที่อินพุทเป็น 0001, 0010, ...จนถึงตัวล่างสุดคือ ผลคูณในกรณีที่อินพุทเป็น 1111 ซึ่งสำหรับ PROM ที่มีจำนวนอินพุท n ตัวจะมีค่าอินพุทที่ เป็นไปได้ทั้งหมดเท่ากับ 2n และค่าอินพุทเหล่านี้จะถูกจัดวางอยู่ในส่วนอะเรย์ของ AND ซึ่งไม่สามารถแก้ไขได้ แต่ในส่วนของ OR จะเป็นส่วนที่อนุญาตให้ทำการโปรแกรมได้ และเนื่องจากการที่ด้าน AND ของ PROM มีการคอมบิเนชันของอินพุทที่เป็นไปได้ทั้งหมด ดังนั้นผู้ออกแบบจึงไม่จำเป็นต้องทำการลดรูปของฟังก์ชันลอจิกที่ออกแบบไว้เลยแต่อย่าง ไรก็ดีการกระทำเช่นนี้อาจทำให้เกิดจำนวนวงจรที่ไม่มีประสิทธิภาพจำนวนมากบนตัว ชิพได้

❑ **PLA (Programmable Logic Array)**

ลักษณะเด่นของ PLA คือสามารถโปรแกรมการเชื่อมต่อได้ทั้งทางด้าน AND และด้าน OR ทำให้มีความยืดหยุ่นในการใช้งานมาก แต่อย่างไรก็ดีข้อเสียที่เห็นได้อย่างชัดเจน ของ PLA คือความ

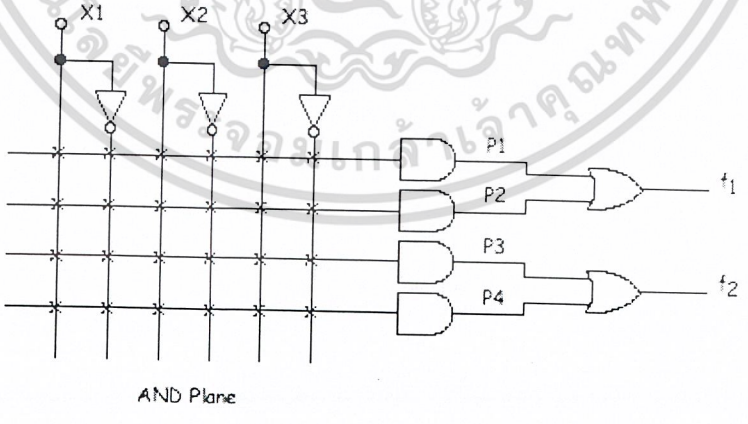
ยุ่งยากในการสร้างและคุณสมบัติทางด้านความเร็วที่ลดลงเนื่องจาก สัญญาณจะต้องวิ่งผ่านอะเรย์ของ AND และ OR



รูปที่ 4.4 วงจรพื้นฐานภายในของ PLA

❑ **PAL (Programmable Array Logic)**

PAL มีลักษณะโครงสร้างที่ใกล้เคียงกับ PROM และ PLA มาก แต่การโปรแกรม PAL จะสามารถทำได้เพียงด้าน AND เท่านั้น



รูปที่ 4.5 วงจรพื้นฐานภายในของ PAL

❑ **EPLD (Erasable Programmable Logic Device)**

EPLD เป็นอุปกรณ์ที่สามารถทำการโปรแกรมได้หลายครั้งซึ่งเหมาะสำหรับการ ทำวงจรต้นแบบ สำหรับเทคโนโลยีที่ใช้ในการสร้างจะเหมือนกับ CMOS EPROM คือ ใช้มอสทรานซิส เตอร์เชื่อมต่อระหว่างสัญญาณอินพุทกับจุดที่ต้องการแทนการ ใช้ฟิวส์แบบเดิมทำให้สามารถโปรแกรมการต่อวงจรภายในอุปกรณ์ด้วยการจ่าย ไฟฟ้าตามขนาดที่กำหนดไว้และลบได้โดยใช้แสงอัลตร้าไวโอเลทฉายผ่านช่อง หน้าต่างกระจกของตัวชิพ

4.2.3.2 Field-Programmable Gate Array (FPGA)

เป็นอุปกรณ์ที่มีความซับซ้อนมากกว่า PLD ไปอีกระดับหนึ่ง ซึ่งในความเป็นจริงแล้ว PLD และ FPGA แตก ต่างกันน้อยมาก สำหรับ FPGA แล้วนับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่าง รวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTL โครงสร้าง ภายในของ FPGA ประกอบไปด้วยอะเรย์ของลอจิกเกทต่างๆมากมาย ซึ่งในปัจจุบันความจุเกทภายใน ตัวชิพ FPGA ได้เพิ่มขึ้น จากระดับไม่กี่พันตัวจนถึงระดับล้านตัวซึ่งสามารถรองรับวงจรดิจิตอลที่มีความ สลับซับซ้อนได้เป็นอย่างดี นอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่ายซึ่งในปัจจุบัน การออกแบบวงจรโดยใช้ FPGA กำลังเป็นที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อย

ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรม ที่แตกต่างกันไป นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบคือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรม โดยการใช้หน่วยความจำ

❑ **การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ**

Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียว ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อมต่อจะขาดจากกัน

Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรม แล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

❑ **การโปรแกรมโดยใช้หน่วยความจำ**

EEPROM Based FPGA

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

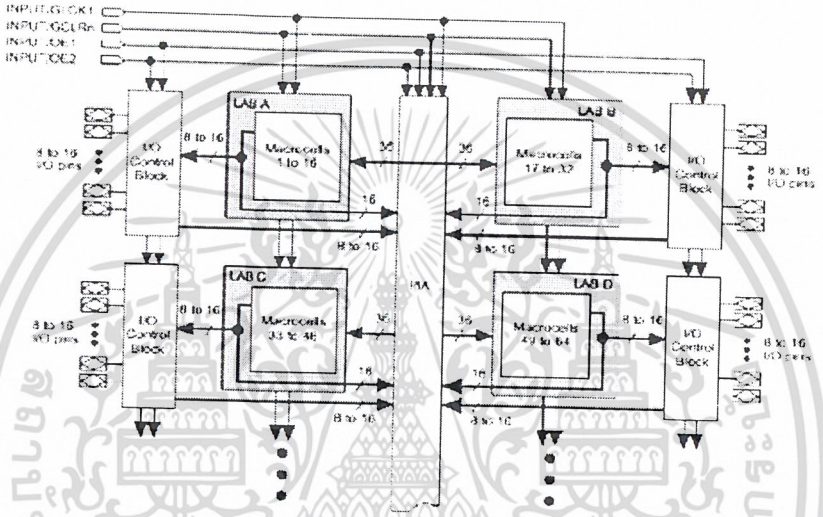
SRAM Based FPGA

FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM) ทำให้สามารถ โปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 - 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการ โปรแกรม

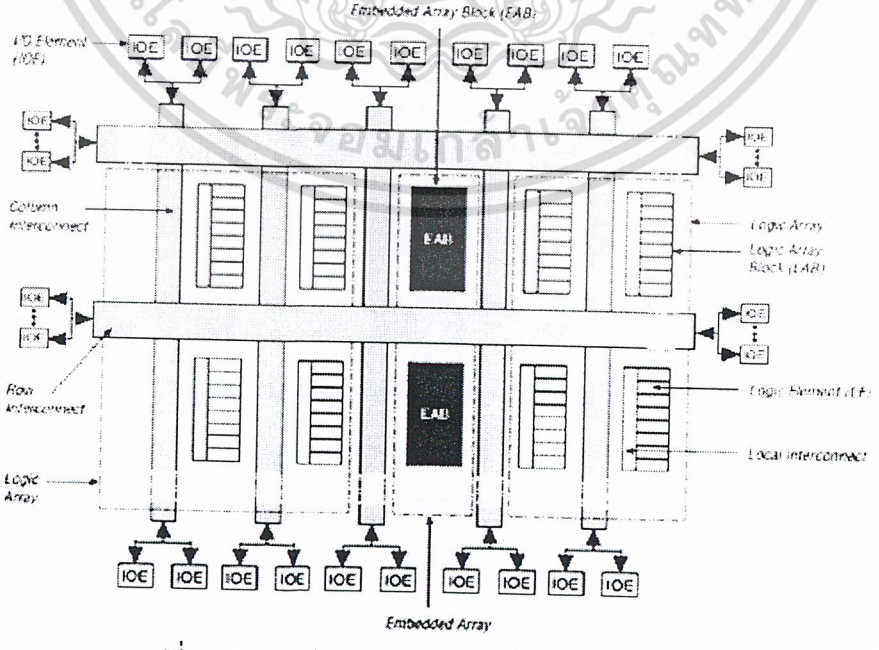
น้อย (ระดับ nsec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และ เหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมใน ภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการ โหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

4.3 โครงสร้างภายในของ FPGA

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อคลอจิกที่สามารถทำการโปรแกรมได้ดังรูปที่ 4.6 และ 4.7



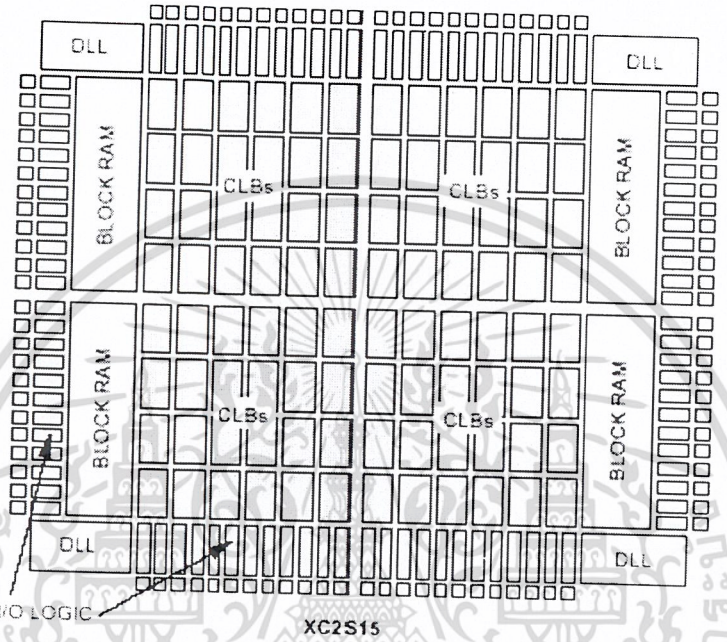
รูปที่ 4.6 โครงสร้างภายในของ FPGA ตระกูล MAX7000S



รูปที่ 4.7 โครงสร้างภายในของ FPGA ตระกูล FLEX10K

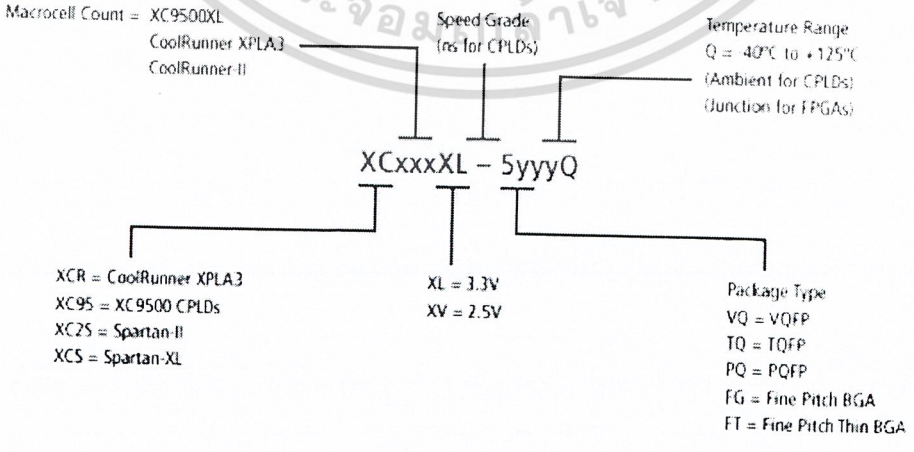
4.3.1 โครงสร้างภายในของ FPGA SpartanII รุ่น XC2S15

การทดลองสร้างวงจรเข้ารหัสและถอดรหัสข้อมูล จะใช้ FPGA SpartanIII รุ่น XC3S200 มีเกทภายในประมาณ 200,000 เกท แต่โครงสร้างภายในมีลักษณะคล้ายกับ FPGA SpartanII รุ่น XC2S15 จึงนำมาแสดงดังรูปที่ 4.8



รูปที่ 4.8 โครงสร้างภายในของ FPGA SpartanII รุ่น XC2S15

4.3.2 วิธีอ่านเบอร์ชิพ FPGA ของบริษัท Xillinx

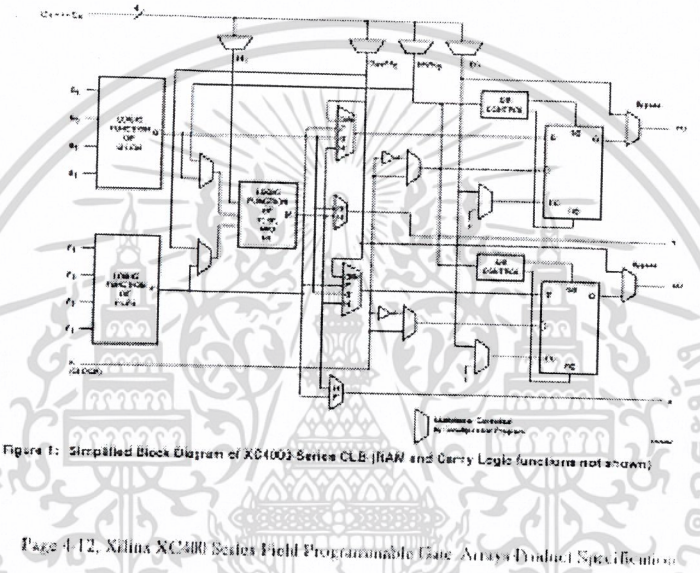


รูปที่ 4.9 วิธีอ่านเบอร์ชิพ FPGA ของบริษัท Xillinx

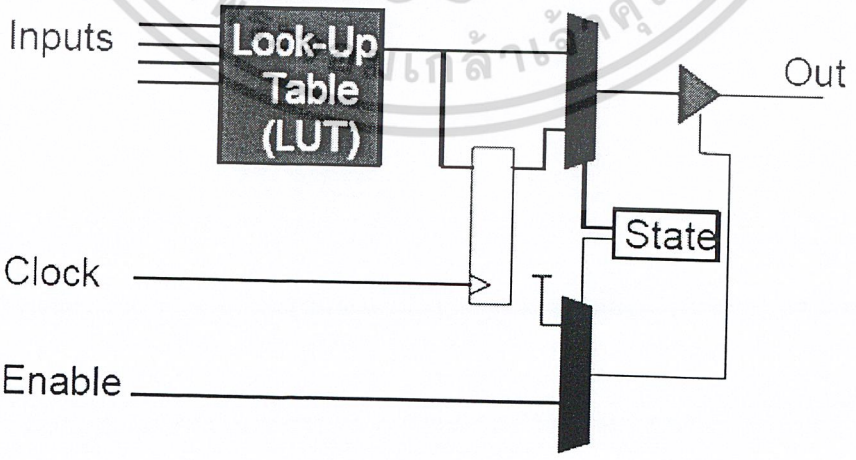
4.4 รายละเอียดของโครงสร้างที่ควรรู้จัก

4.4.1 ซีแอลบี (CLB : Configuration Logic Block)

ภายในซีแอลบีคือ แมทริกซ์ของซีแอลบีแต่ละตัวประกอบด้วย หน่วยของคอมบิเนชั่นลอจิกที่สามารถโปรแกรมได้ (Programmable Combination Logic) และส่วนของรีจิสเตอร์เก็บข้อมูล (Storage Register) ส่วนของวงจรคอมบิเนชั่นลอจิกสามารใช้วงจรทางด้านฟังก์ชั่นบูลีนของอินพุต ส่วนรีจิสเตอร์รับค่าจากส่วนคอมบิเนชั่นหรือโดยตรงจากเอาท์พุทของซีแอลบี สามารถขับวงจรคอมบิเนชั่นลอจิกโดยตรงผ่านเส้นทางเดินย้อนกลับ (Feedback path)



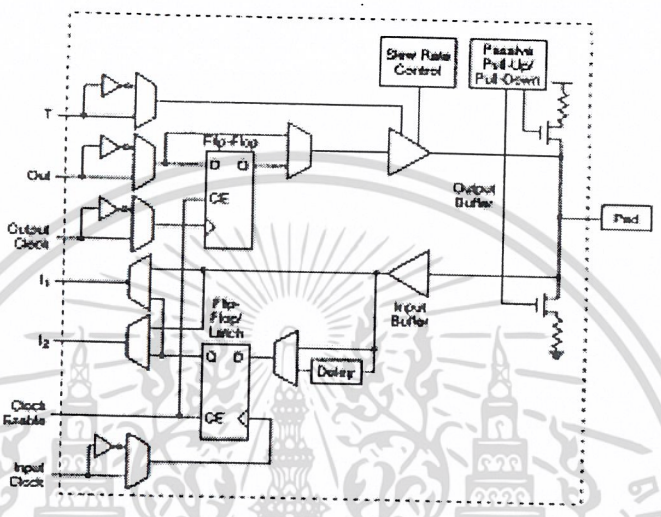
รูปที่ 4.10 แสดงผังวงจรภายในของซีแอลบี



รูปที่ 4.11 แสดงการทำงานของซีแอลบี

4.4.2 ไอโอบี (IOB : Input Output Block)

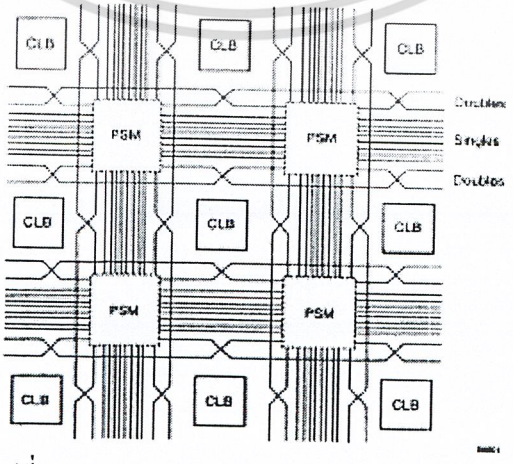
เป็นส่วนติดต่อกับวงจรภายนอกของแอลซีเอส สร้างมาจากส่วนของอุปกรณ์อินพุท/เอ้าท์พุทที่สามารถโปรแกรมได้ แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้อินพุท/เอาท์พุทแบบ 3 สถานะ หรือ ไอโอแบบสองทิศทางก็ได้ โดยอินพุทสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอลและซีมอสเทรดโฮลของไอโอบี แต่ละตัวมีฟลิปฟลอปสามารถใช้เป็นบัพเฟอร์สำหรับอินพุทและเอ้าท์พุท



รูปที่ 4.12 แสดงผังวงจรของไอโอบี

4.4.3 อินเตอร์คอนเน็ค (Interconnect)

ความยืดหยุ่นของการใช้แอลซีเอมาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรมทรัพยากรต่างๆ ที่อยู่ภายในเข้าด้วยกัน การที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปเหมือนกับเกทอาเรย์ทั่วๆ ไป การเชื่อมต่อภายในแอลซีเอประกอบด้วยเน็ทเวิร์ค 2 ทิศทางคือ ทางแถวและคอลัมน์ซึ่งวางอยู่ระหว่าง CLB Programmable Switch จะทำการเชื่อมต่ออินพุทและเอ้าท์พุทของไอโอบีและซีแอลบีที่จุดต่อร่วมระหว่างแถวกับคอลัมน์ สามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่าง ๆ



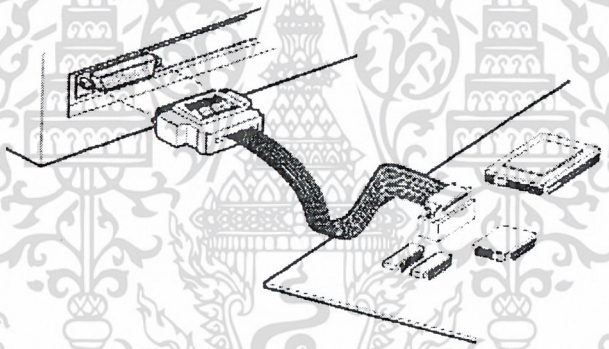
รูปที่ 4.13 แสดงผังวงจรของอินเตอร์คอนเน็ค

4.5 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออก แบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

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

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 4.14 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 4.14 การโปรแกรมลงในชิพ

4.6 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ จากที่ได้กล่าวไปแล้วในบทที่ 2 ในกรณีของการออก แบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้วจึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำ ได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออก แบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยน

แปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

4.6.1 การสังเคราะห์วงจร (Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรม ของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์ นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบ ด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆจึงทำให้ผล ที่ได้มีประสิทธิภาพและในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดล แต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจร ที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับ เทคโนโลยีที่ใช้ เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำ การสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้น เป็นอย่างไร เช่นมีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้น ตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตาม ที่กำหนด

4.6.2 การแบ่งวงจร (Partitioning)

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

4.6.3 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มา แล้วว่าควร จะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะ ได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี ความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณไดไม่หมด การวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กันโดยเฉพาะส่วน ที่มีการเชื่อมต่อสัญญาณด้วยกันนอกจากนั้นการกำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ก็จะมีผลโดยตรงเลยคือซอฟต์แวร์จะวาง I/O ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่ง บางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขา I/O ควรกำหนดตำแหน่งให้เหมาะสม หรือ ไม่ก็ให้ซอฟต์แวร์จัดการเอง

4.6.4 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด (เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกิน ค่าที่กำหนดในข้อบังคับ ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือผู้ออกแบบจะทำการ เชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับ ทางเวลา จะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

4.6.5 ความหน่วงด้านเวลา (Delay)

ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วง ที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์ FPGA เอง ความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay) เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA โดยปกติแล้วค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการ เชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์ และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลาเพื่อให้ ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อ ให้ได้ผลลัพธ์ที่ดีขึ้นค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความ หน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้น เป็นไปตามข้อกำหนด หรือไม่

4.6.6 การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น Model Sim ของบริษัท Model Technology หรือ Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดล เกิดขึ้น

ตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้ เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออก แบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจสอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจร แล้ว เพื่อตรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตาม ข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้ จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้อง ตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวม สามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ ซอฟต์แวร์ตัวเดียวกันตอลดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

4.6.7 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement & Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มี ฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับ อุปกรณ์ FPGA ของแต่ละบริษัทผู้ผลิตคือ ในกรณีที่เป็นอุปกรณ์ FPGA ชนิดที่ต้องโปรแกรมโดย วิธี SRAM นั้น ในการใช้งานผู้ออกแบบจะต้องเก็บข้อมูลวงจรไว้ในหน่วยความจำประเภท EPROM หรือ serial PROM ด้วยเพื่อจะใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปไม่ ต้องดาวน์โหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีกเพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่ แล้ว แต่กรณีที่อุปกรณ์ FPGA เป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Anti fuse ก็ไม่จำ เป็นต้องมีหน่วยความจำสำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูล วงจรลงไป ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออก แบบไว้ได้เลย

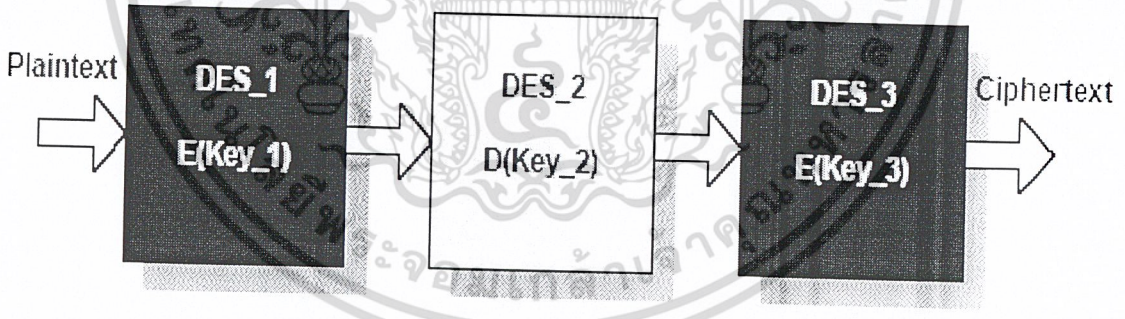
บทที่ 5
การออกแบบวงจรเข้ารหัสและถอดรหัสข้อมูล
ด้วยวิธีการแบบ Triple DES

5.1 แนวทางการออกแบบ

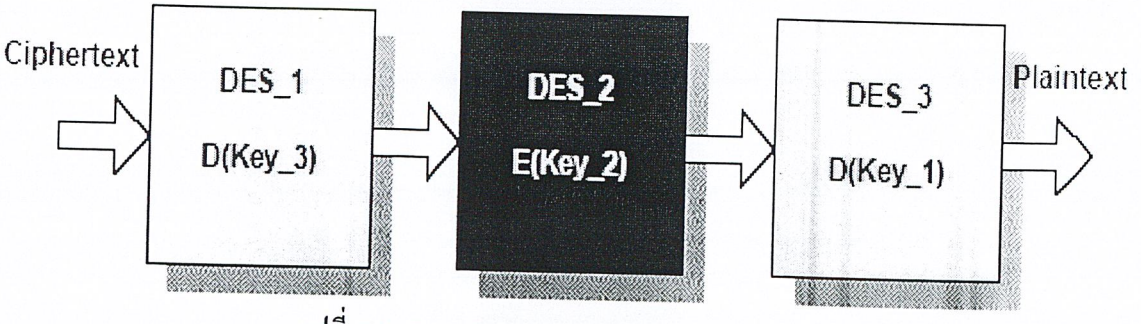
การทำโครงงานครั้งนี้เลือกใช้อัลกอริทึมแบบ Triple DES ในการสร้างวงจรเข้ารหัสและถอดรหัสข้อมูล จากทฤษฎีของการเข้ารหัสและถอดรหัสข้อมูลพบว่า Triple DES เป็นการนำเอา DES Algorithm มากระทำซ้ำกัน 3 ครั้ง แสดงให้เห็นในรูปที่ 5.1 และ 5.2 ทำให้จำนวนของ key เพิ่มขึ้น ความปลอดภัยของข้อมูลจึงมีมากขึ้น เนื้อหาในบทนี้จะเสนอเกี่ยวกับการออกแบบการเข้ารหัสและถอดรหัสข้อมูล โดยจะแสดงขั้นตอนของการออกแบบซึ่งมี 2 ขั้นตอนคือ การออกแบบระดับบล็อกไดอะแกรม และการโปรแกรม

การออกแบบระดับบล็อกไดอะแกรม จะให้เห็นภาพของ Triple DES โดยรวม จากนั้นจะแยก DES ออกมาหนึ่งบล็อกไดอะแกรม และจะแสดง Component ต่างภายในรวมทั้งหน้าที่การทำงานของแต่ละ Component

การโปรแกรม จะโปรแกรมให้ Component ต่างๆ มีหน้าที่การทำงานตามอัลกอริทึม ในการโปรแกรมจะโปรแกรมด้วยภาษา VHDL



รูปที่ 5.1 Triple DES Encryption Block Diagram



รูปที่ 5.2 Triple DES Decryption Block Diagram

5.2 ขั้นตอนการออกแบบ

ขั้นตอนการออกแบบมี 2 ขั้นตอน

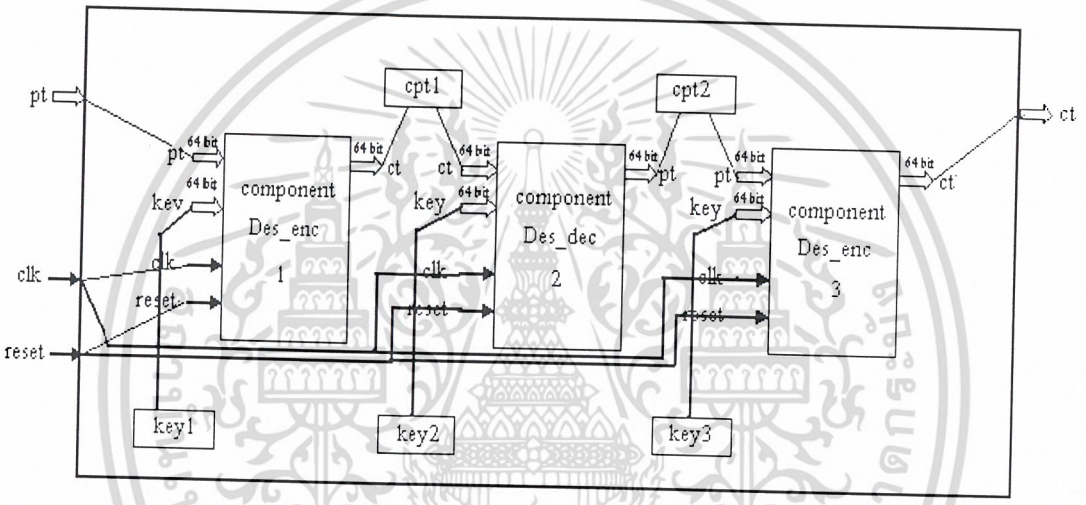
- ❑ ออกแบบ Block Diagram
- ❑ โปรแกรมภาษา VHDL

