การออกแบบระบบ Advanced Encryption Standard ด้วยภาษาวีเอชดีแอล

Advanced Encryption Standard system design using VHDL language

โดย

นายเวชยันต์ ศรสิทธิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

การออกแบบระบบ Advanced Encryption Standard ด้วยภาษาวีเอชดีแอล

Advanced Encryption Standard system design using VHDL language

โดย

นายเวชยันต์ ศรสิทธิ์ เลขประจำตัวนักศึกษา 43010416

อาจารย์ที่ปรึกษา

รศ.ดร.สมศักดิ์ ชุมช่วย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ ปีการศึกษา 2546

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบระบบ Advanced Encryption Standard ด้วยภาษาวีเอชดีแอล

Advanced Encryption Standard system design using VHDL language

ผู้จัดทำ

นายเวชยันต์ ศรสิทธิ์ เลขประจำตัวนักศึกษา 43010416

.....อาจารย์ที่ปรึกษา

(รศ.ดร.สมศักดิ์ ชุมช่วย)

การออกแบบระบบ Advanced Encryption Standard ด้วยภาษาวีเอชดีแอล

Advanced Encryption Standard system design using VHDL language

นายเวชยันต์ ศรสิทธิ์ เลขประจำตัวนักศึกษา 43010416

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....อาจารย์ที่ปรึกษา

(รศ.ดร.สมศักดิ์ ชุมช่วย)

การออกแบบระบบ Advanced Encryption Standard ด้วยภาษาวีเอชดีแอล

เวชยันต์ ศรสิทธิ์
รศ.ดร.สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา
ปีการศึกษา 2546

บทคัดย่อ

โครงงานนี้จะนำเสนอระบบเข้ารหัสและถอดรหัสข้อมูลแบบ Advanced Encryption Standard (AES) ซึ่งออกแบบโดยใช้ภาษาวีเอชดีแอล ภายในปริญญานิพนธ์ฉบับนี้ได้กล่าวถึงอัลกอริทึมในการเข้าสลักของระบบ Advanced Encryption Standard ภาษาวีเอชดีแอล และการออกแบบระบบดังกล่าว

ระบบที่ออกแบบขึ้นสามารถทำหน้าที่เป็นตัวเข้ารหัสหรือถอดรหัสก็ได้ รับและจ่ายข้อมูลในแบบอนุกรมแบบซิงโครนัส ทำงานแบบขนานทีละ 128 บิต ซึ่งจะทำให้สามารถประมวลผลได้เร็ว และผู้ใช้สามารถกำหนดรหัสกุญแจเองได้ นอกจากนั้นระบบนี้ยังรองรับการนำไปโปรแกรมลงเอฟพีจีเอด้วย

Advanced Encryption Standard design using VHDL language

Wetchayan Sornsith

Assoc.Prof.Dr.Somsak Choomchuay

Adviser

Academic year 2003

Abstract

Advanced Encryption standard (AES) which is implemented by using FPGA is proposed. Studying in Advanced Encryption standard algorithm and VHDL language are provided to design the system by using VHDL language.

The system can be either an encryptor or are decryptor. It receives input data and sends output data in serial synchronous mode. The system's core operates with 128 bits parallel data which provide the fast processing. The user can set the cipher key independently. This system supports for programming on FPGA besides.

สารบัญ

เรื่อง **หน้า**

สารบัญ (ต่อ)

สารบัญ (ต่อ)

เรื่อง **หน้า**

สารบัญ (ต่อ)

เรื่อง **หน้า**

สารบัญ (ต่อ)

เรื่อง **หน้า**

สารบัญรูป

รูป **หน้า**

สารบัญรูป(ต่อ)

รูป **หน้า**

บทที่ 4

สารบัญรูป(ต่อ)

รูป หน้า

บทที่ 1
บทนำ

ปัจจุบัน การใช้ชีวิตประจำวันต้องการความสะดวกสบายมากกว่าสมัยก่อน ซึ่งความสะดวกสบายเหล่านี้ ล้วนมาจากเทคโนโลยีที่ก้าวไปข้างหน้าอย่างไม่หยุดยั้ง และเทคโนโลยีทางด้านอิเล็กทรอนิกส์ก็เป็นเทคโนโลยีที่มีความก้าวหน้าไปอย่างรวดเร็ว จะเห็นได้จากเครื่องมือเครื่องใช้ต่าง ๆ ที่ต้องอาศัยชิพอิเล็กทรอนิกส์ในการสั่งให้ปฏิบัติงานให้ได้ผลตามที่ต้องการ เราจึงควรตามเทคโนโลยีที่ก้าวหน้าไปอย่างรวดเร็วนี้ให้ทัน จึงควรส่งเสริมให้มีการศึกษา วิจัย คิดค้นและออกแบบอุปกรณ์หรือวงจรอิเล็กทรอนิกส์ ซึ่งการออกแบบวงจรรวม (IC Design) ก็จัดเป็นวิธีหนึ่ง

ซึ่งในอดีตนักออกแบบวงจรดิจิตอลได้ใช้วิธีการออกแบบวงจรด้วยวิธี Bottom-up Design โดยทำกันที่ระดับ Gate-level สร้างความยุ่งยากมากให้แก่ผู้ออกแบบ ซึ่งต่อมาการออกแบบก็เริ่มซับซ้อนมากขึ้น ผู้ออกแบบจะต้องออกแบบไอซีที่มีจำนวนเกทมากกว่าเสนเกทภายในระยะเวลาอันสั้น วิธีการออกแบบที่เคยใช้กันมาเริ่มไม่ใช่วิธีที่เหมาะสมอีกต่อไป เนื่องจากยุ่งยาก เสียเวลาและมักเกิดข้อผิดพลาดขึ้นบ่อย ๆ จนกระทั่งปัจจุบัน เทคนิคการออกแบบวงจรก้าวหน้าไปมาก ผู้ออกแบบไม่จำเป็นต้องออกแบบในระดับ Gate-level เหมือนดังแต่ก่อน วิธีการออกแบบที่นิยมคือการเขียนวงจรที่ต้องการด้วยภาษาบรรยายการทำงานของวงจร (Hardware Description Language : HDL) ซึ่งสามารถตรวจสอบความถูกต้องได้ด้วยการใช้ Software Simulation จนแน่ใจว่างานที่ออกแบบนั้นมีการทำงานที่ถูกต้อง จึงนำงานนั้นไปเข้าสู่กระบวนการ Synthesis เพื่อแปลง HDL Code ที่เขียนให้อยู่ในระดับเกทซึ่งขั้นตอนดังกล่าวนั้นหากเป็นวิธีการออกแบบวงจรแบบเก่าแล้วจะเป็นงานที่เสียเวลามาก หลังจากได้วงจรในระดับเกทแล้ว จึงนำวงจรไปทำเป็นวงจรรวมหรือบันทึกลงบนอุปกรณ์ประเภท Programmable Logic Devices หรือPLD ต่อไป

ด้วยวิธีการออกแบบดังกล่าว ทำให้การออกแบบ Hardware ในปัจจุบันสามารถทำได้อย่างสะดวกรวดเร็ว ผู้ออกแบบสามารถสร้างวงจรที่มีฟังก์ชั่นการทำงานตามที่ต้องการได้ภายในระยะเวลาอันสั้น

ภาษาบรรยายการทำงานของวงจร (HDL) ในปัจจุบันที่นิยมใช้คือ VHDL พัฒนามาจากภาษา ADA ผู้พัฒนาคือ กระทรวงกลาโหมของสหรัฐอเมริกา มีโครงสร้างทางภาษาคล้ายภาษา PASCAL ทุกหน่วยงานที่ต้องการออกแบบหรือทำการพัฒนาให้กับกระทรวงนี้จะต้องใช้ภาษานี้ จึงทำให้ภาษานี้กลายเป็นภาษามาตรฐานในการนำไปออกแบบ

ภาษา VHDL มีลักษณะที่ผสมผสานกันระหว่าง Object Oriented Language และ Concurrent Programming Language โดยมองส่วนต่าง ๆ เป็นโมดูล (Module) หรือออบเจ็ก (Object) และมีสัญญาณเชื่อมต่อกันระหว่างกัน การส่งสัญญาณต่าง ๆ ในระบบเกิดขึ้นพร้อม ๆ กัน ได้เช่นเดียวกับกระแสไฟที่ไหลไปยังส่วนต่าง ๆ ของวงจร

ภาษา VHDL สามารถนำเอามาใช้ในการออกแบบวงจรได้ตั้งแต่วงจร Combination ขนาดเล็กไปจนถึงวงจรขนาดใหญ่ ซึ่งจะทำให้สามารถออกแบบวงจรที่มีความซับซ้อนได้ง่ายขึ้น อันจะส่งผลให้เกิดการพัฒนาทางด้านเทคโนโลยีในการที่จะสร้างอุปกรณ์ Hardware ใหม่ ๆ ต่อไป

1.1 ความสำคัญและที่มา

การออกแบบวงจรดิจิตอลขนาดใหญ่ที่ระดับ Gate-Level นั้นเป็นงานที่ยุ่งยากซับซ้อนมาก เป็นผลทำให้งานที่พัฒนามีความล่าช้า และเสร็จไม่ตรงตามกำหนด การนำภาษาบรรยายการทำงานของวงจรมาใช้ในการออกแบบจะช่วยแก้ไขปัญหานี้ได้

Advanced Encryption Standard (AES) นับว่าเป็นมาตรฐานใหม่ เพราะได้นำมาใช้อย่างเป็นทางการเมื่อ 2-3 ปีที่ผ่านมานี้เอง ซึ่งอัลกอริทึมของ AES สามารถทำความเข้าใจได้ไม่ยากนัก และการออกแบบโดยใช้ภาษาบรรยายการทำงานของวงจรอย่างไร จะทำให้วงจรสามารถทำงานได้อย่างรวดเร็วและมีขนาดที่เล็ก ซึ่งนับเป็นความท้าทายของงาน และผู้ออกแบบก็จะได้ประสบการณ์อันจะทำให้มีความเชี่ยวชาญในการออกแบบวงจรดิจิตอลมากขึ้น

1.2 วัตถุประสงค์

เพื่อออกแบบให้ได้ตัวเข้ารหัสและถอดรหัส ที่มีอัลกอริทึมตามมาตรฐาน Advanced Encryption Standard โดยใช้ความยาวของ Cipher Key ขนาด 128 บิต และใช้ภาษา VHDL เป็นภาษาในการบรรยายพฤติกรรมการทำงานของวงจรในการออกแบบ

1.3 ขอบเขตของโครงงาน

1. ศึกษาและทำความเข้าใจอัลกอริทึมของ AES ซึ่งจะต้องใช้ในการออกแบบและสร้างตัวเข้ารหัสและถอดรหัส
2. ศึกษาและทำความเข้าใจภาษา VHDL ที่จะใช้บรรยายการทำงานของวงจร
3. ออกแบบตัวเข้ารหัสและถอดรหัส โดยใช้ภาษา VHDL เพื่อให้ได้ตามวัตถุประสงค์

บทที่ 2

Advanced Encryption Standard

มาตรฐานนี้จะกล่าวถึง Rijndael อัลกอริทึมซึ่งเป็น Block Cipher* แบบสมมาตร ที่สามารถประมวลผลข้อมูล (Data block) ขนาด 128 บิต โดยใช้ Cipher Key* ขนาด 128,192 หรือ 256 บิต นอกจากนี้แล้ว Rijndael อัลกอริทึมก็ถูกออกแบบมาให้ทำงานกับข้อมูลขนาดอื่น และ Cipher Key ขนาดอื่นได้ด้วย แต่ไม่ได้ถูกบรรจุอยู่ในมาตรฐานนี้

สำหรับมาตรฐานนี้คำว่า "อัลกอริทึม" ที่ได้กล่าวถึงจะหมายถึง "AES อัลกอริทึม" และอัลกอริทึมนี้สามารถอ้างถึงได้ 3 แบบ ตามขนาดของ Cipher Key 3 ขนาด คือ "AES-128 " , "AES-192 " และ "AES-256 "

ในบทนี้จะได้กล่าวถึงหัวข้อต่าง ๆ ดังต่อไปนี้

1. คำนิยาม, ชื่อย่อ, พารามิเตอร์, สัญลักษณ์ และฟังก์ชั่นต่าง ๆ ที่ใช้ในมาตรฐานนี้
2. ข้อกำหนดที่ใช้ในการอธิบายรายละเอียดของอัลกอริทึม
3. คุณสมบัติทางคณิตศาสตร์ ซึ่งมีประโยชน์ต่อการทำความเข้าใจอัลกอริทึม
4. รายละเอียดของอัลกอริทึมซึ่งครอบคลุมถึง Key expansion , การเข้าสลัก และการถอดสลัก

2.1 คำนิยาม

2.1.1 ชื่อย่อ และอภิธานศัพท์

AES	Advanced Encryption Standard
Affine Transformation	กระบวนการแปลงที่ประกอบด้วยการคูณด้วยเมตริกซ์ แล้วตามด้วย การบอกด้วยเวคเตอร์
Block	ชุดแถวของเลขฐาน 2 (มีลักษณะนามเป็นบิต) ซึ่งประกอบด้วย อิน พุท, เอาท์พุท, State* และ Round Key* ความยาวของชุดแถวก็คือ จำนวนบิตที่มีอยู่ในชุดแถวนั้นนั่นเอง หรืออีกนัยหนึ่ง Block ก็คือ อาเรย์ของ Byte
Byte	จำนวน 8 บิต

* อธิบายไว้ที่หัวข้อ 2.1

Cipher	อนุกรมของการแปลง ซึ่งเปลี่ยน plaintext ไปเป็น Cipher text โดยใช้ Cipher Key
Cipher Key	กุญแจลับซึ่งจะนำไปใช้ในกระบวนการ Key Expansion เมื่อสร้างชุดของ Round Key ซึ่งสามารถแสดงเป็นรูปภาพ เป็นอาเรย์ของ Byte รูปสี่เหลี่ยมผืนผ้ามี 4 แถว และ Nk หลัก
Cipher text	ข้อมูลเอาท์พุทของ Cipher หรือเป็นอินพุทของ Inverse Cipher
Inverse Cipher	อนุกรมของการแปลงซึ่งเปลี่ยน Cipher text ไปเป็น plain text โดยใช้ Cipher Key
Key Expansion	กระบวนการที่ใช้สร้าง Round Key จาก Cipher Key
Plane text	ข้อมูลอินพุทของ Cipher หรือเป็นเอาท์พุทของ Inverse Cipher
Rijndael	อัลกอริทึมเกี่ยวกับรหัสลับซึ่งกล่าวถึงในมาตรฐาน (Advanced Encryption Standard) นี้
State	ผลลัพธ์ระหว่างกลางภายใน Cipher ซึ่งสามารถขยายโดยใช้อาเรย์ของ Byte รูปสี่เหลี่ยมผืนผ้าซึ่งมี 4 แถว และ Nb หลัก
S-box	ตารางการสับเปลี่ยนแบบไม่เป็นเชิงเส้น ซึ่งใช้ในกระบวนการแปลงแบบสับเปลี่ยนไบต์ (Byte substitution transformation) และในกระบวนการ Key Expansion เพื่อแสดงการสับเปลี่ยนค่าของ Byte แบบหนึ่งต่อหนึ่ง
Word	32 บิต หรือ 4 Byte

2.1.2 พารามิเตอร์ของอัลกอริทึม, สัญลักษณ์และฟังก์ชั่น

AddRoundKey ( )	การแปลงใน Cipher และ Inverse Cipher โดย Round Key จะถูกนำไปบวกกับ State โดยการ XOR ความยาวของ Round Key จะเท่ากับขนาดของ State (เช่น ถ้า Nb = 4 ความยาวของ Round Key จะเท่ากับ 128 บิต/16ไบต์)
InvMixColumns ( )	การแปลงใน Iinverse Cipher ซึ่งตรงกันข้ามกับ MixColumn()
InvShiftRows ( )	การแปลงใน Inverse Cipher ซึ่งตรงกันข้ามกับ ShiftRows ( )
InvSubBytes ( )	การแปลงใน Inverse Cipher ซึ่งตรงกันข้ามกับ SubByte ( )
K	Cipher Key

MixColumns ( )	การแปลงใน Cipher ซึ่งนำข้อมูลของ State ทุกหลักมาผสมกันหรือสร้างข้อมูลหลักใหม่ขึ้นมา
Nb	จำนวนหลัก (ซึ่งมี 32 บิต) ที่ประกอบขึ้นเป็น State.สำหรับมาตรฐานนี้ Nb = 4
Nk	จำนวน Word (32 บิต) ที่ประกอบขึ้นเป็น Cipher Key สำหรับมาตรฐานนี้ Nk = 4,6 หรือ 8
Nr	จำนวนรอบซึ่งเป็นฟังก์ชันกับ Nk (ซึ่งคงที่) สำหรับมาตรฐานนี้ Nr = 10,12 หรือ14
Rcon [ ]	อาเรย์ของWord ซึ่งมีค่าคงที่สำหรับแต่ละรอบ
RotWord ( )	ฟังก์ชันในกระบวนการ Key Expansion ซึ่งจะนำข้อมูล 4 byte ไปทำการสับเปลี่ยนแบบวนรอบ สามแถวสุดท้ายของ State แบบ วนรอบ และแต่ละแถวจะเลื่อนไม่เท่ากัน
SubBytes ( )	การแปลงใน Cipher ซึ่งทำการประมวลผล State โดยใช้ S-box แทนที่แต่ละไบต์ของ State อย่างอิสระ
SubWord ( )	คือฟังก์ชันที่ถูกใช้ในกระบวนการ Key Expansion ซึ่งจะนำข้อมูลมาอินพุท 4 ไบต์ ไปสับเปลี่ยนที่ S-box เหมือนกับ SubByte เพื่อสร้าง Word เอาท์พุท
XOR	การปฏิบัติการแบบเอ็กคลูซีฟออร์
$\oplus$	ตัวปฏิบัติการแบบเอ็กคลูซีฟออร์
$\otimes$	การคูณของโพลิโนเมียล 2 ชุด (แต่ละชุดมีดีกรี < 4) แล้วหารเอาเศษด้วย $x^4 + 1$
•	การคูณแบบสนามจำกัด (Finite field multiplication)

2.2. ข้อกำหนด

2.2.1 อินพุทและเอาท์พุท

อินพุทและเอาท์พุทสำหรับ AES อัลกอริทึมประกอบด้วยชุดแถวของเลขฐาน 2 จำนวน 128 บิต ซึ่งในบางครั้งเราอาจเรียกชุดแถวเหล่านี้ว่า "Block" และจำนวนบิตที่ประกอบขึ้นเป็น Block จะเรียกว่าความยาวของ Block Cipher Key สำหรับ AES อัลกอริทึมคือชุดแถวของเลข

ฐาน 2 จำนวน 128, 192 หรือ 256 บิต และสำหรับ มาตรฐานนี้จะไม่อนุญาตให้ใช้ขนาดของอินพุท เอาท์พุทและ Cipher Key ขนาดอื่น

การอ้างถึงบิตที่อยู่ในชุดแถวจะเริ่มนับตั้งแต่ 0 จนถึง ความยาวของชุดแถว–1 โดยใช้ i เป็นดัชนีในการอ้างถึง เพราะฉะนั้น $0 \leq i < 128$, $0 \leq i < 192$ หรือ $0 \leq i < 256$ ขึ้นอยู่กับความยาว Cipher Key

2.2.2 ไบต์

หน่วยพื้นฐานสำหรับการประมวลผลใน AES อัลกอริทึมคือไบต์ อินพุท,เอาท์พุทและ Cipher Key ก็จะถูกประมวลผลในลักษณะของไบต์ โดยชุดแถวข้อมูลซึ่งเป็นเลขฐาน 2 อินพุท เอาท์พุท และCipher Key จะถูกจัดเป็นกลุ่ม ๆ ละ 1 ไบต์เพราะฉะนั้นชุดแถวเหล่านี้มีลักษณะเป็นไบต์อาเรย์ โดยมีสัญลักษณ์แทนไบต์อาเรย์เหล่านี้ 2 แบบ คือ a_n หรือ a[n] โดยที่ a คือ อินพุท , เอาท์พุท หรือ Cipher Key และn เป็นดัชนีในการอ้างถึง ซึ่งมีขอบเขตดังนี้

ความยาวของ Cipher Key = 128 บิต , $0 \leq n < 16$;

ความยาวของ Cipher Key = 192 บิต , $0 \leq n < 24$;

ความยาวของ Cipher Key = 256 บิต , $0 \leq n < 32$;

ความยาวของ Block = 128 บิต, $0 \leq n < 16$;

ค่าของแต่ละไบต์ใน AES อัลกอริทึมสามารถที่จะแสดงโดยใช้โพลิโนเมียลในลักษณะหน่วยของสนามจำกัด(Finite field elements) ได้

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (2.1)$$

ยกตัวอย่าง เช่น {01100011} จะแสดงเป็นหน่วยของสนามจำกัด คือ $x^6 + x^5 + x + 1$

นอกจากนั้นแล้วยังสามารถที่จะระบุค่าของแต่ละไบต์ (ซึ่งเป็นเลขฐาน 2) โดยใช้ตัวเลขฐาน 16 (4 บิตจะแทนด้วยเลขฐาน 16 หนึ่งตัว) ดังรูปที่ 2.1

Bit Pattern	Character
0000	0
0001	1
0010	2
0011	3

Bit Pattern	Character
0100	4
0101	5
0110	6
0111	7

Bit Pattern	Character
1000	8
1001	9
1010	a
1011	b

Bit Pattern	Character
1100	c
1101	d
1110	e
1111	f

รูปที่ 2.1 การใช้เลขฐาน 16 แทนเลขฐาน 2

ดังนั้น {01100011} จะถูกแทนด้วย {63}

การปฏิบัติการบางอย่างของสนามจำกัด จะต้องรวมบิตที่ 9 (b_8) เข้าไปด้วย ซึ่งเมื่อใดที่บิตนี้ปรากฏขึ้น การแทนค่าของไบต์จะมี "{01}" ปรากฏขึ้นนำหน้าค่าของไบต์ เช่น {01} {16}

2.2.3 ไบต์อาเรย์

ไบต์อาเรย์สามารถเขียนแทนรูปแบบดังนี้

$$a_0\ a_1\ a_2 \ldots\ a_{15}$$

และจากชุดแถวของอินพุทขนาด 128 บิต มีรูปแบบเป็น

$$input_0\ input_1\ input_2 \ldots input_{126}\ input_{127}$$

ไบต์อาเรย์จึงมีรูปแบบดังนี้

$$a_0 = \{ input_0, input_1, \ldots, input_7 \}$$

$$a_1 = \{ input_8, input_9, \ldots, input_{15} \}$$

.

.

.

$$a_{15} = \{ input_{120}, input_{121}, \ldots, input_{127} \}$$

รูปแบบเหล่านี้สามารถใช้กับชุดแถวที่ยาวขึ้นได้ (เช่น 192 บิต และ 256 บิต)

ดังนั้นจะได้รูปแบบทั่วไป คือ

$$a_n = \{ input_{8n}, input_{8n+1}, \ldots, input_{8n+7} \} \qquad (2.2)$$

Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
Byte number	0								1								...
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

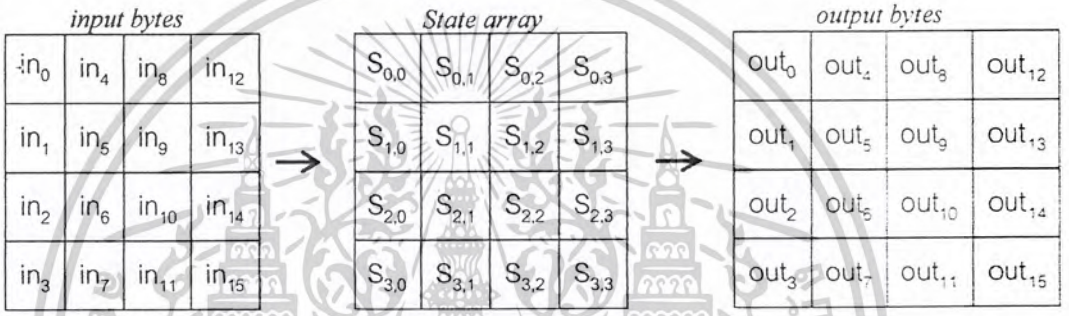
รูปที่ 2.2 ดัชนีสำหรับไบต์และบิตในชุดแถวข้อมูล

2.2.4 State

การปฏิบัติการของ AES อัลกอริทึมนั้นจะแดสงในรูปของไบต์อาเรย์ 2 มิติ ซึ่งเรียกว่า State อันประกอบด้วย 4 แถว แต่ละแถวจะมี Nb เท่ากับความยาวของ Block หารด้วย 32 ภายใน State อาเรย์จะใช้สัญลักษณ์ S แทน แต่ละไบต์และมีดัชนี 2 ตัวเป็นการชี้ตำแหน่ง กล่าวคือ $S_{r,c}$

หรือ S[r,c] โดย r และ c จะบอกถึงแถว (เนื่องจากเป็น 2 มิติ) และหลักของไบต์นั้น ๆ และ r จะมีขอบเขตอยู่ในช่วง $0 \leq r < 4$ ส่วน c จะมีขอบเขตอยู่ในช่วง $0 \leq c < Nb$ สำหรับมาตรฐานนี้ Nb = 4 เพราะฉะนั้น $0 \leq c < 4$

ที่ตำแหน่งเริ่มต้นของ Cipher และ Inverse Cipher ซึ่งจะกล่าวถึงในหัวข้อที่ 2.4 ข้อมูลอินพุทซึ่งเป็นไบต์อาเรย์ $in_0, in_1, \ldots, in_{15}$ จะถูกคัดลอกไปยัง State อาเรย์ดังแสดงในรูปที่ 2.3 ดังนั้นในกระบวนการ Cipher และ Inverse Cipher ก็จะนำ State อาเรย์นี้ไปใช้และ State อาเรย์สุดท้ายก็จะถูกคัดลอกไปยังเอาท์พุท ซึ่งเป็นไบต์อาเรย์ $out_0, out_1, \ldots out_{15}$



รูปที่ 2.3 อินพุทและเอาท์พุท State อาเรย์

ดังนั้นที่ตำแหน่งเริ่มต้นของ Cipher หรือ Inverse Cipher อาเรย์ อินพุท in จะถูกคัดลอกไปยัง State อาเรย์ตามรูปแบบต่อไปนี้

$$3[r,c] = in[r+4c] \text{ สำหรับ } 0 \leq r < 4 \text{ และ } 0 \leq c < Nb \quad (2.3)$$

และที่จุดสุดท้ายของ Cipher หรือ Inverse Cipher State อาเรย์ ก็จะถูกคัดลอกไปยังเอาท์พุทอาเรย์ out ตามรูปแบบต่อไปนี้

$$Out[r+4c] = 5[r,c] \text{ สำหรับ } 0 \leq r < 4 \text{ และ } 0 \leq c < Nb \quad (2.4)$$

2.2.5 State ในรูปแบบหลักอาเรย์

ในแต่ละหลักของ State จะมี 4 ไบต์ซึ่งก็คือ 1 Word นั่นเอง โดย r จะเป็นดัชนีสำหรับ 4 ไบต์ที่อยู่ในแต่ละ Word ดังนั้น State สามารถที่จะแสดงได้ในรูปแบบของอาเรย์ 1 มิติ ของ Word ได้คือ $W_0 \ldots W_3$ ดังรูปที่ 2.4 ซึ่งจะเห็นว่า c คือดัชนีของอาเรย์นี้

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

W_0 W_1 W_2 W_3

รูปที่ 2.4 State อาเรย์ในรูปแบบหลักอาเรย์

2.3 หลักการทางคณิตศาสตร์เบื้องต้น

จากหัวข้อที่ 2.2.2 ทุก ๆ ไบต์ใน AES อัลกอริทึม จะถูกอธิบายโดยใช้ หน่วยในสนามจำกัด หน่วยในสนามจำกัดสามารถบวกและคูณกันได้ แต่ปฏิบัติการเหล่านี้จะต่างไปจากตัวเลขทั่วไป หัวข้อต่อไปนี้จะกล่าวถึง พื้นฐานทางคณิตศาสตร์ที่จำเป็นสำหรับหัวข้อที่ 2.4

2.3.1 สนาม GF (2^8)

2.3.1.1 การบวก

การบวกของหน่วยในสนามจำกัด 2 หน่วย กระทำโดยนำ โพลิโนเมียลที่มีกำลังเท่ากันมาบวกกันแล้วนำสัมประสิทธิ์ที่ได้จากการบวกกันนั้นมา mod ด้วย 2 ก็จะได้สัมประสิทธิ์ของโพลิโนเมียลผลลัพธ์ออกมา การบวกดังกล่าวแสดงได้ด้วยการทำ XOR (ใช้สัญลักษณ์$\oplus$) ยกตัวอย่าง $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$ สิ่งที่ได้ตามมาก็คือการลบกันของโพลิโนเมียลจะเหมือนกับการบวกทุกประการ

อีกนัยหนึ่งการบวกกันของหน่วยในสนามจำกัดสามารถอธิบายโดยใช้การบวกกันของบิตที่ตรงกันแล้วmodด้วย 2 สำหรับ 2 ไบต์ $\{a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0\}$ และ $\{b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\}$ ผลบวกคือ $\{c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0\}$ โดย $c_i = a_i \oplus b_i$ (ยกตัวอย่างเช่น $c_7 = a_7 \oplus b_7$, $c_6 = a_6 \oplus b_6, \ldots, c_0 = a_0 \oplus b_0$)

จากที่กล่าวมาข้างต้น รูปแบบที่แสดงต่อไปนี้มีความหมายเหมือนกัน

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$ (รูปแบบโพลิโนเมียล)

$\{0101011\} \oplus \{10000011\} = \{11010100\}$ (รูปแบบเลขฐาน 2)

$\{57\} \oplus \{83\} = \{d4\}$ (รูปแบบเลขฐาน 16)

2.3.1.2 การคูณ

การคูณกันของโพลิโนเมียลใน GF (2^8) (ใช้สัญลักษณ์ •) คือ การคูณกันของโพลิโนเมียลแล้ว mod ด้วยโพลิโนเมียลดีกรี 8 ซึ่งไม่มีโพลิโนเมียลใดนอกจากตัวมันเองและ 1 หารได้ลงตัว และสำหรับ AES อัลกอรีทึม โพลิโนเมียล นี้คือ

$$m(x) = x^8 + x^4 + x^3 + x + 1 \qquad (2.5)$$

หรือ { 01 } { 1b } ในรูปแบบเลขฐาน 16

ตัวอย่างเช่น $\{57\} \bullet \{83\} = \{c1\}$ เพราะ

$$(x^6 + x^4 + x^3 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

และ

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \text{ modulo } (x^8 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + 1$$

การ mod ด้วย m (x) จะทำให้ผลลัพธ์เป็น โพลิโนเมียล ดีกรีน้อยกว่า 8 ดังนั้นจึงสามารถแทนด้วยเลขฐาน 2 ขนาด 1 ไบต์ได้

การคูณที่นิยามไว้ข้างต้น มี { 01 } เป็นเอกลักษณ์ทางการคูณ สำหรับโพลิโนเมียล b (x) ที่ไม่เป็นศูนย์และมีดีกรีน้อยกว่า 8 สามารถหาอินเวอร์สทางการคูณ ซึ่งใช้สัญลักษณ์คือ b^{-1} (x)ดีกรีโดยการใช้ Euclidean อัลกอริทึมในการคำนวณโพลิโนเมียล a(x) และ c (x) โดยที่

$$b(x)\,a(x) + m(x)\,c(x) = 1 \qquad (2.6)$$

ดังนั้น $a(x) \bullet b(x) \bmod m(x) = 1$ จะได้

$$b^{-1}(x) = a(x) \bmod m(x) \qquad (2.7)$$

นอกจากนั้นสำหรับ a (x) , b (x) และ c (x) ใด ๆ ในสนาม (filed) จะมีคุณสมบัติดังต่อไปนี้

$$a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$$

จากข้อมูลข้างต้นสิ่งที่ได้ตามมาก็คือข้อมูล 1 ไบต์ ซึ่งสามารถมีค่าเป็นไปได้ 256 ค่าที่ใช้ XOR เป็นการบวกและใช้การคูณดังที่นิยามไว้ข้างต้น มีโครงสร้างของสนามจำกัด GF (2^8)

2.3.1.2.1 **การคูณด้วย x**

การคูณ โพลิโนเมียลในสมการที่ (2.1) ด้วย โพลิโนเมียล x จะได้ผลลัพธ์เป็น

$$b_7 x^8 + b_6 x^7 + b_5 x^6 + b_4 x^5 + b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x \qquad (2.8)$$

ผลจากการคูณด้วย x หรือ x • b (x) นี้จะถูกลดรูปให้มีดีกรีน้อยกว่า 8 m (x) ที่ระบุในสมการที่ (2.1) ผลลัพธ์จะถูกลดรูปอยู่แล้วถ้า $b_7 = 0$ แต่ถ้า $b_7 = 1$ จะต้องทำการลดรูปโดยลบ (XOR) ด้วย โพลิโนเมียล m(x)

ดังนั้นการคูณด้วย x หรือ {00000010} สำหรับเลขฐาน 2 หรือ {02} สำหรับเลขฐาน 16 สามารถสร้างขึ้นจริงในระดับไบต์ได้โดยการเลื่อนบิตไปทางซ้ายแล้วคูณจากการเลื่อนดังกล่าว ถ้า โอเวอร์โฟลวหรือบิตที่ 8 ไม่เกิดขึ้นก็ไม่ต้องทำอะไรแต่ถ้าเกิดขึ้นต้องทำการ XOR กับ{1b} อีกทีหนึ่ง การกระทำดังกล่าวกับไบต์ใด ๆ จะใช้สัญลักษณะ xtime ( ) ซึ่งถ้าเกิดการคูณด้วย x ที่มีกำลังมากกว่าหนึ่งสามารถสร้างโดยใช้ xtime ( ) ซ้ำ ๆ กันหลายครั้งและการคูณด้วยค่าคงที่ใด ๆ ก็สามารถช่วยสร้างได้โดยการนำผลลัพธ์จาก xtime ( ) มาบวกกัน ดังตัวอย่าง ข้างล่างนี้

ตัวอย่าง {57} • {13} = {fe} เพราะว่า

{57} • {02} = xtime ({57}) ={ae}

{57} • {04} = xtime({ae}) ={47}

{57} • {08} = xtime ({47}) ={8e}

{57} • {10} = xtime ({8e}) ={07}

ดังนั้น {57} • {13} = {57} • ({01}⊕{02}⊕{10})

= {57} ⊕ {ae} ⊕ {07}

= {fe}

2.3.2 **โพลิโนเมียล ที่มีสัมประสิทธิ์ใน GF (2^8)**

โพลิโนเมียล สามารถกำหนดให้มีสัมประสิทธิ์ใน GF (2^8) ได้โดยทำให้เป็นโพลิโนเมียล ที่มี 4 พจน์ดังนี้

$$a(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0 \qquad (2.9)$$

ซึ่งใช้แสดงถึง 4 ไบต์ หรือ1 Word ในรูปของ $[a_0, a_1, a_2, a_3]$

โพลิโนเมียล ที่จะได้กล่าวถึงในหัวข้อนี้จะมีบางอย่างที่แตกต่างไปจากโพลิโนเมียลที่ใช้ในนิยามของหน่วยในสนามจำกัด แต่โพลิโนเมียล ทั้งสองชนิดนี้จะใช้ตัวแปรเหมือนกันคือ x สัมประสิทธิ์ในหัวข้อนี้เป็นหน่วยในสนามจำกัดโดยตัวเอง สัมประสิทธิ์ เป็นไบต์แทนที่จะเป็นบิต และการคูณของโพลิโนเมียล 4 พจน์นี้ใช้ โพลิโนเมียล ในการลดรูปต่อไปจากเดิม

เพื่ออธิบายเกี่ยวกับการบวกและการคูณ กำหนดให้

$$b(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0 \quad (2.10)$$

การบวกทำได้โดยนำสัมประสิทธิ์ของ x ที่มีกำลังเท่ากันมาบวกกันและการบวกนี้จะใช้ XOR เป็นตัวปฏิบัติการ ดังนั้นจากสมการที่ (2.9)และ(2.10) จะได้ว่า

$$a(x) + b(x) = (a_3 \oplus b_3) x^3 + (a_2 \oplus b_2) x^2 + (a_1 \oplus b_1) x + (a_0 \oplus b_0) \quad (2.11)$$

ในส่วนของการคูณนั้นจะต้องทำ 2 ขั้นตอน กล่าวคือ ขั้นตอนแรก ผลคูณของโพลิโนเมียล $c(x) = a(x) \bullet b(x)$ โดยที่

$$c(x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 \quad (2.12)$$

เมื่อ

$$\begin{aligned} c_0 &= a_0 \bullet b_0 & c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 & c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 & c_6 &= a_3 \bullet b_3 \\ c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \end{aligned} \quad (2.13)$$

จะเห็นว่าผลลัพธ์ c(x) ไม่ได้อยู่ในรูปแบบ 4 ไบต์หรือ Word ดังเช่น a(x) และ b(x) ดังนั้น ขั้นตอนที่สองก็คือขั้นตอนในการลดรูป c(x) ให้อยู่ในรูปแบบ 4 ไบต์หรือ Word โดยการ mod ด้วย โพลิโนเมียล ดีกรี 4 ดังนั้นผลลัพธ์ก็จะได้โพลิโนเมียล ดีกรีน้อยกว่า 4 ซึ่งสำหรับ AES อัลกอริทึม โพลิโนเมียล ดีกรี 4 นี้ คือ $x^4 + 1$ ดังนั้น

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4} \quad (2.14)$$

การคูณแบบ modular ระหว่าง a(x) และ b(x) ซึ่งใช้สัญลักษณ์ $a(x) \otimes b(x)$ จะให้โพลิโนเมียล 4 พจน์ ออกมาคือ

$$a(x) \otimes b(x) = d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0 \quad (2.15)$$

เมื่อ

$$\begin{aligned} d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3 \\ d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \end{aligned} \quad (2.16)$$

เมื่อ a(x) เป็นโพลิโนเมียลคงที่ การกระทำตามสมการที่ (2.15) สามารถจะเขียนในรูปของ เมตริกซ์ได้คือ

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (2.17)$$

เนื่องจาก x^4+1 ไม่ใช่โพลิโนเมียล ที่ไม่มีโพลิโนเมียลใดหารลงตัว บน GF (2^8) การคูณด้วยโพลิโนเมียลคงที่ที่มี 4 พจน์ จึงไม่จำเป็นต้องอินเวอร์สได้ อย่างไรก็ตาม AES อัลกอริทึมได้ระบุโพลิโนเมียลคงที่ 4 พจน์ ซึ่งมีอินเวอร์สไว้คือ

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (2.18)$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \quad (2.19)$$

สำหรับโพลิโนเมียล อื่น ๆ ที่ใช้ใน AES อัลกอริทึมจะมี $a_0 = a_1 = a_2$ {00} และ $a_3 =$ {01} ซึ่งก็คือโพลิโนเมียล x^3 นั่นเอง

2.4 รายละเอียดของอัลกอริทึม

สำหรับ AES อัลกอริทึมความยาวของ Block อินพุท, Blockเอาท์พุทเพราะState คือ 128 บิต ซึ่งถูกเขียนแทนด้วย Nb = 4 ซึ่งจะส่งผลต่อจำนวนหลักใน State

สำหรับ AES อัลกอริทึมความยาวของ Cipher Key คือ 128,192 หรือ 296 บิต ความพยายามของ Cipher Key เขียนแทนด้วย Nk = กับ 4,6 หรือ 8 ซึ่งส่งผลต่อจำนวน Word หรือจำนวนหลักใน Cipher Key

สำหรับ AES อัลกอริทึมจำนวนรอบในการทำงานตามอัลกอริทึมขึ้นกับขนาดของ Cipher Key จำนวนรวมเขียนแทนด้วย Nr โดยที่ Nr = 10 เมื่อ Nk = 4, Nr = 12 , เมื่อ Nk = 6, และ Nr = 14 เมื่อ Nk = 8

สำหรับความยาวของขนาด Block และจำนวนรหัสสอดคล้องกับมาตรฐานนี้ แสดงดังรูปที่ 2.5

	ความยาวของ Cipher Key (Nk words)	ขนาดของ Block (Nb words)	จำนวนรอบ (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

รูปที่ 2.5 ความยาว Key , ขนาดของ Block และจำนวนรอบ ที่สอดคล้องกัน

ทั้ง Cipher และ Inverse Cipher AES อัลกอริทึมจะใช้ฟังก์ชั่นรอบ ซึ่งประกอบด้วยการแปลง 4 อย่างคือ 1.การสับเปลี่ยนไบต์โดยใช้ S-box 2. การเลื่อนแถวใน State อาเรย์ ซึ่งแต่ละแถวจะถูกเลื่อนไม่เท่ากัน 3. การผสมข้อมูลในแต่ละหลักของ State อาเรย์ และ 4.การบวก Round Key กับ State การแปลงทั้ง 4 แห่งนี้จะได้อธิบายในหัวข้อ 2.4.1.1 – 2.4.1.4 และ 2.4.3.1 – 2.4.3.4

2.4.1 Cipher

ที่จุดเริ่มต้นของ Cipher ข้อมูลอินพุทจะถูกคัดลอกไปยัง State อาเรย์ ดังที่ได้อธิบายในหัวข้อ 2.2.4 หลังจากที่บวก Round Key แรก (Initial Round Key) และ State อาเรย์ จะถูกนำไปทำการแปลง โดยการใช้ฟังก์ชันรอบ 10,12หรือ 14 รอบ ขึ้นกับความยาวของ Cipher Key และรอบสุดท้ายจะต่างจาก Nr - 1 รอบแรก กล่าวคือในแต่ละรอบ State จะผ่านการแปลง 4 อย่าง คือ SubBytes ( ) , ShiftRows ( ), MixColumns ( ), และ AddRoundKey ( ) แต่สำหรับรอบสุดท้ายจะมีเพียง 3 อย่าง คือ SubBytes ( ) , ShiftRows ( ) และ AddRoundKey ( ) เท่านั้น สุดท้าย State จะถูกคัดลอกไปยังเอาท์พุทดังที่ได้อธิบายในหัวข้อ 2.2.4

2.4.1.1 การแปลงแบบ SubBytes( )

การแปลงแบบ SubBytes( ) เป็นการสับเปลี่ยนไบต์แบบไม่เป็นเชิงเส้นซึ่งทำงานกับแต่ละไบต์ของ State อย่างอิสระ โดยใช้ S-box S-box ซึ่งสามารถอินเวอร์สได้ ถูกสร้างโดยใช้การแปลง 2 อย่าง คือ

