การออกแบบระบบการเข้ารหัสลับเพื่อปกปิดข้อมูลโดยการเข้ารหัสแบบ DES / 3DES

DESIGN OF CRYPTOGRAPHY SYSTEM USING DES / 3DES ALGORITHM

โดย

นายธนะ คำรณฤทธิศร 44010200

นายนภดล ห้วยแก้ว 44010237

นายนัทธี จันทวัฒน์ 44010249

อาจารย์ที่ปรึกษา

รศ.ดร.กอบชัย เดชหาญ

อ.ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การออกแบบระบบการเข้ารหัสลับเพื่อปกปิดข้อมูลโดยการเข้ารหัสแบบ DES / 3DES

DESIGN OF CRYPTOGRAPHY SYSTEM USING DES / 3DES ALGORITHM

โดย

นายธนะ คำรณฤทธิศร

นายนภดล ห้วยแก้ว

นายนัทธี จันทวัฒน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ภาควิชา วิศวกรรมโทรคมนาคม

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การออกแบบระบบการเข้ารหัสลับเพื่อปกปิดข้อมูลโดยการเข้ารหัสแบบ DES / 3DES**

DESIGN OF CRYPTOGRAPHY SYSTEM USING DES / 3DES ALGORITHM

ผู้จัดทำ

1: นายธนะ	คำรณฤทธิศร	**44010200**
2: นายนภดล	ห้วยแก้ว	**44010237**
3: นายนัทธี	จันทวัฒน์	**44010249**

(อาจารย์ที่ปรึกษา)

(รศ.ดร.กอบชัย เดชหาญ)

(อาจารย์ที่ปรึกษา)

(อ.ศรวัฒน์ ชิวปรีชา)

การออกแบบระบบการเข้ารหัสลับเพื่อปกปิดข้อมูลโดยการเข้ารหัสแบบ DES / 3DES

DESIGN OF CRYPTOGRAPHY SYSTEM USING DES / 3DES ALGORITHM

โดย นายธนะ คำรณฤทธิศร 44010200

นายนภดล ห้วยแก้ว 44010237

นายนัทธี จันทวัฒน์ 44010249

อาจารย์ที่ปรึกษา

รศ.ดร.กอบชัย เดชหาญ

อ.ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

โครงงานนี้นำเสนอการเข้ารหัสลับเพื่อปกปิดข้อมูลสำหรับการรักษาความปลอดภัยของข้อมูลที่ใช้บนบัตรสมาร์ทการ์ด โดยได้ใช้ DES และ 3DES Algorithm ซึ่งเป็น Algorithm ที่มีการใช้งานกันทั่วไปในปัจจุบันในการออกแบบ ซึ่งกุญแจรหัสที่ใช้ในการเข้ารหัสและถอดรหัส มีลักษณะเป็น Private key นอกจากนั้นจะมีส่วนของการออกแบบวงจรหน่วยความจำ (RAM) และวงจรเชื่อมต่อแบบอนุกรม (UART Module) สำหรับเชื่อมต่อเข้ากับคอมพิวเตอร์ เพื่อทดสอบการทำงานของบัตรสมาร์ทการ์ดต้นแบบที่ออกแบบขึ้น

ABSTRACT

The project presents the cryptography system for smart card using DES (Data Encryption Standard) and 3DES (Triple Data Encryption Standard) in design method. Presently, this algorithm are using widely , and the private keys are used for encryption and decryption. Moreover, this project includes the design of memory module (RAM), serial interfacing circuits (UART Module) for interfacing with computer in order to test the proposed smart card prototype design.

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

ที่มาของโครงงาน

ปัจจุบันเทคโนโลยีคอมพิวเตอร์ได้พัฒนาไปอย่างรวดเร็ว การสื่อสารข้อมูลต่างๆก็ทำได้อย่างรวดเร็วเช่นกัน ดังนั้นความปลอดภัยในการรักษาข้อมูลที่เป็นความลับจึงจำเป็นอย่างยิ่ง แม้ว่าในปัจจุบันนี้จะมีระบบรักษาความปลอดภัยในคอมพิวเตอร์ที่ช่วยอำนวยความสะดวกอย่างมากมาย เช่นการใช้รหัสผ่าน, การเข้ารหัสข้อมูล เป็นต้น แต่ข้อจำกัดของมันก็คือต้องใช้คอมพิวเตอร์เพื่อการประมวลผลทั้งสิ้น ดังนั้นหากเราสามารถสร้างคอมโพเน็นต์ (Component) เฉพาะสำหรับการเข้าและถอดรหัสที่สามารถอิมพลีเมนต์ (Implement) บนชิป (Chip) ที่เป็นฮาร์ดแวร์ (Hardware) ได้จะทำให้เราสามารถใช้อุปกรณ์นี้ในการเข้ารหัสและถอดรหัสได้อย่างสะดวก ทั้งยังสามารถนำไปประยุกต์ใช้กับวงจรเข้ารหัสและถอดรหัสอื่นๆได้ด้วย

การสร้างคอมโพเน็นต์สำหรับการเข้ารหัสและถอดรหัสจำเป็นต้องเลือกอัลกอริทึมที่ได้รับความนิยมและเป็นที่ยอมรับในการใช้งาน ในที่นี้เราเลือก DES (Data Encryption Standrad) และ 3DES (Triple Data Encrytion Standard) เพื่อใช้ในการเข้ารหัสและถอดรหัส โดยอัลกอริทึมนี้ได้รับความนิยมในแง่ความซับซ้อนในการทำงาน ความยาวคีย์ (Key) ที่มีขนาดเหมาะสม

สำหรับภาษาที่ใช้ในการออกแบบจะใช้ภาษาวีเอชดีแอล (VHDL) ซึ่งเป็นภาษาที่บรรยายสถาปัตยกรรมของฮาร์ดแวร์ พร้อมทั้งสนับสนุนการออกแบบจากบนลงล่าง (TOP-Down Design) เป็นภาษาที่เหมาะจะนำไปพัฒนาอิมพลีเมนต์บนฮาร์ดแวร์ เพื่อเป็นชิปที่สามารถทำการเข้ารหัสได้

บทที่ 2
ภาษาวีเอชดีแอล (VHDL)

ปัจจุบันในการออกแบบระบบดิจิตอลที่เรารู้จักกันจะเป็นการออกแบบโดยใช้การวาดวงจร โดยใช้โปรแกรมช่วยในการวาด ซึ่งผู้ออกแบบจะต้องมีทักษะสูงในการออกแบบ และต้องใช้เวลามากในการออกแบบระบบจำลองการทำงาน และตลอดจนถึงการแก้ไขความถูกต้องของระบบ ซึ่งในการออกแบบจะต้องอ้างอิงเทคโนโลยีที่ใช้ออกแบบระบบดิจิตอล ถ้าต้องการเปลี่ยนเทคโนโลยีของระบบที่ออกแบบค่อนข้างทำได้ยากและใช้เวลามาก และเมื่อต้องการออกแบบระบบดิจิตอลที่มีความซับซ้อนสูง ยิ่งทำได้ยากหรืออาจทำไม่ได้ โดยใช้กระบวนการออกแบบแบบเก่าๆที่กล่าวมา แต่ในการออกแบบระบบดิจิตอลในปัจจุบันได้มีกระบวนการออกแบบรูปแบบใหม่ที่มีประสิทธิภาพสูง รวดเร็ว และไม่ยึดติดกับเทคโนโลยีที่ใช้ออกแบบ กระบวนการดังกล่าว คือ การออกแบบจากบนลงล่าง ซึ่งกระบวนการดังกล่าวจะใช้บรรยายฮาร์ดแวร์ (HDL: Hardware Description Language) ในการออกแบบ จำลองการทำงานสังเคราะห์วงจร ในรูปแบบเทคโนโลยีที่เราต้องการ และสามารถทดสอบวงจรที่ออกแบบได้บนฮาร์ดแวร์จำพวกชิป FPGA (Field Programmable Gate Arrays) ดังนั้นการออกแบบสามารถทำได้ง่ายและมีความสะดวกรวดเร็วมากยิ่งขึ้น

2.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล

VHDL ย่อมาจาก VHSIC Hardware Description Language เป็นภาษาบรรยายฮาร์ดแวร์ประเภทหนึ่ง โดยภาษาเกิดขึ้นจากโครงการที่มีชื่อ VHSIC (Very High Speed Integrated Circuits) ที่ถูกพัฒนาขึ้นโดยกระทรวงกลาโหมของสหรัฐอเมริกา (DOD: Department of Defense) ในช่วงปี ค.ศ. 1980 โดยเป้าหมายของโครงการนี้ก็เพื่อพัฒนาขีดความสามารถในการออกแบบวงจรรวมให้สูงขึ้นและสามารถทำได้ง่ายมากขึ้น โดยมีเป้าหมายหลัก 2 ประการคือ

- ต้องการภาษาที่สามารถรองรับการออกแบบวงจรที่มีความซับซ้อนได้
- ต้องการภาษาที่เป็นมาตรฐานหรือเป็นภาษากลางที่ทำให้สามารถเผยแพร่ผลงานการออกแบบกันภายในกลุ่มนักออกแบบด้วยกันได้

จนกระทั่งในปี ค.ศ. 1986 ภาษาวีเอชดีแอล ได้เริ่มมีการปรับปรุงภาษา เพื่อให้สามารถกำหนดเป็นมาตรฐานของ IEEE (Institute of Electrical and Electronics Engineers) โดยสามารถประกาศเป็นมาตรฐานได้ในเดือนธันวาคมปี ค.ศ. 1987

2.2 ส่วนประกอบของภาษาวีเอชดีแอล

การใช้ภาษาวีเอชดีแอล ในการออกแบบวงจรดิจิตอลไม่ว่าจะเป็นวงจรพื้นฐานหรือวงจรที่ซับซ้อนทุกวงจรที่จะออกแบบ จะต้องเขียนในรูปแบบของส่วนประกอบวีเอชดีแอล ซึ่งเป็นพื้นฐานของการออกแบบโดยหน่วยการออกแบบในภาษาวีเอชดีแอล จะแบ่งออกเป็นประเภทต่างๆ คือ

1.หน่วยการออกแบบเอนติตี (Entity Design Unit)

2.หน่วยการออกแบบสถาปัตยกรรม (Architecture Design Unit)

3.หน่วยการออกแบบแพ็กเกจ (Package Design Unit)

4.หน่วยการออกแบบโครงแบบ (Configuration Design Unit)

1) การออกแบบเอนติตี (Entity Design Unit)

เป็นหน่วยการออกแบบที่ใช้สำหรับติดต่อระหว่างอุปกรณ์ภายนอกกับวงจรที่จะเขียนขึ้น รวมทั้งการส่งผ่านค่าพารามิเตอร์บางอย่างระหว่างวงจรกับอุปกรณ์ภายนอก โดยรูปแบบการเขียนดังนี้

```
ENTITY entity_name IS
    GENERIC ( generic_list ) ;
    PORT ( port_list ) ;
END entity_name;
```

รูปที่ 2.1 การออกแบบเอนติตี้

ประเภทของพอร์ตที่สามารถประกาศใช้ในเอนติตี มี 4 ประเภทดังนี้

อินพุต (Input)

เป็นพอร์ตทิศทางเดียว ที่นำค่าสัญญาณจากอุปกรณ์ภายนอก เข้ามาภายในวงจร สามารถนำมาป้อนให้กับสัญญาณอื่นหรืออ่านค่าได้แต่ไม่สามารถถูกเขียนจากภายในวงจรได้

เอาต์พุต (Output)

เป็นพอร์ตทิศทางเดียว ที่นำค่าสัญญาณจากวงจร ส่งออกไปยังอุปกรณ์ภายนอก สามารถเขียนจากภายในวงจรได้

อินเอาต์ (In-Out Bidirectional)

เป็นพอร์ต 2 ทิศทาง ที่สามารถส่งถูกเขียนและอ่านได้จากภายในวงจร

บัฟเฟอร์ (Buffer Output with internal feedback)

เป็นพอร์ตเอาต์พุตประเภทหนึ่งที่สามารถอ่านค่ากลับเข้ามาภายในวงจรได้

2) การออกแบบสถาปัตยกรรม (Architecture Design Unit)

เป็นหน่วยการออกแบบส่วนที่ใช้เขียนบรรยายพฤติกรรมการทำงานของวงจรที่ต้องการออกแบบ โดยมีความสัมพันธ์กับสิ่งที่กำหนดในเอนติตี้ มีรูปแบบการเขียนดังนี้

```
ARCHITECTURE arch_name OF entity_name IS
        < declarations >
    BEGIN
        < concurrent_statements >
    END aech_name ;
```

รูปที่ 2.2 การออกแบบสถาปัตยกรรม

3) การออกแบบแพ็กเกจ (Package Design Unit)

แพ็กเกจ จะเป็นหน่วยการออกแบบที่ใช้เก็บข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ในการเขียนรูปแบบบรรยายวงจรดิจิตอล โดยข้อมูลในแพ็กเกจสามารถถูกเรียกใช้ได้โดย เอนติตี้ สถาปัตยกรรม หรือด้วยแพ็กเกจ อื่นๆด้วยคำสั่ง USE Statement การเรียกใช้งานแพ็กเกจ จะใช้คำสั่ง USE Statement โดยมีรูปแบบดังนี้

```
USE library_name. package_name. item;
```

รูปที่ 2.3 การเรียกใช้แพกเกจ

โดยปกติการเขียนแพ็กเกจ แพกเกจ จะแบ่งออกเป็น 2 ส่วนคือ

- แพกเกจ ดีแคลร์เลชั่น (Package declaration)
- แพกเกจ บอดี้ (Package body)

Package declaration

เป็นส่วนที่กำหนดชื่อ (Identifier) ของสิ่งที่ประกาศอยู่ใน แพกเกจ สำหรับนำไปใช้ภายนอกตัวของแพกเกจ ถ้าสิ่งใดถูกประกาศในส่วนของ แพกเกจ บอดี้ แต่ไม่ถูกประกาศใน แพกเกจ ดีแคลร์เลชั่น ก็จะไม่สามารถนำไปใช้งานได้จากภายนอก ซึ่งเปรียบเทียบได้กับการประกาศพอร์ตใน เอนติตี้ ที่มีหน้าที่ติดต่ออุปกรณ์ภายนอก ดังนั้นโดยทั่วไปแล้ว แพกเกจ สามารถเขียนขึ้นได้โดยไม่จำเป็นต้องมีส่วนของ แพกเกจ บอดี้ และยังสามารถถูกนำไปใช้ได้จากภายนอก การเขียน แพกเกจ ดีแคลร์เลชั่น มีรูปแบบดังนี้

```
PACKAGE package_name IS
        constant_ declarations
        type_declarations
        component_declarations
    END package_name ;
```

รูปที่ 2.4 การเขียน แพกเกจ ดีแคลร์เลชั่น

Package body

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ (Sequential statement) ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (Subprogram) ที่ได้ประกาศไว้ใน แพกเกจ ดีแคลร์เลชั่น แล้ว โดยมีรูปแบบการเขียนดังนี้

```
PACKAGE_BODY package_name IS
        constant_declarations
        type_ declarations
        subprogram_ declarations
END package_name ;
```

รูปที่ 2.5 การเขียน แพกเกจ บอดี้

4) การออกแบบโครงแบบ (Configuration Design Unit)

รูปแบบของการเขียนส่วน เอนติตี้ สามารถมีหลายๆสถาปัตยกรรม และภายในสถาปัตยกรรม ยังมีอุปกรณ์ที่เรียกว่า คอมโพเนนต์ (Component) ที่อ้างถึงเอนติตี้ อื่นๆอีก ดังนั้นจึงเกิดคำถามขึ้นว่า ในการจำลองการทำงาน (Simulate) ในแต่ละครั้งซอฟต์แวร์จะนำเอาสถาปัตยกรรม อันไหนไปจำลองการทำงาน คำตอบคือเราจะต้องเขียนส่วนคอนฟิกกูเรชันการกำหนดว่าจะนำเอาสถาปัตยกรรม อันไหนไปใช้งาน

2.3 ออบเจ็กต์ในภาษาวีเอชดีแอล VHDL (VHDL Objects)

ออบเจ็กต์ในภาษาวีเอชดีแอล จะใช้เขียนเพื่อบ่งบอกองค์ประกอบส่วนหนึ่งของรูปแบบภาษา ซึ่งเปรียบได้เหมือนกับภาชนะที่มีไว้สำหรับบรรจุค่าต่างๆมีอยู่ 3 ประเภท คือ

ค่าคงที่ (CONSTANT)

เป็นออบเจ็กต์ที่เมื่อกำหนดค่าเริ่มต้นให้แล้วจะคงค่านั้นไว้ตลอด ไม่สามารถดัดแปลง หรือแก้ไขได้ สามารถประกาศใช้ได้ในส่วนที่เป็นส่วนประกาศต่างๆของโมเดล

สัญญาณ (SIGNAL)

เป็นออบเจ็กต์ที่สามารถกำหนดค่าที่สัมพันธ์กับเวลานั้น หมายความว่า สัญญาณสามารถรับค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง สัญญาณจะรับค่าๆหนึ่งได้จากการขับสัญญาณ ซึ่งสามารถประกาศใช้ได้ในส่วนที่เป็นเนื้อที่ของ คอนเคอเร้น บอดี้ (Concurrent body) เท่านั้น ดังนั้นสัญญาณจึงสามารถถูกนำไปใช้ได้ตลอดในโครงสร้างของโมเดลหรือเรียกว่าเป็นโกลบอลออบเจ็กต์ (Global Object)

ตัวแปร (VARIABLE)

เป็นออบเจ็กต์ที่สามารถกำหนดค่าใดๆให้ได้ และสามารถที่จะเปลี่ยนค่าได้ตลอดเวลาการจำลองการทำงาน แต่จะเก็บค่าเดียวเท่านั้นในขณะเวลาหนึ่ง เนื่องจากตัวแปรสามารถประกาศใช้ได้ส่วนที่เป็นส่วนของ ซีเควนเชียล บอดี้ (Sequential body) เท่านั้นอันได้แก่ ส่วนประกาศของ PROCESS, FUNCTION หรือ PROCEDURE ดังนั้น ตัวแปร จึงสามารถนำไปใช้ได้เฉพาะในขอบเขตที่ถูกประกาศใช้เท่านั้นหรือเรียกว่าเป็นโลคอลออบเจ็กต์

การประกาศออบเจ็กต์ (Object Declaration)

การที่จะใช้งานออบเจ็กต์ต่างๆ ในการเขียนรูปแบบด้วยภาษาวีเอชดีแอล นั้นจะต้องมีการประกาศออบเจ็กต์ใช้ก่อนเสมอ ถึงจะสามารถนำไปใช้งาน การประกาศใช้ออบเจ็กต์ มีรูปแบบดังนี้

```
object_class identifier : TYPE := initial_value
```

รูปที่ 2.6 การประกาศออบเจ็กต์

- ออบเจ็ก คลาส (object_class) ได้แก่ สัญญาณที่เป็นค่าคงที่ หรือ ตัวแปร
- การตั้งชื่อ (Identifier) เป็นไปตามกฎของภาษาวีเอลดีแอล ซึ่งจะกล่าวถึงในส่วนต่อไป
- TYPE คือการกำหนดประเภทของออบเจ็กต์ที่ประกาศนั้นๆ นอกจากนั้นยังสามารถกำหนดค่าเริ่มต้นของออบเจ็กต์ได้ (initial_value) ซึ่งเป็นส่วนเสริม ไม่ต้องกำหนดค่าเริ่มต้นก็ได้

2.4 การเขียนแบบลำดับชั้น (Hierarchical Model)

ในการออกแบบวงจรดิจิตอล บางครั้งจำเป็นต้องมีการแบ่งวงจรเป็นบล็อกย่อยๆตามหน้าที่การทำงาน เนื่องจากระบบที่ต้องการออกแบบมีความซับซ้อนสูงไม่สามารถจะออกแบบได้เพียงบล็อกเดียว ทำให้จำเป็นต้องแบ่งวงจรออกเป็นบล็อกย่อยๆ ซึ่งในภาษาวีเอชดีแอล ก็มีความสามารถในการเขียนแบบลำดับชั้น (Hierarchy) ได้โดยเขียนบรรยายวงจรตั้งแต่ระดับบนสุด (Top model) ที่มีแต่ความสัมพันธ์ของการเชื่อมต่อในแต่ละบล็อกย่อย โดยยังไม่มีรายละเอียดของวงจรในแต่ละบล็อก และในระดับล่างลงไปถึงจะมีรายละเอียดของวงจร การออกแบบที่เป็นลักษณะแบบลำดับชั้น ซึ่งจะทำให้ง่ายในการทำความเข้าใจ ตรวจสอบและแก้ไข

2.5 การประกาศคอมโปเนนต์ (Component Declaration)

เป็นการประกาศคอมโปเนนต์ที่ต้องการนำมาเชื่อมต่อกันภายในวงจรส่วนบนสุด (Top mode) จะเป็นการเขียนในลักษณะของลำดับชั้น (Hierarchy) โดยการประกาศคอมโปเนนต์นั้นจะต้องทำการออกแบบคอมโปเนนต์ดังกล่าวไว้ก่อนเรียบร้อยแล้ว เพียงแต่เราเรียกคอมโปเนนต์ดังกล่าวมาใช้งาน โดยการออกแบบคอมโปเนนต์ ก็คล้ายกับออกแบบวงจรทั่วไป คือสร้างส่วนของ เอนติตี้ และ สถาปัตยกรรม ขึ้นมาก่อน แล้วเก็บไว้ในไลบรารี่ (Library) จากนั้นถ้าวงจรที่เราออกแบบขึ้นมาใหม่ต้องการเรียกใช้วงจรที่เคยออกแบบไว้แล้ว สามารถทำได้โดยประกาศส่วนของโมเดลที่มีอยู่แล้วในรูปแบบของคอมโปเนนต์ (Component Declaration) โดยมีรูปแบบการเขียนดังนี้

```
COMPONENT component_name
GENERIC ( generic_list ) ;
PORT ( port_declaration ) ;
  END COMPONENT ;
```

รูปที่ 2.7 การประกาศ คอมโพเนนต์

ซึ่งตำแหน่งของการประกาศส่วนของคอมโปเนนต์สามารถประกาศในบริเวณระหว่างคำสั่ง ARCHITECTURE กับ BEGIN

การเรียกใช้งานคอมโพเนนต์

เป็นการเรียกใช้คอมโปเนนต์ที่มีอยู่แล้ว โดยการเชื่อมต่อสัญญาณต่างๆ ของโมดูลย่อยที่ประกาศไว้ในส่วนของ คอมโพเนนต์ ดีแคลร์เลชั่น (Component declaration) เข้ากับพอร์ตหรือสัญญาณภายในของวงจรระดับบน (Top model) โดยมีรูปแบบดังนี้

```
Instance_label : component_name
    GENERIC MAP ( generic_association_list )
    PORT MAP ( port_name => sig_name ,
               port_name => sig_name ,
                  . . .
               port_name => sig_name ) ;
```

รูปที่ 2.8 การเรียกใช้ คอมโพเนนต์

2.6 ประเภทข้อมูลในภาษาวีเอชดีแอล

ค่าคงที่ (CONSTANT)

ค่าคงที่เป็นการกำหนดค่าที่ประกาศไว้แล้ว จะไม่สามารถเปลี่ยนแปลงค่าได้ เมื่อนำไปใช้ในการเขียนบรรยายพฤติกรรมวงจร โดยมีรูปแบบการเขียนดังนี้

```
CONSTANT constant_name : type_spec := value ;
```

รูปที่ 2.9 การประกาศค่าคงที่

การประกาศค่าคงที่จะถูกประกาศไว้ส่วนต้นของ Architecture, Package declaration, Package body, Entity หรือในส่วน Process declaration ซึ่งจะมีรูปแบบการใช้งานต่างกันออกไปขึ้นอยู่กับว่าประกาศไว้ที่ใด

สัญญาณ (SIGNAL)

สัญญาณ มีหน้าที่ส่งผ่านข้อมูลระหว่าง คอนเคอเร้น สเตทเม้น(Concurrent statement) ภายในสถาปัตยกรรม และใช้ใน ซีเควนเชียล สเตทเม้น (Sequential Statement) ภายในคำสั่ง Process เพื่อกำหนดพฤติกรรมการทำงานของวงจร โดยสัญญาณจะถูกประกาศไว้ที่บริเวณต่างๆ ได้เช่นเดียวกับค่าคงที่ มีรูปแบบการเขียนดังนี้

```
SIGNAL signal_name : type_spec := initial_value
```

รูปที่ 2.10 การประกาศสัญญาณ

ตัวแปร (VARIABLE)

ตัวแปรในภาษาวีเอชดีแอล จะเป็นตัวแปรในลักษณะแบบ โลคอลออบเจ็กต์ ที่ประกาศใช้ในส่วนที่เป็นซีเควนเชียล บอดี้เท่านั้น โดยปกติจะประกาศตัวแปรไว้ในส่วนของคำสั่ง PROCESS หรือในโปรแกรมย่อยถ้าประกาศตัวแปรไว้ในคำสั่ง PROCESS ใด ก็สามารถใช้งานหรืออ้างอิงถึงตัวแปรนั้นได้เฉพาะ PROCESS นั้น ไม่สามารถนำค่าตัวแปรไปใช้นอก PROCESS ได้ โดยคุณสมบัติของตัวแปรจะแตกต่างกับสัญญาณตรงที่สามารถเปลี่ยนค่าได้ทันทีที่ถูกกำหนดค่าให้ ซึ่งต่างจากสัญญาณ ที่ประกาศใช้ในคำสั่ง PROCESS เมื่อมีการเปลี่ยนแปลงค่า จะไม่เปลี่ยนค่าทันที จะเปลี่ยนค่าหลังการทำงานของคำสั่ง PROCESS สิ้นสุดแล้วเท่านั้น การประกาศตัวแปรมีรายละเอียดดังนี้

```
VARIABLE variable_name : variable_type := initial_value ;
```

รูปที่ 2.11 การประกาศค่าตัวแปร

2.7 ตัวดำเนินการภาษาวีเอชดีแอล

ภาษาวีเอชดีแอลจะมีชุดของโอเปอร์เรเตอร์ ที่ใช้สำหรับการเปรียบเทียบ การกระทำการทางคณิตศาสตร์หรือการกระทำทางด้านตรรกะ (Boolean) โดยโอเปอร์เรเตอร์จะเปรียบเสมือนเครื่องมือที่ใช้สำหรับสร้าง อาทีแอล โมดูล ขึ้นมา โดยชุดของโอเปอร์เรเตอร์ทั้งหมดในภาษาวีเอชดีแอล มีดังต่อไปนี้

NOT	-	inversion
AND	-	and function
NAND	-	not – and function
OR	-	or function
NOR	-	not – or function
XOR	-	exclusive – or function
XNOR	-	exclusive – nor function
=	-	equality
/=	-	inequality
>=	-	greater – than or equal
>	-	greater – than
<=	-	less – than or equal
<	-	less – than

ตารางที่ 2.1 ตัวดำเนินการในภาษาวีเอชดีแอล

2.8 คำสั่งแบบลำดับ (Sequential)