5.2.1 ออกแบบ Block Diagram

5.2.1.1 Triple DES Algorithm

❑ Triple DES Encryption

ในส่วนของการออกแบบบล็อกไดอะแกรมของ Triple DES Encryption ทำการออกแบบตามรูปที่ 5.3 รายละเอียดของแต่ละส่วนมีดังนี้



รูปที่ 5.3 Triple DES Encryption Block Diagram

Input ประกอบด้วย pt (plaintext) ขนาด 64 bit ทำหน้าที่รับ input ที่เป็น plaintext ขนาด 64 bit

clk (clock) ขนาด 1 bit ทำหน้าที่เป็นสัญญาณ clock ให้แก่ระบบ เพื่อให้ระบบนั้นทำงาน synchronize กัน

Reset ขนาด 1 bit ทำหน้าที่เป็นสัญญาณ reset ให้แก่ระบบ เพื่อให้ระบบ ทำงาน synchronize กัน

Output ประกอบด้วย ct (ciphertext) ขนาด 64 bit เป็น output สำหรับส่ง output ที่ได้ จากระบบที่ทำ Encrypt ออกไปสู่ ภายนอกระบบ

Component ภายใน Component Des_enc1 ทำหน้าที่ Encrypt โดยใช้ key1 ตามมาตรฐานของ Triple Des

Component Des_dec2 ทำหน้าที่ Decrypt โดยใช้ key2 ตามมาตรฐานของ Triple Des Component Des_enc3 ทำหน้าที่ Encrypt โดยใช้ key3 ตามมาตรฐานของ Triple Des

Key (key1,key2,key3) ซึ่งเป็น key ภายใน เป็น key ที่มีความแตกต่างกัน 3 key

key1 ใช้สำหรับ component Des_enc1

key2 ใช้สำหรับ component Des_dec2

key3 ใช้สำหรับ component Des_enc3

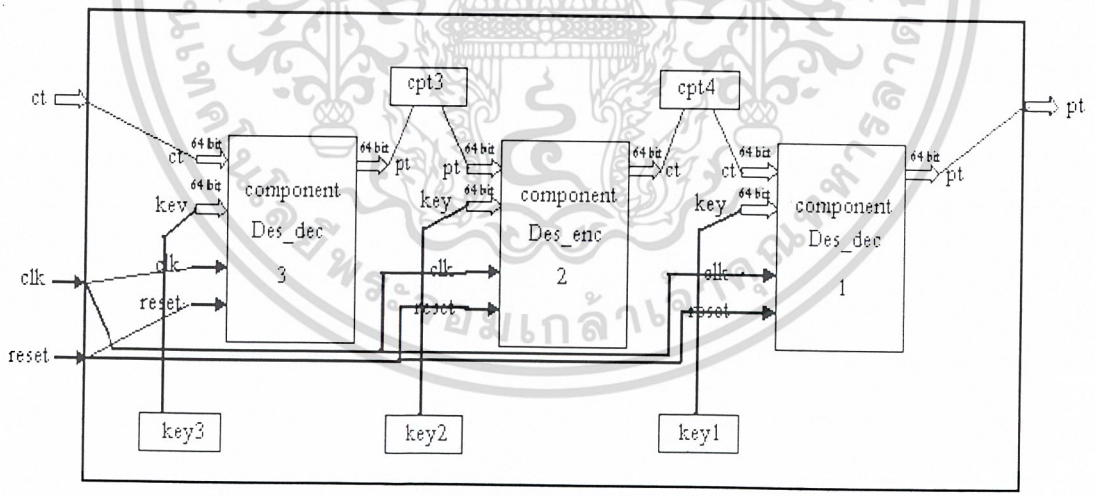
ดังนั้นในระบบจะมี key ทั้งหมด เท่ากับ key1 (64) + key2(64) + key3(64) = 192 bit

cpt1 มีขนาด 64 bit ทำหน้าที่เป็น output ให้กับ component Des_enc1 และเป็น input ให้กับ Des_dec2

cpt2 มีขนาด 64 bit ทำหน้าที่เป็น output ให้กับ component Des_dec2 และเป็น input ให้แก่ Des_enc3

❑ **Triple DES Decryption**

ในส่วนของการออกแบบบล็อกไดอะแกรมของ Triple DES Decryption ทำการออกแบบตามรูปที่ 5.4 รายละเอียดของแต่ละส่วนมีดังนี้



รูปที่ 5.4 Triple DES Decryption Block Diagram

Input ประกอบด้วย ct (ciphertext) ขนาด 64 bit ทำหน้าที่รับ input ที่เป็น ciphertext ขนาด 64 bit

clk (clock) ขนาด 1 bit ทำหน้าที่เป็นสัญญาณ clock ให้แก่ระบบเพื่อให้ระบบนั้นทำงาน synchronize กัน

Reset ขนาด 1 bit ทำหน้าที่เป็นสัญญาณ reset ให้แก่ระบบ เพื่อให้ระบบ ทำงาน synchronize กัน

Output ประกอบด้วย pt (plaintext) ขนาด 64 bit เป็น output สำหรับส่ง output ที่ได้ จากระบบที่ทำ Decrypt ออกไปสู่ภายนอกระบบ (ได้ข้อมูลเป็นตัวเดียวกับตัว ที่เข้ามาใน Triple Des Encryption)

Component ภายใน Component Des_dec3 ทำหน้าที่ Decrypt โดยใช้ key3 ตามมาตรฐาน ของ Triple Des

Component Des_enc2 ทำหน้าที่ Encrypt โดยใช้ key2 ตามมาตรฐานของ Triple Des Component Des_dec1 ทำหน้าที่ Decrypt โดยใช้ key1 ตามมาตรฐานของ Triple Des

key (key1,key2,key3) ซึ่งเป็น key ภายใน เป็น key ที่มีความแตกต่างกัน 3 key

key3 ใช้สำหรับ component Des_enc3

key2 ใช้สำหรับ component Des_dec2

key1 ใช้สำหรับ component Des_enc1

ดังนั้นในระบบจะมี key ทั้งหมด เท่ากับ key1 (64) + key2(64) + key3(64) = 192 bit

cpt3 มีขนาด 64 bit ทำหน้าที่เป็น output ให้กับ component Des_enc3 และเป็น input ให้กับ Des_dec2

cpt4 มีขนาด 64 bit ทำหน้าที่เป็น output ให้กับ component Des_dec2 และเป็น input ให้แก่ component Des_enc1

5.2.1.2 ออกแบบ DES Encryption และ DES Decryption Block Diagram

Triple DES เป็นการออกแบบจาก DES Algorithm มากระทำซ้ำกัน 3 ครั้ง ตามรูปที่ 5.1 และ 5.2 ดังนั้นสิ่งที่เราต้องออกแบบคือ DES Encryption และ DES Decryption อย่างละชุดเท่านั้น (ในชุดอื่นๆ มีลักษณะ Block Diagram ที่เหมือนกันแตกต่างกันตรงที่ key เท่านั้น)

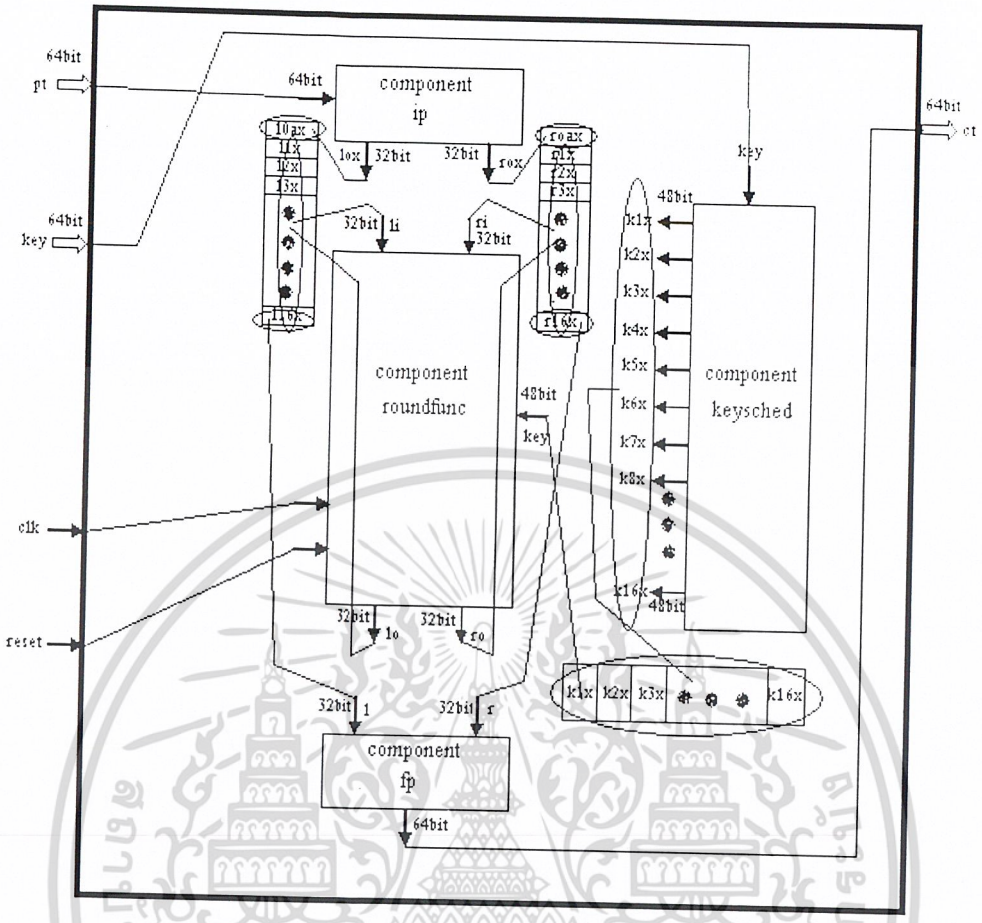
❑ **การออกแบบ DES Encryption**

ในส่วนของการออกแบบบล็อกไดอะแกรมของ DES Encryption ทำการออกแบบตามรูปที่ 5.5 รายละเอียดของแต่ละส่วนมีดังนี้

Input ประกอบด้วย pt (plaintext) ขนาด 64 bit ทำหน้าที่รับ input ที่เป็น plaintext ขนาด 64 bit

clk (clock) ขนาด 1 bit ทำหน้าที่เป็นสัญญาณ clock ให้แก่ระบบ เพื่อให้ระบบนั้นทำงาน synchronize กัน

Reset ขนาด 1 bit ทำหน้าที่เป็นสัญญาณ reset ให้แก่ระบบ เพื่อให้ระบบ ทำงาน synchronize กัน



รูปที่ 5.5 Block diagram DES Encryption

Output ประกอบด้วย ct (ciphertext) ขนาด 64 bit เป็น output สำหรับส่ง output ที่ได้ จาก ระบบที่ทำ Encrypt ออกไปสู่ภายนอกระบบ

Component ภายใน ip component ทำหน้าที่รับ input ขนาด 64 bit จาก pt มาทำการ initial permutation

fp component รับ input จาก l และ r อย่างละ 32 bit รวมเป็น 64 bit มาทำการ final permutation

keysched component รับ input ขนาด 64 bit จาก key มาสร้าง subkey ขนาด 48 bit จำนวน 16 key (ซึ่งเก็บไว้ที่ k1x - k16x) สำหรับแต่ละรอบของ roundfunc ทั้ง 16 รอบ

roundfunc component รับ input จาก ip component และ keysched component เพื่อทำการ คำนวณและวนรอบครบทั้ง 16 รอบ ตามข้อกำหนดของ algorithm แบบ DES เมื่อครบแล้วจะส่ง outputไปให้กับ component fp

ตัวแปร มีดังนี้ L0ax-l16x, r0ax-r16x ขนาด 32 bit และ k1x-k16x ขนาด 48 bit

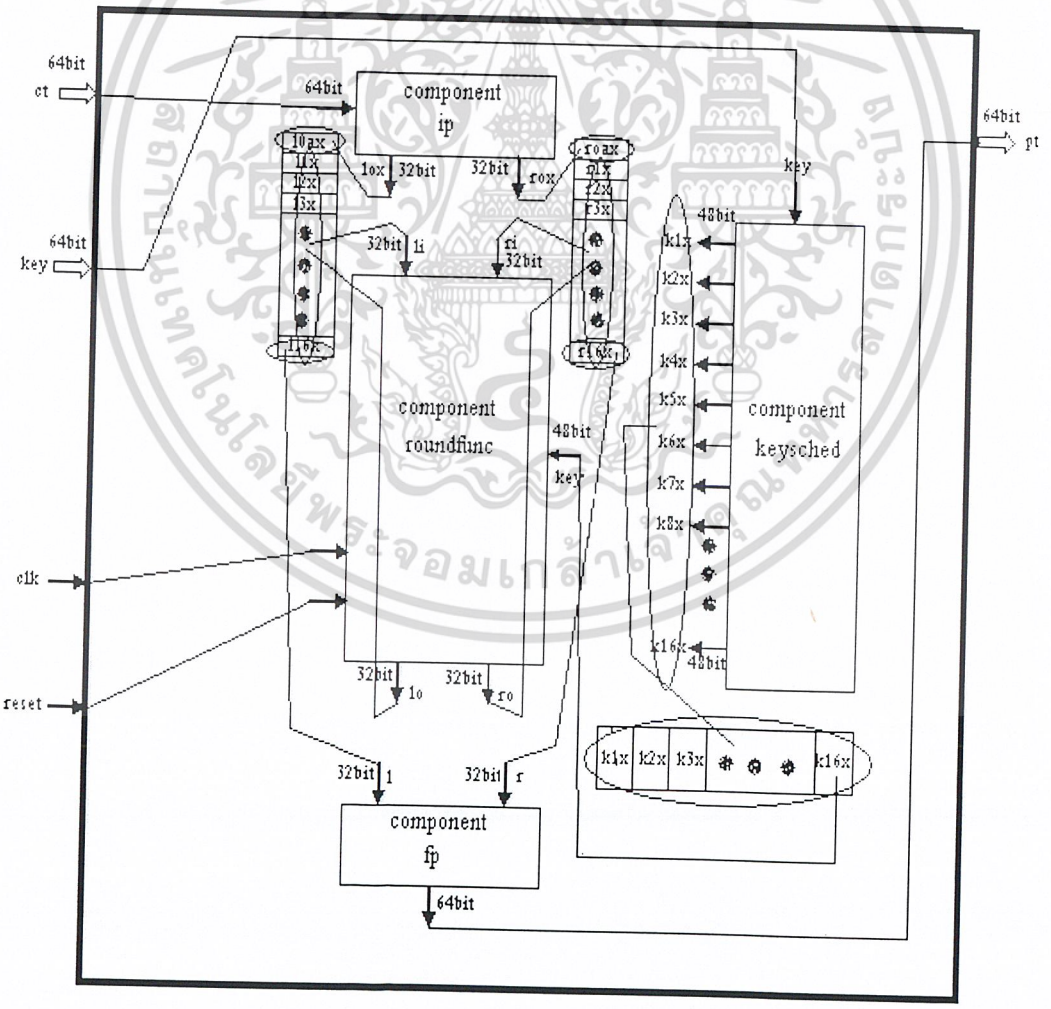
❑ การออกแบบ **DES Decryption**

ในส่วนของการออกแบบบล็อกไดอะแกรมของ DES Decryption ทำการออกแบบตามรูปที่ 5.6 รายละเอียดของแต่ละส่วน องค์ประกอบหลักๆ จะคล้ายกับ ส่วนของ Encryption แต่ที่แตกต่างกันก็มีดังนี้คือ

Input นั้นประกอบด้วย ct (ciphertext) ขนาด 64 bit ทำหน้าที่รับ input ที่เป็น ciphertext ขนาด 64 bit

Output ประกอบด้วย pt (plaintext) ขนาด 64 bit เป็น output สำหรับส่ง output ที่ได้จากระบบที่ทำ Decrypt ออกไปสู่ภายนอกระบบ(ได้ข้อมูลเป็นตัวเดียวกับ ตัวที่เข้ามาใน Des Encryption)

component ภายใน จะเหมือนกันและทำงานแบบเดียวกัน แต่จะต่างกันตรงที่ subkey แรกที่จะให้กับรอบที่หนึ่งของ roundfunc นั้นจะเป็น subkey ที่ 16 ก่อน

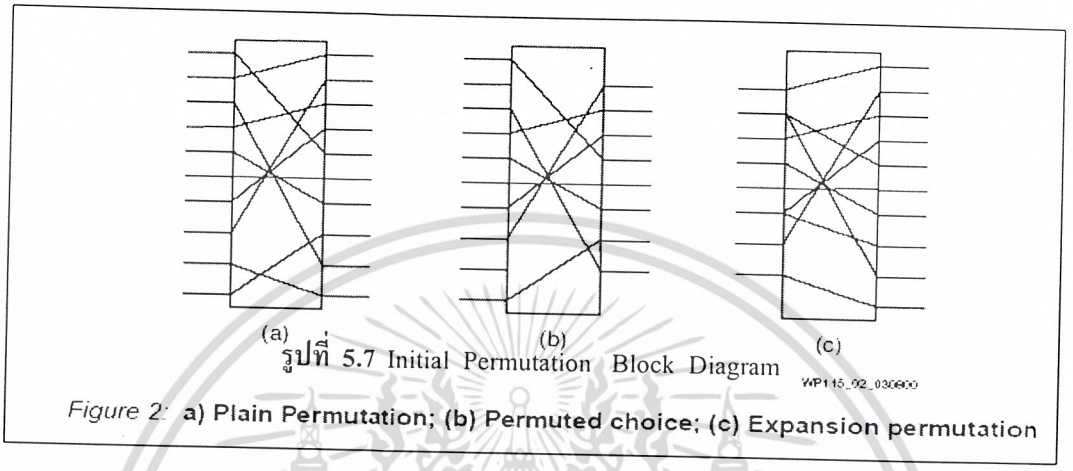


รูปที่ 5.6 Block diagram DES Decryption

5.2.1.3 ออกแบบการทำงานของแต่ละ Component

IP Component

ที่ทำหน้าที่ Initail Permutation เป็นการสลับบิตของข้อมูลเท่านั้น สามารถใช้การ wire สายได้



รูปที่ 5.7 Initial Permutation Block Diagram

Figure 2: **a) Plain Permutation; (b) Permuted choice; (c) Expansion permutation**

ลักษณะการสลับบิตของ IP Component เป็นดังนี้

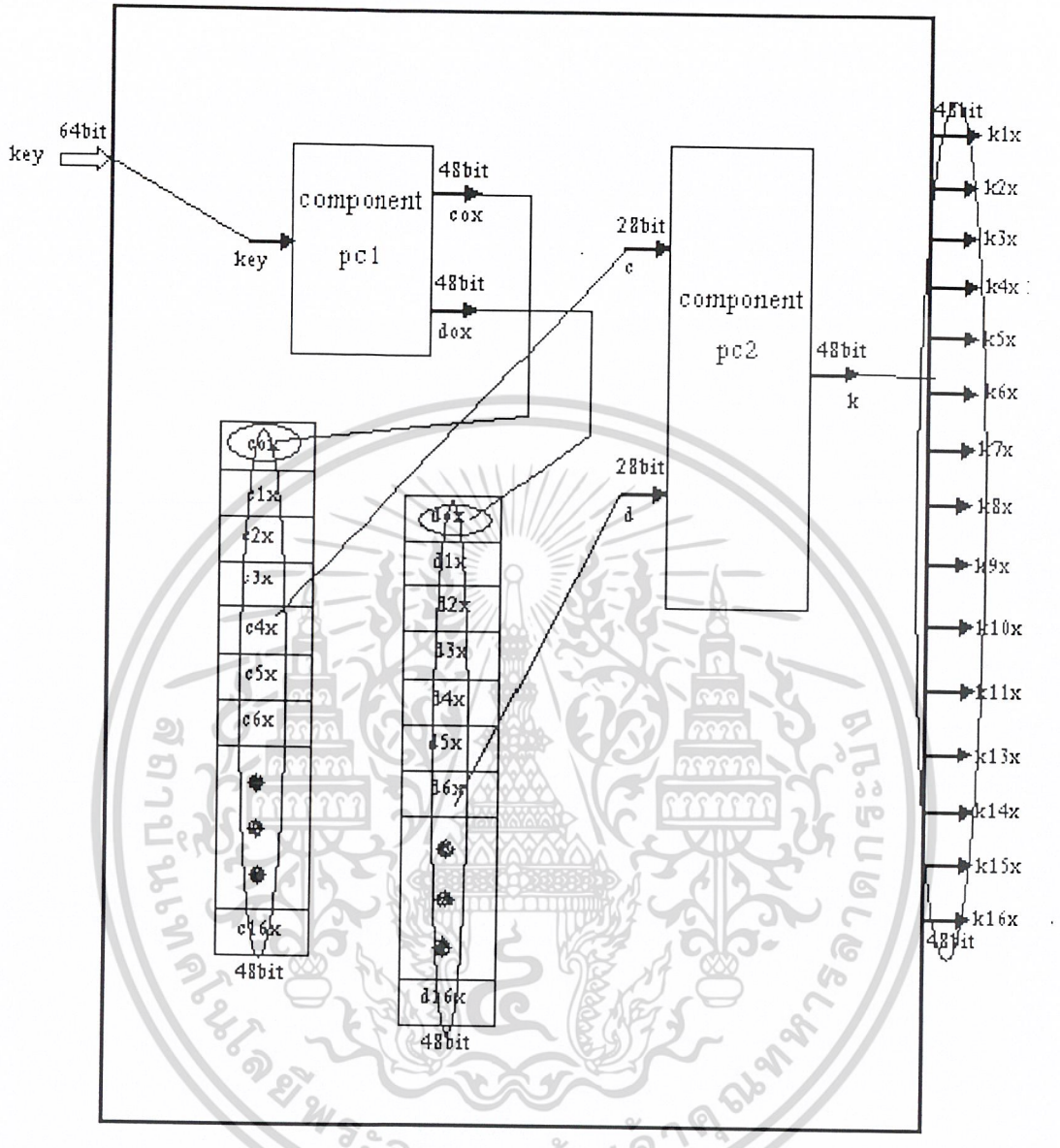
IP

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

Initial Permutation มีหลายรูปแบบ จากรูปที่ 5.7 (a) จะถูกใช้ใน IP Component ส่วนในรูป 5.7 (b) ถูกใช้ใน keyschedualing Component และ รูป 5.7 (c) ถูกใช้ใน roundfunction Component

Keysched Component

ทำหน้าหลักคือการ assign key แต่ละรอบให้แก่ component roundfunc ให้ครบทั้ง 16 รอบตามการทำงานของ Algorithm แบบ Des ใช้หลักการ wire สายเหมือนกัน ซึ่งจะมีความซับซ้อนกว่าสอง component แรกอยู่มาก ดังนั้นต้องแบ่งการทำงานของมันแยกออกเป็น อีก 2 component คือ component pc1 และ component pc2 ซึ่งทั้งสอง component นี้ เป็นการ permutation จะได้ block diagram ดังรูปที่ Block diagram keysched component



รูปที่ **5.8** Keysched Block Diagram

Component keysched นี้ จะทำงานเลยโดยไม่ต้องรอ สัญญาณ clk กล่าวคือมันจะให้ outpout ตั้งแต่ k1x จนถึง k16x ออกมาเลยเมื่อมี key เข้ามาที่ output keysched นี้มี component ย่อยภายในอยู่ 2 component คือ

i> Componet pc1 (สำหรับ ตัด bit ที่ 8, 16, 24, 32, 40, 48, 56, 64 ออกตามหลักการของ DES) ดังนี้

PC-1

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

ii> **Component pc2** เป็นการนำเอา bit ที่ได้จากการ shift จาก c และ d มารวมกันแล้วทำงาน permute ดังนี้

PC-2

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

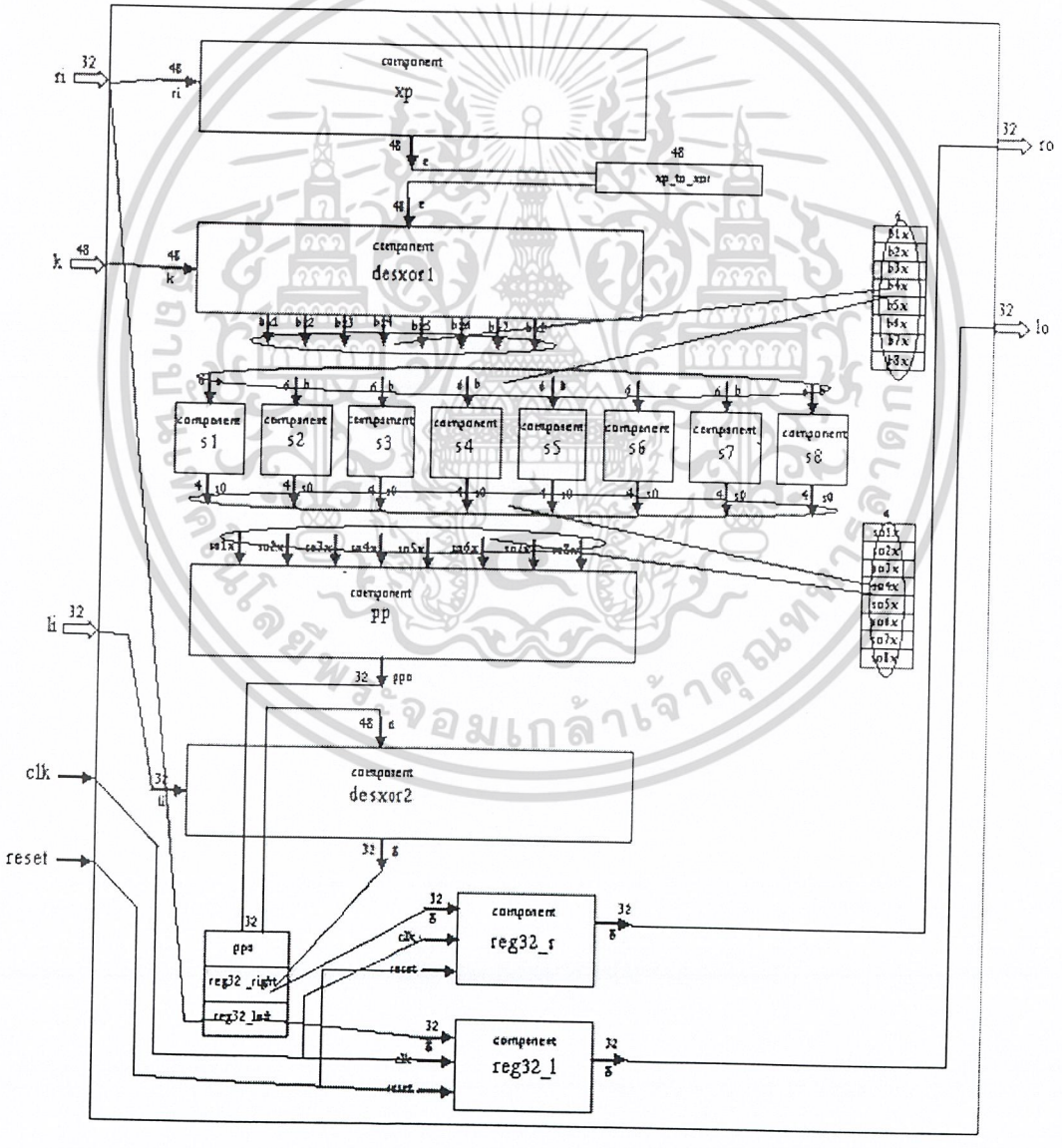
component keysched จะมีการทำงานโดยการ shif ซ้ายค่าใน cox และ dox ไปจนถึง c16x และ d16x ตามลำดับซึ่งจำนวนของการทำ Left shifs ในแต่ละครั้งจะเป็นไปตามนี้

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Roundfunc Component

Rounfunc Component ส่วนนี้มีการทำงานที่ซับซ้อน จึงต้องออกแบบโดยแตก ออก เป็น component ย่อยๆ ได้หลายส่วน ดังนี้

1. component xp
2. component desxor1
3. component s1 จนถึง s8
4. component pp
5. component desxor2
6. compon reg32



รูปที่ 5.9 Roundfunc Block Diagram

ส่วนประกอบต่าง ๆ ภายใน Roundfunc Component และการทำงานภายในมีดังนี้

- **component xp** รับ ri (32 bit) แล้วทำ Permute แบบขยาย เพื่อเป็น 48 bit

E BIT-SELECTION TABLE

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

- **component desxor1** เพื่อรับ input เป็น key (48bit) มา xor กับ output ที่มาจาก xp component