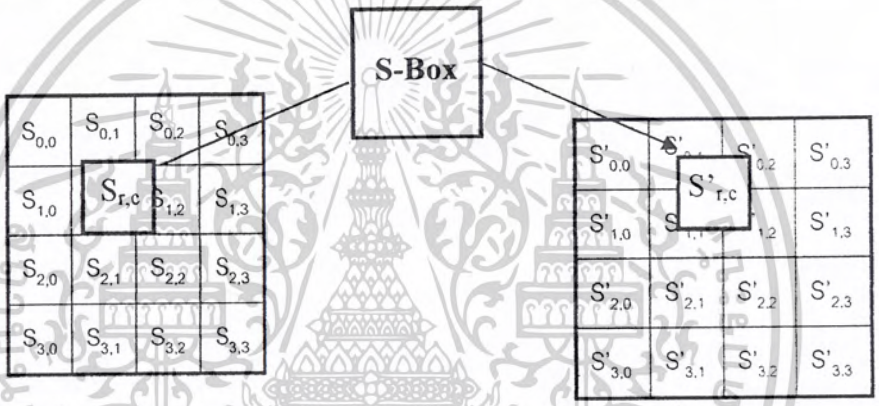
1. หาค่าอินเวอร์สทางการคูณในสนามจำกัด GF (2^8) ดังที่ได้อธิบายไว้ในหัวข้อ 2.3.1.2 และอินเวอร์สของ {00} คือ {00}
2. ทำการแปลงโดยใช้ Affine transformation (บน GF (2^8)) :

$$b_i' = b_i \oplus_{(i+4)mod8} \oplus_{(i+5)mod8} \oplus_{(i+6)mod8} \oplus_{(i+7)mod8} \oplus c_i \qquad (2.20)$$

โดยที่ b_i คือ บิตที่ i ของไบต์ และ c_i คือบิตที่ i ของ c ซึ่งมีค่า {63} หรือ {01100011} Affine transformation ดังสมการที่ (2.20) สามารถเขียนในรูปของเมตริกซ์ได้ด้วย

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (2.21)$$

รูปที่2.6 แสดงผลของการแปลงแบบ SubByte(ของ State)



รูปที่ 2.6 การแปลงแบบ SubBytes( )

S-box ซึ่งใช้ในการแปลงแบบ SubByte จะแสดงในรูปของตัวเลขฐาน 16 ดังรูปที่ 2.7 ตัวอย่างเช่น ถ้า $S_{1,1}$ = {53} ค่าที่ได้จากการสับเปลี่ยนนี้หาได้จาก แถวเลข "5" และหลักเลข "3" ดังในรูปที่ 2.7 เพราะฉะนั้น $S_{1,1}$ คือ {ed}

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	a	65	7a	a	08
	c	ba	78	25	2e	1c	a6	b4	c6	8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	el	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

รูปที่ 2.7 S-Box

2.4.1.2 การแปลงแบบ ShiftRows ( )

ในการแปลงแบบ ShiftRows ( ) ไบต์ใน 3 แถวสุดท้ายของ State จะถูกเลื่อนแบบวงกลมด้วยค่าที่ไม่เท่ากัน และแถวแรก (r = 0) จะไม่ถูกเลื่อน

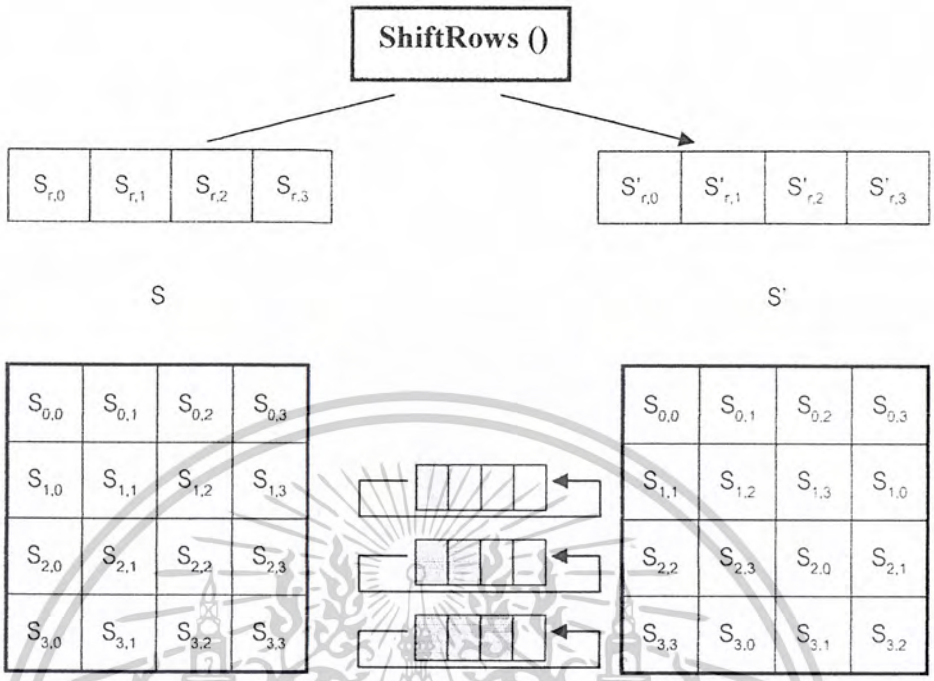
การทำการแปลงแบบ ShiftRows ( ) สามารถเขียนได้ดังนี้

$$S'_{r,c} = S_r ,(C + Shift (r,Nb)) \text{ mod } Nb \text{ สำหรับ } 0 < r < 4 \text{ และ } 0 \leq c < Nb \qquad (2.22)$$

โดยที่ ค่าในการเลื่อนหรือ shift (r,Nb) ขึ้นกับ r ดังนี้ (สำหรับ Nb = 4)

$$\text{Shift (1, 4) = 1 , shift (2, 4) = 2 เพราะ shift (3, 4) = 3} \qquad (2.23)$$

การทำเช่นนี้จะทำให้เกิดการเคลื่อนย้ายไบต์ไปตำแหน่งที่ต่ำกว่าในแถวเดียวกัน (ค่า c ที่ต่ำกว่า) ในขณะที่ไบต์ที่อยู่ต่ำสุดจะขึ้นไปอยู่สูงสุดแทน การแปลงแบบ ShiftRows() แสดงดังรูปที่2.8



รูปที่ 2.8 การแปลงแบบ ShiftRows ( )

2.4.1.3 **การแปลงแบบ** MixColumns ( )

การแปลงแบบ MixColumns ( ) กระทำกับ State แบบหลักต่อหลัก โดยพิจารณาให้แต่ละหลักเป็น โพลิโนเมียล 4 พจน์ ดังที่ได้อธิบายไว้ในหัวข้อที่ 2.3.2

แต่ละหลักใน State จะถูกพิจารณาเป็นโพลิโนเมียล บน GP (2^8) แล้วคูณแบบ modular(mod ด้วย x^4+1) ด้วย a(x) ตามสมการที่ (2.23)

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \qquad (2.23)$$

ดังที่ได้อธิบายในหัวข้อ 2.3.2 การคูณแบบ modular นี้สามารถเขียนเป็นการคูณแบบเมตริกซ์ได้ ถ้าให้ $S'(x) = a(x) \otimes S(x)$ เขียนเป็นเมตริกซ์ได้เป็น

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{สำหรับ } 0 \le c < Nb \qquad (2.24)$$

จากการคูณแบบเมตริกซ์ข้างต้นจะได้

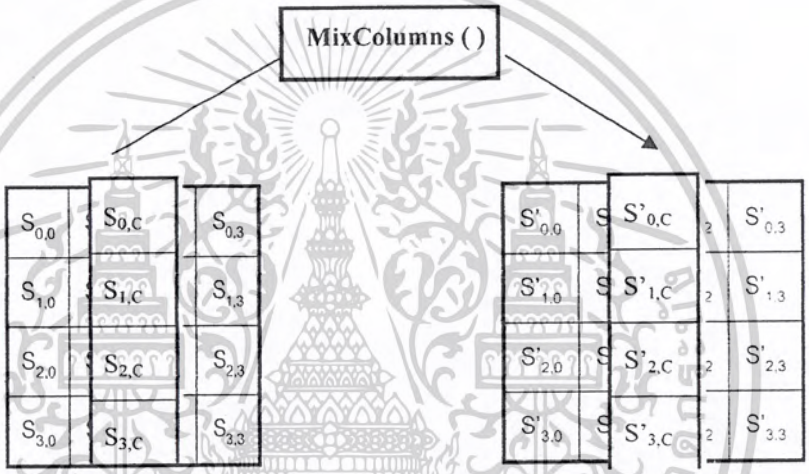
$$S'_{0,c} = (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

$$S'_{1,c} = S_{0,c} \oplus (\{02\} \bullet S_{1,c}) \oplus (\{03\} \bullet S_{2,c}) \oplus S_{3,c}$$

$$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \bullet S_{2,c}) \oplus (\{03\} \bullet S_{3,c})$$

$$S'_{3,c} = (\{03\} \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \bullet S_{3,c})$$

การแปลงแบบ MixColumns แสดงดังรูปที่ 2.9



รูปที่ 2.9 การแปลงแบบ MixColumns ( )

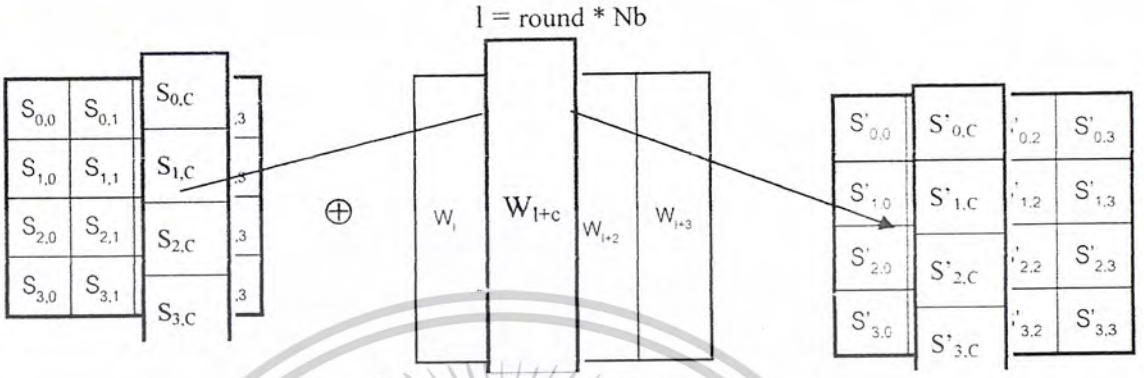
2.4.1.4 การแปลงแบบ AddRoundKey ( )

ในการแปลงแบบ AddRoundKey ( ) Round Key จะถูกบวก เข้ากับ State โดยการ XOR Round Key แต่ละชุดประกอบด้วย Nb Word ซึ่งได้มาจากกระบวนการ Key Expansion แต่ละ Nb Word เหล่านี้จะถูกนำไปบวกกับแต่ละหลักของ State กล่าวคือ

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [W_{round * Nb+C}] \quad \text{สำหรับ } 0 \leq c < Nb \qquad (2.25)$$

โดยที่ $[W_i]$ คือ Key 1 Word จากตารางKey ซึ่งได้กล่าวถึงในหัวข้อต่อไป และ round คือค่าที่อยู่ในช่วง $0 \leq$ round $\leq$ Nr ใน Cipher การบวก Round Key แรก จะเกิดขึ้นเมื่อ round = 0 ก่อนที่จะดำเนินตามฟังก์ชันรอบ

การทำงานของการแปลงนี้ แสดงดังรูปที่ 2.10 เมื่อ I = round * Nb ตำแหน่งของไบต์ภายใน Word ของตาราง Key อธิบายในหัวข้อ 2.2.1



รูปที่ 2.10 การแปลงแบบ AddRoundKey ( )

2.4.2 Key Expansion

AES อัลกอริทึม จะนำ Cipher Key ไปผ่านกระบวนการ Key Expansion เพื่อสร้างตาราง Key ซึ่งจะมีทั้งหมด Nb (Nr+1) word อัลกอริทึมในการสร้างตาราง Key นี้ต้องการ Nb word ตั้งต้น ในแต่ละรอบของ Nr รอบ ก็จะต้องการ Key Nb word เมื่อสร้าง Key ชุดใหม่สำหรับรอบต่อไปออกมา ฉะนั้น จะได้ตาราง Key ซึ่งประกอบด้วย Word อาเรย์แบบเชิงเส้น ใช้สัญลักษณ์ $[W_i]$ โดยที่ i มีค่าอยู่ในช่วง $0 \leq i < Nb(Nr+1)$

การขยายของ Key อินพุทไปยังตาราง Key เป็นไปตาม คำสั่งเทียมในรูปที่ 2.11

SubWord ( ) เป็นฟังก์ชันในการนำ Word อินพุทไปสับเปลี่ยนด้วย S-box จะให้ Word เอาท์พุทออกมา RotWord ( ) เป็นฟังก์ชันในการนำ Word อินพุท $[a_0, a_1, a_2, a_3]$ ไปจัดเรื่องใหม่และให้ Word เอาท์พุท $[a_3, a_2, a_1, a_0]$ ออกมา Rcon[i] เป็น Word อาเรย์ซึ่งมีค่าคงที่ คือ $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ ซึ่ง x^{i-1} เป็นโพลิโนเมียลใน $GP(2^8)$ (x คือ {02}) ดังที่ได้อธิบายในหัวข้อ 2.3.1.2 และ i เริ่มต้นตั้งแต่ 1

จากรูปที่ 2.11 จะเห็นว่า Nk Word แรก ของKey ที่ถูกขยายถูกเติมด้วย Cipher Key และ Word ต่อ ๆ ไป (W[i]) จะมีค่าเท่ากับ Word ก่อนหน้า (W[i-1]) XOR กับ Word ก่อนหน้า Nk Word (W[i-Nk])

```
KeyExpansion (byte key [4*Nk], word w [ Nb* (Nr+1) ], Nk)
begin
        word temp

        i = 0

        while (i < Nk)
                w [ i ] = word ( key [4 * 1], key [4*i+1], key [4*i+2], key [4*i+3] )
        end while

        i = Nk

        while (Ii < Mb * (Nr+1))
                temp = w[i-]
                if (i mod Nk = 0)
                        Temp = SubWord (RotWord (temp)) xor Rcon [i/Nk]
                else if (Nk < 6 and i mod Nk = 4)
                        Temp = SubWord (temp)
                end if
                w [ Ii ] = w[i-Nk] xor temp
                i = i + 1
        end while
end

Note that Nk=4, 6, and 8 do not all have to be implemented; they are all included in the conditional statement above for conciseness.
```

รูปที่ 2.11 คำสั่งเทียมสำหรับกระบวนการ Key Expansion

สำหรับ Word ที่อยู่ในตำแหน่งที่เป็นจำนวนเท่าของ Nk Word นี้จะได้จากการ XOR ระหว่าง W[i-1] ,W[i-Nk], เพราะ Rcon[i] และสำหรับ Cipher Key ขนาด 256 บิต (Nk = 8) มีสิ่งที่ต่างจาก Cipher Key 128 เพราะ 192 บิต เล็กน้อย คือ ถ้า Nk =8 และ i-4 เป็นจำนวนเท่าของ Nk แล้ว SubWord ( ) จะถูกดำเนินการกับ W[i-1] ก่อนจะไปทำ XOR

2.4.3 Inverse Cipher

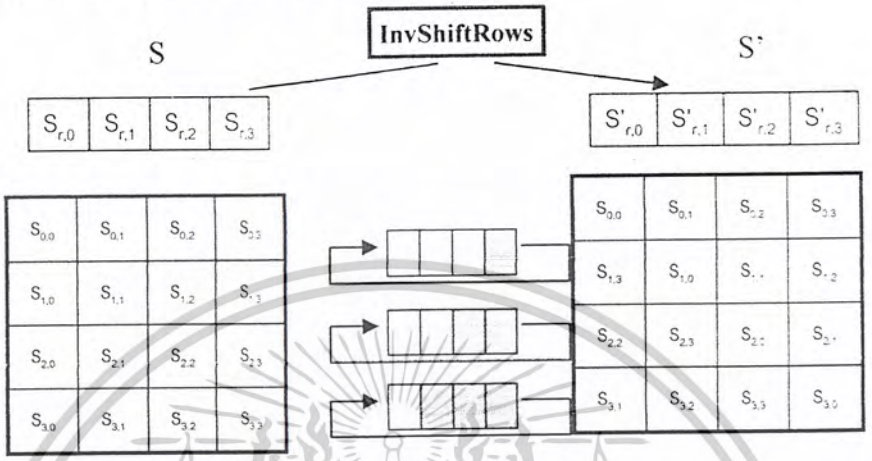
Inverse Cipher จะใช้การแปลงดังต่อไปนี้คือ InvShiftRows( ), InvSubByte () , InvMixColumns ( ) และ AddRoundKey ( )ในการประมวลผล State ซึ่งจะได้กล่าวถึงดังต่อไปนี้

2.4.3.1 การแปลงแบบ InvShiftRows ( )

การแปลงแบบ InvShiftRows ( ) จะตรงกันข้ามกับการแปลงแบบ ShiftRows ( ) โดยที่ใน 3 แถวสุดท้ายจะถูกเลื่อนไบต์แบบวนรอบด้วยค่าที่ไม่เท่ากัน และแถวแรกจะไม่ถูกเลื่อน แถวที่ถูกเลื่อนจะทำการเลื่อนไปเป็นจำนวน Nb-shift (r, Nb) ไบต์ โดยที่ค่าของการเลื่อน (shift(r,Nb)) ขึ้นกับค่า r ตามสมการ(2.23) การแปลงแบบ InvShiftRows () สามารถเขียนได้ดังนี้

$S'_{r,\,(c+shift(r,Nb))mod\ Nb} = S_{r,c}$ สำหรับ $0< r <4$ และ $0 \leq c < Nb$ (2.26)

การแปลงแบบ InvShiftRows () แสดงดังรูปที่2.12



รูปที่ 2.12 การแปลงแบบ InvShiftRows ( )

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

รูปที่ 2.13 อินเวอร์สของ S-Box

2.4.3.2 การแปลงแบบ InvSubBytes ( )

การแปลงแบบ InvSubBytes ( )จะตรงกันข้ามกับ SubBytes ( ) โดยที่จะนำแต่ละไบต์ใน State ไปทำการสับเปลี่ยนไบต์โดยใช้อินเวอร์สของ S-box ซึ่งได้มาจากการทำอินเวอร์สของ Affine transformation ตามด้วยทำอินเวอร์สการคูณใน GF(2^8)

อินเวอร์สของ S-box ซึ่งใช้ในการแปลงแบบ InvSubBytes () แสดงดังรูปที่ 2.13

2.4.3.3 การแปลงแบบ InvMixColumns ()

การแปลงแบบ InvMixColumns () จะตรงกันข้ามกับ การแปลงแบบ MixColumns () โดยจะกระทำกับ State แบบหลักต่อหลัก โดยพิจารณาให้แต่ละหลักเป็น โพลิโนเมียล 4 พจน์ ดังที่ได้อธิบายไว้ในหัวข้อที่ 2.3.2

แต่ละหลักใน State จะถูกพิจารณาเป็นโพลิโนเมียล บน GP (2^8) แล้วคูณแบบ modular(mod ด้วย x^4+1) ด้วย $a^{-1}(x)$ ตามสมการที่ 2.27

$$a^{-1}(x) = \{0b\}\, x^3 + \{0d\}\, x^2 + \{09\}\, x + \{0e\} \qquad (2.27)$$

ดังที่ได้อธิบายในหัวข้อ 2.3.2 การคูณแบบ modular นี้สามารถเขียนเป็นการคูณแบบเมตริกซ์

ได้ ถ้าให้ $S'(x) = a^{-1}(x) \otimes S(x)$ เขียนเป็นเมตริกซ์ได้เป็น

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{สำหรับ } 0 \le c < Nb \qquad (2.28)$$

จากการคูณแบบเมตริกซ์ข้างต้นจะได้

$$S'_{0,c} = (\{0e\} \bullet S_{0,c}) \oplus (\{0b\} \bullet S_{1,c}) \oplus (\{0d\} \bullet S_{2,c}) \oplus (\{09\} \bullet S_{3,c})$$

$$S'_{1,c} = (\{09\} \bullet S_{0,c}) \oplus (\{0e\} \bullet S_{1,c}) \oplus (\{0b\} \bullet S_{2,c}) \oplus (\{0d\} \bullet S_{3,c})$$

$$S'_{2,c} = (\{0d\} \bullet S_{0,c}) \oplus (\{09\} \bullet S_{1,c}) \oplus (\{0e\} \bullet S_{2,c}) \oplus (\{0b\} \bullet S_{3,c})$$

$$S'_{3,c} = (\{0b\} \bullet S_{0,c}) \oplus (\{0d\} \bullet S_{1,c}) \oplus (\{09\} \bullet S_{2,c}) \oplus (\{0e\} \bullet S_{3,c})$$

2.4.3.4 อินเวอร์สของการแปลงแบบ AddRoundKey ()

การแปลงแบบ AddRoundKey () ซึ่งได้อธิบายไว้ในหัวข้อ 2.4.1.4 เป็นอินเวอร์สด้วยตัวเองอยู่แล้ว ฉะนั้นอินเวอร์สของการแปลงแบบ AddRoundKey () คือ การทำ XOR นั่นเอง

2.4.3.5 Inverse Cipherที่ให้ผลลัพธ์เท่ากัน

ในการทำInverse Cipher โดยวิธีตรงดังที่ได้อธิบายไว้ในหัวข้อ 2.4.3 ลำดับในการแปลงจะต่างจากการทำ Cipher ในขณะที่รูปแบบของตาราง Key สำหรับการเข้าสลักและถอดสลักยังคงเหมือนกัน อย่างไรก็ตามด้วยคุณสมบัติของ AES อัลกอริทึม สามรถที่จะใช้ลำดับการแปลงในการทำInverse Cipher เหมือนกับ Cipher โดยจะให้ผลลัพธ์เท่ากันแต่การทำเช่นนี้จะทำได้ก็ต่อเมื่อมีการเปลี่ยนแปลงตาราง Key

คุณสมบัติ 2 ประการที่ ทำให้สามารถใช้ลำดับการแปลงเหมือน Cipherได้ก็คือ

1.การแปลงแบบ SubBytes ( ) และ ShiftRows ( ) สามารถสลับกันได้ กล่าวคือ เมื่อทำการแปลงแบบ SubBytes ( ) ตามด้วย การแปลงแบบ ShiftRows ( ) จะให้ผลลัพธ์เหมือนกับการแปลงแบบ ShiftRows ( ) ก่อน แล้วตามด้วยการแปลงแบบ SubBytes ( ) และสำหรับ InvSubBytes ( ) และ InvShiftRows ( ) ก็สามารถทำได้เช่นกัน

2. MixColumsns ( ) และ InvMixColumns ( ) มีความเป็นเชิงเส้นเมื่อเปรียบเทียบกับหลักอินพุท หมายความว่า InvMixColumns ( ) (State) XOR InvMixColumns ( ) (Round Key)

คุณสมบัติเหล่านี้ทำให้ InvSubBytes ( ) และ InvShiftRows ( ) สามารถสลับกันได้และ AddRowsKey ( ) InvMixColumns ( ) ก็สามารถสลับกันได้ด้วย โดยมีข้อแม้ว่าหลักหรือ Word ของตาราง Key ของการถอดสลักจะถูกเปลี่ยนแปลงโดยการใช้การแปลงแบบ InvMixColumns

โดยการสลับกันระหว่าง AddRowsKey ( ) กับ InvMixColumns ( ) หลังจากที่ได้ดัดแปลงตาราง Key ในครั้งแรก โดยใช้การแปลงแบบ InvMixColumns ( ) สำหรับรอบที่ 1 จนถึงรอบที่

Nr-1 การทำในลักษณะนี้ Word แรก และ Word สุดท้าย ของ Nb Word จะไม่ถูกดัดแปลงไปด้วย

การเปลี่ยนแปลงเหล่านี้จะทำให้ Inverse Cipher มีโครงสร้างที่ดีกว่า Inverse Cipher โดยวิธีตรง คำสั่งเทียมสำหรับ Inverse Cipher ที่ถูกดัดแปลงนี้แสดงดังรูปที่ 2.14 (Word อาเรย์ dw [] คือตาราง Key ของการถอดสลักที่ถูกดัดแปลงแล้ว และการดัดแปลงในกระบวนการ Key Expansion ปรากฏอยู่ในรูปที่ 2.14 ด้วย)

```
EqInvCipher (byte in [4*Nb] , byte out [4*Nb] , word dw [Nb*(Nr+1)])
begin
        byte state [4*,Nb]

        state = in

        AddRoundKey (state , dw [Nr*Nb , (Nr+1) *Nb-1])

        for round = Nr-1 step -1 downto 1
                InvSubBytes (state)
                InvShiftRows (state)
                InvMixColumns (state)
                AdRoundKey (state , dw[round*Nb , (round+1) *Nb-1])
        end for

        InvSubBytes (state)
        InvShiftRows (state)
        AdRoundKey (state , dw [0 , Nb-1] )

        out = state
end
```

For the Equivalent Inverse Cipher, the following pseudo code is added at the end of the Key Expansion routine (Sec. 5.2) :

```
        for i = 0 step 1 to (Nr+1) *Nb-1
                dw[i] = w [I]
        end for

        for round = 1 step 1 to Nr-1
                InvMixColumns (dw[round*Nb, (round+1)*Nb-1])    //  note change of
type
        end for
```

Note that, since InvMixColumns operates on a two-dimensional array of bytes while the Round Keys are held in an array of words, the call to InvMixColumns in this code sequence involves a change of type (i.e. the input to InvMixcolumns ( ) is normally the State array, which is considered to be a two-dimensional array of bytes, whereas the input here is a Round Key computed as a one-dimensional array of words).

รูปที่ 2.14 คำสั่งเทียมสำหรับ Inverse Cipher ที่ดัดแปลงแล้ว

บทที่ 3
ภาษาวีเอชดีแอล (VHDL)

วีเอชดีแอล (VHDL) ย่อมาจากคำว่า VHSIC Hardware Description Language (VSHIC:Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) VHDL เป็นภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ทั้งในระดับเกทจนถึงระดับดิจิตอลที่ซับซ้อน ทั้งนี้เนื่องจาก VHDL เป็นภาษาที่เหมาะในการนำมาใช้ในการเลียนแบบการทำงานของอุปกรณ์ อีกทั้งยังมีความยืดหยุ่นและไม่ถูกจำกัดโดยความสามารถทางเทคโนโลยีใด ๆ ทำให้ประหยัดเวลาและค่าใช้จ่ายในการออกแบบ จึงได้มีการนำมาใช้กันอย่างกว้างขวางในวงการอุตสาหกรรม รูปแบบของภาษาวีเอชดีแอล ประกอบด้วย 2 ส่วนใหญ่ ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล (Sequential Language) และภาษาคอนเคอร์เรนท์ (Concurrent Language) การโปรแกรมด้วยภาษาวีเอชดีแอลสามารถเขียนได้ทั้งสองรูปแบบรวมกัน นอกจากนี้ ตัวภาษายังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยเข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้และสามารถกำหนดรูปแบบไวยากรณ์ (Syntax) อีกทั้งยังมีการตรวจสอบความหมายของภาษาว่าจะซิมูเลท (Simulate) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดยวีเอชดีแอลต้องผ่านการซิมูเลทเพื่อตรวจสอบการทำงาน ฉะนั้น ในการคอมไพล์ (Compile) จะมีการตรวจสอบทั้งวงจรและซีมูเลทชั้นซีแมนติก (Semantic) อย่างไรก็ตามถึงแม้ตัวภาษาจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้โดยไม่จำเป็นต้องศึกษารายละเอียดทั้งหมด

3.1 แนะนำวีเอชดีแอล (Introduction to VHDL)

ในช่วงฤดูร้อนของปี 1981 สถาบันเพื่อป้องกัน (The Institute for Defense Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่ง เพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้น เรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHISC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวกับวงจรรวมที่มีความเร็วสูงมาก (Very High Speed Integrated Circuit) ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านของการกำหนดพฤติกรรมการ

ทำงานของวงจร ทำให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งวงจรของดิจิตอลทั่ว ๆ ไป และในระบบที่แตกต่างกันออกไป เช่น พฤติกรรมการทำงานของระบบเรดาร์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ได้ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้จะสามารถถูกใช้ได้ตลอดในทุก ๆ ระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง (System Level) จนถึงระดับที่ต่ำกว่า (Lower hardware level) สามารถใช้ภาษาเดียวกันได้โดยตลอด ทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างบุคลที่ทำงานร่วมกันได้เป็นอย่างดี

3.1.1 **ข้อกำหนด** (VHDL Requirement)

ในเอกสารของ DoD (Department of Defense Requirement for Hardware Description Language) ซึ่งออกมาในเดือนมกราคมปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

3.1.1.1 **ลักษณะทั่วไป** (Generation Features)

เอกสารของ DoD กำหนดไว้ว่า VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการเลียนแบบ (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์คือ ระบบจนถึงระดับเกทอีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลจริง ๆ ทุก ๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อม ๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ถือว่าเป็นข้อกำหนดที่สำคัญอย่างหนึ่งใน VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้ว ความพร้อมเพรียงหมายถึงทุก ๆ คำสั่ง องค์ประกอบเกท หรือวงจรต่าง ๆ จะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่า ได้มีการปฏิบัติไปพร้อม ๆ กัน)

3.1.1.2 **สนับสนุนการออกแบบแบบลำดับขั้น** (Support for Design Hierarchy)

การออกแบบลำดับขั้น เป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลาย ๆ ระดับในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตนเองหรือถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อย ๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้

3.1.1.3 **ไลบรารี** (Library Support)

VHDL ได้สนับสนุนการมีไลบราลีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้อง ควรจะถูกเก็บไว้ในไลบรารีหลังจาก ที่ได้ผ่านการคอมไพล์เรียบร้อยแล้ว เพื่อให้ผู้ออกแบบคนอื่น ๆ สามารถนำไปใช้ได้ด้วย

3.1.1.4 **ลำดับคำสั่ง** (Sequential Statement)

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if-then-else และ loop ทั่ว ๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานพร้อมเพรียงกันเช่นเดิม

3.1.1.5 **การกำหนดคุณสมบัติ** (Generic Design)

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่น ๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้น ๆ ภาษาสำหรับการออกแบบที่ดีควรจะสามารถให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลา โหลดและเงื่อนไขทางสภาพแวดล้อมอื่น ๆ ความสามารถในการกำหนดคุณสมบัติก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

3.1.1.6 **ชนิดของข้อมูล** (Type Declaration and usage)

VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด UIT และ BOOLEAN เท่านั้น แต่ยังไม่สามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับ (Enumerate Type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นใส่เองก็ได้

3.1.1.7 **โปรแกรมย่อย** (Use of Subprogram)

ความสามารถในการใช้ฟังก์ชันและโพรซีเยอร์ (Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL เราสามารถที่จะใช้โประแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก (Logic) การกำหนดตัวกระทำต่าง ๆ ทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

3.1.1.8 **การควบคุมเวลา** (Timing Control)

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการการตรวจสอบ การออกแบบเกต หรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ (Event) นอกจากนี้ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

3.1.1.9 **การกำหนดแบบโครงสร้าง** (Structural specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุก ๆ ระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวม ที่เกิดจากองค์ประกอบย่อยที่ต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

3.2 ความสามารถของภาษาวีเอชดีแอล (Capability)

- ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิพกับผู้ออกแบบ (CAD Tools)
- ใช้เป็นสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่าง ซีเออี (CAE) และซีเอดีทูล (CAD Tools) เช่นตัวภาษาซ้อสโค้ด (SOURCE CODE) ของ VHDL สามารถคอมไพล์โดยใช้คอมไพล์เลอร์ (Compiler) และซีมูเลเตอร์ (Simulator) ได้หลายตัวแตกต่างกัน
- ภาษา VHDL สนับสนุนการออกแบบ แบบท็อปดาวน์ (Top Down Design) และแบบบัททอมอัพ (Bottom Up Design) หรือผสมกันทั้งสองแบบ
- ตัวภาษา VHDL เป็นแบบทั่วไป (Generic) ไม่อิงเทคโนโลยีอันใดอันหนึ่ง ในขณะเดียวกัน ก็สนับสนุนหลาย ๆ เทคโนโลยี
- ตัวภาษา VHDL สามารถอ่านและทำความเข้าใจได้โดยมนุษย์
- สนับสนุนการออกแบบทั้งระบบซิงโครนัส (Synchronous) และอะซิงโครนัส (Asynchronous)
- ตัวภาษา VHDL เป็นภาษามาตรฐานรับรองโดย IEEA และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษา VHDL สามารถเคลื่อนย้ายไปยังระบบใด ๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้
- สามารถเขียนโมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล (ขึ้นอยู่กับซอฟแวร์)

- ภาษา VHDL สนับสนุนการเขียนถึง 3 รูปแบบ ได้แก่ แบบบีเฮฟวีเออร์ (Behavioral Style) แบบสตัคเจอรัล (Structural Style) แบบดาต้าโฟลว์ (Data Flow) หรือสามารถเขียนรวมกันได้ 3 รูปแบบ

- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของส่วนประกอบ (Component) ฟังก์ชั่นโพรซีเดอร์ (Function Procedure) และแพคเกต (Package)

- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation delay , Min-Max Delay , Setup , Holding Time สามารถอธิบายได้โดยตัวภาษา

- ภาษา VHDL เป็นมาตรฐานที่ใช้โดยบริษัทและผู้ออกแบบหลาย ๆ แห่ง ฉะนั้น จึงง่ายที่จะทำความเข้าใจถึงแม้ว่าจะมาจากแหล่งต่าง ๆ

- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะว่าตัวแปรได้ตรวจสอบตัวแปรทงซิมูเลชั่นซีแมนติกไว้ด้วย

3.3 หลักการสร้างโมเดลโดยใช้ภาษาวีเอชดีแอล (General VHDL Modeling Principles)

วีเอชดีแอลเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ ซึ่งจะช่วยในการสร้างและออกแบบวงจรรวมดิจิตอลและส่วนประกอบต่าง ๆ อาจใช้อธิบายระบบหรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ (Component Block) จากนั้นก็ทำการจำลองการทำงาน (Simulate) โดยที่รูปแบบนั้นยังไม่ได้สร้างขึ้นจริงหรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจรเกทเลเวลต่อไป ประโยชน์จริงของการใช้วงจร วีเอชดีแอลเป็น Design Tools แทนการสร้างต้นแบบ (Prototype) ขึ้นมาจริง คือเราสามารถอธิบาย Product Idea, Product Proposal, Product Specification ในรูปของ Text จากนั้นก็นำคอมไพล์เพื่อดู Timing การทำงานแล้วแก้ไข (Refine) จนกว่าจะได้ Specification ตามต้องการ เมื่อ product ได้ผลตามที่ต้องการแล้ว จึงนำไปสู่การสังเคราะห์ (Synthesis) เพื่อให้ได้เกทเลเวล Schematic เพื่อนำไปสร้างเป็นต้นแบบจริงต่อไป ซึ่งต้นแบบทีสร้างนั้นทำงานได้จริงเพราะได้ทำการ Simulate เรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้างต้นแบบได้มาก

ตัวภาษา VHDL สนับสนุนหลักการต่าง ๆ ให้เขียนแก้ไขและบำรุงรักษาวงจรดิจิตอลที่มีความซับซ้อนให้เป็นไปอย่างรวดเร็วและมีประสิทธิภาพโดยมีหลักเกณฑ์ดังนี้

และอ่านเข้าใจได้ง่าย หลักการนี้จะสนับสนุนหลักการ Abstraction คือสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่ารูปแบบนั้นจะถูกสร้างขึ้นมาอย่างไร เป็นต้น การซ่อนรายละเอียดภายในโมดูลทำให้ความสนใจของผู้ออกแบบนั้นสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ เช่นในการอธิบายการทำงานของฟลิปฟลอป ไม่ต้องสนใจว่า NAND เกท จะทำงานอย่างไร จะต่อกันภายในอย่างไร โดย NAND เกท สามารถเขียนขึ้นมาแล้วคอมไพล์เก็บไว้ในไลบรารี ผู้ที่ออกแบบฟลิปฟลอประดับสูงขึ้นมา เพียงแต่ต้องรู้ว่า จะเชื่อมต่ออินพุท/เอาท์พุทของ NAND เกท มาใช้งานได้อย่างไร โดยไม่ต้องสนใจว่า NAND เกท จะถูกสร้างและพัฒนามาอย่างไร ประโยชน์อีกอย่างหนึ่งคือ ป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL โมเดลไปยังที่อื่น ทำให้เราป้องกันทรัพย์สินทางปัญญาได้อีกในระดับหนึ่ง

3.3.5 Uniforminity

Uniforminity เป็นหลักการอีกอย่างหนึ่งที่ช่วยในการอธิบายฮาร์ดแวร์ด้วยภาษา VHDL หมายถึง การสร้างโมดูลของรหัส ในลักษณะคล้ายกัน โดยใช้ตัวภาษา VHDL Building Block ทำให้เกิดการเขียนรหัสที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้คำอธิบาย (Comment) เป็นต้น ทำให้การพัฒนาโมดูลทำความเข้าใจง่าย

3.4 องค์ประกอบพื้นฐานในวีเอชดีแอล (Basic concept in VHDL)

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบภายใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 3.1 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ ENTITY ตามด้วยชื่อขององค์ประกอบ และคำ IS ภายในบรรยายถึงพอร์ตการติดต่อ อินพุท/เอาท์พุท พอร์ตขององค์ประกอบ ส่วนลักษณะกายภาพภายนอกอื่น ๆ เช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท/เอาท์พุท และพารามิเตอร์อื่น ๆที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 3.1 การบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังคำว่า BEGIN เป็นต้นไป

3.4.1 การกำหนดการเชื่อมต่อ (Interface Description)

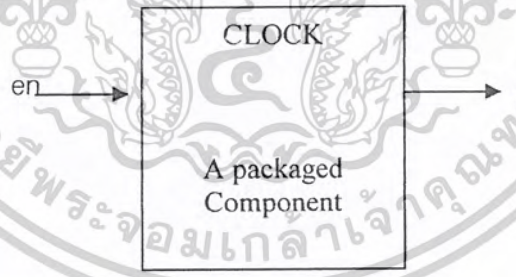
การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ต สำหรับการติดต่อกับองค์ประกอบภายนอกอื่น ๆ ดังตัวอย่างในรูปที่ 3.2 บรรทัดแรกเป็นการ

กำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ clock component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ในวงเล็บ IN และ OUT กำหนดโหมดของสัญญาณเป็นอินพุทหรือเอาท์พุท BIT แสดงชนิดของข้อมูล

```
ENTITY component_ name IS
        Input and output ports.
        Physical and other parameters.
END   component_name;
```

```
ARCHITECTURE identifier OF compionent_name IS Declarations.
BEGIN
        Specification of  functionality of the component
        In terms of its input lines and as influenced
        By physical and other parameters.
END  identifier;
```

รูปที่ 3.1 แสดงการกำหนดค่าการเชื่อมต่อและสถาปัตยกรรม




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

รูปที่ 3.2 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อของ clock component

3.4.2 **การกำหนดรูปแบบการบรรยาย** (Architecture Description)

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในเทอมของ clock component ในรูปที่ 3.3 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี en เป็นอินพุทและมีclk เป็นเอาท์พุท PROCESS เป็นค่าเริ่มต้นสำหรับการบรรยาย

ในเชิงพฤติกรรม ภายในโพรเซสกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น "0" ถ้าสัญญาณ en มีค่าเป็น "1" ค่าของ periodic จะถูกคอมพลีเมนต์และส่งค่าให้กับ clk ซึ่งเป็นสัญญาณเอาท์พุท คำสั่ง WAIT กำหนดให้สัญญาณมีคาบเป็นเวลา 1 ไมโครวินาที

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

รูปที่ 3.3 แสดงการบรรยายเชิงพฤติกรรมของ Clock_component

3.4.3 โปรแกรมย่อย (Subprogram)

การใช้ฟังก์ชั่นและโพรซีเยอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่ว ๆ ไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่น ถ้าเราให้ฟังก์ชั่นแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกะจริง ๆ ในขณะที่เราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าหน่วงเวลา แล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 3.4 แสดงการใช้ฟังก์ชั่นโพรซีเยอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิต เป็นค่าจำนวนเต็ม รูปที่ 3.5 แสดงการใช้ฟังก์ชั่นโดยกำหนดให้ X เป็นตัวแปรชนิดบิตแทนการกระทำในสมการบูลีน

```
TYPE byte IS ARRAY (7 DOWN TO) OF BIT
PROCEDURE byte _to_integer (ib:IN byte;oi: OUT INTEGER) IS
            VARIABLE result : INTERGER : = 0;
BEGIN
            FOR 1 IN 0 TO 7 LOOP
                        IF ib (I) = '1' THEN
                                    Result : = result + 2 **I;
                        END IF;
END LOOP;
Oi : = result;
END byte_to_interger;
```

รูปที่ 3.4 แสดงการใช้โพรซีเยอร์

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

รูปที่ 3.5 แสดงการใช้ฟังก์ชั่น

3.4.4 โอเปอร์เรเตอร์ (VHDL OPERATORS)

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 3.6

PREDEFINED OPERATORS

LOGICAL OPERATORS : NOT AND OR NAND NOR XOR
OPERAND TYPE : BIT BOOLEAN
RESULT TYPE : BIT BOOLEAN

RELATIONAL OPERATORS : = /= < <= > >=
OPERAND TYPE : any type
RESULT TYPE : Boolean

ARITHMETIC OPERATORS : + - */** MOD REM ABS
OPERAND TYPE : INTEGER REAL Physical
RESULT TYPE : INTEGER REAL Physical

XONCANTINATION OPERATOR : &
OPERAND TYPE : array of any type
RESULT TYPE : array of any type

รูปที่ 3.6 แสดงตัวกระทำใน VHDL

3.4.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

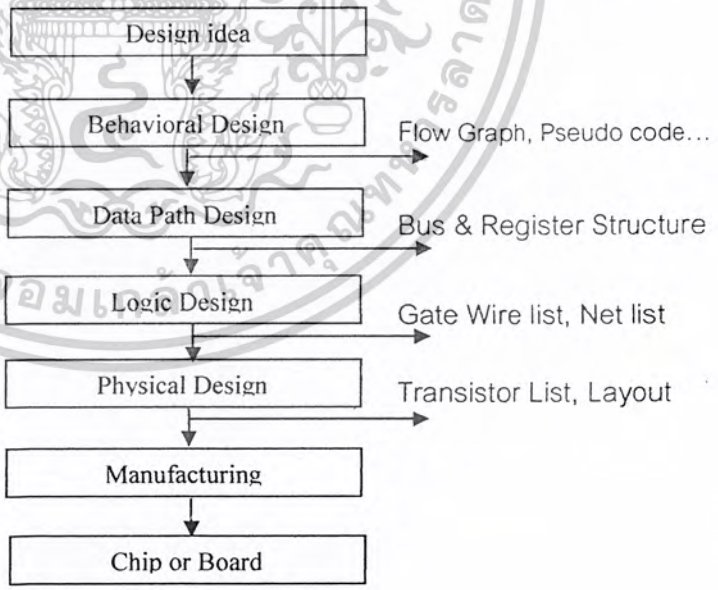
ในวงจรอิเล็กทรอนิกส์ อุปกรณ์ทุกตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (always active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยเสมอในทุก ๆ เหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงาน ที่อยู่ภายในส่วนของสถาปัตยกรรมการบรรยายจะมีการทำงานที่พร้อมเพรียงกันเสมอหรือแม้แต่โพรเซสที่มีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หามมีหลาย ๆ โพรเซสอยู่ภายในโครงการสร้างเดียวกันทุก ๆ โพรเซสก็จะทำงานไปพร้อม ๆ กันด้วย

3.4.6 **สัญญาณและตัวแปร** (Signals and Variable)

สัญญาณที่เป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กันสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น W <= AFTER 12 ns หมายถึง กำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 ns.