ภาษาวีเอชดีแอลเป็นภาษาลักษณะโครงสร้าง ซึ่งแต่ละส่วนมีความสำคัญแตต่างกัน ในการออกแบบวงจรดิจิตอล ซึ่งภาษาวีเอชดีแอลเป็นภาษาที่มีลักษณะเป็นคอนเคอเร้น คือชุดคำสั่งทุกๆคำสั่งที่เขียนไว้ใต้คำสั่ง ARCHITECTURE จะถูกประมวลผลพร้อมๆกันไม่ขึ้นอยู่กับลำดับก่อนหลัง ไม่เหมือนกับการเขียนภาษาทางซอฟต์แวร์ ที่มีลำดับความสำคัญของโค้ดในแต่ละบรรทัด แต่ถ้าต้องการเขียนในรูปแบบลำดับ ภาษาวีเอชดีแอล ก็สามารถเขียนได้โดยมีเงื่อนไขอยู่ว่า การเขียนคำสั่งต่างๆ ที่เป็นแบบลำดับ จะต้องอยู่ภายใต้คำสั่งพิเศษ นั่นคือคำสั่ง PROCESS เท่านั้น ซึ่งคำสั่งที่เป็นรูปแบบ ซีเควนเชียลที่มีให้ใช้ในภาษาวีเอชดีแอล มีดังนี้

IF Statements

IF – END

IF – ELSE

IF – ELSIF – ELSE

CASE Statements

LOOP Statements

FOR LOOP

WHILE LOOP

WAIT Statement

WAIT FOR

WAIT ON

WAIT UNTIL

WAIT

การใช้คำสั่ง PROCESS

จะเป็นการอธิบายคำสั่ง PROCESS ซึ่งเป็นคำสั่งประเภท คอนเคอเร้น สเตทเม้น (Concurrent Statement) แต่ภายในคำสั่งPROCESS จะบรรจุไปด้วยคำสั่งที่เป็นแบบ ซีเควนเชียล สเตทเม้น (Sequential Statement) ซึ่งเป็นคำสั่งที่ทำงานตามลำดับ ซึ่งเป็นประโยชน์มากในการออกแบบซึ่งผู้ออกแบบสามารถเขียนในลักษณะแบบอัลกอริทึม (Algorithm description) ได้ คล้ายคลึงกับการออกแบบโปรแกรมทางซอฟต์แวร์ (Software) โดยคำสั่ง PROCESS นี้จะถูกกำหนดในส่วนของ ARCHITECTURE และคำสั่ง PROCESS นี้สามารถสังเคราะห์ได้ (Synthesizable) มีรูปแบบการเขียนดังนี้

```
Optional_label : PROCESS (optional_sentitivity_list)
        Declaration
BEGIN
        Sequential_statements ;
…
END PROCESS optional_label ;
```

รูปที่ 2.12 การเรียกใช้ PROCESS

โดยรูปแบบของการเขียน PROCESS จะมี 2 รูปแบบ คือ

- คอมบิเนชั่น โปรเซส (Combination process)
- คล็อก โพรเซส (Clocked process)

คอมบิเนชั่น โปรเซส

จะใช้สำหรับเขียนเพื่อออกแบบวงจรในส่วนของคอมไบเนชั่นลอจิก (Combination logic) ในระบบดิจิตอลและคล็อก โพรเซส จะใช้สำหรับเขียนเพื่อออกแบบวงจรที่มีอุปกรณ์ที่ใช้สัญญาณนาฬิกาในการทำงาน เช่น ฟลิปฟลอบ (Flip – Flop) และรวมถึงมีส่วนของวงจรส่วนคอมบิเนชั่นด้วย

การใช้คำสั่ง IF

คำสั่ง IF ในภาษาวีเอชดีแอล มีหลายรูปแบบดังนี้

```
- - IF statement
IF condition THEN
Sequential_statements
END IF ;
- - IF – ELSE  statement
IF condition THEN
Sequential_statements
ELSE
END IF ;
```

รูปที่ 2.13 การเรียกใช้ IF

การใช้คำสั่ง IF – END

```
- - IF Statement
IF condition THEN
Sequential_statement
END IF ;
```

รูปที่ 2.14 การเรียกใช้ IF – END

เงื่อนไขเป็นจริง (True) ทำ ซีเควนเชียล สเตทเม้น

เงื่อนไขเป็นเท็จ (False) ไม่ทำ ซีเควนเชียล สเตทเม้น (END IF)

การใช้คำสั่ง IF – ELSE

```
IF condition THEN
Sequential_statements(1)
ELSE
Sequential_statements(2)
END IF ;
```

รูปที่ 2.15 การเรียกใช้ IF – ELSE

เงื่อนไขเป็นจริง (True) ทำ ซีเควนเชียล สเตทเม้น (1)

เงื่อนไขเป็นเท็จ (False) ทำ ซีเควนเชียล สเตทเม้น (2)

บทที่3
ทฤษฎีอัลกอริทึม

3.1 ดีอีเอส (DES: Data Encryption Standard)

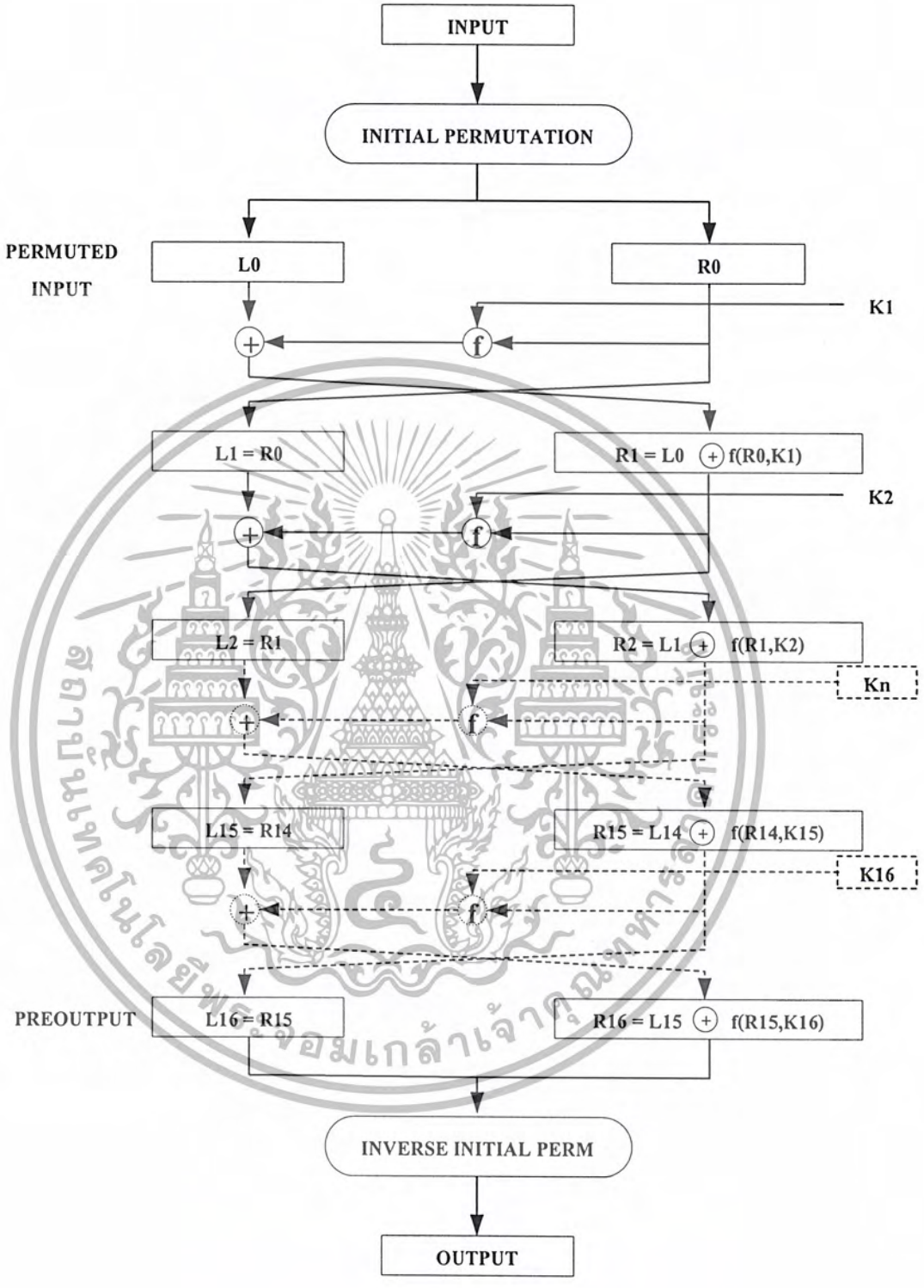
ปัจจุบันได้มีการนำเทคโนโลยีต่างๆ มาใช้ในการลดความเสี่ยงและรักษาความปลอดภัยในการทำกิจกรรมทางพาณิชย์อีเล็กทรอนิกส์ อาทิ การใช้วิธีการเข้ารหัสข้อมูล (Encryption) มาทำการปกป้องความลับของข้อมูล ซึ่งก็เป็นวิธีที่ได้ผลดีพอสมควรทีเดียว แต่ก็ยังมีปัญหาในด้านมาตรฐานและการจัดการที่ดี การเข้ารหัสข้อมูลก่อนที่จะส่งผ่านระบบเครือข่ายอินเตอร์เน็ต นั้นสามารถที่จะปกป้องข้อมูลได้ เพราะข้อมูลจะถูกเปลี่ยนจากข้อความที่สามารถอ่านเข้าใจได้ (Clear Text) ไปเป็นข้อมูลที่ไม่สามารถจะอ่านและเข้าใจได้ (Cipher Text) ดังนั้นหากมีการขโมยข้อมูลไปในระหว่างการส่งข้อมูลนั้น จากระบบเครือข่ายก็แทบจะไม่มีประโยชน์ เนื่องจากผู้ที่สามารถทำการถอดรหัสและอ่านข้อมูลนั้นได้จะต้องเป็นผู้ที่มีกุญแจอีเล็กทรอนิกส์ลับเท่านั้น ซึ่งกุญแจรหัสลับนี้จะให้ไว้กับผู้ส่งและผู้รับข้อมูลเท่านั้น ซึ่งระบบดีอีเอส หรือ ระบบ DES (Data Encryption Standard) เป็นระบบที่นิยมแพร่หลายที่สุดระบบหนึ่ง โดยเฉพาะกลุ่มที่ให้บริการทางด้านการเงิน เช่น ธนาคาร หรือบริษัทไฟแนนซ์ต่างๆ

3.2 การคำนวณการเข้ารหัสแบบดีอีเอส

DES (Data Encryption Standard)เป็นการเข้ารหัสข้อมูลโดยผ่านอัลกอริทึมทางคณิตศาสตร์ สำหรับกระบวนการเข้ารหัสนั้น ข้อมูลที่เราต้องการจะเข้ารหัสเรียกว่า เพลนเท็กซ์ (Plain Text) จะถูกนำมาผ่านอัลกอริทึมทางคณิตศาสตร์โดยใช้คีย์ชุดหนึ่ง ส่วนข้อมูลที่ได้จากเข้ารหัสนั้นเราจะเรียกว่า ไซเฟอร์เท็กซ์ (Cipher Text)ซึ่งเป็นข้อมูลที่เราไม่สามารถอ่านให้เข้าใจได้ และข้อมูลดังกล่าวนี้จะถูกส่งไปยังผู้รับ เมื่อได้รับข้อมูลที่เป็นไซเฟอร์เท็กซ์มาแล้วจะทำการถอดรหัสโดยใช้คีย์ชุดเดียวกันกับคีย์ที่ใช้เข้ารหัส จะทำให้ได้ข้อมูลที่เป็นเพลนเท็กซ์ออกมา คีย์ที่ใช้จะประกอบด้วยข้อมูลเลขฐานสองขนาด 64 บิต โดยที่ 56 บิตมาจากการสุ่มตัวเลข ส่วนอีก 8 บิตเกิดจากการทำพาริตีบิต (Parity Bit) เพื่อทำการตรวจสอบความผิดพลาดของข้อมูลนั่นเอง ผู้ที่จะทำการเข้ารหัสข้อมูลจะมีคีย์เพื่อใช้ในการเข้ารหัส อัลกอริทึมที่ใช้ในการเข้ารหัสนั้นจะเป็นมาตรฐานที่เรารู้จักกันดี แต่ความแตกต่างในการเข้ารหัสอยู่ที่คีย์ที่เราสร้างขึ้นมาใช้ในการเข้ารหัส โดยคีย์แต่ละคีย์นั้นจะทำให้ได้ข้อมูลที่ผ่านการเข้ารหัส (ไซเฟอร์เท็กซ์) ที่แตกต่างกันดังนั้นความปลอดภัยในการเข้ารหัสข้อมูลจึงอยู่ที่การจัดการคีย์ที่เหมาะสม ข้อมูลที่ผ่านการเข้ารหัสจะถูกถอดรหัสกลับคืนมาโดยการใช้คีย์เดียวกันกับที่ใช้เข้ารหัสเท่านั้น สำหรับผู้ที่รับข้อมูลที่ผ่านการเข้ารหัสไป ถึงแม้ว่าจะรู้อัลกอริทึม แต่ถ้าหากไม่มีคีย์ที่ถูกต้องก็ไม่สามารถทำการถอดรหัสข้อมูลออกมาได้

อัลกอริทึมนี้ถูกออกแบบเพื่อให้สามารถทำการเข้ารหัสและถอดรหัสของข้อมูลที่มีขนาด 64 บิต บล็อกที่อยู่ภายใต้การควบคุมของคีย์ขนาด 64 บิตสำหรับการถอดรหัสนั้นจะสามารถทำได้ก็ต่อเมื่อมีคีย์อัน

เดียวกันกับที่ใช้เข้ารหัสโดยการถอดรหัสนั้น จะเป็นการทำย้อนกลับของกระบวนการเข้ารหัสนั่นเองบล็อกข้อมูลที่เข้ารหัสจะถูกสลับสับเปลี่ยนตำแหน่งก่อน จากนั้นจะนำไปผ่านไซเฟอร์ฟังก์ชันจนในขั้นสุดท้ายจึงทำการสับเปลี่ยนตำแหน่งย้อนกลับดังรูปที่ 3.1



รูปที่ 3.1 กระบวนการเข้ารหัสแบบดีอีเอส

3.2.1 กระบวนการสับเปลี่ยนตำแหน่งไอพี (IP: Initial Permutation)

ข้อมูลที่เข้ามาจะเป็นข้อมูล 64 บิตโดยเรียงตำแหน่งจากบิตค่าน้อยสุดเป็นบิตที่ 1 และบิตที่มีค่ามากสุดเป็นบิตสุดท้ายคือบิตที่ 64 ตามตารางที่ 3.1

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

ตารางที่ 3.1 ตารางค่าการจัดเรียงตำแหน่งของข้อมูล

หลังจากนั้นนำข้อมูลมาทำการสลับตำแหน่งตามตารางไอพี ตามรูปที่ 3.2 โดยที่บิตที่ 58 ของข้อมูลเมื่อทำการผ่านการสลับตำแหน่ง บิตดังกล่าวจะย้ายมาอยู่บิตที่ 1 ในตาราง ส่วนบิตที่ 50 นั้นเมื่อผ่านการสลับตำแหน่งก็จะย้ายมาอยู่บิตที่ 2 ในตาราง เป็นต้น

IP								
58	50	42	34	26	18	10	2	
60	52	44	36	28	20	12	4	L
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	R
61	53	45	37	29	21	13	5	
63	55	47	39	31	23	15	7	

ตารางที่ 3.2 การสับเปลี่ยนตำแหน่งไอพี

3.2.2 การคำนวณการเข้ารหัส

1.ข้อมูลที่ทำการสับเปลี่ยนตำแหน่งแล้วจะมีขนาดเท่าเดิมคือขนาด 64 บิต ซึ่งถูกนำมาเป็นอินพุตในการคำนวณการเข้ารหัส

2.ข้อมูล 64 บิตดังกล่าว จะถูกแบ่งออกเป็นสองส่วนคือบล็อก L และบล็อก R โดยแต่ละบล็อกจะมีขนาด 32 บิต โดยเราเริ่มคำนวณ ค่าที่ L_0 และ R_0

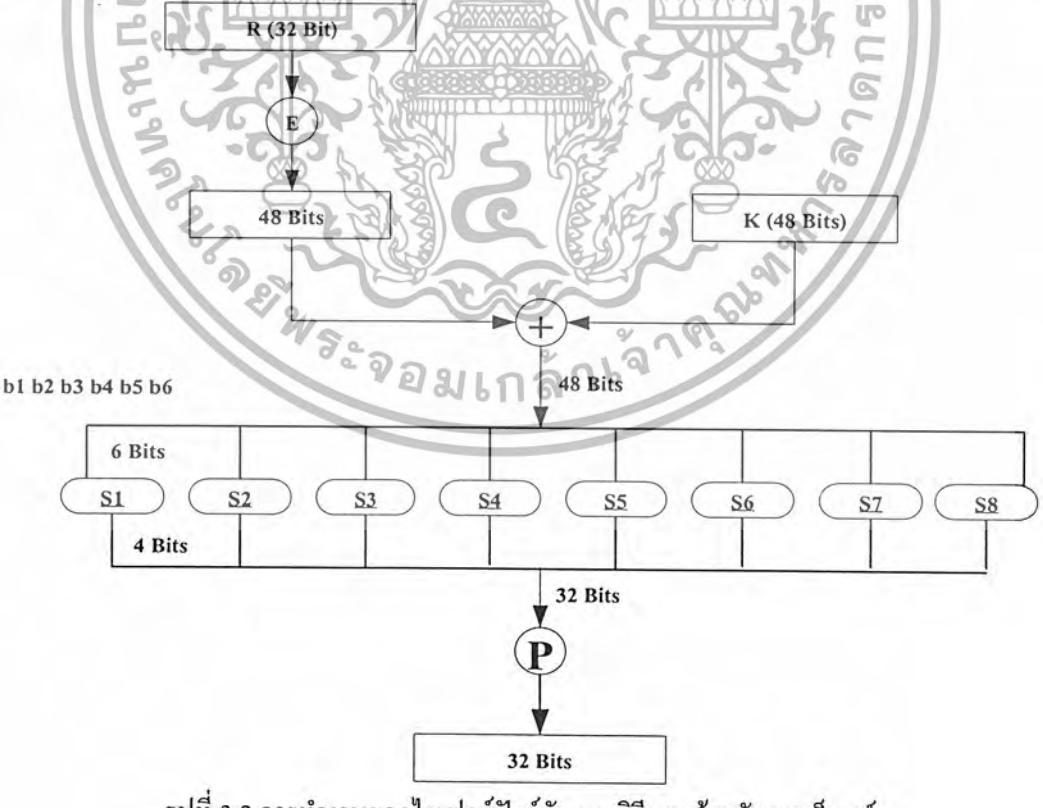
3.การทำงานบนบล็อก L และบล็อก R จะมีทั้งสิ้น 16 รอบการทำงาน โดยแต่ละรอบการทำงานเราจะแทนด้วยค่า n ซึ่ง n จะมีค่าตั้งแต่ 1 ถึง 16 โดยแต่ละบล็อกจะมีการคำนวณดังนี้

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

$$R_n = L_{n-1} \text{ XOR } f(R_{n-1}, K_n)$$

โดยบล็อก L ที่เกิดขึ้นนั้นจะมาจากค่าของบล็อก R ก่อนหน้านี้ และบล็อก R ที่เกิดขึ้นใหม่ในแต่ละรอบจะเกิดจากการเอ็กคลูซีพออ ระหว่าง L บล็อกในรอบนั้น กับผลลัพธ์ของไซเฟอร์ฟังก์ชัน ($f(R_{n-1}, K_n)$) ที่มีอินพุตเป็นบล็อก R ในรอบก่อนหน้านี้และค่าคีย์ย่อยในรอบนั้นๆ

การทำงานของไซเฟอร์ฟังก์ชันและวิธีการเข้ารหัสเอสบ็อกซ์ (Cipher Function and S-Box) ตามรูปที่ 3.2 การทำงานของไซเฟอร์ฟังก์ชันและวิธีการเข้ารหัสเอสบ็อกซ์ (Cipher Function and S-Box) มีการทำงานดังนี้ เริ่มจากการสลับตำแหน่งตามตารางอี ตามตารางที่ 3.3 เป็นฟังก์ชันที่มีการรับอินพุตขนาด 32 บิต แล้วสามารถสร้างเอาต์พุตออกมาขนาด 48 บิต โดยค่าอินพุตที่รับเข้ามานั้นคือ บล็อก R ขนาด 32 บิตนั่นเอง ฟังก์ชัน E มีลักษณะเดียวกันกับการสลับเปลี่ยนตำแหน่งไอพี เพียงแต่ค่าในบางตำแหน่งจะมีการใช้มากกว่าหนึ่งครั้ง



รูปที่ 3.2 การทำงานของไซเฟอร์ฟังก์ชันและวิธีการเข้ารหัสเอสบ็อกซ์

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

ตารางที่ 3.3 การสลับตำแหน่งตามตารางอี

ผลลัพธ์ที่ได้จากการผ่านฟังก์ชัน อี จะทำให้ได้ข้อมูลขนาด 48 บิต โดยข้อมูลนี้จะนำมา เอ็กคลูซีพออ (XOR) กับ K ซึ่งมีขนาด 48 บิต ข้อมูลที่ได้จะถูกแบ่งออกเป็น 8 ส่วนๆละ 6 บิต แล้วส่งให้ เอสบ็อกซ์ เพื่อทำการหาผลลัพธ์ โดยให้ B_i เป็นข้อมูลที่ผ่านเข้ามาที่ S_i หลังจากนำ E มา เอ็กคลูซีพออ กับ K แล้ว โดยมีขนาด 6 บิต (b_1,b_2,b_3,b_4,b_5,b_6) ให้ j เป็นเลขจำนวนเต็มที่มีค่า 1 ถึง 8 เราจะแบ่ง B_j ออกเป็นสองส่วนคือ แถว (Row) และ หลัก (Column) โดยที่ค่าของ แถว = b_1b_6 และส่วนของ หลัก คือ $b_2b_3b_4b_5$ นำค่า แถว และ หลัก มาเทียบในตารางเอส ดังแสดงในตาราง 3.4 จะทำให้ได้ค่าผลลัพธ์ 4 บิตออกมา

S_1

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

S_2

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

S_3

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

S_4

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

S_5

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

S_6

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

S_7

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

S_8

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

ตาราง 3.4 การเทียบค่าตามตารางเอส

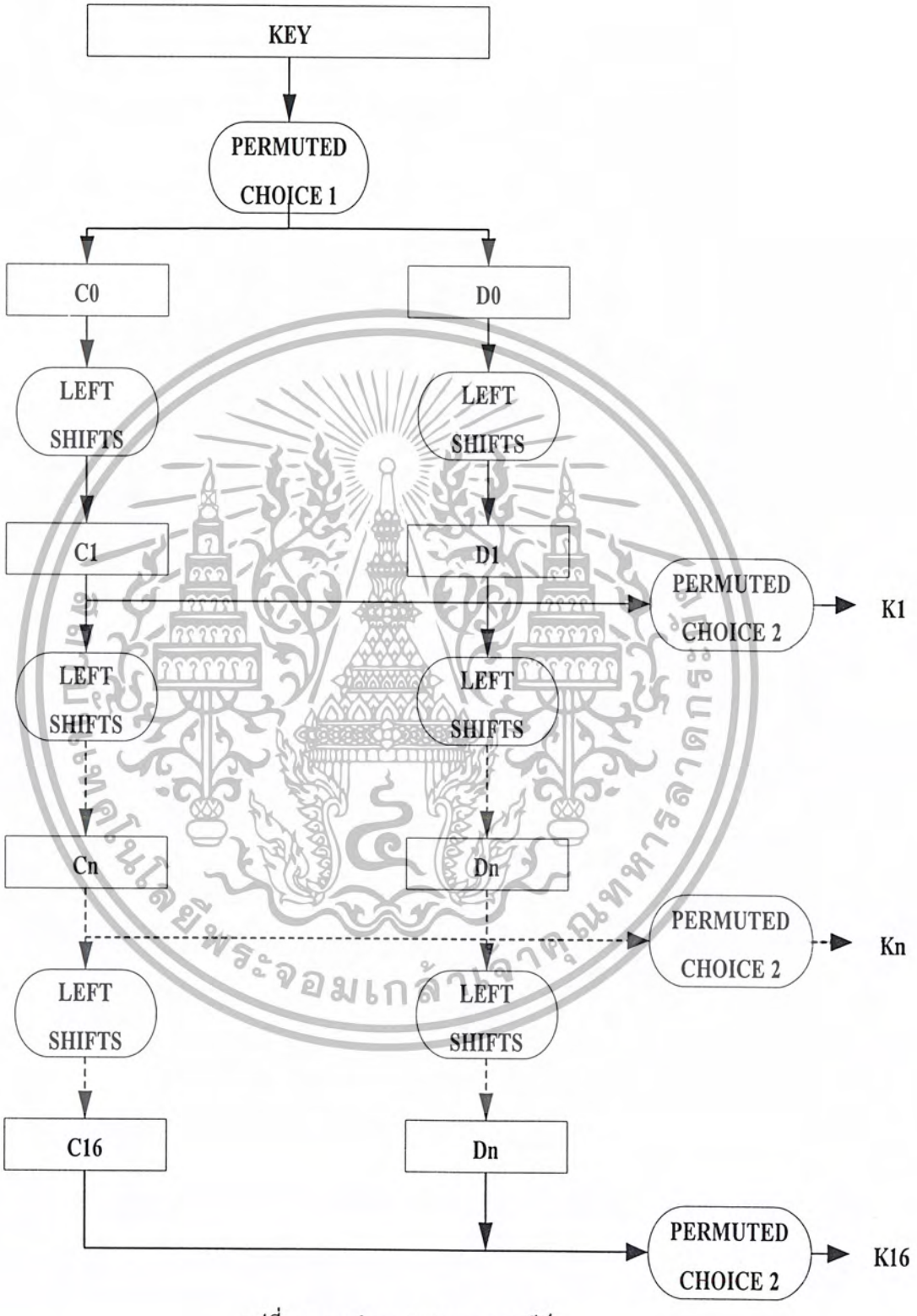
หลังจากที่เราได้ทำการเทียบค่าตามตารางเอส แล้วเราจะนำค่านั้นมาทำการสลับตำแหน่งตามตารางที่ 3.5

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

ตารางที่ 3.5 การสลับตำแหน่งของตารางพี

3.2.3 การคำนวณหมายเลขกุญแจ

ในแต่ละรอบของกระบวนการเข้ารหัสข้อมูลกับคีย์จำนวน 16 คีย์นั้น จะใช้ค่าของคีย์แต่ละชุดที่มีขนาด 48 บิต โดยที่ค่าของคีย์ทั้ง 16 ชุดนั้นได้มาจากคีย์ซึ่งมีขนาด 64 บิต โดยที่บิตตำแหน่งที่ 8,16,...,64 เป็นบิตที่ใช้ตรวจสอบความผิดพลาด โดยกระบวนการในการคำนวณหาค่าคีย์ดังแสดงในรูปที่ 3.3



รูปที่ 3.3 การคำนวณหาหมายเลขคีย์

โดยในตอนแรกเราจะนำคีย์ขนาด 64 บิตนำมาเรียงลำดับตามลำดับเหมือนกับการเรียงตำแหน่งของข้อมูลหลังจากนั้นเราจะตัดบิตตำแหน่งที่ 8,16,...,64 เป็นบิตที่ใช้ตรวจสอบความผิดพลาดออกไป แล้วมาทำการสลับตำแหน่งตามตารางพีซี 1 (PC-1) ตามตารางที่ 3.6