- **component s1 จนถึง s8** เป็นการ component เพื่อเลือกข้อชุดใดออกมา โดยจะมี input ขนาด 8bitและoutput ที่ออกมา ขนาด 4 bit การสร้าง component s1 ถึง s8 สร้างจาก

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

- **component pp** สร้างดังนี้

P

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

- **component desxor2** เอา output ที่ได้จาก component pp มา xor กับที่มาจาก ri การทำงานทั้งจะเป็นไปตามสมการ

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

- **compon reg32** กำหนดให้ว่า การทำงานต้องทำงานที่ขอบขาขึ้นของสัญญาณ clk

FP Component

ที่ทำหน้าที่ Final Permutation เป็นการสลับบิตของข้อมูลเท่านั้น สามารถใช้การ wire สายได้ ลักษณะการสลับบิตของ IP Component เป็นดังนี้

IP^{-1}

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

5.2.2 โปรแกรมภาษา VHDL

ในครั้งแรก จะต้องโปรแกรมให้กับแต่ละ Component ก่อน มีส่วนที่จะต้องโปรแกรมดังนี้

5.2.2.1 IP Component

5.2.2.2 Keysched Component

5.2.2.3 Roundfunc Component

5.2.2.4 FP Component

หลังจากนั้นจึงทำการรวม Component ต่างๆ เข้าด้วยกัน เป็น DES Encryption หนึ่งบล็อก และ DES Decryption อีกหนึ่งบล็อก

ขั้นตอนต่อมาจะทำการโปรแกรมเพื่อรวมเอา DES Encryption และ DES Decryption เข้าด้วยกันให้ได้ตามบล็อกไดอะแกรมในรูปที่ 5.2 และ 5.2 จึงถือเป็นการเสร็จสิ้นการการออกแบบทั้งกระบวนการ (รายละเอียดของโปรแกรมภาษา VHDL ตามภาคผนวก ก)

บทที่ 6
การทดลองและสรุปผล

ในบทนี้เป็นการทดลองวงจรการเข้ารหัสและถอดรหัสข้อมูลที่ได้ออกแบบ และการสรุปผล ในส่วนแรกจะเป็นการแสดงผลการทดลองโดยจะทำการทดลองแต่ละส่วนที่ได้ออกแบบ เปรียบเทียบกันในหลายๆองค์ประกอบเพื่อให้ทราบความแตกต่างว่าการทดลองที่ได้มีข้อดี ข้อเสีย ข้อผิดพลาดอย่างไร ผลการทดลองจะแสดงเป็นรูปภาพเพื่อความเข้าใจและเห็นภาพ จากนั้นจะนำข้อมูลที่ได้จากการทดลองมาเปรียบเทียบกับการทดลองของผู้ค้นคว้าวิจัยในต่างประเทศ เพื่อทำการเปรียบเทียบว่าผลการทดลองที่ได้เหมือนหรือต่างกันอย่างไร สามารถสร้างวงจรลงบนเอฟพีจีเอได้หรือไม่ สุดท้ายจะทำการสรุปผลการทดลอง โดยจะเสนอรายละเอียดในส่วนของการทดลองที่ได้ทั้งหมดว่า ผลสุดท้ายในการทดลองของโครงงานครั้งนี้ได้ข้อมูลโดยรวมเป็นอย่างไร มีการแสดงผลการทดลองเป็นตารางเพื่อเปรียบเทียบข้อมูลให้ชัดเจนยิ่งขึ้น

6.1 การทดลอง

จะทำการทดลองในสองส่วนด้วยกันคือ

- การจำลองการทำงานด้วย ModelSim SE Plus 6.0
- การ Implement ลงบนอุปกรณ์ FPGA

6.1.1 การจำลองการทำงานด้วย ModelSim SE Plus 6.0

ขั้นตอนการจำลองการทำงานด้วย ModelSim SE PLUS 6.0 จะต้องเขียนไฟล์ Test Bench เพื่อเป็นอินพุทให้กับโปรแกรมที่เขียนขึ้น โดยจะทำการทดสอบรับค่าอินพุทและแสดงผลออกทางเอ้าท์พุทให้ทราบ แต่ละโปรแกรมก็จะมีไฟล์ Test Bench สำหรับแต่ละโปรแกรม ในหัวข้อนี้จะทำการจำลองการทำงานของแต่ละส่วน เพื่อให้เห็นการทำงานอย่างชัดเจน และเปรียบเทียบความแตกต่างในการทำงาน ดังนี้

- DES Encryption
- DES Decrytion
- DES Encryption & Decrytion
- TDES Encryption
- TDES Decrytion
- TDES Encryption & Decrytion

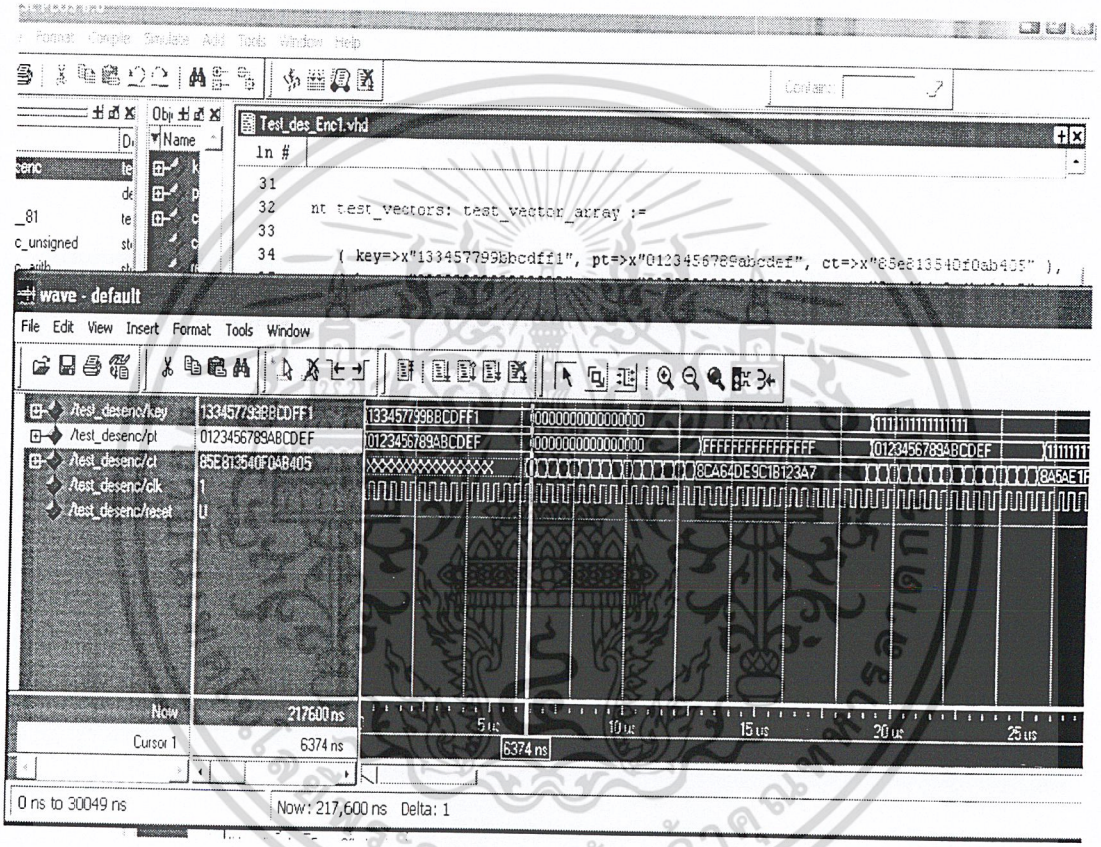
6.1.1.1 DES Encryption

ไฟล์ Test Bench กำหนดค่าอินพุท plain text , key และค่าเอ้าท์พุท cipher text และจำลองการทำงานตามรูปที่ 6.1 ดังนี้

Input => Plain text : 0123456789ABCDEF

Key : 133457799BBCDFF1

Output => Cipher text : 85E813540F0AB405



รูปที่ 6.1 ผลการจำลองการทำงานของ DES Encryption

ผลการจำลองการทำงานหลังจากที่ได้ Run โปรแกรม ModelSim SE PLUS 6.0 โปรแกรมจะนำค่าอินพุทต่างๆ ที่ได้กำหนดในไฟล์ Test Bench ไปทำการจำลองเป็นค่าอินพุทของโปรแกรม และแสดงผลเป็นเอ้าท์พุท เมื่อโปรแกรมทำงานครบตามกระบวนการก็จะให้เอ้าท์พุทออกมาตามรูปที่ 6.1

จากรูปที่ 6.1 เมื่อป้อนอินพุท plaintext => 0123456789ABCDEF และ key => 133457799BBCDFF1 เอ้าท์พุทที่ได้จะเป็นตามที่เรากำหนดไว้ในไฟล์ Test Bench คือ ได้ cipher => 85E813540F0AB405

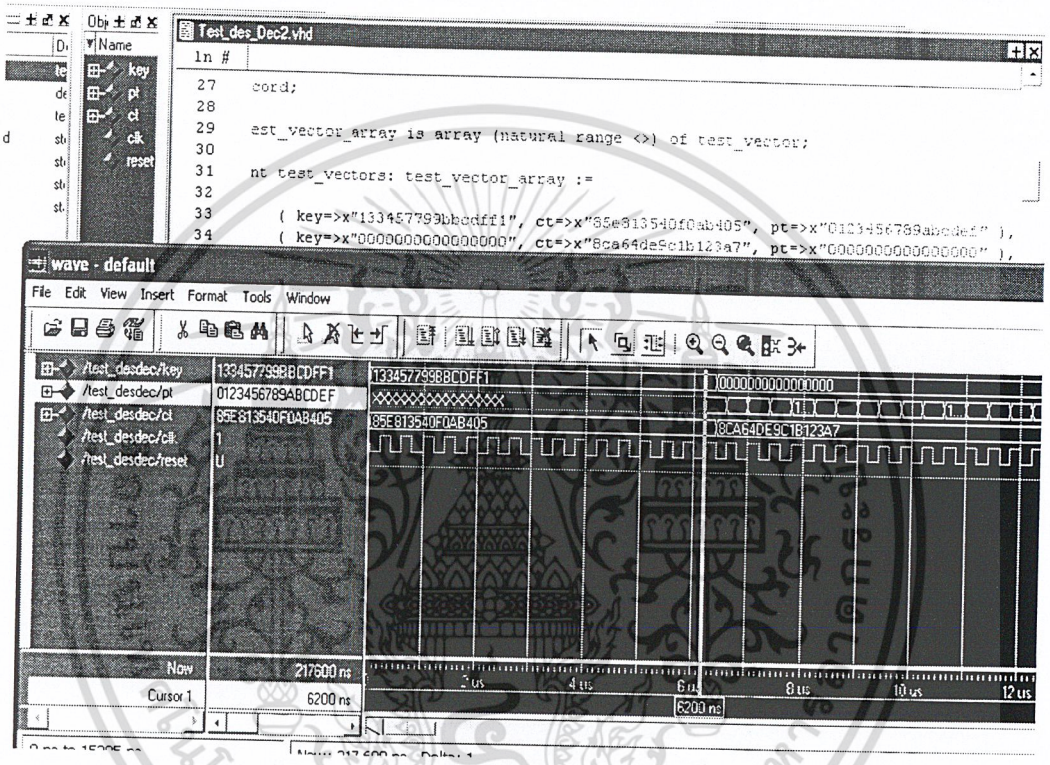
6.1.1.2 DES Decryption

ไฟล์ Test Bench กำหนดค่าอินพุท cipher text , key และค่าเอ้าท์พุท plain text และจำลองการทำงานตามรูปที่ 6.2 ดังนี้

Input => Cipher text : 85E813540F0AB405

Key : 133457799BBCDFF1

Output => Plain text : 0123456789ABCDEF



รูปที่ 6.2 ผลการจำลองการทำงานของ DES Decryption

ผลการจำลองการทำงานหลังจากที่ได้ Run โปรแกรม ModelSim SE PLUS 6.0 โปรแกรมจะนำค่าอินพุทต่างๆ ที่ได้กำหนดในไฟล์ Test Bench ไปทำการจำลองเป็นค่าอินพุทของโปรแกรม และแสดงผลเป็นเอ้าท์พุท เมื่อโปรแกรมทำงานครบตามกระบวนการก็จะให้เอ้าท์พุทออกมาตามรูปที่ 6.2

จากรูปที่ 6.2 เมื่อป้อนอินพุท cipher => 85E813540F0AB405 และ key => 133457799BBCDFF1 เอ้าท์พุทที่ได้จะเป็นตามที่เรากำหนดไว้ในไฟล์ Test Bench คือ ได้ plaintext => 0123456789ABCDEF

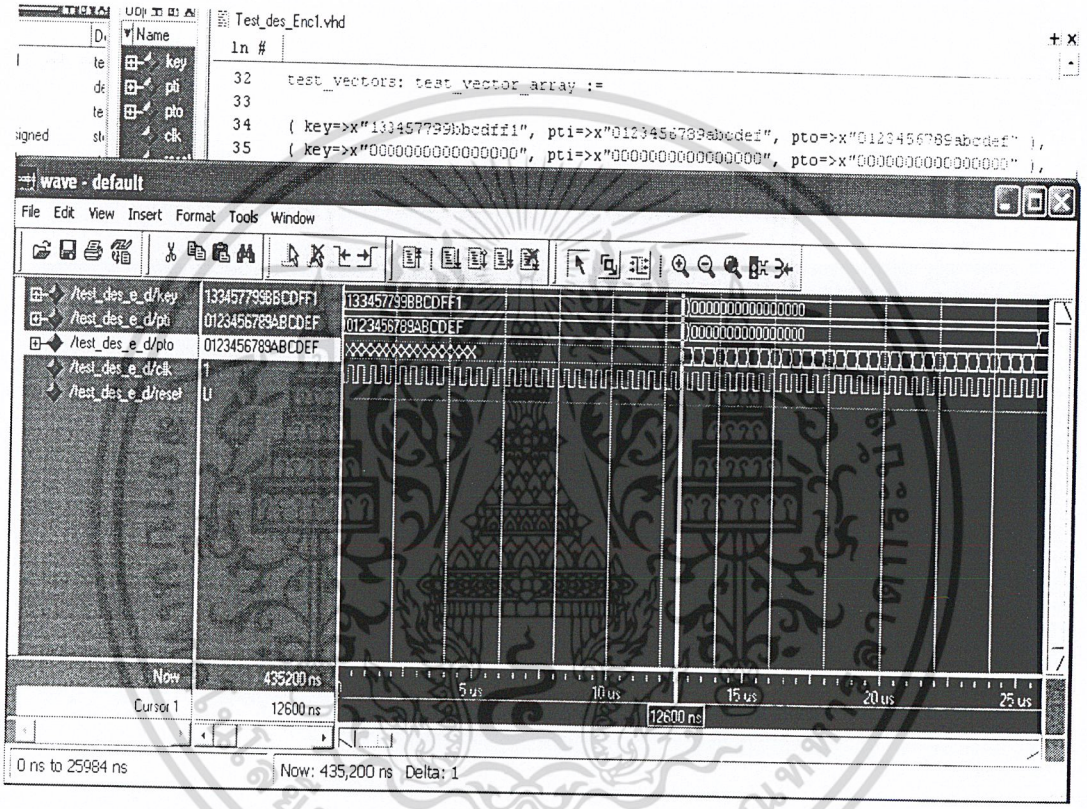
6.1.1.3 DES Decryption & Encryption

ไฟล์ Test Bench กำหนดค่าอินพุท plain text , key และค่าเอ้าท์พุท cipher text และจำลองการทำงานตามรูปที่ 6.3 ดังนี้

Input => Plain text In : 0123456789ABCDEF

Key : 133457799BBCDFF1

Output => Plain text Out : 0123456789ABCDEF



รูปที่ 6.3 ผลการจำลองการทำงานของ DES Encryption & Decryption

ผลการจำลองการทำงานหลังจากที่ได้ Run โปรแกรม ModelSim SE PLUS 6.0 โปรแกรมจะนำค่าอินพุทต่างๆ ที่ได้กำหนดในไฟล์ Test Bench ไปทำการจำลองเป็นค่าอินพุทของโปรแกรม และแสดงผลเป็นเอ้าท์พุท เมื่อโปรแกรมทำงานครบตามกระบวนการก็จะให้เอ้าท์พุทออกมาตามรูปที่ 6.3

จากรูปที่ 6.3 เมื่อป้อนอินพุท Plain text In => 0123456789ABCDEF และ key => 133457799BBCDFF1 เอ้าท์พุทที่ได้จะเป็นตามที่เรากำหนดไว้ในไฟล์ Test Bench คือ ได้ Plain text Out => 0123456789ABCDEF

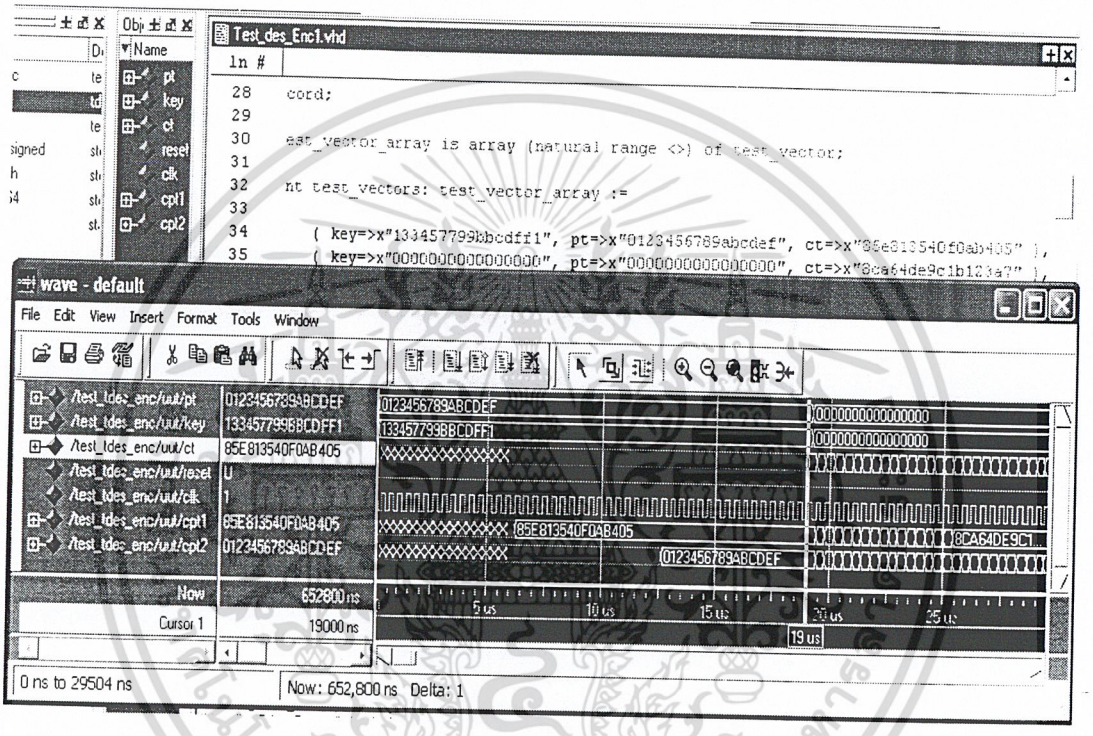
6.1.1.4 TDES Encryption

ไฟล์ Test Bench กำหนดค่าอินพุท plain text , key และค่าเอ้าท์พุท cipher text และจำลองการทำงานตามรูปที่ 6.4 ดังนี้

Input => Plain text : 0123456789ABCDEF

Key : 133457799BBCDFF1

Output => Cipher text : 85E813540F0AB405



รูปที่ 6.4 ผลการจำลองการทำงานของ TDES Encryption

ผลการจำลองการทำงานหลังจากที่ได้ Run โปรแกรม ModelSim SE PLUS 6.0 โปรแกรมจะนำค่าอินพุทต่างๆ ที่ได้กำหนดในไฟล์ Test Bench ไปทำการจำลองเป็นค่าอินพุทของโปรแกรม และแสดงผลเป็นเอ้าท์พุท เมื่อโปรแกรมทำงานครบตามกระบวนการก็จะให้เอ้าท์พุทออกมาตามรูปที่ 6.4

จากรูปที่ 6.4 เมื่อป้อนอินพุท plaintext => 0123456789ABCDEF และ key => 133457799BBCDFF1 เอ้าท์พุทที่ได้จะเป็นตามที่เรากำหนดไว้ในไฟล์ Test Bench คือ ได้ cipher => 85E813540F0AB405

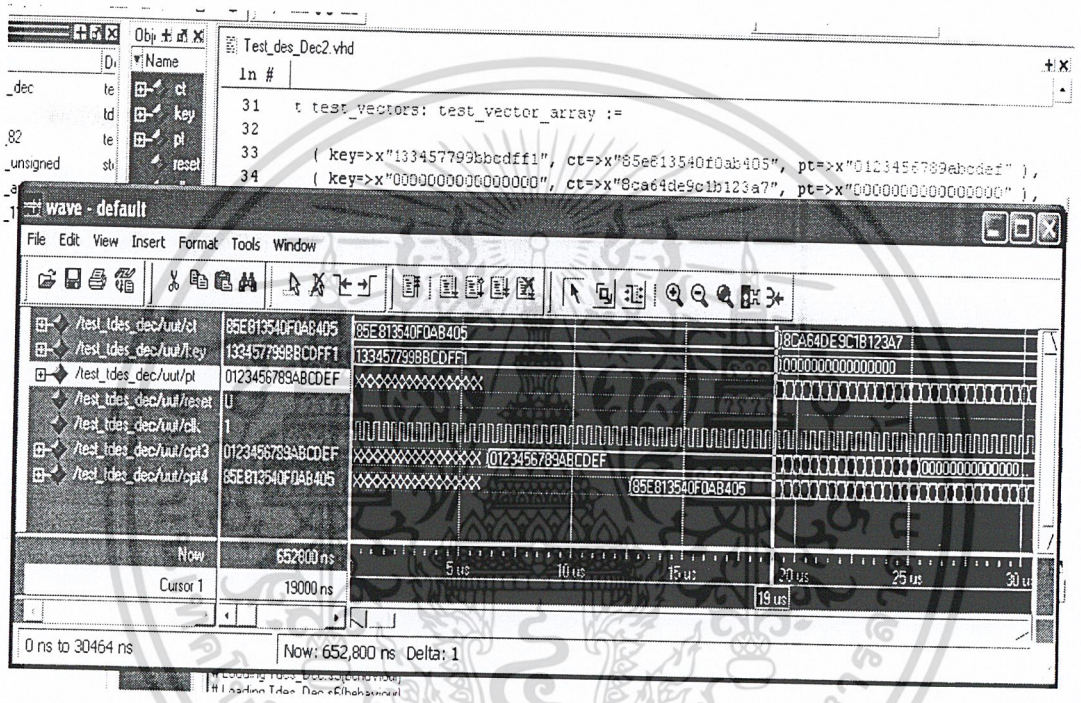
6.1.1.5 TDES Decryption

ไฟล์ Test Bench กำหนดค่าอินพุท cipher text , key และค่าเอ้าท์พุท plain text และจำลองการทำงานตามรูปที่ 6.5 ดังนี้

Input => Cipher text : 85E813540F0AB405

Key : 133457799BBCDFF1

Output => Plain text : 0123456789ABCDEF



รูปที่ 6.5 ผลการจำลองการทำงานของ TDES Decryption

ผลการจำลองการทำงานหลังจากที่ได้ Run โปรแกรม ModelSim SE PLUS 6.0 โปรแกรมจะนำค่าอินพุทต่างๆ ที่ได้กำหนดในไฟล์ Test Bench ไปทำการจำลองเป็นค่าอินพุทของโปรแกรม และแสดงผลเป็นเอ้าท์พุท เมื่อโปรแกรมทำงานครบตามกระบวนการก็จะให้เอ้าท์พุทออกมาตามรูปที่ 6.5

จากรูปที่ 6.5 เมื่อป้อนอินพุท cipher => 85E813540F0AB405 และ key => 133457799BBCDFF1 เอ้าท์พุทที่ได้จะเป็นตามที่เรากำหนดไว้ในไฟล์ Test Bench คือ ได้ plaintext => 0123456789ABCDEF

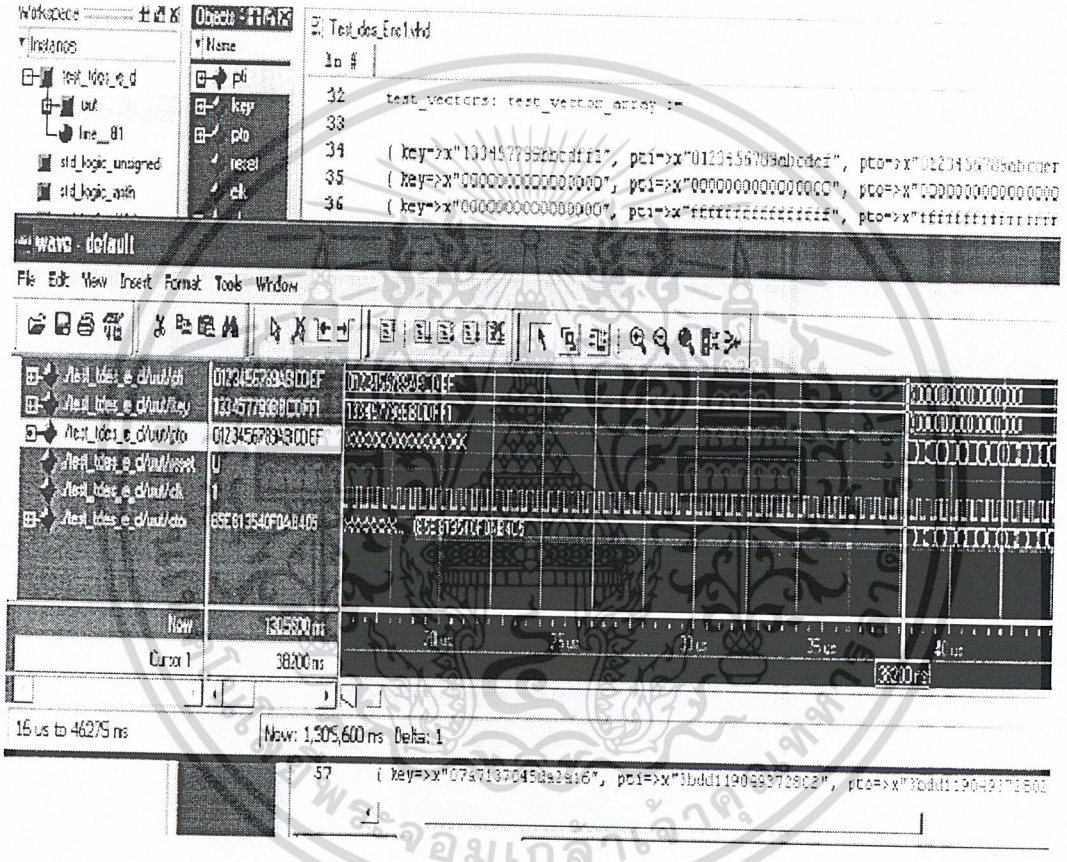
6.1.1.6 TDES Decryption & Encryption

ไฟล์ Test Bench กำหนดค่าอินพุท plain text , key และค่าเอ้าท์พุท cipher text และจำลองการทำงานตามรูปที่ 6.6 ดังนี้

Input => Plain text In : 0123456789ABCDEF

Key : 133457799BBCDFF1

Output => Plain text Out : 0123456789ABCDEF



รูปที่ 6.6 ผลการจำลองการทำงานของ TDES Encryption & Decryption

ผลการจำลองการทำงานหลังจากที่ได้ Run โปรแกรม ModelSim SE PLUS 6.0 โปรแกรมจะนำค่าอินพุทต่างๆ ที่ได้กำหนดในไฟล์ Test Bench ไปทำการจำลองเป็นค่าอินพุทของโปรแกรม และแสดงผลเป็นเอ้าท์พุท เมื่อโปรแกรมทำงานครบตามกระบวนการก็จะให้เอ้าท์พุทออกมาตามรูปที่ 6.6

จากรูปที่ 6.6 เมื่อป้อนอินพุท Plain text In => 0123456789ABCDEF และ key => 133457799BBCDFF1 เอ้าท์พุทที่ได้จะเป็นตามที่เรากำหนดไว้ในไฟล์ Test Bench คือ ได้ Plain text Out => 0123456789ABCDEF

6.1.2 การ Implement ลงบนอุปกรณ์ FPGA

การออกแบบระบบดิจิตอลด้วยโปรแกรมภาษา VHDL หลังจากที่ได้ออกแบบ เขียนโปรแกรมภาษา VHDL เพื่อสร้างวงจรตามที่ไดออกแบบแล้ว และทำการจำลองการทำงานเรียบร้อย แล้วขั้นตอนต่อไปคือการ Implement ลงบนฮาร์ดแวร์ ในส่วนของการ Implement มีการกระทำ 2 ขั้นตอนหลักๆ คือ การ Synthesis และการ Implement ในหัวข้อนี้จะแสดงผลการ Synthesis ให้เห็นเพียงเท่านั้น เพราะเมื่อสามารถ Synthesis ผ่านกระบวนการก็หมายความว่า สามารถ Implement ลงบนฮาร์ดแวร์ได้อย่างแน่นอน