3.5 โครงสร้างของวีเอชดีแอล

ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั้งได้ออกมาเป็น อุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้ จะต้องผ่านขั้นตอนต่าง ๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้นตอน ทำการเพิ่มเติมตามความจำเป็นและเข้ากระบวนการออกแบบในขั้นต่อไป รูปที่ 3.7 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่ว ๆ ไป ซึ่งขั้นแรกผู้ออกแบบกำหนดแนวคิดในการออกแบบเสียก่อนและทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นในขั้นตอนนี้จึงมีความจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo Code)



รูปที่ 3.7 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นต่อไปเป็นการออกระบบเส้นทางของข้อมูล ผู้ออกแบบจะต้องกำหนดส่วนประกอบของรีจีสเตอร์ ผู้ออกแบบจะกำหนดส่วนของรีจีสเตอร์ (Register) และวงจรตรรกะ (Logic) ที่จำเป็นทั้งหมดที่ประกอบกันเป็นระบบที่สมบูรณ์แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการควบคุมในการเคลื่อนย้ายข้อมูลระหว่างรีจีสเตอร์และวงจรตรรกะจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้

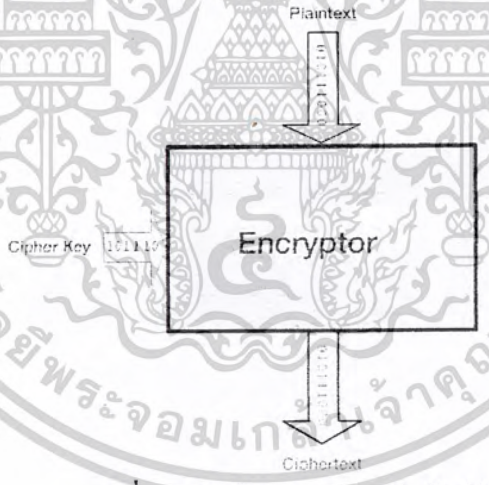
การออกแบบวงจรตรรกะจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิปฟลอปเป็นส่วนของอุปกรณ์แยกต่าง ๆ ได้แก่ รีจีสเตอร์เก็บข้อมูลวงจรตรรกะและส่วนควบคุมฮาร์ดแวร์ ซึ่งในขั้นสุดท้ายจะได้ออกมาเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอบนั่นเอง

ต่อมาเป็นขั้นตอนของการเปลี่ยนเครือข่ายการโยงใยในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์และเลย์เอาท์ (Transistor list and layout) ในขั้นตอนนี้เกี่ยวข้องกับการจัดวางเกทและฟลิปฟลอปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซล ขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

บทที่ 4
การออกแบบวงจรด้วยภาษา VHDL

ในการออกแบบวงจรดิจิตอลด้วยภาษา VHDL นั้น เป็นที่นิยมกันอย่างแพร่หลาย เนื่องจากมีข้อดีหลายประการ เช่น ลดความยุ่งยากในการออกแบบ เพราะในปัจจุบันมีเครื่องมือช่วยออกแบบที่มีประสิทธิภาพ ทำให้ผู้ออกแบบไม่ต้องลงลึกถึงในระดับเกท เพียงแต่ใช้ภาษา VHDL บรรยายพฤติกรรมของวงจร และใช้เครื่องมือช่วย ก็จะได้วงจรในระดับเกทอย่างอัตโนมัติ และที่สำคัญภาษา VHDL สนับสนุนการออกแบบในลักษณะ Top – Down ซึ่งจะทำให้การพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อนทำได้ง่ายขึ้น

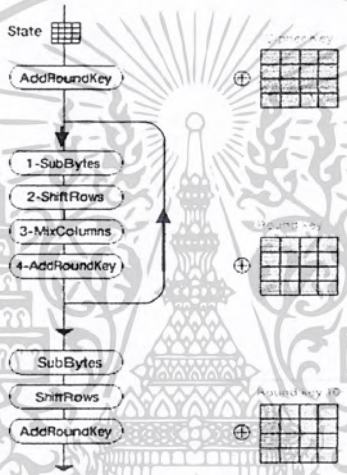
จากคุณสมบัติของภาษา VHDL ที่สนับสนุนการออกแบบในลักษณะ Top – Down โครงงานนี้จึงนำคุณสมบัติดังกล่าวมาใช้ในการออกแบบด้วย ซึ่งจะกล่าวถึงต่อไป



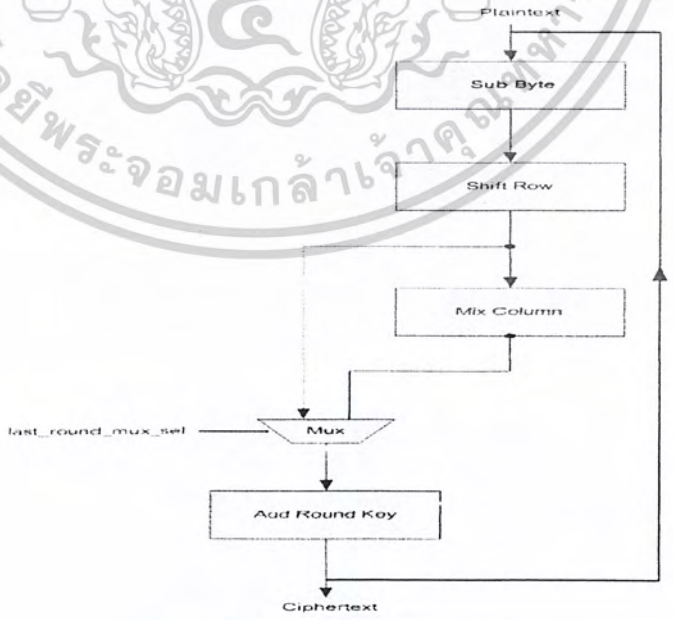
รูปที่ 4.1 บล็อกไดอะแกรมของตัวเข้ารหัสข้อมูล

รูปที่ 4.1 แสดงให้เห็นถึงระดับบนสุด(Top level) ของโครงงานนี้ คือตัวเข้ารหัสข้อมูลโดยใช้การเข้ารหัสข้อมูลตามมาตรฐาน Advanced Encryption Standard ตัวเข้ารหัสข้อมูลจะทำการเข้ารหัสข้อมูลที่ทางเข้าที่เรียกว่า "Plaintext" โดยการนำรหัสลับที่เรียกว่า "Cipher Key" ไปจัดการกับ Plaintext และจะได้ข้อมูลที่ถูกเข้ารหัสแล้วซึ่งเรียกว่า "Ciphertext" ออกมา

จากรายละเอียดของอัลกอริทึ่มที่ใช้ในการเข้ารหัสดังที่กล่าวในบทที่ 2 จะเห็นว่าต้องมีการทำงานเป็นลำดับ จาก SubBytes() ShiftRows() MixColumns() และ AddRoundKey() ทั้งหมด 10 รอบจึงจะได้ผลลัพธ์ หรือเอาท์พุทที่เรียกว่า "Cipher text" ออกมาดังรูปที่ 4.2 เนื่องจากข้อจำกัดทางด้านอุปกรณ์ที่ใช้ในการทำโครงงาน โครงงานนี้จึงออกแบบโดยสร้างแต่ละส่วน(คือ SubBytes() , ShiftRows() , MixColumns() และ AddRoundKey()) เพียง 1 ชุด แล้วให้ข้อมูลไหลวนซ้ำจนครบ 10 รอบ จึงได้ Cipher text ออกมา และมีมัลติเพล็กเซอร์สำหรับรอบสุดท้ายดังรูปที่ 4.3

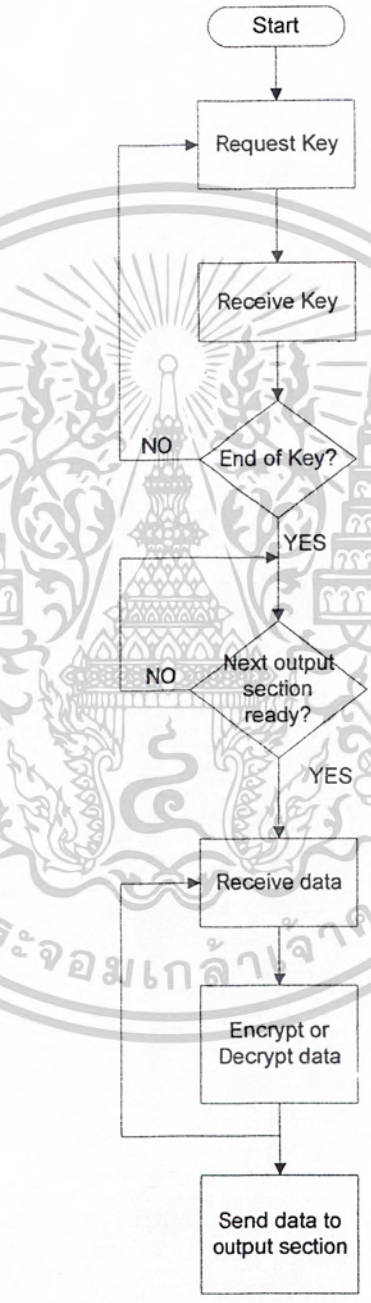


รูปที่ 4.2 กระบวนการเข้ารหัสข้อมูลตามมาตรฐาน Advanced Encryption Standard



รูปที่ 4.3 บล็อกไดอะแกรมของการเข้ารหัส

4.1 กระบวนการทำงานของระบบ



รูปที่ 4.4 Flow chart แสดงขั้นตอนการทำงานของระบบ

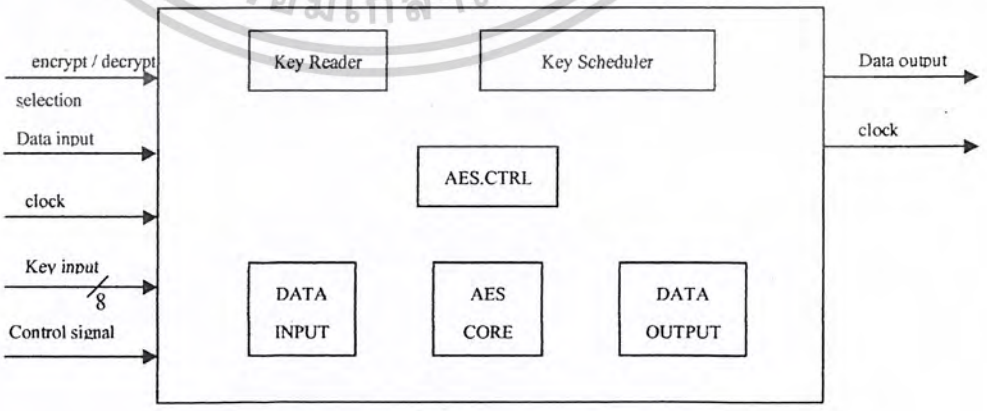
ในหัวข้อนี้จะกล่าวถึงขั้นตอนการทำงานของระบบที่ออกแบบขึ้น เพื่อความเข้าใจในส่วนของการออกแบบ ซึ่งจะกล่าวถึงในหัวข้อถัดไป

รูปที่ 4.4 แสดงขั้นตอนการทำงานของระบบ เมื่อกดปุ่ม Start หรือ เริ่มเปิดเครื่อง ระบบจะร้องขอ Cipher Key จากผู้ใช้งาน ซึ่งในขั้นตอนการป้อน Cipher Key นี้ ระบบถูกออกแบบให้รับ Key ได้ทีละ 8 บิต เพราะฉะนั้นจึงต้องป้อน 16 ครั้ง จึงครบ 128 บิต หลังจากที่ป้อน Key ครบแล้ว จะต้องมีสัญญาณ End of Key มาบอกให้กับระบบเพื่อให้ระบบรู้ว่าขณะนี้ Key ครบแล้ว และต้องทำขั้นตอนต่อไป คือการตรวจสอบระบบถัดไปทางด้านเอาท์พุท ว่าพร้อมที่จะรับข้อมูลหรือยัง ถ้าพร้อมแล้วระบบดังกล่าวจะต้องส่งลอจิก '1' มาแจ้งให้ทราบ ถ้ายังไม่พร้อม (ลอจิก '0') ระบบก็จะรอจนกว่าจะพร้อมจึงจะทำขั้นตอนต่อไป คือการรับข้อมูลเข้ามาประมวลผล ซึ่งข้อมูลที่รับมาจะเป็นแบบอนุกรมแบบซิงโครนัส ข้อมูลนี้จะต้องถูกแปลงให้เป็นแบบขนาน 128 บิตโดยภาคอินพุท เพื่อส่งต่อให้กับส่วนประมวลผล หลังจากที่ประมวลผลเสร็จ ข้อมูลก็จะอยู่ในรูปแบบขนาน 128 บิตเช่นกัน ภาคเอาท์พุทจะเป็นส่วนจัดการให้ข้อมูลอยู่ในรูปแบบอนุกรมเช่นที่รับเข้ามา และส่งออกไปยังระบบภายนอกต่อไป

4.2 คอมโพเน็นต์ต่างๆในระบบ

ในส่วนของการออกแบบนั้น เราจะออกแบบโดยใช้ลักษณะ Top – Down Design ซึ่งจะทำให้ออกแบบได้ง่ายขึ้น และจะกล่าวถึงในแต่ละส่วนดังต่อไปนี้

4.2.1 คอมโพเน็นต์ AES Chip (Top Level)



รูปที่ 4.5 แสดงเอ็นทิตี้ของ AES Chip

คอมโพเน็นต์ AES chip ประกอบด้วยส่วนต่าง ๆ ดังต่อไปนี้

1. ส่วนเข้าและถอดรหัส (AES CORE) ทำหน้าที่เข้ารหัส หรือถอดรหัสสัญญาณ

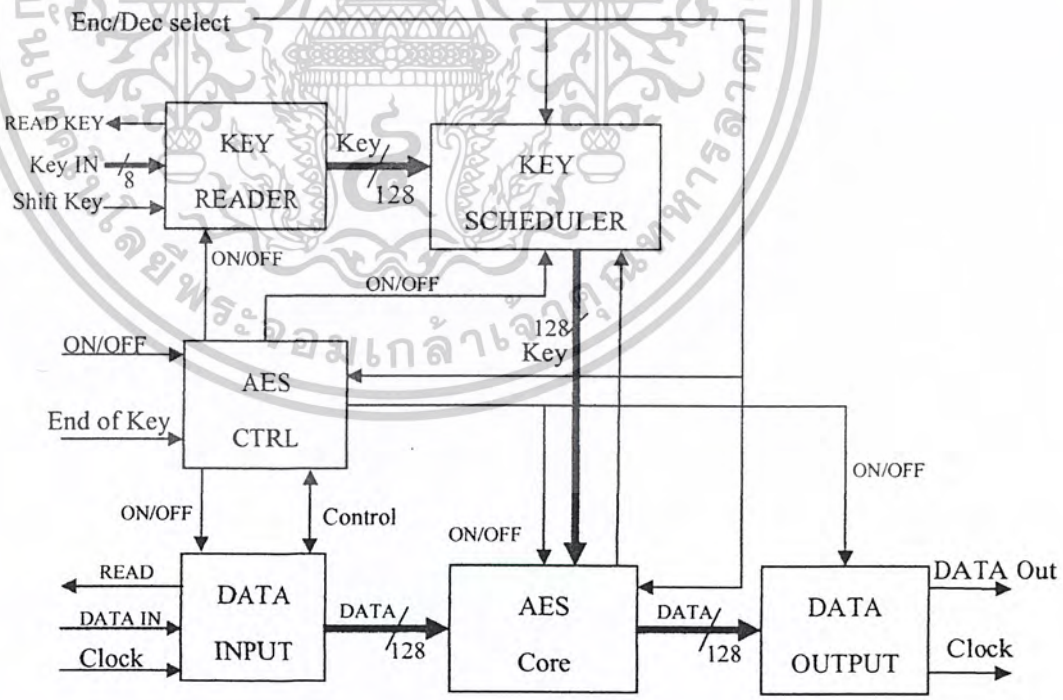
2. ส่วนสร้าง Round key (Key Scheduler) ทำหน้าที่สร้าง Round Key ให้กับ AES CORE สำหรับกระบวนการ Add Round Key ในแต่ละรอบ

3. ส่วนอ่านรหัสกุญแจ (Key Reader) ทำหน้าที่อ่านรหัสกุญแจจากภายนอกให้กับส่วนสร้าง Round key

4. ส่วนรับข้อมูล (DATA INPUT) ทำหน้าที่รับข้อมูลอนุกรมแบบซิงโครนัสจากภายนอก เพื่อส่งให้กับส่วนAES CORE ในแบบขนาน 128 บิต

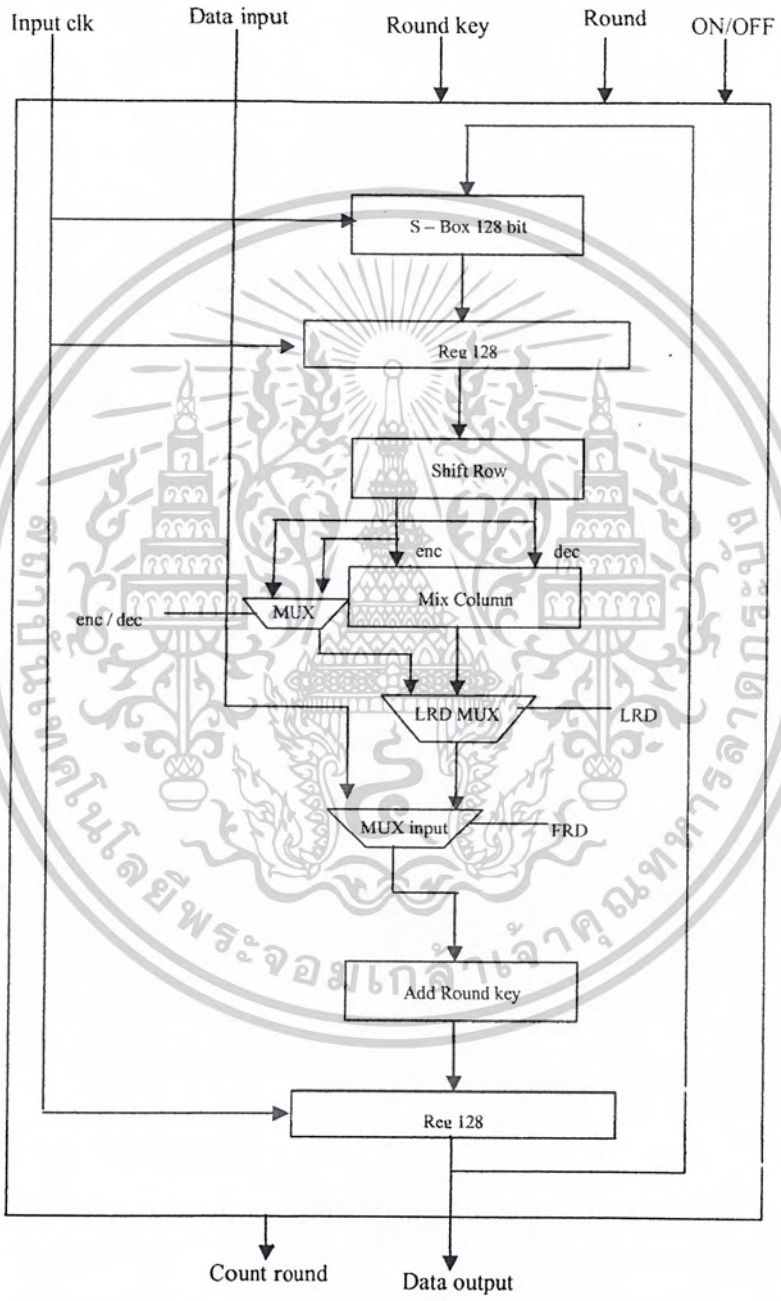
5. ส่วนส่งข้อมูลออก (DATA OUTPUT) ทำหน้าที่ส่งข้อมูลที่ได้จากการประมวลผลโดย AES CORE ออกไปภายนอกในแบบอนุกรมซิงโครนัส

6. ส่วนควบคุม (AES CTRL) ทำหน้าที่ควบคุมสัญญาณต่าง ๆ ให้ระบบทำงานได้อย่างถูกต้อง



รูปที่ 4.6 บล็อกไดอะแกรมแสดงการเชื่อมต่อโมดูลย่อยของ AES Chip

4.2.2 คอมโพเน็นต์ AES CORE (Level 1)



รูปที่ 4.7 แสดงเอ็นทิตี้ของ AES CORE

คอมโพเน็นต์ AES CORE ทำหน้าที่เข้ารหัสหรือถอดรหัสสัญญาณที่ส่งมาจาก DATA INPUT และสัญญาณที่ผ่านการเข้ารหัส หรือถอดรหัสแล้ว จะถูกส่งไปยัง DATA OUTPUT เพื่อส่งไปภายนอกชิปต่อไป โดยภายในคอมโพเน็นต์

AES CORE ประกอบด้วย คอมโพเน็นต์ย่อย ๆ ดังนี้

1. S BOX 128 bit
2. Shift Row
3. Mix Column
4. Add Round key
5. Reg 128
6. Counter 2 bit
7. MUX
8. Last round mux

<u>คำอธิบายขาสัญญาณ</u>

ON/OFF (in) : เป็นสัญญาณปิด/เปิด ให้คอมโพเน็นต์ทำงานหรือไม่ทำงาน

Input clk (in) : เป็นสัญญาณนาฬิกา

Round (in) : เป็นสัญญาณแสดงรอบในการคำนวณ มีขนาด 4 บิต

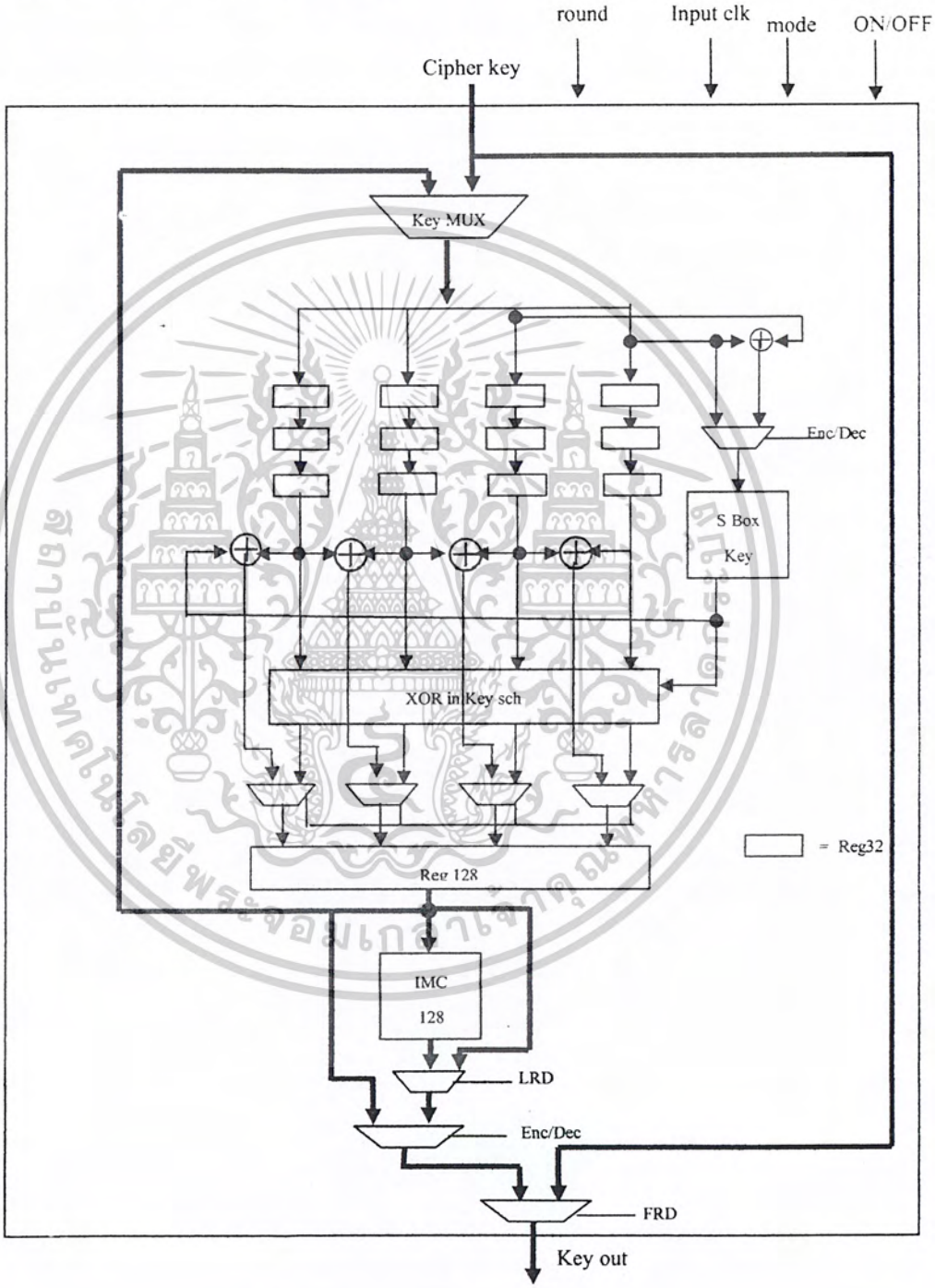
DATA input (in) : เป็นสัญญาณข้อมูลขาเข้าซึ่งจะถูกนำไปประมวลผล มีขนาด 128 บิต

Count Round (out) : เป็นสัญญาณที่ส่งไปยัง AES CORE เพื่อนับรอบในการคำนวณ

Round key (in) : เป็นสัญญาณ Round key ซึ่งส่งมาจาก Key Scheduler มีขนาด 128 บิต

Data output (out) : เป็นสัญญาณข้อมูลขาออกซึ่งผ่านการประมวลผลจาก AES CORE แล้ว มีขนาด 128 บิต

4.2.3 คอมโพเน็นต์ Key Scheduler (Level 1)



รูปที่ 4.8 แสดงเอ็นทิตี้ของ Key Scheduler

คอมโพเน็นต์ Key Scheduler ทำหน้าที่สร้าง Round key ให้กับ AES CORE โดยรับ key ที่เป็นอินพุทมาจาก Key Reader ใช้สัญญาณนาฬิกาในการกำหนดจังหวะการทำงานและจะทำงานพร้อมกับส่วน AES CORE

Key Scheduler ประกอบด้วย คอมโพเน็นต์ย่อยดังนี้

1. Reg 128
2. Key Teg 128
3. XOR in Key sch
4. MUX 32
5. XOR 32
6. Reg 32
7. Key Mux
8. IMC 128
9. MUX 128
10. LRD MUX

<u>คำอธิบายขาสัญญาณ</u>

ON/OFF (in) : เป็นสัญญาณใช้ในการเปิดปิดการทำงานของคอมโพเน็นต์ (1 = เปิด, 0 = ปิด)

Mode (in) : ใช้เลือกให้ทำงานเป็น Encryptor หรือ Decryptor (1 = Enc, 0 = Dec)

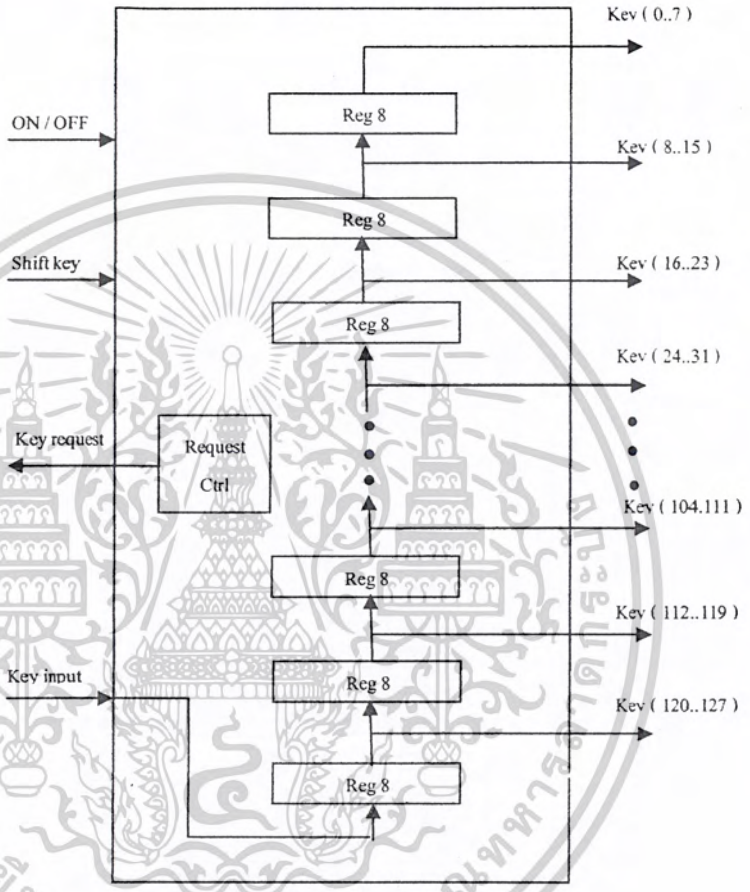
Input clk (in) : ใช้เป็นขาสัญญาณนาฬิกาในการกำหนดจังหวะการทำงาน

Round (in) : เป็นสัญญาณบอกรอบในการคำนวณ มีขนาด 4 บิต

Cipher Key (in) : เป็น Key ที่ส่งมาจากส่วน Key Reader ใช้สำหรับประมวลผลหา Round key มีขนาด 128 บิต

Key out (out) : เป็น Round Key ส่งไปให้กับส่วน AES CORE มีขนาด 128 บิต

4.2.4 คอมโพเน็นต์ Key Reader (Level 1)



รูปที่ 4.9 แสดงเอ็นทิตี้ของ Key Reader

คอมโพเน็นต์ Key Reader ทำหน้าที่อ่าน Key ที่ผู้ใช้ป้อนจากภายนอก ในการอ่านจะอ่านทีละ 8 บิต 16 ครั้ง และจะจัดการ Key ที่อ่านเข้ามาส่งให้กับ Key Scheduler แบบ 128 บิต ภายใน คอมโพเน็นต์ Key Reader ประกอบด้วยคอมโพเน็นต์ย่อย ๆ ดังนี้

1. Request signal control
2. Reg 8 KR

คำอธิบายขาสัญญาณ

ON/OFF (in) : สำหรับการเปิด/ปิด คอมโพเน็นต์ให้ทำงานหรือไม่ทำงาน

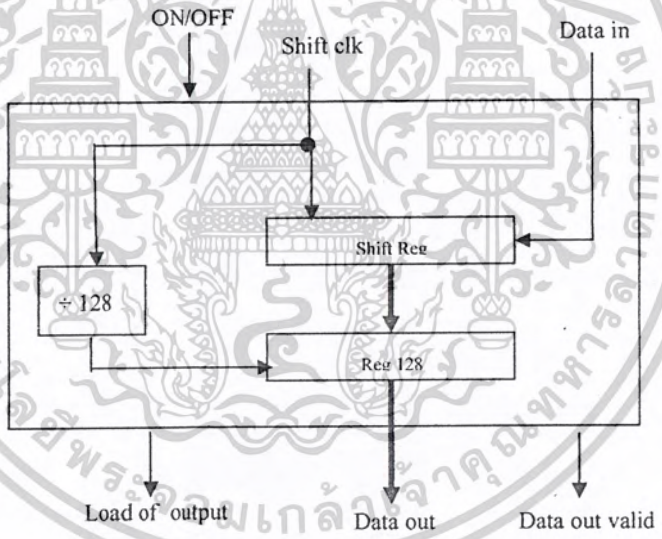
Request (out) : เป็นสัญญาณที่ส่งออกไปภายนอกเพื่อแสดงการร้องของ Key

Key input (in) : เป็นสัญญาณ Key จากผู้ใช้ส่งเข้ามาเพื่อเป็น Cipher Key มีขนาด 7 บิต

Shift Key (in) : เป็นสัญญาณที่ใช้ในการเลื่อน Key ทีละ 8 บิต เข้าไปใน Key Reader

Key out (out) : เป็น Key เอาท์พุทสำหรับส่งให้กับส่วน Key Scheduler เพื่อสร้าง Round Key มีขนาด 128 บิต

4.2.5 คอมโพเน็นต์ DATA INPUT(Level 1)



รูปที่ 4.10 แสดงเอ็นทิตี้ของ DATA INPUT

คอมโพเน็นต์ DATA INPUT ทำหน้าที่รับสัญญาณข้อมูลซึ่งเข้ามาในลักษณะอนุกรมในแบบซิงโครนัส เมื่อรับเข้ามาแล้วจะต้องจัดการข้อมูลใหม่ให้อยู่ในรูปแบบขนาน 128 บิต เพื่อที่ส่งต่อให้กับส่วน AES CORE ไปประมวลผล ภายในคอมโพเน็นต์ DATA INPUT ประกอบด้วยคอมโพเน็นต์ย่อยๆ ดังนี้

1. Counter 7 bit
2. Shift reg 128

3. reg 128

<u>คำอธิบายขาสัญญาณ</u>

ON/OFF (in) : เป็นสัญญาณเปิดปิดการทำงานของคอมโพเน็นต์

Shift clk (in) : เป็นสัญญาณนาฬิกาที่ใช้ในการเลื่อนข้อมูลเข้า

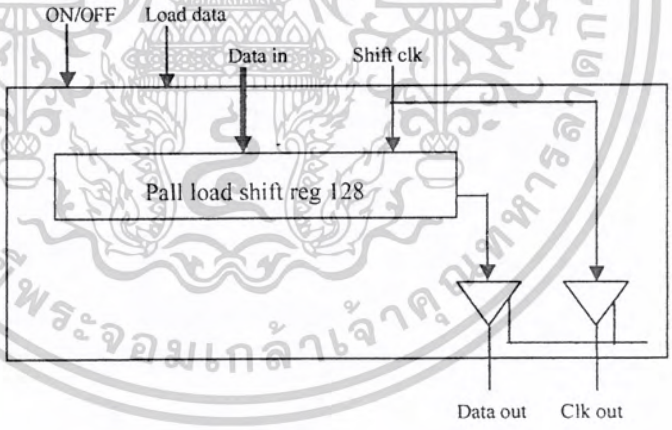
Data in (in) : เป็นสัญญาณข้อมูลซึ่งซิงโครไนซ์กับสัญญาณนาฬิกา Shift clk

Data out (out) : เป็นสัญญาณข้อมูลขาออกส่งให้กับ AES CORE มีขนาด 128 บิต

Data out valid (out) : เป็นสัญญาณที่ส่งไปยังส่วนควบคุม (AES CORE) ซึ่งจะใช้ในการควบคุมการเปิดคอมโพเน็นต์ AES CORE

Load of output (out) : เป็นสัญญาณที่ส่งไปยังส่วน DATA OUTPUT เพื่อใช้ในการโหลดข้อมูลเข้าไปยังส่วน DATA OUTPUT

4.2.6 DATA OUTPUT (Level 1)



รูปที่ 4.11 แสดงเอ็นทิตี้ของ DATA OUTPUT

คอมโพเน็นต์ DATA OUTPUT ทำหน้าที่รับสัญญาณที่ผ่านการเข้าหรือถอดรหัสจาก AES CORE เพื่อนำมาจัดให้อยู่ในลักษณะข้อมูลอนุกรมแล้วส่งออกไปภายนอก ซึ่งประกอบด้วยคอมโพเน็นต์ย่อยดังนี้

1. Pall load shift Reg 128

2. Tri state buffer 2 bit

คำอธิบายขาสัญญาณ

ON/OFF (in) : เป็นสัญญาณที่ใช้เปิดปิดการทำงานของคอมโพเน็นต์

Load data (in) : เป็นสัญญาณที่ส่งมาจากส่วน DATA INPUT เพื่อใช้ในการโหลดข้อมูลที่ผ่านการเข้าหรือถอดรหัสแล้วเข้าไปในคอมโพเน็นต์

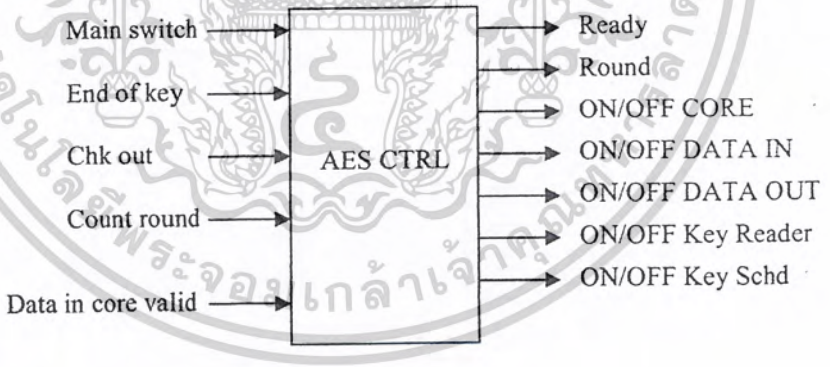
Data in (in) : เป็นสัญญาณข้อมูลขาเข้าที่ผ่านการเข้ารหัสหรือถอดรหัสจาก AES CORE มาแล้ว

Shift clk (in) : เป็นสัญญาณนาฬิกาที่ใช้ในการเลื่อนข้อมูลออกทีละ 1 บิต เพื่อให้เป็นข้อมูลในลักษณะอนุกรม

Data out (out) : เป็นสัญญาณข้อมูลขาออก ซึ่งเป็นข้อมูลอนุกรม

Clk out (out) : เป็นสัญญาณนาฬิกาซึ่งส่งออกไปภายนอกให้ซิงโครไนซ์กับข้อมูล Data out

4.2.7 คอมโพเน็นต์ AES CTRL (Level 1)



รูปที่ 4.12 แสดงเอ็นทิตี้ของ AES CTRL

คอมโพเน็นต์ AES CTRL ทำหน้าที่เป็นส่วนควบคุมการทำงานของระบบให้ระบบทำงานได้อย่างถูกต้อง ในการควบคุมก็จะเป็นการควบคุมการปิด เปิด การทำงานของคอมโพเน็นต์อื่น ๆ

คำอธิบายขาสัญญาณ

Main switch (in) : เป็นสัญญาณควบคุมการปิดเปิดการทำงานของระบบ

End of Key (in) : เป็นสัญญาณที่ส่งมาจากภายนอกในส่วนของการส่ง Key ใช้ในการแจ้งให้ระบบทราบว่าระบบได้รับ Key เสร็จสิ้นแล้ว

Chk out (in) : เป็นสัญญาณที่ส่งมาจากระบบภายนอกทางฝั่งเอาท์พุท เพื่อบอกว่าพร้อมที่จะรับข้อมูลแล้ว ซึ่งจะต้องตรวจสอบก่อนที่จะรับข้อมูลมาเข้าหรือถอดรหัส

Ready (out) : เป็นสัญญาณที่ส่งออกไปยังระบบภายนอกทางฝั่งอินพุท เพื่อบอกว่าพร้อมที่จะรับข้อมูลมาเข้าหรือถอดรหัสแล้ว

Count round (in) : เป็นสัญญาณที่ส่งมาจาก AES CORE เพื่อทำการนับรอบในการคำนวณ

Round (out) : เป็นสัญญาณที่มีขนาด 4 บิต ใช้บอกรอบในการคำนวณให้กับ AES CORE และ Key Scheduler

Data in core valid (in) : เป็นสัญญาณที่ส่งมาจากส่วน DATA INPUT เพื่อบอกว่ามีข้อมูลพร้อมที่จะเข้าหรือถอดรหัสแล้ว ซึ่ง AES CTRL ก็จะไปสั่งให้ AES CORE ทำงานต่อไป

ON/OFF CORE (out) : เป็นสัญญาณที่ส่งไปควบคุมการปิดเปิดคอมโพเน็นต์ AES CORE

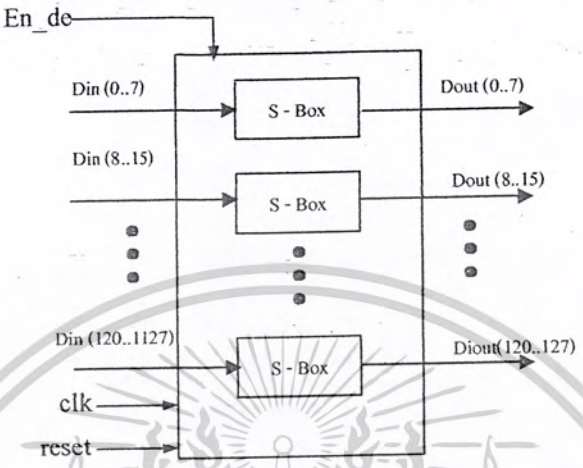
ON/OFF DATA IN (out) : เป็นสัญญาณที่ส่งไปควบคุมการปิดเปิดคอมโพเน็นต์ DATA INPUT

ON/OFF DATA OUT (out) : เป็นสัญญาณที่ส่งไปควบคุมการปิดเปิดคอมโพเน็นต์ DATA OUTPUT

ON/OFF Key reader (out) : เป็นสัญญาณที่ส่งไปควบคุมการปิดเปิดคอมโพเน็นต์ Key reader