PC-1

57	49	41	33	25	17	9	C
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	D
7	62	54	46	38	30	22	
14	6	61	53	45	37	29	
21	13	5	28	20	12	4	

ตารางที่ 3.6 การสลับตำแหน่งตามตารางพีซี1

ผลลัพธ์จากการทำ พีซี1 จะถูกแบ่งเป็น 2 ส่วนๆละ 28 บิต เรียกว่า C และ D ใช้ในการหาค่าหมายเลขกุญแจ K_i โดยกำหนดให้ C_i และ D_i ซึ่งได้มาจาก C และ Dใช้หาค่า K_i มีสมการดังนี้

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

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

LS_i คือการเลื่อนบิตไปทางซ้ายเท่ากับจำนวนครั้งที่กำหนดตามตารางที่ 3.7

ลำดับ i	จำนวนครั้งที่เลื่อน
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

ตารางที่ 3.7 จำนวนครั้งที่ทำการเลื่อนบิตไปทางซ้ายในแต่ละรอบ

และสมการหมายเลขกุญแจ K_i คือ

$$K_i = PC\text{-}2(C_iD_i)$$

หลังจากนั้นนำค่า C_iD_i มาสลับตำแหน่งตามตารางพีซี 2 (PC-2) ดังแสดงตามตารางที่ 3.8

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

ตารางที่ 3.8 การสลับตำแหน่งตามตาราง พีซี 2

3.2.4 การสับเปลี่ยนตำแหน่งย้อนกลับ (Inverse Initial Permutation: IP^{-1})

โดยหลังจากที่เราได้ค่า L_{16}และ R_{16} มาแล้วเราจะนำค่าทั้งสองส่วนมาทำการสลับตำแหน่งอีกครั้งตามตารางที่ 3.9

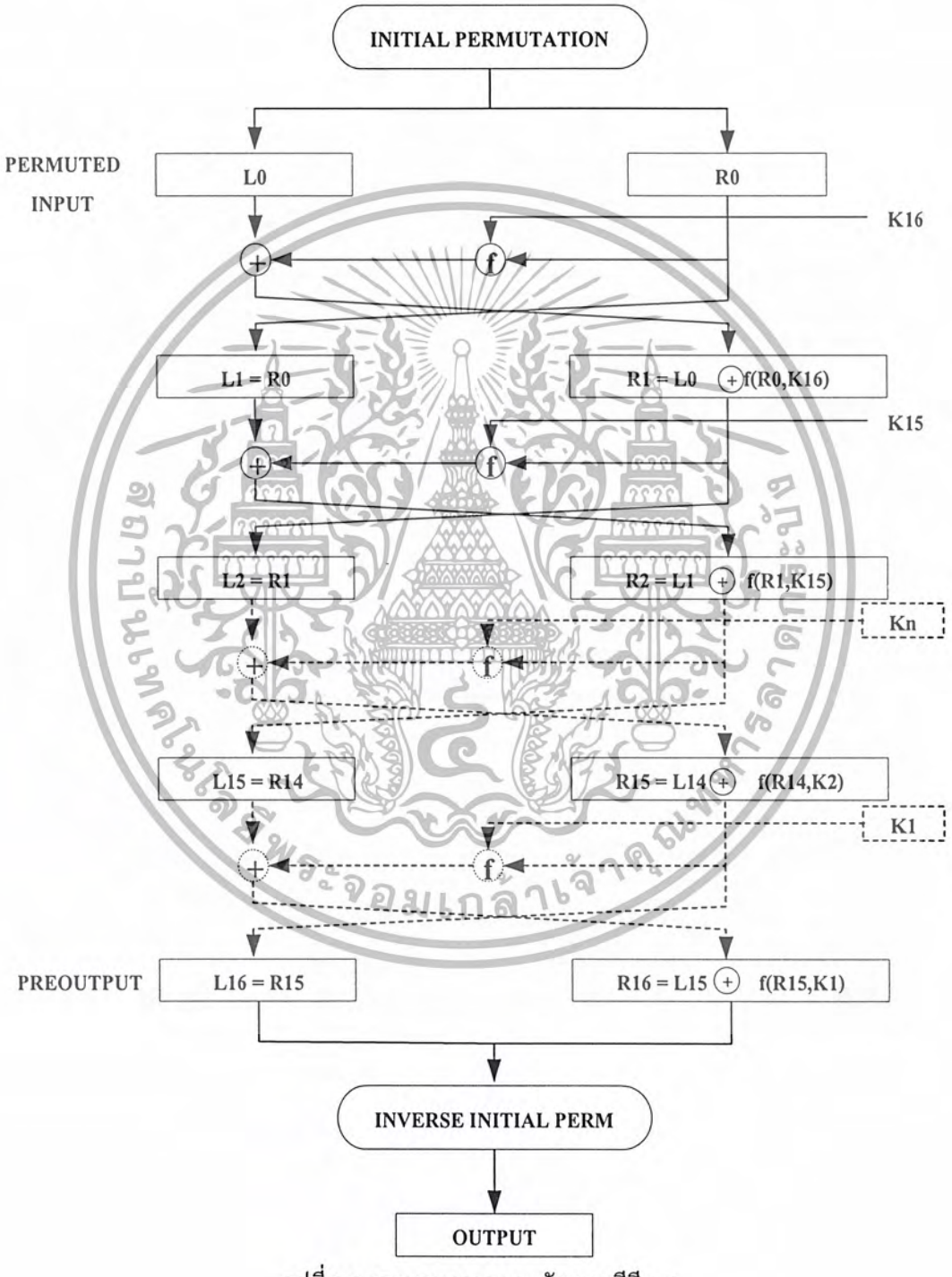
IP^{-1}

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	41	11	51	19	59	27
34	2	40	10	50	18	58	26
33	1	41	9	49	17	57	25

ตารางที่ 3.9 การสลับตำแหน่งตามตารางไอพีอินเวอร์ส

3.3 กระบวนการถอดรหัสแบบดีอีเอส

สำหรับกระบวนการถอดรหัสจะมีลักษณะคล้ายกับขั้นตอนการเข้ารหัสเพียงแต่คีย์ที่ใช้ในแต่ละรอบจะสลับกัน ดังแสดงในรูป 3.4



รูปที่ 3.4 กระบวนการถอดรหัสแบบดีอีเอส

3.4 <u>ตัวอย่าง</u> การเข้ารหัสแบบดีอีเอสโดยให้ P เป็นข้อมูลเพลนเท็กซ์ P = 0123456789ABCDEF เป็นเลขฐานสิบหก โดยสามารถเขียนใหม่ให้อยู่ในรูปของเลขฐานสอง และเราจะได้ข้อมูล 64 บิตบล็อก

P = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

แบ่งออกเป็น **L** และ **R**

L = 0000 0001 0010 0011 0100 0101 0110 0111

R = 1000 1001 1010 1011 1100 1101 1110 1111

สมมติคีย์มีเป็น K = 133457799BBCDFF1 เป็นเลขฐานสิบหก โดยบิตที่ 8,16,24,...,64 จะถูกตัดออก โดยสามารถเขียนได้เป็นเลขฐานสองเป็น

K = 0001 0011 0011 0100 0101 0111 0111 1001 1001 1011 1011 1100 1101 1111 1111 0001

ขั้นตอนการเข้ารหัสดีอีเอส มีวิธีการดังต่อไปนี้

ขั้นตอน 1: สร้าง 16 คีย์จากข้อมูล 48 บิต

โดยนำคีย์ขนาด 64 บิตนั้นมาเรียงตามตาราง พีซี 1 จะได้คีย์ขนาด 56 บิต

K+ = 1111 0000 1100 1100 1010 1010 1111 0101 0101 0110 0110 0111 1000 1111

หลังจากนั้นนำคีย์มาแบ่งออกเป็นสองส่วนคือ C_0 และ D_0 โดยแต่ละส่วนมีขนาด 28 บิต

C_0 = 1111000011001100101010101111

D_0 = 0101010101100110011110001111

หลังจากนั้นนำ C_0 และ D_0 มาทำการเลื่อนข้อมูลไปทางซ้ายเราจะได้ C_n and D_n โดยที่ 1<=n<=16

C_1 = 1110000110011001010101011111

D_1 = 1010101011001100111100011110

C_2 = 1100001100110010101010111111

D_2 = 0101010110011001111000111101

$C_3 = 00001100110010101010111111111$

$D_3 = 01010110011001111000111101 01$

$C_4 = 00110011001010101011111111100$

$D_4 = 01011001100111100011110101 01$

$C_5 = 11001100101010101111111110000$

$D_5 = 01100110011110001111010101 01$

$C_6 = 00110010101010111111111000011$

$D_6 = 10011001111000111101010101 01$

$C_7 = 11001010101011111111100001100$

$D_7 = 01100111100011110101010101 10$

$C_8 = 00101010101111111100001100 11$

$D_8 = 10011110001111010101010110 01$

$C_9 = 01010101011111111000011001 10$

$D_9 = 00111100011110101010101100 11$

$C_{10} = 01010101111111100001100 11001$

$D_{10} = 11110001111010101010110 01100$

$C_{11} = 01010111111110000110011001 01$

$D_{11} = 11000111101010101011001 10011$

$C_{12} = 01011111111000011001100 10101$

$D_{12} = 00011110101010101100 11001111$

$C_{13} = 01111111100001100110010 10101$

$D_{13} = 01111010101010110011001 11100$

C_{14} = 11111110000110011001010101010

D_{14} = 11101010101011001100111110001

C_{15} = 11111000011001100101010101111

D_{15} = 10101010101100110011111000111

C_{16} = 11111000011001100101010101111

D_{16} = 01010101011001100111100011111

นำค่า C_nD_n ที่ได้มาทำการสลับตำแหน่งตามตารางพีซี 2 จะได้คีย์ดังต่อไปนี้

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

K_2 = 011110 011010 111011 011001 110110 111100 100111 100101

K_3 = 010101 011111 110010 001010 010000 101100 111110 011001

K_4 = 011100 101010 110111 010110 110110 110011 010100 011101

K_5 = 011111 001110 110000 000111 111010 110101 001110 101000

K_6 = 011000 111010 010100 111110 010100 000111 101100 101111

K_7 = 111011 001000 010010 110111 111101 100001 100010 111100

K_8 = 111101 111000 101000 111010 110000 010011 101111 111011

K_9 = 111000 001101 101111 101011 111011 011110 011110 000001

K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111

K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110

K_{12} = 011101 010111 000111 110101 100101 000110 011111 101001

K_{13} = 100101 111100 010111 010001 111110 101011 101001 000001

K_{14} = 010111 110100 001110 110111 111100 101110 011100 111010

K_{15} = 101111 111001 000110 001101 001111 010011 111100 001010

K_{16} = 110010 110011 110110 001011 000011 100001 011111 110101

ขั้นตอน 2: เข้ารหัส 64-บิต ของแต่ละบล็อกข้อมูล

โดยนำข้อมูลเพลนเท็กซ์ขนาด 64 บิตมาทำการสลับตำแหน่งตามตารางไอพี เช่น

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

จะได้ข้อมูลที่ผ่านการสลับตำแหน่งเป็น

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

หลังจากนั้นนำข้อมูลที่ได้มาแบ่งเป็นสองส่วนคือ L_0และ R_0 ซึ่งแต่ละส่วนมีขนาด 32 บิต

จาก **IP** ข้างบนเราจะได้ L_0 และ R_0 ดังนี้

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

หลังจากนั้นนำ L_0 และ R_0 มาทำการหา L_n และ R_n จาก

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

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

โดยที่ 1<=n<=16

เช่น n = 1 จะได้

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010

$R_1 = L_0 + f(R_0, K_1)$

ซึ่งเราสามารถหาไซเฟอร์ฟังก์ชันได้โดยนำค่า R มาทำการสลับตำแหน่งตามตารางอี เช่น

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

เมื่อเราสลับตำแหน่งแล้วเราจะได้

E(R_0) = 011110 100001 010101 010101 011110 100001 010101 010101

แล้วนำ $\mathbf{E}(R_{n-1})$ มา เอ็กคลูซีพออ กับ K_n ดังนี้

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$

$$\mathbf{E}\ (R_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$$

$$K_1 + \mathbf{E}\ (R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111$$

หลังจากนั้นนำข้อมูลมาผ่าน เอสบ็อกซ์ ทีละ 6 บิต 8 ครั้ง ซึ่งเราจะได้

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$

แล้วนำข้อมูลที่ได้มาสลับตำแหน่งตามตารางพี จะได้

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

ดังนั้น

$$R_1 = L_0 + f(R_0, K_1)$$

$$= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$$

$$+ 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$$= 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100$$

ในรอบต่อไปเราจะได้ $L_2 = R_1$ และ $R_2 = L_1 + f(R_1, K_2)$ และเมื่อผ่านไป 16 รอบแล้วเราจะได้ L_{16} และ R_{16} ขนาด 64 บิตคือ

$$L_{16} = 0100\ 0011\ 0100\ 0010\ 0011\ 0010\ 0011\ 0100$$

$$R_{16} = 0000\ 1010\ 0100\ 1100\ 1101\ 1001\ 1001\ 0101$$

แล้วนำ L_{16} และ R_{16} มาทำการสลับตำแหน่งตามตาราง IP^{-1} จะได้

$$IP^{-1} = 10000101\ 11101000\ 00010011\ 01010100\ 00001111\ 00001010\ 10110100\ 00000101$$

ซึ่งคือ 85E813540F0AB405 ในเลขฐานสิบหก

สรุปคือเมื่อนำข้อมูล **P** = 0123456789ABCDEF มาทำการเข้ารหัสแบบดีอีเอสแล้วจะได้ข้อมูลไซเฟอร์เท็กซ์หรือข้อมูลที่เข้ารหัสแล้วเป็น **CT** = 85E813540F0AB405 ส่วนการถอดรหัสนั้นทำเหมือนกับการเข้ารหัสโดยใช้ข้อมูลอินพุตคือ **CT** = 85E813540F0AB405 และเริ่มการคำนวณที่ K16 ก่อนไปจนถึง K_1 แล้วเราจะได้ข้อมูลเพลนเท็กซ์กลับมาซึ่งก็คือ **P** = 0123456789ABCDEF

3.5 การคำนวณการเข้ารหัสแบบทริปเปิลดีอีเอส (3DES)

ให้ $E_K(I)$ และ $D_K(I)$ แทนการเข้ารหัสและถอดรหัสข้อมูลด้วยดีอีเอสตามลำดับโดยใช้คีย์ K ของดีอีเอสตามลำดับ กระบวนการเข้าและถอดรหัสของ ทริปเปิลดีอีเอส ประกอบขึ้นจากกระบวนการเข้าและถอดรหัสของ ดีอีเอส หลายขั้นตอน ดังต่อไปนี้

3.5.1 การเข้ารหัสของ ทริปเปิลดีอีเอส โดยแปลงบล็อก P (Plain Text) ขนาด 64 บิต ไปเป็นบล็อก C (Cipher Text) ขนาด 64 บิต นิยามดังต่อไปนี้

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$

รูปที่ 3.5 กระบวนการเข้ารหัส ทริปเปิล ดีอีเอส

3.5.2 การถอดรหัสของทริปเปิลดีอีเอส โดยแปลงบล็อก C (Cipher Text) ไปเป็นบล็อก P (Plain Text) ขนาด 64 บิต นิยามดังต่อไปนี้

$$P = D_{K1}(E_{K2}(D_{K3}(C)))$$

รูปที่ 3.6 กระบวนการถอดรหัส ทริปเปิล ดีอีเอส

มาตรฐานระบุตัวเลือกการปรับแต่งสำหรับกลุ่มคีย์ (K_1,K_2,K_3) ดังต่อไปนี้

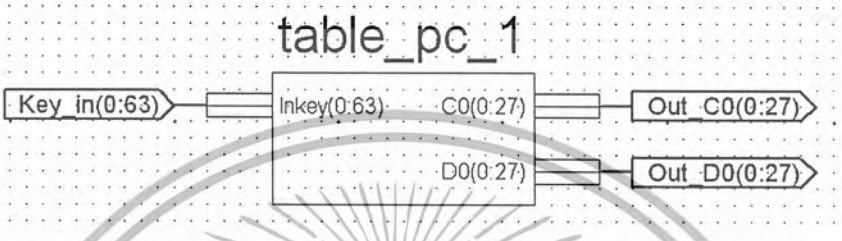
1. การปรับแต่งค่าแบบที่ 1 : K_1,K_2 และ K_3 เป็นคีย์อิสระ (Independent Key)
2. การปรับแต่งค่าแบบที่ 2 : K_1 และ K_2 เป็นคีย์อิสระและ $K_3 = K_1$
3. การปรับแต่งค่าแบบที่ 3 : $K_1 = K_2 = K_3$

บทที่ 4
การทดลองและผลการทดลอง

4.1 ส่วนการคำนวนรหัสกุญแจ

4.1.1 ส่วนการคำนวนหาค่ารหัสกุญแจ

ตารางพีซี 1 (PC-1 Table)

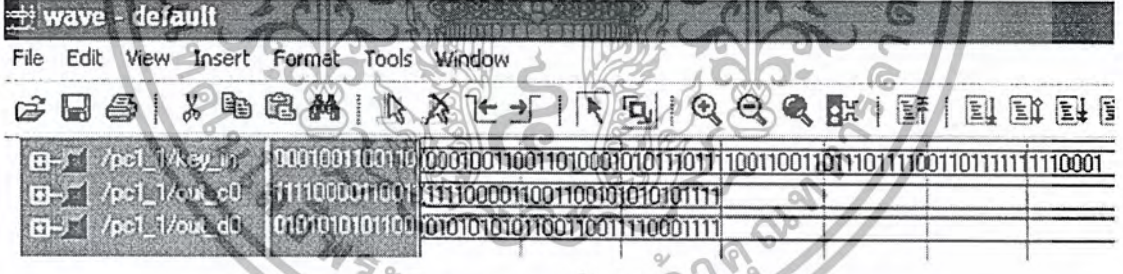


รูปที่ 4.1 ตารางพีซี1

รหัสกุญแจ 0001001100110100010101110111100110011011101111001101111111111110001

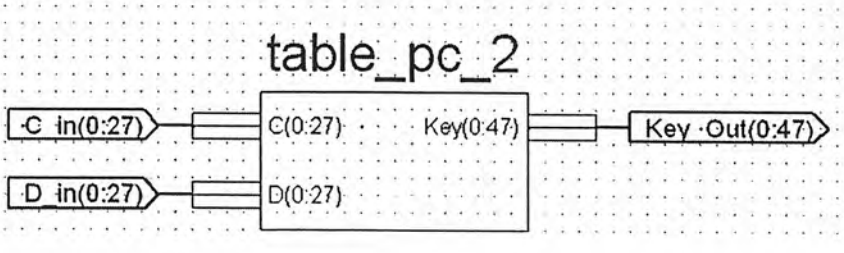
เอาท์พุท C0 11110000110011000101010101111

เอาท์พุท D0 01010101011001100111100011111



รูปที่ 4.2 ผลการทดลองตารางพีซี1

ตารางพีซี 2 (PC-2 Table)



รูปที่ 4.3 ตาราง พีซี 2

4.1.2 การคำนวนรหัสกุญแจ

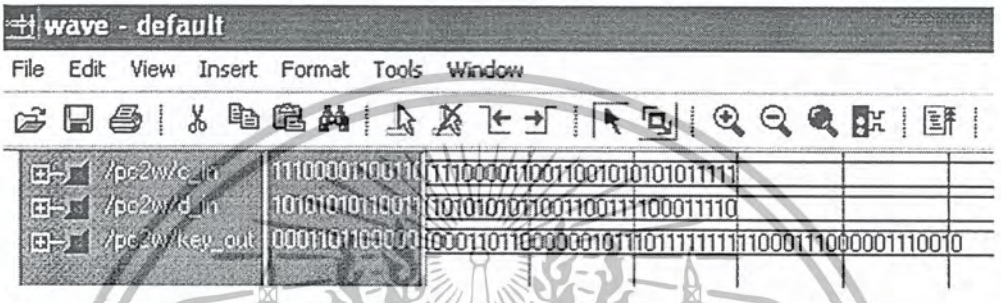
จะมีการคำนวนทั้งหมด 16 รอบด้วยกัน โดยจะได้ผลลัพธ์เป็นรหัสกุญแจทั้งหมด 16 ค่า

รอบที่1

อินพุท C1 11100001100110010101010111111

อินพุท D1 10101010110011001111100011110

กุญแจ 1 000110110000000101110111111111100011100000111001O



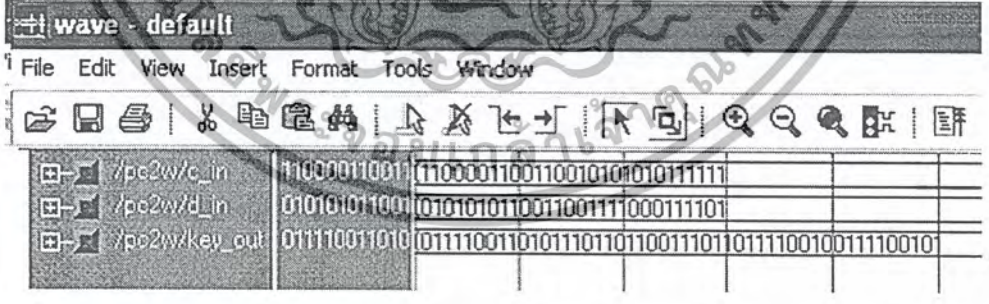
รูปที่ 4.4 ผลการทดลองตาราง พีซี2 รอบที่ 1

รอบที่2

อินพุท C2 11000011001100101010101111111

อินพุท D2 01010101100110011110000111101

กุญแจ 2 011110011010111011011001110110111100100111100101



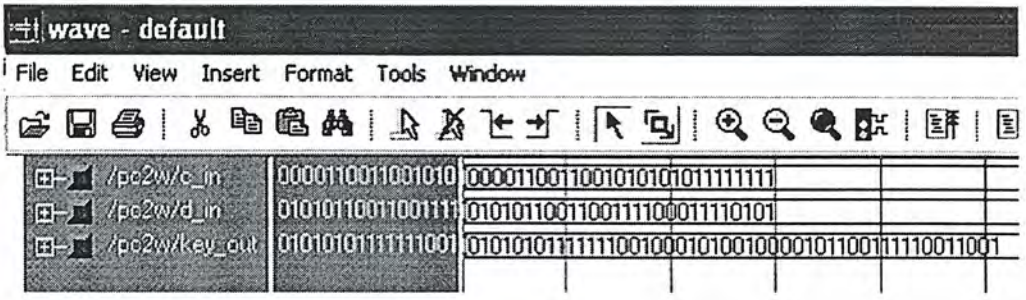
รูปที่ 4.5 ผลการทดลองตาราง พีซี2 รอบที่ 2

รอบที่3

อินพุท C3 00001100110010101010111111111

อินพุท D3 01010110011001111000111101O1

กุญแจ 3 010101011111111001000101001000001011001111100110O1



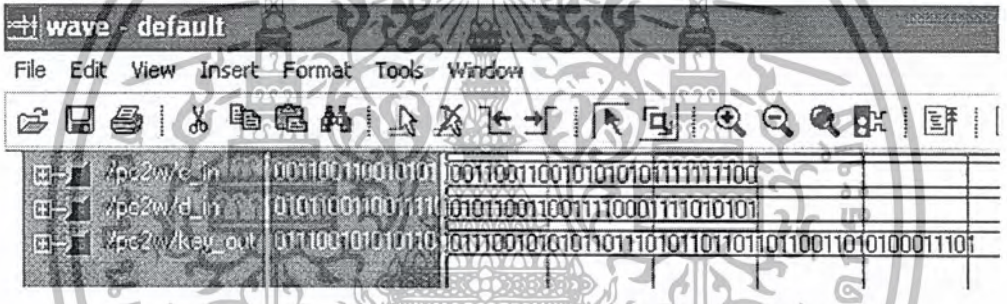
รูปที่ 4.6 ผลการทดลองตาราง พีซี2 รอบที่ 3

รอบที่4

อินพุท C4 00110011001010101011111111100

อินพุท D4 01011001100111100011110101010

กุญแจ 4 011100101010110111010110110110110011010100011101



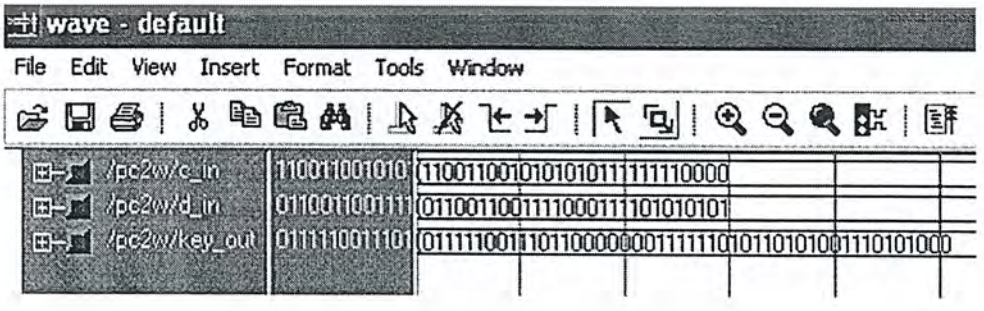
รูปที่ 4.7 ผลการทดลองตาราง พีซี2 รอบที่ 4

รอบที่5

อินพุท C5 11001100101010101111111110000

อินพุท D5 01100110011110001111010101010

กุญแจ 5 011111001110110000000111111010110101001110101000



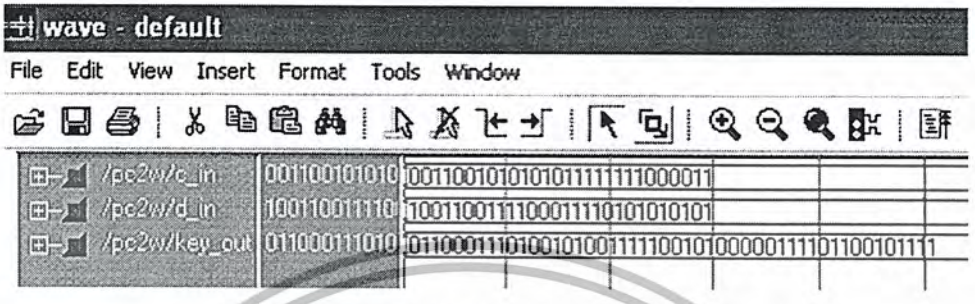
รูปที่ 4.8 ผลการทดลองตาราง พีซี2 รอบที่ 5

รอบที่6

อินพุท C6 00110010101010111111111000011

อินพุท D6 10011001111000111101010101010

กุญแจ 6 01100011101001010011111001010000011111011001011111



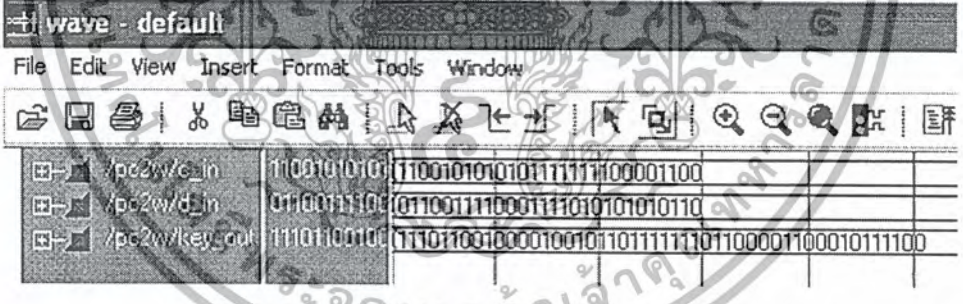
รูปที่ 4.9 ผลการทดลองตาราง พีซี2 รอบที่ 6