การ Synthesis จะทำการ Synthesis โดยแบ่งเป็น 2 หัวข้อ ตามรุ่นของ FPGA เพื่อเปรียบเทียบ และหาข้อสรุปของการทดลองดังนี้

6.1.2.1 FPGA รุ่น Spartan III รุ่น XC3S200 tq144

การ Synthesis จะต้องกำหนดคุณสมบัติของอุปกรณ์ FPGA ที่จะทำการ Implement ลงไป ในที่นี้ได้กำหนดให้เป็น FPGA Spartan III รุ่น XC3S200 tq144 ของบริษัท XILINX ความหมายของตัวอักษรของรุ่นได้เสนอให้ทราบแล้วในบทที่ 4 แต่จะกล่าวถึงอีกครั้งเพื่อความเข้าใจยิ่งขึ้น แต่จะกล่าวถึงเฉพาะส่วนที่เกี่ยวข้องเท่านั้น

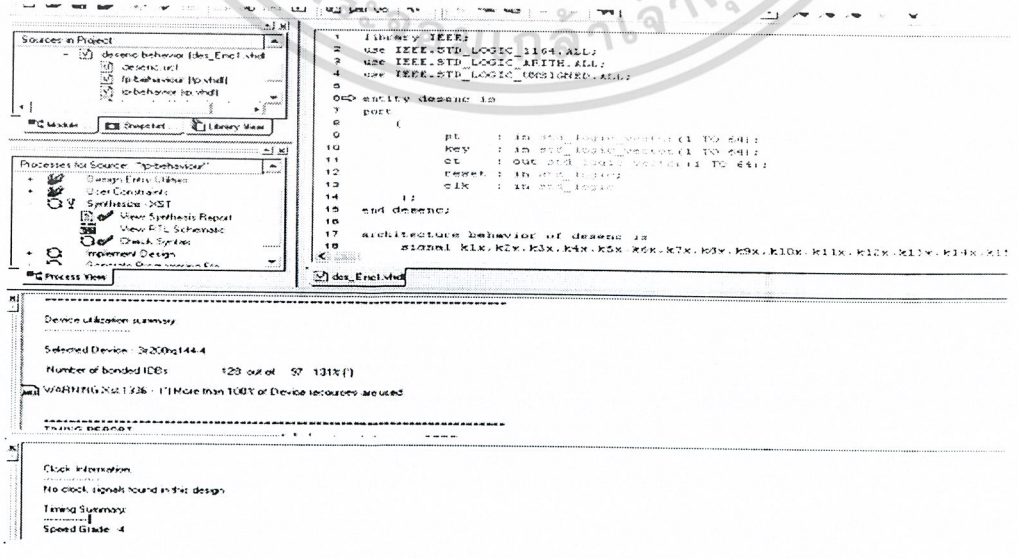
XC3S200 หมายถึง มีจำนวนเกท 200,000 ตัว

tq144 หมายถึง มีจำนวนขา 144 ขา

I> Synthesis แต่ละ Component

❑ **IP Component**

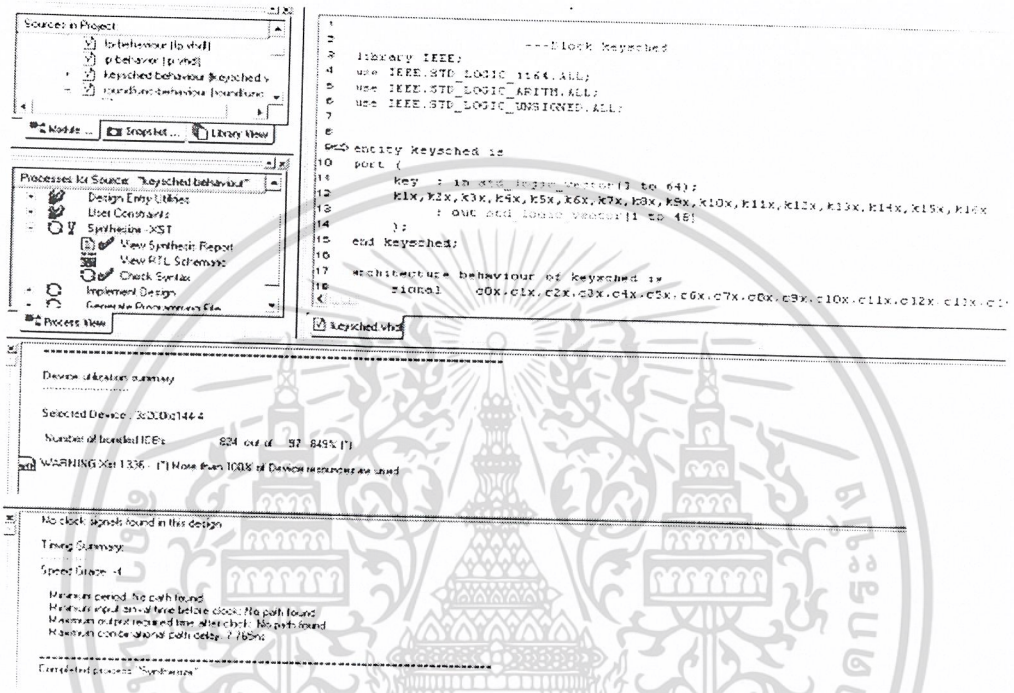
ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 128 แต่ FPGA รุ่นนี้มีเพียง 97 เท่านั้น (เกินไปถึง 131%) รายละเอียดตามรูปที่ 6.7



รูปที่ 6.7 ผลการ Synthesis IP Component

❑ **keysched Component**

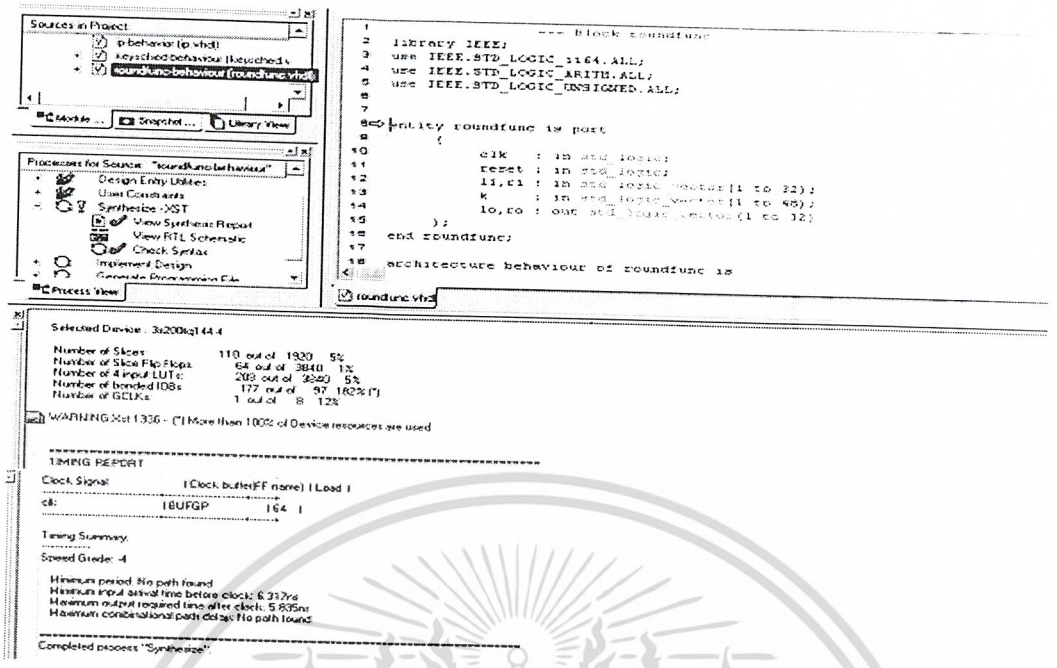
ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 824 แต่ FPGA รุ่นนี้มีเพียง 97 เท่านั้น (เกินไปถึง 849%) รายละเอียดตามรูปที่ 6.8



รูปที่ 6.8 ผลการ Synthesis keysched Component

❑ **roundfunc Component**

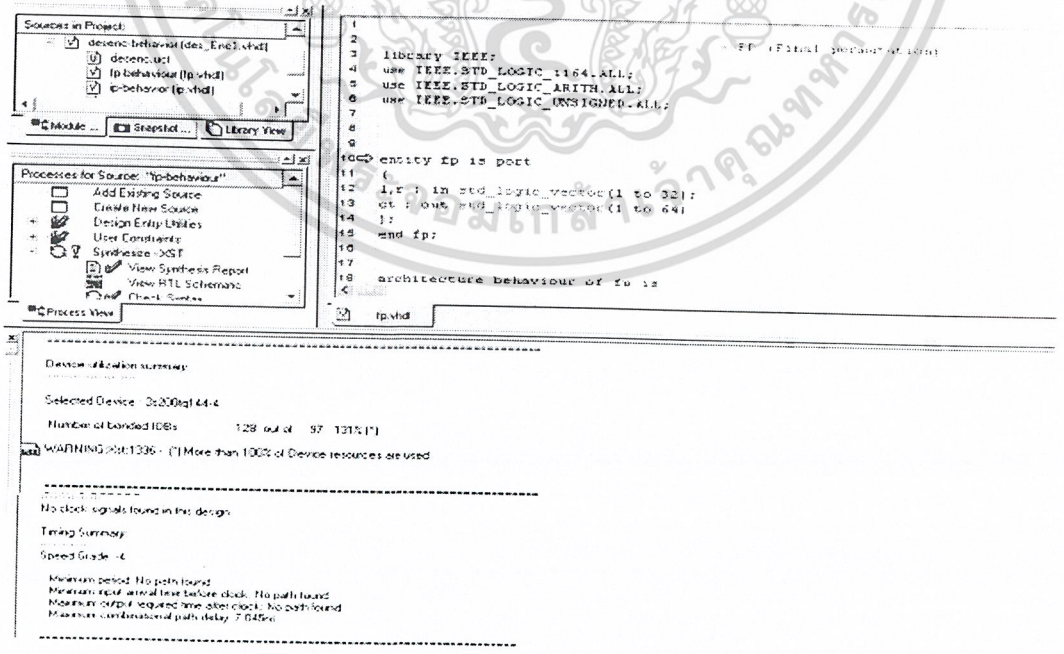
ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 177 แต่ FPGA รุ่นนี้มีเพียง 97 เท่านั้น (เกินไปถึง 182%) แต่ในส่วนอื่นเพียงตอความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.9



รูปที่ 6.9 ผลการ Synthesis Roundfunc Component

❑ FP Component

ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 128 แต่ FPGA รุ่นนี้มีเพียง 97 เท่านั้น (เกินไปถึง 131%) รายละเอียดตามรูปที่ 6.10



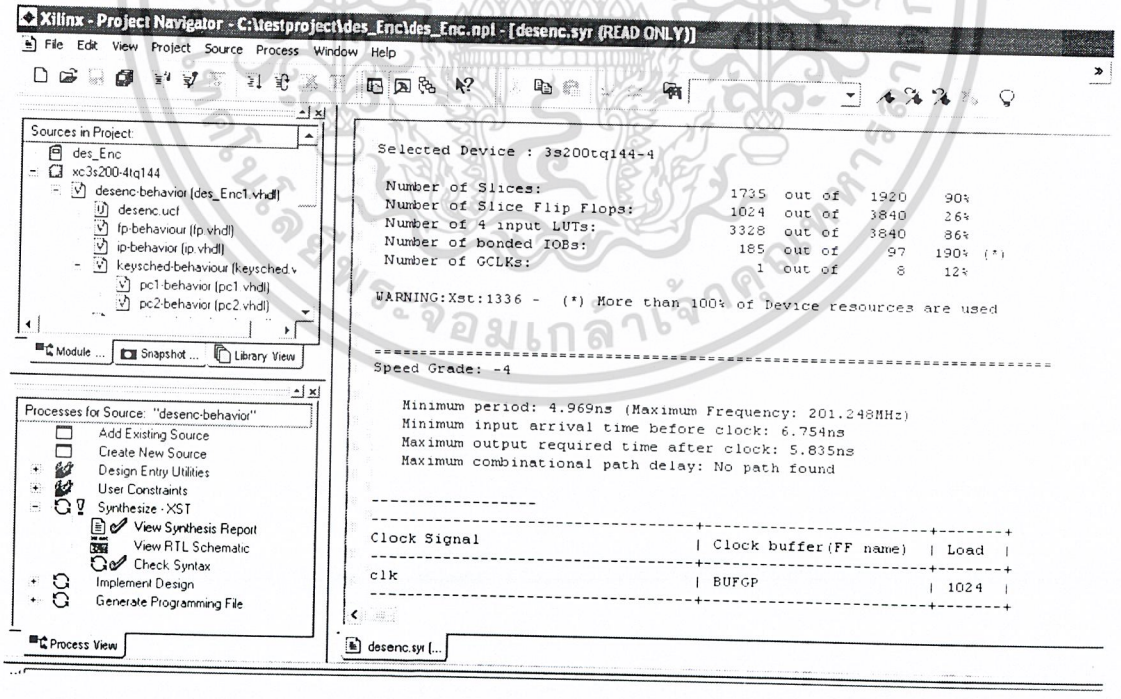
รูปที่ 6.10 ผลการ Synthesis FP Component

II> Synthesis DES

หัวข้อที่ผ่านมาจะทำการ Synthesis แต่ละ Component เพื่อดูข้อผิดพลาดของแต่ละ Component ซึ่งเป็นรูปแบบของการออกแบบลักษณะ Top Down Design คือออกแบบและทดสอบทีละส่วนให้สามารถทำงานได้แล้วจึงออกแบบระบบใหญ่ขึ้น หากเกิดปัญหาให้ทำการแก้ไขแล้วจึงทำการทดลองใหม่อีกครั้ง ในหัวข้อนี้จำทำการ Synthesis เป็นบล็อกไดอะแกรมที่ได้ออกแบบ ถึงแม้ว่าในส่วนของ Component ย่อย ยังมีส่วนที่มีปัญหาก็ตาม แต่ก็ยังสรุปไม่ได้ว่าเกิดปัญหาจากส่วนใด ซึ่งมีหลายองค์ประกอบที่ต้องพิจารณา การ Synthesis DES นี้ ยังคงใช้คุณสมบัติของ FPGA เดิมอยู่คือ FPGA รุ่น Spartan III รุ่น XC3S200 tq144 ของบริษัท XILINX

❑ **DES Encryption**

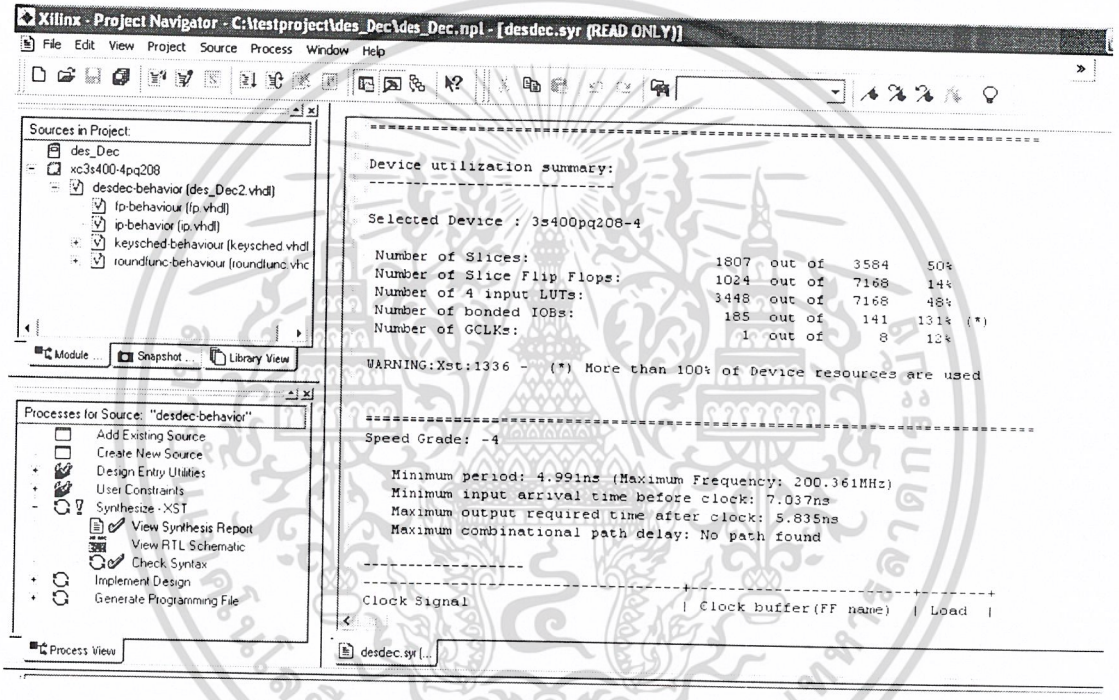
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 185 แต่ FPGA รุ่นนี้มีเพียง 97 เท่านั้น (เกินไปถึง 190%) แต่ในส่วนอื่นเพียงตอความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.11



รูปที่ 6.11 ผลการ Synthesis DES Encryption

❑ DES Decryption

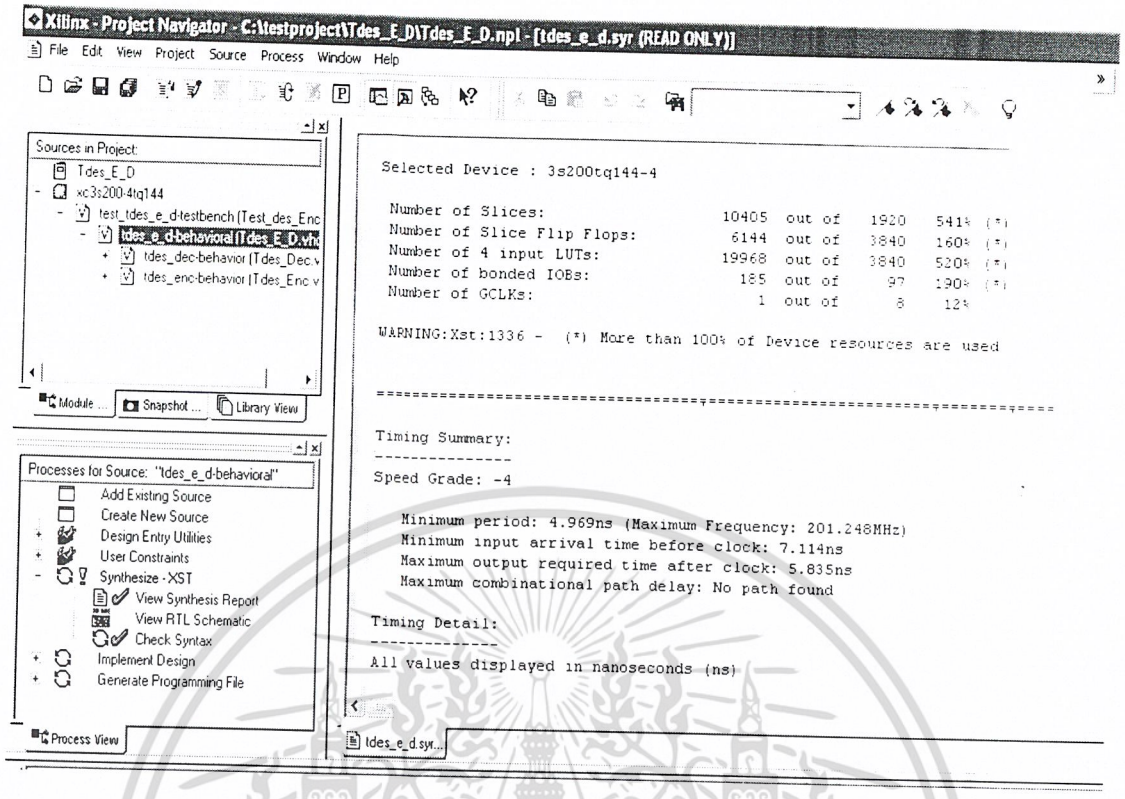
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 185 แต่ FPGA รุ่นนี้มีเพียง 97 เท่านั้น (เกินไปถึง 190%) แต่ในส่วนอื่นเพียงตอความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.12



รูปที่ 6.12 ผลการ Synthesis DES Decryption

❑ DES Encryption & Decryption

ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้แสดงให้เห็นปัญหาของการ Sythesis ในหลายส่วนด้วยกัน เนื่องจากคุณสมบัติของ FPGA ไม่เพียงพอต่อการออกแบบ สาเหตุเกิดขึ้นจาก 4 ส่วน คือ Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ ส่วน GCLKs ไม่มีปัญหาใดๆ สรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.13



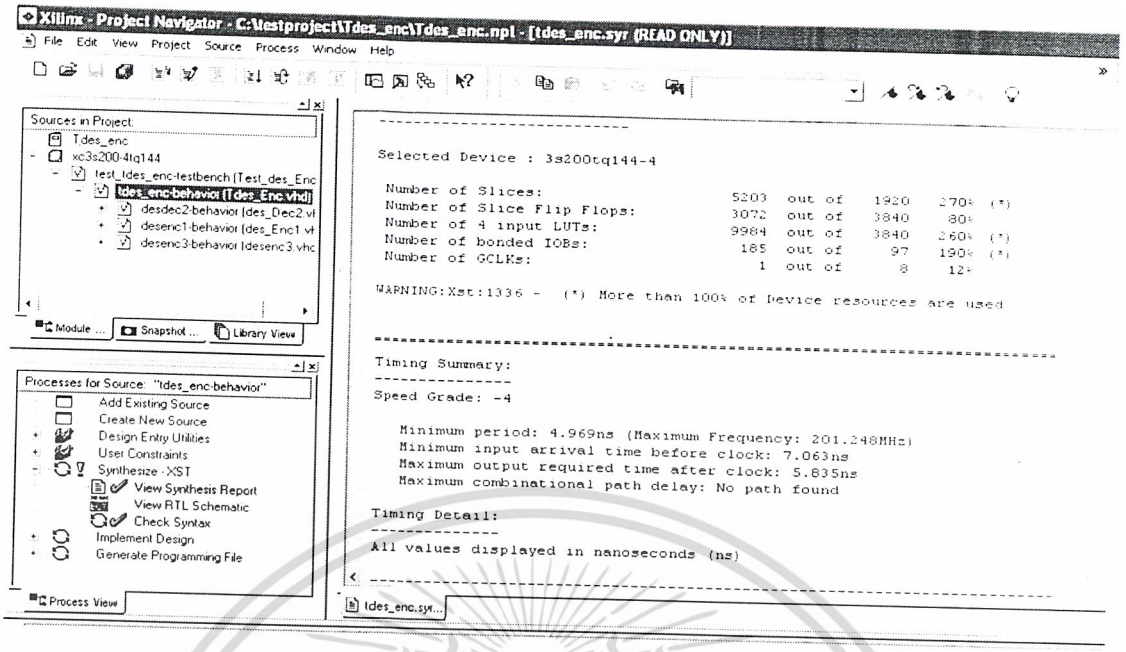
รูปที่ 6.13 ผลการ Synthesis DES Encryption & Descryption

III> Synthesis TDES

หัวข้อที่ผ่านมาจะทำการ Synthesis DES ซึ่งผลก็คือไม่สามารถทำการ Synthesis ให้ผ่านกระบวนการได้ แต่การทดลองยังดำเนินต่อไปเพื่อหาความแตกต่างของผลกระทบของคุณสมบัติของ FPGA ต่อการออกแบบ ในหัวข้อนี้จะทำการ Synthesis TDES และยังคงใช้คุณสมบัติของ FPGA เดิมอยู่คือ FPGA รุ่น Spartan III รุ่น XC3S200 tq144 ของบริษัท XILINX

❑ **TDES Encryption**

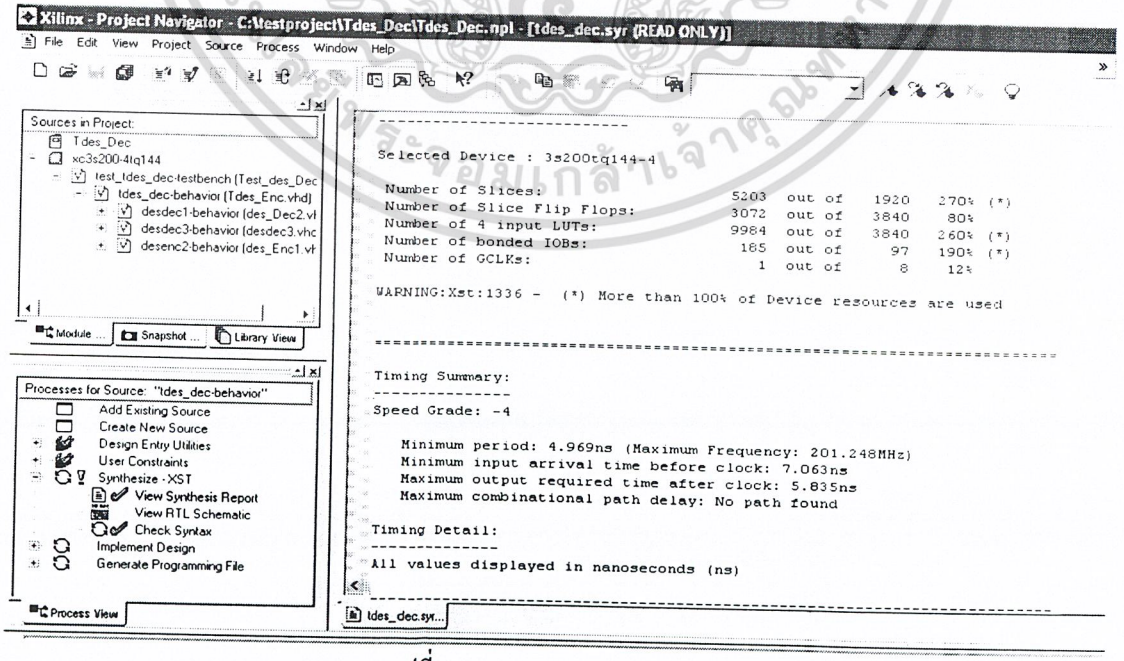
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้ มีปัญหาในส่วนของ Number of slices , Number of 4 input LUT และ IOBs ไม่เพียงพอต่อการออกแบบ แต่ในส่วนของ Number of Slices Flip-Flop และ GCLKs ไม่มีปัญหาและเพียงพอ โดยสรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.14



รูปที่ 6.14 ผลการ Synthesis TDES Encryption

❑ TDES Decryption

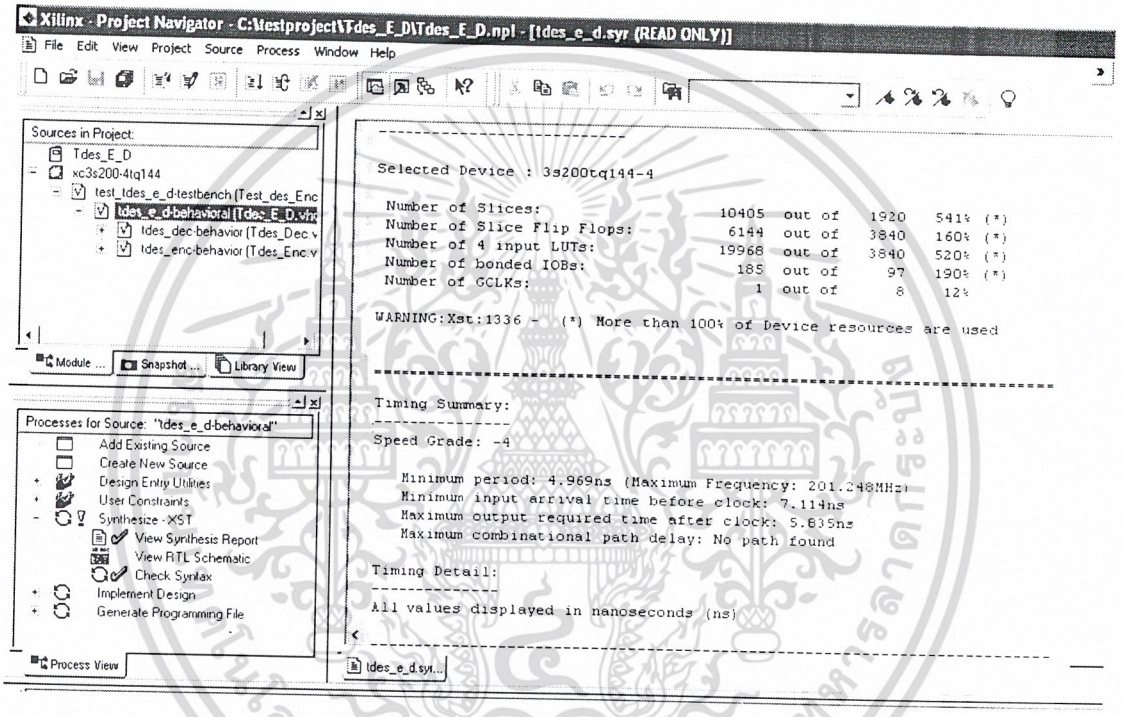
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้ มีปัญหาในส่วนของ Number of slices , Number of 4 input LUT และ IOBs ไม่เพียงพอต่อการออกแบบ แต่ในส่วนของ Number of Slices Flip-Flop และ GCLKs ไม่มีปัญหาและเพียงพอ โดยสรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.15