ON/OFF Key schd (out) : เป็นสัญญาณที่ส่งไปควบคุมการปิดเปิดคอมโพเน็นต์ Key Scheduler

4.2.8 คอมโพเน็นต์ S – Box 128 bit (Level 2)



รูปที่ 4.13 แสดงเอ็นทิตี้ของ S – Box 128 bit

คอมโพเน็นต์ S – Box 128 bit จะทำหน้าที่รับสัญญาณอินพุทขนาด 128 บิต มาแปลงแบบ SubByte ซึ่งภายใน คอมโพเน็นต์ S – Box 128 bit ก็จะประกอบด้วยคอมโพเน็นต์ S – Box 16 ชุด เรียงขนานกัน โดย S-Box แต่ละอันจะรับข้อมูล 8 บิต ไปแปลงแบบ SubByte

<u>คำอธิบายขาสัญญาณ</u>

Din (in) : เป็นสัญญาณข้อมูลอินพุทขนาด 128 บิต ที่จะนำไปแปลงแบบ SubByte

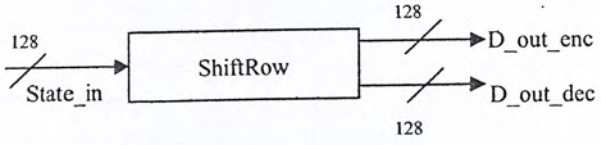
Dout (out) : เป็นสัญญาณข้อมูลเอาท์พุทขนาด 128 บิต ที่ผ่านการแปลงแบบ SubByte แล้ว

En/De (in) : (ดูในคอมโพเน็นต์ AES chip ที่ระดับบนสุด (สัญญาณ mode))

Clk (in) : เป็นสัญญาณนาฬิกาที่ใช้กำหนดจังหวะการทำงาน

Reset (in) : เป็นสัญญาณสำหรับรีเซตค่ารีจิสเตอร์ภายในคอมโพเน็นต์

4.2.9 คอมโพเน็นต์ ShiftRow (Level 2)



รูปที่ 4.14 แสดงเอ็นทิตี้ของ ShiftRow

คอมโพเน็นต์ ShiftRow ทำหน้าที่แปลงข้อมูลอนิพุทแบบ ShiftRow ซึ่งจะให้เอาท์พุทออกมา 2 ชุด ชุดหนึ่งสำหรับการ Encrypt และอีกชุดหนึ่งสำหรับการ Decrypt

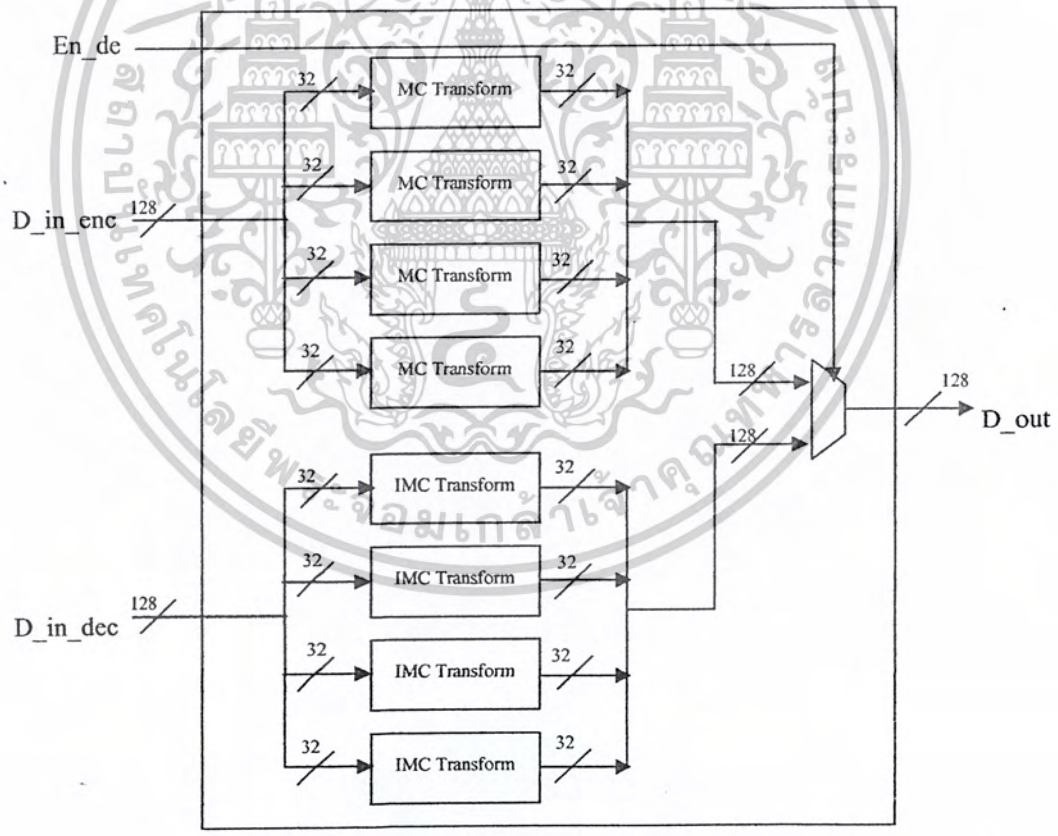
<u>คำอธิบายขาสัญญาณ</u>

State in (in) : เป็นสัญญาณินพุทมีขนาด 128 บิต

Dout enc (out) : เป็นสัญญาณเอาท์พุทที่ผ่านการแปลงแบบ ShiftRow แล้วมีขนาด 128 บิต

Dout Dec (out) : : เป็นสัญญาณเอาท์พุทที่ผ่านการแปลงแบบ Inv Shift Row แล้วมีขนาด 128 บิต

4.2.10 คอมโพเน็นต์ MixColumn (Level 2)



รูปที่ 4.15 แสดงเอ็นทิตี้ของ MixColumn

คอมโพเน็นต์ MixColumn จะรับสัญญาณอินพุทเข้ามา 2 ชุด 1 ชุดจะถูกนำไปแปลงแบบ MixColumn และอีก 1 ชุดจะถูกนำไปแปลงแบบ Inv Mixcolumn ซึ่งภายในคอมโพเน็นต์ Mixcolumn จะประกอบด้วย คอมโพเน็นต์ต่าง ๆ ดังนี้

1. MC Transform
2. IMC Transform
3. MUX 128

<u>คำอธิบายขาสัญญาณ</u>

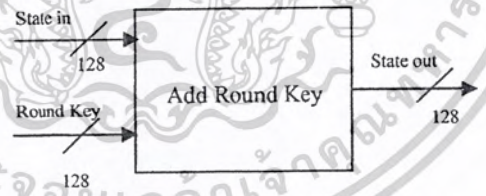
Din enc (in) : เป็นสัญญาณข้อมูลอินพุทมีขนาด 128 บิต สัญญาณนี้จะถูกนำไปแปลงแบบ MixColumn

Din dec (in) : เป็นสัญญาณข้อมูลอินพุทมีขนาด 128 บิต สัญญาณนี้จะถูกนำไปแปลงแบบ InvMixColumn

Dout (out) : เป็นสัญญาณข้อมูลเอาท์พุทมีขนาด 128 บิต

Enc/Dec (in) : (ดูในคอมโพเน็นต์ AES chip ที่ระดับบนสุด (สัญญาณ mode))

4.2.11 คอมโพเน็นต์ AddRoundKey (Level 2)



รูปที่ 4.16 แสดงเอ็นทิตี้ของ AddRoundKey

คอมโพเน็นต์ AddRoundKey ทำหน้าที่แปลงสัญญาณข้อมูลอินพุทแบบ AddRoundKey

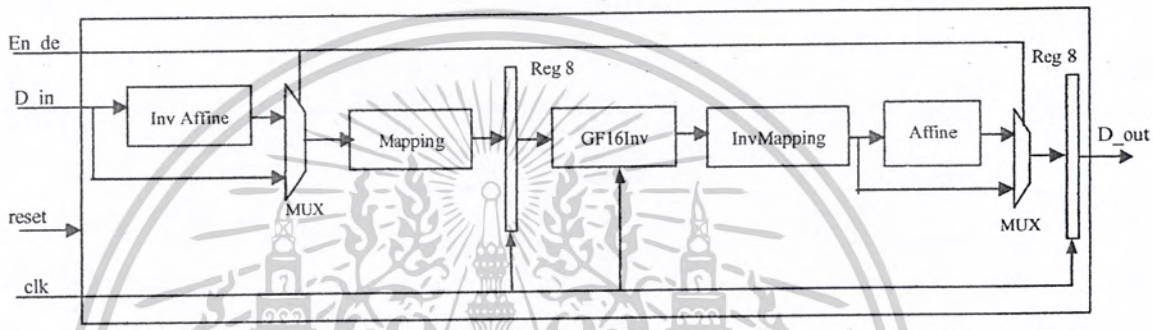
<u>คำอธิบายขาสัญญาณ</u>

State in (in) : เป็นสัญญาณข้อมูลอินพุทที่จะนำไปแปลงแบบ AddRoundKey มีขนาด 128 บิต

State out (out) : สัญญาณข้อมูลเอาท์พุทที่ได้จากการแปลงแบบ AddRoundKey มีขนาด 128 บิต

Round Key (in) : เป็นสัญญาณ Round Key ที่สร้างโดย Key Scheduler

4.2.12 คอมโพเน็นต์ S-Box (Level 3)



รูปที่ 4.17 แสดงเอ็นทิตี้ของ S-Box

คอมโพเน็น S-Box จะรับข้อมูลขนาด 8 บิตมาทำการแปลงแบบ SubByte ซึ่งภายใน S-Box ประกอบด้วยคอมโพเน็นต์ย่อยดังนี้

1. Inv Affine
2. Mapping
3. GF 16 inv
4. Inv Mapping
5. Affine
6. MUX
7. Reg 8

<u>คำอธิบายขาสัญญาณ</u>

Din (in) : เป็นสัญญาณอินพุทขนาด 8 บิต

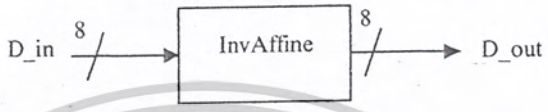
Dout (out) : เป็นสัญญาณเอาท์พุทขนาด 8 บิต ที่ผ่านการแปลงระบบ SubByte แล้ว

Reset (in) : ใช้เป็นสัญญาณรีเซตการทำงานของรีจิสเตอร์

Clk (in) : เป็นสัญญาณนาฬิกาควบคุมจังหวะการทำงานของคอมโพเน็นต์ (ใช้กับรีจิสเตอร์)

Enc/Dec (in) : (ดูในคอมโพเน็นต์ AES chip ที่ระดับบนสุด (สัญญาณ mode))

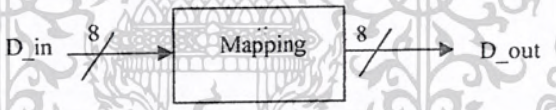
4.2.13 คอมโพเน็นต์ InvAffine (Level 4)



รูปที่ 4.18 แสดงเอ็นทิตี้ของ InvAffine

คอมโพเน็นต์ InvAffine ทำหน้าที่ Inverse Affine Transform ซึ่งเป็นหนึ่งในขั้นตอนการแปลงแบบ InvSubByte ในกระบวนการถอดรหัส

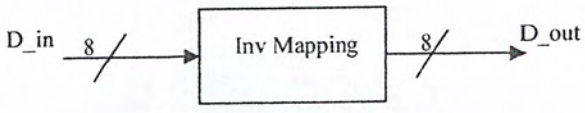
4.2.14 คอมโพเน็นต์ Mapping (Level 4)



รูปที่ 4.19 แสดงเอ็นทิตี้ของ Mapping

คอมโพเน็นต์ Mapping ทำหน้าที่แปลงข้อมูลอินพุทซึ่งอยู่ใน $GF(2^8)$ ให้เป็น $GF((2^4)^2)$

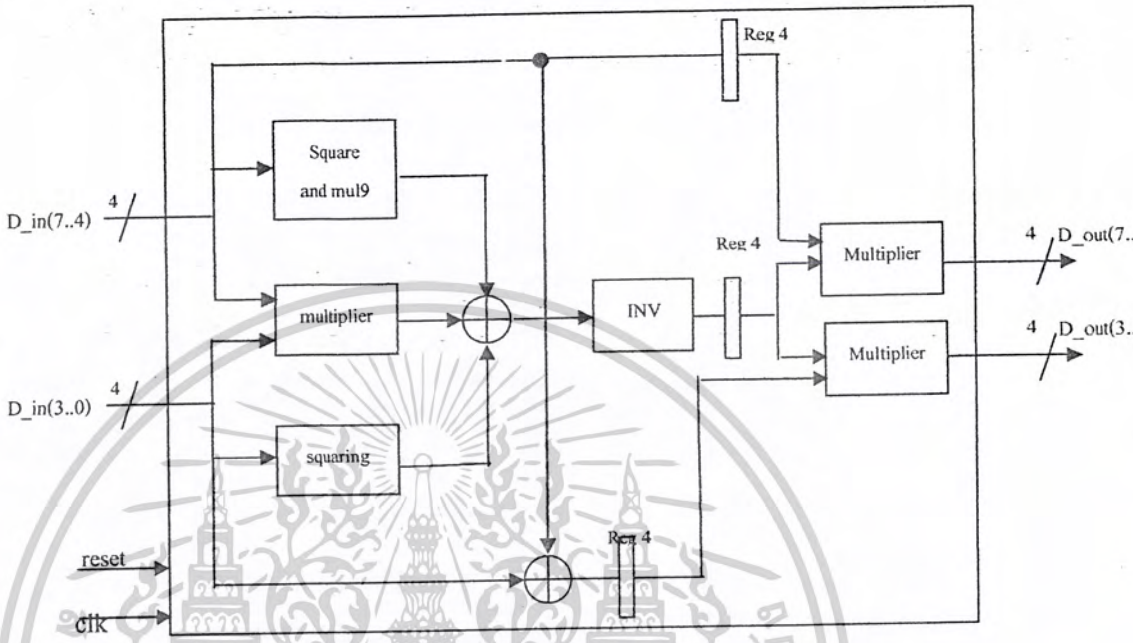
4.2.15 คอมโพเน็นต์ Inv Mapping (Level 4)



รูปที่ 4.20 แสดงเอ็นทิตี้ของ InvMapping

คอมโพเน็นต์ Inv Mapping ทำหน้าที่แปลงข้อมูลอินพุทซึ่งอยู่ใน $GF(2^4)^2$) ให้เป็น $GF(2^8)$

4.2.16 คอมโพเน็นต์ GF16 inv (Level 4)

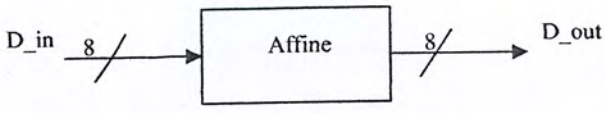


รูปที่ 4.21 แสดงเอ็นทิตี้ของ GF16 inv

คอมโพเน็นต์ GF16 inv ทำหน้าที่หาอินเวอร์สทางการคูณของข้อมูลอินพุทซึ่งอยู่ใน GF($(2^4)^2$) ภายใน คอมโพเน็นต์ GF16 inv ประกอบด้วย คอมโพเน็นต์ย่อยดังนี้

1. Multiplier : ใช้ในการคูณจำนวนสองจำนวนใน GF(2^4)
2. Squaring : ใช้สำหรับยกกำลังสองของตัวเลขใน GF(2^4)
3. Square and mul 9 : ใช้สำหรับยกกำลังสองแล้วคูณด้วย 9 ใน GF(2^4)
4. XOR : ใช้สำหรับบวกเลขใน GF(2^4)
5. INV : ใช้หาอินเวอร์สทางการคูณของตัวเลขใน GF(2^4)

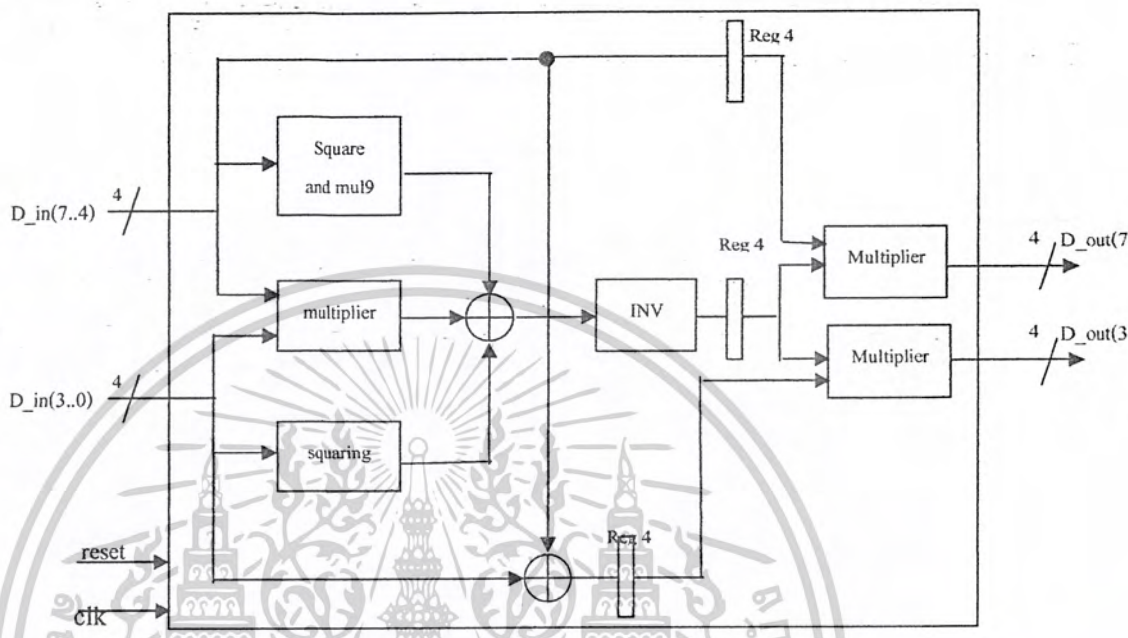
4.2.17 คอมโพเน็นต์ Affine (Level 4)



รูปที่ 4.22 แสดงเอ็นทิตี้ของ Affine

คอมโพเน็นต์ Affine ทำหน้าที่ Affine Transform ซึ่งเป็นหนึ่งในขั้นตอนการแปลงแบบ SubByte

4.2.16 คอมโพเน็นต์ GF16 inv (Level 4)

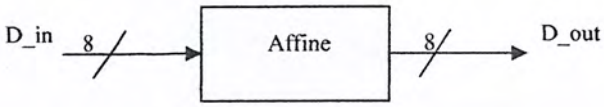


รูปที่ 4.21 แสดงเอ็นทิตี้ของ GF16 inv

คอมโพเน็นต์ GF16 inv ทำหน้าที่หาอินเวอร์สทางการคูณของข้อมูลอินพุทซึ่งอยู่ใน GF($(2^4)^2$) ภายใน คอมโพเน็นต์ GF16 inv ประกอบด้วย คอมโพเน็นต์ย่อยดังนี้

1. Multiplier : ใช้ในการคูณจำนวนสองจำนวนใน GF(2^4)
2. Squaring : ใช้สำหรับยกกำลังสองของตัวเลขใน GF(2^4)
3. Square and mul 9 : ใช้สำหรับยกกำลังสองแล้วคูณด้วย 9 ใน GF(2^4)
4. XOR : ใช้สำหรับบวกเลขใน GF(2^4)
5. INV : ใช้หาอินเวอร์สทางการคูณของตัวเลขใน GF(2^4)

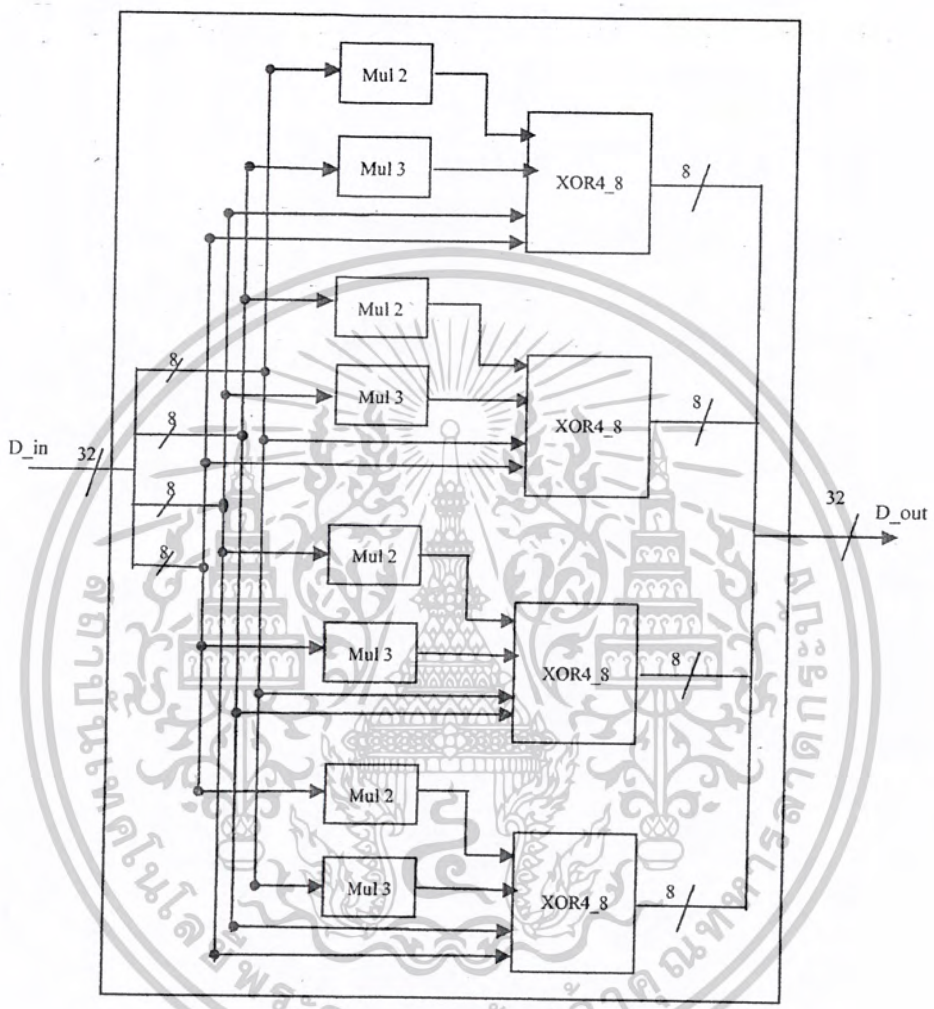
4.2.17 คอมโพเน็นต์ Affine (Level 4)



รูปที่ 4.22 แสดงเอ็นทิตี้ของ Affine

คอมโพเน็นต์ Affine ทำหน้าที่ Affine Transform ซึ่งเป็นหนึ่งในขั้นตอนการแปลงแบบ SubByte

4.2.18 คอมโพเน็นต์ MC Transform (Level 3)

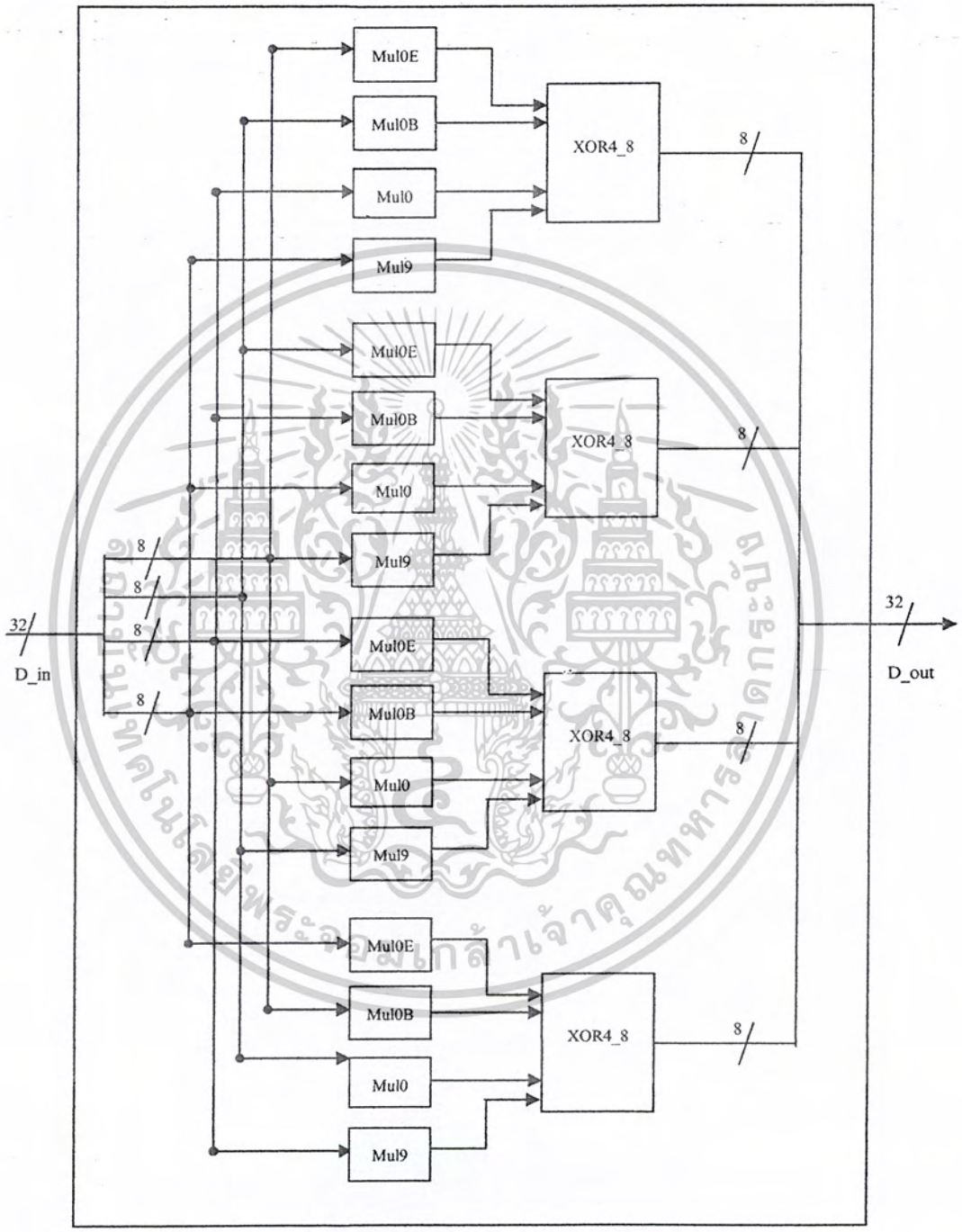


รูปที่ 4.23 แสดงเอ็นทิตี้ของ MC Transform

คอมโพเน็นต์ MC Transform จะรับข้อมูลขนาด 32 บิตมาทำการแปลงแบบ MixColumn ซึ่งภายใน MC Transform ประกอบด้วยคอมโมเน็นต์ย่อยดังนี้

1. mul3 : สัญญาณที่ผ่านคอมโพเน็นต์จะถูกถูณด้วย (03)
2. mul2 : สัญญาณที่ผ่านคอมโพเน็นต์นี้จะถูกคูณด้วย (02)
3. XOR4_8 : ใช้สำหรับบวกจำนวน 4 จำนวนใน GF (2^8) ซึ่งแต่ละจำนวนมีขนาด 8 บิต

4.2.19 คอมโพเน็นต์ IMC Transform (Level 3)

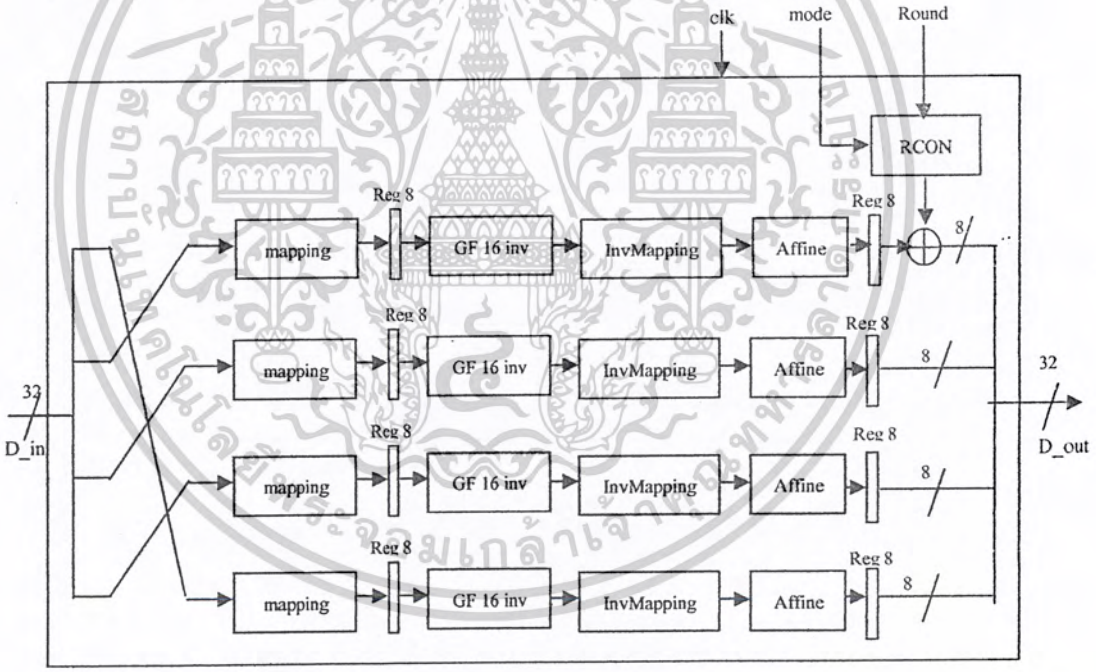


รูปที่ 4.24 แสดงเอ็นทิตี้ของ IMC Transform

คอมโพเน็นต์ IMC Transform จะรับข้อมูลขนาด 32 บิต มาทำการแปลงแบบ InvMixColumn ซึ่งภายใน IMC Transform ประกอบด้วยคอมโพเน็นต์ย่อยดังนี้

1. mul9 : สัญญาณที่ผ่านคอมโพเน็นต์นี้จะถูกคูณด้วย (09)
2. mul0B : สัญญาณที่ผ่านคอมโพเน็นต์นี้จะถูกคูณด้วย (0B)
3. mul0D : สัญญาณที่ผ่านคอมโพเน็นต์นี้จะถูกคูณด้วย (0D)
4. mul0E : สัญญาณที่ผ่านคอมโพเน็นต์นี้จะถูกคูณด้วย (0E)
5. XOR4_8 : ใช้สำหรับบวกจำนวน 4 จำนวนใน GF (2^8) ซึ่งแต่ละจำนวนมีขนาด 8 บิต

4.2.20 คอมโพเน็นต์ S-Box Key (Level 2)



รูปที่ 4.25 แสดงเอ็นทิตี้ของ S-Box Key

คอมโพเน็นต์ S-Box Key เป็นส่วนหนึ่งของคอมโพเน็นต์ Key Scheduler ซึ่งได้รวมฟังก์ชั่นการ ShiffByte, SubWORK และAddRcon เอาไว้ แสดงดังรูปด้านบน ภายใน S-Box Key

ประกอบด้วยคอมโพเน็นต์ย่อย ๆ คือ Mapping, Reg8, GF 16 inv, Inv Mapping, Affine, XOR และ Rcon

<u>คำอธิบายขาสัญญาณ</u>

Din (in) : เป็นสัญญาณข้อมูลอินพุทมีขนาด 32 บิต

Dout (out) : เป็นสัญญาณข้อมูลเอาท์พุทมีขนาด 32 บิต

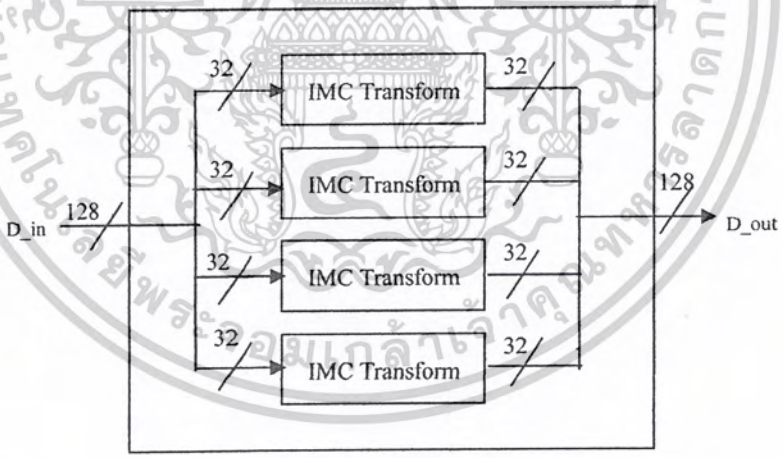
Reset (in) : เป็นสัญญาณสำหรับรีเซตค่าของรีจิสเตอร์ภายใน S-Box Key

Clk (in) : เป็นสัญญาณนาฬิกาที่ใช้กำหนดจังหวะการทำงานของคอมโพเน็นต์ (ใช้กับรีจิสเตอร์)

Round (in) : เป็นสัญญาณบอกรอบในการคำนวณ เพื่อป้อนให้กับ Rcon มีขนาด 4 บิต

En/de (in) : (ดูในคอมโพเน็นต์ AES chip ที่ระดับบนสุด (สัญญาณ mode))

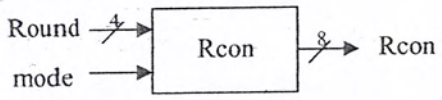
4.2.21 คอมโพเน็นต์ IMC 128 (Level 2)



รูปที่ 4.26 แสดงเอ็นทิตี้ของ IMC 128

คอมโพเน็นต์ IMC128 ประกอบด้วยคอมโพเน็นต์ IMC Transform 4 ชุด ขนานกันทำหน้าที่ InvMixColumn Transform คอมโพเน็นต์นี้เป็นส่วนหนึ่งของ Key Scheduler ใช้กับโหมด Decryption เท่านั้น

4.2.22 คอมโพเน็นต์ Rcon (Level 3)



รูปที่ 4.27 แสดงเอ็นทิตี้ของ Rcon

โพเน็นต์ Rcon เป็นส่วนหนึ่งของ Key Scheduler ทำหน้าที่สร้าง ค่าคงที่ของรอบ (Round constant) ซึ่งในแต่ละรอบจะมีค่าไม่เท่ากัน ในกระบวนการสร้าง Round Key ตามอัลกอริทึม AES

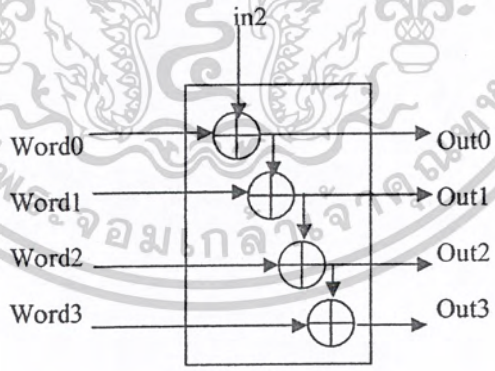
คำอธิบายขาสัญญาณ

Round (in) : เป็นสัญญาณบอกรอบในการคำนวณ มีขนาด 4 บิต

Mode (in) : (ดูในคอมโพเน็นต์ AES chip ที่ระดับบนสุด (สัญญาณ mode))

Rcon (out) : คือสัญญาณเอาท์พุทขนาด 8 บิต เป็นค่าของ Rcon ในรอบต่าง ๆ

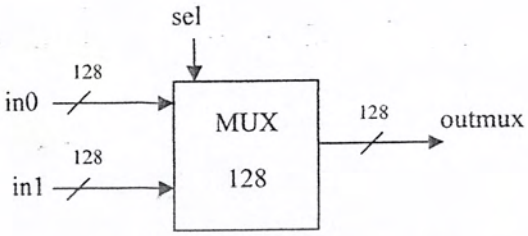
4.2.23 คอมโพเน็นต์ XOR in keysch (Level 2)



รูปที่ 4.28 แสดงเอ็นทิตี้ของ XOR in keysch

คอมโพเน็นต์ XOR in Keysch เป็นส่วนหนึ่งของ Key Scheduler แสดงฟังก์ชั่นการทำงานดังรูปด้านบน

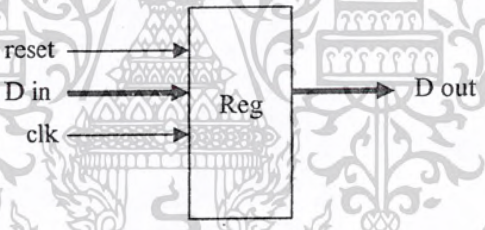
4.2.24 คอมโพเน็นต์ MUX 128



รูปที่ 4.29 แสดงเอ็นทิตี้ของ MUX 128

MUX 128 ทำหน้าที่เป็นมัลติเพล็กเซอร์ขนาด 128 บิต โดยถ้าสัญญาณ sel มีค่าเป็น High จะเลือก in1 ออกไปที่เอาท์พุท แต่ถ้า sel มีค่าเป็น Low จะเลือก in0 ออกไปที่เอาท์พุทแทน

4.2.25 คอมโพเน็นต์ Reg

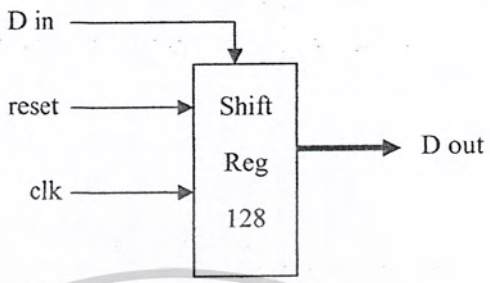


รูปที่ 4.30 แสดงเอ็นทิตี้ของ Reg

Reg ทำหน้าที่เป็นรีจิสเตอร์ ซึ่งในระบบนี้มีอยู่หลายขนาด เช่น Reg8 คือ รีจิสเตอร์ขนาด 8 บิต หรือ Reg 128 ก็คือ รีจิสเตอร์ขนาด 128 บิต เป็นต้น

การทำงานจะอาศัยสัญญาณนาฬิกา โดยเมื่อตรวจพบสัญญาณนาฬิกาเป็นขอบขาลง ก็จะนำข้อมูล D in ไปเก็บไว้ในรีจิสเตอร์ ส่วนค่าของ D out ก็คือค่าของข้อมูลที่อยู่ในรีจิสเตอร์นั่นเอง

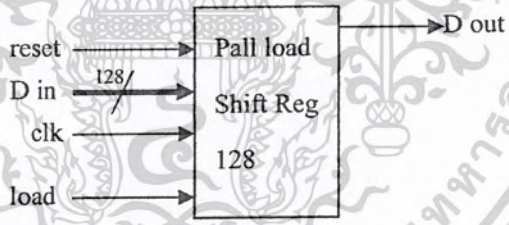
4.2.26 คอมโพเน็นต์ Shift Reg 128



รูปที่ 4.31 แสดงเอ็นทิตี้ของ Shift Reg 128

Shift Reg ทำหน้าที่เป็นชิฟรีจิสเตอร์ การทำงานจะอาศัยสัญญาณนาฬิกา โดยเมื่อตรวจพบสัญญาณนาฬิกาเป็นขอบขาลง ก็จะนำข้อมูล D in ซึ่งเป็นข้อมูลอนุกรม ไปเก็บไว้ในรีจิสเตอร์ ส่วนค่าของ D out ก็คือค่าของข้อมูลที่อยู่ในรีจิสเตอร์นั่นเอง

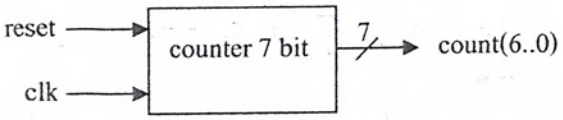
4.2.27 คอมโพเน็นต์ Pall load Shift Reg 128



รูปที่ 4.32 แสดงเอ็นทิตี้ของ Pall load Shift Reg 128

Pall load Shift Reg 128 ทำหน้าที่เป็นชิฟรีจิสเตอร์ การทำงานจะอาศัยสัญญาณนาฬิกา โดยเมื่อตรวจพบสัญญาณนาฬิกาเป็นขอบขาลง และสัญญาณ load เป็น High ก็จะนำข้อมูล D in ไปเก็บไว้ในรีจิสเตอร์ แต่ถ้าสัญญาณนาฬิกาเป็นขอบขาลงและสัญญาณ load ไม่เป็น High ก็จะเป็นการเลื่อนข้อมูลออกมาที่เอาท์พุท D out ทีละ 1 บิต

4.2.28 คอมโพเน็นต์ Counter 7 bit



รูปที่ 4.33 แสดงเอ็นทิตี้ของ Counter 7 bit

Counter 7 bit ถูกใช้เป็นตัวหารความถี่ของสัญญาณนาฬิกาในส่วน DATA INPUT เพื่อใช้ในการแปลงข้อมูลอนุกรมเป็นข้อมูลขนานขนาด 128 บิต กล่าวคือจะใช้สัญญาณ count(6) เป็นสัญญาณนาฬิกาให้กับรีจิสเตอร์ Reg

คอมโพเน็นต์ Count3 ก็เพื่อเป็นตัวหารความถี่เช่นกัน ซึ่งถูกนำไปใช้เป็นสัญญาณนาฬิกาให้กับ AES CORE และ Key Scheduler ส่วน counter 2 bit นั้นใช้ในการนับรอบในการคำนวณ

บทที่ 5
การทดลอง และผลการทดลอง

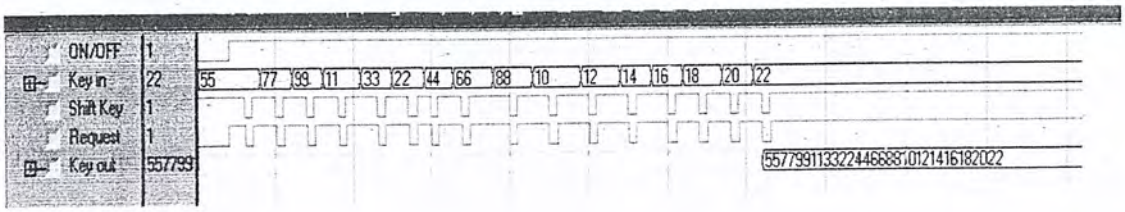
ในบทนี้จะเป็นการกล่าวถึงส่วนของการทดสอบการทำงานของวงจรที่ถูกออกแบบว่าสามารถทำงานตามที่ออกแบบได้หรือไม่ การทดสอบจะทำโดยการจำลองการทำงาน(Simulation) ของวงจรเพื่อดูผลการตอบสนองของสัญญาณต่างๆ