รอบที่7

อินพุท C7 11001010101011111111100001100

อินพุท D7 01100111100011110101010101110

กุญแจ 7 11101100100000100101101111111101100001100010111100



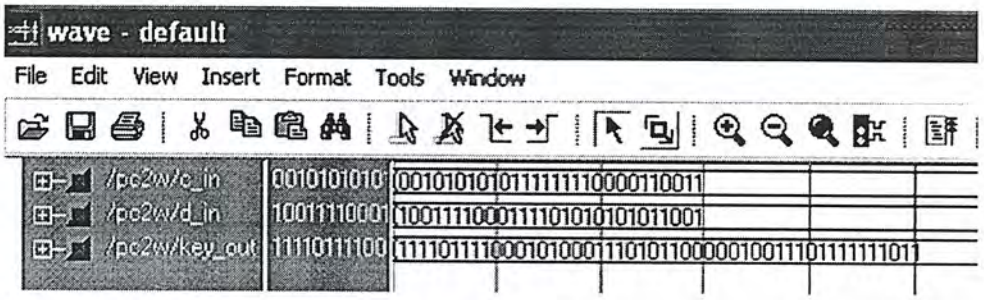
รูปที่ 4.10 ผลการทดลองตาราง พีซี2 รอบที่ 7

รอบที่8

อินพุท C8 00101010101111111110000110011

อินพุท D8 10011110001111010101010011001

กุญแจ 8 11110111100010100011101011000001001110111111111011



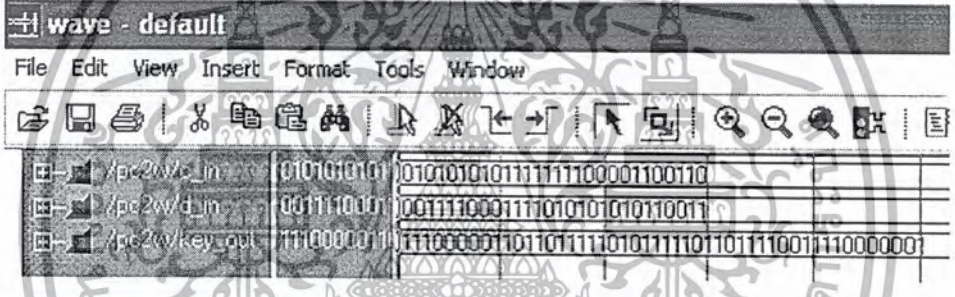
รูปที่ 4.11 ผลการทดลองตาราง พีซี2 รอบที่ 8

รอบที่9

อินพุท C9 010101010111111111000011001l0

อินพุท D9 00111100011110101010110011

กุญแจ 9 1110000011011011111010111111011011111001111000000l



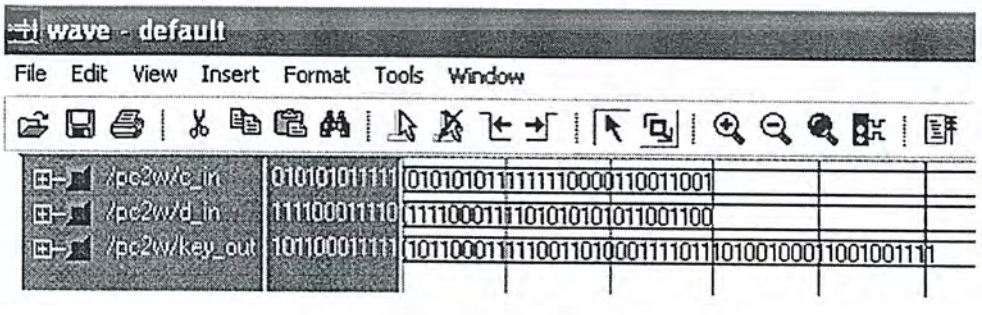
รูปที่ 4.12 ผลการทดลองตาราง พีซี2 รอบที่ 9

รอบที่10

อินพุท C10 0101010111111110000110011001

อินพุท D10 11110001111010101011001100

กุญแจ 10 1011000111110011010001111011101001000110010011111



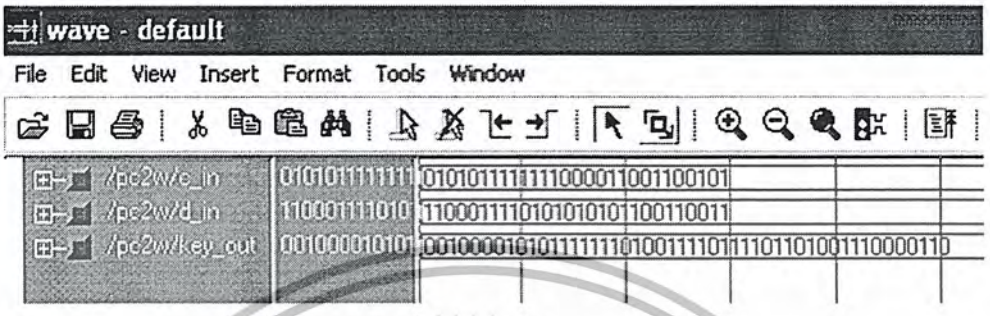
รูปที่ 4.13 ผลการทดลองตาราง พีซี2 รอบที่ 10

รอบที่11

อินพุท C11 0101011111111000011001100101

อินพุท D11 1100011110101010101100110011

กุญแจ 11 00100001010111111101001111011110110100111000011 0



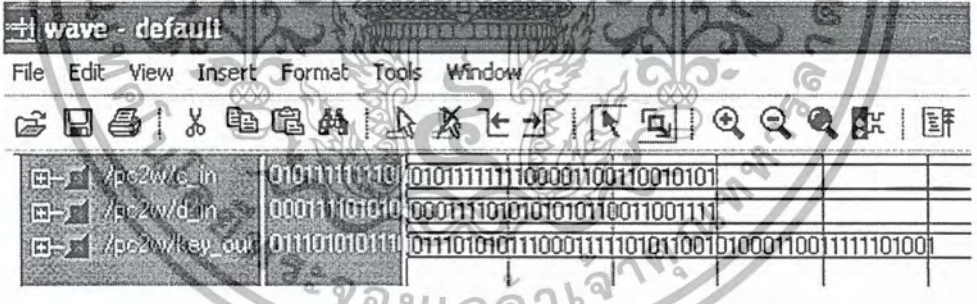
รูปที่ 4.14 ผลการทดลองตาราง พีซี2 รอบที่ 11

รอบที่12

อินพุท C12 0101111111100001100110010101

อินพุท D12 0001111010101010110011001111

กุญแจ 12 0111010101110001111110101100101000110011111101001



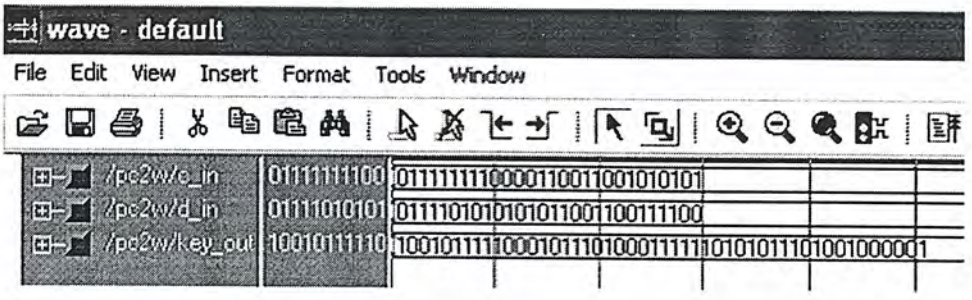
รูปที่ 4.15 ผลการทดลองตาราง พีซี2 รอบที่ 12

รอบที่13

อินพุท C13 0111111110000110011001010101

อินพุท D13 0111101010101011001100111100

กุญแจ 13 1001011111000101110100011111110101011101001000001



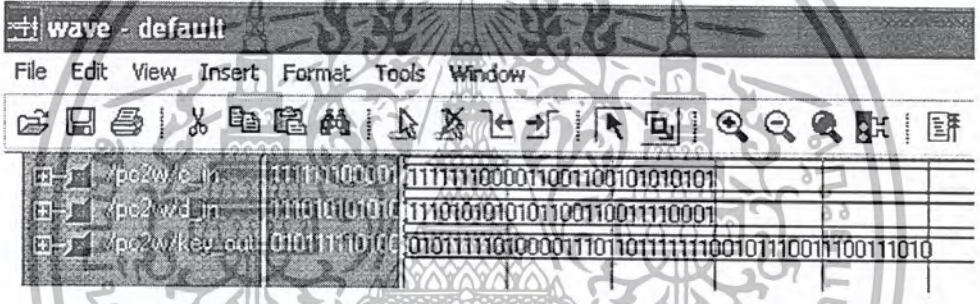
รูปที่ 4.16 ผลการทดลองตาราง พีซี2 รอบที่ 13

รอบที่14

อินพุท C14 11111110000110011001010101010

อินพุท D14 11101010101011001100111110001

กุญแจ 14 01011111010000111011011111111001011100111001110 10



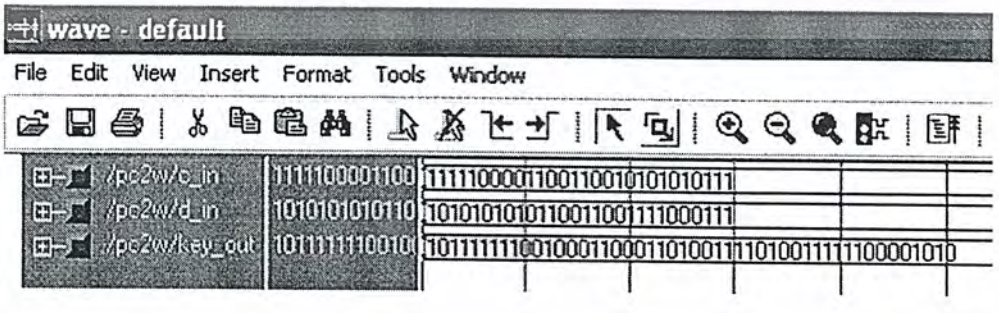
รูปที่ 4.17 ผลการทดลองตาราง พีซี2 รอบที่ 14

รอบที่15

อินพุท C15 11111000011001100101010101011 1

อินพุท D15 10101010101100110011111000111

กุญแจ 15 10111111100100011000110100111101001111110000101 0



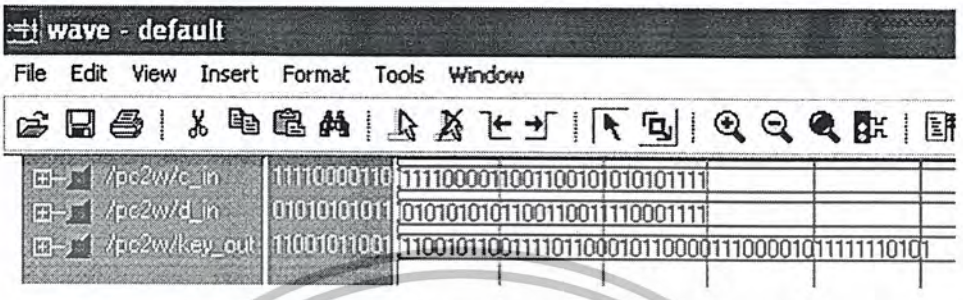
รูปที่ 4.18 ผลการทดลองตาราง พีซี2 รอบที่ 15

รอบที่16

อินพุท C16 11110000110011001010101011111

อินพุท D16 01010101011001100111100011111

กุญแจ 16 1100101100111101100010110000111000010111111110101



รูปที่ 4.19 ผลการทดลองตาราง พีซี2 รอบที่ 16

4.2 ส่วนการเข้ารหัสข้อมูล

4.2.1 ตารางไอพี (IP Table)

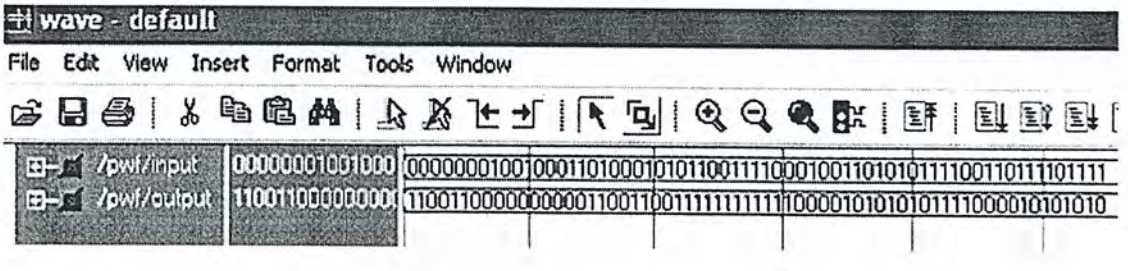
เป็นส่วนที่ใช้สำหรับสลับตำแหน่งของบิตข้อมูล

ตารางไอพี

รูปที่ 4.20 ตาราง ไอพี

อินพุท 0000000100100011010001010110011111000100110101011110011010111101111

เอาท์พุท 1100110000000000110011001111111111111000010101010111110000101010101

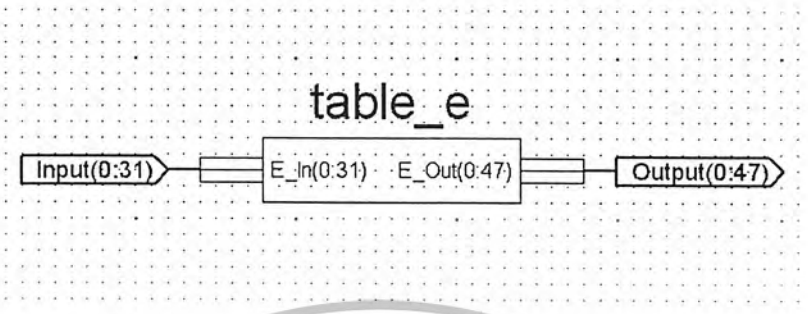


รูปที่ 4.21 ผลการทดลองตาราง ไอพี

4.2.2 ไซเฟอร์ ฟังก์ชั่น (Cypher Function)

ประกอบไปด้วยฟังชันที่สำคัญต่างๆ ดังนี้

ตารางอี (E Table)



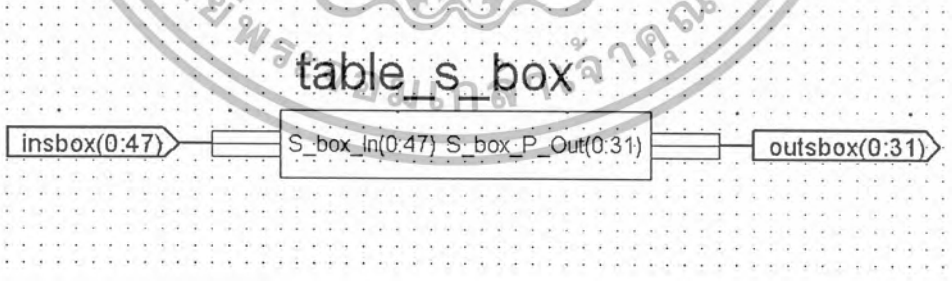
รูปที่ 4.22 ตารางอี

อินพุท 1111 0000 1010 1010 1111 0000 1010 1010

เอาท์พุท 011110 100001 010101 010101 011110 100001 010101 010101

รูปที่ 4.23 ผลการทดลองตารางอี

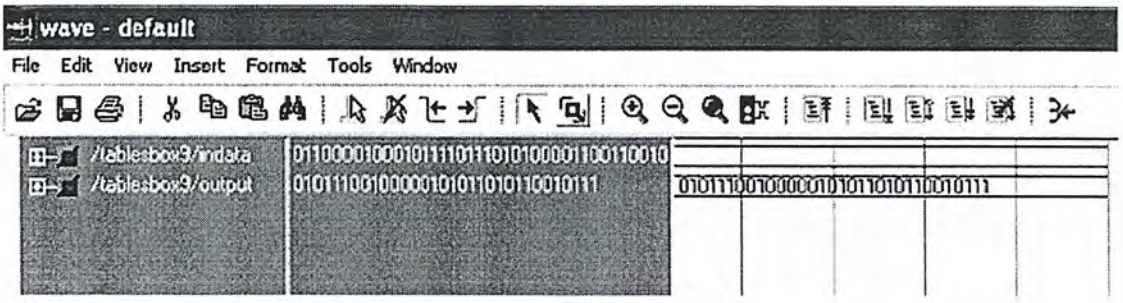
เอส บ็อกซ์ (S-Box)



รูปที่ 4.24 ตารางเอสบ็อกซ์

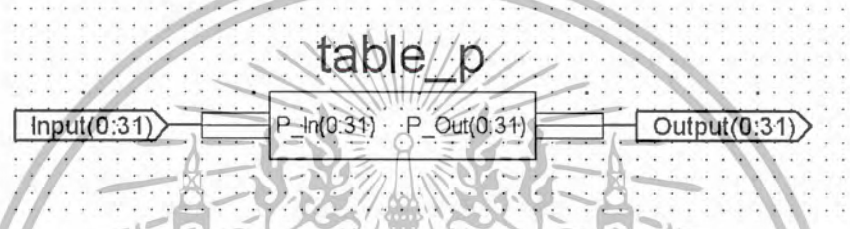
อินพุท 011000 010001 011110 111010 100001 100110 010100 100111

เอาท์พุท 0101 1100 1000 0010 1011 0101 1001 0111



รูปที่ 4.25 ผลการทดลองตารางเอสบ็อกซ์

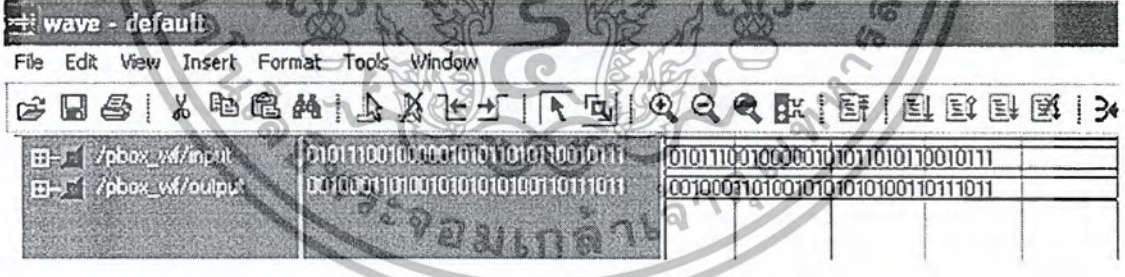
ตารางพี (P Table)



รูปที่ 4.26 ตารางพี

อินพุท 0101 1100 1000 0010 1011 0101 1001 0111

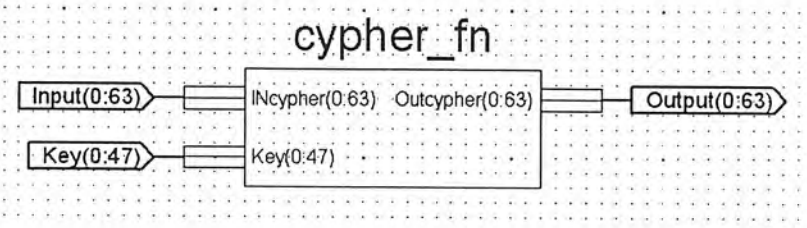
เอาท์พุท 0010 0011 0100 1010 1010 1001 1011 1011



รูปที่ 4.27 ผลการทดลองตารางพี

เมื่อนำส่วนต่างๆ ของไซเฟอร์ ฟังก์ชั่น มารวมกันจะได้ผลการทดลองดังต่อไปนี้ โดยในการเข้ารหัสข้อมูลจะมีการนำข้อมูลและรหัสกุญแจมาเข้า ไซเฟอร์ ฟังก์ชั่นทั้งหมด 16 รอบ

รวมไซเฟอร์ ฟังก์ชั่น



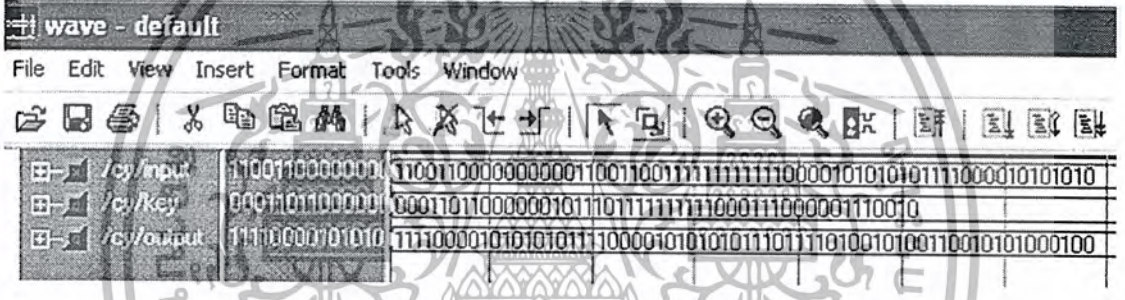
รูปที่ 4.28 ไซเฟอร์ ฟังก์ชั่น

รอบที่ 1

กุญแจ 00011011000000010111011111111111000111000001110010

อินพุท 11001100000000001100110011111111111100001010101011110000101010101

เอาท์พุท 11110000101010101111000010101010111011110100101001100101010000100



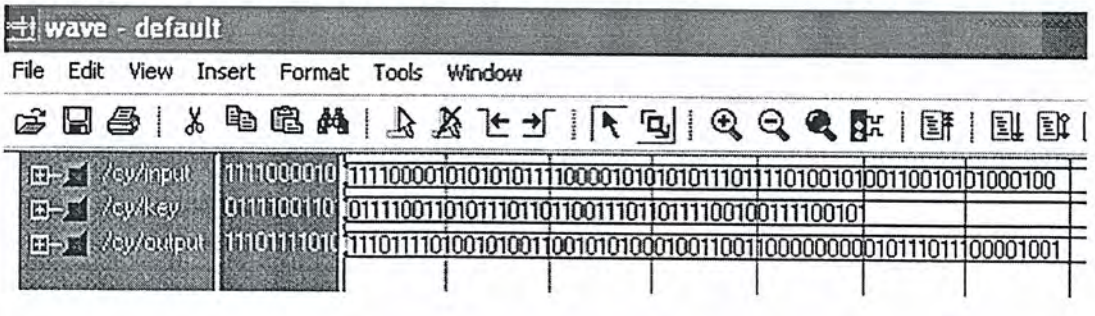
รูปที่ 4.29 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 1

รอบที่ 2

กุญแจ 01111001101011101101100111011011110010011110010 1

อินพุท 11110000101010101111000010101010111011110100101001100101010000100

เอาท์พุท 11101111010010100110010101000100110011000000000101110111000010 01



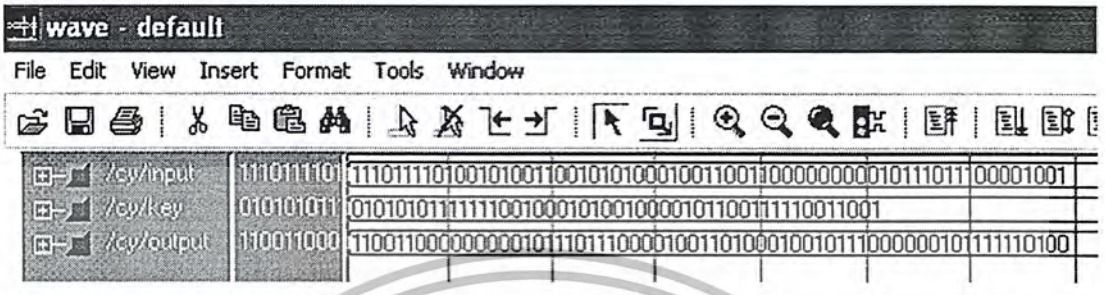
รูปที่ 4.30 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 2

รอบที่ 3

กุญแจ 01010101111111001000101001000010110011111100110 01

อินพุท 11101111010010100110010101000100110011000000000101110111000010 01

เอาท์พุท 11001100000000001011101110000100110100010010111000000101111110100



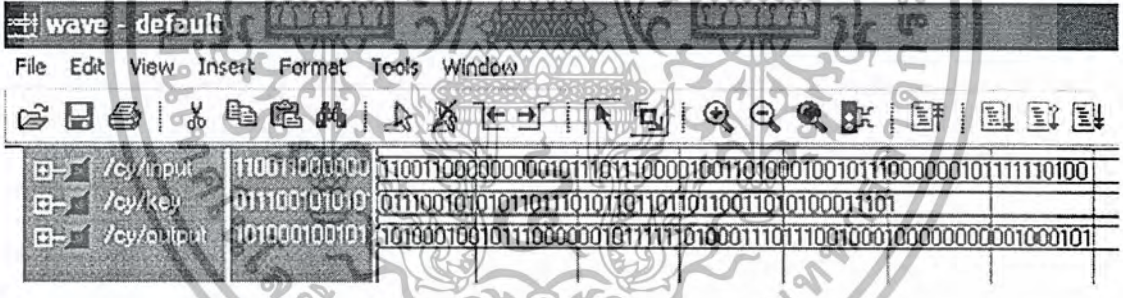
รูปที่ 4.31 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 3

รอบที่ 4

กุญแจ 01110010101011011101011011011011001101010000111 01

อินพุท 11001100000000001011101110000100110100010010111000000101111110100

เอาท์พุท 10100010010111000000101111110100 01110111001000100000000001000101



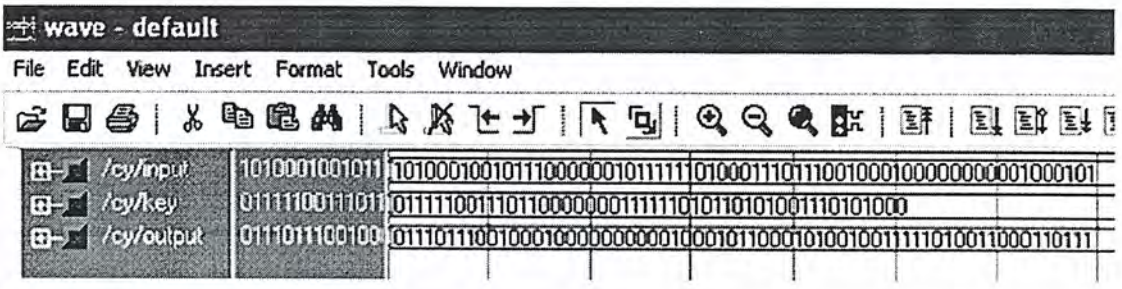
รูปที่ 4.32 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 4