รูปที่ 6.15 ผลการ Synthesis TDES Decryption

❑ **TDES Encryption & Decryption**

ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้แสดงให้เห็นปัญหาของการ Sythesis ในหลายส่วนด้วยกัน เนื่องจากคุณสมบัติของ FPGA ไม่เพียงพอต่อการออกแบบ สาเหตุเกิดขึ้นจาก 4 ส่วน คือ Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ ส่วน GCLKs ไม่มีปัญหาใดๆ สรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.16



รูปที่ 6.16 ผลการ Synthesis TDES Encryption & Descryption

6.1.2.2 FPGA รุ่น Spartan III รุ่น XC3S400 pq208

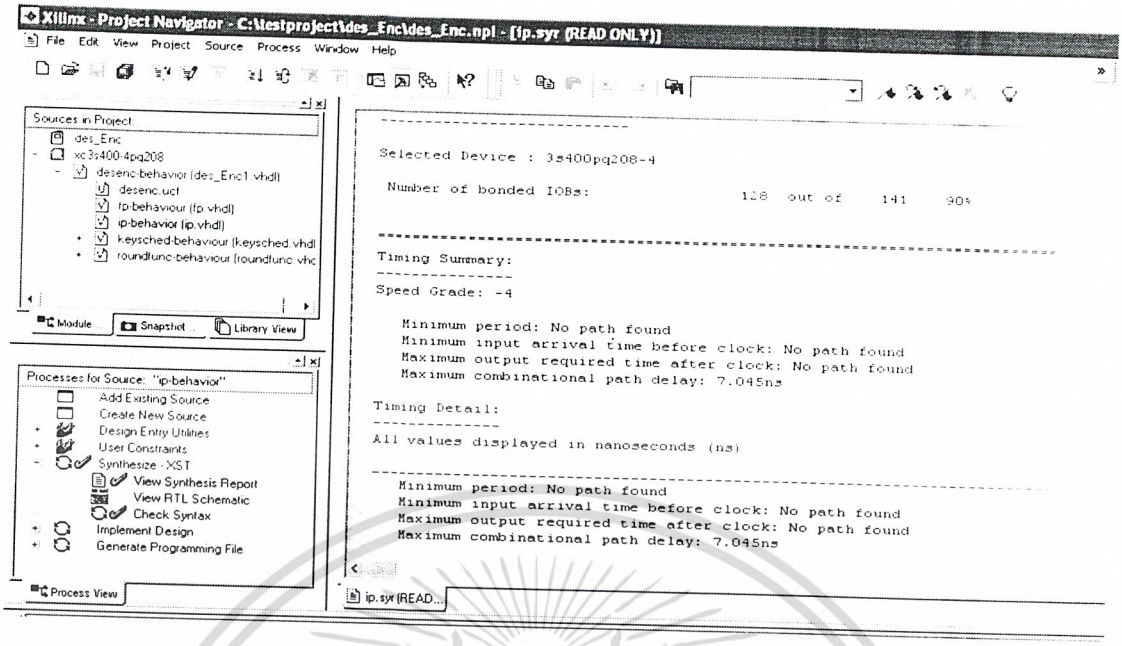
การ Synthesis ในหัวข้อนี้ใช้ FPGA Spartan III รุ่น XC3S400 pq208 ของบริษัท XILINX

XC3S400	หมายถึง	มีจำนวนเกท 400,000 ตัว
pq208	หมายถึง	มีจำนวนขา 208 ขา

I> Synthesis แต่ละ Component

❑ **IP Component**

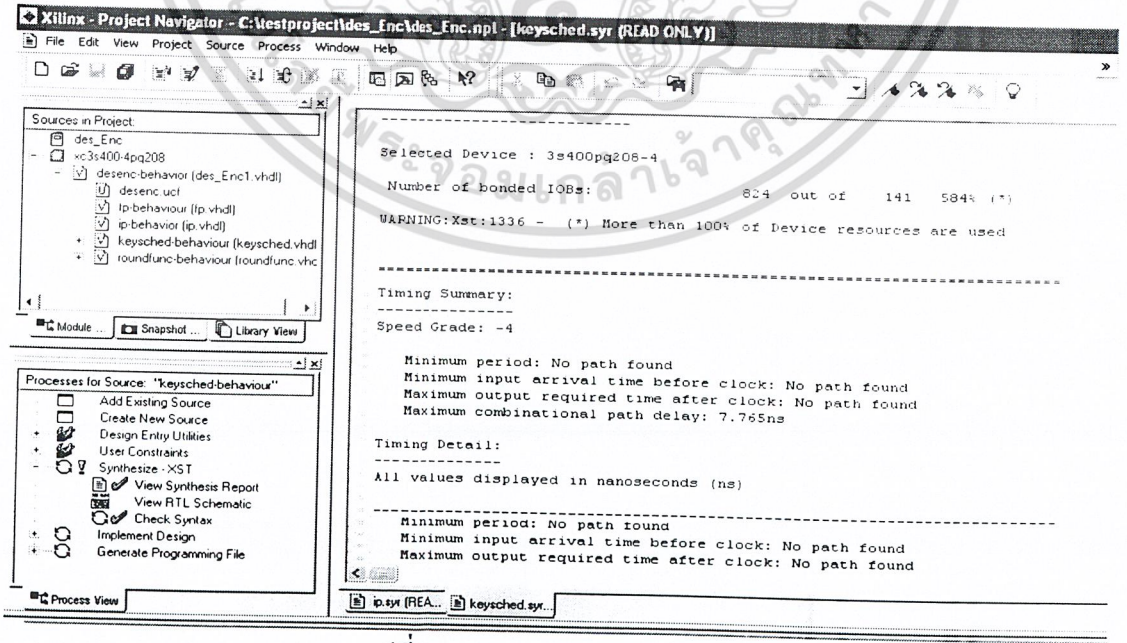
ผลการ Synthesis สามารถทราบได้ว่า สามารถทำการ Synthesis ผ่านกระบวนการ เพราะไม่มีเครื่องหมาย ! แสดงที่หน้า Synthesis-XST แสดงว่าสามารถ Implement ลงบน FPGA ได้



รูปที่ 6.17 ผลการ Synthesis IP Component

❑ keysched Component

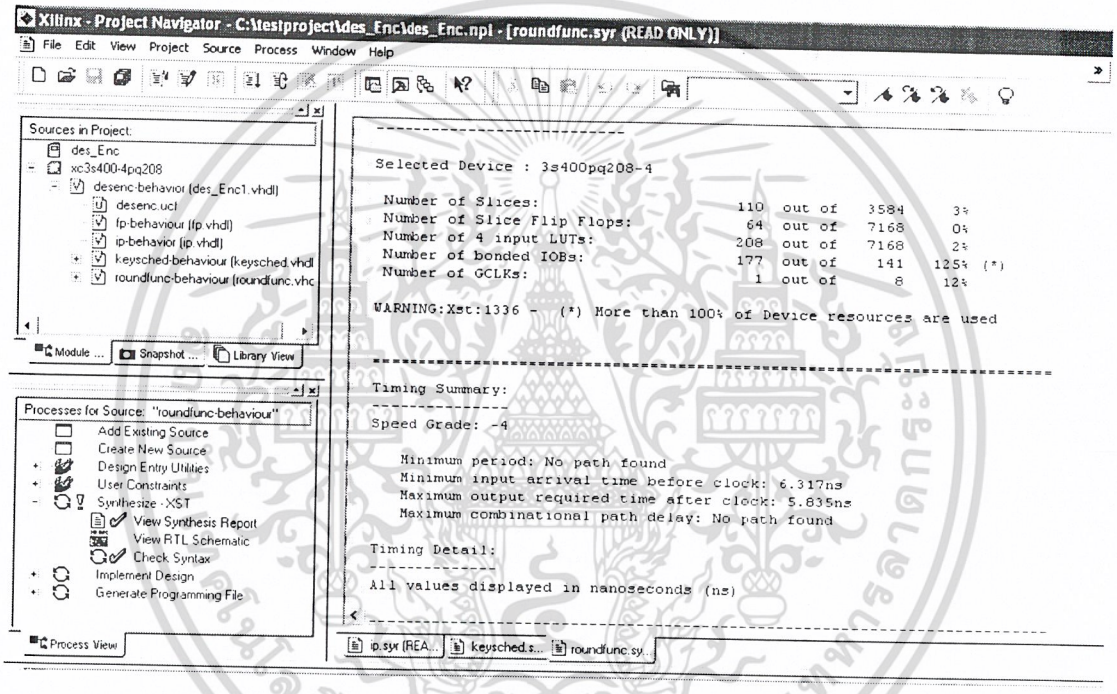
ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 824 แต่ FPGA รุ่นนี้มีเพียง 141 เท่านั้น (เกินไปถึง 584%) รายละเอียดตามรูปที่ 6.18



รูปที่ 6.18 ผลการ Synthesis keysched Component

❑ roundfunc Component

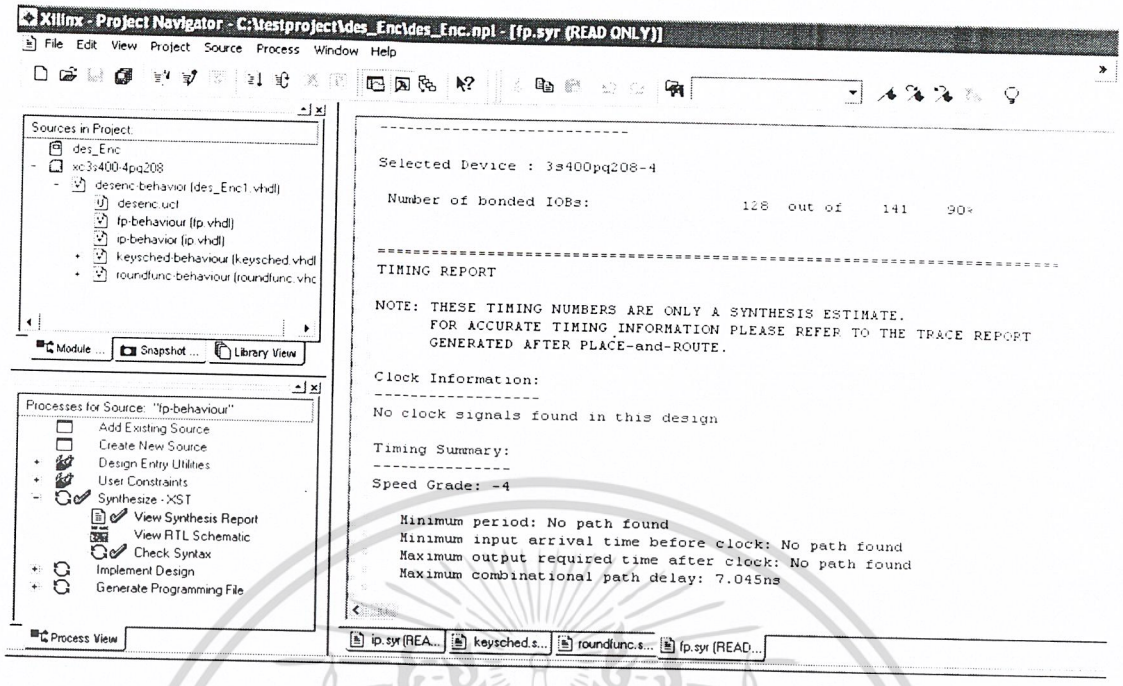
ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 177 แต่ FPGA รุ่นนี้มีเพียง 144 เท่านั้น (เกินไปถึง 125%) แต่ในส่วนอื่นเพียงตอความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.19



รูปที่ 6.19 ผลการ Synthesis Roundfunc Component

❑ FP Component

ผลการ Synthesis สามารถทราบได้ว่า ไม่สามารถทำการ Synthesis ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 128 แต่ FPGA รุ่นนี้มีเพียง 141 เท่านั้น (เกินไปถึง 131%) รายละเอียดตามรูปที่ 6.20



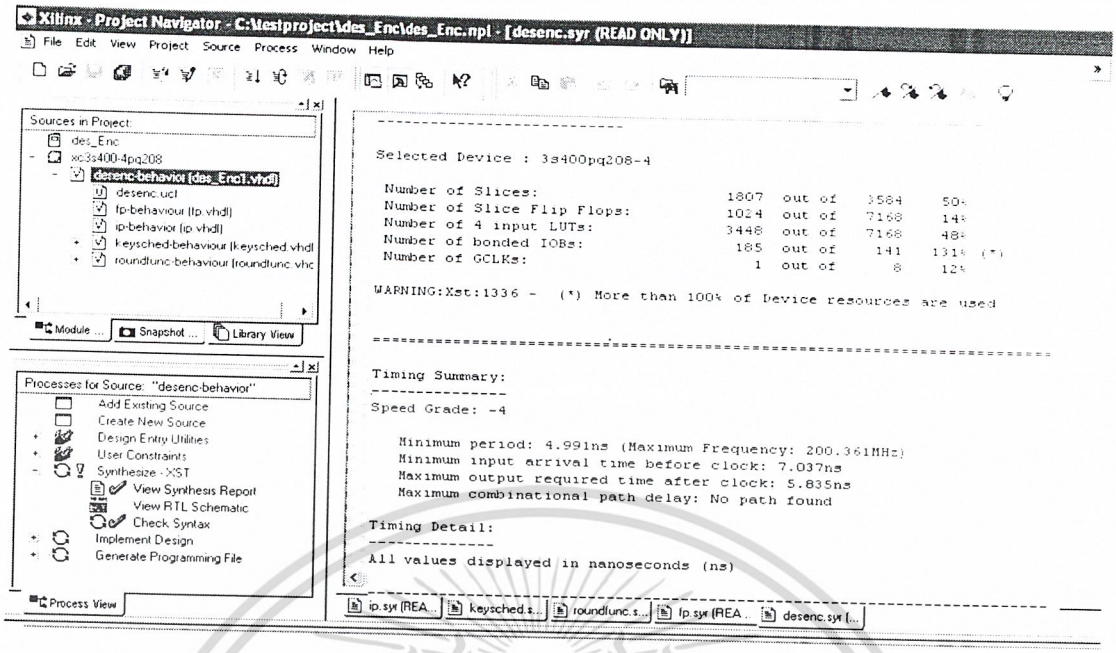
รูปที่ 6.20 ผลการ Synthesis FP Component

II> Synthesis DES

หัวข้อที่ผ่านมาจะทำการ Synthesis แต่ละ Component เพื่อดูข้อผิดพลาดของแต่ละ Component ในหัวข้อนี้จำทำการ Synthesis เป็นบล็อกไดอะแกรมที่ได้ออกแบบ ถึงแม้ว่าในส่วนของ Component ย่อย ยังมีส่วนที่มีปัญหาก็ตาม แต่ก็ยังสรุปไม่ได้ว่าเกิดปัญหาจากส่วนใด ซึ่งมีหลายองค์ประกอบที่ต้องพิจารณา การ Synthesis DES นี้ ยังคงใช้คุณสมบัติของ FPGA เดิมอยู่คือ FPGA รุ่น Spartan III รุ่น XC3S400 pq208 ของบริษัท XILINX

❑ **DES Encryption**

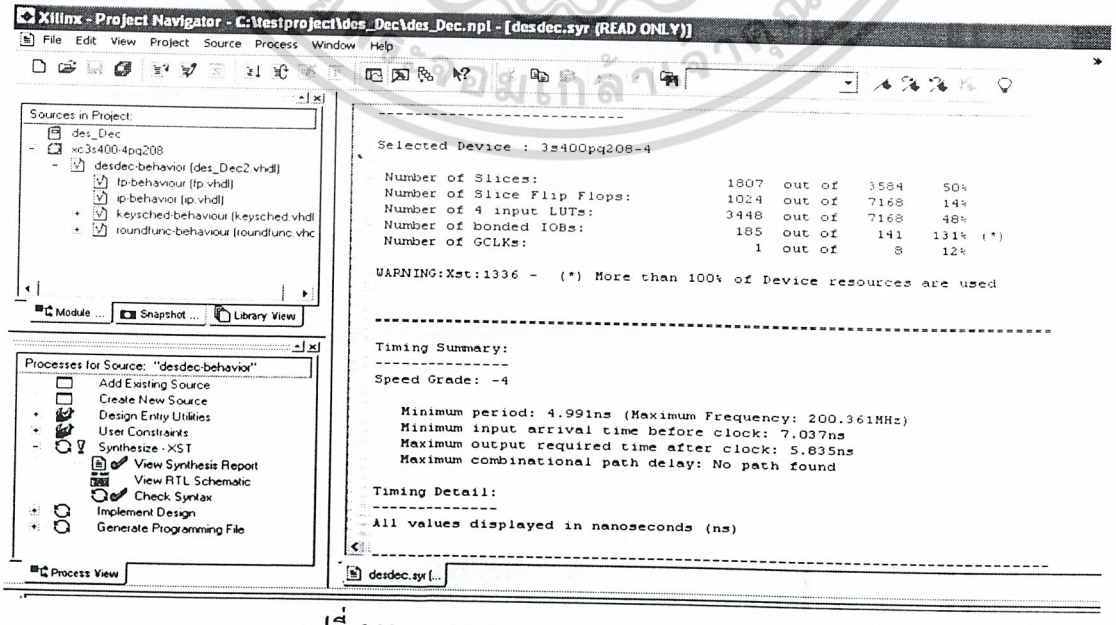
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 185 แต่ FPGA รุ่นนี้มีเพียง 141 เท่านั้น (เกินไปถึง 131%) แต่ในส่วนอื่นเพียงตอบความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.21



รูปที่ 6.21 ผลการ Synthesis DES Encryption

❑ DES Decryption

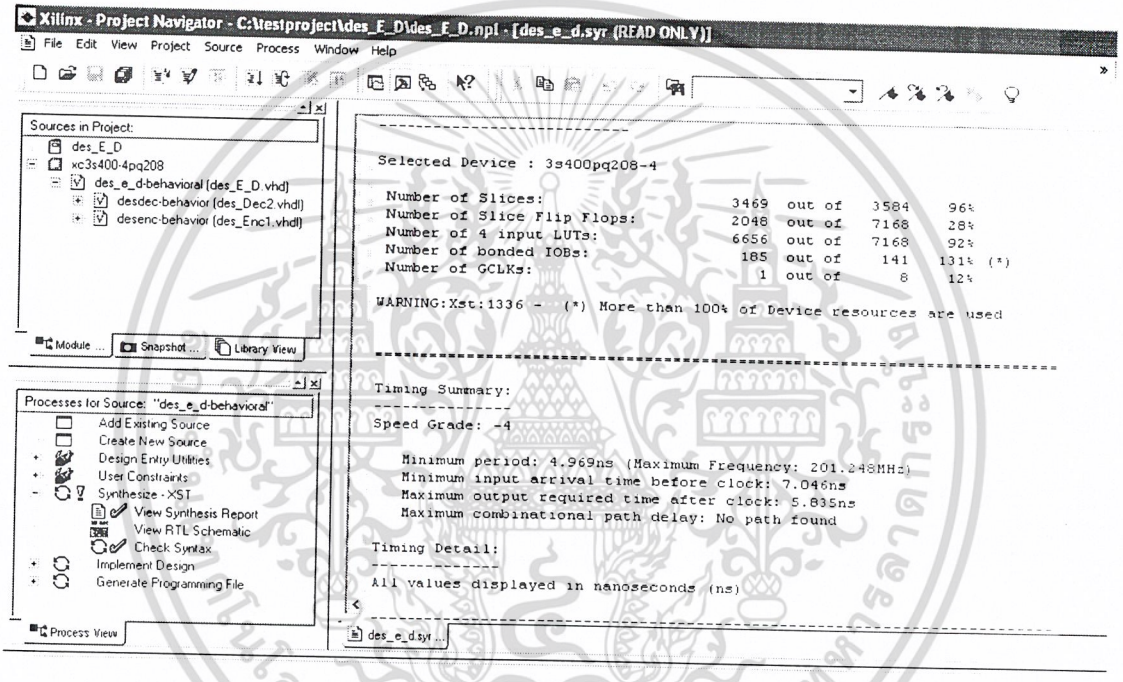
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 185 แต่ FPGA รุ่นนี้มีเพียง 141 เท่านั้น (เกินไปถึง 131%) แต่ในส่วนอื่นเพียงตอบความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.22



รูปที่ 6.22 ผลการ Synthesis DES Decryption

❑ **DES Encryption & Decryption**

ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST สาเหตุเนื่องจากมีการใช้ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ IOBs เท่ากับ 185 แต่ FPGA รุ่นนี้มีเพียง 141 เท่านั้น (เกินไปถึง 131%) แต่ในส่วนอื่นเพียงตอบความต้องการในการออกแบบ ไม่ว่าจะเป็น Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ GCLKs ไม่มีปัญหาใดๆ แต่โดยสรุป แม้ว่าทรัพยากรภายใน FPGA ส่วนอื่นๆจะเพียงพอ แต่ถ้ามีส่วนใดส่วนหนึ่งไม่เพียงพอแล้วก็จะทำให้การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.23



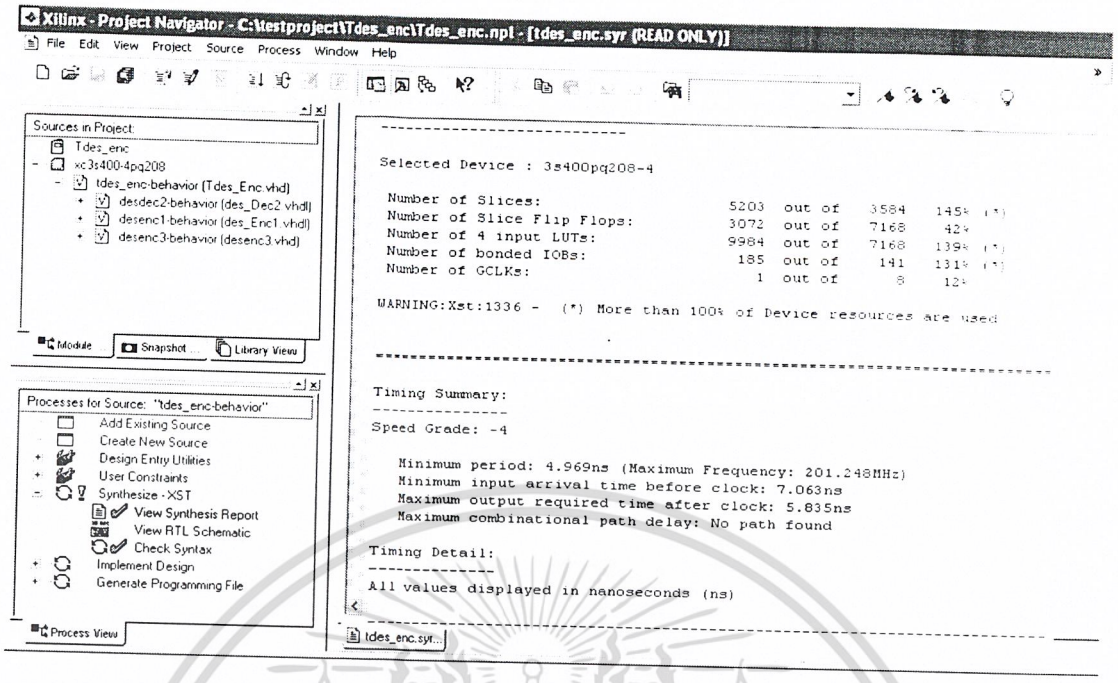
รูปที่ 6.23 ผลการ Synthesis DES Encryption & Descryption

III> Synthesis TDES

หัวข้อที่ผ่านมาจะทำการ Synthesis DES ซึ่งผลก็คือไม่สามารถทำการ Synthesis ให้ผ่านกระบวนการได้ แต่การทดลองยังดำเนินต่อไปเพื่อหาความแตกต่างของผลกระทบของคุณสมบัติของ FPGA ต่อการออกแบบ ในหัวข้อนี้จะทำการ Synthesis TDES และยังคงใช้คุณสมบัติของ FPGA เดิมอยู่คือ FPGA รุ่น Spartan III รุ่น XC3S400 pq144 ของบริษัท XILINX

❑ **TDES Encryption**

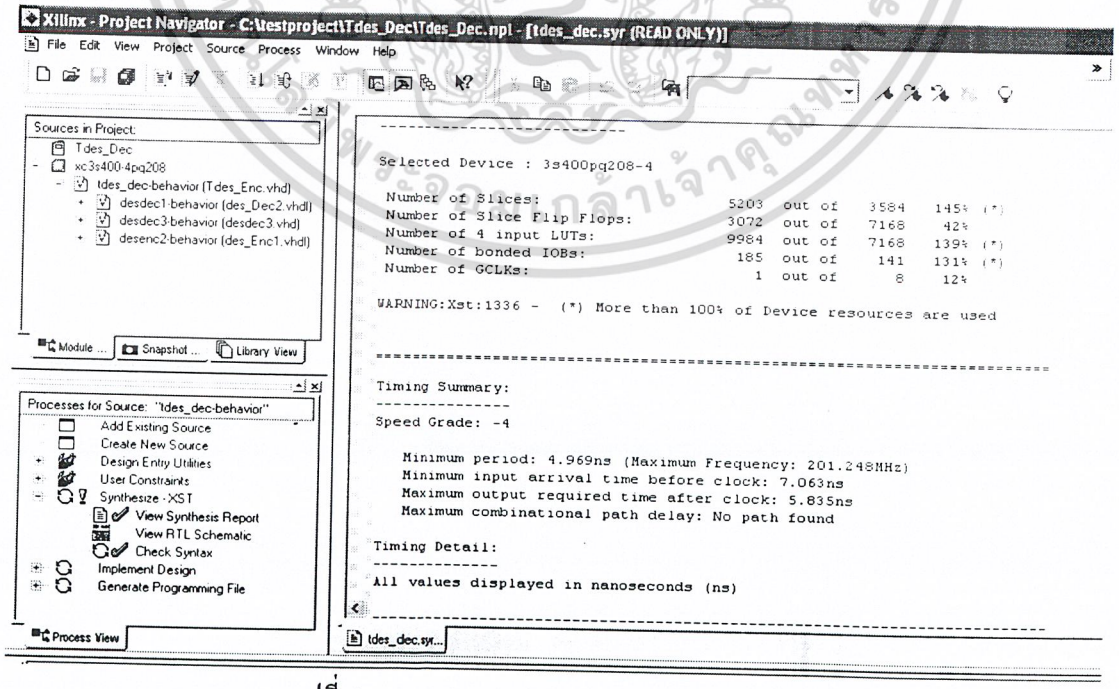
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้ มีปัญหาในส่วนของ Number of slices , Number of 4 input LUT และ IOBs ไม่เพียงพอต่อการออกแบบ แต่ในส่วนของ Number of Slices Flip-Flop และ GCLKs ไม่มีปัญหาและเพียงพอ โดยสรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.24



รูปที่ 6.24 ผลการ Synthesis TDES Encryption

❑ TDES Decryption

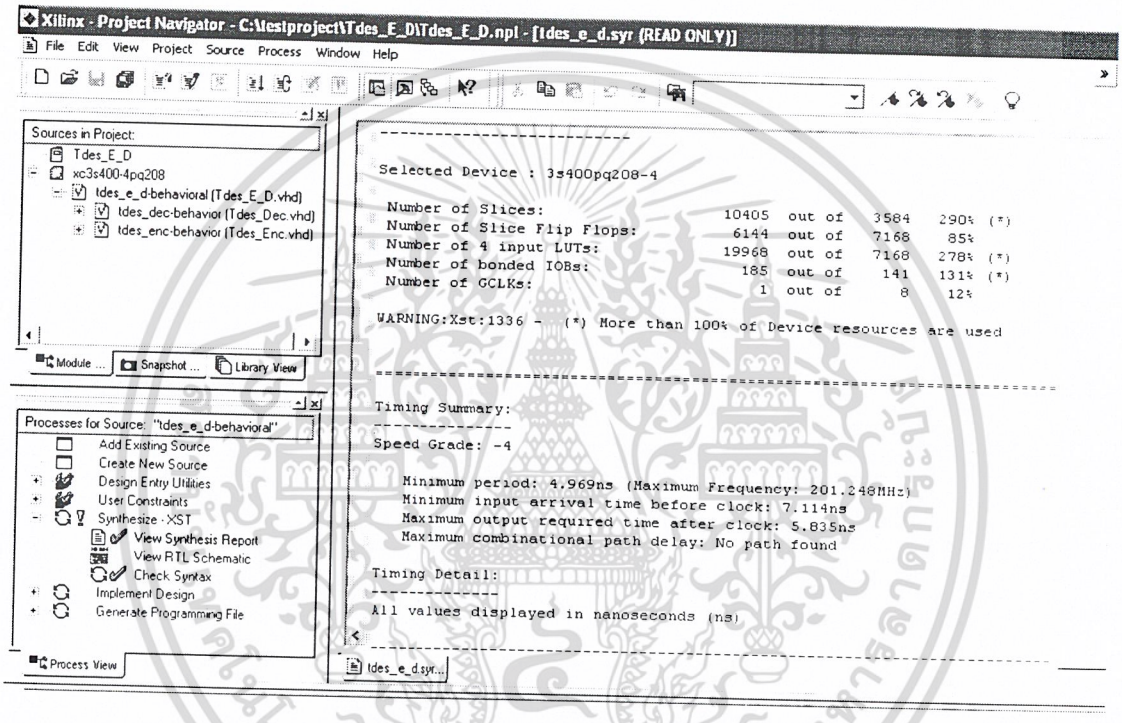
ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้ มีปัญหาในส่วนของ Number of slices , Number of 4 input LUT และ IOBs ไม่เพียงพอต่อการออกแบบ แต่ในส่วนของ Number of Slices Flip-Flop และ GCLKs ไม่มีปัญหาและเพียงพอ โดยสรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.25