5.1 การจำลองการทำงาน(Simulation)

หลังจากได้ออกแบบวงจรโดยการอธิบายลักษณะพฤติกรรมของวงจร(Hardware Description Language) เป็นที่เรียบร้อยแล้วผู้ออกแบบจำเป็นจะต้องทำการจำลองการทำงานเพื่อดูผลการตอบสนองของสัญญาณต่างๆของวงจรต่ออินพุทที่กำหนดให้ ซึ่งอ้างอิงจากตัวอย่างการเข้ารหัสใน [3] และได้คัดลอกตัวอย่างดังกล่าวไว้ในภาคผนวก ง แล้ว ซึ่งหากผลลัพธ์ที่ได้ไม่ถูกต้องจะได้ทำการแก้ไขต่อไป สำหรับโครงงานนี้ได้ใช้โปรแกรมจำลองการทำงาน ModelSim 5.6 SE ควบคู่กับ ISE WEBPack 6.1 ในการจำลองการทำงาน ซึ่งได้กล่าวถึงวิธีการใช้งานอย่างคร่าวๆไว้ในภาคผนวก ค

5.1.1 ส่วนอ่านรหัสกุญแจ(Key Reader)

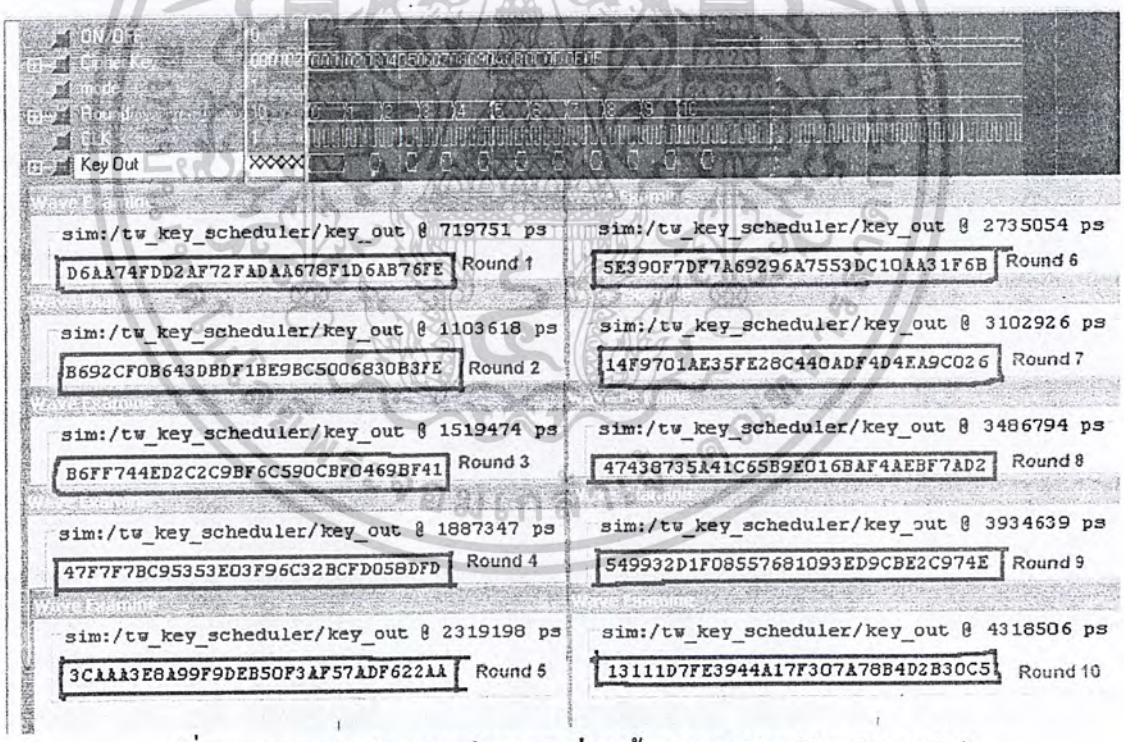
ส่วนอ่านรหัสกุญแจจะทำหน้าที่ร้องขอรหัสกุญแจ(Key) และอ่านรหัสกุญแจจากระบบภายนอก พร้อมทั้งจัดสรรรูปแบบรหัสกุญแจให้อยู่ในรูปแบบ 128 บิต เพื่อพร้อมใช้งานต่อในส่วนสร้าง Round Key (Key Scheduler) ในการร้องขอรหัสกุญแจ ส่วนอ่านรหัสกุญแจจะส่งสัญญาณ Request เป็น High ไปยังส่วนที่จะส่งรหัสเข้ามา และเมื่อรับรหัสเข้ามาแล้วสัญญาณ Request จะเป็น Low เพื่อบ่งบอกว่าได้รับรหัสชุดนั้นแล้ว ในการรับรหัสกุญแจเข้าไป จะรับทีละ 8 บิต โดยระบบภายนอกที่ส่งรหัสกุญแจเข้ามาจะต้องส่งสัญญาณมาชิฟข้อมูล (รหัสกุญแจ) เข้าไปด้วย หลังจากที่ส่งครบ 128 บิตแล้ว(ส่ง 16 ครั้ง) จะต้องส่งสัญญาณ End of Key มาที่ AES CTRL เพื่อบอกว่าการส่งรหัสกุญแจเสร็จสิ้นแล้ว



รูปที่ 5.1 แสดงผลการจำลองการทำงานของส่วนอ่านรหัสกุญแจ (Key Reader)

5.1.2 ส่วนสร้าง Round Key (Key Scheduler)

ส่วนสร้าง Round Key จะทำหน้าที่สร้าง Round Key ให้กับส่วน AES CORE เพื่อใช้ในการแปลงแบบ AddRoundKey โดยจะรับอินพุทมาจากส่วนอ่านรหัสกุญแจ (Key Reader) เพื่อนำมหา Key ในรอบต่อไป ซึ่งจะใช้เวลาเท่ากับสัญญาณนาฬิกา 4 ลูกต่อการคำนวณ 1 รอบ



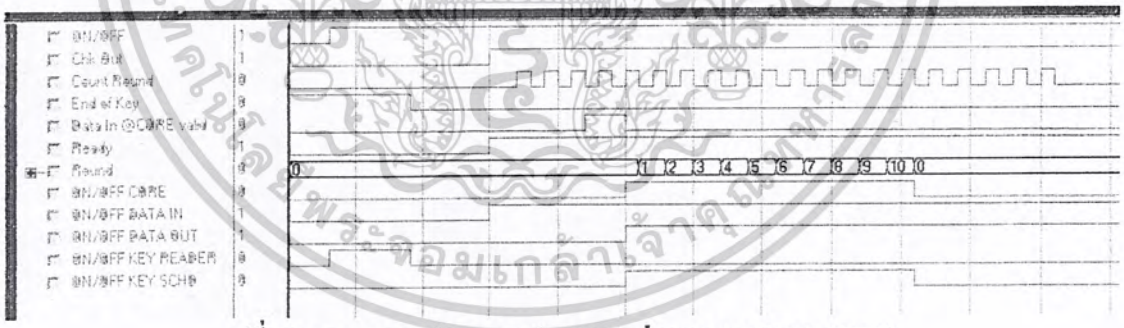
รูปที่ 5.2 แสดงผลการจำลองการทำงานของส่วนสร้าง Round Key (Key Scheduler)

5.1.3 ส่วนควบคุม (AES CTRL)

ส่วนควบคุมจะทำหน้าที่ควบคุมการทำงานของระบบให้สามารถทำงานได้อย่างถูกต้อง ซึ่งการควบคุมโดยหลัก ๆ ก็คือจะควบคุม การปิดเปิดคอมโพเน็นต์ทั้งห้าของระบบ คือ Key Reader, Key Scheduler, DATA INPUT,AES CORE และ DATA OUTPUT

ในส่วนการทำงานนั้นเมื่อระบบเริ่มทำงาน ส่วนควบคุมนี้จะส่งสัญญาณไปเปิดคอมโพเน็นต์ Key Reader ทันที เพื่อรับ Key และเมื่อรับ Key เสร็จ ก็จะส่งสัญญาณไปปิด ต่อจากนั้นก็จะตรวจสอบสถานะที่ภาคถัดไปทางด้านเอาท์พุท ว่าพร้อมที่จะรับข้อมูลหรือยัง ถ้าพร้อม ภาคดังกล่าวจะส่งสัญญาณ High มายังส่วนควบคุม หลังจากนั้นส่วนควบคุมจึงส่งสัญญาณ Ready เป็น High เพื่อบอกระบบภายนอกว่าพร้อมรับข้อมูลแล้ว พร้อมกันนั้นก็จะส่งสัญญาณไปเปิดคอมโพเน็นต์ DATA INPUT ด้วย

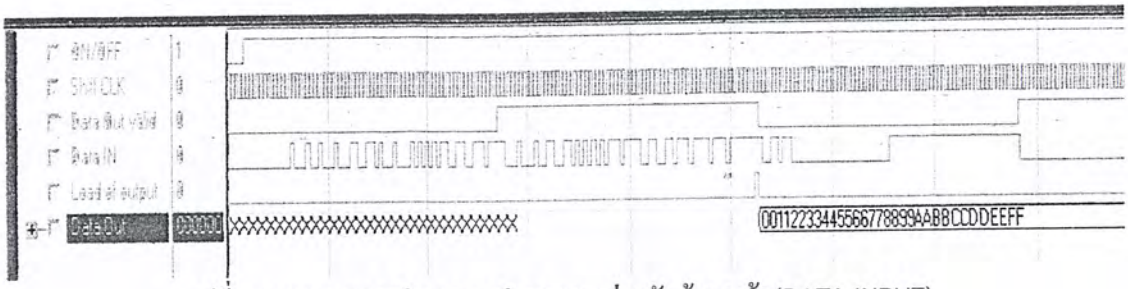
หลังจากที่ DATA INPUT รับข้อมูลครบ 128 บิตแล้วจะส่งสัญญาณไปบอกให้ส่วนควบคุมเปิดการทำงาน AES CORE ซึ่งขณะที่ AES CORE ทำงานก็จะนับรอบให้กับส่วนควบคุมด้วย ทันทีที่รอบการคำนวณเปลี่ยนจาก 10 เป็น 0 (AES CORE ทำงานเสร็จสิ้น) ส่วนควบคุมก็จะส่งสัญญาณมาปิดการทำงานของ AES CORE ส่วนของ DATA OUTPUT นั้นจะถูกเปิดการทำงานเมื่อข้อมูล 128 บิตที่สองเข้ามาที่ DATA INPUT ครบและจะถูกปิดการทำงานเมื่อระบบถูกปิดการทำงาน



รูปที่ 5.3 แสดงผลการจำลองการทำงานของส่วนควบคุม (AES CTRL)

5.1.4 ส่วนรับข้อมูลเข้า (DATA INPUT)

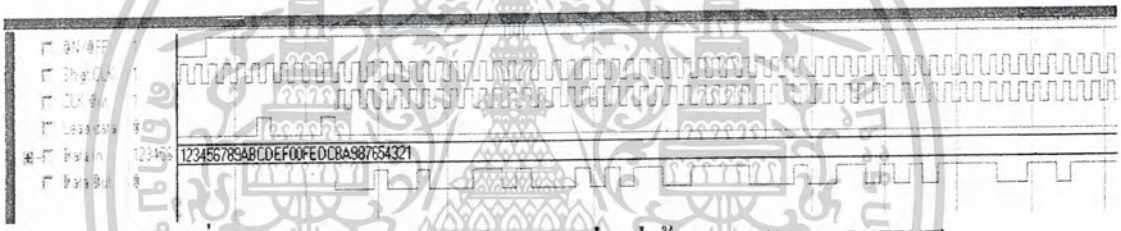
ส่วนรับข้อมูลเข้าจะทำหน้าที่รับข้อมูลจากระบบภายนอก ซึ่งส่งข้อมูลเข้ามาในแบบอนุกรมแบบซิงโครนัส และจะต้องทำการแปลงข้อมูลที่เป็นอนุกรมนั้นให้เป็นแบบขนาน 128 บิต เพื่อส่งต่อให้กับ AES CORE ประมวลผลต่อไป



รูปที่ 5.4 แสดงผลการจำลองการทำงานของส่วนรับข้อมูลเข้า (DATA INPUT)

5.1.5 ส่วนส่งข้อมูลออก (DATA OUTPUT)

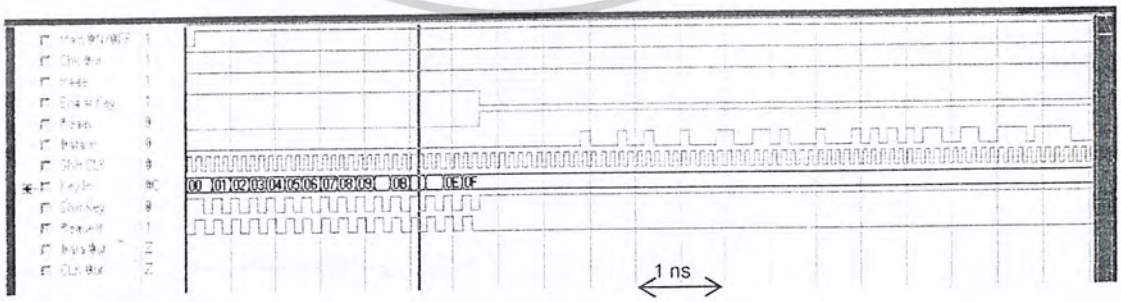
ส่วนส่งข้อมูลออกจะทำหน้าที่รับข้อมูลจาก AES CORE ซึ่งเป็นข้อมูลแบบขนาน 128 บิต แล้วแปลงข้อมูลดังกล่าวให้เป็นแบบอนุกรมแบบซิงโครนัส เพื่อส่งออกไปยังระบบภายนอก



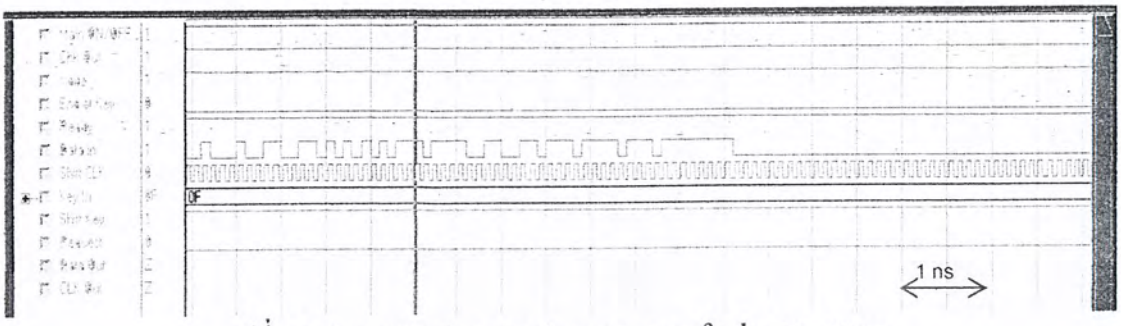
รูปที่ 5.5 แสดงผลการจำลองการทำงานของส่วนส่งข้อมูลออก (DATA OUTPUT)

5.1.6 จำลองการทำงานทั้งระบบ (Top Level simulation)

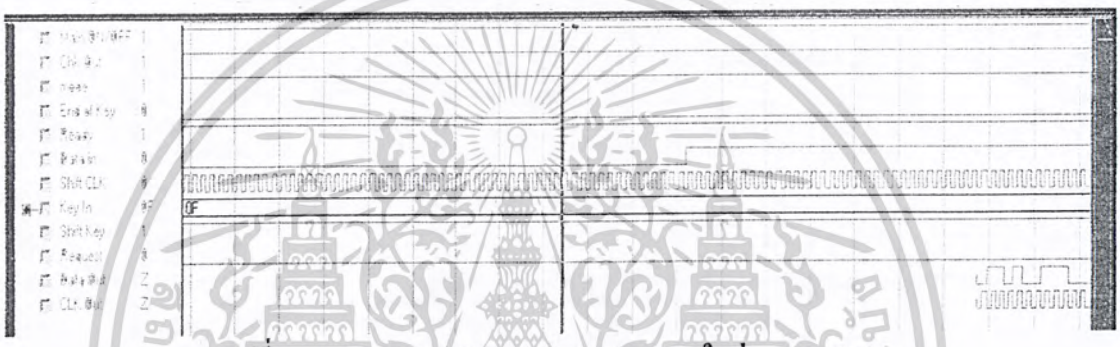
หลังจากที่แต่ละส่วนของระบบสามารถทำงานได้อย่างถูกต้องแล้ว ขั้นตอนต่อไปก็จะต้องนำทุก ๆ ส่วนมาประกอบเข้าด้วยกันเพื่อดูผลลัพธ์สุดท้ายว่าถูกต้องหรือไม่ ซึ่งการทำงานของระบบก็เป็นไปตามโฟลวชาร์ตรูปที่ 4.4 ดังที่ได้อธิบายไปแล้วในบทที่ 4



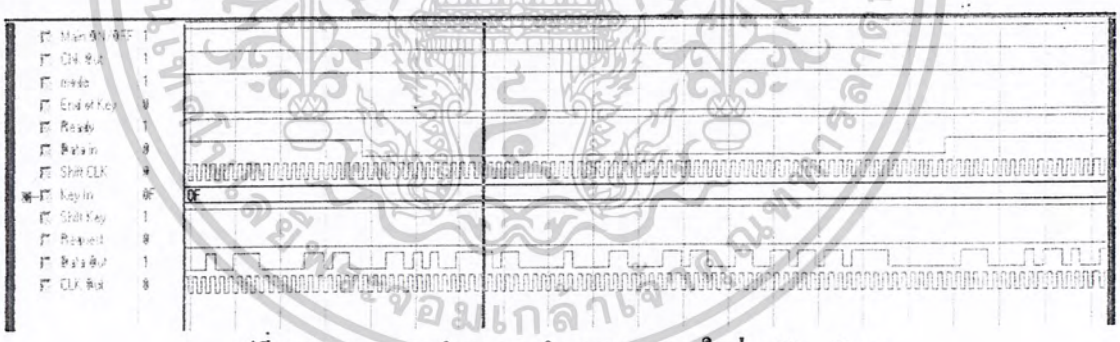
รูปที่ 5.6 แสดงผลการจำลองการทำงานของระบบ ในช่วง 0 – 10 ns



รูปที่ 5.7 แสดงผลการจำลองการทำงานของระบบ ในช่วง 10 – 20 ns



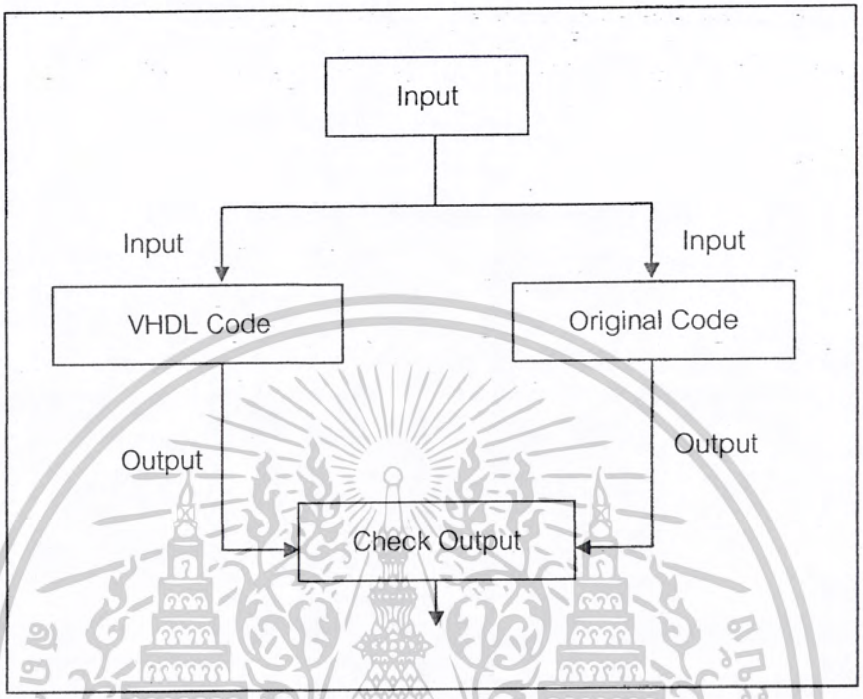
รูปที่ 5.8 แสดงผลการจำลองการทำงานของระบบ ในช่วง 20 – 30 ns



รูปที่ 5.9 แสดงผลการจำลองการทำงานของระบบ ในช่วง 30 – 40 ns

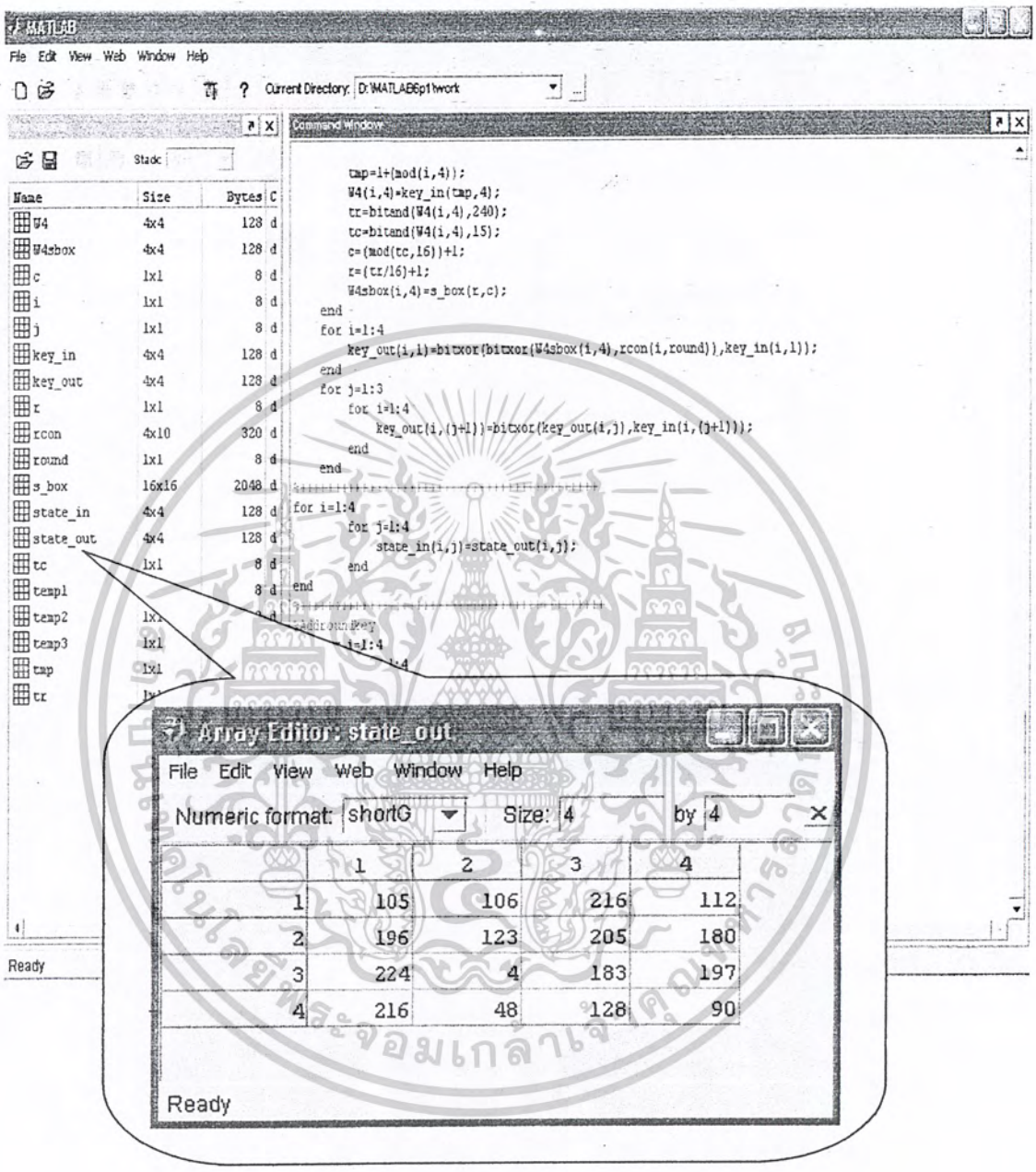
5.2 การทดสอบผล

สำหรับการวิเคราะห์ผลการทดลองสามารถทำได้โดยนำโค้ดวีเอชดีแอลที่ได้ออกแบบไว้จำลองการทำงานดังในหัวข้อที่ผ่านมา โดยให้รับค่าอินพุตค่าหนึ่ง จากนั้นนำโค้ดภาษาใดก็ได้ที่เป็นอัลกอริทึมแบบเดียวกันมาเอ็กซิคิวต์ โดยรับค่าอินพุตที่เหมือนกันแล้ว ดูผลลัพธ์ว่าได้ผลลัพธ์ตรงกันหรือไม่ ดังแสดงรูปที่ 5.10



รูปที่ 5.10 แสดงขั้นตอนการตรวจสอบความถูกต้องของโค้ด

สำหรับ Original Code ในโครงงานนี้ ได้เขียนขึ้นโดยใช้ MATLAB Code ซึ่งแสดงไว้ในภาคผนวก ข โดย โค้ด Project_enc.m สำหรับการเข้ารหัส และ Project_dec.m สำหรับการถอดรหัส ส่วนโค้ดอื่นๆเป็นฟังก์ชั่นที่ใช้ใน M-file ทั้งสองข้างต้น ดังนั้นการนำโค้ดดังกล่าวมาเอ็กซิคิวต์ต้องกระทำบนโปรแกรม MATLAB โดยการ Copy โค้ดดังกล่าวแล้ว Paste ไว้ที่ Command window จากนั้นก็กดปุ่ม Enter ก็จะปรากฏผลลัพธ์ที่ Work Space ดังรูปที่ 5.11 นอกจากนั้นยังสามารถอ้างอิงกับตัวอย่างการเข้ารหัส หรือถอดรหัสในภาคผนวก ง ได้



รูปที่ 5.11 แสดงผลลัพธ์จากการเอ็กซิคิวต์ M-file Project_enc หรือ Project_dec

5.3 สรุปผลการทดลอง

จากผลการจำลองการทำงานของโค้ดภาษาวีเอชดีแอลที่เขียนขึ้น เปรียบเทียบกับผลลัพธ์ที่เขียนจาก MATLAB และ ตัวอย่างการเข้าและถอดรหัสในภาคผนวก ง ปรากฏว่าผลลัพธ์ทั้งสามสามารถให้ผลได้ตรงกัน จึงสามารถสรุปได้ว่า โค้ดภาษาวีเอชดีแอลที่มาจากการออกแบบวงจรสามารถทำงานเป็นตัวเข้ารหัสหรือถอดรหัสได้ตามวัตถุประสงค์ที่ต้องการ

บทที่ 6
บทสรุป

6.1 ขั้นตอนการทำโครงงาน

ขั้นแรกจะต้องศึกษาและทำความเข้าใจอัลกอริทึมของ AES เสียก่อน จากความเข้าใจดังกล่าวจะต้องตอบให้ได้ว่า ขั้นตอนในการเข้ารหัสหรือถอดรหัสทั้งหมดประกอบด้วยฟังก์ชั่นการทำงานอะไรบ้าง และแต่ละฟังก์ชั่นนั้นทำงานอย่างไร พยายามเขียนเป็นบล็อกไดอะแกรมออกมาจะเห็นภาพมากขึ้น ขั้นตอนต่อไปคือ ศึกษาและทำความเข้าใจภาษาวีเอชดีแอล ว่ามีองค์ประกอบอย่างไร มีฟังก์ชั่นอะไรให้ใช้บ้าง และจะเขียนอย่างไรให้ถูกต้อง ต่อมาจะต้องนำบล็อกไดอะแกรมที่เราเขียนเอาไว้มาเชื่อมโยงกับภาษาวีเอชดีแอลให้ได้ คือการนำภาษาวีเอชดีแอลมาบรรยายการทำงานของวงจรโดยให้คุณลักษณะ Top – Down Design เข้าช่วย และหลังจากที่แต่ละบล็อกถูกออกแบบเสร็จแล้ว จะต้องกลับไปแก้ไขจนกว่าจะถูกต้อง

6.2 ปัญหาและการแก้ไข

หากงานใดไม่มีปัญหาให้แก้ไขแสดงว่างานนั้นไม่ก่อให้เกิดการเรียนรู้ .. โครงงานนี้ก็มีปัญหาเช่นกัน ปัญหาที่สำคัญของโครงงานนี้ เกี่ยวกับความเข้าใจในการออกแบบวงจรดิจิตอล และภาษาวีเอชดีแอล เนื่องจากวงจรใช้สัญญาณนาฬิกาในการกำหนดจังหวะการทำงาน การซิงค์สัญญาณจะต้องเป็นไปอย่างถูกต้อง มิฉะนั้นจะทำให้ผลลัพธ์ที่ออกมาผิดพลาด และในส่วนของภาษาวีเอชดีแอล มักจะเขียนผิดข้อกำหนดของภาษาทำให้เมื่อคอมไพล์ออกมาแล้วผิดพลาด และที่สำคัญคือจะต้องเขียนให้วงจรทำงานได้เร็วและมีขนาดเล็ก ซึ่งวิธีที่จะเขียนแล้ววงจรทำงานได้เร็วขึ้นนั้นจะต้องไม่เขียนคำสั่งที่เป็นซีเควนเชียล คำสั่งที่เป็นคอนเคอร์เรนท์ จะทำให้วงจรทำงานได้เร็วขึ้น

6.3 แนวทางการพัฒนาโครงงาน

เมื่อได้นำโค้ดภาษาวีเอชดีแอลไป Synthesis ปรากฏว่าวงจรยังมีขนาดใหญ่อยู่ วิธีหนึ่งที่จะทำให้เล็กลงคือ เปลี่ยนระบบประมวลผลที่เป็น 128 บิต ให้ลดลงเป็น 64 หรือ 32 บิต ก็ช่วยลดพื้นที่ลดได้ เนื่องจากระบบนี้ยังต้องทำงานในแบบซิงโครนัส หากพัฒนาให้สามารถทำงานได้ทั้งแบบซิงโครนัสและอะซิงโครนัสได้ก็จะดีมากขึ้น

ภาคผนวก

ภาคผนวก ก

โค้ดภาษาวีเอชดีแอลของวงจรที่ออกแบบ

```
----------------------------------------------TOP LEVEL----------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY AES_iteration IS
      port( main_switch :in std_logic;
            chk_out         :in std_logic;
            mode       :in std_logic;
            End_of_key  :in std_logic;
            ready      :out std_logic:='0';
            Data_IN         :in std_logic;
            shift_clk      :in std_logic;
            Data_out        :out std_logic:='Z';
            clk_out         :out std_logic:='Z';
            key_in          :in std_logic_vector(7 downto 0);
            shift_key      :in std_logic;
            request         :out std_logic:='1'
            );
END AES_iteration ;

ARCHITECTURE struct OF AES_iteration IS
      COMPONENT KEY_READER
      PORT (
            switch_ON_OFF: in std_logic;
            byte_in: in std_logic_vector(7 downto 0);
            shift_key: in std_logic;
            request: out std_logic:='1';
            KEY: out std_logic_vector(0 to 127)
      );
      END COMPONENT ;
      COMPONENT AES_CTRL
      port(
            main_switch :in std_logic;
            chk_out          :in std_logic;
            count_round :in std_logic;
            End_of_key  :in std_logic; ---if end --> =='0'
            data_in_core_valid:in std_logic;
            ready      :out std_logic;
            round      :buffer std_logic_vector(3 downto 0);
            switch_core :buffer std_logic:='0';
            switch_data_in:out std_logic:='0';
            switch_data_out:out std_logic :='0';
            switch_key_reader:out std_logic:='0';
            switch_key_schd:out std_logic:='0');
      END COMPONENT ;
      COMPONENT DATA_INPUT
```

```
port(
      switch_on_off     :in std_logic;
      shift_clk         :in std_logic;
      Data_out_valid    :out std_logic;
      Data_IN           :in std_logic;
      load_of_output    :out std_logic;
      Data_out          :out std_logic_vector(0 to 127)
      );
END COMPONENT ;
COMPONENT DATA_OUTPUT
port(
      switch_on_off     :in std_logic;
      shift_clk        :in std_logic;
      clk_out           :out std_logic;
      load_data         :in std_logic;
      Data_in           :in std_logic_vector(0 to 127);
      Data_out         :out std_logic);
END COMPONENT ;
COMPONENT AES_CORE
port (
      switch_on_off    :in std_logic;
      ip_clk          :in std_logic;
      mode    :in std_logic;
      round   :in std_logic_vector(3 downto 0);
      Data_in          :in std_logic_vector(0 to 127);
      count   :out std_logic;
      round_key :in std_logic_vector(0 to 127);
      Data_out    :out std_logic_vector(0 to 127)
      );
END COMPONENT ;
COMPONENT KEY_SCHEDULER
port(
      switch_on_off:in std_logic;
      cipher_key  :in std_logic_vector(0 to 127);
      mode    :in std_logic;
      round   :in std_logic_vector(3 downto 0);
      ip_clk  :in std_logic;
      KEY_out  :out std_logic_vector(0 to 127)
      );
END COMPONENT ;
COMPONENT count3
port( clk   :in std_logic;
      clk_out :out std_logic
      );
END COMPONENT ;

      Signal
clk,sw_core,sw_data_in,sw_data_out,sw_key_reader,sw_key_schd,count_roun
d:std_logic;
      Signal Data_valid,Ld:std_logic;
      Signal round_key,user_key,plaintext,ciphertext:std_logic_vector(0
to 127);
      Signal round:std_logic_vector(3 downto 0);
```

```
BEGIN
      AES1:KEY_READER PORT MAP
(switch_ON_OFF=>sw_key_reader,byte_in=>key_in,shift_key=>shift_key,requ
est=>request,     KEY=>user_key);
      AES2:AES_CTRL     port map(main_switch=>main_switch,
      chk_out=>chk_out, count_round=>count_round,
            End_of_key=>End_of_key,data_in_core_valid=>Data_valid
      ,ready=>ready,    round=>round      ,

      switch_core=>sw_core,switch_data_in=>sw_data_in,switch_data_out=>
sw_data_out,

      switch_key_reader=>sw_key_reader,switch_key_schd=>sw_key_schd);
      AES3:DATA_INPUT port map(switch_on_off=>sw_data_in,
shift_clk=>shift_clk,Data_out_valid=>Data_valid,
            Data_IN=>Data_IN,load_of_output=>Ld,
      Data_out=>plaintext);
      AES4:DATA_OUTPUT port map(switch_on_off=>sw_data_out,
      shift_clk=>shift_clk,clk_out=>clk_out,load_data=>Ld,
            Data_in=>ciphertext, Data_out=>Data_out   );
      AES5:AES_CORE     port map(switch_on_off=>sw_core,ip_clk=>clk,
      mode=>mode, round=>round      ,     Data_in=>plaintext,
            count=>count_round, round_key=>round_key,
Data_out=>ciphertext    );
      AES6:KEY_SCHEDULER      port map(switch_on_off=>sw_key_schd,
      cipher_key=>user_key,   mode=>mode,round=>round ,
            ip_clk=>clk, KEY_out=>round_key );
      AES7:count3 port map(clk=>shift_clk, clk_out=>clk);
END struct;
```

LEVEL 1

- AES CORE

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY AES_CORE IS
      port (
            switch_on_off     :in std_logic;
            ip_clk        :in std_logic;
            mode    :in std_logic;
            round   :in std_logic_vector(3 downto 0);
            Data_in       :in std_logic_vector(0 to 127);
            count   :out std_logic;
            round_key :in std_logic_vector(0 to 127);
            Data_out     :out std_logic_vector(0 to 127)
            );
END AES_CORE ;

ARCHITECTURE behave OF AES_CORE IS
```

```
COMPONENT S_BOX128bit_beh2
port( din  : in std_logic_vector(0 to 127);
      reset: in std_logic;
      clk  : in std_logic;
      en_de: in std_logic; --'1'=>encrypt , '0' => decrypt
      dout : out std_logic_vector(0 to 127)
    );
END COMPONENT ;
COMPONENT shiftrow
PORT (      state_in: in std_logic_vector(0 to 127);
      d_out_enc: out std_logic_vector(0 to 127);
      d_out_dec: out std_logic_vector(0 to 127) );
END COMPONENT ;
COMPONENT MixColumn_beh
PORT (
      d_in_enc: in std_logic_vector(0 to 127);
      d_in_dec: in std_logic_vector(0 to 127);
      d_out: out std_logic_vector(0 to 127);
      en_de: in std_logic      );
END COMPONENT ;
COMPONENT AddRoundkey
PORT( RoundKey,state    : in std_logic_vector(0 to 127) ;
      state_out         : out std_logic_vector (0 to 127) );
END COMPONENT ;
COMPONENT reg128_it
PORT (
      sw_on_off: in std_logic;
      d_in: in std_logic_vector(0 to 127);
      d_out: out std_logic_vector(0 to 127);
      clk: in std_logic
);
END COMPONENT ;
COMPONENT counter2bit
port( clk,reset   :in std_logic;
      count4      :out std_logic
      );
END COMPONENT ;
COMPONENT mux2_inputcore_it
port( init ,innext : in std_logic_vector(0 to 127);
      outmux : out std_logic_vector(0 to 127);
      sel : in std_logic_vector(3 downto 0));
END COMPONENT ;
COMPONENT Last_round_mux
port( inlast ,inbe4 : in std_logic_vector(0 to 127);
      outmux : out std_logic_vector(0 to 127);
      sel : in std_logic_vector(3 downto 0));
END COMPONENT ;
COMPONENT mux2_128
port( in0 ,in1 : in std_logic_vector(0 to 127);
      outmux : out std_logic_vector(0 to 127);
      sel : in std_logic);
END COMPONENT ;
```

```
      SIGNAL
ARK_out,mux_ip_out,output,SB_out,SR_out_enc,SR_out_dec,LRD,MC_out,ToARK
:std_logic_vector(0 to 127);
      SIGNAL clk:std_logic;
BEGIN
      clk<= ip_clk and switch_on_off;

      CORE1:mux2_inputcore_it port map(   init=>Data_in ,innext=>ToARK,
      outmux=>mux_ip_out,     sel=>round);
      CORE2:AddRoundkey PORT MAP(
      RoundKey=>round_key,state=>mux_ip_out,state_out=>ARK_out);
      CORE3:reg128_it PORT MAP(sw_on_off=>switch_on_off,d_in=>ARK_out,
      d_out=>output,    clk=>clk);
      CORE4:S_BOX128bit_beh2 port map(    din=>output,
reset=>switch_on_off    ,     clk=>clk, en_de=>mode,  dout=>SB_out);
      CORE5:shiftrow PORT MAP(      state_in=>SB_out,
      d_out_enc=>SR_out_enc,  d_out_dec=>SR_out_dec);
      CORE6:MixColumn_beh     PORT MAP (d_in_enc=>SR_out_enc,
      d_in_dec=>SR_out_dec,   d_out=>MC_out,    en_de=>mode);
      CORE7:mux2_128    port map(
      in0=>SR_out_dec ,in1=>SR_out_enc,outmux=>LRD,   sel=>mode);
      CORE8:Last_round_mux port map(      inlast=>LRD ,inbe4=>MC_out,
outmux=>ToARK,    sel=>round);
      CORE9:counter2bit port map(   clk=>clk,reset=>switch_on_off,
      count4=>count);

      DATA_out<=output;

END behave;
```

- KEY READER

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY KEY_READER IS

      PORT (
            switch_ON_OFF: in std_logic;
            byte_in: in std_logic_vector(7 downto 0);
            shift_key: in std_logic;
            request: out std_logic:='1';
            KEY: out std_logic_vector(0 to 127)
      );

END KEY_READER;
ARCHITECTURE rtl OF KEY_READER IS
      COMPONENT request_sig_ctrl
      port( shift:in std_logic;
            sw_on_off:in std_logic;
            request:out std_logic
            );
```

```
      SIGNAL
ARK_out,mux_ip_out,output,SB_out,SR_out_enc,SR_out_dec,LRD,MC_out,ToARK
:std_logic_vector(0 to 127);
      SIGNAL clk:std_logic;
BEGIN
      clk<= ip_clk and switch_on_off;

      CORE1:mux2_inputcore_it port map(   init=>Data_in ,innext=>ToARK,
      outmux=>mux_ip_out,     sel=>round);
      CORE2:AddRoundkey PORT MAP(
      RoundKey=>round_key,state=>mux_ip_out,state_out=>ARK_out);
      CORE3:reg128_it PORT MAP(sw_on_off=>switch_on_off,d_in=>ARK_out,
      d_out=>output,    clk=>clk);
      CORE4:S_BOX128bit_beh2 port map(    din=>output,
reset=>switch_on_off    ,     clk=>clk, en_de=>mode,  dout=>SB_out);
      CORE5:shiftrow PORT MAP(      state_in=>SB_out,
      d_out_enc=>SR_out_enc,  d_out_dec=>SR_out_dec);
      CORE6:MixColumn_beh     PORT MAP (d_in_enc=>SR_out_enc,
      d_in_dec=>SR_out_dec,   d_out=>MC_out,    en_de=>mode);
      CORE7:mux2_128    port map(
      in0=>SR_out_dec ,in1=>SR_out_enc,outmux=>LRD,   sel=>mode);
      CORE8:Last_round_mux port map(      inlast=>LRD ,inbe4=>MC_out,
outmux=>ToARK,    sel=>round);
      CORE9:counter2bit port map(   clk=>clk,reset=>switch_on_off,
      count4=>count);

      DATA_out<=output;

END behave;
```

- KEY READER

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY KEY_READER IS

      PORT (
            switch_ON_OFF: in std_logic;
            byte_in: in std_logic_vector(7 downto 0);
            shift_key: in std_logic;
            request: out std_logic:='1';
            KEY: out std_logic_vector(0 to 127)
      );

END KEY_READER;
ARCHITECTURE rtl OF KEY_READER IS
      COMPONENT request_sig_ctrl
      port( shift:in std_logic;
            sw_on_off:in std_logic;
            request:out std_logic
            );
```

```
	END COMPONENT ;
	COMPONENT reg8_it_KR
	PORT (
		sw_on_off: in std_logic;
		d_in: in std_logic_vector(7 downto 0);
		d_out: out std_logic_vector(7 downto 0);
		clk: in std_logic
	);
	END COMPONENT ;

	SIGNAL a,b,c,d,e,f,g,h,i,j,k,l,m,n,o : std_logic_vector( 7 downto
0)   ;