รอบที่ 5

กุญแจ 01111100111011000000011111101011010100111010 1000

อินพุท 10100010010111000000101111110100 01110111001000100000000001000101

เอาท์พุท 01110111001000100000000001000101 10001010010011111010011000110111



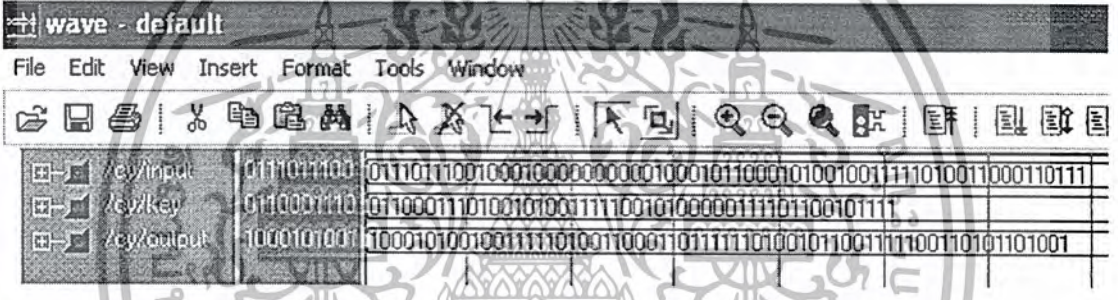
รูปที่ 4.33 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 5

รอบที่ 6

กุญแจ 01100011101001010011111001010000011110110010111 1

อินพุท 01110111001000100000000001000101 10001010010011111010011000110111

เอาท์พุท 10001010010011111010011000110111 11101001011001111100110101101001



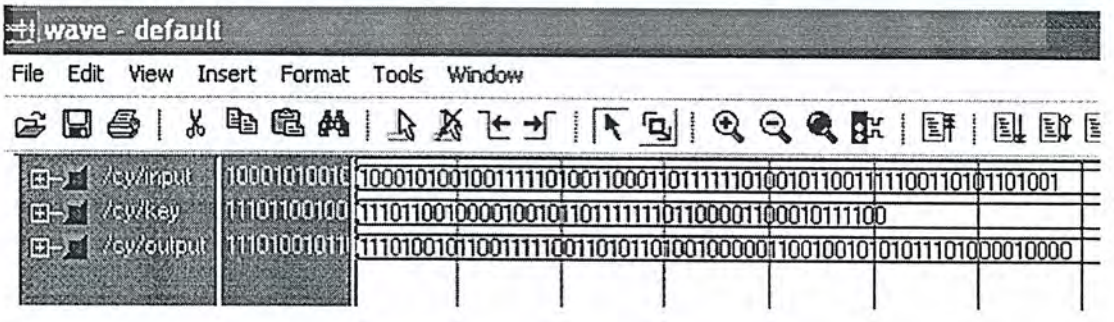
รูปที่ 4.34 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 6

รอบที่ 7

กุญแจ 11101100100001001011011111111011000011000101111 00

อินพุท 10001010010011111010011000110111 11101001011001111100110101101001

เอาท์พุท 11101001011001111100110101101001 00000110010010101011101000010000



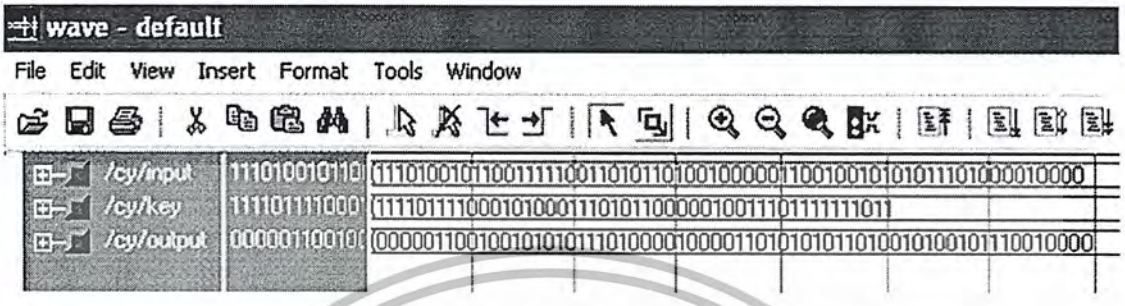
รูปที่ 4.35 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 7

รอบที่ 8

กุญแจ 11110111100010100011101011000001001110111111110 11

อินพุท 11101001011001111100110101101001 00000110010010101011101000010000

เอาท์พุท 00000110010010101011101000010000 11010101011010010100101110010000



รูปที่ 4.36 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 8

รอบที่ 9

กุญแจ 11100000110110111110101111101101111001111000000 1

อินพุท 00000110010010101011101000010000 11010101011010010100101110010000

เอาท์พุท 11010101011010010100101110010000 00100100011111100110001100111101 0

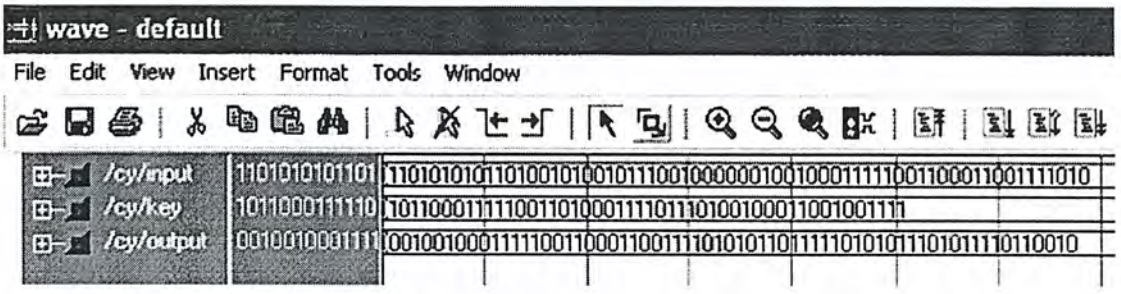
รูปที่ 4.37 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 9

รอบที่ 10

กุญแจ 10110001111100110100011110111010010001100100111 1

อินพุท 11010101011010010100101110010000 00100100011111100110001100111101 0

เอาท์พุท 00100100011111100110001100111101 0 10110111110101011101011110110010



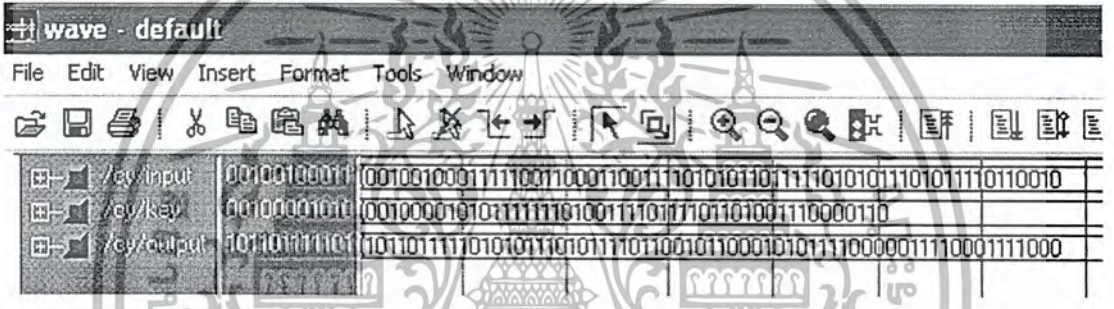
รูปที่ 4.38 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 10

รอบที่ 11

กุญแจ 00100001010111111101001111011110110100111000011 0

อินพุท 00100100011111001100011001111010 10110111110101011101011110110010

เอาท์พุท 10110111110101011101011110110010 11000101011110000011110001111000



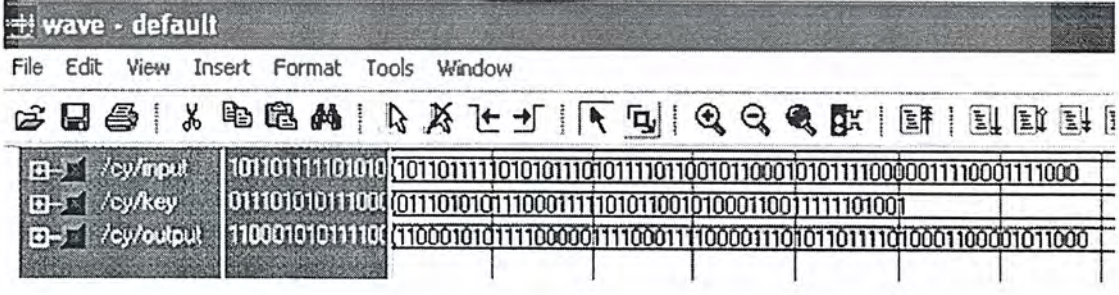
รูปที่ 4.39 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 11

รอบที่ 12

กุญแจ 01110101011100011111010110010100011001111111010 01

อินพุท 10110111110101011101011110110010 11000101011110000011110001111000

เอาท์พุท 11000101011110000011110001111000 01110101101111010001100001011000



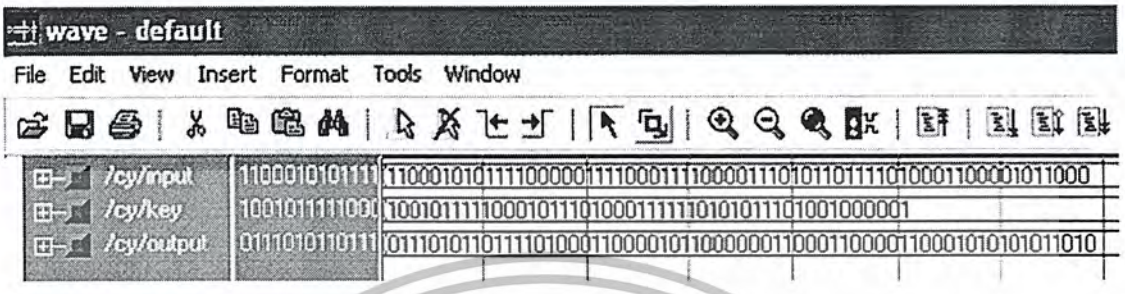
รูปที่ 4.40 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 12

รอบที่ 13

กุญแจ 10010111110001011101000111111010101110100100000 1

อินพุท 11000101011110000011110001111000 01110101101111010001100001011000

เอาท์พุท 01110101101111010001100001011000 00011000110000110001010101011010



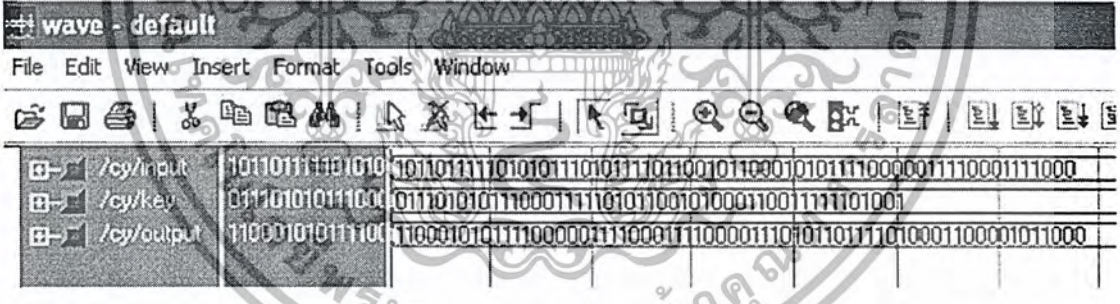
รูปที่ 4.41 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 13

รอบที่ 14

กุญแจ 01011111010000111011011111110010111001110011 1010

อินพุท 01110101101111010001100001011000000110001100001100010101010011010

เอาท์พุท 00011000110000110001010101011010 11000010100011001001011000001101



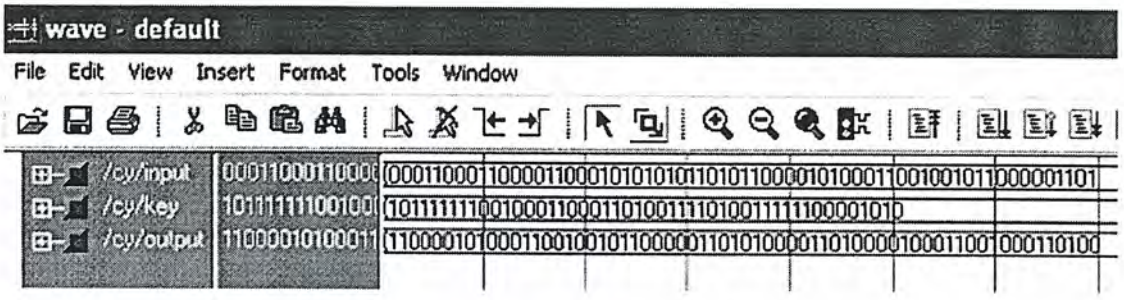
รูปที่ 4.42 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 14

รอบที่ 15

กุญแจ 10111111100100011000110100111101001111110000 1010

อินพุท 00011000110000110001010101011010 11000010100011001001011000001101

เอาท์พุท 11000010100011001001011000001101 01000011010000100011001000110100



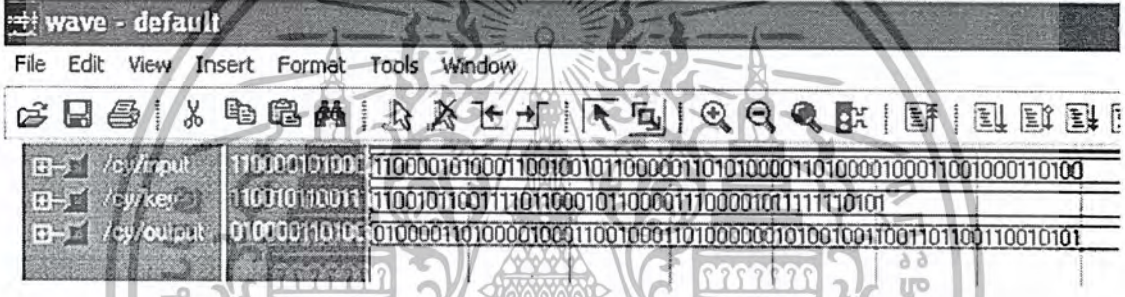
รูปที่ 4.43 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 15

รอบที่ 16

กุญแจ 1100101100111101100010110000111000010111111110101

อินพุท 1100001010001100100101100000 1101 0100001101000010001100100011 0100

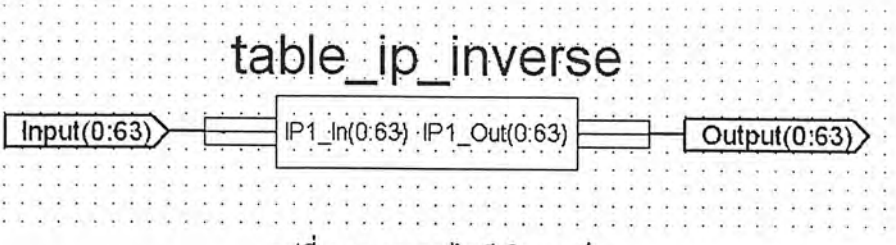
เอาท์พุท 0100001101000010001100100011 0100 0000101001001100110110011001 0101



รูปที่ 4.44 ผลการทดลอง ไซเฟอร์ ฟังก์ชั่น รอบที่ 16

หลังจากที่นำข้อมูลและรหัสกุญแจมาเข้า ไซเฟอร์ ฟังก์ชั่น 16 รอบแล้ว จะนำผลลัพธ์ที่ได้มาเข้าตาราง ไอพี อินเวอร์ส เพื่อทำการสลับบิตของข้อมูลอีกครั้งหนึ่ง โดยผลลัพธ์ที่ได้ จะเรียกว่า ไซเฟอร์เท็กส์ (Cypher Text)

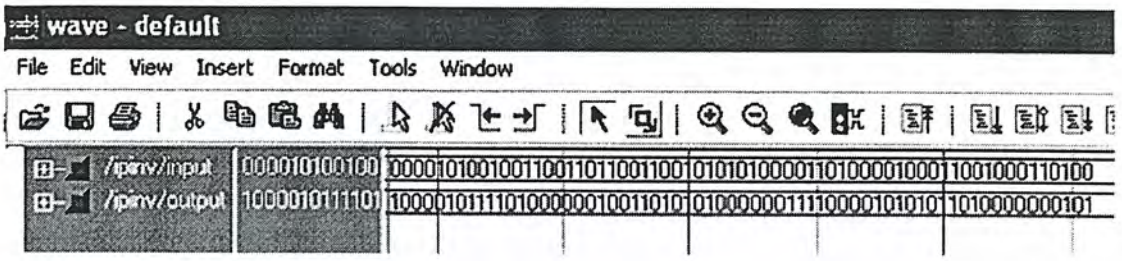
4.2.3 ตารางไอพี อินเวอร์ส (IP Inverse)



รูปที่ 4.45 ตาราง ไอพี อินเวอร์ส

อินพุท 0000101001001100110110011001010101000011010000100011001000110100

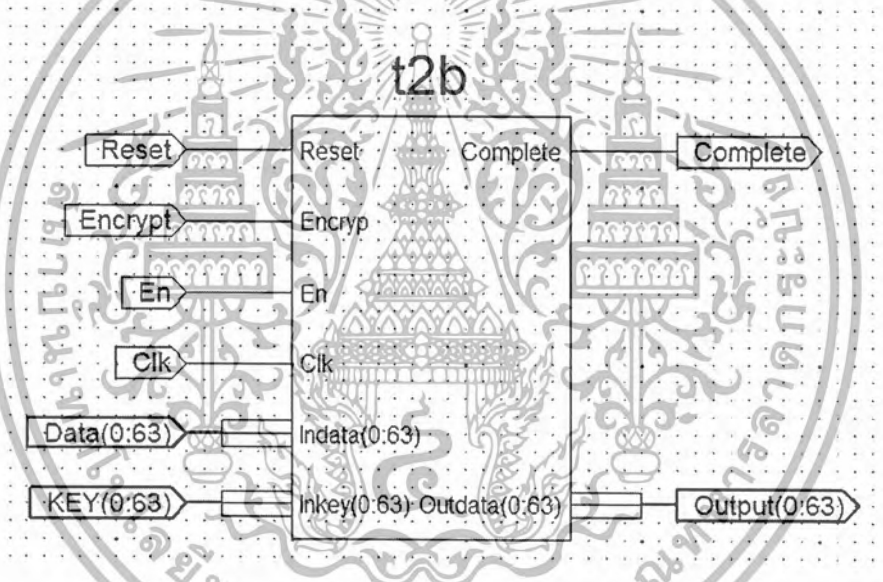
เอาท์พุท 1000010111101000000100110101010000000111100001010101101000000000101



รูปที่ 4.46 ผลการทดลองตาราง ไอพี อินเวอร์ส

เมื่อนำฟังชันต่างๆมารวมกันจะเป็นการเข้ารหัสข้อมูลแบบ ดีอีเอส โดยจะมี รูปวงจรรวม ของการเข้ารหัส แบบ ดีอีเอส เป็นดังนี้

4.2.4 การเข้ารหัสแบบดีอีเอส

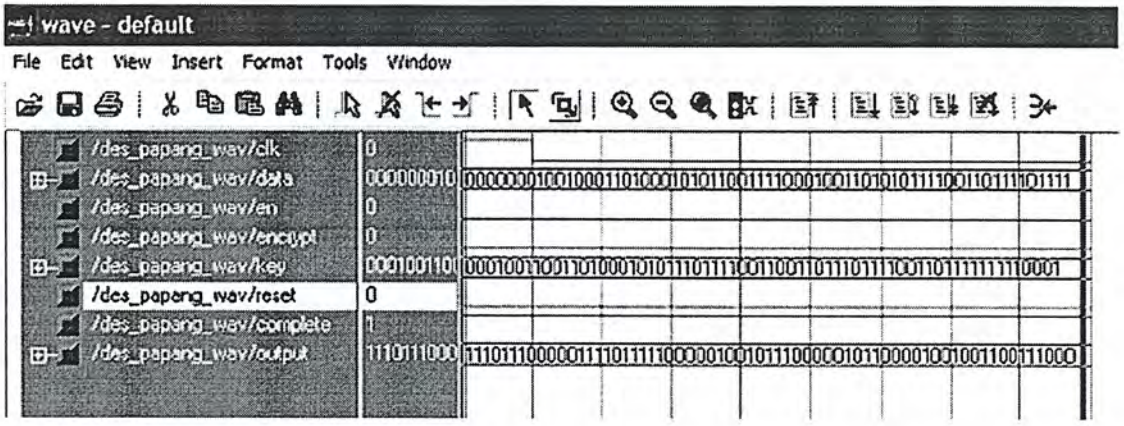


รูปที่ 4.47 วงจร รวมของ ดีอีเอส

อินพุท 0000000100100011010001010110011111000100110101011110011011110111111

กุญแจ 0001001100110100010101110111100110011011101111001101111111111110001

เอาท์พุท 1000010111101000000100110101010000001111000010101011010000000101

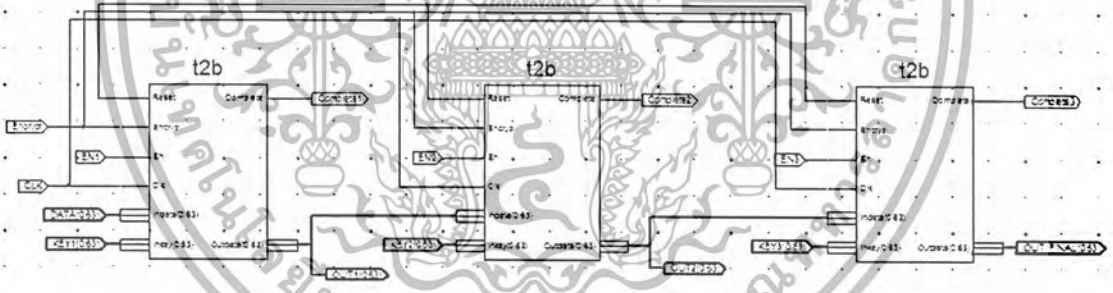


รูปที่ 4.48 ผลการทดลอง ดีอีเอส

ในการทำ ทริปเปิล ดีอีเอส (Triple DES) คือการนำเอาข้อมูลมาเข้ารหัสแบบ ดีอีเอส 3 รอบ โดยจะสามารทำการเข้ารหัสโดยใช้รหัสกุญแจที่เหมือนกันหรือแตกต่างกันก็ได้

4.3 การเข้ารหัสแบบ ทริปเปิล ดีอีเอส

4.3.1แบบรหัสกุญแจทั้ง 3 ตัวเป็นตัวเดียวกัน



รูปที่ 4.49 ทริปเปิล ดีอีเอส

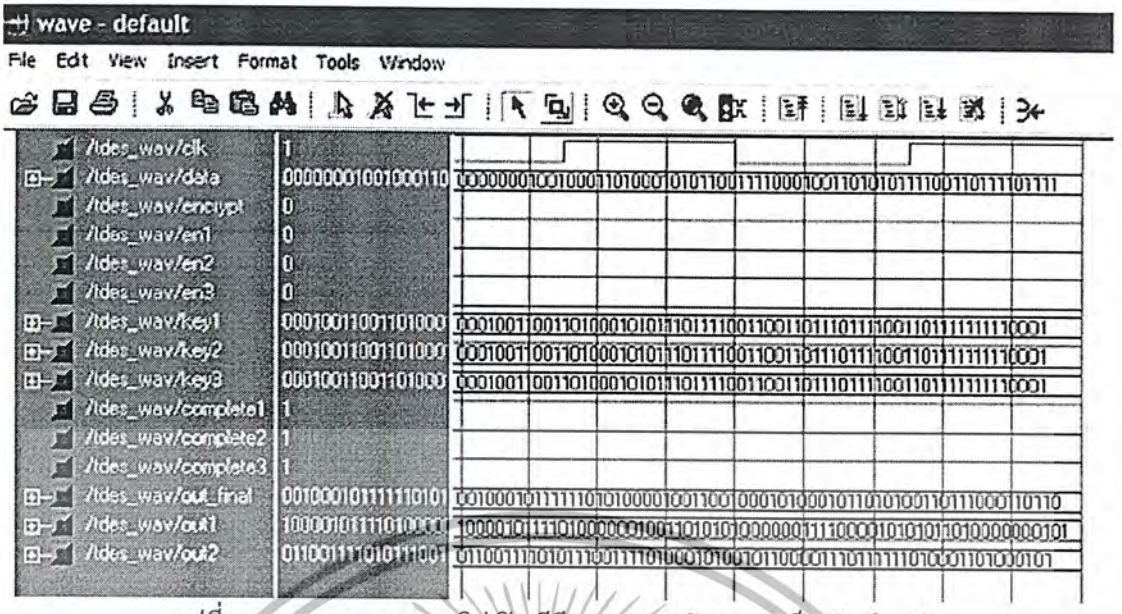
อินพุท 0000000100100011010001010110011110001001101010111100110111101111

กุญแจ1 0001001100110100010101110111100110011011101111001101111111110001

กุญแจ2 0001001100110100010101110111100110011011101111001101111111110001

กุญแจ3 0001001100110100010101110111100110011011101111001101111111110001

เอาท์พุท 0010001011111110101000010011001000101000101101010011011100011 0110



รูปที่ 4.50 ผลการทดลอง ทริปเปิล ดีอีเอส แบบรหัสกุญแจเป็นตัวเดียวกัน

4.3.1 แบบรหัสกุญแจทั้ง 3 ตัวแตกต่างกัน

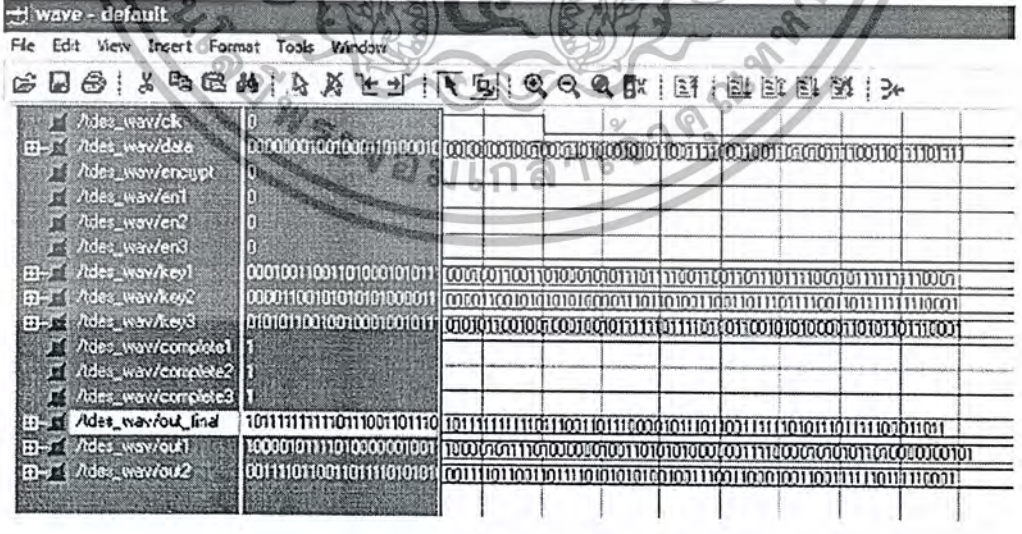
อินพุท 0000000100100011010001010110011111000100110101011110011011110111