รูปที่ 6.25 ผลการ Synthesis TDES Decryption

❑ **TDES Encryption & Decryption**

ผลการ Synthesis ไม่ผ่านกระบวนการ เพราะเครื่องหมาย ! แสดงที่หน้า Synthesis-XST ในการทดลองนี้แสดงให้เห็นปัญหาของการ Sythesis ในหลายส่วนด้วยกัน เนื่องจากคุณสมบัติของ FPGA ไม่เพียงพอต่อการออกแบบ สาเหตุเกิดขึ้นจาก 4 ส่วน คือ Number of slices , Number of Slices Flip-Flop , Number of 4 input LUT และ IOBs เกินจำนวนที่ FPGA มี การออกแบบต้องการ ส่วน GCLKs ไม่มีปัญหาใดๆ สรุปแล้ว การ Synthesis ไม่ผ่านกระบวนการ รายละเอียดตามรูปที่ 6.26



รูปที่ 6.26 ผลการ Synthesis TDES Encryption & Decryption

6.2 การเปรียบเทียบการทดลองกับผู้ค้าคว้าวิจัยในต่างประเทศ

จากการทดลองจะเห็นว่า กระบวนการ Synthesis ไม่สามารถทำได้ เครื่องมือที่ใช้ในการโปรแกรมและ Synthesis ได้จำลองรุ่นของ FPGA หลายรุ่น แต่รุ่นที่เลือกใช้ในการทดลองเป็นรุ่นที่มีใช้ในท้องตลาดปัจจุบัน แต่ก็ยังไม่สามารถที่จะ Implement ลงบน FPGA ได้ จึงได้ทำการศึกษาผลงานวิจัยของต่างประเทศเพื่อนำมาเปรียบเทียบ และหาข้อสรุปว่าปัญหาเกิดขึ้นอย่างไร ข้อมูลที่จะนำเสนอต่อไปนี้เป็นการเปรียบเทียบผลงานค้นคว้าวิจัยของบริษัท XILINX กับ การทดลองที่ได้ทดลองไปแล้วในช่วงแรกของบทที่ 6

ข้อมูลที่ได้ค้นคว้าของบริษัท XILINX เป็นการนำเสนอผลงานในการ Implement High Speed DES and Triple DES Encryptor / Decryptor ผู้ที่ทำการวิจัยคือ Vikram Pasham และ Steve Trimberger ผลงานของผู้วิจัยต่างประเทศได้แสดงถึงการสร้างวงจร Encryption เพียงอย่างเดียว พร้อม

ทั้งสรุปผลการวิจัยออกมาให้ทราบ แต่การสร้างวงจร Decryption ผู้วิจัยเพียงแต่กล่าวถึงทฤษฎีเท่านั้น ไม่ได้มีผลการทดลอง และสรุปผลงานวิจัยออกมาให้ทราบ

ดังนั้นในการเปรียบเทียบการทำโครงงานกับการวิจัยของต่างประเทศ จะสรุปให้เห็นถึงความแตกต่างของการสร้างวงจร Encryption เท่านั้น ซึ่งพอจะทำให้ผู้ศึกษาสามารถทราบ และเปรียบเทียบข้อมูลในหลายๆ ด้านได้เป็นอย่างดี การเปรียบเทียบแสดงตามตารางที่ 6.1

6.2.1 DES Implement Result (DES Encryption)

Device	Number of LUTs (% of LUTs)	Number of Slices Flip-Flops (% of Flip-Flops)	Performance (maximum clock rate)	Comment
VirtexII XC2V1000-5 FG456	5036 (49%)	5185 (50%)	237 MHz	ผู้วิจัยต่างประเทศ บริษัท XILINX
SpartanIII XC3S400 pq208	3448 (48%)	1024 (14%)	200.361 MHz	การทดลองตาม รูปที่ 6.21

ตารางที่ 6.1 เปรียบเทียบการใช้ทรัพยากรของ FPGA ในการ Implement DES Encryption ของการค้นคว้าวิจัยของต่างประเทศ กับ การทดลอง

ตารางที่ 6.1 แสดงให้เห็นถึงความแตกต่าง และจำนวนของการใช้ทรัพยากรภายใน การใช้ทรัพยากรภายใน FPGA ระหว่างผู้วิจัยต่างประเทศกับการทดลอง มีความแตกต่างกันอย่างมาก ตัวอย่าง เช่น Number of LUTs (% of LUTs) ผู้วิจัยต่างประเทศใช้จำนวนทรัพยากรในการ Implement DES Encryptor ถึง 5036 แต่ จากการทดลองใช้เพียง 3448 เท่านั้น และอีกหนึ่งตัวอย่างคือ Number of Slices Flip-Flops (% of Flip-Flops) ผู้วิจัยต่างประเทศใช้จำนวนทรัพยากรในการ Implement ถึง 5185 ในขณะที่ การทดลองใช้เพียง 1024 เท่านั้น แต่การทำวิจัยของต่างประเทศ สามารถ Implement ลง บน FPGA ได้ เนื่องจากเหตุผลสำคัญคือ โปรแกรมที่ผู้วิจัยต่างประเทศใช้ในการ Synthesis มีการจำลองรุ่นของ FPGA ที่มีคุณสมบัติสูงกว่าที่ใช้ในการทดลอง และรุ่นของ FPGA ที่ใช้ในการ Implement มีทรัพยากรเพียงพอ จากตารางจะสรุปคุณสมบัติของ FPGA ที่ผู้วิจัยต่างประเทศใช้ให้ทราบดังนี้

การเปรียบเทียบคุณสมบัติของ FPGA
ในการ Implement DES Encryption

รุ่น	จำนวนเกท	จำนวนขา
VirtexII รุ่น XC2V1000-5 FG456	1,000,000	456
SpartanIII รุ่น XC3S400 pq208	400,000	208

ตารางที่ 6.2 เปรียบเทียบการคุณสมบัติของ FPGA ในการ Implement DES Encryption ที่ผู้วิจัยต่างประเทศใช้ กับการทดลอง

6.2.2 TDES Implement Result (TDES Encryption)

Device	Number of LUTs (% of LUTs)	Number of Slices Flip-Flops (% of Flip-Flops)	Performance (maximum clock rate)	Comment
VirtexII XC2V3000-5 FG676	16,181 (56%)	15,759 (50%)	207 MHz	ผู้วิจัยต่างประเทศ บริษัท XILINX
SpartanIII XC3S400 pq208	9984 (139%)	3027 (42%)	201.248 MHz	การทดลองตาม รูปที่ 6.24

ตารางที่ 6.3 เปรียบเทียบการใช้ทรัพยากรของ FPGAในการ Implement TDES Encryption ของการค้นคว้าวิจัยของต่างประเทศ กับ การทดลอง

ตารางที่ 6.3 แสดงให้เห็นถึงความแตกต่าง และจำนวนของการใช้ทรัพยากรภายใน การใช้ทรัพยากรภายใน FPGA ระหว่างผู้วิจัยต่างประเทศกับการทดลอง มีความแตกต่างกันอย่างมาก ตัวอย่าง เช่น Number of LUTs (% of LUTs) ผู้วิจัยต่างประเทศใช้จำนวนทรัพยากรในการ Implement TDES Encryptor ถึง 16,181 แต่ จากการทดลองใช้เพียง 9984 เท่านั้น แต่ถ้าหากคิดเป็นเปอร์เซ็นแล้ว การทดลองใช้มากถึง 139% เนื่องจากเป็นการเปรียบเทียบเปอร์เซ็นภายในตัว FPGA รุ่นที่ทำการทดลอง (ซึ่งจะแสดงให้เห็นคุณบัติของ FPGA ตามตารางที่ 6.4) และอีกหนึ่งตัวอย่างคือ Number of Slices Flip-Flops (% of Flip-Flops) ผู้วิจัยต่างประเทศใช้จำนวนทรัพยากรในการ Implement ถึง 15,759 ในขณะที่ การทดลองใช้เพียง 3027 เท่านั้น แต่การทำวิจัยของต่างประเทศ สามารถ Implement ลงบน

FPGA ได้ เนื่องจากเหตุผลสำคัญคือ โปรแกรมที่ผู้วิจัยต่างประเทศใช้ในการ Synthesis มีการจำลองรุ่นของ FPGA ที่มีคุณสมบัติสูงกว่าที่ใช้ในการทดลอง และรุ่นของ FPGA ที่ใช้ในการ Implement มีทรัพยากรเพียงพอ จากตารางจะสรุปคุณสมบัติของ FPGA ที่ผู้วิจัยต่างประเทศใช้ให้ทราบดังนี้

การเปรียบเทียบคุณสมบัติของ FPGA
ในการ Implement TDES Encryption

รุ่น	จำนวนเกท	จำนวนขา
VirtexII รุ่น XC2V3000-5 FG676	3,000,000	676
SpartanIII รุ่น XC3S400 pq208	400,000	208

ตารางที่ 6.4 เปรียบเทียบการคุณสมบัติของ FPGA ในการ Implement TDES Encryption ที่ผู้วิจัยต่างประเทศใช้ กับการทดลอง

6.3 สรุปผลการทดลอง

จากการทดลอง ได้ทำการทดลองในสองส่วนหลักด้วยกันคือ การจำลองการทำงานของโปรแกรมโดยใช้โปรแกรม ModelSim SE Plus 6.0 และทำการทดลอง Implement ลงบนอุปกรณ์ FPGA ซึ่งสามารถสรุปผลได้ดังนี้

6.3.1 การจำลองกการทำงานด้วย ModelSim SE Plus 6.0

ขั้นที่ 1 ได้จำลองการทำงานของ DES Encryption , DES Decryption และ DES Encryption & DES Decryption เพื่อตรวจสอบดูว่าให้ผลลัพธ์ออกมาถูกต้องตามทฤษฎีหรือไม่ ผลปรากฏว่า การทำงานของ DES Encryption & DES Decryption เมื่อทำงานจนครบกระบวนการแล้วให้ผลลัพธ์ออกมาถูกต้อง นั่นคือ เมื่อนำ Plantext เข้าสู่ DES Encryption จะได้ผลลัพธ์ออกมาค่าหนึ่งซึ่งค่านี้ถูกเรียกว่า Ciphertext แล้วนำ Ciphertext ตัวเดียวกันนี้ป้อนกลับเข้าไปใน DES Decryption จะได้ผลลัพธ์สุดท้ายออกมาเป็น Plantext ตัวเดิม โดยในส่วนของ key นั้น ได้กำหนดไว้ตั้งแต่เริ่มต้นไว้แล้วซึ่ง key ในการ Encryption และ Decryption เป็นตัวเดียวกัน

ตัวอย่างจากการทดลอง (ตามรูปที่ 6.3)

DES Encryption & DES Decryption

Plantext _in : 0123456789abcdef

Key : 133457799bbcdff1

Plantext _out : 0123456789abcdef

ขั้นที่ 2 ได้จำลองการทำงานของ TDES Encryption , TDES Decryption และ TDES Encryption & TDES Decryption เพื่อตรวสอบดูว่าให้ผลลัพธ์ออกมาถูกต้องตามทฤษฎีหรือไม่ ผลปรากฏว่า การทำงานของ TDES Encryption & TDES Decryption เมื่อทำงานจนครบกระบวนการแล้ว ให้ผลลัพธ์ออกมาถูกต้อง นั่นคือ เมื่อนำ Plantext เข้าสู่ TDES Encryption จะได้ผลลัพธ์ออกมาค่าหนึ่งซึ่งค่านี้ถูกเรียกว่า Ciphertext แล้วนำ Ciphertext ตัวเดียวกันนี้ป้อนกลับเข้าไปใน TDES Decryption จะได้ผลลัพธ์สุดท้ายออกมาเป็น Plantext ตัวเดิม โดยในส่วนของ key นั้น ได้กำหนดไว้ตั้งแต่เริ่มต้นไว้แล้วซึ่ง key ในการ Encryption และ Decryption เป็นตัวเดียวกัน

ตัวอย่างจากการทดลอง (ตามรูปที่ 6.6)

DES Encryption & DES Decryption

Plantext _in : 0123456789abcdef

Key : 133457799bbcdff1

Plantext _out : 0123456789abcdef

สรุปผลการจำลองการทำงาน

ในขั้นตอนของการจำลองการทำงานเข้ารหัสและถอดรหัสข้อมูลโดยใช้อัลกอริทึมแบบ TDES สิ่งสำคัญในขั้นตอนนี้คือโปรแกรมภาษา VHDL ถ้าหากทำการโปรแกรมไม่ถูกต้องจะให้ผลลัพธ์ออกมาไม่ถูกต้อง แต่การทดลองครั้งนี้ สามารถโปรแกรมภาษา VHDL ได้อย่างถูกต้อง เนื่องจากผลการทดสอบแสดงให้เห็นแล้วว่าผลลัพธ์ของโปรแกรมสามารถทำได้อย่างถูกต้องตามทฤษฎี สิ่งที่แสดงให้เห็นคือ การทดสอบโดยการนำ Plantext ที่ป้อนให้กับโปรแกรมเพื่อทำการเข้ารหัส จะได้ Ciphertext ออกมาค่าหนึ่ง และเมื่อนำ Ciphertext ตัวเดียวกันนี้ป้อนกลับเข้าไปในโปรแกรมการถอดรหัส จะทำให้ได้ Plantext ตัวเดิมออกมา โดยทำการทดสอบตัวอย่างหลายๆตัวอย่าง โปรแกรมการจำลองการทำงานก็สามารถทำงานได้อย่างถูกต้องเสมอ ซึ่งเป็นสิ่งที่สามารถแสดงถึงความสำเร็จได้

6.2.2 การ Implement ลงบนอุปกรณ์ FPGA

ในการทดลองขั้นนี้ การที่จะทำนำวงจรที่ออกแบบลงบนอุปกรณ์ FPGA จะต้องทำทั้งหมด 2 กระบวนการ คือการ Synthensis และ การ Implement ตามลำดับ แต่การทดลองครั้งนี้ไม่สามารถทำการ Implement ลงบนฮาร์ดแวร์ได้ เนื่องจากการ Synthensis ไม่ผ่านกระบวนการ

การทดลองพยายามหาข้อสรุปสาเหตุที่ทำการ Synthensis ไม่ผ่านกระบวนการ โดยตั้งสมมุติฐานไว้ คือ ข้อจำกัดของ FPGA ข้อมูลที่จะแสดงต่อไปนี้ จะแสดงผลการทดลองจากสมมุติฐานที่ตั้งขึ้น

- **ข้อจำกัดของ FPGA**

ข้อมูลนี้จะแสดงการเปรียบเทียบผลลัพธ์ที่ได้จากการ Synthensis โดยใช้ FPGA จำนวน 2 รุ่นที่แตกต่างกัน คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 ได้ผลลัพธ์ดังนี้

I) DES Algorithm

การ Synthensis DES Encryption

Resource	Spartan III รุ่น XC3S200 Tp144	Spartan III รุ่น XC3S400 pq208
Number of Slices	1735 <1920> .90%	1807 <3584> 50%
Number of Slices Flip Flop	1024 <3840> 26%	1024 <7168> 14%
Number of 4 Input LUTs	3328 <3840> 86%	3448 <7168> 48%
Number of bonded IOBs	1735 <97> 190%	185 <141> 131%
Number of GCLKs	1 <8> 12%	1 <8> 12%
Maximum Frequency	201.248 MHz	200.361 MHz

ตารางที่ 6.5 เปรียบเทียบการใช้ทรัพยากรของ FPGA สองรุ่น ในการ Synthensis DES Encryption

จากตารางที่ 6.5 เป็นการเปรียบเทียบการใช้ทรัพยากรในการ Synthesis วงจรที่ออกแบบ โดยเปรียบเทียบระหว่าง FPGA 2 รุ่น คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 นอกจากข้อมูลในตารางนอกจากจะแสดงการใช้ทรัพยากรแล้ว ยังแสดงทรัพยากรที่มีอยู่ภายใน FPGA แต่ละรุ่น ยกตัวอย่างเช่น FPGA SpartanIII รุ่น XC3S200 tq144 มีการใช้ Number of Slices 1735 จากที่มีอยู่ 1920 คิดเป็น 90%

การ Synthesis DES Encryption

Resource	Spartan III รุ่น XC3S200 Tp144	Spartan III รุ่น XC3S400 pq208
Number of Slices	1735 <1920> 90%	1807 <3584> 50%
Number of Slices Flip Flop	1024 <3840> 25%	1024 <7168> 14%
Number of 4 Input LUTs	3328 <3840> 86%	3448 <7168> 48%
Number of bonded IOBs	185 <97> 190%	185 <141> 131%
Number of GCLKs	1 <8> 12%	1 <8> 12%
Maximum Frequency	201.248 MHz	200.361 MHz

ตารางที่ 6.6 เปรียบเทียบการใช้ทรัพยากรของ FPGA สองรุ่น ในการ Synthensis DES Decryption

จากตารางที่ 6.6 เป็นการเปรียบเทียบการใช้ทรัพยากรในการ Synthesis วงจรที่ออกแบบ โดยเปรียบเทียบระหว่าง FPGA 2 รุ่น คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 ข้อมูลในตารางนอกจากจะแสดงการใช้ทรัพยากรแล้ว ยังแสดงทรัพยากรที่มีอยู่ภายใน FPGA แต่ละรุ่น ยกตัวอย่างเช่น FPGA SpartanIII รุ่น XC3S400 pq208 มีการใช้ Number of Slices 1807 จากที่มีอยู่ 3584 คิดเป็น 50%

การ Synthensis DES Encryption & DES Encryption

Resource	Spartan III รุ่น XC3S200 tp144	Spartan III รุ่น XC3S400 pq208
Number of Slices	10405 <1920> 541%	3469 <3584> 96%
Number of Slices Flip Flop	6144 <3840> 160%	2048 <7168> 28%
Number of 4 Input LUTs	19968 <3840> 520%	6656 <7168> 92%
Number of bonded IOBs	185 <97> 190%	185 <141> 131%
Number of GCLKs	1 <8> 12%	1 <8> 12%
Maximum Frequency	201.248 MHz	201.248 MHz

ตารางที่ 6.7 เปรียบเทียบการใช้ทรัพยากรของ FPGA สองรุ่น ในการ Synthensis DES Encryption & DES Decryption

จากตารางที่ 6.7 เป็นการเปรียบเทียบการใช้ทรัพยากรในการ Synthesis วงจรที่ออกแบบ โดยเปรียบเทียบระหว่าง FPGA 2 รุ่น คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 นอกจากข้อมูลในตารางนอกจากจะแสดงการใช้ทรัพยากรแล้ว ยังแสดงทรัพยากรที่มีอยู่ภายใน FPGA แต่ละรุ่น ยกตัวอย่างเช่น FPGA SpartanIII รุ่น XC3S200 tq144 มีการใช้ Number of Slices 10,405 จากที่มีอยู่ 1,920 คิดเป็น 541% มีการใช้ Number of Slices Flip Flop 6,144 จากที่มีอยู่ 3,840 คิดเป็น 160% ในขณะที่ SpartanIII รุ่น XC3S400 pq208 มีการใช้ Number of Slices 3,469 จากที่มีอยู่ 3,584 คิดเป็น 96% มีการใช้ Number of Slices Flip Flop 2,048 จากที่มีอยู่ 7,168 คิดเป็น 28% รายละเอียดส่วนอื่นก็สามารถเปรียบเทียบให้เห็นตามตาราง โดยรวมแล้ว ความต้องการใช้ทรัพยากรจะมีค่าเท่ากันหรือใกล้เคียงกัน แต่แตกต่างกันตรงที่จำนวนทรัพยากรของ FPGA แต่ละรุ่นมีไม่เท่ากัน

II) TDES Algorithm

การ Synthensis TDES Encryption

Resource	Spartan III รุ่น XC3S200 Tp144	Spartan III รุ่น XC3S400 pq208
Number of Slices	5203 <1920> 270%	5203 <3584> 145%
Number of Slices Flip Flop	3072 <3840> 80%	3072 <7168> 42%
Number of 4 Input LUTs	9984 <3840> 260%	9984 <7168> 139%
Number of bonded IOBs	185 <97> 190%	185 <141> 131%
Number of GCLKs	1 <8> 12%	1 <8> 12%
Maximum Frequency	201.248 MHz	201.248 MHz

ตารางที่ 6.8 เปรียบเทียบการใช้ทรัพยากรของ FPGA สองรุ่น ในการ Synthensis TDES Encryption

จากตารางที่ 6.8 เป็นการเปรียบเทียบการใช้ทรัพยากรในการ Synthesis วงจรที่ออกแบบ โดยเปรียบเทียบระหว่าง FPGA 2 รุ่น คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 นอกจากข้อมูลในตารางนอกจากจะแสดงการใช้ทรัพยากรแล้ว ยังแสดงทรัพยากรที่มีอยู่ภายใน FPGA แต่ละรุ่น ยกตัวอย่างเช่น FPGA SpartanIII รุ่น XC3S200 tq144 มีการใช้ Number of Slices 5203 จากที่มีอยู่ 1920 คิดเป็น 270%

การ Synthensis TDES Decryption

Resource	Spartan III รุ่น XC3S200 Tp144	Spartan III รุ่น XC3S400 pq208
Number of Slices	15203 <1920> 270%	5203 <3584> 145%
Number of Slices Flip Flop	3027 <3840> 80%	3072 <7168> 42%
Number of 4 Input LUTs	9984 <3840> 260%	9984 <7168> 139%
Number of bonded IOBs	185 <97> 190%	185 <141> 131%
Number of GCLKs	1 <8> 12%	1 <8> 12%
Maximum Frequency	201.248 MHz	201.248 MHz

ตารางที่ 6.9 เปรียบเทียบการใช้ทรัพยากรของ FPGA สองรุ่น ในการ Synthensis TDES Decryption

จากตารางที่ 6.9 เป็นการเปรียบเทียบการใช้ทรัพยากรในการ Synthesis วงจรที่ออกแบบ โดยเปรียบเทียบระหว่าง FPGA 2 รุ่น คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 ข้อมูลในตารางนอกจากจะแสดงการใช้ทรัพยากรแล้ว ยังแสดงทรัพยากรที่มีอยู่ภายใน FPGA แต่ละรุ่น ยกตัวอย่างเช่น FPGA SpartanIII รุ่น XC3S400 pq208 มีการใช้ Number of Slices 3027 จากที่มีอยู่ 3840 คิดเป็น 80%

การ Synthensis TDES Encryption & TDES Encryption

Resource	Spartan III รุ่น XC3S200 tp144	Spartan III รุ่น XC3S400 pq208
Number of Slices	10405 <1920> 541%	10405 <3584> 290%
Number of Slices Flip Flop	6144 <3840> 160%	6144 <7168> 85%
Number of 4 Input LUTs	19968 <3840> 520%	19968 <7168> 278%
Number of bonded IOBs	185 <97> 190%	185 <141> 131%
Number of GCLKs	1 <8> 12%	1 <8> 12%
Maximum Frequency	201.248 MHz	201.248 MHz

ตารางที่ 6.10 เปรียบเทียบการใช้ทรัพยากรของ FPGA สองรุ่น ในการ Synthensis TDES Encryption & TDES Decryption

จากตารางที่ 6.10 เป็นการเปรียบเทียบการใช้ทรัพยากรในการ Synthesis วงจรที่ออกแบบ โดยเปรียบเทียบระหว่าง FPGA 2 รุ่น คือ SpartanIII รุ่น XC3S200 tq144 และ SpartanIII รุ่น XC3S400 pq208 นอกจากข้อมูลในตารางนอกจากจะแสดงการใช้ทรัพยากรแล้ว ยังแสดงทรัพยากรที่มีอยู่ภายใน FPGA แต่ละรุ่น ยกตัวอย่างเช่น FPGA SpartanIII รุ่น XC3S200 tq144 มีการใช้ Number of Slices 10,405 จากที่มีอยู่ 1,920 คิดเป็น 541% มีการใช้ Number of Slices Flip Flop 6,144 จากที่มีอยู่ 3,840 คิดเป็น 160% ในขณะที่ SpartanIII รุ่น XC3S400 pq208 มีการใช้ Number of Slices 10,405 จากที่มีอยู่ 3,584 คิดเป็น 290% มีการใช้ Number of Slices Flip Flop 6,144 จากที่มีอยู่ 7,168 คิดเป็น 85% รายละเอียดส่วนอื่นก็สามารถเปรียบเทียบให้เห็นตามตาราง โดยรวมแล้ว ความต้องการใช้ทรัพยากรจะมีค่าเท่ากันหรือใกล้เคียงกัน แต่แตกต่างกันตรงที่จำนวนทรัพยากรของ FPGA แต่ละรุ่นมีไม่เท่ากัน

สรุปสมมุติฐาน : ข้อจำกัดของ FPGA

จากการสรุปในหัวข้อข้างต้นทำให้ทราบว่า รุ่นของ FPGA มีผลกระทบต่อการ Implement เนื่องจากว่า FPGA รุ่นที่เราทดลองจะมีขอบเขตข้อจำกัดระดับหนึ่ง แต่เมื่อทำการเปลี่ยนรุ่นของ FPGA ให้มีคุณสมบัติที่ดีขึ้น ผลการทดลองจะแสดงให้เห็นว่าเปอร์เซ็นต์การใช้ทรัพยากรภายใน FPGA เพียงพอในบางส่วน จึงสามารถสรุปได้ว่า การเลือใช้ FPGA ให้เหมาะสมกับการออกแบบ เป็นสิ่งสำคัญอย่างยิ่ง ดังนั้นการที่จะตัดสินใจเลือก FPGA ของบริษัทใด รุ่นใด ควรศึกษาข้อจำกัดของอุปกรณ์ และความต้องการของผู้ออกแบบ

6.4 สรุปผลการทำโครงงาน

โครงงานได้ทำการทดลองในสองส่วน คือ การจำลองการทำงานด้วย ModelSim SE Plus 6.0 และการ Implement ลงบนอุปกรณ์ FPGA ซึ่งผลการทดลองที่ได้ เราสามารถจำลองการทำงานของโปรแกรมได้อย่างถูกต้อง แต่การที่จะ Implement ลงบนอุปกรณ์ FPGA ไม่สามารถทำได้เนื่องจากติดปัญหาหลายๆ ด้าน จึงสามารถสรุปผลการทดลองได้ดังนี้

1. การจำลองการทำงาน สิ่งสำคัญคือการโปรแกรมภาษา VHDL สามารถโปรแกรมภาษา VHDL ได้อย่างถูกต้อง รวมทั้งการออกแบบก็สามารถทำได้อย่างถูกต้องเช่นกัน เนื่องจากผลการทดสอบแสดงให้เห็นแล้วว่าผลลัพธ์ของโปรแกรมสามารถทำได้อย่างถูกต้องตามทฤษฎี

2. การเลือกใช้อุปกรณ์ FPGA มีผลกระทบต่อการ Implement เนื่องจากว่า FPGA จะมีขอบเขตข้อจำกัดของตัวเอง โดยในการทำโครงงานครั้งนี้ FPGA ที่เลือก เป็น SpartanIII รุ่น XC3S200 tq144 เป็น FPGA ที่มีเกทมากถึง 200,000 เกท และจำนวนขา 144 ขา ซึ่งเป็นรุ่นที่ออกมาล่าสุด และเป็นรุ่นที่ดีที่สุดในปัจจุบันด้วย แต่ก็ยังไม่สามารถ Implement ลงบน FPGA ได้