BEGIN
	KR1:request_sig_ctrl port map(
	shift=>shift_key,sw_on_off=>switch_ON_OFF,request=>request);


	KR3:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>byte_in,    d_out=>a,    clk=>shift_key);
	KR4:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>a,    d_out=>b,    clk=>shift_key);
	KR5:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>b,    d_out=>c,    clk=>shift_key);
	KR6:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>c,    d_out=>d,    clk=>shift_key);
	KR7:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>d,    d_out=>e,    clk=>shift_key);
	KR8:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>e,    d_out=>f,    clk=>shift_key);
	KR9:reg8_it_KR    PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>f,    d_out=>g,    clk=>shift_key);
	KR10:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>g,    d_out=>h,    clk=>shift_key);
	KR11:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>h,    d_out=>i,    clk=>shift_key);
	KR12:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>i,    d_out=>j,    clk=>shift_key);
	KR13:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>j,    d_out=>k,    clk=>shift_key);
	KR14:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>k,    d_out=>l,    clk=>shift_key);
	KR15:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>l,    d_out=>m,    clk=>shift_key);
	KR16:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>m,    d_out=>n,    clk=>shift_key);
	KR17:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>n,    d_out=>o,    clk=>shift_key);
	KR18:reg8_it_KR   PORT MAP(sw_on_off=>switch_ON_OFF,
	d_in=>o,    d_out=>KEY(0 to 7),    clk=>shift_key);

	KEY(8 to 15)<=o;
	KEY(16 to 23)<=n;
```

```
      KEY(24 to 31)<=m;
      KEY(32 to 39)<=l;
      KEY(40 to 47)<=k;
      KEY(48 to 55)<=j;
      KEY(56 to 63)<=i;
      KEY(64 to 71)<=h;
      KEY(72 to 79)<=g;
      KEY(80 to 87)<=f;
      KEY(88 to 95)<=e;
      KEY(96 to 103)<=d;
      KEY(104 to 111)<=c;
      KEY(112 to 119)<=b;
      KEY(120 to 127)<=a;

END rtl;
```

-AES CTRL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY AES_CTRL IS
      port(
            main_switch :in std_logic;
            chk_out     :in std_logic;
            count_round :in std_logic;
            End_of_key  :in std_logic; ---if end --> ='0'
            data_in_core_valid:in std_logic;
            ready     :out std_logic;
            round     :buffer std_logic_vector(3 downto 0);
            switch_core :buffer std_logic:='0';
            switch_data_in:out std_logic:='0';
            switch_data_out:out std_logic :='0';
            switch_key_reader:out std_logic:='0';
            switch_key_schd:out std_logic:='0');
END AES_CTRL ;

ARCHITECTURE behave OF AES_CTRL IS

BEGIN

      ready <= (not(End_of_key)) and chk_out and main_switch;

      round_counter:process(switch_core,count_round)
                  variable count :std_logic_vector(3 downto 0);
                   begin
                     if switch_core = '0' then
                      count:=(others=>'0');
                      round<=count;
```

```
                    elsif (count_round'event and count_round ='0')
then
                     count:=count+1;
                     round<=count;
                    end if;
                  end process;


on_off_core_and_Keysch:process(data_in_core_valid,main_switch,round,cou
nt_round)
                  begin
                  if round="1011" then
                       switch_core<='0';
                       switch_key_schd <='0';

                  elsif round ="1010" then
                       if (count_round'event and count_round='0')then
                       switch_core<='0';
                       switch_key_schd <='0';
                       end if;

                  elsif  main_switch='1' then
                       if (data_in_core_valid'event and
data_in_core_valid='0')then
                       switch_core<='1';
                       switch_key_schd <='1';
                       end if;
                  end if;

            end process;

       switch_data_in <= (not(End_of_key)) and chk_out and main_switch;
       on_off_out:process(main_switch,data_in_core_valid)
                  begin
                       if main_switch ='0' then
                          switch_data_out<='0';
                       elsif (data_in_core_valid'event and
data_in_core_valid='0') then
                          switch_data_out<='1';
                       end if;
                  end process;

       switch_key_reader<= main_switch and End_of_key;

END behave;
```

- DATA INPUT

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY DATA_INPUT IS
      port(
            switch_on_off      :in std_logic;
            shift_clk          :in std_logic;
            Data_out_valid     :out std_logic;
            Data_IN            :in std_logic;
            load_of_output     :out std_logic;
            Data_out           :out std_logic_vector(0 to 127)
            );
END DATA_INPUT ;

ARCHITECTURE behave OF DATA_INPUT IS
      COMPONENT shift_reg128_it
      port(
            clk,sw_on_off,d_in:     in std_logic;
              shift_out: out std_logic_vector(0 to 127)
              );
      END COMPONENT ;
      COMPONENT counter7bit_it
      port(
            clk:in std_logic;
            reset:in std_logic;
            count:out std_logic_vector(6 downto 0)
      );
      END COMPONENT ;
      COMPONENT reg128_it

      PORT (
            sw_on_off: in std_logic;
            d_in: in std_logic_vector(0 to 127);
            d_out: out std_logic_vector(0 to 127);
            clk: in std_logic
      );
      END COMPONENT ;

      signal shift_out :std_logic_vector(0 to 127);
      signal count :std_logic_vector(6 downto 0);

BEGIN
      DTI1:shift_reg128_it    port
map(clk=>shift_clk,sw_on_off=>switch_on_off,
                              d_in=>Data_IN, shift_out=>shift_out);
      DTI2:counter7bit_it     port
map(clk=>shift_clk,reset=>switch_on_off,  count=>count);
      DTI3:reg128_it PORT MAP (sw_on_off=>switch_on_off,
      d_in=>shift_out,
                              d_out=>Data_out,  clk=>count(6));
```

```
      load_of_output<=(count(0) and count(1) and count(2)and
count(3)and count(4)and count(5) and count(6));
      Data_out_valid <= count(6);
END behave;
```

- DATA OUTPUT

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY DATA_OUTPUT IS
      port(
            switch_on_off     :in std_logic;
            shift_clk         :in std_logic;
            clk_out           :out std_logic;
            load_data         :in std_logic;
            Data_in           :in std_logic_vector(0 to 127);
            Data_out          :out std_logic);
END DATA_OUTPUT ;

ARCHITECTURE behave OF DATA_OUTPUT IS
      COMPONENT Tri_state_buffer2bit
      port(
            enable :in std_logic;
            input  :in std_logic_vector(1 downto 0);
            output :out std_logic_vector(1 downto 0));
      END COMPONENT ;
      COMPONENT Pall_load_shift_reg128
      Port( clk,reset,load:in std_logic;
            d_in  :in std_logic_vector(0 to 127);
            d_out :out std_logic
            );
      END COMPONENT ;

      signal shift_out,en_out : std_logic;
      signal enable_output : std_logic:='1';
      signal output,buff_out : std_logic_vector(1 downto 0);

BEGIN
      D_OUT1:Pall_load_shift_reg128 Port MAP(
      clk=>shift_clk,reset=>switch_on_off ,
                              load=>load_data,  d_in=>Data_in      ,
      d_out=>shift_out);
      D_OUT2:Tri_state_buffer2bit   port
map(enable=>en_out,input=>output,output=>buff_out);
      en_out <=(not enable_output) and switch_on_off;
      output<=shift_out & shift_clk;
      Data_out<=buff_out(1);
      clk_out     <=buff_out(0);
```

```
        process(load_data)
        variable count :integer:=0;
        begin
              if load_data'event and load_data = '0' then
                   count:= count +1 ;
                    if count > 2 then
                     enable_output <= '0';
                  end if;
              end if;
      end process;

END behave;
```

- KEY SCHEDULER

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY KEY_SCHEDULER IS
      port(
            switch_on_off:in std_logic;
            cipher_key  :in std_logic_vector(0 to 127);
            mode        :in std_logic;
            round       :in std_logic_vector(3 downto 0);
            --start          :in std_logic;
            ip_clk         :in std_logic;
            --finish         :out std_logic;
            KEY_out        :out std_logic_vector(0 to 127)
            );
END KEY_SCHEDULER ;

ARCHITECTURE struct OF KEY_SCHEDULER IS
      COMPONENT reg128_it
      PORT (
            sw_on_off: in std_logic;
            d_in: in std_logic_vector(0 to 127);
            d_out: out std_logic_vector(0 to 127);
            clk: in std_logic
            );
      END COMPONENT ;
      COMPONENT keyreg128_it
      PORT (
            d_in: in std_logic_vector(0 to 127);
            d_out: out std_logic_vector(0 to 127);
            clk: in std_logic
            );
      END COMPONENT ;
      COMPONENT XORinKEYsch_it
      port(
            word0 : in std_logic_vector(0 to 31);
            word1 : in std_logic_vector(0 to 31);
```

```
            word2 : in std_logic_vector(0 to 31);
            word3 : in std_logic_vector(0 to 31);
            i_p2 : in std_logic_vector(0 to 31);
            o_p0 : out std_logic_vector(0 to 31);
            o_p1 : out std_logic_vector(0 to 31);
            o_p2 : out std_logic_vector(0 to 31);
            o_p3 : out std_logic_vector(0 to 31));
      END COMPONENT ;
      COMPONENT mux2_32
      port( in0 ,in1 : in std_logic_vector(0 to 31);
            outmux : out std_logic_vector(0 to 31);
            sel : in std_logic);
      END COMPONENT ;
      COMPONENT S_BOX_Key_beh_it
      PORT (
            din: in std_logic_vector(0 to 31);
            reset: in std_logic;
            mode:in std_logic;
            clk: in std_logic;
            round:in std_logic_vector(3 downto 0);
            dout: out std_logic_vector(0 to 31)
      );
      END COMPONENT ;
      COMPONENT XOR32
      PORT (
            in0: in std_logic_vector(0 to 31);
            y: out std_logic_vector(0 to 31);
            in1: in std_logic_vector(0 to 31)
      );
      END COMPONENT ;
      COMPONENT reg32_it
      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(0 to 31);
            d_out: out std_logic_vector(0 to 31);
            clk: in std_logic
      );
      END COMPONENT ;
      COMPONENT key_mux_it
      port (      sel: in std_logic_vector(3 downto 0)  ;--as 'round'
signal
            original_key, pre_roundkey : in std_logic_vector (0 to
127) ;
            key_muxed : out  std_logic_vector (0 to 127)) ;
      END COMPONENT ;
      COMPONENT IMC128_beh
      port (
            d_in : in std_logic_vector(0 to 127);
            d_out : out std_logic_vector(0 to 127)
            );
      END COMPONENT ;
      COMPONENT mux2_128
      port( in0 ,in1 : in std_logic_vector(0 to 127);
            outmux : out std_logic_vector(0 to 127);
```

```
            sel : in std_logic);
      END COMPONENT ;
      COMPONENT Last_round_mux
      port( inlast ,inbe4 : in std_logic_vector(0 to 127);
                  outmux : out std_logic_vector(0 to 127);
                  sel : in std_logic_vector(3 downto 0));
      END COMPONENT ;

      SIGNAL W0,W1,W2,W3,toShift,ShiftByte,toXOR,o0,o1,o2,o3,iW3 :
std_logic_vector(0 to 31);
      Signal
W00,W000,W0000,W11,W111,W1111,W22,W222,W2222,W33,W333,W3333:std_logic_v
ector(0 to 31);
      Signal
io0,io1,io2,io3,iW33,iW333,Wp0,Wp1,Wp2,Wp3 :std_logic_vector(0 to 31);
      SIGNAL Key_in,ToReg,out_ENC,keyout,out_DEC,IMCout:
std_logic_vector(0 to 127);
      SIGNAL clk :std_logic;

BEGIN
      clk<= ip_clk and switch_on_off;

      KeyMux:key_mux_it port map(   sel=>round,
      original_key=>cipher_key,
                        pre_roundkey=>out_ENC,
      key_muxed=>Key_in);
      W0 <= Key_in(0 to 31);
      W1 <= Key_in(32 to 63);
      W2 <= Key_in(64 to 95);
      W3 <= Key_in(96 to 127);

      WORD0_1:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W0,d_out=>W00);
      WORD0_2:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W00,d_out=>W000);
      WORD0_3:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W000,d_out=>W0000);

      WORD1_1:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W1,d_out=>W11);
      WORD1_2:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W11,d_out=>W111);
      WORD1_3:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W111,d_out=>W1111);

      WORD2_1:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W2,d_out=>W22);
      WORD2_2:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W22,d_out=>W222);
      WORD2_3:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W222,d_out=>W2222);

      WORD3_1:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W3,d_out=>W33);
```

```
      WORD3_2:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W33,d_out=>W333);
      WORD3_3:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>W333,d_out=>W3333);

      KeySch1:mux2_32 port map(     in0=>iW3,in1=>W3,outmux=>toShift,
      sel=>mode);

      ShiftByte<=toshift(8 to 31) & toshift(0 to 7);

      KeySch2:S_BOX_Key_beh_it PORT
MAP(din=>ShiftByte,reset=>switch_on_off,mode=>mode,
                                clk=>clk,round=>round,
      dout=>toXOR);
      KeySch3:XORinKEYsch_it  port
map(word0=>W0000,word1=>W1111,word2=>W2222,word3=>W3333,

      i_p2=>toXOR,o_p0=>o0,o_p1=>o1,o_p2=>o2,o_p3=>o3);

      KeySch4:XOR32 PORT MAP(in0=>W0000,   in1=>toXOR ,y=>io0);
      KeySch5:XOR32 PORT MAP(in0=>W0000,   in1=>W1111 ,y=>io1);
      KeySch6:XOR32 PORT MAP(in0=>W1111,   in1=>W2222 ,y=>io2);
      KeySch7:XOR32 PORT MAP(in0=>W2,      in1=>W3 ,y=>iW3);
      iWORD3_1:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>iW3,d_out=>iW33);
      iWORD3_2:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>iW33,d_out=>iW333);
      iWORD3_3:reg32_it PORT
MAP(reset=>switch_on_off,clk=>clk,d_in=>iW333,d_out=>io3);
      KeySch8:mux2_32 port map(     in0=>io0 ,in1=>o0, outmux=>Wp0,
      sel=>mode);
      KeySch9:mux2_32 port map(     in0=>io1 ,in1=>o1, outmux=>Wp1,
      sel=>mode);
      KeySch10:mux2_32 port map(    in0=>io2 ,in1=>o2,outmux=>Wp2,
      sel=>mode);
      KeySch11:mux2_32 port map(    in0=>io3 ,in1=>o3, outmux=>Wp3,
      sel=>mode);

      ToReg <= Wp0 & Wp1 & Wp2 & Wp3;
      KeySch12:keyreg128_it PORT
MAP(d_in=>ToReg,d_out=>out_ENC,clk=>clk);
      KeySch13:IMC128_beh port map( d_in=>out_ENC,     d_out=>IMCout );
      KeySch15:Last_round_mux port map(inlast=>out_ENC, inbe4=>IMCout,
outmux=>out_DEC, sel=>round);
      KeySch14:mux2_128 port map(   in0=>out_DEC ,in1=>out_ENC,
outmux=>keyout, sel=>mode );
      KeyMux_out:key_mux_it   port map(   sel=>round,
      original_key=>cipher_key,
                        pre_roundkey=>keyout, key_muxed=>KEY_out);

END struct;
```

- Count 3

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY count3 IS
      port( clk    :in std_logic;
            clk_out :out std_logic
            );
END count3 ;

ARCHITECTURE behave OF count3 IS
      signal count :std_logic_vector(1 downto 0):="00";
BEGIN
      process(clk)
      begin
            if clk'event and clk = '1'then
               if count = "10" then
                  count <= "00";
               else
                  count <= count + 1;
               end if;
            end if;
      end process;
      clk_out <= count(1);
END behave;
```

LEVEL 2

- S-BOX 128 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY S_BOX128bit_beh2 IS
      port( din  : in std_logic_vector(0 to 127);
            reset: in std_logic;
            clk  : in std_logic;
            en_de: in std_logic; --'1'=>encrypt , '0' => decrypt
            dout : out std_logic_vector(0 to 127)
          );

END S_BOX128bit_beh2 ;

ARCHITECTURE behave OF S_BOX128bit_beh2 IS
      COMPONENT S_BOX_beh2
      PORT (
```

```
            din: in std_logic_vector(7 downto 0);
            reset: in std_logic;
            clk: in std_logic;
            en_de: in std_logic; --'1'=>encrypt , '0' => decrypt
            dout: out std_logic_vector(7 downto 0)
      );
      END COMPONENT ;

BEGIN
      Sbox0:S_BOX_beh2 PORT MAP(din=>din(0 to 7),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(0 to 7));
      Sbox1:S_BOX_beh2 PORT MAP(din=>din(8 to 15),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(8 to 15));
      Sbox2:S_BOX_beh2 PORT MAP(din=>din(16 to 23),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(16 to 23));
      Sbox3:S_BOX_beh2 PORT MAP(din=>din(24 to 31),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(24 to 31));
      Sbox4:S_BOX_beh2 PORT MAP(din=>din(32 to 39),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(32 to 39));
      Sbox5:S_BOX_beh2 PORT MAP(din=>din(40 to 47),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(40 to 47));
      Sbox6:S_BOX_beh2 PORT MAP(din=>din(48 to 55),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(48 to 55));
      Sbox7:S_BOX_beh2 PORT MAP(din=>din(56 to 63),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(56 to 63));
      Sbox8:S_BOX_beh2 PORT MAP(din=>din(64 to 71),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(64 to 71));
      Sbox9:S_BOX_beh2 PORT MAP(din=>din(72 to 79),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(72 to 79));
      Sbox10:S_BOX_beh2 PORT MAP(din=>din(80 to 87),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(80 to 87));
      Sbox11:S_BOX_beh2 PORT MAP(din=>din(88 to 95),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(88 to 95));
      Sbox12:S_BOX_beh2 PORT MAP(din=>din(96 to 103),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(96 to 103));
      Sbox13:S_BOX_beh2 PORT MAP(din=>din(104 to 111),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(104 to 111));
      Sbox14:S_BOX_beh2 PORT MAP(din=>din(112 to 119),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(112 to 119));
      Sbox15:S_BOX_beh2 PORT MAP(din=>din(120 to 127),
reset=>reset,clk=>clk,en_de=>en_de,dout=>dout(120 to 127));


END behave;
```

- ShiftRow

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
```

```
ENTITY shiftrow IS
      PORT (      state_in: in std_logic_vector(0 to 127);
            d_out_enc: out std_logic_vector(0 to 127);
            d_out_dec: out std_logic_vector(0 to 127) );

END shiftrow;
ARCHITECTURE A OF shiftrow IS

SIGNAL S0,S1,S2,S3,S4,S5,S6,S7: std_logic_vector( 0 TO 7 ) ;
SIGNAL S8,S9,S10,S11,S12,S13,S14,S15  : std_logic_vector( 0 To 7 )   ;
SIGNAL T0,T1,T2,T3,TI0,TI1,TI2,TI3: std_logic_vector( 0 TO 31 )    ;

BEGIN

                  S0<=state_in(0 TO 7);
                  S1<=state_in(8 TO 15);
                  S2<=state_in(16 TO 23);
                  S3<=state_in(24 TO 31);
                  S4<=state_in(32 TO 39);
                  S5<=state_in(40 TO 47);
                  S6<=state_in(48 TO 55);
                  S7<=state_in(56 TO 63);
                  S8<=state_in(64 TO 71);
                  S9<=state_in(72 TO 79);
                  S10<=state_in(80 TO 87);
                  S11<=state_in(88 TO 95);
                  S12<=state_in(96 TO 103);
                  S13<=state_in(104 TO 111);
                  S14<=state_in(112 TO 119);
                  S15<=state_in(120 TO 127);
      T0 <= (S0&S5&S10&S15);
      T1 <= (S4&S9&S14&S3);
      T2 <= (S8&S13&S2&S7);
      T3 <= (S12&S1&S6&S11);
      TI0 <= (S0&S13&S10&S7);
      TI1 <= (S4&S1&S14&S11);
      TI2 <= (S8&S5&S2&S15);
      TI3 <= (S12&S9&S6&S3);
            d_out_enc <= T0 & T1 & T2 &T3;
            d_out_dec <= TI0 & TI1 & TI2 & TI3;

END A;
```

- MixColumn

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY MixColumn_beh IS

      PORT (
```

```
            d_in_enc: in std_logic_vector(0 to 127);
            d_in_dec: in std_logic_vector(0 to 127);
            d_out: out std_logic_vector(0 to 127);
            en_de: in std_logic     );

END MixColumn_beh;
ARCHITECTURE behave OF MixColumn_beh IS
      COMPONENT IMCtranform_beh
      PORT (
            d_in: in std_logic_vector(0 to 31);
            d_out: out std_logic_vector(0 to 31)      );

      END COMPONENT;
      COMPONENT MCtransform_beh
      PORT (
            d_in: in std_logic_vector(0 to 31);
            d_out: out std_logic_vector(0 to 31)      );
      END COMPONENT ;
      COMPONENT switch_out
      PORT (
            enc : in std_logic_vector(0 to 31);
            dec : in std_logic_vector(0 to 31);
            outp : out std_logic_vector(0 to 31);
            sel : in std_logic      );
      END COMPONENT;


      SIGNAL a,b,c,d,ia,ib,ic,id: std_logic_vector(0 to 31)   ;

BEGIN

MC4: MCtransform_beh port map (d_in =>d_in_enc(0 to 31), d_out =>a);
MC5: IMCtranform_beh port map (d_in =>d_in_dec(0 to 31),d_out =>ia);
MC6: switch_out port map (enc=>a, dec=>ia, outp=>d_out(0 to 31),
sel=>en_de);

MC11: MCtransform_beh port map (d_in =>d_in_enc(32 to 63), d_out =>b);
MC12: IMCtranform_beh port map (d_in =>d_in_dec(32 to 63),d_out =>ib);
MC13: switch_out port map (enc=>b, dec=>ib, outp=>d_out(32 to 63),
                           sel=>en_de);

MC18: MCtransform_beh port map (d_in =>d_in_enc(64 to 95), d_out =>c);
MC19: IMCtranform_beh port map (d_in =>d_in_dec(64 to 95),d_out =>ic);
MC20: switch_out port map (enc=>c, dec=>ic ,outp=>d_out(64 to 95),
                           sel=>en_de);

MC25: MCtransform_beh port map (d_in =>d_in_enc(96 to 127), d_out =>d);
MC26: IMCtranform_beh port map (d_in =>d_in_dec(96 to 127),d_out =>id);
MC27: switch_out port map (enc=>d, dec=>id, outp=>d_out(96 to 127),
                           sel=>en_de);


END behave;
```

- AddRoundKey

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY AddRoundkey IS
PORT( RoundKey,state     : in std_logic_vector(0 to 127) ;
      state_out          : out std_logic_vector (0 to 127) );
END AddRoundkey ;

ARCHITECTURE A OF AddRoundkey IS
BEGIN
      state_out <= (state xor RoundKey);

END A;
```

- Reg 128

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg128_it IS
      PORT (
            sw_on_off: in std_logic;
            d_in: in std_logic_vector(0 to 127);
            d_out: out std_logic_vector(0 to 127);
            clk: in std_logic
      );

END reg128_it;
ARCHITECTURE behave OF reg128_it IS

BEGIN

       PROCESS ( sw_on_off , clk  )

      BEGIN
            if clk'event and clk ='0' then
                  if sw_on_off = '1' then
                        d_out<= d_in;
                  end if;
            end if;
      END PROCESS;

END behave;
```

- Counter 2 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY counter2bit IS
      port( clk,reset    :in std_logic;
            count4..       :out std_logic
            );
END counter2bit ;

ARCHITECTURE behave OF counter2bit IS
      SIGNAL count: std_logic_vector(1 downto 0);
BEGIN
      process_count: PROCESS (clk,reset)
      BEGIN
      IF (reset = '0') THEN
            count <= (others => '1');
      ELSIF (clk'event AND clk = '0') THEN
            count <= count + '1';
      END IF;
      END PROCESS;
      count4 <=count(1);
END behave;
```

- MUX input

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2_inputcore_it is
port( init ,innext : in std_logic_vector(0 to 127);
      outmux : out std_logic_vector(0 to 127);
      sel : in std_logic_vector(3 downto 0));
end mux2_inputcore_it;

architecture behave of mux2_inputcore_it is
begin
outmux <= init when sel = "0000" else
        innext ;
end behave;
```

- MUX 128

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2_128 is
port( in0 ,in1 : in std_logic_vector(0 to 127);
      outmux : out std_logic_vector(0 to 127);
      sel : in std_logic);
```

```
end mux2_128;

architecture A of mux2_128 is
begin
outmux <= in0 when sel = '0' else
        in1 ;
end;
```

- Last Round MUX

```
library ieee;
use ieee.std_logic_1164.all;

entity Last_round_mux is
port( inlast ,inbe4 : in std_logic_vector(0 to 127);
      outmux : out std_logic_vector(0 to 127);
      sel : in std_logic_vector(3 downto 0));
end Last_round_mux;

architecture behave of Last_round_mux is
begin
outmux <= inlast when sel = "1010" else
          inbe4 ;
end behave;
```

- Request Signal Control

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY request_sig_ctrl IS
      port( shift:in std_logic;
            sw_on_off:in std_logic;
            request:out std_logic
            );
END request_sig_ctrl ;

ARCHITECTURE behave OF request_sig_ctrl IS
      signal temp :std_logic;
BEGIN

 temp <= shift after 10 ns  when sw_on_off='1' else  '1';
 request <= temp and sw_on_off;

END behave;
```

- Reg 8 KR

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg8_it_KR IS

      PORT (
            sw_on_off: in std_logic;
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0);
            clk: in std_logic
      );
END reg8_it_KR;
ARCHITECTURE behave OF reg8_it_KR IS
BEGIN
       PROCESS ( sw_on_off ,  clk  )

      BEGIN
            if sw_on_off='1'then
                if (clk'event and clk ='0')then
                    d_out<= d_in;
                end if;
            end if;
      END PROCESS;
END behave;
```

- Key Reg 128

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY keyreg128_it IS
      PORT (
            d_in: in std_logic_vector(0 to 127);
            d_out: out std_logic_vector(0 to 127);
            clk: in std_logic
      );

END keyreg128_it;
ARCHITECTURE behave OF keyreg128_it IS
BEGIN

       PROCESS ( clk  )

      BEGIN
            if clk'event and clk ='0' then
                d_out<= d_in;
            end if;
      END PROCESS;

END behave;
```

- XORinKEYsch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY XORinKEYsch_it IS
      port(
      word0 : in std_logic_vector(0 to 31);
      word1 : in std_logic_vector(0 to 31);
      word2 : in std_logic_vector(0 to 31);
      word3 : in std_logic_vector(0 to 31);
      i_p2 : in std_logic_vector(0 to 31);
      o_p0 : out std_logic_vector(0 to 31);
      o_p1 : out std_logic_vector(0 to 31);
      o_p2 : out std_logic_vector(0 to 31);
      o_p3 : out std_logic_vector(0 to 31));

END XORinKEYsch_it ;

ARCHITECTURE A OF XORinKEYsch_it IS
      COMPONENT XOR32
      PORT (
            in0: in std_logic_vector(0 to 31);
            y: out std_logic_vector(0 to 31);
            in1: in std_logic_vector(0 to 31)
      );
      END COMPONENT ;
      SIGNAL o1,o2,o3,o4: std_logic_vector(0 to 31)   ;

BEGIN
      add1:XOR32 port map (in0=>word0, in1=>i_p2, y=> o1);
      add2:XOR32 port map (in0=>word1, in1=>o1, y=>o2);
      add3:XOR32 port map (in0=>word2, in1=>o2, y=>o3);
      add4:XOR32 port map (in0=>word3,in1=>o3, y=>o4);
      o_p0 <= o1 ;
      o_p1 <= o2 ;
      o_p2 <= o3 ;
      o_p3 <= o4 ;

END A;
```

- MUX 32

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2_32 is
port( in0 ,in1 : in std_logic_vector(0 to 31);
      outmux : out std_logic_vector(0 to 31);
      sel : in std_logic);
end mux2_32;

architecture A of mux2_32 is
```

```
begin
      outmux <= in0 when sel = '0' else
                in1 ;
end;
```

- S-BOX Key

```
Library ieee;
USE IEEE.std_logic_1164.all;

ENTITY S_BOX_Key_beh_it IS
      PORT (
            din: in std_logic_vector(0 to 31);
            reset: in std_logic;
            mode:in std_logic;
            clk: in std_logic;
            round:in std_logic_vector(3 downto 0);
            dout: out std_logic_vector(0 to 31)
      );

END S_BOX_Key_beh_it;
ARCHITECTURE struct OF S_BOX_Key_beh_it IS

      COMPONENT mapping_beh
      PORT ( x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)                    );
      END COMPONENT ;
      COMPONENT GF16_inversion_beh_it

      PORT (      reset,clk:in std_logic;
            din: in std_logic_vector(7 downto 0);
            dout: out std_logic_vector(7 downto 0)
      );
      END COMPONENT ;
      COMPONENT inv_mapping_beh
      PORT (
            x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)       );
      END COMPONENT ;
      COMPONENT affine_beh
      PORT ( x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)                    );
      END COMPONENT ;
      COMPONENT reg8_it
      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0);
            clk: in std_logic
      );
      END COMPONENT ;
      COMPONENT Rcon_beh
            PORT ( round : in std_logic_vector(3 downto 0);
```

```
                mode :in std_logic; --'1'=Ncrypt, '0'=Dcrypt
                rcon : out std_logic_vector(7 downto 0)
                );
    END COMPONENT ;
    COMPONENT XOR2_8
    PORT (in0 : in std_logic_vector(7 downto 0);
          in1 : in std_logic_vector(7 downto 0);
          outp : out std_logic_vector(7 downto 0));
    END COMPONENT ;

    SIGNAL
map0,map1,map2,map3,inv0,inv1,inv2,inv3,sync0,sync1,sync2,sync3:
std_logic_vector( 7 DOWNTO 0 )   ;
    SIGNAL
imap0,imap1,imap2,imap3,rcon,AF0out,AF1out,AF2out,AF3out,Addout:
std_logic_vector( 7 DOWNTO 0 )   ;

BEGIN
    Rc:Rcon_beh PORT MAP(round=>round, mode=>mode, rcon=>rcon);

    SB0: mapping_beh  PORT MAP( x=>din(0 to 7), y=>map0);
    PLine1:reg8_it PORT MAP(
    reset=>reset,clk=>clk,d_in=>map0,d_out=>sync0);
    SB1: GF16_inversion_beh_it    PORT
MAP(reset=>reset,clk=>clk,din=>sync0 ,dout=>inv0);
    SB2: inv_mapping_beh    PORT MAP(x=>inv0, y=>imap0);
    SB3: affine_beh   PORT MAP ( x=>imap0,    y=>AF0out);
    addRcon:XOR2_8 PORT MAP(in0=>rcon, in1=>AF0out,outp=>Addout);
    PLine5:reg8_it PORT
MAP(reset=>reset,clk=>clk,d_in=>Addout,d_out=>dout(0 to 7));


    SB4: mapping_beh  PORT MAP( x=>din(8 to 15), y=>map1);
    PLine2:reg8_it PORT MAP(
    reset=>reset,clk=>clk,d_in=>map1,d_out=>sync1);
    SB5: GF16_inversion_beh_it    PORT
MAP(reset=>reset,clk=>clk,din=>sync1 ,dout=>inv1);
    SB6: inv_mapping_beh    PORT MAP(x=>inv1, y=>imap1);
    SB7: affine_beh   PORT MAP ( x=>imap1,    y=>AF1out);
    PLine6:reg8_it PORT
MAP(reset=>reset,clk=>clk,d_in=>AF1out,d_out=>dout(8 to 15));

    SB8: mapping_beh  PORT MAP( x=>din(16 to 23), y=>map2);
    PLine3:reg8_it PORT MAP(
    reset=>reset,clk=>clk,d_in=>map2,d_out=>sync2);
    SB9: GF16_inversion_beh_it    PORT
MAP(reset=>reset,clk=>clk,din=>sync2 ,dout=>inv2);
    SB10: inv_mapping_beh   PORT MAP(x=>inv2, y=>imap2);
    SB11: affine_beh  PORT MAP ( x=>imap2,    y=>AF2out);
    PLine7:reg8_it PORT
MAP(reset=>reset,clk=>clk,d_in=>AF2out,d_out=>dout(16 to 23));

    SB12: mapping_beh PORT MAP( x=>din(24 to 31), y=>map3);
```

```
      PLine4:reg8_it PORT MAP(
      reset=>reset,clk=>clk,d_in=>map3,d_out=>sync3);
      SB13: GF16_inversion_beh_it   PORT
MAP(reset=>reset,clk=>clk,din=>sync3 ,dout=>inv3);
      SB14: inv_mapping_beh   PORT MAP(x=>inv3, y=>imap3);
      SB15: affine_beh  PORT MAP ( x=>imap3,    y=>AF3out);
      PLine8:reg8_it PORT
MAP(reset=>reset,clk=>clk,d_in=>AF3out,d_out=>dout(24 to 31));

END struct;
```

- XOR32

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY XOR32 IS
      PORT (
            in0: in std_logic_vector(0 to 31);
            y: out std_logic_vector(0 to 31);
            in1: in std_logic_vector(0 to 31)
      );
END XOR32;

ARCHITECTURE A OF XOR32 IS
BEGIN
      y<= in0 xor in1;
END A;
```

- Reg32

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg32_it IS
      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(0 to 31);
            d_out: out std_logic_vector(0 to 31);
            clk: in std_logic
      );
END reg32_it;
ARCHITECTURE behave OF reg32_it IS
BEGIN

       PROCESS ( reset , clk  )

      BEGIN
            if reset = '0' then
               d_out <= (others=>'Z');

            elsif clk'event and clk ='0' then
```

```
                d_out<= d_in;
            end if;
      END PROCESS;
END behave;
```

- Key MUX

```
Library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;

entity key_mux_it is
port (      sel: in std_logic_vector(3 downto 0)  ;--as 'round' signal
      original_key, pre_roundkey : in std_logic_vector (0 to 127) ;
      key_muxed : out  std_logic_vector (0 to 127)) ;

end key_mux_it ;

Architecture behave of key_mux_it is

begin

      Key_muxed<=original_key when sel = "0000" else  pre_roundkey;
end  behave;
```

- IMC 128

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY IMC128_beh IS
      port (
            d_in : in std_logic_vector(0 to 127);
            d_out : out std_logic_vector(0 to 127)
            );
END IMC128_beh ;

ARCHITECTURE struct OF IMC128_beh IS
      COMPONENT IMCtranform_beh
      port (
            d_in : in std_logic_vector(0 to 31);
            d_out : out std_logic_vector(0 to 31)
            );
      END COMPONENT ;

BEGIN
C0:IMCtranform_beh port map(d_in=>d_in(0 to 31),d_out=>d_out(0 to 31));
C1:IMCtranform_beh port map(d_in=>d_in(32 to 63),d_out=>d_out(32 to
63));
C2:IMCtranform_beh port map(d_in=>d_in(64 to 95),d_out=>d_out(64 to
95));
```

```
C3:IMCtranform_beh port map (d_in=>d_in(96 to 127),d_out=>d_out(96 to
127));
END struct;
```

- Shift Reg 128

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY shift_reg128_it IS
      port(
            clk,sw_on_off,d_in:     in std_logic;
              shift_out: out std_logic_vector(0 to 127)
              );
END shift_reg128_it ;

ARCHITECTURE behave OF shift_reg128_it IS
      signal shift_reg:std_logic_vector(0 to 127);
BEGIN
      process (clk,sw_on_off)
       begin
           if sw_on_off='0' then
            shift_reg<=(others=>'Z');
           elsif clk'event and clk='0' then
              shift_reg<=shl(shift_reg,"1");
              shift_reg(127)<=d_in;
           end if;
        end process;
        shift_out<=shift_reg;
END behave;
```

- Counter7bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY counter7bit_it IS
      port(
            clk:in std_logic;
            reset:in std_logic;
            count:out std_logic_vector(6 downto 0)
      );
END counter7bit_it ;

ARCHITECTURE behave OF counter7bit_it IS
      signal temp:std_logic_vector(6 downto 0);
BEGIN
      process_count: PROCESS (clk,reset)
      BEGIN
```

```
            IF (reset = '0') THEN
                  temp <= (others => '0');
            ELSIF (clk'event AND clk = '0') THEN
                  temp <= temp + '1';
            END IF;
      END PROCESS;

      count<=temp;
END behave;
```

- Parallel load shift reg128

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY Pall_load_shift_reg128 IS
      Port( clk,reset,load:in std_logic;
            d_in  :in std_logic_vector(0 to 127);
            d_out :out std_logic
             );
END Pall_load_shift_reg128 ;

ARCHITECTURE behave OF Pall_load_shift_reg128 IS
      signal reg :std_logic_vector(0 to 127);
BEGIN
       PROCESS (clk,reset)
      BEGIN
            if reset='0' then
               reg<=(others=>'0');
            elsif clk'event and clk ='0'then
               if load = '1' then
                  reg <= d_in;
               else
                  reg<=shl(reg,"1");
                  reg(127)<='0';
                end if;
            end if;
      END PROCESS;
      d_out<=reg(0);
END behave;
```

- Tri state buffer 2 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY Tri_state_buffer2bit IS
      port(
            enable :in std_logic;
            input  :in std_logic_vector(1 downto 0);
            output :out std_logic_vector(1 downto 0));
END Tri_state_buffer2bit ;
```

```
ARCHITECTURE behave OF Tri_state_buffer2bit IS

BEGIN
      output <= input when enable = '1' else  (others=>'Z');
END behave;
```

----------LEVEL 3----------

- S-BOX

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY S_BOX_beh2 IS
      PORT (
            din: in std_logic_vector(7 downto 0);
            reset: in std_logic;
            clk: in std_logic;
            en_de: in std_logic; --'1'=>encrypt , '0' => decrypt
            dout: out std_logic_vector(7 downto 0)
      );

END S_BOX_beh2;
ARCHITECTURE behave OF S_BOX_beh2 IS
      COMPONENT inv_affine_beh
      PORT ( x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)                    );
      END COMPONENT ;
      COMPONENT mux2_8
      port( in0 ,in1 : in std_logic_vector(7 downto 0);
      outmux : out std_logic_vector(7 downto 0);
      sel : in std_logic);
      END COMPONENT ;
      COMPONENT reg8
      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0);
            clk: in std_logic
            );
      END COMPONENT ;

      COMPONENT mapping_beh
      PORT ( x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)                    );
      END COMPONENT ;

      COMPONENT GF16_inversion_beh_it
      PORT (      reset,clk:in std_logic;
            din: in std_logic_vector(7 downto 0);
            dout: out std_logic_vector(7 downto 0)
```

```
      );
      END COMPONENT ;

      COMPONENT inv_mapping_beh
      PORT (
            x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)      );
      END COMPONENT ;
      COMPONENT affine_beh
      PORT ( x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)                  );
      END COMPONENT ;

      SIGNAL  out0,out1,out2,out3,out4,out5,out6,out7:
std_logic_vector( 7 DOWNTO 0 )   ;

BEGIN
      SB0: inv_affine_beh PORT MAP ( x=>din,    y=>out0);
      SB1: mux2_8 port
map(in0=>out0 ,in1=>din ,outmux=>out1 ,sel=>en_de);
      SB3: mapping_beh  PORT MAP( x=>out1, y=>out2);
      SB2: reg8 PORT MAP(reset=>reset,d_in=>out2,d_out=>out3,
      clk=>clk);
      SB4: GF16_inversion_beh_it    PORT MAP(din=>out3 ,dout=>out4,
clk=>clk ,reset=>reset);
      SB5: inv_mapping_beh    PORT MAP(x=>out4, y=>out5);
      SB7: affine_beh   PORT MAP ( x=>out5,     y=>out6);
      SB8: mux2_8 port
map(in0=>out5 ,in1=>out6 ,outmux=>out7 ,sel=>en_de);
      SB9: reg8 PORT MAP(reset=>reset,d_in=>out7,d_out=>dout,
      clk=>clk);
END behave;
```

- Mapping

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mapping_beh IS

      PORT ( x : in std_logic_vector(7 downto 0);
            y : out std_logic_vector(7 downto 0)                  );

END mapping_beh ;

ARCHITECTURE behave OF mapping_beh IS

BEGIN
      y(0)<= (x(0)xor x(2)xor x(3)xor x(4)xor x(6)xor x(7));
      y(1)<= (x(1)xor x(3));
```

```
      y(2)<= (x(1)xor x(4)xor x(6));
      y(3)<= (x(1)xor x(2)xor x(6)xor x(7));
      y(4)<= (x(4)xor x(5)xor x(6));
      y(5)<= (x(1)xor x(4)xor x(6)xor x(7));
      y(6)<= (x(2)xor x(3)xor x(5)xor x(7));
      y(7)<= (x(5)xor x(7));
END behave;
```

- GF16 inversion

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY GF16_inversion_beh_it IS

      PORT (      reset,clk:in std_logic;
            din: in std_logic_vector(7 downto 0);
            dout: out std_logic_vector(7 downto 0)
      );

END GF16_inversion_beh_it;
ARCHITECTURE A OF GF16_inversion_beh_it IS
      COMPONENT multipier_beh
      PORT (
            c: out std_logic_vector(3 downto 0);
            b: in std_logic_vector(3 downto 0);
            a: in std_logic_vector(3 downto 0)  );
      END COMPONENT ;
      COMPONENT square_and_mul9_beh
      PORT (
            x : IN std_logic_vector( 3 DOWNTO 0 ) ;
            y : OUT std_logic_vector( 3 DOWNTO 0 )     );
      END COMPONENT ;
      COMPONENT squaring_beh
      PORT (
            x : IN std_logic_vector( 3 DOWNTO 0 ) ;
            y : OUT std_logic_vector( 3 DOWNTO 0 )     );
      END COMPONENT ;
      COMPONENT INV_beh
      PORT (
            y: out std_logic_vector(3 downto 0);
            x: in std_logic_vector(3 downto 0)        );
      END COMPONENT ;
      COMPONENT XOR3_4
      PORT (in0 : in std_logic_vector(3 downto 0);
            in1 : in std_logic_vector(3 downto 0);
            in2 : in std_logic_vector(3 downto 0);
            outp : out std_logic_vector(3 downto 0));
      END COMPONENT ;
      COMPONENT XOR2_4
      PORT (in0 : in std_logic_vector(3 downto 0);
            in1 : in std_logic_vector(3 downto 0);
            outp : out std_logic_vector(3 downto 0));
```

```
	END COMPONENT ;
	COMPONENT reg4
	PORT (
		reset: in std_logic;
		d_in: in std_logic_vector(3 downto 0);
		d_out: out std_logic_vector(3 downto 0);
		clk: in std_logic
	);
	END COMPONENT ;