กุญแจ1 0001001100110100010101110111110011001101110111110011011111111110001

กุญแจ2 0000110010101010100000111011010011001101110111110011011111111110001

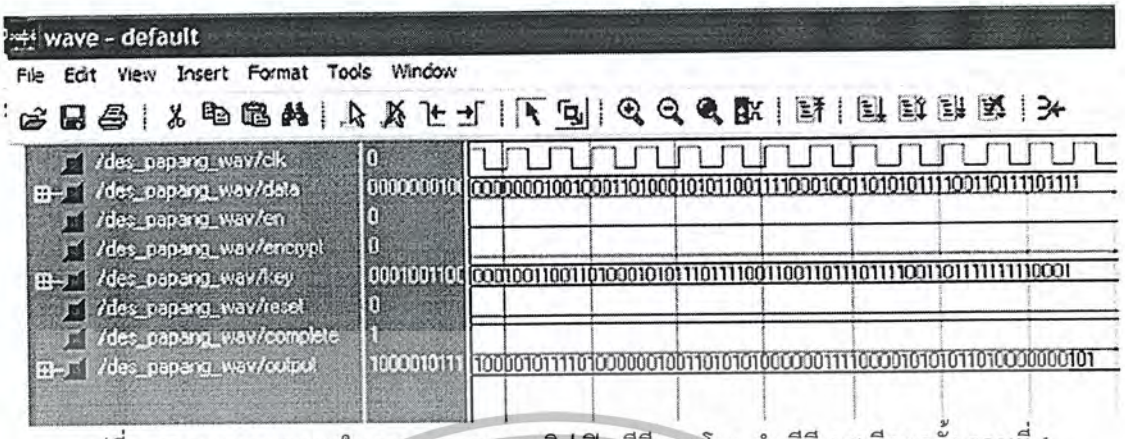
กุญแจ3 0101011001001000100101111111011110100110010101000011010101011110001

เอาท์พุท 1011111111111011100110111000001011101100111111010111011111000011011



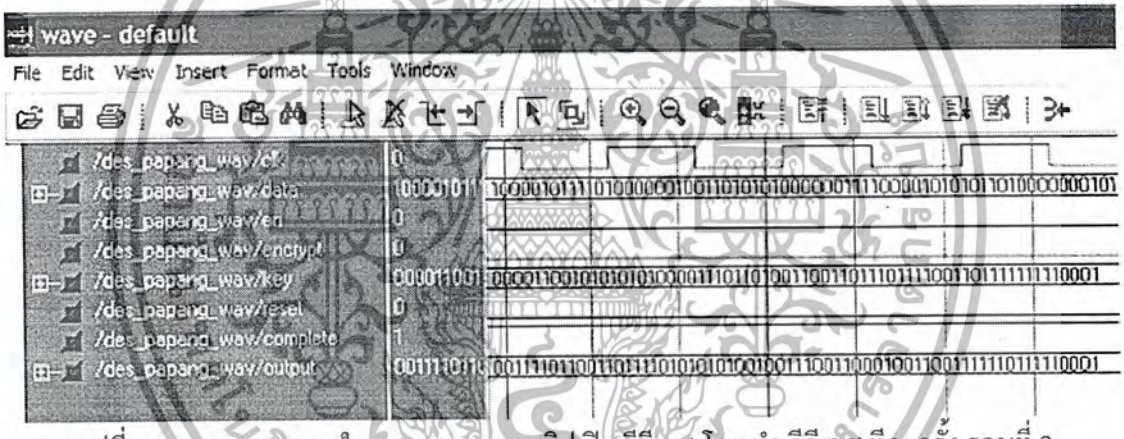
รูปที่ 4.51 ผลการทดลอง ทริปเปิล ดีอีเอส แบบรหัสกุญแจเป็นคนละตัว

ตรวจสอบการเข้ารหัส ดีอีเอส รอบที่ 1



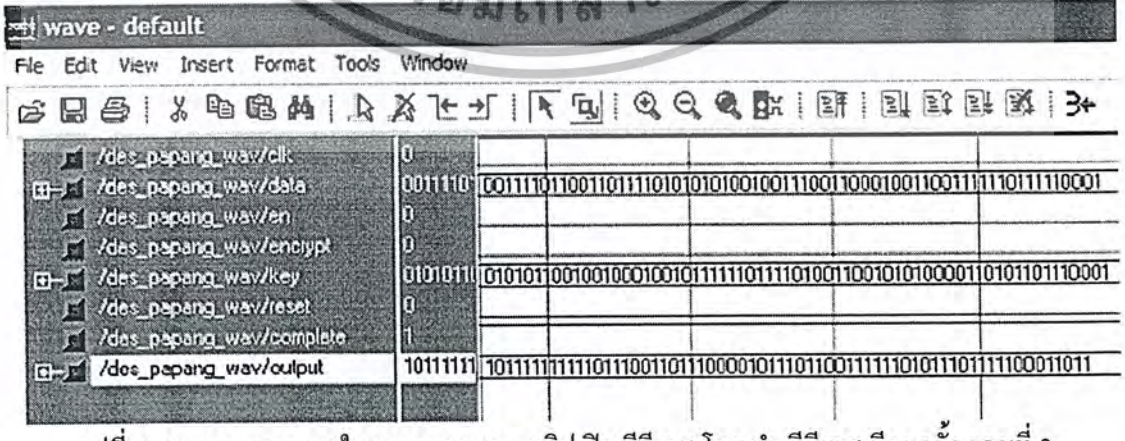
รูปที่ 4.52 ผลการทดลองในการตรวจสอบ ทริปเปิล ดีอีเอส โดย ทำ ดีอีเอส ทีละครั้ง รอบที่ 1

ตรวจสอบการเข้ารหัส ดีอีเอส รอบที่ 2



รูปที่ 4.53 ผลการทดลองในการตรวจสอบ ทริปเปิล ดีอีเอส โดย ทำ ดีอีเอส ทีละครั้ง รอบที่ 2

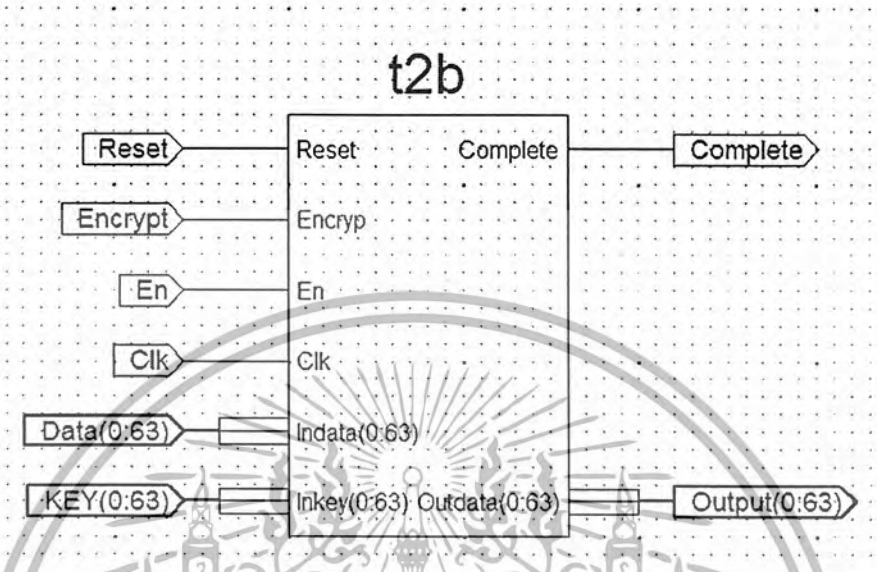
ตรวจสอบการเข้ารหัส ดีอีเอส รอบที่ 3



รูปที่ 4.53 ผลการทดลองในการตรวจสอบ ทริปเปิล ดีอีเอส โดย ทำ ดีอีเอส ทีละครั้ง รอบที่ 3

4.4 การถอดรหัสของ ดีอีเอส และ ทริปเปิล ดีอีเอส

4.4.1 การถอดรหัสแบบ ดีอีเอส

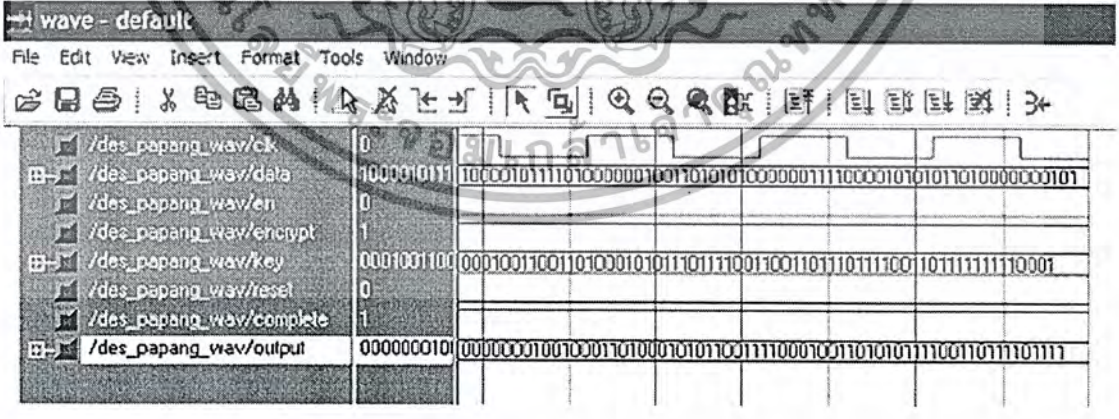


รูปที่ 4.54 วงจรรวมของการถอดรหัสแบบ ดีอีเอส

อินพุท 1000010111101000000100110101010000001111000010101011010000000101

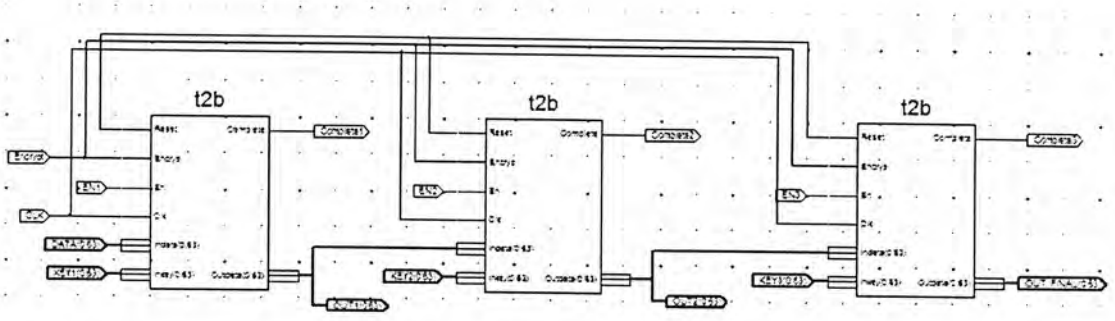
กุญแจ 0001001100110100010101110111100110011011101111001101111111110001

เอาท์พุท 0000000100100011010001010110011111000100110101011110011011111011111



รูปที่ 4.55 ผลการทดลองถอดรหัส ดีอีเอส โดยใช้ข้อมูลและรหัสกุญแจชุดเดิม

4.4.2 การถอดรหัสแบบ ทริปเปิล ดีอีเอส



รูปที่ 4.56 วงจรรวมของการถอดรหัส ทริปเปิล ดีอีเอส

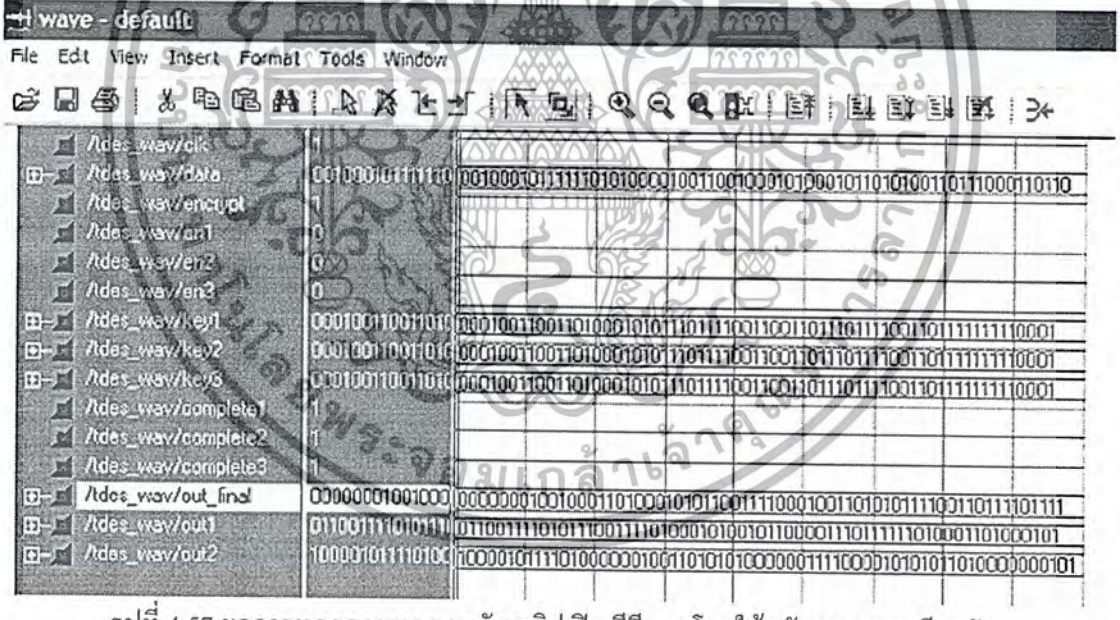
อินพุท 0010001011111110101000010011001000101000101101010011011100011O110

กุญแจ1 0001001100110100010101110111100110011011101111001101111111110001

กุญแจ2 0001001100110100010101110111100110011011101111001101111111110001

กุญแจ3 0001001100110100010101110111100110011011101111001101111111110001

เอาท์พุท 0000000100100011010001010110011111000100110101011110011011110111



รูปที่ 4.57 ผลการทดลองการถอดรหัส ทริปเปิล ดีอีเอส โดยใช้รหัสกุญแจชุดเดียวกัน

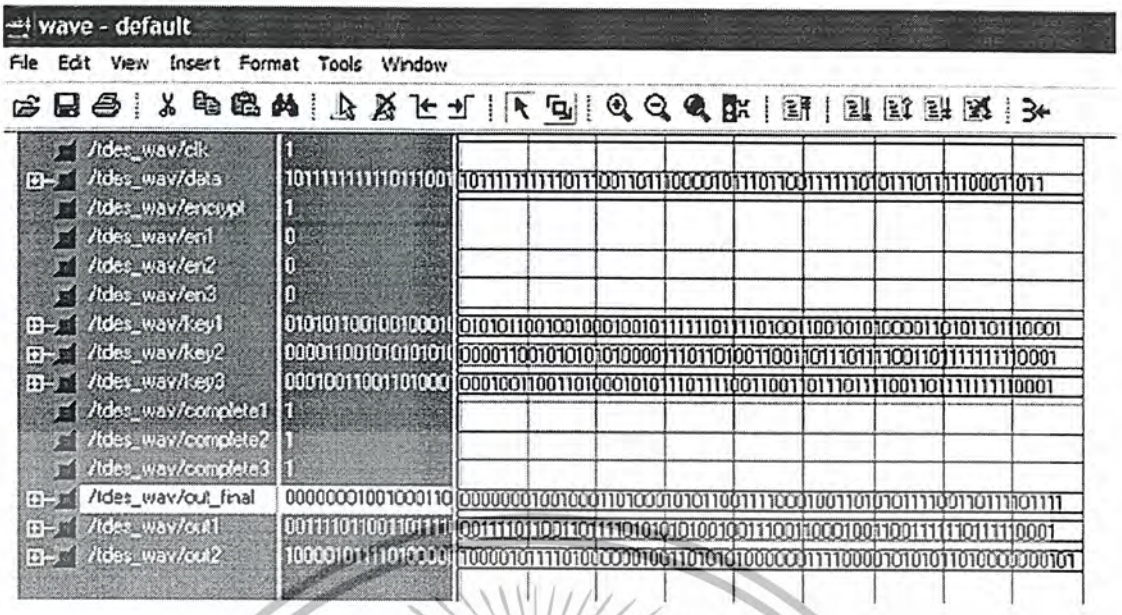
อินพุท 1011111111110111001101110000101110110011111110101110111111000110 11

กุญแจ1 0101011001001000100101111111011110100110010101000011010110111000 01

กุญแจ2 0000110010101010100001110110100110011011101111001101111111110001

กุญแจ3 0001001100110100010101110111100110011011101111001101111111110001

เอาท์พุท 0000000100100011010001010110011111000100110101011110011011110111



รูปที่ 4.58 ผลการทดลองการถอดรหัส ทริปเปิล ดีอีเอส โดยใช้รหัสกุญแจคนละชุด

4.5 ผลการทดลองฮาร์ดแวร์

ในส่วนของการทดลองฮาร์ดแวร์ จะเป็นการทดลองการเข้ารหัสแบบดีอีเอสเมื่อทำการโปรแกรมลงบนชิปเอฟพีจีเอ โดยจะมีการป้อนข้อมูลที่จะทำการเข้ารหัสและรหัสกุญแจจากเครื่องคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมและเปรียบเทียบกับผลการทดลองจากซอฟแวร์

4.5.1 การทดลองเข้ารหัสแบบดีอีเอส

ในการทดลองจะกำหนดให้มีรัสกุญแจเป็นตัวอักษรความยาว 8 ตัวอักษร ในที่นี้กำหนดให้เป็น '12345678' และอินพุทที่จะนำมาเข้ารหัสเป็นตัวอักษรที่ความยาว 16 ตัวอักษร หรือเป็นเลขฐานสองความยาว 128 บิต ในที่นี้กำหนดให้เป็น 'ABCDEFGHIJKLMNOP' โดยในการเข้ารหัสการกำหนดที่ Encrypt/Decrypt (0 = เข้ารหัส, 1 = ถอดรหัส) ในขั้นตอนนี้จะเป็นการทดลองการเข้ารหัส จึงกำหนดให้เป็น 0 = เข้ารหัส

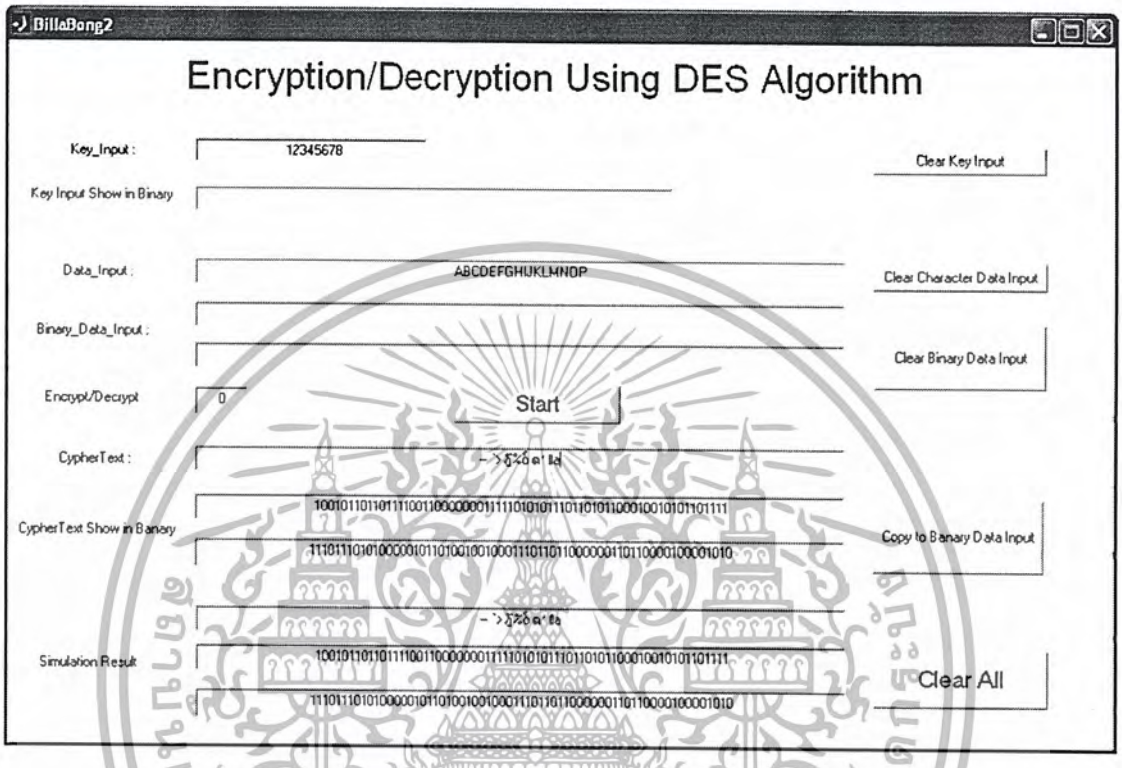
ผลการทดลองที่ได้จะมีการเปรียบเทียบผลการทดลองระหว่างผลที่ทำการเข้ารหัสในฮาร์แวร์และผลการทดลองที่ทำการเข้ารหัสโดยซอฟแวร์ เป็นดังนี้

อินพุท ABCDEFGHIJKLMNOP

กุญแจ 12345678

เอาท์พุท 1001011011011110011000000011111010101110110101100010010101101111

1110111010100000101101001001000111011011000000011011000010000101O



รูปที่ 4.59 การเข้ารหัสแบบดีอีเอสบนชิปเอฟพีจีเอ

4.5.2 การทดลองถอดรหัสแบบดีอีเอส

ในการทดลองถอดรหัสแบบดีอีเอส จะทำโดยการนำข้อมูลที่ได้จากการเข้ารหัสมาทำการถอดรหัส โดยใช้รหัสกุญแจตัวเดียวกันกับที่ทำการเข้ารหัส และผลการทดลองที่ถูกต้องจะต้องเป็นข้อมูลที่ทำการเข้ารหัสในครั้งแรก ก็คือ 'ABCDEFGHIJKLMNOP' โดยในการทดลองการถอดรหัส จะต้องกำหนด Encrypt/Decrypt ให้เป็น 1 = ถอดรหัส เป็นดังนี้

อินพุท 1001011011011110011000000011111010101110110101100010010101101111

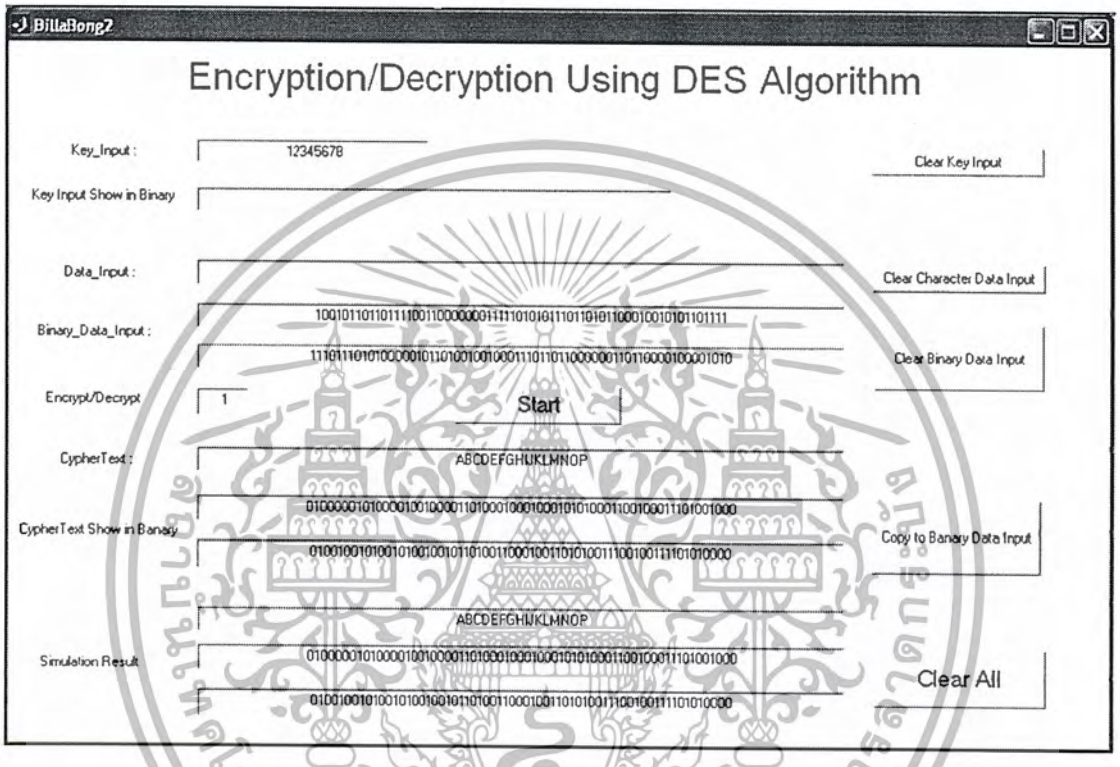
1110111010100000101101001001000111011011000000110110000100001010

กุญแจ 12345678

เอาท์พุท ABCDEFGHIJKLMNOP

0100000101000010010000110100010001000101010001100100011101001000

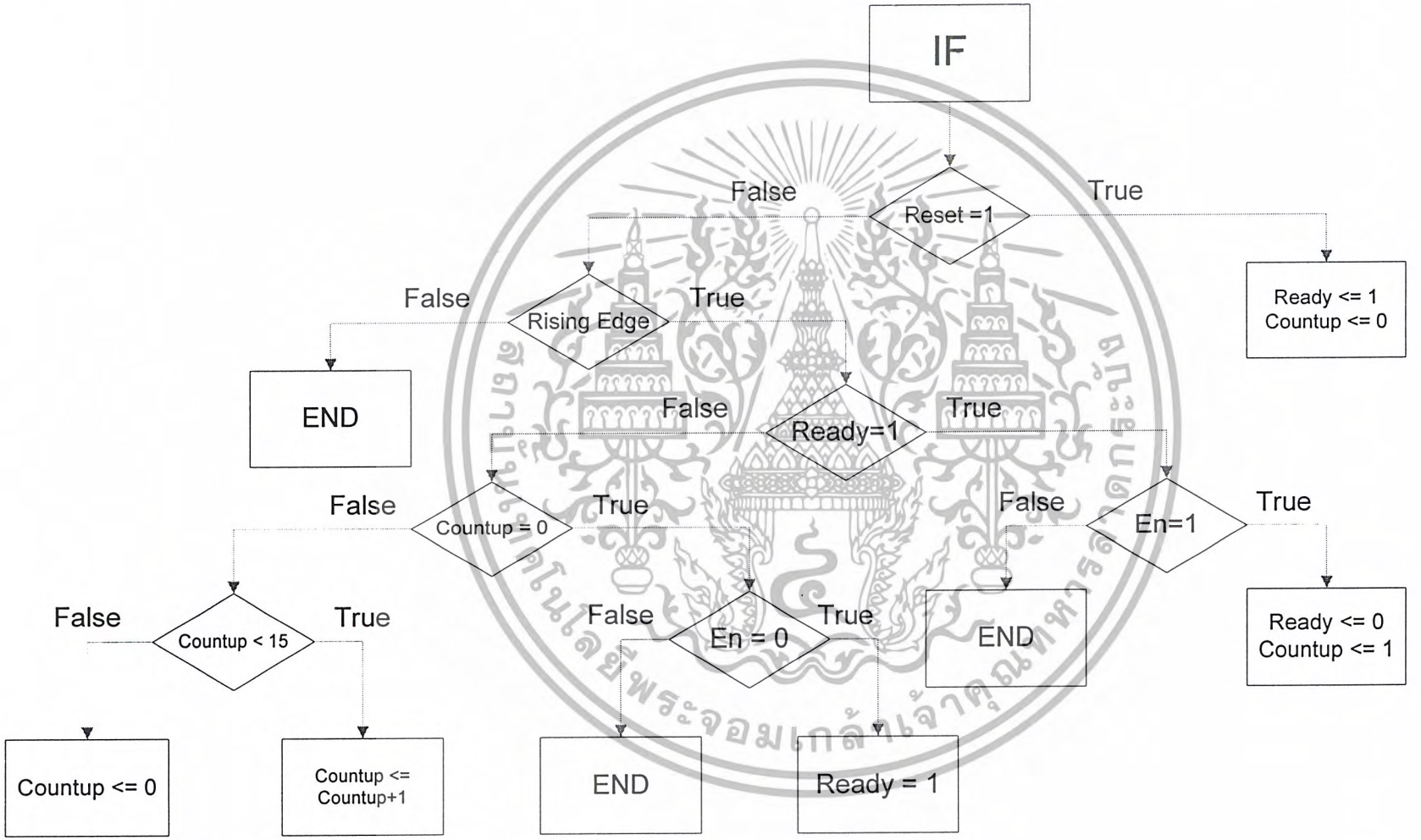
0100100101001010010010110100110001001101010011100100111101010000