3. การโปรแกรมและการออกแบบวงจรการเข้ารหัสและถอดรหัสข้อมูลของอัลกอริทึมแบบ TDES มีผลกระทบต่อการ Implement เนื่องจากว่า โปรแกรมสามารถทำได้ถูกต้องแต่ยังไม่ Optimize ดีพอ และการออกแบบวงจร จำนวนอินพุทหรือเอ้าท์พุท มีผลกระทบต่อทรัพยากรภายใน FPGA สาเหตุทั้งสองประการนี้จึงทำให้ไม่สามารถ Implement ลงบนฮาร์ดแวร์ได้

6.5 อุปกรณ์ที่เหมาะสมในการ Implement

คุณสมบัติของ FPGA ที่ต้องการในการออกแบบและ Implement TDES Algorith เป็นรุ่นเดียวกับที่ผู้วิจัยต่างประเทศใช้คือ FPGA VirtexII รุ่น XC2V3000-5 FG676 ซึ่งมีเกทมากถึง 3,000,000 เกท และมีขามากถึง 676 ขา ซึ่งคาดว่าน่าจะรองรับการ Implement โครงงานครั้งนี้ได้ ทั้งนี้ยังไม่ได้มีการทดลอง Synthesis เพราะเครื่องมือในการโปรแกรมไม่ได้มีการจำลองรุ่นดังกล่าวเอาไว้ จึงไม่สามารถนำผลมาแสดงให้ทราบๆ ได้

บทที่ 7
บทสรุปและวิจารณ์

7.1 บทสรุป

โครงงานอุปกรณ์การเข้ารหัสและถอดรัสข้อมูลโดยใช้ FPGA ได้เริ่มดำเนินการมาตั้งแต่ 15 มิ.ย. 47 ถึง 15 มี.ค. 48 รวมเวลาการทำโครงงานประมาณ 10 เดือน ประสบผลสำเร็จเป็นที่น่าพอใจ ถึงแม้ว่าไม่สมบูรณ์ 100% ก็ตาม แต่ถือได้ว่าประสบผลสำเร็จอย่างมากในการทำโครงงานครั้งนี้ เพราะจากการทดลองแสดงให้เห็นว่า สามารถจำลองการทำงานและให้ผลลัพธ์ออกมาถูกต้องตามทฤษฎี แต่ในส่วนของขั้นตอนการ Implement ลงบนฮาร์ดแวร์ ไม่สามารถทำได้เนื่องจากข้อจำกัดของอุปกรณ์ FPGA จึงเป็นปัญหาหนึ่งที่พบในระหว่างการทำโครงงาน นอกจากนี้ยังพบปัญหาอื่นๆอีกหลายด้านเช่นกัน ในหัวข้อนี้จะนำเสนอบทสรุปของการทำโครงงานตั้งแต่เริ่มต้นจนถึงสิ้นสุดการทำโครงงาน ดังนี้

ภาคเรียนที่1

เป็นการเริ่มต้นการทำโครงงาน จะต้องทำการศึกษาข้อมูลทฤษฎีที่เกี่ยวข้อง ทั้งในเรื่องการเข้ารหัสและถอดรหัสข้อมูล ภาษา VHDL และอุปกรณ์ FPGA โดยดำเนินการตามแผนการทำโครงงานคือ

แผนการทำโครงงาน : ศึกษาทฤษฎีที่เกี่ยวข้อง

1. ศึกษาการทฤษฎีการเข้ารหัสและถอดรัสข้อมูลวิธีต่างๆ ที่มีใช้ในปัจจุบัน และเลือกอัลกอริทึมที่เหมาะสมในการทำโครงงาน

2. ศึกษาโครงสร้างและคุณสมบัติของอุปกรณ์ประเภท FPGA ในการสร้างอุปกรณ์เข้ารหัสและถอดรหัสข้อมูล รวมทั้งข้อจำกัดของอุปกรณ์ที่มีผลกระทบต่อการออกแบบ

3 ศึกษาภาษารูปแบบการเขียนโปรแกรมด้วยภาษาวีเอชดีแอล ในการออกแบบระบบฮาร์ดแวร์ดิจิตอล ตั้งแต่การออกแบบ แก้ไขตรวจสอบ จำลองการทำงาน และสังเคราะห์วงจรบนอุปกรณ์ประเภท FPGA

ผลการศึกษาตามแผนพอสรุปได้ดังนี้

1. ศึกษาการทฤษฎีการเข้ารหัสและถอดรัสข้อมูล ผู้ทำโครงงานได้ทำการศึกษาอัลกอริทึมหลายรูปแบบที่เหมาะสมในการ Implement ลงบนฮาร์ดแวร์ เมื่อเลือกได้แล้วต้องทำความเข้าใจอัลกอริทึมนั้นให้ดีเพื่อที่จะสามารถออกแบบได้ถูกต้อง ผลสรุปของการศึกษา ผู้ทดลองได้เลือกอัลกอริทึมแบบ Triple DES

2. ศึกษาโครงสร้างและคุณสมบัติของอุปกรณ์ประเภท FPGA เป็นการศึกษาทฤษฎี ยังไม่ได้ทดลองอุปกรณ์อย่างจริงจัง จึงมีความเข้าใจไม่มากนัก

3. ศึกษาภาษารูปแบบการเขียนโปรแกรมด้วยภาษาวีเอชดีแอล เป็นการศึกษาทฤษฎีเช่นกัน เนื่องจากเป็นภาษาใหม่ที่ผู้ทำโครงงานไม่คุ้นเคย จึงยังไม่มีการทดลองโปรแกรมแต่อย่างใด

ดังนั้น ในภาคเรียนที่ 1 สิ่งที่ได้มากที่สุดคือการความเข้าใจในอัลกอริทึมของการเข้ารหัสและถอดรหัสข้อมูล ซึ่งต้องใช้เวลามากพอสมควรในการศึกษา

ภาคเรียนที่ 2

เป็นการดำเนินงานในช่วงที่ 2 ต่อจากภาคเรียนที่ 1 การทำงานส่วนใหญ่เป็นการออกแบบการเข้ารหัสและถอดรหัสข้อมูล การโปรแกรม การทดลอง และการจัดทำเอกสาร โดยดำเนินการตามแผนเช่นเดียวกับภาคเรียนที่ 1 ดังนี้

แผนการทำโครงงาน : ออกแบบและทดลอง

1. ออกแบบวงจรการเข้ารหัสและถอดรหัสข้อมูลโดยใช้อัลกอริทึมที่ศึกษา
2. เขียนโปรแกรมด้วยภาษาวีเอชดีแอล และสร้างวงจรลงในอุปกรณ์ FPGA
3. ทดลองและทดสอบด้วยการจำลองการทำงาน
4. ทดลอง Implement ลงบนอุปกรณ์
5. สรุปผลการทดลอง และจัดทำเอกสาร

ผลการศึกษาตามแผนพอสรุปได้ดังนี้

1. การออกแบบการเข้ารหัสและถอดรหัสข้อมูล เป็นการออกแบบลักษณะ Top - down Design พบปัญหาคือการออกแบบที่ไม่เหมาะสมต้องทำการออกแบบ แก้ไข ทดลอง ส่วนประกอบย่อยให้มีการทำงานที่ถูกต้องเสียก่อน จึงจะออกแบบในส่วนต่อไปได้ ในส่วนนี้ปัญหาที่เกิดจากการออกแบบมีไม่มากนัก แต่กลับเกิดจากการติดตั้งโปรแกรมที่ใช้ในการทดลอง กระทำยากมาก เมื่อสามารถติดตั้งได้แล้วยังต้องฝึกการใช้เครื่องมือต่างๆ ในโปรแกรมการทดลอง ซึ่งต้องใช้เวลานานพอสมควร

2. เขียนโปรแกรมด้วยภาษาวีเอชดีแอล ปัญหาในส่วนนี้มีไม่มากนักเนื่องจากได้ศึกษาทฤษฎีมาบ้างแล้ว จึงทำการโปรแกรมและทดลองแก้ไขไปพร้อมๆ กันได้

3. ขั้นการทดลอง มีการจำลองการทำงาน และทดลอง Implement ลงบนอุปกรณ์ ขั้นตอนนี้ถือเป็นขั้นตอนสำคัญที่สุดขั้นตอนหนึ่ง ผู้จัดทำได้แสดงผลการทดลองโดยการแสดงเป็นรูปภาพ เพื่อให้เห็นว่าได้ทำการทดลองจริง และได้ผลลัพธ์ออกมาเช่นไร เนื้อหาในส่วนนี้มีจำนวน 1 บทในเอกสาร ถือว่ามากพอสมควร

4. การทำเอกสาร ในส่วนนี้ผู้ทำโครงงานได้จัดหาข้อมูลที่ดีและสมบูรณ์จากหลายแหล่งเช่น ทางอินเตอร์เน็ท หนังสือต่างประเทศ หนังสือภายในประเทศ เพื่อหาทฤษฎีต่างๆ มารวบรวมไว้ในเอกสารเล่มนี้ ในเอกสารยังได้แสดงผลการทดลองที่มากพอสมควร ที่จะแสดงให้เห็นผลการทำโครงงาน และเป็นแนวทางสำหรับผู้สนใจได้ศึกษาต่อไป ซึ่งได้ทำการแก้ไขหลายครั้งกว่าจะได้เอกสารที่สมบูรณ์

ดังนั้น ในภาคเรียนที่ 2 เป็นช่วงสุดท้ายของการทำโครงงาน ผู้ทำโครงงานได้ทำการทดลองเพื่อให้ได้ผลสรุปของการทำโครงงาน รวมทั้งจัดทำเอกสารประกอบการทำโครงงาน จนเป็นผลสำเร็จ ถึงแม้ว่าบทสรุปสุดท้าย โครงงานจะไม่ประสบผลสำเร็จ 100% แต่ก็ได้บรรลุวัตถุประสงค์ที่ตั้งไว้ ในเอกสารชุดนี้จึงมีข้อมูลทั้งหมดของการทำโครงงานครั้งนี้ที่เป็นประโยชน์ต่อการศึกษาต่อไป

7.2 วิจารณ์การทำโครงงาน

7.2.1 ปัญหาของการทำโครงงาน

- **การออกแบบ**

ปัญหาของการออกแบบที่ได้กล่าวในหัวข้อ 7.1 แล้วนั้น คือ ความชำนาญในการออกแบบการเข้ารหัสและถอดรหัสข้อมูล ตามอัลกอริทึมแบบ TDES ต้องอาศัยเวลา โดยทำการออกแบบ แก้ไข และทดลอง ต้องทำไปพร้อมๆกัน เริ่มจากการออกแบบแต่ละส่วนย่อยและทดลองการทำงานให้ถูกต้อง ถ้ามีข้อผิดพลาดก็กลับมาแก้ไข เมื่อถูกต้องแล้วก็ออกแบบระบบที่ใหญ่ขึ้น แล้วทำการทดสอบการทำงานถ้าผิดพลาดก็แก้ไข ทำเช่นนี้จนกระทั่งได้ผลลัพธ์ที่ถูกต้องสมบูรณ์ จึงใช้เวลามากในส่วนนี้

- **การโปรแกรมภาษา VHDL และการใช้โปรแกรมจำลองการทำงาน**

จากที่กล่าวไปแล้วว่า ภาษา VHDL เป็นภาษาใหม่สำหรับผู้ทำโครงงาน ทำให้การศึกษาและทำความเข้าใจต้องใช้เวลามาก ไม่สามารถเขียนโปรแกรมให้สมบูรณ์ได้ภายในระยะเวลาอันสั้น ทำให้เกิดปัญหาอย่างมากเช่นกัน ต้องปรึกษาอาจารย์ที่ปรึกษา และค้นคว้าจากตำราหลายๆ เล่ม ในการเขียนโปรแกรมเพื่อทำโครงงานครั้งนี้ ในส่วนของการใช้โปรแกรมจำลองการทำงาน คือ โปรแกรม ModelSim SE Plus 6.0 ผู้ทำโครงงานได้ติดปัญหาในส่วนของการติดตั้งเนื่องจากไม่ผ่านลิขสิทธิ์ โดยได้ทดลองติดตั้งเป็นเวลานานพอสมควรจึสำเร็จ จากนั้นปัญหาก็คือ การใช้งานโปรแกรมที่ติดตั้ง เนื่องจากเป็นรุ่นที่ไม่มีเอกสารบอกวิธีการใช้ ผู้ทำโครงงานจึงได้ค้นคว้าจากอินเตอร์และศึกษาวิธีใช้จนสามารถใช้งานโปรแกรมดังกลาวได้

- **การ Implement ลงบนฮาร์ดแวร์**

การ Implement ลงบนฮาร์ดแวร์ เป็นปัญหาที่สำคัญที่สุดที่ทำให้โครงงานนี้ประสบผลสำเร็จได้ไม่ถึง 100% เนื่องจากเกิดปัญหาจากข้อจำกัดของ FPGA ในส่วนของการ Synthesis เพื่อ Implement ลงบนอุปกรณ์ สามารถทำได้แต่ไม่ผ่านกระบวนการ แม้ว่าผู้ทำโครงงานจะได้ทดลองเปลี่ยนรุ่นของ FPGA ที่ดีกว่า และมีจำนวนเกทที่มากกว่า ซึ่งมีในปัจจุบัน ก็ไม่สามารถทำได้

- **เครื่องมือที่ใช้ในการทำโครงงาน**

เครื่องมือหรือโปรแกรมที่ใช้ในการทำโครงงานครั้งนี้ ถือว่ายังมีความพร้อมในระดับหนึ่งเท่านั้น ไม่ถือว่าสมบูรณ์ เนื่องจากเป็นซอฟท์แวร์ที่ให้มาใช้เพื่อการศึกษา จึงทำให้การทำโครงงานครั้งนี้ไม่ประสบผลสำเร็จเท่าที่ควร ตัวอย่างเช่น รุ่นของ FPGA ที่มีให้เลือกใช้ มีเพียงไม่กี่รุ่น จึงไม่สามารถจำลองการทำงานได้

7.2.2 ข้อเสนอแนะของการทำโครงงาน

โครงงานอุปกรณ์เข้ารหัสและถอดรหัสข้อมูลด้วยอุปกรณ์ FPGA เป็นโครงงานที่น่าสนใจมาก ซึ่งการทำโครงงานครั้งนี้ยังไม่ประสบผลสำเร็จ 100% ถ้าสามารถพัฒนาให้สำเร็จตามวัตถุประสงค์จนได้อุปกรณ์เข้ารหัสและถอดรหัสข้อมูล จะเป็นโครงงานที่มีประโยชน์มาก ข้อเสนอแนะอีกประการหนึ่งก็คือ ควรให้มีการพัฒนาทางด้านซอฟท์แวร์ควบคู่ไปกับการทำโครงงาน เพื่อจะได้เห็นประโยชน์และประยุกต์ใช้งานได้

การพัฒนาส่วนที่ 1 ควรให้มีการพัฒนาโครงงานนี้ต่อไปให้สำเร็จ โดยมีการออกแบบโปรแกรมให้ Optimize ยิ่งขึ้น ซึ่งจะเป็นการพัฒนาควบคู่ไปกับอุปกรณ์ FPGA ที่จะออกรุ่นใหม่ที่มีข้อจำกัดทางด้านฮาร์ดแวร์น้อยลง

การพัฒนาในส่วนที่ 2 ควรให้มีการพัฒนาให้สามารถใช้กับการสื่อสารข้อมูลระบบเครือข่ายคอมพิวเตอร์ เพราะอุปกรณ์นี้มีความสำคัญในการรักษาความปลอดภัยของข้อมูลมาก โดยอาจพัฒนาให้อุปกรณ์อยู่บน LAN Card และสร้าง Application ขึ้นมาใช้งาน

ส่วนหนึ่งของการทำโครงงานครั้งนี้ ผู้ที่จะศึกษาเพื่อพัฒนาต่อควรศึกษาข้อมูลในเอกสารชุดนี้เพื่อเป็นแนวทาง และหาข้อมูลเพิ่มเติมจากแหล่งข้อมูลอื่นมาประกอบกัน เชื่อว่าจะสามารถทำโครงงานนี้ให้เสร็จสมบูรณ์ 100% ได้อย่างแน่นอน

ภาคผนวก ก
โปรแกรมภาษาวีเอชดีแอล

โปรแกรมส่วนของ IP Component

ip .vhdl (Initail Permutation)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ip is port
(
        pt : in std_logic_vector(1 TO 64);
        l0x : out std_logic_vector(1 TO 32);
        r0x : out std_logic_vector(1 TO 32)
);
end ip;
architecture behavior of ip is
begin
        l0x(1)<=pt(58); l0x(2)<=pt(50); l0x(3)<=pt(42); l0x(4)<=pt(34);
        l0x(5)<=pt(26); l0x(6)<=pt(18); l0x(7)<=pt(10); l0x(8)<=pt(2);
        l0x(9)<=pt(60); l0x(10)<=pt(52); l0x(11)<=pt(44); l0x(12)<=pt(36);
        l0x(13)<=pt(28); l0x(14)<=pt(20); l0x(15)<=pt(12); l0x(16)<=pt(4);
        l0x(17)<=pt(62); l0x(18)<=pt(54); l0x(19)<=pt(46); l0x(20)<=pt(38);
        l0x(21)<=pt(30); l0x(22)<=pt(22); l0x(23)<=pt(14); l0x(24)<=pt(6);
        l0x(25)<=pt(64); l0x(26)<=pt(56); l0x(27)<=pt(48); l0x(28)<=pt(40);
        l0x(29)<=pt(32); l0x(30)<=pt(24); l0x(31)<=pt(16); l0x(32)<=pt(8);
        r0x(1)<=pt(57); r0x(2)<=pt(49); r0x(3)<=pt(41); r0x(4)<=pt(33);
        r0x(5)<=pt(25); r0x(6)<=pt(17); r0x(7)<=pt(9); r0x(8)<=pt(1);
        r0x(9)<=pt(59); r0x(10)<=pt(51); r0x(11)<=pt(43); r0x(12)<=pt(35);
        r0x(13)<=pt(27); r0x(14)<=pt(19); r0x(15)<=pt(11); r0x(16)<=pt(3);
        r0x(17)<=pt(61); r0x(18)<=pt(53); r0x(19)<=pt(45); r0x(20)<=pt(37);
        r0x(21)<=pt(29); r0x(22)<=pt(21); r0x(23)<=pt(13); r0x(24)<=pt(5);
```

```
        r0x(25)<=pt(63); r0x(26)<=pt(55); r0x(27)<=pt(47); r0x(28)<=pt(39);
        r0x(29)<=pt(31); r0x(30)<=pt(23); r0x(31)<=pt(15); r0x(32)<=pt(7);
end behavior;
```

โปรแกรมส่วนของ **FP Component**

fp.vhdl (Final Permutation)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fp is port
(
        l,r : in std_logic_vector(1 to 32);
        ct : out std_logic_vector(1 to 64)
);
end fp;
architecture behaviour of fp is
begin
        ct(1)<=r(8); ct(2)<=l(8); ct(3)<=r(16); ct(4)<=l(16); ct(5)<=r(24); ct(6)<=l(24);
        ct(7)<=r(32); ct(8)<=l(32); ct(9)<=r(7); ct(10)<=l(7); ct(11)<=r(15); ct(12)<=l(15);
        ct(13)<=r(23); ct(14)<=l(23); ct(15)<=r(31); ct(16)<=l(31); ct(17)<=r(6); ct(18)<=l(6);
        ct(19)<=r(14); ct(20)<=l(14); ct(21)<=r(22); ct(22)<=l(22); ct(23)<=r(30); ct(24)<=l(30);
        ct(25)<=r(5); ct(26)<=l(5); ct(27)<=r(13); ct(28)<=l(13); ct(29)<=r(21); ct(30)<=l(21);
        ct(31)<=r(29); ct(32)<=l(29); ct(33)<=r(4); ct(34)<=l(4); ct(35)<=r(12); ct(36)<=l(12);
        ct(37)<=r(20); ct(38)<=l(20); ct(39)<=r(28); ct(40)<=l(28); ct(41)<=r(3); ct(42)<=l(3);
        ct(43)<=r(11); ct(44)<=l(11); ct(45)<=r(19); ct(46)<=l(19); ct(47)<=r(27); ct(48)<=l(27);
        ct(49)<=r(2); ct(50)<=l(2); ct(51)<=r(10); ct(52)<=l(10); ct(53)<=r(18); ct(54)<=l(18);
        ct(55)<=r(26); ct(56)<=l(26); ct(57)<=r(1); ct(58)<=l(1); ct(59)<=r(9); ct(60)<=l(9);
        ct(61)<=r(17); ct(62)<=l(17); ct(63)<=r(25); ct(64)<=l(25);
end behaviour;
```

โปรแกรมส่วนของ keyschedual

keysched.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity keysched is
port (
        key     : in std_logic_vector(1 to 64);
        k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
                : out std_logic_vector(1 to 48)
        );
end keysched;
architecture behaviour of keysched is
        signal  c0x,c1x,c2x,c3x,c4x,c5x,c6x,c7x,c8x,c9x,c10x,c11x,c12x,c13x,c14x,c15x,c16x :
                std_logic_vector(1 to 28);
        signal  d0x,d1x,d2x,d3x,d4x,d5x,d6x,d7x,d8x,d9x,d10x,d11x,d12x,d13x,d14x,d15x,d16x :
                std_logic_vector(1 to 28);

        component pc1
        port (
                key         : in std_logic_vector(1 TO 64);          --- Component Block pc1
                c0x,d0x : out std_logic_vector(1 TO 28)              --- use to cut
8,16,24,32,40,48,56,64
                );
        end component;
        component pc2
        port (
                c,d : in std_logic_vector(1 TO 28);          --- Component Block pc2
                k : out std_logic_vector(1 TO 48)
                );
        end component;
```

```
begin
pc_1: pc1
        port map ( key=>key, c0x=>c0x, d0x=>d0x );
                c1x<=To_StdLogicVector(to_bitvector(c0x) rol 1);
                d1x<=To_StdLogicVector(to_bitvector(d0x) rol 1);
                c2x<=To_StdLogicVector(to_bitvector(c1x) rol 1);
                d2x<=To_StdLogicVector(to_bitvector(d1x) rol 1);    --- Shift creat Cn ,Dn
(1<=n<=16)
                c3x<=To_StdLogicVector(to_bitvector(c2x) rol 2);
                d3x<=To_StdLogicVector(to_bitvector(d2x) rol 2);
                c4x<=To_StdLogicVector(to_bitvector(c3x) rol 2);
                d4x<=To_StdLogicVector(to_bitvector(d3x) rol 2);
                c5x<=To_StdLogicVector(to_bitvector(c4x) rol 2);
                d5x<=To_StdLogicVector(to_bitvector(d4x) rol 2);
                c6x<=To_StdLogicVector(to_bitvector(c5x) rol 2);
                d6x<=To_StdLogicVector(to_bitvector(d5x) rol 2);
                c7x<=To_StdLogicVector(to_bitvector(c6x) rol 2);
                d7x<=To_StdLogicVector(to_bitvector(d6x) rol 2);
                c8x<=To_StdLogicVector(to_bitvector(c7x) rol 2);
                d8x<=To_StdLogicVector(to_bitvector(d7x) rol 2);
                c9x<=To_StdLogicVector(to_bitvector(c8x) rol 1);
                d9x<=To_StdLogicVector(to_bitvector(d8x) rol 1);
                c10x<=To_StdLogicVector(to_bitvector(c9x) rol 2);
                d10x<=To_StdLogicVector(to_bitvector(d9x) rol 2);
                c11x<=To_StdLogicVector(to_bitvector(c10x) rol 2);
                d11x<=To_StdLogicVector(to_bitvector(d10x) rol 2);
                c12x<=To_StdLogicVector(to_bitvector(c11x) rol 2);
                d12x<=To_StdLogicVector(to_bitvector(d11x) rol 2);
                c13x<=To_StdLogicVector(to_bitvector(c12x) rol 2);
                d13x<=To_StdLogicVector(to_bitvector(d12x) rol 2);
                c14x<=To_StdLogicVector(to_bitvector(c13x) rol 2);
                d14x<=To_StdLogicVector(to_bitvector(d13x) rol 2);
                c15x<=To_StdLogicVector(to_bitvector(c14x) rol 2);
                d15x<=To_StdLogicVector(to_bitvector(d14x) rol 2);
```

```
                c16x<=To_StdLogicVector(to_bitvector(c15x) rol 1);
                d16x<=To_StdLogicVector(to_bitvector(d15x) rol 1);
    pc2x1: pc2 port map ( c=>c1x, d=>d1x, k=>k1x );
    pc2x2: pc2 port map ( c=>c2x, d=>d2x, k=>k2x );
    pc2x3: pc2 port map ( c=>c3x, d=>d3x, k=>k3x );
    pc2x4: pc2 port map ( c=>c4x, d=>d4x, k=>k4x );
    pc2x5: pc2 port map ( c=>c5x, d=>d5x, k=>k5x );
    pc2x6: pc2 port map ( c=>c6x, d=>d6x, k=>k6x );
    pc2x7: pc2 port map ( c=>c7x, d=>d7x, k=>k7x );        --- assing to subkey
    pc2x8: pc2 port map ( c=>c8x, d=>d8x, k=>k8x );
    pc2x9: pc2 port map ( c=>c9x, d=>d9x, k=>k9x );
    pc2x10: pc2 port map ( c=>c10x, d=>d10x, k=>k10x );
    pc2x11: pc2 port map ( c=>c11x, d=>d11x, k=>k11x );
    pc2x12: pc2 port map ( c=>c12x, d=>d12x, k=>k12x );
    pc2x13: pc2 port map ( c=>c13x, d=>d13x, k=>k13x );
    pc2x14: pc2 port map ( c=>c14x, d=>d14x, k=>k14x );
    pc2x15: pc2 port map ( c=>c15x, d=>d15x, k=>k15x );
    pc2x16: pc2 port map ( c=>c16x, d=>d16x, k=>k16x );
    end behaviour;
```

pc1.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity pc1 is port
(
        key : in std_logic_vector(1 TO 64);
        c0x,d0x : out std_logic_vector(1 TO 28)
);
end pc1;
architecture behavior of pc1 is
signal XX : std_logic_vector(1 to 56);
begin
```

```
XX(1)<=key(57); XX(2)<=key(49); XX(3)<=key(41); XX(4)<=key(33);
XX(5)<=key(25); XX(6)<=key(17); XX(7)<=key(9); XX(8)<=key(1);
XX(9)<=key(58); XX(10)<=key(50); XX(11)<=key(42); XX(12)<=key(34);
XX(13)<=key(26); XX(14)<=key(18); XX(15)<=key(10); XX(16)<=key(2);
XX(17)<=key(59); XX(18)<=key(51); XX(19)<=key(43); XX(20)<=key(35);
XX(21)<=key(27); XX(22)<=key(19); XX(23)<=key(11); XX(24)<=key(3);
XX(25)<=key(60); XX(26)<=key(52); XX(27)<=key(44); XX(28)<=key(36);
XX(29)<=key(63); XX(30)<=key(55); XX(31)<=key(47); XX(32)<=key(39);
XX(33)<=key(31); XX(34)<=key(23); XX(35)<=key(15); XX(36)<=key(7);
XX(37)<=key(62); XX(38)<=key(54); XX(39)<=key(46); XX(40)<=key(38);
XX(41)<=key(30); XX(42)<=key(22); XX(43)<=key(14); XX(44)<=key(6);
XX(45)<=key(61); XX(46)<=key(53); XX(47)<=key(45); XX(48)<=key(37);
XX(49)<=key(29); XX(50)<=key(21); XX(51)<=key(13); XX(52)<=key(5);
XX(53)<=key(28); XX(54)<=key(20); XX(55)<=key(12); XX(56)<=key(4);
c0x<=XX(1 to 28); d0x<=XX(29 to 56);
end behavior;
```

pc2.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity pc2 is
port
(
        c,d : in std_logic_vector(1 TO 28);
        k : out std_logic_vector(1 TO 48)
);
end pc2;
architecture behavior of pc2 is
        signal YY : std_logic_vector(1 to 56);
begin
        YY(1 to 28)<=c; YY(29 to 56)<=d;
```