SIGNAL temp1,temp2,temp3,b,c,p,q : std_logic_vector( 3 DOWNTO 0 );
SIGNAL xor0,xor1,xor2,out3,out4,out5 : std_logic_vector( 3 DOWNTO 0 );

BEGIN
	b<= din(7 downto 4);
	c<= din(3 downto 0);
GF16inv0: multipier_beh   port map (c=>xor0,    b=>c, a=>b);
GF16inv2: square_and_mul9_beh port map (x=>b, y=>xor1);
GF16inv5: squaring_beh    port map (x=>c, y=>xor2);
GF16inv8: XOR3_4 port map (in0=>xor0, in1=>xor1, in2=>xor2, outp=>out3);
GF16inv11: INV_beh        port map  (x=>out3,y=>out5  );
GF16inv9: XOR2_4          port map (in0=>b, in1=>c, outp=>out4);
GF16inv10:reg4 PORT MAP(reset=>reset,clk=>clk,d_in=>out5,d_out=>temp1);
GF16inv12:reg4 PORT MAP(reset=>reset,clk=>clk,d_in=>out4,d_out=>temp2);
GF16inv13:reg4 PORT MAP(reset=>reset,clk=>clk,  d_in=>b,d_out=>temp3);
GF16inv14: multipier_beh  port map (c=>p,b=>temp3,a=>temp1);
GF16inv15: multipier_beh   port map (c=>q,b=>temp2,a=>temp1);
	dout(7 downto 4)<=p;
	dout(3 downto 0)<=q;
END A;
```

- Inverse Mapping

```
Library ieee;
use ieee.std_logic_1164.all;

ENTITY inv_mapping_beh IS
	PORT (
		x : in std_logic_vector(7 downto 0);
		y : out std_logic_vector(7 downto 0)
	);
END inv_mapping_beh ;

ARCHITECTURE behave OF inv_mapping_beh IS

BEGIN
	y(0)<=(x(0)xor x(4)xor x(6));
	y(1)<=(x(4)xor x(5)xor x(7));
	y(2)<=(x(1)xor x(4)xor x(5)xor x(6));
	y(3)<=(x(1)xor x(4)xor x(5)xor x(7));
	y(4)<=(x(1)xor x(3)xor x(4)xor x(6));
	y(5)<=(x(2)xor x(5)xor x(7));
```

```
      y(6)<=(x(1)xor x(2)xor x(3)xor x(5)xor x(6)xor x(7));
      y(7)<=(x(2)xor x(5));
END behave;
```

- Affine Transform

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY affine_beh IS
      port(
            x :in std_logic_vector(7 downto 0);
            y :out std_logic_vector(7 downto 0)
            );
END affine_beh ;

ARCHITECTURE behave OF affine_beh IS

BEGIN
      y(0)<=not(x(0)xor x(4)xor x(5)xor x(6)xor x(7));
      y(1)<=not(x(0)xor x(1)xor x(5)xor x(6)xor x(7));
      y(2)<=(x(0)xor x(1)xor x(2)xor x(6)xor x(7));
      y(3)<=(x(0)xor x(1)xor x(2)xor x(3)xor x(7));
      y(4)<=(x(0)xor x(1)xor x(2)xor x(3)xor x(4));
      y(5)<=not(x(1)xor x(2)xor x(3)xor x(4)xor x(5));
      y(6)<=not(x(2)xor x(3)xor x(4)xor x(5)xor x(6));
      y(7)<=(x(3)xor x(4)xor x(5)xor x(6)xor x(7));
END behave;
```

- Reg 8

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg8_it IS
      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0);
            clk: in std_logic
      );
END reg8_it;
ARCHITECTURE behave OF reg8_it IS

BEGIN
       PROCESS ( reset , d_in , clk  )
      BEGIN
            if reset = '0' then
               d_out <= (others=>'Z');
            elsif clk'event and clk ='0' then
                d_out<= d_in;
            end if;
```

```
      END PROCESS;
END behave;
```

- Rcon

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY Rcon_beh IS
            PORT ( round : in std_logic_vector(3 downto 0);
                   mode :in std_logic; --'1'=Ncrypt, '0'=Dcrypt
                   rcon : out std_logic_vector(7 downto 0)
                   );
END Rcon_beh ;

ARCHITECTURE behave OF Rcon_beh IS

BEGIN
      process(round,mode)
      variable temp:integer ;
      variable t_rcon: std_logic_vector(7 downto 0);
      begin
            temp:=conv_integer(round);
            if mode = '1' then
              case temp is
                when 0 =>  t_rcon := "00000000";
                when 1 =>  t_rcon := "00000001";
                when 2 =>  t_rcon := "00000010";
                when 3 =>  t_rcon := "00000100";
                when 4 =>  t_rcon := "00001000";
                when 5 =>  t_rcon := "00010000";
                when 6 =>  t_rcon := "00100000";
                when 7 =>  t_rcon := "01000000";
                when 8 =>  t_rcon := "10000000";
                when 9 =>  t_rcon := "00011011";
                when 10 => t_rcon := "00110110";
                WHEN OTHERS => NULL;
              end case;

            else
               case temp is
                when 0 =>  t_rcon := "00000000";
                when 10 => t_rcon := "00000001";
                when 9 =>  t_rcon := "00000010";
                when 8 =>  t_rcon := "00000100";
                when 7 =>  t_rcon := "00001000";
                when 6 =>  t_rcon := "00010000";
                when 5 =>  t_rcon := "00100000";
                when 4 =>  t_rcon := "01000000";
                when 3 =>  t_rcon := "10000000";
                when 2 =>  t_rcon := "00011011";
                when 1 =>  t_rcon := "00110110";
```

```
            WHEN OTHERS => NULL;
          end case;

        end if;
        rcon<=t_rcon;
    end process;
END behave;
```

- XOR 2 input 8 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY XOR2_8 IS
    PORT (in0 : in std_logic_vector(7 downto 0);
          in1 : in std_logic_vector(7 downto 0);
          outp : out std_logic_vector(7 downto 0));
END XOR2_8 ;

ARCHITECTURE A OF XOR2_8 IS

BEGIN
    outp <= in0 xor in1;
END;
```

- InvMixColumn Transform

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY IMCtranform_beh IS
    PORT (
        d_in: in std_logic_vector(0 to 31);
        d_out: out std_logic_vector(0 to 31)
    );

END IMCtranform_beh;
ARCHITECTURE behave OF IMCtranform_beh IS
    COMPONENT mul09_beh
    PORT (
        d_in: in std_logic_vector(7 downto 0);
        d_out: out std_logic_vector(7 downto 0)   );
    END COMPONENT ;
    COMPONENT mul0B_beh
    PORT (
        d_in: in std_logic_vector(7 downto 0);
        d_out: out std_logic_vector(7 downto 0)   );
    END COMPONENT ;
    COMPONENT mul0D_beh
    PORT (
        d_in: in std_logic_vector(7 downto 0);
        d_out: out std_logic_vector(7 downto 0)   );
```

```
	END COMPONENT ;
	COMPONENT mul0E_beh
	PORT (
		d_in: in std_logic_vector(7 downto 0);
		d_out: out std_logic_vector(7 downto 0)	);
	END COMPONENT ;
	COMPONENT XOR4_8
	PORT (
		in0: in std_logic_vector(7 downto 0);
		outp: out std_logic_vector(7 downto 0);
		in2: in std_logic_vector(7 downto 0);
		in1: in std_logic_vector(7 downto 0);
		in3: in std_logic_vector(7 downto 0)	);
	END COMPONENT;
	SIGNAL a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p : std_logic_vector(7
downto 0)	;
BEGIN
	IMCt0: mul09_beh port map (d_in => d_in(24 to 31),d_out => a);
	IMCt1: mul0B_beh port map (d_in => d_in(8 to 15),d_out => c);
	IMCt2: mul0D_beh port map (d_in => d_in(16 to 23),d_out => b);
	IMCt3: mul0E_beh port map (d_in => d_in(0 to 7),d_out => d);
	IMCt4: XOR4_8 port map (in0 => a,in1 => b,in2 =>c,in3 => d,outp
=> d_out(0 to 7) );

	IMCt5: mul09_beh port map (d_in => d_in(0 to 7),d_out => e);
	IMCt6: mul0B_beh port map (d_in => d_in(16 to 23),d_out => f);
	IMCt7: mul0D_beh port map (d_in => d_in(24 to 31),d_out => g);
	IMCt8: mul0E_beh port map (d_in => d_in(8 to 15),d_out => h);
	IMCt9: XOR4_8 port map (in0 => e,in1 => f,in2 =>g,in3 =>h,outp =>
d_out(8 to 15) );

	IMCt10: mul09_beh port map (d_in => d_in(8 to 15),d_out => i);
	IMCt11: mul0B_beh port map (d_in => d_in(24 to 31),d_out => j);
	IMCt12: mul0D_beh port map (d_in => d_in(0 to 7),d_out => k);
	IMCt13: mul0E_beh port map (d_in => d_in(16 to 23),d_out => l);
	IMCt14: XOR4_8 port map (in0 => i,in1 => j,in2 =>k,in3 => l,outp
=> d_out(16 to 23) );

	IMCt15: mul09_beh port map (d_in => d_in(16 to 23),d_out => m);
	IMCt16: mul0B_beh port map (d_in => d_in(0 to 7),d_out => n);
	IMCt17: mul0D_beh port map (d_in => d_in(8 to 15),d_out => o);
	IMCt18: mul0E_beh port map (d_in => d_in(24 to 31),d_out => p);
	IMCt19: XOR4_8 port map (in0 => m,in1 => n,in2 =>o,in3 => p,outp
=> d_out(24 to 31) );
END behave;
```

- MixColumn Transform

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY MCtransform_beh IS
```

```
      PORT (
            d_in: in std_logic_vector(0 to 31);
            d_out: out std_logic_vector(0 to 31)
      );

END MCtransform_beh;
ARCHITECTURE behave OF MCtransform_beh IS
      COMPONENT mul2_beh
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)    );
      END COMPONENT ;
      COMPONENT mul3_beh
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)    );
      END COMPONENT ;
      COMPONENT XOR4_8
      PORT (
            in0: in std_logic_vector(7 downto 0);
            outp: out std_logic_vector(7 downto 0);
            in2: in std_logic_vector(7 downto 0);
            in1: in std_logic_vector(7 downto 0);
            in3: in std_logic_vector(7 downto 0)      );
      END COMPONENT;
      SIGNAL a,b,c,d,e,f,g,h : std_logic_vector(7 downto 0) ;

BEGIN
      MCt0: mul2_beh port map (d_in => d_in(0 to 7),d_out => a);
      MCt1: mul3_beh port map (d_in => d_in(8 to 15),d_out => b);
      MCt2: XOR4_8 port map (in0 => a,in1 => b,in2 =>d_in(16 to 23),
                  in3 => d_in(24 to 31),outp => d_out(0 to 7) );

      MCt3: mul2_beh port map (d_in => d_in(8 to 15),d_out => c);
      MCt4: mul3_beh port map (d_in => d_in(16 to 23),d_out => d);
      MCt5: XOR4_8 port map (in0 => c,in1 => d,in2 =>d_in(24 to 31),
                  in3 => d_in(0 to 7),outp => d_out(8 to 15) );

      MCt6: mul2_beh port map (d_in => d_in(16 to 23),d_out => e);
      MCt7: mul3_beh port map (d_in => d_in(24 to 31),d_out => f);
      MCt8: XOR4_8 port map (in0 => e,in1 => f,in2 =>d_in(0 to 7),
                  in3 => d_in(8 to 15),outp => d_out(16 to 23) );

      MCt9: mul2_beh port map (d_in => d_in(24 to 31),d_out => g);
      MCt10: mul3_beh port map (d_in => d_in(0 to 7),d_out => h);
      MCt11: XOR4_8 port map (in0 => g,in1 => h,in2 =>d_in(8 to 15),
                  in3 => d_in(16 to 23),outp => d_out(24 to 31) );
END behave;
```

- Switch Output

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
```

```
ENTITY switch_out IS
      PORT (
            enc : in std_logic_vector(0 to 31);
            dec : in std_logic_vector(0 to 31);
            outp : out std_logic_vector(0 to 31);
            sel : in std_logic
      );
END switch_out ;

ARCHITECTURE A OF switch_out IS
BEGIN
      outp <= enc when sel = '1' else dec ;
END A;
```

---------- LEVEL 4 ----------

-Inverse Affine Transform

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY inv_affine_beh IS
      port(
            x :in std_logic_vector(7 downto 0);
            y :out std_logic_vector(7 downto 0)
      );
END inv_affine_beh ;

ARCHITECTURE behave OF inv_affine_beh IS

BEGIN
      y(0)<= not(x(2)xor x(5)xor x(7));
      y(1)<= (x(0)xor x(3)xor x(6));
      y(2)<= not(x(1)xor x(4)xor x(7));
      y(3)<= (x(0)xor x(2)xor x(5));
      y(4)<= (x(1)xor x(3)xor x(6));
      y(5)<= (x(2)xor x(4)xor x(7));
      y(6)<= (x(0)xor x(3)xor x(5));
      y(7)<= (x(1)xor x(4)xor x(6));
END behave;
```

- MUX 8 bit

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2_8 is
```

```
port( in0 ,in1 : in std_logic_vector(7 downto 0);
      outmux : out std_logic_vector(7 downto 0);
      sel : in std_logic);
end mux2_8;

architecture A of mux2_8 is
begin
      outmux <= in0 when sel = '0' else in1;
end;
```

- Reg 8 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg8 IS
      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0);
            clk: in std_logic
      );
END reg8;

ARCHITECTURE behave OF reg8 IS
BEGIN
       PROCESS ( reset , d_in , clk )
      BEGIN
            if reset = '0' then
               d_out <= (others=>'Z');

            elsif clk'event and clk ='0' then
                d_out<= d_in;
            end if;
      END PROCESS;
END behave;
```

- Inversion in GF16

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY INV_beh IS
      PORT (
            y: out std_logic_vector(3 downto 0);
            x: in std_logic_vector(3 downto 0)
            );
END INV_beh;

ARCHITECTURE behave OF INV_beh IS
```

```
BEGIN
      process(x)
      variable temp : integer range 0 to 15;
      begin
            temp := conv_integer(x);
            case temp is
            when 0 => y<="0000" ;
            when 1 => y<="0001" ;
            when 2 => y<="1001" ;
            when 3 => y<="1110" ;
            when 4 => y<="1101" ;
            when 5 => y<="1011" ;
            when 6 => y<="0111" ;
            when 7 => y<="0110" ;
            when 8 => y<="1111" ;
            when 9 => y<="0010" ;
            when 10 => y<="1100" ;
            when 11 => y<="0101" ;
            when 12 => y<="1010" ;
            when 13 => y<="0100" ;
            when 14 => y<="0011" ;
            when 15 => y<="1000" ;
            end case;
      end process;

END behave;
```

- Multiple in GF16

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY multipier_beh IS
      PORT (
            c: out std_logic_vector(3 downto 0);
            b: in std_logic_vector(3 downto 0);
            a: in std_logic_vector(3 downto 0)
      );
END multipier_beh;

ARCHITECTURE behave OF multipier_beh IS

BEGIN
      c(0)<=((a(0)and b(0))xor(a(3)and b(1))xor(a(2)and
b(2))xor(a(1)and b(3)));
      c(1)<=((a(1)and b(0))xor(a(0)and b(1))xor(a(3)and
b(1))xor(a(3)and b(2))xor(a(2)and b(2))xor(a(2)and b(3))xor(a(1)and
b(3)));
      c(2)<=((a(2)and b(0))xor(a(1)and b(1))xor(a(0)and
b(2))xor(a(3)and b(2))xor(a(3)and b(3))xor(a(2)and b(3)));
      c(3)<=((a(3)and b(0))xor(a(2)and b(1))xor(a(1)and
b(2))xor(a(0)and b(3))xor(a(3)and b(3)));
END behave;
```

- Reg 4 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY reg4 IS

      PORT (
            reset: in std_logic;
            d_in: in std_logic_vector(3 downto 0);
            d_out: out std_logic_vector(3 downto 0);
            clk: in std_logic
      );

END reg4;
ARCHITECTURE behave OF reg4 IS

BEGIN
       PROCESS ( reset , d_in , clk  )
      BEGIN
            if reset = '0' then
              d_out <= (others=>'Z');
            elsif clk'event and clk ='0' then
               d_out<= d_in;
            end if;
      END PROCESS;
END behave;
```

- Square and multply by 9

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY square_and_mul9_beh IS

      PORT (
            x : IN std_logic_vector( 3 DOWNTO 0 ) ;
            y : OUT std_logic_vector( 3 DOWNTO 0 )
      );
END square_and_mul9_beh ;

ARCHITECTURE behave OF square_and_mul9_beh IS
BEGIN
      y(0)<=x(0);
      y(1)<=x(1) xor x(3);
      y(2)<=x(3);
      y(3)<=x(0) xor x(2);
END behave;
```

- Squaring

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY squaring_beh IS
      PORT (
            x : IN std_logic_vector( 3 DOWNTO 0 ) ;
            y : OUT std_logic_vector( 3 DOWNTO 0 )
      );
END squaring_beh ;

ARCHITECTURE behave OF squaring_beh IS

BEGIN
      y(0)<=x(0)xor x(2);
      y(1)<=x(2);
      y(2)<=x(1)xor x(3);
      y(3)<=x(3);
END behave;
```

- XOR 2 input 4 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;


ENTITY XOR2_4 IS
      PORT (in0 : in std_logic_vector(3 downto 0);
            in1 : in std_logic_vector(3 downto 0);
            outp : out std_logic_vector(3 downto 0));
END XOR2_4 ;

ARCHITECTURE A OF XOR2_4 IS
BEGIN
      outp <= in0 xor in1;
END;
```

- XOR 3 input 4 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY XOR3_4 IS
      PORT (in0 : in std_logic_vector(3 downto 0);
            in1 : in std_logic_vector(3 downto 0);
            in2 : in std_logic_vector(3 downto 0);
            outp : out std_logic_vector(3 downto 0));
END XOR3_4 ;
ARCHITECTURE A OF XOR3_4 IS
BEGIN
      outp <= in0 xor in1 xor in2;
END;
```

- Multiply by 02 (x02)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mul2_beh IS

      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)
      );
END mul2_beh;

ARCHITECTURE behave OF mul2_beh IS
BEGIN
      d_out(0) <= d_in(7);
      d_out(1) <= d_in(7) xor d_in(0);
      d_out(2) <= d_in(1);
      d_out(3) <= d_in(7) xor d_in(2);
      d_out(4) <= d_in(7) xor d_in(3);
      d_out(5) <= d_in(4);
      d_out(6) <= d_in(5);
      d_out(7) <= d_in(6);
END behave;
```

- Multiply by 03 (x03)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mul3_beh IS
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)
      );
END mul3_beh;

ARCHITECTURE behave OF mul3_beh IS
BEGIN
      d_out(0) <= d_in(7) xor d_in(0);
      d_out(1) <= d_in(7) xor d_in(1) xor d_in(0);
      d_out(2) <= d_in(1)xor d_in(2);
      d_out(3) <= d_in(7) xor d_in(2)xor d_in(3);
      d_out(4) <= d_in(7) xor d_in(3)xor d_in(4);
      d_out(5) <= d_in(4)xor d_in(5);
      d_out(6) <= d_in(5)xor d_in(6);
      d_out(7) <= d_in(6)xor d_in(7);
END behave;
```

- Multiply by 09 (x09)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mul09_beh IS
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)
      );
END mul09_beh;

ARCHITECTURE behave OF mul09_beh IS
BEGIN
      d_out(0)<=d_in(0) xor d_in(5);
      d_out(1)<=d_in(1) xor d_in(5)xor d_in(6);
      d_out(2)<=d_in(2) xor d_in(6)xor d_in(7);
      d_out(3)<=d_in(0) xor d_in(3)xor d_in(5)xor d_in(7);
      d_out(4)<=d_in(1) xor d_in(4)xor d_in(5)xor d_in(6);
      d_out(5)<=d_in(2) xor d_in(5)xor d_in(6)xor d_in(7);
      d_out(6)<=d_in(6) xor d_in(3)xor d_in(7);
      d_out(7)<=d_in(4) xor d_in(7);

END behave;
```

- Multiply by 0B (x0B)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mul0B_beh IS
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)
      );
END mul0B_beh;

ARCHITECTURE behave OF mul0B_beh IS
BEGIN
      d_out(0)<=d_in(0) xor d_in(5)xor d_in(7);
      d_out(1)<=d_in(0) xor d_in(1)xor d_in(5)xor d_in(6)xor d_in(7);
      d_out(2)<=d_in(1) xor d_in(2)xor d_in(6)xor d_in(7);
      d_out(3)<=d_in(0) xor d_in(2)xor d_in(3)xor d_in(5);
      d_out(4)<=d_in(1) xor d_in(3)xor d_in(4)xor d_in(5)xor d_in(6)xor
d_in(7);
      d_out(5)<=d_in(2) xor d_in(4)xor d_in(5)xor d_in(6)xor d_in(7);
      d_out(6)<=d_in(3) xor d_in(5)xor d_in(6)xor d_in(7);
      d_out(7)<=d_in(4) xor d_in(6)xor d_in(7);
END behave;
```

- Multiply by 0D (x0D)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mul0D_beh IS
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)
      );
END mul0D_beh;

ARCHITECTURE behave OF mul0D_beh IS
BEGIN
      d_out(0)<=d_in(0) xor d_in(5)xor d_in(6);
      d_out(1)<=d_in(1) xor d_in(5)xor d_in(7);
      d_out(2)<=d_in(0) xor d_in(2)xor d_in(6);
      d_out(3)<=d_in(0) xor d_in(1)xor d_in(3)xor d_in(5)xor d_in(6)xor
d_in(7);
      d_out(4)<=d_in(1) xor d_in(2)xor d_in(4)xor d_in(5)xor d_in(7);
      d_out(5)<=d_in(2) xor d_in(3)xor d_in(5)xor d_in(6);
      d_out(6)<=d_in(3) xor d_in(4)xor d_in(6)xor d_in(7);
      d_out(7)<=d_in(4) xor d_in(5)xor d_in(7);
END behave;
```

- Multiply by 0E (x0E)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY mul0E_beh IS
      PORT (
            d_in: in std_logic_vector(7 downto 0);
            d_out: out std_logic_vector(7 downto 0)
      );
END mul0E_beh;

ARCHITECTURE behave OF mul0E_beh IS
BEGIN
      d_out(0)<=d_in(5) xor d_in(6)xor d_in(7);
      d_out(1)<=d_in(0) xor d_in(5);
      d_out(2)<=d_in(0) xor d_in(1)xor d_in(6);
      d_out(3)<=d_in(0) xor d_in(1)xor d_in(2)xor d_in(5)xor d_in(6);
      d_out(4)<=d_in(1) xor d_in(2)xor d_in(3)xor d_in(5);
      d_out(5)<=d_in(2) xor d_in(3)xor d_in(4)xor d_in(6);
      d_out(6)<=d_in(3) xor d_in(4)xor d_in(5)xor d_in(7);
      d_out(7)<=d_in(4) xor d_in(5)xor d_in(6);
END behave;
```

- XOR 4 input 8 bit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY XOR4_8 IS
      PORT (
            in0: in std_logic_vector(7 downto 0);
            outp: out std_logic_vector(7 downto 0);
            in2: in std_logic_vector(7 downto 0);
            in1: in std_logic_vector(7 downto 0);
            in3: in std_logic_vector(7 downto 0));
END XOR4_8;

ARCHITECTURE A OF XOR4_8 IS
BEGIN
      outp <= in0 xor in1 xor in2 xor in3;
END A;
```

ภาคผนวก ข

โค้ด M-File สำหรับการเข้าและถอดรหัสแบบ AES

หัวข้อนี้จะแสดงโค้ด M-File ซึ่งใช้กับโปรแกรม MATLAB เพื่อหาผลลัพธ์ของการเข้าและถอดรหัสตามอัลกอริทึม AES และใช้เปรียบเทียบผลลัพธ์กับโค้ดภาษาวีเอชดีแอลในการทดลองด้วย โดย ไฟล์ Project_enc.m ใช้กับการเข้ารหัส และProject_dec.m ใช้กับการถอดรหัส และตัวแปร State in คืออินพุทของระบบ และ State Out คือเอาท์พุทของระบบ

Project_enc.m

```
state_in=[hex2dec('00') hex2dec('44') hex2dec('88') hex2dec('cc');
          hex2dec('11') hex2dec('55') hex2dec('99') hex2dec('dd');
          hex2dec('22') hex2dec('66') hex2dec('aa') hex2dec('ee');
          hex2dec('33') hex2dec('77') hex2dec('bb') hex2dec('ff')];

state_out=[hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
           hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
           hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
           hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')];

W4=        [hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
            hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
            hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
            hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')];

key_in= [hex2dec('00') hex2dec('04') hex2dec('08') hex2dec('0c');
         hex2dec('01') hex2dec('05') hex2dec('09') hex2dec('0d');
         hex2dec('02') hex2dec('06') hex2dec('0a') hex2dec('0e');
         hex2dec('03') hex2dec('07') hex2dec('0b') hex2dec('0f')];

key_out=[hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
         hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
         hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
         hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')];

rcon= [hex2dec('01') hex2dec('02') hex2dec('04') hex2dec('08')
       hex2dec('10') hex2dec('20') hex2dec('40') hex2dec('80')
       hex2dec('1b') hex2dec('36');
       hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
       hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
       hex2dec('00') hex2dec('00');
       hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
       hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
       hex2dec('00') hex2dec('00');
```

```
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00')];

s_box=[hex2dec('63') hex2dec('7c') hex2dec('77') hex2dec('7b')
      hex2dec('f2') hex2dec('6b') hex2dec('6f') hex2dec('c5')
      hex2dec('30') hex2dec('01') hex2dec('67') hex2dec('2b')
      hex2dec('fe') hex2dec('d7') hex2dec('ab') hex2dec('76');
      hex2dec('ca') hex2dec('82') hex2dec('c9') hex2dec('7d')
      hex2dec('fa') hex2dec('59') hex2dec('47') hex2dec('f0')
      hex2dec('ad') hex2dec('d4') hex2dec('a2') hex2dec('af')
      hex2dec('9c') hex2dec('a7') hex2dec('72') hex2dec('c0');
      hex2dec('b7') hex2dec('fd') hex2dec('93') hex2dec('26')
      hex2dec('36') hex2dec('3f') hex2dec('f7') hex2dec('cc')
      hex2dec('34') hex2dec('a5') hex2dec('e5') hex2dec('f1')
      hex2dec('71') hex2dec('d8') hex2dec('31') hex2dec('15');
      hex2dec('04') hex2dec('c7') hex2dec('23') hex2dec('c3')
      hex2dec('18') hex2dec('96') hex2dec('05') hex2dec('9a')
      hex2dec('07') hex2dec('12') hex2dec('80') hex2dec('e2')
      hex2dec('eb') hex2dec('27') hex2dec('b2') hex2dec('75');
      hex2dec('09') hex2dec('83') hex2dec('2c') hex2dec('1a')
      hex2dec('1b') hex2dec('6e') hex2dec('5a') hex2dec('a0')
      hex2dec('52') hex2dec('3b') hex2dec('d6') hex2dec('b3')
      hex2dec('29') hex2dec('e3') hex2dec('2f') hex2dec('84');
      hex2dec('53') hex2dec('d1') hex2dec('00') hex2dec('ed')
      hex2dec('20') hex2dec('fc') hex2dec('b1') hex2dec('5b')
      hex2dec('6a') hex2dec('cb') hex2dec('be') hex2dec('39')
      hex2dec('4a') hex2dec('4c') hex2dec('58') hex2dec('cf');
      hex2dec('d0') hex2dec('ef') hex2dec('aa') hex2dec('fb')
      hex2dec('43') hex2dec('4d') hex2dec('33') hex2dec('85')
      hex2dec('45') hex2dec('f9') hex2dec('02') hex2dec('7f')
      hex2dec('50') hex2dec('3c') hex2dec('9f') hex2dec('a8');
      hex2dec('51') hex2dec('a3') hex2dec('40') hex2dec('8f')
      hex2dec('92') hex2dec('9d') hex2dec('38') hex2dec('f5')
      hex2dec('bc') hex2dec('b6') hex2dec('da') hex2dec('21')
      hex2dec('10') hex2dec('ff') hex2dec('f3') hex2dec('d2');
      hex2dec('cd') hex2dec('0c') hex2dec('13') hex2dec('ec')
      hex2dec('5f') hex2dec('97') hex2dec('44') hex2dec('17')
      hex2dec('c4') hex2dec('a7') hex2dec('7e') hex2dec('3d')
      hex2dec('64') hex2dec('5d') hex2dec('19') hex2dec('73');
      hex2dec('60') hex2dec('81') hex2dec('4f') hex2dec('dc')
      hex2dec('22') hex2dec('2a') hex2dec('90') hex2dec('88')
      hex2dec('46') hex2dec('ee') hex2dec('b8') hex2dec('14')
      hex2dec('de') hex2dec('5e') hex2dec('0b') hex2dec('db');
      hex2dec('e0') hex2dec('32') hex2dec('3a') hex2dec('0a')
      hex2dec('49') hex2dec('06') hex2dec('24') hex2dec('5c')
      hex2dec('c2') hex2dec('d3') hex2dec('ac') hex2dec('62')
      hex2dec('91') hex2dec('95') hex2dec('e4') hex2dec('79');
      hex2dec('e7') hex2dec('c8') hex2dec('37') hex2dec('6d')
      hex2dec('8d') hex2dec('d5') hex2dec('4e') hex2dec('a9')
      hex2dec('6c') hex2dec('56') hex2dec('f4') hex2dec('ea')
      hex2dec('65') hex2dec('7a') hex2dec('ae') hex2dec('08');
      hex2dec('ba') hex2dec('78') hex2dec('25') hex2dec('2e')
      hex2dec('1c') hex2dec('a6') hex2dec('b4') hex2dec('c6')
```

```
      hex2dec('e8') hex2dec('dd') hex2dec('74') hex2dec('1f')
      hex2dec('4b') hex2dec('bd') hex2dec('8b') hex2dec('8a');
      hex2dec('70') hex2dec('3e') hex2dec('b5') hex2dec('66')
      hex2dec('48') hex2dec('03') hex2dec('f6') hex2dec('0e')
      hex2dec('61') hex2dec('35') hex2dec('57') hex2dec('b9')
      hex2dec('86') hex2dec('c1') hex2dec('1d') hex2dec('9e');
      hex2dec('e1') hex2dec('f8') hex2dec('98') hex2dec('11')
      hex2dec('69') hex2dec('d9') hex2dec('8e') hex2dec('94')
      hex2dec('9b') hex2dec('1e') hex2dec('87') hex2dec('e9')
      hex2dec('ce') hex2dec('55') hex2dec('28') hex2dec('df');
      hex2dec('8c') hex2dec('a1') hex2dec('89') hex2dec('0d')
      hex2dec('bf') hex2dec('e6') hex2dec('42') hex2dec('68')
      hex2dec('41') hex2dec('99') hex2dec('2d') hex2dec('0f')
      hex2dec('b0') hex2dec('54') hex2dec('bb') hex2dec('16')];

  round=0;
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%Addroundkey
    for i=1:4
        for j=1:4
            state_out(i,j)=bitxor(state_in(i,j),key_in(i,j));
        end
    end
    round = round+1;
%++++++++++++++++++++++++++++++++++++++
while round < 11
    for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
    end
%++++++++++++++++++++++++++++++++++++++++
%Subbyte

    for i=1:4
        for j=1:4
        c=(mod(state_in(i,j),16))+1;
        r=(bitand(state_in(i,j),240)/16)+1;
        state_out(i,j)=s_box(r,c);
        end
    end
%++++++++++++++++++++++++++++++++
for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
    end
%++++++++++++++++++++++++++++++++++++++++++++++++
%ShiftRow
    for j=1:4
        temp1=1+(mod(j,4));
        temp2=1+(mod((j+1),4));
        temp3=1+(mod((j+2),4));
        state_out(1,j)=state_in(1,j);
```

```
        state_out(2,j)=state_in(2,temp1);
        state_out(3,j)=state_in(3,temp2);
        state_out(4,j)=state_in(4,temp3);
    end
%+++++++++++++++++++++++++++++++
for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
end
%+++++++++++++++++++++++++++++++++++++++++++
if round ~= 10
%MixColumn
    for j=1:4

state_out(1,j)=bitxor(bitxor(state_in(3,j),state_in(4,j)),bitxor(mul2
                (state_in(1,j)),mul3(state_in(2,j))));

state_out(2,j)=bitxor(bitxor(state_in(4,j),state_in(1,j)),bitxor(mul2
                (state_in(2,j)),mul3(state_in(3,j))));

state_out(3,j)=bitxor(bitxor(state_in(1,j),state_in(2,j)),bitxor(mul2
                (state_in(3,j)),mul3(state_in(4,j))));

state_out(4,j)=bitxor(bitxor(state_in(2,j),state_in(3,j)),bitxor(mul2
                (state_in(4,j)),mul3(state_in(1,j))));
    end
end
%+++++++++++++++++++++++++++++++++++++++

%KeyExpansion   input is key_in
    for i=1:4

        tmp=1+(mod(i,4));
        W4(i,4)=key_in(tmp,4);
        tr=bitand(W4(i,4),240);
        tc=bitand(W4(i,4),15);
        c=(mod(tc,16))+1;
        r=(tr/16)+1;
        W4sbox(i,4)=s_box(r,c);
    end
    for i=1:4
key_out(i,1)=bitxor(bitxor(W4sbox(i,4),rcon(i,round)),key_in(i,1));
    end
    for j=1:3
        for i=1:4
            key_out(i,(j+1))=bitxor(key_out(i,j),key_in(i,(j+1)));
        end
    end
%+++++++++++++++++++++++++++++++++++++++++++
for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
```

```
end
%+++++++++++++++++++++++++++++++++++++++++++
%Addroundkey
    for i=1:4
        for j=1:4
            state_out(i,j)=bitxor(state_in(i,j),key_out(i,j));
        end
    end
%+++++++++++++++++++++++++++++++++++++++++++
    for i=1:4
        for j=1:4
            key_in(i,j)=key_out(i,j);
        end
    end
   round = round+1;
end
```

Project_dec.m

```
state_in=[hex2dec('69') hex2dec('6a') hex2dec('d8') hex2dec('70');
          hex2dec('c4') hex2dec('7b') hex2dec('cd') hex2dec('b4');
          hex2dec('e0') hex2dec('04') hex2dec('b7') hex2dec('c5');
          hex2dec('d8') hex2dec('30') hex2dec('80') hex2dec('5a')];

state_out=[hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
           hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
           hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
           hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')];

key_in= [hex2dec('13') hex2dec('e3') hex2dec('f3') hex2dec('4d');
         hex2dec('11') hex2dec('94') hex2dec('07') hex2dec('2b');
         hex2dec('1d') hex2dec('4a') hex2dec('a7') hex2dec('30');
         hex2dec('7f') hex2dec('17') hex2dec('8b') hex2dec('c5')];

key_out=[hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
         hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
         hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00');
         hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')];

rcon= [hex2dec('36') hex2dec('1b') hex2dec('80') hex2dec('40')
      hex2dec('20') hex2dec('10') hex2dec('08') hex2dec('04')
      hex2dec('02') hex2dec('01');
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00');
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00');
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00') hex2dec('00') hex2dec('00')
      hex2dec('00') hex2dec('00')];
```

```
s_boxkey=[hex2dec('63') hex2dec('7c') hex2dec('77') hex2dec('7b')
hex2dec('f2') hex2dec('6b') hex2dec('6f') hex2dec('c5') hex2dec('30')
hex2dec('01') hex2dec('67') hex2dec('2b') hex2dec('fe') hex2dec('d7')
hex2dec('ab') hex2dec('76');
      hex2dec('ca') hex2dec('82') hex2dec('c9') hex2dec('7d')
hex2dec('fa') hex2dec('59') hex2dec('47') hex2dec('f0') hex2dec('ad')
hex2dec('d4') hex2dec('a2') hex2dec('af') hex2dec('9c') hex2dec('a7')
hex2dec('72') hex2dec('c0');
      hex2dec('b7') hex2dec('fd') hex2dec('93') hex2dec('26')
hex2dec('36') hex2dec('3f') hex2dec('f7') hex2dec('cc') hex2dec('34')
hex2dec('a5') hex2dec('e5') hex2dec('f1') hex2dec('71') hex2dec('d8')
hex2dec('31') hex2dec('15');
      hex2dec('04') hex2dec('c7') hex2dec('23') hex2dec('c3')
hex2dec('18') hex2dec('96') hex2dec('05') hex2dec('9a') hex2dec('07')
hex2dec('12') hex2dec('80') hex2dec('e2') hex2dec('eb') hex2dec('27')
hex2dec('b2') hex2dec('75');
      hex2dec('09') hex2dec('83') hex2dec('2c') hex2dec('1a')
hex2dec('1b') hex2dec('6e') hex2dec('5a') hex2dec('a0') hex2dec('52')
hex2dec('3b') hex2dec('d6') hex2dec('b3') hex2dec('29') hex2dec('e3')
hex2dec('2f') hex2dec('84');
      hex2dec('53') hex2dec('d1') hex2dec('00') hex2dec('ed')
hex2dec('20') hex2dec('fc') hex2dec('b1') hex2dec('5b') hex2dec('6a')
hex2dec('cb') hex2dec('be') hex2dec('39') hex2dec('4a') hex2dec('4c')
hex2dec('58') hex2dec('cf');
      hex2dec('d0') hex2dec('ef') hex2dec('aa') hex2dec('fb')
hex2dec('43') hex2dec('4d') hex2dec('33') hex2dec('85') hex2dec('45')
hex2dec('f9') hex2dec('02') hex2dec('7f') hex2dec('50') hex2dec('3c')
hex2dec('9f') hex2dec('a8');
      hex2dec('51') hex2dec('a3') hex2dec('40') hex2dec('8f')
hex2dec('92') hex2dec('9d') hex2dec('38') hex2dec('f5') hex2dec('bc')
hex2dec('b6') hex2dec('da') hex2dec('21') hex2dec('10') hex2dec('ff')
hex2dec('f3') hex2dec('d2');
      hex2dec('cd') hex2dec('0c') hex2dec('13') hex2dec('ec')
hex2dec('5f') hex2dec('97') hex2dec('44') hex2dec('17') hex2dec('c4')
hex2dec('a7') hex2dec('7e') hex2dec('3d') hex2dec('64') hex2dec('5d')
hex2dec('19') hex2dec('73');
      hex2dec('60') hex2dec('81') hex2dec('4f') hex2dec('dc')
hex2dec('22') hex2dec('2a') hex2dec('90') hex2dec('88') hex2dec('46')
hex2dec('ee') hex2dec('b8') hex2dec('14') hex2dec('de') hex2dec('5e')
hex2dec('0b') hex2dec('db');
      hex2dec('e0') hex2dec('32') hex2dec('3a') hex2dec('0a')
hex2dec('49') hex2dec('06') hex2dec('24') hex2dec('5c') hex2dec('c2')
hex2dec('d3') hex2dec('ac') hex2dec('62') hex2dec('91') hex2dec('95')
hex2dec('e4') hex2dec('79');
      hex2dec('e7') hex2dec('c8') hex2dec('37') hex2dec('6d')
hex2dec('8d') hex2dec('d5') hex2dec('4e') hex2dec('a9') hex2dec('6c')
hex2dec('56') hex2dec('f4') hex2dec('ea') hex2dec('65') hex2dec('7a')
hex2dec('ae') hex2dec('08');
      hex2dec('ba') hex2dec('78') hex2dec('25') hex2dec('2e')
hex2dec('1c') hex2dec('a6') hex2dec('b4') hex2dec('c6') hex2dec('e8')
hex2dec('dd') hex2dec('74') hex2dec('1f') hex2dec('4b') hex2dec('bd')
hex2dec('8b') hex2dec('8a');
      hex2dec('70') hex2dec('3e') hex2dec('b5') hex2dec('66')
hex2dec('48') hex2dec('03') hex2dec('f6') hex2dec('0e') hex2dec('61')
```

```
hex2dec('35') hex2dec('57') hex2dec('b9') hex2dec('86') hex2dec('c1')
hex2dec('1d') hex2dec('9e');
      hex2dec('e1') hex2dec('f8') hex2dec('98') hex2dec('11')
hex2dec('69') hex2dec('d9') hex2dec('8e') hex2dec('94') hex2dec('9b')
hex2dec('1e') hex2dec('87') hex2dec('e9') hex2dec('ce') hex2dec('55')
hex2dec('28') hex2dec('df');
      hex2dec('8c') hex2dec('a1') hex2dec('89') hex2dec('0d')
hex2dec('bf') hex2dec('e6') hex2dec('42') hex2dec('68') hex2dec('41')
hex2dec('99') hex2dec('2d') hex2dec('0f') hex2dec('b0') hex2dec('54')
hex2dec('bb') hex2dec('16')];


 s_box=[hex2dec('52') hex2dec('09') hex2dec('6a') hex2dec('d5')
hex2dec('30') hex2dec('36') hex2dec('a5') hex2dec('38') hex2dec('bf')
hex2dec('40') hex2dec('a3') hex2dec('9e') hex2dec('81') hex2dec('f3')
hex2dec('d7') hex2dec('fb');
      hex2dec('7c') hex2dec('e3') hex2dec('39') hex2dec('82')
hex2dec('9b') hex2dec('2f') hex2dec('ff') hex2dec('87') hex2dec('34')
hex2dec('8e') hex2dec('43') hex2dec('44') hex2dec('c4') hex2dec('de')
hex2dec('e9') hex2dec('cb');
      hex2dec('54') hex2dec('7b') hex2dec('94') hex2dec('32')
hex2dec('a6') hex2dec('c2') hex2dec('23') hex2dec('3d') hex2dec('ee')
hex2dec('4c') hex2dec('95') hex2dec('0b') hex2dec('42') hex2dec('fa')
hex2dec('c3') hex2dec('4e');
      hex2dec('08') hex2dec('2e') hex2dec('a1') hex2dec('66')
hex2dec('28') hex2dec('d9') hex2dec('24') hex2dec('b2') hex2dec('76')
hex2dec('5b') hex2dec('a2') hex2dec('49') hex2dec('6d') hex2dec('8b')
hex2dec('d1') hex2dec('25');
      hex2dec('72') hex2dec('f8') hex2dec('f6') hex2dec('64')
hex2dec('86') hex2dec('68') hex2dec('98') hex2dec('16') hex2dec('d4')
hex2dec('a4') hex2dec('5c') hex2dec('cc') hex2dec('5d') hex2dec('65')
hex2dec('b6') hex2dec('92');
      hex2dec('6c') hex2dec('70') hex2dec('48') hex2dec('50')
hex2dec('fd') hex2dec('ed') hex2dec('b9') hex2dec('da') hex2dec('5e')
hex2dec('15') hex2dec('46') hex2dec('57') hex2dec('a7') hex2dec('8d')
hex2dec('9d') hex2dec('84');
      hex2dec('90') hex2dec('d8') hex2dec('ab') hex2dec('00')
hex2dec('8c') hex2dec('bc') hex2dec('d3') hex2dec('0a') hex2dec('f7')
hex2dec('e4') hex2dec('58') hex2dec('05') hex2dec('b8') hex2dec('b3')
hex2dec('45') hex2dec('06');
      hex2dec('d0') hex2dec('2c') hex2dec('1e') hex2dec('8f')
hex2dec('ca') hex2dec('3f') hex2dec('0f') hex2dec('02') hex2dec('c1')
hex2dec('af') hex2dec('bd') hex2dec('03') hex2dec('01') hex2dec('13')
hex2dec('8a') hex2dec('6b');
      hex2dec('3a') hex2dec('91') hex2dec('11') hex2dec('41')
hex2dec('4f') hex2dec('67') hex2dec('dc') hex2dec('ea') hex2dec('97')
hex2dec('f2') hex2dec('cf') hex2dec('ce') hex2dec('f0') hex2dec('b4')
hex2dec('e6') hex2dec('73');
      hex2dec('96') hex2dec('ac') hex2dec('74') hex2dec('22')
hex2dec('e7') hex2dec('ad') hex2dec('35') hex2dec('85') hex2dec('e2')
hex2dec('f9') hex2dec('37') hex2dec('e8') hex2dec('1c') hex2dec('75')
hex2dec('df') hex2dec('6e');
```