รูปที่ 4.60 การถอดรหัสแบบดีอีเอสบนชิปเอฟพีจีเอ

ภาคผนวก

ภาคผนวก ก. Flow Chart แสดงการทำงานของโปรแกรม



ภาคผนวก ข. Source Code ที่เขียนด้วยภาษา VHDL

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Use IEEE.STD_LOGIC_ARITH.ALL;
Use IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity T2B is
    Port ( Indata   : in  std_logic_vector(0 to 63);
           Inkey    : in  std_logic_vector(0 to 63);
           Outdata  : out std_logic_vector(0 to 63);
           Reset    : in  std_logic;
           Encryp   : in  std_logic;
           En       : in  std_logic;
           Clk      : in  std_logic;
           Complete : out std_logic);
End T2B;

Architecture Des of T2B is

           signal  K01: std_logic_vector(0 to 47);
           signal  K02: std_logic_vector(0 to 47);
           signal  K03: std_logic_vector(0 to 47);
           signal  K04: std_logic_vector(0 to 47);
           signal  K05: std_logic_vector(0 to 47);
           signal  K06: std_logic_vector(0 to 47);
           signal  K07: std_logic_vector(0 to 47);
           signal  K08: std_logic_vector(0 to 47);
           signal  K09: std_logic_vector(0 to 47);
           signal  K10: std_logic_vector(0 to 47);
           signal  K11: std_logic_vector(0 to 47);
           signal  K12: std_logic_vector(0 to 47);
           signal  K13: std_logic_vector(0 to 47);
           signal  K14: std_logic_vector(0 to 47);
           signal  K15: std_logic_vector(0 to 47);

           signal Mykey     : std_logic_vector(0 to 47);
           signal Incypher  : std_logic_vector(0 to 63);
           signal Outcypher : std_logic_vector(0 to 63);
           signal Countup   : integer range 0 to 16;
           signal Ready     : std_logic;

           signal b1: std_logic_vector(0 to 5);
           signal b2: std_logic_vector(0 to 5);
           signal b3: std_logic_vector(0 to 5);
           signal b4: std_logic_vector(0 to 5);
           signal b5: std_logic_vector(0 to 5);
           signal b6: std_logic_vector(0 to 5);
           signal b7: std_logic_vector(0 to 5);
           signal b8: std_logic_vector(0 to 5);
           signal s1: std_logic_vector(0 to 3);
           signal s2: std_logic_vector(0 to 3);
           signal s3: std_logic_vector(0 to 3);
           signal s4: std_logic_vector(0 to 3);
           signal s5: std_logic_vector(0 to 3);
           signal s6: std_logic_vector(0 to 3);
           signal s7: std_logic_vector(0 to 3);
           signal s8: std_logic_vector(0 to 3);
```

```
Begin
      process (clk, En, Encryp)
      Begin

            If rising_edge(clk) then
                  If En = '1' then
                        If Encryp = '1' then

      K15 <=      inkey( 9) & inkey(50) & inkey(33) & inkey(59) &
                  inkey(48) & inkey(16) & inkey(32) & inkey(56) &
                  inkey( 1) & inkey( 8) & inkey(18) & inkey(41) &
                  inkey( 2) & inkey(34) & inkey(25) & inkey(24) &
                  inkey(43) & inkey(57) & inkey(58) & inkey( 0) &
                  inkey(35) & inkey(26) & inkey(17) & inkey(40) &
                  inkey(21) & inkey(27) & inkey(38) & inkey(53) &
                  inkey(36) & inkey( 3) & inkey(46) & inkey(29) &
                  inkey( 4) & inkey(52) & inkey(22) & inkey(28) &
                  inkey(60) & inkey(20) & inkey(37) & inkey(62) &
                  inkey(14) & inkey(19) & inkey(44) & inkey(13) &
                  inkey(12) & inkey(61) & inkey(54) & inkey(30);

      K14 <=      inkey( 1) & inkey(42) & inkey(25) & inkey(51) &
                  inkey(40) & inkey( 8) & inkey(24) & inkey(48) &
                  inkey(58) & inkey( 0) & inkey(10) & inkey(33) &
                  inkey(59) & inkey(26) & inkey(17) & inkey(16) &
                  inkey(35) & inkey(49) & inkey(50) & inkey(57) &
                  inkey(56) & inkey(18) & inkey( 9) & inkey(32) &
                  inkey(13) & inkey(19) & inkey(30) & inkey(45) &
                  inkey(28) & inkey(62) & inkey(38) & inkey(21) &
                  inkey(27) & inkey(44) & inkey(14) & inkey(20) &
                  inkey(52) & inkey(12) & inkey(29) & inkey(54) &
                  inkey( 6) & inkey(11) & inkey(36) & inkey( 5) &
                  inkey( 4) & inkey(53) & inkey(46) & inkey(22);

      K13 <=      inkey(50) & inkey(26) & inkey( 9) & inkey(35) &
                  inkey(24) & inkey(57) & inkey( 8) & inkey(32) &
                  inkey(42) & inkey(49) & inkey(59) & inkey(17) &
                  inkey(43) & inkey(10) & inkey( 1) & inkey( 0) &
                  inkey(48) & inkey(33) & inkey(34) & inkey(41) &
                  inkey(40) & inkey( 2) & inkey(58) & inkey(16) &
                  inkey(60) & inkey( 3) & inkey(14) & inkey(29) &
                  inkey(12) & inkey(46) & inkey(22) & inkey( 5) &
                  inkey(11) & inkey(28) & inkey(61) & inkey( 4) &
                  inkey(36) & inkey(27) & inkey(13) & inkey(38) &
                  inkey(53) & inkey(62) & inkey(20) & inkey(52) &
                  inkey(19) & inkey(37) & inkey(30) & inkey( 6);

      K12 <=      inkey(34) & inkey(10) & inkey(58) & inkey(48) &
                  inkey( 8) & inkey(41) & inkey(57) & inkey(16) &
                  inkey(26) & inkey(33) & inkey(43) & inkey( 1) &
                  inkey(56) & inkey(59) & inkey(50) & inkey(49) &
                  inkey(32) & inkey(17) & inkey(18) & inkey(25) &
                  inkey(24) & inkey(51) & inkey(42) & inkey( 0) &
                  inkey(44) & inkey(54) & inkey(61) & inkey(13) &
                  inkey(27) & inkey(30) & inkey( 6) & inkey(52) &
                  inkey(62) & inkey(12) & inkey(45) & inkey(19) &
                  inkey(20) & inkey(11) & inkey(60) & inkey(22) &
                  inkey(37) & inkey(46) & inkey( 4) & inkey(36) &
                  inkey( 3) & inkey(21) & inkey(14) & inkey(53);
```

```
K11 <=      inkey(18) & inkey(59) & inkey(42) & inkey(32) &
            inkey(57) & inkey(25) & inkey(41) & inkey( 0) &
            inkey(10) & inkey(17) & inkey(56) & inkey(50) &
            inkey(40) & inkey(43) & inkey(34) & inkey(33) &
            inkey(16) & inkey( 1) & inkey( 2) & inkey( 9) &
            inkey( 8) & inkey(35) & inkey(26) & inkey(49) &
            inkey(28) & inkey(38) & inkey(45) & inkey(60) &
            inkey(11) & inkey(14) & inkey(53) & inkey(36) &
            inkey(46) & inkey(27) & inkey(29) & inkey( 3) &
            inkey( 4) & inkey(62) & inkey(44) & inkey( 6) &
            inkey(21) & inkey(30) & inkey(19) & inkey(20) &
            inkey(54) & inkey( 5) & inkey(61) & inkey(37);

K10 <=      inkey( 2) & inkey(43) & inkey(26) & inkey(16) &
            inkey(41) & inkey( 9) & inkey(25) & inkey(49) &
            inkey(59) & inkey( 1) & inkey(40) & inkey(34) &
            inkey(24) & inkey(56) & inkey(18) & inkey(17) &
            inkey( 0) & inkey(50) & inkey(51) & inkey(58) &
            inkey(57) & inkey(48) & inkey(10) & inkey(33) &
            inkey(12) & inkey(22) & inkey(29) & inkey(44) &
            inkey(62) & inkey(61) & inkey(37) & inkey(20) &
            inkey(30) & inkey(11) & inkey(13) & inkey(54) &
            inkey(19) & inkey(46) & inkey(28) & inkey(53) &
            inkey( 5) & inkey(14) & inkey( 3) & inkey( 4) &
            inkey(38) & inkey(52) & inkey(45) & inkey(21);

K09 <=      inkey(51) & inkey(56) & inkey(10) & inkey( 0) &
            inkey(25) & inkey(58) & inkey( 9) & inkey(33) &
            inkey(43) & inkey(50) & inkey(24) & inkey(18) &
            inkey( 8) & inkey(40) & inkey( 2) & inkey( 1) &
            inkey(49) & inkey(34) & inkey(35) & inkey(42) &
            inkey(41) & inkey(32) & inkey(59) & inkey(17) &
            inkey(27) & inkey( 6) & inkey(13) & inkey(28) &
            inkey(46) & inkey(45) & inkey(21) & inkey( 4) &
            inkey(14) & inkey(62) & inkey(60) & inkey(38) &
            inkey( 3) & inkey(30) & inkey(12) & inkey(37) &
            inkey(52) & inkey(61) & inkey(54) & inkey(19) &
            inkey(22) & inkey(36) & inkey(29) & inkey(5);

K08 <=      inkey(35) & inkey(40) & inkey(59) & inkey(49) &
            inkey( 9) & inkey(42) & inkey(58) & inkey(17) &
            inkey(56) & inkey(34) & inkey( 8) & inkey( 2) &
            inkey(57) & inkey(24) & inkey(51) & inkey(50) &
            inkey(33) & inkey(18) & inkey(48) & inkey(26) &
            inkey(25) & inkey(16) & inkey(43) & inkey( 1) &
            inkey(11) & inkey(53) & inkey(60) & inkey(12) &
            inkey(30) & inkey(29) & inkey( 5) & inkey(19) &
            inkey(61) & inkey(46) & inkey(44) & inkey(22) &
            inkey(54) & inkey(14) & inkey(27) & inkey(21) &
            inkey(36) & inkey(45) & inkey(38) & inkey( 3) &
            inkey( 6) & inkey(20) & inkey(13) & inkey(52);

K07 <=      inkey(56) & inkey(32) & inkey(51) & inkey(41) &
            inkey( 1) & inkey(34) & inkey(50) & inkey( 9) &
            inkey(48) & inkey(26) & inkey( 0) & inkey(59) &
            inkey(49) & inkey(16) & inkey(43) & inkey(42) &
            inkey(25) & inkey(10) & inkey(40) & inkey(18) &
            inkey(17) & inkey( 8) & inkey(35) & inkey(58) &
            inkey( 3) & inkey(45) & inkey(52) & inkey( 4) &
            inkey(22) & inkey(21) & inkey(60) & inkey(11) &
```

```
                inkey(53) & inkey(38) & inkey(36) & inkey(14) &
                inkey(46) & inkey( 6) & inkey(19) & inkey(13) &
                inkey(28) & inkey(37) & inkey(30) & inkey(62) &
                inkey(61) & inkey(12) & inkey(5) & inkey(44);

K06 <=          inkey(40) & inkey(16) & inkey(35) & inkey(25) &
                inkey(50) & inkey(18) & inkey(34) & inkey(58) &
                inkey(32) & inkey(10) & inkey(49) & inkey(43) &
                inkey(33) & inkey( 0) & inkey(56) & inkey(26) &
                inkey( 9) & inkey(59) & inkey(24) & inkey( 2) &
                inkey( 1) & inkey(57) & inkey(48) & inkey(42) &
                inkey(54) & inkey(29) & inkey(36) & inkey(19) &
                inkey( 6) & inkey( 5) & inkey(44) & inkey(62) &
                inkey(37) & inkey(22) & inkey(20) & inkey(61) &
                inkey(30) & inkey(53) & inkey( 3) & inkey(60) &
                inkey(12) & inkey(21) & inkey(14) & inkey(46) &
                inkey(45) & inkey(27) & inkey(52) & inkey(28);

K05 <=          inkey(24) & inkey( 0) & inkey(48) & inkey( 9) &
                inkey(34) & inkey( 2) & inkey(18) & inkey(42) &
                inkey(16) & inkey(59) & inkey(33) & inkey(56) &
                inkey(17) & inkey(49) & inkey(40) & inkey(10) &
                inkey(58) & inkey(43) & inkey( 8) & inkey(51) &
                inkey(50) & inkey(41) & inkey(32) & inkey(26) &
                inkey(38) & inkey(13) & inkey(20) & inkey( 3) &
                inkey(53) & inkey(52) & inkey(28) & inkey(46) &
                inkey(21) & inkey( 6) & inkey( 4) & inkey(45) &
                inkey(14) & inkey(37) & inkey(54) & inkey(44) &
                inkey(27) & inkey( 5) & inkey(61) & inkey(30) &
                inkey(29) & inkey(11) & inkey(36) & inkey(12);

K04 <=          inkey( 8) & inkey(49) & inkey(32) & inkey(58) &
                inkey(18) & inkey(51) & inkey( 2) & inkey(26) &
                inkey( 0) & inkey(43) & inkey(17) & inkey(40) &
                inkey( 1) & inkey(33) & inkey(24) & inkey(59) &
                inkey(42) & inkey(56) & inkey(57) & inkey(35) &
                inkey(34) & inkey(25) & inkey(16) & inkey(10) &
                inkey(22) & inkey(60) & inkey( 4) & inkey(54) &
                inkey(37) & inkey(36) & inkey(12) & inkey(30) &
                inkey( 5) & inkey(53) & inkey(19) & inkey(29) &
                inkey(61) & inkey(21) & inkey(38) & inkey(28) &
                inkey(11) & inkey(52) & inkey(45) & inkey(14) &
                inkey(13) & inkey(62) & inkey(20) & inkey(27);

K03 <=          inkey(57) & inkey(33) & inkey(16) & inkey(42) &
                inkey( 2) & inkey(35) & inkey(51) & inkey(10) &
                inkey(49) & inkey(56) & inkey( 1) & inkey(24) &
                inkey(50) & inkey(17) & inkey( 8) & inkey(43) &
                inkey(26) & inkey(40) & inkey(41) & inkey(48) &
                inkey(18) & inkey( 9) & inkey( 0) & inkey(59) &
                inkey( 6) & inkey(44) & inkey(19) & inkey(38) &
                inkey(21) & inkey(20) & inkey(27) & inkey(14) &
                inkey(52) & inkey(37) & inkey( 3) & inkey(13) &
                inkey(45) & inkey( 5) & inkey(22) & inkey(12) &
                inkey(62) & inkey(36) & inkey(29) & inkey(61) &
                inkey(60) & inkey(46) & inkey( 4) & inkey(11);

K02 <=          inkey(41) & inkey(17) & inkey( 0) & inkey(26) &
                inkey(51) & inkey(48) & inkey(35) & inkey(59) &
                inkey(33) & inkey(40) & inkey(50) & inkey( 8) &
                inkey(34) & inkey( 1) & inkey(57) & inkey(56) &
```

```
                inkey(10) & inkey(24) & inkey(25) & inkey(32) &
                inkey( 2) & inkey(58) & inkey(49) & inkey(43) &
                inkey(53) & inkey(28) & inkey( 3) & inkey(22) &
                inkey( 5) & inkey( 4) & inkey(11) & inkey(61) &
                inkey(36) & inkey(21) & inkey(54) & inkey(60) &
                inkey(29) & inkey(52) & inkey( 6) & inkey(27) &
                inkey(46) & inkey(20) & inkey(13) & inkey(45) &
                inkey(44) & inkey(30) & inkey(19) & inkey(62);

        K01 <=  inkey(25) & inkey( 1) & inkey(49) & inkey(10) &
                inkey(35) & inkey(32) & inkey(48) & inkey(43) &
                inkey(17) & inkey(24) & inkey(34) & inkey(57) &
                inkey(18) & inkey(50) & inkey(41) & inkey(40) &
                inkey(59) & inkey( 8) & inkey( 9) & inkey(16) &
                inkey(51) & inkey(42) & inkey(33) & inkey(56) &
                inkey(37) & inkey(12) & inkey(54) & inkey( 6) &
                inkey(52) & inkey(19) & inkey(62) & inkey(45) &
                inkey(20) & inkey( 5) & inkey(38) & inkey(44) &
                inkey(13) & inkey(36) & inkey(53) & inkey(11) &
                inkey(30) & inkey( 4) & inkey(60) & inkey(29) &
                inkey(28) & inkey(14) & inkey( 3) & inkey(46);
else

        K01 <=  inkey( 1) & inkey(42) & inkey(25) & inkey(51) &
                inkey(40) & inkey( 8) & inkey(24) & inkey(48) &
                inkey(58) & inkey( 0) & inkey(10) & inkey(33) &
                inkey(59) & inkey(26) & inkey(17) & inkey(16) &
                inkey(35) & inkey(49) & inkey(50) & inkey(57) &
                inkey(56) & inkey(18) & inkey( 9) & inkey(32) &
                inkey(13) & inkey(19) & inkey(30) & inkey(45) &
                inkey(28) & inkey(62) & inkey(38) & inkey(21) &
                inkey(27) & inkey(44) & inkey(14) & inkey(20) &
                inkey(52) & inkey(12) & inkey(29) & inkey(54) &
                inkey( 6) & inkey(11) & inkey(36) & inkey( 5) &
                inkey( 4) & inkey(53) & inkey(46) & inkey(22);

        K02 <=  inkey(50) & inkey(26) & inkey( 9) & inkey(35) &
                inkey(24) & inkey(57) & inkey( 8) & inkey(32) &
                inkey(42) & inkey(49) & inkey(59) & inkey(17) &
                inkey(43) & inkey(10) & inkey( 1) & inkey( 0) &
                inkey(48) & inkey(33) & inkey(34) & inkey(41) &
                inkey(40) & inkey( 2) & inkey(58) & inkey(16) &
                inkey(60) & inkey( 3) & inkey(14) & inkey(29) &
                inkey(12) & inkey(46) & inkey(22) & inkey( 5) &
                inkey(11) & inkey(28) & inkey(61) & inkey( 4) &
                inkey(36) & inkey(27) & inkey(13) & inkey(38) &
                inkey(53) & inkey(62) & inkey(20) & inkey(52) &
                inkey(19) & inkey(37) & inkey(30) & inkey(6);

        K03 <=  inkey(34) & inkey(10) & inkey(58) & inkey(48) &
                inkey( 8) & inkey(41) & inkey(57) & inkey(16) &
                inkey(26) & inkey(33) & inkey(43) & inkey( 1) &
                inkey(56) & inkey(59) & inkey(50) & inkey(49) &
                inkey(32) & inkey(17) & inkey(18) & inkey(25) &
                inkey(24) & inkey(51) & inkey(42) & inkey( 0) &
                inkey(44) & inkey(54) & inkey(61) & inkey(13) &
                inkey(27) & inkey(30) & inkey( 6) & inkey(52) &
                inkey(62) & inkey(12) & inkey(45) & inkey(19) &
                inkey(20) & inkey(11) & inkey(60) & inkey(22) &
                inkey(37) & inkey(46) & inkey( 4) & inkey(36) &
                inkey( 3) & inkey(21) & inkey(14) & inkey(53);
```

```
K04 <=      inkey(18) & inkey(59) & inkey(42) & inkey(32) &
            inkey(57) & inkey(25) & inkey(41) & inkey( 0) &
            inkey(10) & inkey(17) & inkey(56) & inkey(50) &
            inkey(40) & inkey(43) & inkey(34) & inkey(33) &
            inkey(16) & inkey( 1) & inkey( 2) & inkey( 9) &
            inkey( 8) & inkey(35) & inkey(26) & inkey(49) &
            inkey(28) & inkey(38) & inkey(45) & inkey(60) &
            inkey(11) & inkey(14) & inkey(53) & inkey(36) &
            inkey(46) & inkey(27) & inkey(29) & inkey( 3) &
            inkey( 4) & inkey(62) & inkey(44) & inkey( 6) &
            inkey(21) & inkey(30) & inkey(19) & inkey(20) &
            inkey(54) & inkey( 5) & inkey(61) & inkey(37);

K05 <=      inkey( 2) & inkey(43) & inkey(26) & inkey(16) &
            inkey(41) & inkey( 9) & inkey(25) & inkey(49) &
            inkey(59) & inkey( 1) & inkey(40) & inkey(34) &
            inkey(24) & inkey(56) & inkey(18) & inkey(17) &
            inkey( 0) & inkey(50) & inkey(51) & inkey(58) &
            inkey(57) & inkey(48) & inkey(10) & inkey(33) &
            inkey(12) & inkey(22) & inkey(29) & inkey(44) &
            inkey(62) & inkey(61) & inkey(37) & inkey(20) &
            inkey(30) & inkey(11) & inkey(13) & inkey(54) &
            inkey(19) & inkey(46) & inkey(28) & inkey(53) &
            inkey( 5) & inkey(14) & inkey( 3) & inkey( 4) &
            inkey(38) & inkey(52) & inkey(45) & inkey(21);

K06 <=      inkey(51) & inkey(56) & inkey(10) & inkey( 0) &
            inkey(25) & inkey(58) & inkey( 9) & inkey(33) &
            inkey(43) & inkey(50) & inkey(24) & inkey(18) &
            inkey( 8) & inkey(40) & inkey( 2) & inkey( 1) &
            inkey(49) & inkey(34) & inkey(35) & inkey(42) &
            inkey(41) & inkey(32) & inkey(59) & inkey(17) &
            inkey(27) & inkey( 6) & inkey(13) & inkey(28) &
            inkey(46) & inkey(45) & inkey(21) & inkey( 4) &
            inkey(14) & inkey(62) & inkey(60) & inkey(38) &
            inkey( 3) & inkey(30) & inkey(12) & inkey(37) &
            inkey(52) & inkey(61) & inkey(54) & inkey(19) &
            inkey(22) & inkey(36) & inkey(29) & inkey( 5);

K07 <=      inkey(35) & inkey(40) & inkey(59) & inkey(49) &
            inkey( 9) & inkey(42) & inkey(58) & inkey(17) &
            inkey(56) & inkey(34) & inkey( 8) & inkey( 2) &
            inkey(57) & inkey(24) & inkey(51) & inkey(50) &
            inkey(33) & inkey(18) & inkey(48) & inkey(26) &
            inkey(25) & inkey(16) & inkey(43) & inkey( 1) &
            inkey(11) & inkey(53) & inkey(60) & inkey(12) &
            inkey(30) & inkey(29) & inkey( 5) & inkey(19) &
            inkey(61) & inkey(46) & inkey(44) & inkey(22) &
            inkey(54) & inkey(14) & inkey(27) & inkey(21) &
            inkey(36) & inkey(45) & inkey(38) & inkey( 3) &
            inkey( 6) & inkey(20) & inkey(13) & inkey(52);

K08 <=      inkey(56) & inkey(32) & inkey(51) & inkey(41) &
            inkey( 1) & inkey(34) & inkey(50) & inkey( 9) &
            inkey(48) & inkey(26) & inkey( 0) & inkey(59) &
            inkey(49) & inkey(16) & inkey(43) & inkey(42) &
            inkey(25) & inkey(10) & inkey(40) & inkey(18) &
            inkey(17) & inkey( 8) & inkey(35) & inkey(58) &
            inkey( 3) & inkey(45) & inkey(52) & inkey( 4) &
            inkey(22) & inkey(21) & inkey(60) & inkey(11) &
```

```
                inkey(53) & inkey(38) & inkey(36) & inkey(14) &
                inkey(46) & inkey( 6) & inkey(19) & inkey(13) &
                inkey(28) & inkey(37) & inkey(30) & inkey(62) &
                inkey(61) & inkey(12) & inkey( 5) & inkey(44);

K09 <=          inkey(40) & inkey(16) & inkey(35) & inkey(25) &
                inkey(50) & inkey(18) & inkey(34) & inkey(58) &
                inkey(32) & inkey(10) & inkey(49) & inkey(43) &
                inkey(33) & inkey( 0) & inkey(56) & inkey(26) &
                inkey( 9) & inkey(59) & inkey(24) & inkey( 2) &
                inkey( 1) & inkey(57) & inkey(48) & inkey(42) &
                inkey(54) & inkey(29) & inkey(36) & inkey(19) &
                inkey( 6) & inkey( 5) & inkey(44) & inkey(62) &
                inkey(37) & inkey(22) & inkey(20) & inkey(61) &
                inkey(30) & inkey(53) & inkey( 3) & inkey(60) &
                inkey(12) & inkey(21) & inkey(14) & inkey(46) &
                inkey(45) & inkey(27) & inkey(52) & inkey(28);

K10 <=          inkey(24) & inkey( 0) & inkey(48) & inkey( 9) &
                inkey(34) & inkey( 2) & inkey(18) & inkey(42) &
                inkey(16) & inkey(59) & inkey(33) & inkey(56) &
                inkey(17) & inkey(49) & inkey(40) & inkey(10) &
                inkey(58) & inkey(43) & inkey( 8) & inkey(51) &
                inkey(50) & inkey(41) & inkey(32) & inkey(26) &
                inkey(38) & inkey(13) & inkey(20) & inkey( 3) &
                inkey(53) & inkey(52) & inkey(28) & inkey(46) &
                inkey(21) & inkey( 6) & inkey( 4) & inkey(45) &
                inkey(14) & inkey(37) & inkey(54) & inkey(44) &
                inkey(27) & inkey( 5) & inkey(61) & inkey(30) &
                inkey(29) & inkey(11) & inkey(36) & inkey(12);

K11 <=          inkey( 8) & inkey(49) & inkey(32) & inkey(58) &
                inkey(18) & inkey(51) & inkey( 2) & inkey(26) &
                inkey( 0) & inkey(43) & inkey(17) & inkey(40) &
                inkey( 1) & inkey(33) & inkey(24) & inkey(59) &
                inkey(42) & inkey(56) & inkey(57) & inkey(35) &
                inkey(34) & inkey(25) & inkey(16) & inkey(10) &
                inkey(22) & inkey(60) & inkey( 4) & inkey(54) &
                inkey(37) & inkey(36) & inkey(12) & inkey(30) &
                inkey( 5) & inkey(53) & inkey(19) & inkey(29) &
                inkey(61) & inkey(21) & inkey(38) & inkey(28) &
                inkey(11) & inkey(52) & inkey(45) & inkey(14) &
                inkey(13) & inkey(62) & inkey(20) & inkey(27);