```
k(1)<=YY(14); k(2)<=YY(17); k(3)<=YY(11); k(4)<=YY(24); k(5)<=YY(1);
k(6)<=YY(5);
k(7)<=YY(3); k(8)<=YY(28); k(9)<=YY(15); k(10)<=YY(6); k(11)<=YY(21);
k(12)<=YY(10);
k(13)<=YY(23); k(14)<=YY(19); k(15)<=YY(12); k(16)<=YY(4); k(17)<=YY(26);
k(18)<=YY(8);
k(19)<=YY(16); k(20)<=YY(7); k(21)<=YY(27); k(22)<=YY(20);k(23)<=YY(13);
k(24)<=YY(2);
k(25)<=YY(41); k(26)<=YY(52); k(27)<=YY(31); k(28)<=YY(37);k(29)<=YY(47);
k(30)<=YY(55);
k(31)<=YY(30); k(32)<=YY(40); k(33)<=YY(51); k(34)<=YY(45);k(35)<=YY(33); k(36)<=
YY(48);
k(37)<=YY(44); k(38)<=YY(49); k(39)<=YY(39); k(40)<=YY(56);k(41)<=YY(34);
k(42)<=YY(53);
k(43)<=YY(46); k(44)<=YY(42); k(45)<=YY(50); k(46)<=YY(36);k(47)<=YY(29);
k(48)<=YY(32);
end behavior;
```

โปรแกรมส่วนของ **Roundfunction**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity roundfunc is port
        (
                clk   : in std_logic;
                reset : in std_logic;
                li,ri : in std_logic_vector(1 to 32);
                k     : in std_logic_vector(1 to 48);
                lo,ro : out std_logic_vector(1 to 32)
        );
end roundfunc;
```

```
architecture behaviour of roundfunc is
        signal xp_to_xor : std_logic_vector(1 to 48);
        signal b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
                                : std_logic_vector(1 to 6);
        signal so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
                                : std_logic_vector(1 to 4);
        signal ppo,r_toreg32,l_toreg32
                                : std_logic_vector(1 to 32);
component xp
        port (
                ri : in std_logic_vector(1 TO 32);
                e  : out std_logic_vector(1 TO 48)
                );
end component;
component desxor1
        port (
                e   : in std_logic_vector(1 TO 48);
                b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
                        : out std_logic_vector (1 TO 6);
                k       : in std_logic_vector (1 TO 48)
                );
end component;
component s1
        port (
                b  : in std_logic_vector(1 to 6);
                so : out std_logic_vector(1 to 4)
                );
end component;
component s2
        port (
                b  : in std_logic_vector(1 to 6);
                so : out std_logic_vector(1 to 4)
                );
end component;
```

```
component s3
        port (
            b : in std_logic_vector(1 to 6);
            so : out std_logic_vector(1 to 4)
        );
end component;
component s4
        port (
            b : in std_logic_vector(1 to 6);
            so : out std_logic_vector(1 to 4)
            );
end component;
component s5
        port (
            b : in std_logic_vector(1 to 6);
            so : out std_logic_vector(1 to 4)
            );
end component;
component s6
        port (
            b : in std_logic_vector(1 to 6);
            so : out std_logic_vector(1 to 4)
        );
end component;
component s7
        port (
            b : in std_logic_vector(1 to 6);
            so : out std_logic_vector(1 to 4)
        );
end component;
component s8
        port (
            b : in std_logic_vector(1 to 6);
            so : out std_logic_vector(1 to 4)
```

```
                    );
    end component;
    component pp
            port (
                    so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
                            : in std_logic_vector(1 to 4);
                    ppo  : out std_logic_vector(1 to 32)
                    );
    end component;
    component desxor2
            port (
                    d,l  : in std_logic_vector(1 to 32);
                    q          : out std_logic_vector(1 to 32)
                    );
    end component;

    component reg32
            port (
                    a          : in std_logic_vector (1 to 32);
                    q          : out std_logic_vector (1 to 32);
                    reset : in std_logic;
                    clk : in std_logic
                    );
    end component;
    begin
    xpension: xp
            port map ( ri=>ri, e=>xp_to_xor );

    des_xor1: desxor1
            port map ( e=>xp_to_xor, k=>k, b1x=>b1x, b2x=>b2x, b3x=>b3x, b4x=>b4x, b5x=>b5x,
                                    b6x=>b6x, b7x=>b7x, b8x=>b8x );
    s1a: s1
            port map ( b=>b1x, so=>so1x );
    s2a: s2
```

```
            port map ( b=>b2x, so=>so2x );
  s3a: s3
            port map ( b=>b3x, so=>so3x );
  s4a: s4
            port map ( b=>b4x, so=>so4x );
  s5a: s5
            port map ( b=>b5x, so=>so5x );
  s6a: s6
            port map ( b=>b6x, so=>so6x );
  s7a: s7
            port map ( b=>b7x, so=>so7x );
  s8a: s8
            port map ( b=>b8x, so=>so8x );

  pperm: pp
            port map ( so1x=>so1x, so2x=>so2x, so3x=>so3x, so4x=>so4x, so5x=>so5x,
            so6x=>so6x, so7x=>so7x, so8x=>so8x, ppo=>ppo );
  des_xor2: desxor2
            port map ( d=>ppo, l=>li, q=>r_toreg32 );
  l_toreg32<=ri;
  register32_left: reg32
            port map ( a=>l_toreg32, q=>lo, reset=>reset, clk=>clk );
  register32_right: reg32
            port map ( a=>r_toreg32, q=>ro, reset=>reset, clk=>clk );
  end;
```

desxor1.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity desxor1 is port
        (
                e    : in std_logic_vector(1 TO 48);
```

```
                b1x,b2x,b3x,b4x,b5x,b6x,b7x,b8x
                        : out std_logic_vector (1 TO 6);
                k    : in std_logic_vector (1 TO 48)
        );
end desxor1;
architecture behavior of desxor1 is
        signal XX : std_logic_vector( 1 to 48);
begin
        XX<=k xor e;
        b1x<=XX(1 to 6);
        b2x<=XX(7 to 12);
        b3x<=XX(13 to 18);
        b4x<=XX(19 to 24);
        b5x<=XX(25 to 30);
        b6x<=XX(31 to 36);
        b7x<=XX(37 to 42);
        b8x<=XX(43 to 48);
end behavior;
```

desxor2.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity desxor2 is port
(
d,l : in std_logic_vector(1 to 32);
q : out std_logic_vector(1 to 32)
);
end desxor2;
architecture behaviour of desxor2 is
begin
        q<=d xor l;
end;
```

pp.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity pp is port
(
so1x,so2x,so3x,so4x,so5x,so6x,so7x,so8x
: in std_logic_vector(1 to 4);
ppo : out std_logic_vector(1 to 32)
);
end pp;
architecture behaviour of pp is
signal XX : std_logic_vector(1 to 32);
begin
        XX(1 to 4)<=so1x; XX(5 to 8)<=so2x; XX(9 to 12)<=so3x; XX(13 to 16)<=so4x;
        XX(17 to 20)<=so5x; XX(21 to 24)<=so6x; XX(25 to 28)<=so7x; XX(29 to 32)<=so8x;
        ppo(1)<=XX(16); ppo(2)<=XX(7); ppo(3)<=XX(20); ppo(4)<=XX(21);
        ppo(5)<=XX(29); ppo(6)<=XX(12); ppo(7)<=XX(28); ppo(8)<=XX(17);
        ppo(9)<=XX(1); ppo(10)<=XX(15); ppo(11)<=XX(23); ppo(12)<=XX(26);
        ppo(13)<=XX(5); ppo(14)<=XX(18); ppo(15)<=XX(31); ppo(16)<=XX(10);
        ppo(17)<=XX(2); ppo(18)<=XX(8); ppo(19)<=XX(24); ppo(20)<=XX(14);
        ppo(21)<=XX(32); ppo(22)<=XX(27); ppo(23)<=XX(3); ppo(24)<=XX(9);
        ppo(25)<=XX(19); ppo(26)<=XX(13); ppo(27)<=XX(30); ppo(28)<=XX(6);
        ppo(29)<=XX(22); ppo(30)<=XX(11); ppo(31)<=XX(4); ppo(32)<=XX(25);
end;
```

reg32.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity reg32 is
port(
        a : in std_logic_vector (1 to 32);
        q : out std_logic_vector (1 to 32);
        reset : in std_logic;
        clk : in std_logic
);
end reg32;
architecture synth of reg32 is
        signal memory : std_logic_vector (1 to 32) ;
begin
process(clk,reset)
begin
        if(clk = '1' and clk'event) then        -- on affecte la mémoire interne au coup d'horloge
        memory <= a;
        end if;
        if(reset = '1') then
        memory <= (others => '0');
        end if;
end process;
q <= memory;
end synth;
```

s1.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
 entity s1 is port
(
b : in std_logic_vector(1 to 6);
so : out std_logic_vector(1 to 4)
);
end s1;
```

```
architecture behaviour of s1 is
begin
process(b)
        begin
                case b is
                        when "000000"=> so<=To_StdLogicVector(Bit_Vector'(x"c"));
                        when "000010"=> so<=To_StdLogicVector(Bit_Vector'(x"4"));
                        when "000100"=> so<=To_StdLogicVector(Bit_Vector'(x"d"));
                        when "000110"=> so<=To_StdLogicVector(Bit_Vector'(x"1"));
                        when "001000"=> so<=To_StdLogicVector(Bit_Vector'(x"2"));
                        when "001010"=> so<=To_StdLogicVector(Bit_Vector'(x"f"));
                        when "001100"=> so<=To_StdLogicVector(Bit_Vector'(x"b"));
                        when "001110"=> so<=To_StdLogicVector(Bit_Vector'(x"8"));
                        when "010000"=> so<=To_StdLogicVector(Bit_Vector'(x"3"));
                        when "010010"=> so<=To_StdLogicVector(Bit_Vector'(x"a"));
                        when "010100"=> so<=To_StdLogicVector(Bit_Vector'(x"6"));
                        when "010110"=> so<=To_StdLogicVector(Bit_Vector'(x"c"));
                        when "011000"=> so<=To_StdLogicVector(Bit_Vector'(x"5"));
                        when "011010"=> so<=To_StdLogicVector(Bit_Vector'(x"9"));
                        when "011100"=> so<=To_StdLogicVector(Bit_Vector'(x"0"));
                        when "011110"=> so<=To_StdLogicVector(Bit_Vector'(x"7"));
                        when "000001"=> so<=To_StdLogicVector(Bit_Vector'(x"0"));
                        when "000011"=> so<=To_StdLogicVector(Bit_Vector'(x"f"));
                        when "000101"=> so<=To_StdLogicVector(Bit_Vector'(x"7"));
                        when "000111"=> so<=To_StdLogicVector(Bit_Vector'(x"4"));
                        when "001001"=> so<=To_StdLogicVector(Bit_Vector'(x"e"));
                        when "001011"=> so<=To_StdLogicVector(Bit_Vector'(x"2"));
                        when "001101"=> so<=To_StdLogicVector(Bit_Vector'(x"d"));
                        when "001111"=> so<=To_StdLogicVector(Bit_Vector'(x"1"));
                        when "010001"=> so<=To_StdLogicVector(Bit_Vector'(x"a"));
                        when "010011"=> so<=To_StdLogicVector(Bit_Vector'(x"6"));
                        when "010101"=> so<=To_StdLogicVector(Bit_Vector'(x"c"));
                        when "010111"=> so<=To_StdLogicVector(Bit_Vector'(x"b"));
                        when "011001"=> so<=To_StdLogicVector(Bit_Vector'(x"9"));
```

```
when "011011"=> so<=To_StdLogicVector(Bit_Vector'(x"5"));
when "011101"=> so<=To_StdLogicVector(Bit_Vector'(x"3"));
when "011111"=> so<=To_StdLogicVector(Bit_Vector'(x"8"));
when "100000"=> so<=To_StdLogicVector(Bit_Vector'(x"4"));
when "100010"=> so<=To_StdLogicVector(Bit_Vector'(x"1"));
when "100100"=> so<=To_StdLogicVector(Bit_Vector'(x"e"));
when "100110"=> so<=To_StdLogicVector(Bit_Vector'(x"8"));
when "101000"=> so<=To_StdLogicVector(Bit_Vector'(x"d"));
when "101010"=> so<=To_StdLogicVector(Bit_Vector'(x"6"));
when "101100"=> so<=To_StdLogicVector(Bit_Vector'(x"2"));
when "101110"=> so<=To_StdLogicVector(Bit_Vector'(x"b"));
when "110000"=> so<=To_StdLogicVector(Bit_Vector'(x"f"));
when "110010"=> so<=To_StdLogicVector(Bit_Vector'(x"c"));
when "110100"=> so<=To_StdLogicVector(Bit_Vector'(x"9"));
when "110110"=> so<=To_StdLogicVector(Bit_Vector'(x"7"));
when "111000"=> so<=To_StdLogicVector(Bit_Vector'(x"3"));
when "111010"=> so<=To_StdLogicVector(Bit_Vector'(x"a"));
when "111100"=> so<=To_StdLogicVector(Bit_Vector'(x"5"));
when "111110"=> so<=To_StdLogicVector(Bit_Vector'(x"0"));
when "100001"=> so<=To_StdLogicVector(Bit_Vector'(x"f"));
when "100011"=> so<=To_StdLogicVector(Bit_Vector'(x"c"));
when "100101"=> so<=To_StdLogicVector(Bit_Vector'(x"8"));
when "100111"=> so<=To_StdLogicVector(Bit_Vector'(x"2"));
when "101001"=> so<=To_StdLogicVector(Bit_Vector'(x"4"));
when "101011"=> so<=To_StdLogicVector(Bit_Vector'(x"9"));
when "101101"=> so<=To_StdLogicVector(Bit_Vector'(x"1"));
when "101111"=> so<=To_StdLogicVector(Bit_Vector'(x"7"));
when "110001"=> so<=To_StdLogicVector(Bit_Vector'(x"5"));
when "110011"=> so<=To_StdLogicVector(Bit_Vector'(x"b"));
when "110101"=> so<=To_StdLogicVector(Bit_Vector'(x"3"));
when "110111"=> so<=To_StdLogicVector(Bit_Vector'(x"e"));
when "111001"=> so<=To_StdLogicVector(Bit_Vector'(x"a"));
when "111011"=> so<=To_StdLogicVector(Bit_Vector'(x"0"));
when "111101"=> so<=To_StdLogicVector(Bit_Vector'(x"6"));
```

```
                    when others=> so<=To_StdLogicVector(Bit_Vector'(x"d"));
                    end case;
        end process;
end;
```

desenc.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity desenc is
port
  (
        pt    : in std_logic_vector(1 TO 64);
        key   : in std_logic_vector(1 TO 64);
        ct    : out std_logic_vector(1 TO 64);
        reset : in std_logic;
        clk   : in std_logic
    );
end desenc;
architecture behavior of desenc is
    signal k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
        : std_logic_vector(1 to 48);
    signal l0xa,l1x,l2x,l3x,l4x,l5x,l6x,l7x,l8x,l9x,l10x,l11x,l12x,l13x,l14x,l15x,l16x
        : std_logic_vector(1 to 32);
    signal r0xa,r1x,r2x,r3x,r4x,r5x,r6x,r7x,r8x,r9x,r10x,r11x,r12x,r13x,r14x,r15x,r16x
        : std_logic_vector(1 to 32);
component keysched
    port (
        key  : in std_logic_vector(1 to 64);
        k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
            : out std_logic_vector(1 to 48)
        );
end component;
```

```
component ip
        port (
                pt  : in std_logic_vector(1 TO 64);
                l0x : out std_logic_vector(1 TO 32);
                r0x : out std_logic_vector(1 TO 32)
                );
end component;
component roundfunc
        port (
                clk   : in std_logic;
                reset : in std_logic;
                li,ri : in std_logic_vector(1 to 32);
                k     : in std_logic_vector(1 to 48);
                lo,ro : out std_logic_vector(1 to 32)
                );
end component;
component fp
        port (
                l,r : in std_logic_vector(1 to 32);
                ct  : out std_logic_vector(1 to 64)
                );
end component;
begin
keyscheduling: keysched
        port map ( key=>key, k1x=>k1x, k2x=>k2x, k3x=>k3x, k4x=>k4x, k5x=>k5x, k6x=>k6x,
                        k7x=>k7x, k8x=>k8x, k9x=>k9x, k10x=>k10x, k11x=>k11x, k12x=>k12x,
                        k13x=>k13x, k14x=>k14x, k15x=>k15x, k16x=>k16x
                        );
iperm: ip
        port map ( pt=>pt, l0x=>l0xa, r0x=>r0xa );

round1: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l0xa, ri=>r0xa, k=>k1x, lo=>l1x, ro=>r1x );
round2: roundfunc
```

```
        port map ( clk=>clk, reset=>reset, li=>l1x, ri=>r1x, k=>k2x, lo=>l2x, ro=>r2x );
round3: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l2x, ri=>r2x, k=>k3x, lo=>l3x, ro=>r3x );
round4: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l3x, ri=>r3x, k=>k4x, lo=>l4x, ro=>r4x );
round5: roundfunc
        port map ( clk=>clk, reset=>reset,  li=>l4x, ri=>r4x, k=>k5x, lo=>l5x, ro=>r5x );
round6: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l5x, ri=>r5x, k=>k6x, lo=>l6x, ro=>r6x );
round7: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l6x, ri=>r6x, k=>k7x, lo=>l7x, ro=>r7x );
round8: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l7x, ri=>r7x, k=>k8x, lo=>l8x, ro=>r8x );
round9: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l8x, ri=>r8x, k=>k9x, lo=>l9x, ro=>r9x );
round10: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l9x, ri=>r9x, k=>k10x, lo=>l10x, ro=>r10x );
round11: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l10x, ri=>r10x, k=>k11x, lo=>l11x, ro=>r11x );
round12: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l11x, ri=>r11x, k=>k12x, lo=>l12x, ro=>r12x );
round13: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l12x, ri=>r12x, k=>k13x, lo=>l13x, ro=>r13x );
round14: roundfunc
        port map ( clk=>clk, reset=>reset,  li=>l13x, ri=>r13x, k=>k14x, lo=>l14x, ro=>r14x );
round15: roundfunc
        port map ( clk=>clk, reset=>reset,  li=>l14x, ri=>r14x, k=>k15x, lo=>l15x, ro=>r15x );
round16: roundfunc
        port map ( clk=>clk, reset=>reset,  li=>l15x, ri=>r15x, k=>k16x, lo=>l16x, ro=>r16x );
fperm: fp
        port map ( l=>r16x, r=>l16x, ct=>ct );
end behavior;
```

desdec.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity desdec is
port
  (
            ct    : in std_logic_vector(1 TO 64);
            key   : in std_logic_vector(1 TO 64);
            pt    : out std_logic_vector(1 TO 64);
            reset : in std_logic;
            clk   : in std_logic
      );
end desdec;
architecture behavior of desdec is
      signal k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
            : std_logic_vector(1 to 48);
      signal l0xa,l1x,l2x,l3x,l4x,l5x,l6x,l7x,l8x,l9x,l10x,l11x,l12x,l13x,l14x,l15x,l16x
            : std_logic_vector(1 to 32);
      signal r0xa,r1x,r2x,r3x,r4x,r5x,r6x,r7x,r8x,r9x,r10x,r11x,r12x,r13x,r14x,r15x,r16x
            : std_logic_vector(1 to 32);
component keysched
      port (
            key  : in std_logic_vector(1 to 64);
            k1x,k2x,k3x,k4x,k5x,k6x,k7x,k8x,k9x,k10x,k11x,k12x,k13x,k14x,k15x,k16x
                  : out std_logic_vector(1 to 48)
            );
end component;
component ip
      port (
            pt  : in std_logic_vector(1 TO 64);
            l0x : out std_logic_vector(1 TO 32);
            r0x : out std_logic_vector(1 TO 32)
```

```
                    );
    end component;
    component roundfunc
            port (
                    clk   : in std_logic;
                    reset : in std_logic;
                    li,ri : in std_logic_vector(1 to 32);
                    k     : in std_logic_vector(1 to 48);
                    lo,ro : out std_logic_vector(1 to 32)
                    );
    end component;
    component fp
            port (
                    l,r : in std_logic_vector(1 to 32);
                    ct  : out std_logic_vector(1 to 64)
                    );
    end component;
    begin
    keyscheduling: keysched
            port map ( key=>key, k1x=>k1x, k2x=>k2x, k3x=>k3x, k4x=>k4x, k5x=>k5x, k6x=>k6x,
                    k7x=>k7x, k8x=>k8x, k9x=>k9x, k10x=>k10x, k11x=>k11x, k12x=>k12x,
                    k13x=>k13x, k14x=>k14x, k15x=>k15x, k16x=>k16x
                    );
    iperm: ip
            port map (·pt=>ct, l0x=>l0xa, r0x=>r0xa );
    round1: roundfunc
            port map ( clk=>clk, reset=>reset, li=>l0xa, ri=>r0xa, k=>k16x, lo=>l1x, ro=>r1x );
    round2: roundfunc
            port map ( clk=>clk, reset=>reset, li=>l1x, ri=>r1x, k=>k15x, lo=>l2x, ro=>r2x );
    round3: roundfunc
            port map ( clk=>clk, reset=>reset, li=>l2x, ri=>r2x, k=>k14x, lo=>l3x, ro=>r3x );
    round4: roundfunc
            port map ( clk=>clk, reset=>reset, li=>l3x, ri=>r3x, k=>k13x, lo=>l4x, ro=>r4x );
    round5: roundfunc
```

```
        port map ( clk=>clk, reset=>reset,   li=>l4x, ri=>r4x, k=>k12x, lo=>l5x, ro=>r5x );
round6: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l5x, ri=>r5x, k=>k11x, lo=>l6x, ro=>r6x );
round7: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l6x, ri=>r6x, k=>k10x, lo=>l7x, ro=>r7x );
round8: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l7x, ri=>r7x, k=>k9x, lo=>l8x, ro=>r8x );
round9: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l8x, ri=>r8x, k=>k8x, lo=>l9x, ro=>r9x );
round10: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l9x, ri=>r9x, k=>k7x, lo=>l10x, ro=>r10x );
round11: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l10x, ri=>r10x, k=>k6x, lo=>l11x, ro=>r11x );
round12: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l11x, ri=>r11x, k=>k5x, lo=>l12x, ro=>r12x );
round13: roundfunc
        port map ( clk=>clk, reset=>reset, li=>l12x, ri=>r12x, k=>k4x, lo=>l13x, ro=>r13x );
round14: roundfunc
        port map ( clk=>clk, reset=>reset,   li=>l13x, ri=>r13x, k=>k3x, lo=>l14x, ro=>r14x );
round15: roundfunc
        port map ( clk=>clk, reset=>reset,   li=>l14x, ri=>r14x, k=>k2x, lo=>l15x, ro=>r15x );
round16: roundfunc
        port map ( clk=>clk, reset=>reset,   li=>l15x, ri=>r15x, k=>k1x, lo=>l16x, ro=>r16x );
fperm: fp
        port map ( l=>r16x, r=>l16x, ct=>pt );
end behavior;
```

Tdes_enc.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Tdes_Enc is
port
```

```
(
        pt    : in std_logic_vector(1 TO 64);
        key   : in std_logic_vector(1 TO 64);
        ct    : out std_logic_vector(1 TO 64);
        reset : in std_logic;
        clk   : in std_logic
    );
end Tdes_Enc;
architecture behavior of Tdes_Enc is
    signal cpt1, cpt2 : std_logic_vector(1 to 64);
component desenc1
    port (
        pt    : in std_logic_vector(1 TO 64);
        key   : in std_logic_vector(1 TO 64);
        ct    : out std_logic_vector(1 TO 64);
        reset : in std_logic;
        clk   : in std_logic
        );
end component;
component desdec2
    port (
        ct    : in std_logic_vector(1 TO 64);
        key   : in std_logic_vector(1 TO 64);
        pt    : out std_logic_vector(1 TO 64);
        reset : in std_logic;
        clk   : in std_logic
        );
end component;
component desenc3
    port (
        pt    : in std_logic_vector(1 TO 64);
        key   : in std_logic_vector(1 TO 64);
        ct    : out std_logic_vector(1 TO 64);
        reset : in std_logic;
```

```
                clk   : in std_logic
                );
end component;
begin
        Tdes_Enc1: desenc1
                port map ( pt=>pt, key=>key, clk=>clk, reset=>reset, ct=>cpt1 );
        Tdes_Dec2: desdec2
                port map ( ct=>cpt1, key=>key, clk=>clk, reset=>reset, pt=>cpt2 );
        Tdes_Enc3: desenc3
                port map ( pt=>cpt2, key=>key, clk=>clk, reset=>reset, ct=>ct );
end behavior;
```

Tdes_Dec .vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Tdes_Dec is
port
  (
                ct    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                        pt    : out std_logic_vector(1 TO 64);
               reset : in std_logic;
               clk   : in std_logic
        );
end Tdes_Dec;
architecture behavior of Tdes_Dec is
        signal cpt3, cpt4 : std_logic_vector(1 to 64);
component desdec3
        port (
                ct    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                pt    : out std_logic_vector(1 TO 64);
```

```
                reset : in std_logic;
                clk   : in std_logic
                );
    end component;
    component desenc2
            port (
                pt    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                ct    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
    end component;
    component desdec1
            port (
                ct    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                pt    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
    end component;
    begin
    Tdes_Dec1: desdec3
            port map ( ct=>ct, key=>key, clk=>clk, reset=>reset, pt=>cpt3 );
    Tdes_Enc2: desenc2
            port map ( pt=>cpt3, key=>key, clk=>clk, reset=>reset, ct=>cpt4 );
    Tdes_Dec3: desdec1
            port map ( ct=>cpt4, key=>key, clk=>clk, reset=>reset, pt=>pt );
    end behavior;
```

Tdes_Enc.vhdl(key in)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Tdes_Enc is
port
  (
                pt    : in std_logic_vector(1 TO 64);
                ct    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
        );
end Tdes_Enc;
architecture behavior of Tdes_Enc is
        signal cpt1, cpt2, key1, key2, key3 : std_logic_vector(1 to 64);
component desenc1
        port (
                pt    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                ct    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
end component;
component desdec2
        port (
                ct    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                pt    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
end component;
component desenc3
        port (
                pt    : in std_logic_vector(1 TO 64);
```

```
                key   : in std_logic_vector(1 TO 64);
                ct    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
end component;
begin
        key1 <= x"0123456789abcdef" ;
        key2 <= x"5555555555555555" ;
        key3 <= x"fedcba9876543210" ;
Tdes_Enc1: desenc1
        port map ( pt=>pt, key=>key1, clk=>clk, reset=>reset, ct=>cpt1 );
Tdes_Dec2: desdec2
        port map ( ct=>cpt1, key=>key2, clk=>clk, reset=>reset, pt=>cpt2 );
Tdes_Enc3: desenc3
        port map ( pt=>cpt2, key=>key3, clk=>clk, reset=>reset, ct=>ct );
end behavior;
```

Tdes_Enc.vhdl(key in)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Tdes_Enc is
port
  (
                pt    : in std_logic_vector(1 TO 64);
                ct    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
        );
end Tdes_Enc;
architecture behavior of Tdes_Enc is
        signal cpt3, cpt4, key1, key2, key3 : std_logic_vector(1 to 64);
```

```
component desdec3
        port (
                ct    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                pt    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
end component;
component desenc2
        port (
                pt    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                ct    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
end component;
component desdec1
        port (
                ct    : in std_logic_vector(1 TO 64);
                key   : in std_logic_vector(1 TO 64);
                pt    : out std_logic_vector(1 TO 64);
                reset : in std_logic;
                clk   : in std_logic
                );
end component;
begin
        key1 <= x"0123456789abcdef" ;
        key2 <= x"5555555555555555" ;
        key3 <= x"fedcba9876543210" ;
Tdes_Dec3:  desdec3
        port map ( ct=>ct, key=>key3, clk=>clk, reset=>reset, pt=>cpt3 );
Tdes_Enc2: desenc2
```

```
        port map ( pt=>cpt3, key=>key2, clk=>clk, reset=>reset, ct=>cpt4 );
Tdes_Des1: desenc1
        port map ( ct=>cpt4, key=>key3, clk=>clk, reset=>reset, pt=>pt );
end behavior;
```

บรรณานุกรม

[1] "Cryptographic Algorithms for Protection of Computer Data During Transmission and Dormant Storage," *Federal Register* 38, No. 93 (May 15, 1973).

[2] *Data Encryption Standard*, Federal Information Processing Standard (FIPS) Publication 46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (January 1977).

[3] Carl H. Meyer and Stephen M. Matyas, *Cryptography: A New Dimension in Computer Data Security*, John Wiley & Sons, New York, 1982.

[4] Dorthy Elizabeth Robling Denning, *Cryptography and Data Security*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

[5] D.W. Davies and W.L. Price, *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronics Funds Transfer*, Second Edition, John Wiley & Sons, New York, 1984, 1989.

[6] Miles E. Smid and Dennis K. Branstad, "The Data Encryption Standard: Past and Future," in Gustavus J. Simmons, ed., *Contemporary Cryptography: The Science of Information Integrity*, IEEE Press, 1992.

[7] Douglas R. Stinson, *Cryptography: Theory and Practice*, CRC Press, Boca Raton, 1995.

[8] Bruce Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, New York, 1996.

[9] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

[10] Christof Paar , "Applied Cryptography And Data Security" (Lecture Notes), Ruhr-Universitat Bochum(http: www.crypto.ruhr-uni-bochum.de) May 2000

[11] S.A Vanstone A.J Menezes, P.C Oorschot, "Handbook of Applied Cryptography", CRC Press, 1997

[11] ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล, ชำนาญ ปัญญาใส,วัชรากร หนูทอง, ซีเอ็ดยูเคชั่น

Web sites

[12] http://www.free-ip.com/DES/index.html (Free_DES implementation)

[13] http://www.aci.net/kalliste/des.html (The DES Algorithm IIlustrated)

[14] http://www.mathweb.free.fr/crypto/moderne/des.php3 (La saga du Des)

[15] http://www.ailogictechnology.com