K12 <=          inkey(57) & inkey(33) & inkey(16) & inkey(42) &
                inkey( 2) & inkey(35) & inkey(51) & inkey(10) &
                inkey(49) & inkey(56) & inkey( 1) & inkey(24) &
                inkey(50) & inkey(17) & inkey( 8) & inkey(43) &
                inkey(26) & inkey(40) & inkey(41) & inkey(48) &
                inkey(18) & inkey( 9) & inkey( 0) & inkey(59) &
                inkey( 6) & inkey(44) & inkey(19) & inkey(38) &
                inkey(21) & inkey(20) & inkey(27) & inkey(14) &
                inkey(52) & inkey(37) & inkey( 3) & inkey(13) &
                inkey(45) & inkey( 5) & inkey(22) & inkey(12) &
                inkey(62) & inkey(36) & inkey(29) & inkey(61) &
                inkey(60) & inkey(46) & inkey( 4) & inkey(11);

K13 <=          inkey(41) & inkey(17) & inkey( 0) & inkey(26) &
                inkey(51) & inkey(48) & inkey(35) & inkey(59) &
                inkey(33) & inkey(40) & inkey(50) & inkey( 8) &
                inkey(34) & inkey( 1) & inkey(57) & inkey(56) &
```

```
                inkey(10) & inkey(24) & inkey(25) & inkey(32) &
                inkey( 2) & inkey(58) & inkey(49) & inkey(43) &
                inkey(53) & inkey(28) & inkey( 3) & inkey(22) &
                inkey( 5) & inkey( 4) & inkey(11) & inkey(61) &
                inkey(36) & inkey(21) & inkey(54) & inkey(60) &
                inkey(29) & inkey(52) & inkey( 6) & inkey(27) &
                inkey(46) & inkey(20) & inkey(13) & inkey(45) &
                inkey(44) & inkey(30) & inkey(19) & inkey(62);

K14 <=          inkey(25) & inkey( 1) & inkey(49) & inkey(10) &
                inkey(35) & inkey(32) & inkey(48) & inkey(43) &
                inkey(17) & inkey(24) & inkey(34) & inkey(57) &
                inkey(18) & inkey(50) & inkey(41) & inkey(40) &
                inkey(59) & inkey( 8) & inkey( 9) & inkey(16) &
                inkey(51) & inkey(42) & inkey(33) & inkey(56) &
                inkey(37) & inkey(12) & inkey(54) & inkey( 6) &
                inkey(52) & inkey(19) & inkey(62) & inkey(45) &
                inkey(20) & inkey( 5) & inkey(38) & inkey(44) &
                inkey(13) & inkey(36) & inkey(53) & inkey(11) &
                inkey(30) & inkey( 4) & inkey(60) & inkey(29) &
                inkey(28) & inkey(14) & inkey( 3) & inkey(46);

K15 <=          inkey(17) & inkey(58) & inkey(41) & inkey( 2) &
                inkey(56) & inkey(24) & inkey(40) & inkey(35) &
                inkey( 9) & inkey(16) & inkey(26) & inkey(49) &
                inkey(10) & inkey(42) & inkey(33) & inkey(32) &
                inkey(51) & inkey( 0) & inkey( 1) & inkey( 8) &
                inkey(43) & inkey(34) & inkey(25) & inkey(48) &
                inkey(29) & inkey( 4) & inkey(46) & inkey(61) &
                inkey(44) & inkey(11) & inkey(54) & inkey(37) &
                inkey(12) & inkey(60) & inkey(30) & inkey(36) &
                inkey( 5) & inkey(28) & inkey(45) & inkey( 3) &
                inkey(22) & inkey(27) & inkey(52) & inkey(21) &
                inkey(20) & inkey( 6) & inkey(62) & inkey(38);
                        End if;
                End if;
        End if;
End Process;

Process (clk, countup,Encryp)
Begin
        If rising_edge(clk) then
                case countup is
                        when 0 =>
                                If Encryp = '1' then
mykey <=        inkey(17) & inkey(58) & inkey(41) & inkey( 2) &
                inkey(56) & inkey(24) & inkey(40) & inkey(35) &
                inkey( 9) & inkey(16) & inkey(26) & inkey(49) &
                inkey(10) & inkey(42) & inkey(33) & inkey(32) &
                inkey(51) & inkey (0) & inkey( 1) & inkey( 8) &
                inkey(43) & inkey(34) & inkey(25) & inkey(48) &
                inkey(29) & inkey( 4) & inkey(46) & inkey(61) &
                inkey(44) & inkey(11) & inkey(54) & inkey(37) &
                inkey(12) & inkey(60) & inkey(30) & inkey(36) &
                inkey( 5) & inkey(28) & inkey(45) & inkey( 3) &
                inkey(22) & inkey(27) & inkey(52) & inkey(21) &
                inkey(20) & inkey( 6) & inkey(62) & inkey(38);
        Else
mykey <=        inkey( 9) & inkey(50) & inkey(33) & inkey(59) &
                inkey(48) & inkey(16) & inkey(32) & inkey(56) &
                inkey( 1) & inkey( 8) & inkey(18) & inkey(41) &
```

```
                inkey( 2) & inkey(34) & inkey(25) & inkey(24) &
                inkey(43) & inkey(57) & inkey(58) & inkey( 0) &
                inkey(35) & inkey(26) & inkey(17) & inkey(40) &
                inkey(21) & inkey(27) & inkey(38) & inkey(53) &
                inkey(36) & inkey( 3) & inkey(46) & inkey(29) &
                inkey( 4) & inkey(52) & inkey(22) & inkey(28) &
                inkey(60) & inkey(20) & inkey(37) & inkey(62) &
                inkey(14) & inkey(19) & inkey(44) & inkey(13) &
                inkey(12) & inkey(61) & inkey(54) & inkey(30);
        End If;
                        when 1  =>mykey <= K01;
                        when 2  =>mykey <= K02;
                        when 3  =>mykey <= K03;
                        when 4  =>mykey <= K04;
                        when 5  =>mykey <= K05;
                        when 6  =>mykey <= K06;
                        when 7  =>mykey <= K07;
                        when 8  =>mykey <= K08;
                        when 9  =>mykey <= K09;
                        when 10 =>mykey <= K10;
                        when 11 =>mykey <= K11;
                        when 12 =>mykey <= K12;
                        when 13 =>mykey <= K13;
                        when 14 =>mykey <= K14;
                        when 15 =>mykey <= K15;
                        when others =>
                    End case;
            End If;

End Process;

Process (clk, countup, reset, En)
    variable C17: std_logic_vector(1 to 64);
Begin
    If reset = '1' then
            Complete <= '1';
    Elsif rising_edge(clk) then
            case countup is
                        when 0 =>
                                If En = '1' then
Incypher <=  indata(57) & indata(49) & indata(41) & indata(33) &
             indata(25) & indata(17) & indata( 9) & indata( 1) &
             indata(59) & indata(51) & indata(43) & indata(35) &
             indata(27) & indata(19) & indata(11) & indata( 3) &
             indata(61) & indata(53) & indata(45) & indata(37) &
             indata(29) & indata(21) & indata(13) & indata( 5) &
             indata(63) & indata(55) & indata(47) & indata(39) &
             indata(31) & indata(23) & indata(15) & indata( 7) &
             indata(56) & indata(48) & indata(40) & indata(32) &
             indata(24) & indata(16) & indata( 8) & indata( 0) &
             indata(58) & indata(50) & indata(42) & indata(34) &
             indata(26) & indata(18) & indata(10) & indata( 2) &
             indata(60) & indata(52) & indata(44) & indata(36) &
             indata(28) & indata(20) & indata(12) & indata( 4) &
             indata(62) & indata(54) & indata(46) & indata(38) &
             indata(30) & indata(22) & indata(14) & indata( 6);
                                        Complete <= '0';
                                End If;
    If ready = '0' then
                        C17(1 to 32) := Outcypher(32 to 63);
                        C17(33 to 64) := Outcypher(0 to 31);
```

```
      outdata <=  C17(40) & C17( 8) & C17(48) & C17(16) &
                  C17(56) & C17(24) & C17(64) & C17(32) &
                  C17(39) & C17( 7) & C17(47) & C17(15) &
                  C17(55) & C17(23) & C17(63) & C17(31) &
                  C17(38) & C17( 6) & C17(46) & C17(14) &
                  C17(54) & C17(22) & C17(62) & C17(30) &
                  C17(37) & C17( 5) & C17(45) & C17(13) &
                  C17(53) & C17(21) & C17(61) & C17(29) &
                  C17(36) & C17( 4) & C17(44) & C17(12) &
                  C17(52) & C17(20) & C17(60) & C17(28) &
                  C17(35) & C17( 3) & C17(43) & C17(11) &
                  C17(51) & C17(19) & C17(59) & C17(27) &
                  C17(34) & C17( 2) & C17(42) & C17(10) &
                  C17(50) & C17(18) & C17(58) & C17(26) &
                  C17(33) & C17( 1) & C17(41) & C17(9 ) &
                  C17(49) & C17(17) & C17(57) & C17(25);
                        Complete <= '1';
            End If;
                        when others =>
                              Incypher <= Outcypher;
                              Complete <= '0';
                  End case;
            End if;

      End Process;

      b1 <= (Incypher(63) & Incypher(36) & Incypher(32 to 35)) xor
(mykey( 0) & mykey( 5) & mykey(1  to 4 ));
      b2 <= (Incypher(35) & Incypher(40) & Incypher(36 to 39)) xor
(mykey( 6) & mykey(11) & mykey(7  to 10));
      b3 <= (Incypher(39) & Incypher(44) & Incypher(40 to 43)) xor
(mykey(12) & mykey(17) & mykey(13 to 16));
      b4 <= (Incypher(43) & Incypher(48) & Incypher(44 to 47)) xor
(mykey(18) & mykey(23) & mykey(19 to 22));
      b5 <= (Incypher(47) & Incypher(52) & Incypher(48 to 51)) xor
(mykey(24) & mykey(29) & mykey(25 to 28));
      b6 <= (Incypher(51) & Incypher(56) & Incypher(52 to 55)) xor
(mykey(30) & mykey(35) & mykey(31 to 34));
      b7 <= (Incypher(55) & Incypher(60) & Incypher(56 to 59)) xor
(mykey(36) & mykey(41) & mykey(37 to 40));
      b8 <= (Incypher(59) & Incypher(32) & Incypher(60 to 63)) xor
(mykey(42) & mykey(47) & mykey(43 to 46));

      with b1 select
            s1 <= x"e" when "000000",
                  x"4" when "000001",
                  x"d" when "000010",
                  x"1" when "000011",
                  x"2" when "000100",
                  x"f" when "000101",
                  x"b" when "000110",
                  x"8" when "000111",
                  x"3" when "001000",
                  x"a" when "001001",
                  x"6" when "001010",
                  x"c" when "001011",
                  x"5" when "001100",
                  x"9" when "001101",
                  x"0" when "001110",
                  x"7" when "001111",
                  x"0" when "010000",
```

```
                    x"f" when "010001",
                    x"7" when "010010",
                    x"4" when "010011",
                    x"e" when "010100",
                    x"2" when "010101",
                    x"d" when "010110",
                    x"1" when "010111",
                    x"a" when "011000",
                    x"6" when "011001",
                    x"c" when "011010",
                    x"b" when "011011",
                    x"9" when "011100",
                    x"5" when "011101",
                    x"3" when "011110",
                    x"8" when "011111",
                    x"4" when "100000",
                    x"1" when "100001",
                    x"e" when "100010",
                    x"8" when "100011",
                    x"d" when "100100",
                    x"6" when "100101",
                    x"2" when "100110",
                    x"b" when "100111",
                    x"f" when "101000",
                    x"c" when "101001",
                    x"9" when "101010",
                    x"7" when "101011",
                    x"3" when "101100",
                    x"a" when "101101",
                    x"5" when "101110",
                    x"0" when "101111",
                    x"f" when "110000",
                    x"c" when "110001",
                    x"8" when "110010",
                    x"2" when "110011",
                    x"4" when "110100",
                    x"9" when "110101",
                    x"1" when "110110",
                    x"7" when "110111",
                    x"5" when "111000",
                    x"b" when "111001",
                    x"3" when "111010",
                    x"e" when "111011",
                    x"a" when "111100",
                    x"0" when "111101",
                    x"6" when "111110",
                    x"d" when "111111",
                    "XXXX" when others;

with b2 select
      s2 <= x"f" when "000000",
                    x"1" when "000001",
                    x"8" when "000010",
                    x"e" when "000011",
                    x"6" when "000100",
                    x"b" when "000101",
                    x"3" when "000110",
                    x"4" when "000111",
                    x"9" when "001000",
                    x"7" when "001001",
                    x"2" when "001010",
```

```
                x"d" when "001011",
                x"c" when "001100",
                x"0" when "001101",
                x"5" when "001110",
                x"a" when "001111",
                x"3" when "010000",
                x"d" when "010001",
                x"4" when "010010",
                x"7" when "010011",
                x"f" when "010100",
                x"2" when "010101",
                x"8" when "010110",
                x"e" when "010111",
                x"c" when "011000",
                x"0" when "011001",
                x"1" when "011010",
                x"a" when "011011",
                x"6" when "011100",
                x"9" when "011101",
                x"b" when "011110",
                x"5" when "011111",
                x"0" when "100000",
                x"e" when "100001",
                x"7" when "100010",
                x"b" when "100011",
                x"a" when "100100",
                x"4" when "100101",
                x"d" when "100110",
                x"1" when "100111",
                x"5" when "101000",
                x"8" when "101001",
                x"c" when "101010",
                x"6" when "101011",
                x"9" when "101100",
                x"3" when "101101",
                x"2" when "101110",
                x"f" when "101111",
                x"d" when "110000",
                x"8" when "110001",
                x"a" when "110010",
                x"1" when "110011",
                x"3" when "110100",
                x"f" when "110101",
                x"4" when "110110",
                x"2" when "110111",
                x"b" when "111000",
                x"6" when "111001",
                x"7" when "111010",
                x"c" when "111011",
                x"0" when "111100",
                x"5" when "111101",
                x"e" when "111110",
                x"9" when "111111",
                "XXXX" when others;

with b3 select
      s3 <= x"a" when "000000",
                x"0" when "000001",
                x"9" when "000010",
                x"e" when "000011",
                x"6" when "000100";
```

```
x"3" when "000101",
x"f" when "000110",
x"5" when "000111",
x"1" when "001000",
x"d" when "001001",
x"c" when "001010",
x"7" when "001011",
x"b" when "001100",
x"4" when "001101",
x"2" when "001110",
x"8" when "001111",
x"d" when "010000",
x"7" when "010001",
x"0" when "010010",
x"9" when "010011",
x"3" when "010100",
x"4" when "010101",
x"6" when "010110",
x"a" when "010111",
x"2" when "011000",
x"8" when "011001",
x"5" when "011010",
x"e" when "011011",
x"c" when "011100",
x"b" when "011101",
x"f" when "011110",
x"1" when "011111",
x"d" when "100000",
x"6" when "100001",
x"4" when "100010",
x"9" when "100011",
x"8" when "100100",
x"f" when "100101",
x"3" when "100110",
x"0" when "100111",
x"b" when "101000",
x"1" when "101001",
x"2" when "101010",
x"c" when "101011",
x"5" when "101100",
x"a" when "101101",
x"e" when "101110",
x"7" when "101111",
x"1" when "110000",
x"a" when "110001",
x"d" when "110010",
x"0" when "110011",
x"6" when "110100",
x"9" when "110101",
x"8" when "110110",
x"7" when "110111",
x"4" when "111000",
x"f" when "111001",
x"e" when "111010",
x"3" when "111011",
x"b" when "111100",
x"5" when "111101",
x"2" when "111110",
x"c" when "111111",
"XXXX" when others;
```

```
with b4 select
      s4 <= x"7" when "000000",
                    x"d" when "000001",
                    x"e" when "000010",
                    x"3" when "000011",
                    x"0" when "000100",
                    x"6" when "000101",
                    x"9" when "000110",
                    x"a" when "000111",
                    x"1" when "001000",
                    x"2" when "001001",
                    x"8" when "001010",
                    x"5" when "001011",
                    x"b" when "001100",
                    x"c" when "001101",
                    x"4" when "001110",
                    x"f" when "001111",
                    x"d" when "010000",
                    x"8" when "010001",
                    x"b" when "010010",
                    x"5" when "010011",
                    x"6" when "010100",
                    x"f" when "010101",
                    x"0" when "010110",
                    x"3" when "010111",
                    x"4" when "011000",
                    x"7" when "011001",
                    x"2" when "011010",
                    x"c" when "011011",
                    x"1" when "011100",
                    x"a" when "011101",
                    x"e" when "011110",
                    x"9" when "011111",
                    x"a" when "100000",
                    x"6" when "100001",
                    x"9" when "100010",
                    x"0" when "100011",
                    x"c" when "100100",
                    x"b" when "100101",
                    x"7" when "100110",
                    x"d" when "100111",
                    x"f" when "101000",
                    x"1" when "101001",
                    x"3" when "101010",
                    x"e" when "101011",
                    x"5" when "101100",
                    x"2" when "101101",
                    x"8" when "101110",
                    x"4" when "101111",
                    x"3" when "110000",
                    x"f" when "110001",
                    x"0" when "110010",
                    x"6" when "110011",
                    x"a" when "110100",
                    x"1" when "110101",
                    x"d" when "110110",
                    x"8" when "110111",
                    x"9" when "111000",
                    x"4" when "111001",
                    x"5" when "111010",
                    x"b" when "111011",
```

```
                    x"c" when "111100",
                    x"7" when "111101",
                    x"2" when "111110",
                    x"e" when "111111",
                    "XXXX" when others;

with b5 select
     s5 <= x"2" when "000000",
                    x"c" when "000001",
                    x"4" when "000010",
                    x"1" when "000011",
                    x"7" when "000100",
                    x"a" when "000101",
                    x"b" when "000110",
                    x"6" when "000111",
                    x"8" when "001000",
                    x"5" when "001001",
                    x"3" when "001010",
                    x"f" when "001011",
                    x"d" when "001100",
                    x"0" when "001101",
                    x"e" when "001110",
                    x"9" when "001111",
                    x"e" when "010000",
                    x"b" when "010001",
                    x"2" when "010010",
                    x"c" when "010011",
                    x"4" when "010100",
                    x"7" when "010101",
                    x"d" when "010110",
                    x"1" when "010111",
                    x"5" when "011000",
                    x"0" when "011001",
                    x"f" when "011010",
                    x"a" when "011011",
                    x"3" when "011100",
                    x"9" when "011101",
                    x"8" when "011110",
                    x"6" when "011111",
                    x"4" when "100000",
                    x"2" when "100001",
                    x"1" when "100010",
                    x"b" when "100011",
                    x"a" when "100100",
                    x"d" when "100101",
                    x"7" when "100110",
                    x"8" when "100111",
                    x"f" when "101000",
                    x"9" when "101001",
                    x"c" when "101010",
                    x"5" when "101011",
                    x"6" when "101100",
                    x"3" when "101101",
                    x"0" when "101110",
                    x"e" when "101111",
                    x"b" when "110000",
                    x"8" when "110001",
                    x"c" when "110010",
                    x"7" when "110011",
                    x"1" when "110100",
                    x"e" when "110101",
```

```
                    x"2" when "110110",
                    x"d" when "110111",
                    x"6" when "111000",
                    x"f" when "111001",
                    x"0" when "111010",
                    x"9" when "111011",
                    x"a" when "111100",
                    x"4" when "111101",
                    x"5" when "111110",
                    x"3" when "111111",
                    "XXXX" when others;

with b6 select
     s6 <= x"c" when "000000",
                    x"1" when "000001",
                    x"a" when "000010",
                    x"f" when "000011",
                    x"9" when "000100",
                    x"2" when "000101",
                    x"6" when "000110",
                    x"8" when "000111",
                    x"0" when "001000",
                    x"d" when "001001",
                    x"3" when "001010",
                    x"4" when "001011",
                    x"e" when "001100",
                    x"7" when "001101",
                    x"5" when "001110",
                    x"b" when "001111",
                    x"a" when "010000",
                    x"f" when "010001",
                    x"4" when "010010",
                    x"2" when "010011",
                    x"7" when "010100",
                    x"c" when "010101",
                    x"9" when "010110",
                    x"5" when "010111",
                    x"6" when "011000",
                    x"1" when "011001",
                    x"d" when "011010",
                    x"e" when "011011",
                    x"0" when "011100",
                    x"b" when "011101",
                    x"3" when "011110",
                    x"8" when "011111",
                    x"9" when "100000",
                    x"e" when "100001",
                    x"f" when "100010",
                    x"5" when "100011",
                    x"2" when "100100",
                    x"8" when "100101",
                    x"c" when "100110",
                    x"3" when "100111",
                    x"7" when "101000",
                    x"0" when "101001",
                    x"4" when "101010",
                    x"a" when "101011",
                    x"1" when "101100",
                    x"d" when "101101",
                    x"b" when "101110",
                    x"6" when "101111";
```

```
                    x"4" when "110000",
                    x"3" when "110001",
                    x"2" when "110010",
                    x"c" when "110011",
                    x"9" when "110100",
                    x"5" when "110101",
                    x"f" when "110110",
                    x"a" when "110111",
                    x"b" when "111000",
                    x"e" when "111001",
                    x"1" when "111010",
                    x"7" when "111011",
                    x"6" when "111100",
                    x"0" when "111101",
                    x"8" when "111110",
                    x"d" when "111111",
                    "XXXX" when others;

with b7 select
     s7 <= x"4" when "000000",
                    x"b" when "000001",
                    x"2" when "000010",
                    x"e" when "000011",
                    x"f" when "000100",
                    x"0" when "000101",
                    x"8" when "000110",
                    x"d" when "000111",
                    x"3" when "001000",
                    x"c" when "001001",
                    x"9" when "001010",
                    x"7" when "001011",
                    x"5" when "001100",
                    x"a" when "001101",
                    x"6" when "001110",
                    x"1" when "001111",
                    x"d" when "010000",
                    x"0" when "010001",
                    x"b" when "010010",
                    x"7" when "010011",
                    x"4" when "010100",
                    x"9" when "010101",
                    x"1" when "010110",
                    x"a" when "010111",
                    x"e" when "011000",
                    x"3" when "011001",
                    x"5" when "011010",
                    x"c" when "011011",
                    x"2" when "011100",
                    x"f" when "011101",
                    x"8" when "011110",
                    x"6" when "011111",
                    x"1" when "100000",
                    x"4" when "100001",
                    x"b" when "100010",
                    x"d" when "100011",
                    x"c" when "100100",
                    x"3" when "100101",
                    x"7" when "100110",
                    x"e" when "100111",
                    x"a" when "101000",
                    x"f" when "101001",
```

```
                x"6" when "101010",
                x"8" when "101011",
                x"0" when "101100",
                x"5" when "101101",
                x"9" when "101110",
                x"2" when "101111",
                x"6" when "110000",
                x"b" when "110001",
                x"d" when "110010",
                x"8" when "110011",
                x"1" when "110100",
                x"4" when "110101",
                x"a" when "110110",
                x"7" when "110111",
                x"9" when "111000",
                x"5" when "111001",
                x"0" when "111010",
                x"f" when "111011",
                x"e" when "111100",
                x"2" when "111101",
                x"3" when "111110",
                x"c" when "111111",
                "XXXX" when others;

with b8 select
    s8 <= x"d" when "000000",
                x"2" when "000001",
                x"8" when "000010",
                x"4" when "000011",
                x"6" when "000100",
                x"f" when "000101",
                x"b" when "000110",
                x"1" when "000111",
                x"a" when "001000",
                x"9" when "001001",
                x"3" when "001010",
                x"e" when "001011",
                x"5" when "001100",
                x"0" when "001101",
                x"c" when "001110",
                x"7" when "001111",
                x"1" when "010000",
                x"f" when "010001",
                x"d" when "010010",
                x"8" when "010011",
                x"a" when "010100",
                x"3" when "010101",
                x"7" when "010110",
                x"4" when "010111",
                x"c" when "011000",
                x"5" when "011001",
                x"6" when "011010",
                x"b" when "011011",
                x"0" when "011100",
                x"e" when "011101",
                x"9" when "011110",
                x"2" when "011111",
                x"7" when "100000",
                x"b" when "100001",
                x"4" when "100010",
                x"1" when "100011",
```

```
                        x"9" when "100100",
                        x"c" when "100101",
                        x"e" when "100110",
                        x"2" when "100111",
                        x"0" when "101000",
                        x"6" when "101001",
                        x"a" when "101010",
                        x"d" when "101011",
                        x"f" when "101100",
                        x"3" when "101101",
                        x"5" when "101110",
                        x"8" when "101111",
                        x"2" when "110000",
                        x"1" when "110001",
                        x"e" when "110010",
                        x"7" when "110011",
                        x"4" when "110100",
                        x"a" when "110101",
                        x"8" when "110110",
                        x"d" when "110111",
                        x"f" when "111000",
                        x"c" when "111001",
                        x"9" when "111010",
                        x"0" when "111011",
                        x"3" when "111100",
                        x"5" when "111101",
                        x"6" when "111110",
                        x"b" when "111111",
                        "XXXX" when others;

Outcypher(32 to 63) <=  (s4(3) & s2(2) & s5(3) & s6(0) &
                         s8(0) & s3(3) & s7(3) & s5(0) &
                         s1(0) & s4(2) & s6(2) & s7(1) &
                         s2(0) & s5(1) & s8(2) & s3(1) &
                         s1(1) & s2(3) & s6(3) & s4(1) &
                         s8(3) & s7(2) & s1(2) & s3(0) &
                         s5(2) & s4(0) & s8(1) & s2(1) &
                         s6(1) & s3(2) & s1(3) & s7(0) ) xor
                         Incypher(0 to 31);

Outcypher(0 to 31)  <=   Incypher(32 to 63);

Process (clk,reset,ready,En,countup)

      Begin

            If Reset = '1' then
                  ready <= '1';
                  countup <= 0;

            Elsif rising_edge(clk) then
                  If ready = '1' then
                        If En = '1' then
                              ready   <= '0';
                              countup <= 1;
                        End If;
                  Else
                        If countup  = 0 then
                              If En = '0' then
                                    ready <= '1';
```

```
                        End If;
                    Elsif countup < 15 then
                        countup <= countup + 1;
                    Else
                        countup <= 0;
                    End If;
                End If;
            End If;
        End Process;
End Des;
```

หนังสืออ้างอิง

[1] จักรกริศน์ เขียวสะอาด และ นริศภิญโญวัฒนากร ,*" การออกแบบวงจรดิจิตอลด้วยภาษาวีเอชดีแอล "* , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2538

[2] โอภาส สุวิกะปกรณ์กุล ,*'' การออกแบบวงจรป้องกันการคัดลอกซอฟแวร์โดยใช้เอฟพีจีเอ "*, คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2538

[3] Welsh , Dominic ,*" Code and Cryptography "*, Oxford : Clarendon , 1988

[4] Denning , Dorothy E , *" Cryptography and Data Security "* , Readding , MA : Addison-Wesly , 1982

[5] ชำนาญ ปัญญาใส และ วัชรากร หนูทอง , *" ภาษาวีเอชดีแอล สำหรับการออกแบบวงจรดิจิตอล "*, กรุงเทพฯ : ซีเอ็ดยูเคชั่น , 2547