```
       hex2dec('47') hex2dec('f1') hex2dec('1a') hex2dec('71')
hex2dec('1d') hex2dec('29') hex2dec('c5') hex2dec('89') hex2dec('6f')
hex2dec('b7') hex2dec('62') hex2dec('0e') hex2dec('aa') hex2dec('18')
hex2dec('be') hex2dec('1b');
       hex2dec('fc') hex2dec('56') hex2dec('3e') hex2dec('4b')
hex2dec('c6') hex2dec('d2') hex2dec('79') hex2dec('20') hex2dec('9a')
hex2dec('db') hex2dec('c0') hex2dec('fe') hex2dec('78') hex2dec('cd')
hex2dec('5a') hex2dec('f4');
       hex2dec('1f') hex2dec('dd') hex2dec('a8') hex2dec('33')
hex2dec('88') hex2dec('07') hex2dec('c7') hex2dec('31') hex2dec('b1')
hex2dec('12') hex2dec('10') hex2dec('59') hex2dec('27') hex2dec('80')
hex2dec('ec') hex2dec('5f');
       hex2dec('60') hex2dec('51') hex2dec('7f') hex2dec('a9')
hex2dec('19') hex2dec('b5') hex2dec('4a') hex2dec('0d') hex2dec('2d')
hex2dec('e5') hex2dec('7a') hex2dec('9f') hex2dec('93') hex2dec('c9')
hex2dec('9c') hex2dec('ef');
       hex2dec('a0') hex2dec('e0') hex2dec('3b') hex2dec('4d')
hex2dec('ae') hex2dec('2a') hex2dec('f5') hex2dec('b0') hex2dec('c8')
hex2dec('eb') hex2dec('bb') hex2dec('3c') hex2dec('83') hex2dec('53')
hex2dec('99') hex2dec('61');
       hex2dec('17') hex2dec('2b') hex2dec('04') hex2dec('7e')
hex2dec('ba') hex2dec('77') hex2dec('d6') hex2dec('26') hex2dec('e1')
hex2dec('69') hex2dec('14') hex2dec('63') hex2dec('55') hex2dec('21')
hex2dec('0c') hex2dec('7d')];


   round=0;
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%Addroundkey
    for i=1:4
        for j=1:4
            state_out(i,j)=bitxor(state_in(i,j),key_in(i,j));
        end
    end
    round = round+1;
%++++++++++++++++++++++++++++++++++++++++++++
while round < 11
    for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
    end
%++++++++++++++++++++++++++++++++++++++++++++
%Subbyte

    for i=1:4
        for j=1:4
        c=(mod(state_in(i,j),16))+1;
        r=(bitand(state_in(i,j),240)/16)+1;
        state_out(i,j)=s_box(r,c);
        end
    end
%++++++++++++++++++++++++++++++
for i=1:4
```

```
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
    end
%+++++++++++++++++++++++++++++++++++++++++++
%ShiftRow
    for j=1:4
        temp3=1+(mod(j,4));
        temp2=1+(mod((j+1),4));
        temp1=1+(mod((j+2),4));
        state_out(1,j)=state_in(1,j);
        state_out(2,j)=state_in(2,temp1);
        state_out(3,j)=state_in(3,temp2);
        state_out(4,j)=state_in(4,temp3);
    end
%+++++++++++++++++++++++++++++++
for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
end
%+++++++++++++++++++++++++++++++++++++++++++
if round ~= 10
%MixColumn
    for j=1:4

state_out(1,j)=bitxor(bitxor(mul0E(state_in(1,j)),mul0B(state_in(2,j)))
,bitxor(mul0D(state_in(3,j)),mul9(state_in(4,j))));

state_out(2,j)=bitxor(bitxor(mul0E(state_in(2,j)),mul0B(state_in(3,j)))
,bitxor(mul0D(state_in(4,j)),mul9(state_in(1,j))));

state_out(3,j)=bitxor(bitxor(mul0E(state_in(3,j)),mul0B(state_in(4,j)))
,bitxor(mul0D(state_in(1,j)),mul9(state_in(2,j))));

state_out(4,j)=bitxor(bitxor(mul0E(state_in(4,j)),mul0B(state_in(1,j)))
,bitxor(mul0D(state_in(2,j)),mul9(state_in(3,j))));
    end
end
%+++++++++++++++++++++++++++++++++++++++
%KeyExpansion   input is key_in
    for i=1:4
        key_out(i,4)=bitxor(key_in(i,4),key_in(i,3));
        key_out(i,3)=bitxor(key_in(i,3),key_in(i,2));
        key_out(i,2)=bitxor(key_in(i,2),key_in(i,1));
    end
    for i=1:4
        tmp=1+(mod(i,4));
        Wshifted(i,4)=key_out(tmp,4);
        c=(mod(Wshifted(i,4),16))+1;
        r=(bitand(Wshifted(i,4),240)/16)+1;
        Wsbox(i,4)=s_boxkey(r,c);
    end
    for i=1:4
```

```
key_out(i,1)=bitxor(bitxor(Wsbox(i,4),rcon(i,round)),key_in(i,1));
    end
    for i=1:4
        for j=1:4
        toIMC_key(i,j)=key_out(i,j);
        key_in(i,j)=key_out(i,j);
       end
    end
    if round ~=10
     for j=1:4

key_out(1,j)=bitxor(bitxor(mul0E(toIMC_key(1,j)),mul0B(toIMC_key(2,j)))
,bitxor(mul0D(toIMC_key(3,j)),mul9(toIMC_key(4,j))));

key_out(2,j)=bitxor(bitxor(mul0E(toIMC_key(2,j)),mul0B(toIMC_key(3,j)))
,bitxor(mul0D(toIMC_key(4,j)),mul9(toIMC_key(1,j))));

key_out(3,j)=bitxor(bitxor(mul0E(toIMC_key(3,j)),mul0B(toIMC_key(4,j)))
,bitxor(mul0D(toIMC_key(1,j)),mul9(toIMC_key(2,j))));

key_out(4,j)=bitxor(bitxor(mul0E(toIMC_key(4,j)),mul0B(toIMC_key(1,j)))
,bitxor(mul0D(toIMC_key(2,j)),mul9(toIMC_key(3,j))));
     end
   end
%++++++++++++++++++++++++++++++++++++++++++++
for i=1:4
        for j=1:4
            state_in(i,j)=state_out(i,j);
        end
end
%Addroundkey
    for i=1:4
        for j=1:4
            state_out(i,j)=bitxor(state_in(i,j),key_out(i,j));
        end
    end
%++++++++++++++++++++++++++++++++++++++++++++

    round = round+1;
end
```

mul2.m

```
function y=mul2(x)
    m=bitshift(x,1);
    if m > 255
       n=bitand(255,m);
       y=bitxor(27,n);     % xor 1B(hex)
    else
        y=m;
    end
```

mul3.m

```
function y=mul3(x)
    m=bitshift(x,1);
    if m > 255
       n=bitand(255,m);
       k=bitxor(27,n);    % xor 1B(hex)
       y=bitxor(k,x);
    else
        y=bitxor(m,x);
    end
```

mul9.m

```
function y=mul9(x)
    p=bitshift(x,1);
    if p > 255
       n=bitand(255,p);
       p=bitxor(27,n);    % xor 1B(hex)
    end
    q=bitshift(p,1);
    if q > 255
       n=bitand(255,q);
       q=bitxor(27,n);    % xor 1B(hex)
   end
   s=bitshift(q,1);
    if s > 255
       n=bitand(255,s);
       s=bitxor(27,n);    % xor 1B(hex)
   end
   y=bitxor(s,x);
```

mul0B.m

```
function y=mul0B(x)
    p=bitshift(x,1);
    if p > 255
       n=bitand(255,p);
       p=bitxor(27,n);     % xor 1B(hex)
    end
    q=bitshift(p,1);
    if q > 255
       n=bitand(255,q);
       q=bitxor(27,n);     % xor 1B(hex)
   end
   s=bitshift(q,1);
    if s > 255
       n=bitand(255,s);
       s=bitxor(27,n);     % xor 1B(hex)
   end
   y=bitxor(bitxor(s,p),x);
```

mul0D.m

```
function y=mul0D(x)
    p=bitshift(x,1);
    if p > 255
       n=bitand(255,p);
       p=bitxor(27,n);     % xor 1B(hex)
    end
    q=bitshift(p,1);
    if q > 255
       n=bitand(255,q);
       q=bitxor(27,n);     % xor 1B(hex)
   end
   s=bitshift(q,1);
    if s > 255
       n=bitand(255,s);
       s=bitxor(27,n);     % xor 1B(hex)
   end
   y=bitxor(bitxor(s,q),x);
```

mul0E.m

```
function y=mul0E(x)
    p=bitshift(x,1);
    if p > 255
       n=bitand(255,p);
       p=bitxor(27,n);     % xor 1B(hex)
    end
    q=bitshift(p,1);
    if q > 255
       n=bitand(255,q);
       q=bitxor(27,n);     % xor 1B(hex)
   end
   s=bitshift(q,1);
    if s > 255
       n=bitand(255,s);
       s=bitxor(27,n);     % xor 1B(hex)
   end
   y=bitxor(bitxor(s,p),q);
```

ภาคผนวก ค
การใช้งานโปรแกรม ModelSim

ModelSim เป็นซอฟต์แวร์ที่ใช้ในการคอมไพล์และจำลองการทำงาน (Simulate) ของภาษาHDL (Hardware Description Language) สำหรับโครงงานนี้ได้ใช้ ModelSim SE PLUS 5.6 ร่วมในการออกแบบ ซึ่งหัวข้อนี้จะกล่าวถึงการใช้งาน ModelSim SE PLUS 5.6 ขั้นพื้นฐานดังต่อไปนี้

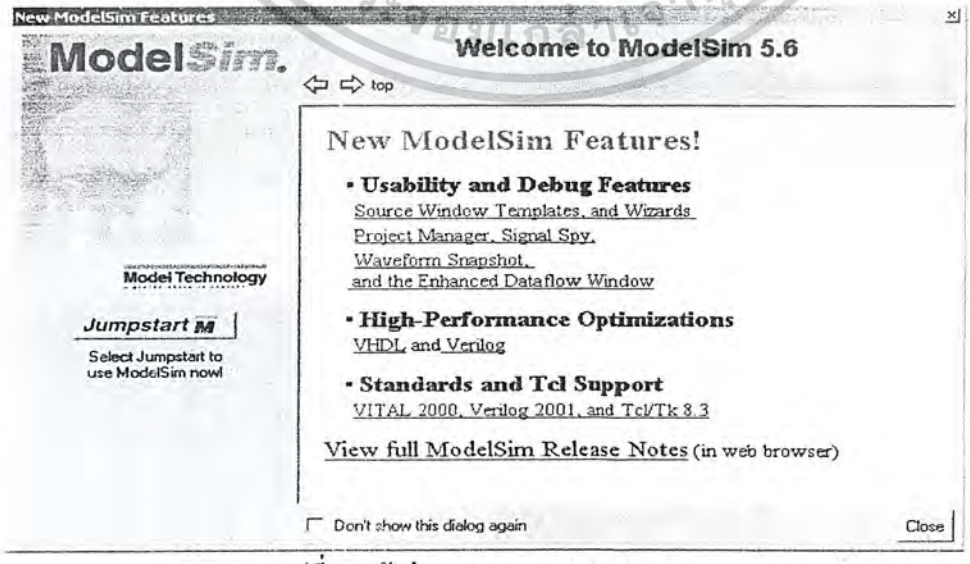
1. การสร้าง Project

Project เป็นที่เก็บรวบรวมคำสั่งหรือโค้ดภาษา HDL ที่เขียนขึ้น ซึ่งง่ายในการเรียกใช้ด้วยเครื่องมือ ต่าง ๆ ของซอฟต์แวร์ (ModelSim) และมีประโยชน์ในการจัดการไฟล์ต่าง ๆ ให้เป็นระบบและตั้งค่าในการจำลองการทำงาน อย่างน้อยที่สุด Project จะใช้เป็นที่เก็บ Work library ดังนั้นจากที่กล่าวมาทั้งหมด Project อาจจะประกอบด้วย:

- คำสั่งหรือโค้ดภาษา HDL ที่เขียนขึ้น หรือโค้ดที่ใช้อ้างอิง
- ไฟล์อื่น ๆ เช่น README หรือเอกสารอื่น ๆ ที่เกี่ยวข้อง
- Local library

ขั้นตอนในการสร้าง Project

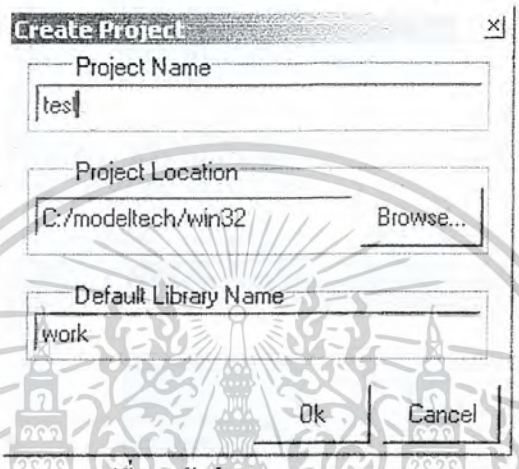
1.เปิดโปรแกรมจากไอคอน  บน Desktop หรือใน Start menu เมื่อเปิดโปรแกรมขึ้นมาครั้งแรกจะเห็นหน้าต่าง Welcome to ModelSim (ถ้าหน้าต่างดังกล่าวไม่ปรากฏขึ้นมาเราอาจเรียกขึ้นมาโดยการเลือก Help > Welcome Menu จากหน้าต่างหลัก)



รูปที่ 1 หน้าต่าง Welcome to ModelSim

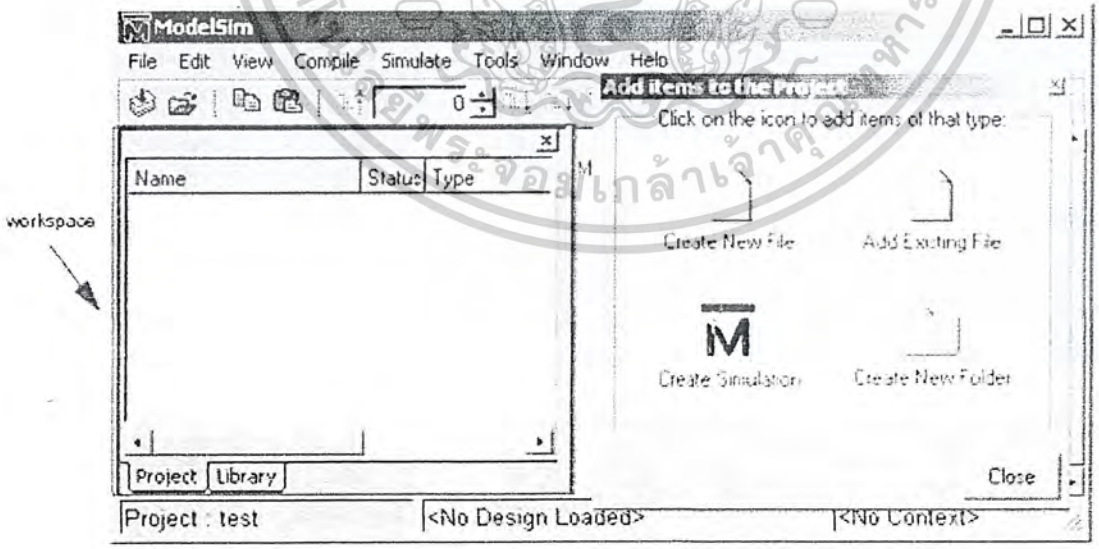
เราสามารถเข้าถึงข้อมูลต่าง ๆ เกี่ยวกับ ModelSim ได้จากหน้าต่างนี้ แต่สำหรับขั้นตอนนี้ให้คลิก Close เพื่อปิดหน้าต่างดังกล่าว

2. เลือก **File > New > Project** (จากหน้าต่างหลัก) จะปรากฏหน้าต่าง Create Project ในช่อง Project Name ให้ใส่ชื่อ Project ที่ต้องการ ในช่อง Project Location ให้เลือกไดเรกทอรี่ที่ต้องการเก็บไฟล์ Project และในช่อง Default Library Name ให้ใส่ "Work"



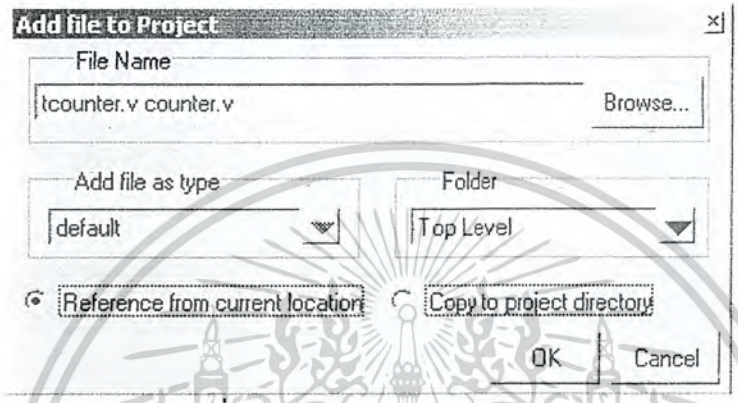
รูปที่ 2 หน้าต่าง Create Project

เมื่อกด Ok จะปรากฏแท็บ Project ใน Workspace ของหน้าต่างหลัก และหน้าต่าง Add items to the Project ขึ้นมา



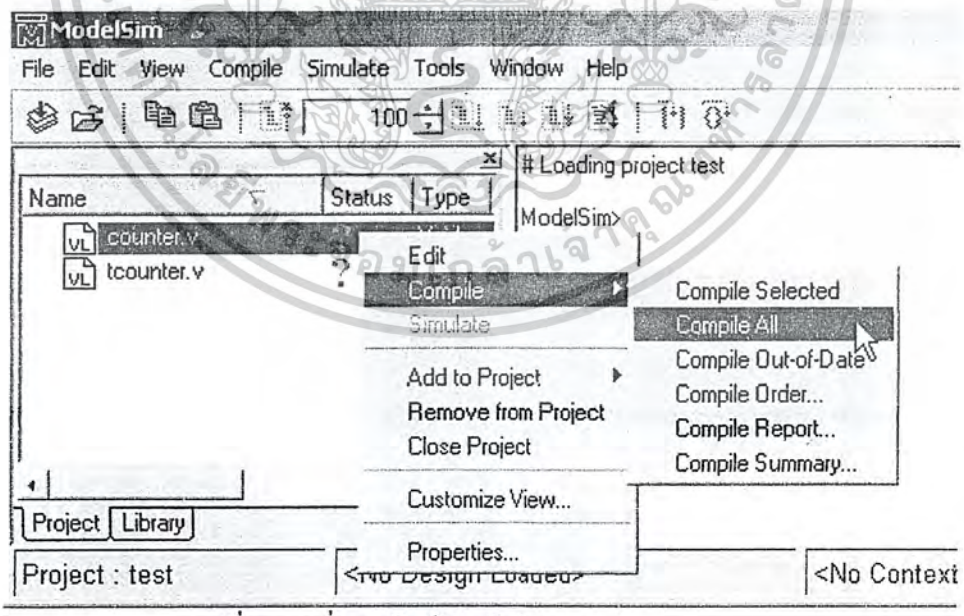
รูปที่ 3 หน้าต่าง Add items to the Project

3. ขั้นตอนต่อไปเป็นการใส่ไฟล์ของ Design unit เข้าไปใน Project ทำได้โดยการคลิกที่ Add Existing File ในหน้าต่าง Add items to the Project จากตัวอย่างนี้เป็นการใส่ไฟล์ Verilog จำนวน2 ไฟล์ โดยการคลิกที่ปุ่ม Browse ในหน้าต่าง Add File to Project เพื่อหาไฟล์ Verilog ที่เขียนไว้แล้ว จากตัวอย่างคือ *tcounter.v* และ *counter.v* จากนั้นเลือก Reference from current location และคลิก Ok



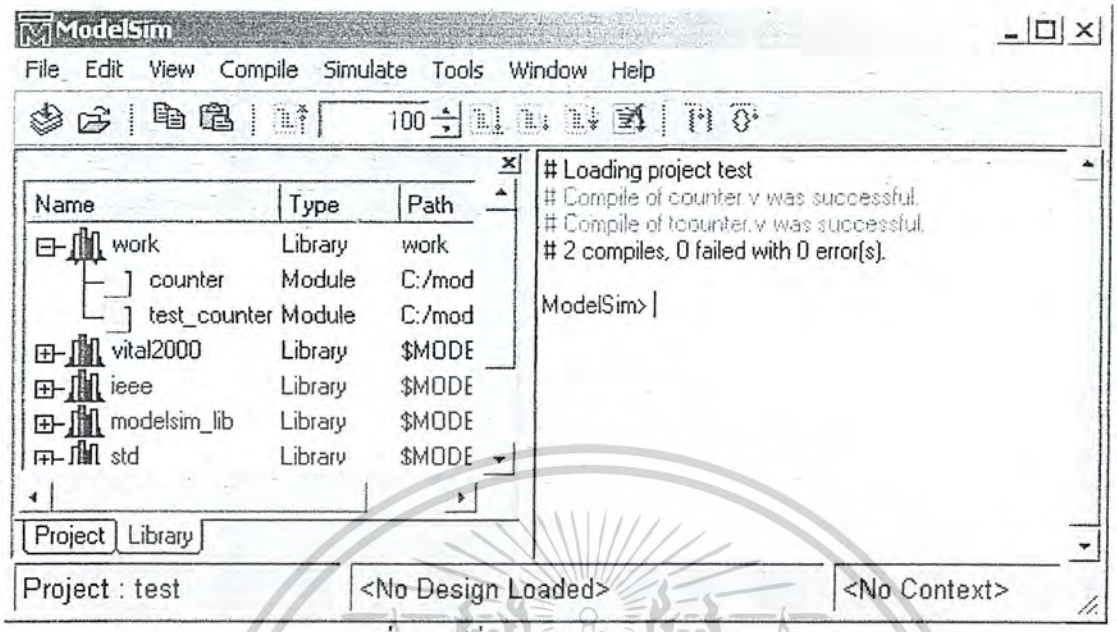
รูปที่ 4 หน้าต่าง Add File to Project

4. คลิกเม้าส์ปุ่มขวาในหน้า Project และเลือก Compile > Compile All



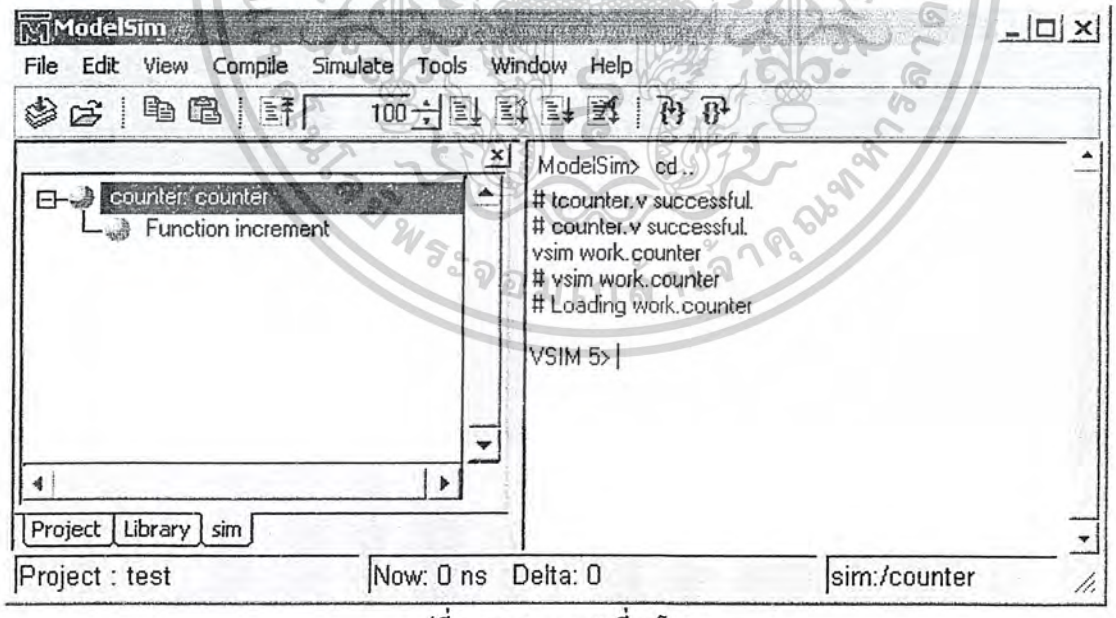
รูปที่ 5 การสั่งคอมไพล์ในหน้าต่าง Project

5. ไฟล์ทั้งหมดใน Project จะถูกคอมไพล์ เมื่อคลิกที่แท็บ Libraby จะเห็น Design unit ที่ถูกคอมไพล์อยู่ในรายการดังรูปข้างล่าง



รูปที่ 6 ไฟล์ที่ถูกคอมไพล์ใน Library Work

6. ขั้นตอนสุดท้าย คือการโหลด Design unit โดยการดับเบิ้ลคลิกที่ Design unit ที่ต้องการในหน้า Library จะเห็นหน้าใหม่ปรากฏขึ้นใน Workspace ซึ่งจะแสดงโครงสร้างของ Design unit ที่เลือก



รูปที่ 7 Design unit ที่ถูกโหลด

เมื่อมาถึงตำแหน่งนี้เราสามารถที่จะวิเคราะห์หรือ Debug และจำลองการทำงานของ Design unit ได้แล้ว ซึ่งจะกล่าวถึงในหัวข้อต่อไป หากต้องการสิ้นสุดการจำลองการทำงานและปิด

Project ทำได้โดยเลือก Simulate > End Simulation และยืนยันหากต้องการออกจากการทำลองการทำงาน ต่อไปเลือกFile > Close > Project และยืนยันหากต้องการปิด Project และเลือก Yes เพื่ออัพเดทไฟล์ Project ของคุณที่มีการเปลี่ยนแปลงในระหว่างการทำงาน

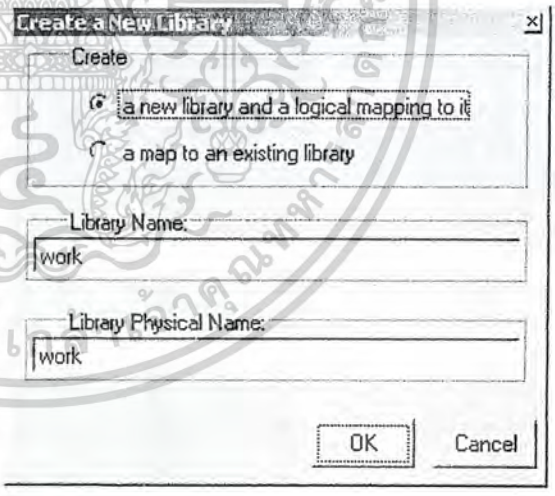
ไฟล์นามสกุล .mpf จะถูกสร้างขึ้นในไดเรกทอรี่ที่ใช้ทำงานซึ่งไฟล์นี้จะมีข้อมูลเกี่ยวกับ Project ที่เราสร้างขึ้น ModelSim จะเปิด Project นี้อย่างอัตโนมัติเมื่อคุณเปิดใช้งานโปรแกรมในครั้งต่อไป

2. การจำลองการทำงานไฟล์ภาษา VHDL เบื้องต้น

<u>การคอมไพล์ไฟล์ที่เขียนขึ้น</u>

1. เริ่มต้นโดยการสร้างไดเรกทอรี่ใหม่แล้วคัดลอก ไฟล์ VHDL (.vhd) ทั้งหมดที่ต้องการจะคอมไพล์มาไว้ในไดเรกทอรี่ที่สร้างใหม่นี้ แล้วตั้งให้ไดเรกทอรี่นี้เป็นไดเรกทอรี่ที่ ModelSim ใช้ทำงานโดยการเลือก File >Change Directory (จากหน้าต่างหลัก)

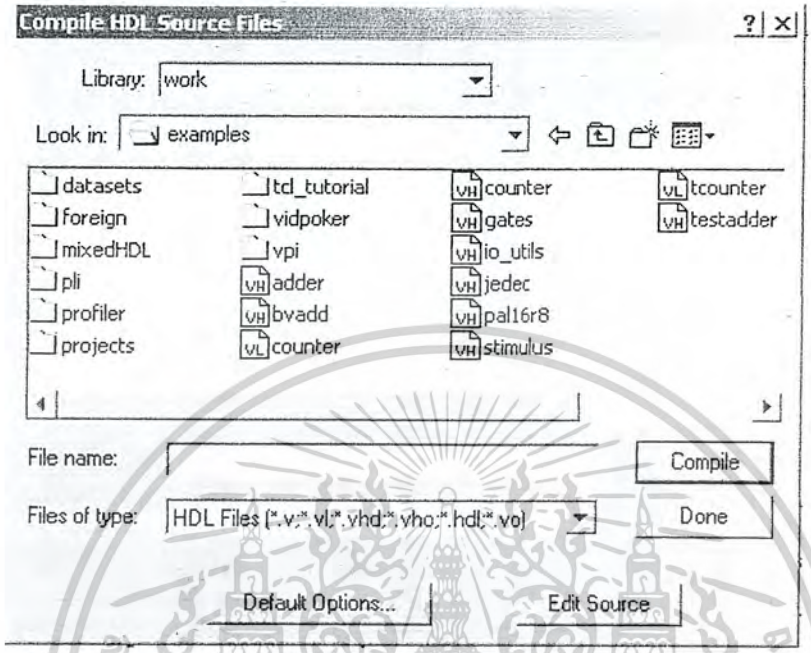
2. ก่อนที่จะคอมไพล์โค้ดภาษา HDL จะต้องมี Library ที่ใช้เก็บผลการคอมไพล์ เพื่อสร้างไดเรกทอรี่ดังกล่าวเลือก File > New > Library จากหน้าต่างหลักจะปรากฏหน้าต่าง Create a New Library แล้วเลือก Create : a new library and a logical mapping to it แล้วพิมพ์ "Work" ในช่อง Library Name แล้วเลือก Ok ไดเรกทอรี่ย่อยชื่อ Work ซึ่งเป็นไดเรกทอรี่ที่ใช้เก็บผลการคอมไพล์จะถูกสร้างขึ้นในไดเรกทอรี่ปัจจุบันที่ ModelSim ทำงานอยู่ ModelSim จะเก็บไฟล์ที่ชื่อ _info ไว้ในไดเรกทอรี่ย่อยดังกล่าว



รูปที่ 8 หน้าต่าง Create a New Library

3. คอมไพล์ไฟล์ที่อยู่ใน Library ที่สร้างขึ้นใหม่โดยเลือก Compile > Compile หรือคลิกที่ไอคอน  หน้าต่าง Compile HDL Source File จะปรากฏขึ้นแล้วเลือกไฟล์ที่ต้องการคอมไพล์จากหน้าต่างดังกล่าวแล้วคลิกที่ Compile แล้วเลือก Doneเมื่อคอมไพล์เสร็จ

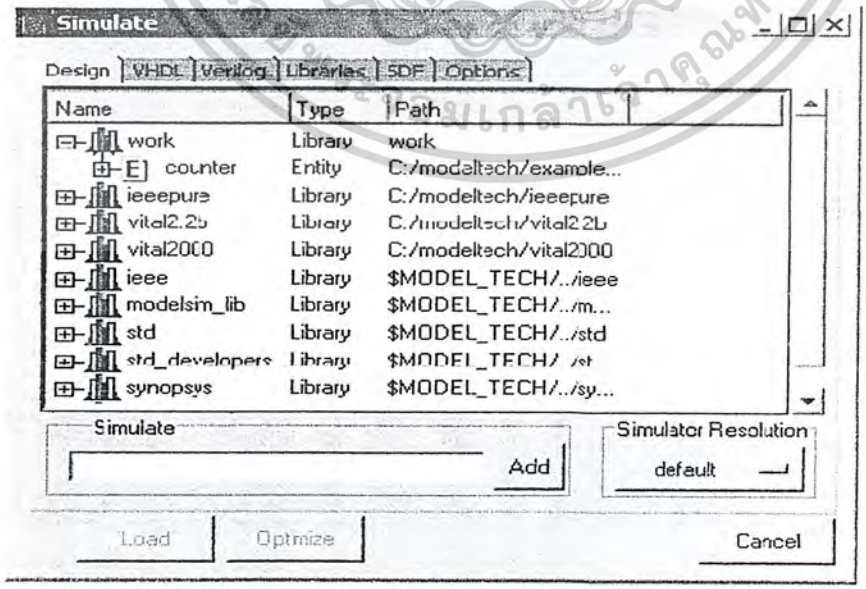
เราสามารถคอมไพล์ได้หลายไฟล์ในการเปิดหน้าต่างนี้เพียงครั้งเดียวโดยการเลือกไฟล์ที่ต้องการคอมไพล์แล้วคลิกที่ Compile ทีละไฟล์



รูปที่ 9 หน้าต่าง Compile HDL Source File

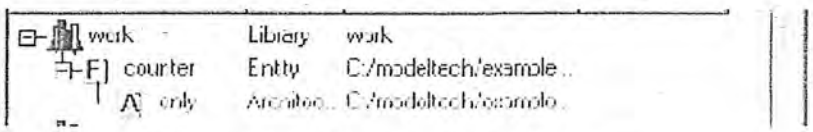
การโหลด Design

1. โหลด Design โดยเลือก Simulate > Simulate จะปรากฏหน้าต่าง Simulate ขึ้นมา เมื่อคลิกที่เครื่องหมาย "+" หน้า 'Work' จะเห็น Design unit ที่ถูกคอมไพล์ไว้แล้ว



รูปที่ 10 หน้าต่าง Simulate

ถ้า Design unit เป็น Entity (เช่น counter ในรูปข้างบน) สามารถขยาย เพื่อดู Architecture ที่มีใน Entity ได้ ดังรูปที่ 11

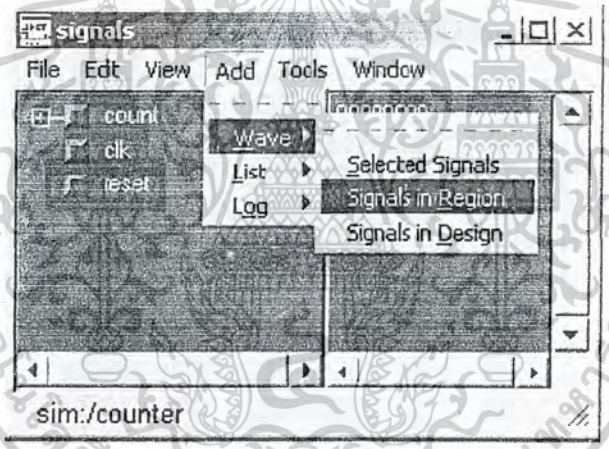


รูปที่ 11 การขยายเพื่อดู Architecture ที่มีใน Entity

เลือก Entity ที่ต้องการโหลด แล้วคลิกที่ Load

2. เลือก View > All Windows จากเมนูในหน้าต่างหลัก เพื่อเปิดทุกๆหน้าต่างของ ModelSim

3. ใส่สัญญาณให้กับหน้าต่าง Wave โดยเลือก Add > Wave >Signal in Region จากเมนูในหน้าต่าง Signals



รูปที่ 12 หน้าต่าง Signals

<u>การจำลองการทำงาน (Simulation)</u>

ต่อไปนี้จะเป็นตัวอย่างการจำลองการทำงานของวงจร Counter (Counter.vhd) จะเริ่มต้นโดยการสร้างสัญญาณนาฬิกาเป็นสัญญาณอินพุท

1. คลิกและพิมพ์คำสั่งต่อไปนี้ที่ VSIM prompt: ในหน้าต่างหลัก

   Force clk 1 50, 0 100 –repeat 100

   หรืออาจสร้างสัญญาณนาฬิกาได้จากเมนูในหน้าต่าง Signals โดยเลือก Edit > Clock

   ModelSim จะตีความดังกล่าวดังนี้

   - สัญญาณ clk จะมีค่าเป็น 1 หลังจากเวลาผ่านไป 50 ns
   - สัญญาณ clk จะมีค่าเป็น 0 หลังจากเวลาผ่านไป 100 ns

- สร้างสัญญาณในลักษณะเดียวกันนี้ทุก ๆ 100 ns

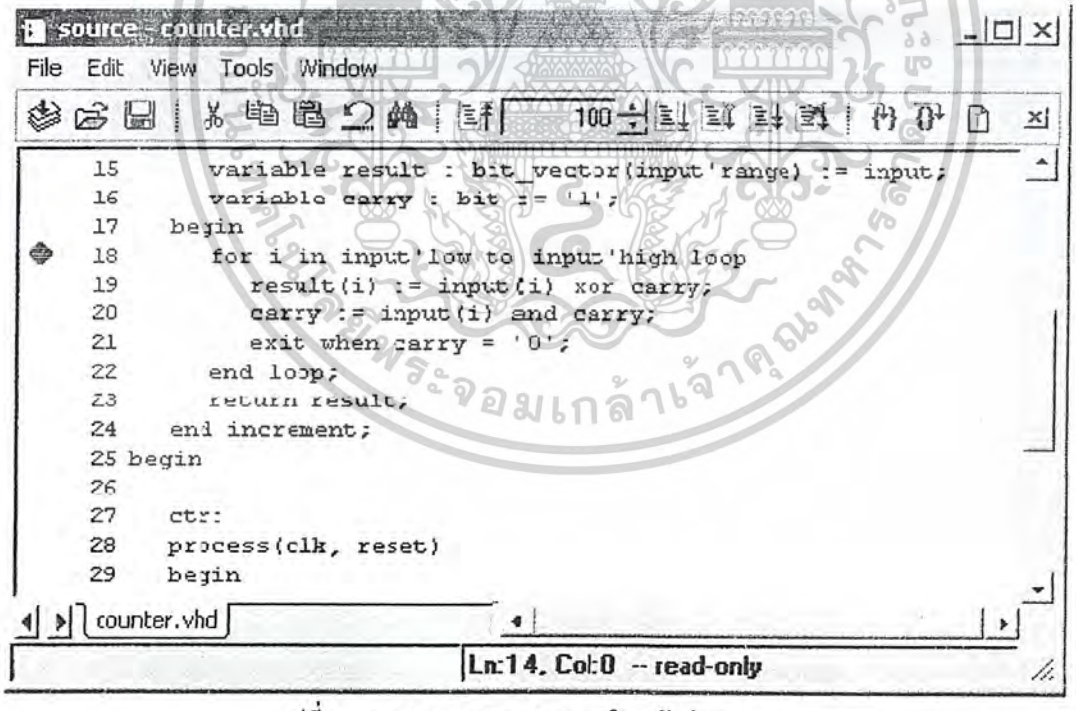
2. ในการ Run เพื่อจำลองการทำงานสามารถทำได้ 2 แบบ คือ Run และ Run-All ซึ่งจะกล่าวถึงดังต่อไปนี้

Run จะจำลองการทำงานโดยจะสิ้นสุดเมื่อเวลาผ่านไป 100 ns

Run-All จะจำลองการทำงานไปเรื่อย ๆ ไม่มีที่สิ้นสุดจนกว่าเราจะสั่งหยุด ซึ่งจะกล่าวถึงในข้อต่อไป

3. เลือกปุ่ม Brake  ที่อยู่บนแถบเครื่องมือของหน้าต่างหลักหรือหน้าต่าง Wave เพื่อหยุดการจำลองการทำงาน ลูกศรที่ชี้อยู่ในหน้าต่าง Source บ่งบอกถึงคำสั่งถัดไปที่จะถูก Execute

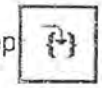
4. ขั้นต่อไปเราจะกำหนด Brakepoint ในฟังก์ชันบนบรรทัดที่ 18 โดยการคลิกบริเวณ เลขบรรทัด 18 หลังจากคลิก จะเห็นจุดสีแดงปรากฏขึ้นที่หน้าตัวเลข 18 ซึ่งเป็นตัวเลขแสดงบรรทัดที่เรากำหนด Brakepoint หากเราคลิกที่บริเวณเดิมอีกครั้งจะเป็นการยกเลิก Brake point และหากต้องการลบ Brakepoint ทำโดยคลิกขวาแล้วเลือก Remove Brakepoint 18



รูปที่ 13 การกำหนด Breakpoint ในหน้าต่าง source

เราสามารถกำหนด Brakepoint ได้เฉพาะบรรทัดที่สามารถ Execute ได้(ตัวอักษรสีน้ำเงิน)เท่านั้น

5. คลิกที่ปุ่ม Continue Run  เพื่อให้ ModelSim ดำเนินการจำลองการทำงานต่อจนกระทั่งถึง Brakepoint ที่กำหนดไว้

6. คลิกที่ปุ่ม Step  เมื่อต้องการ Run ทีละขั้น เพื่อดูการเปลี่ยนแปลงค่าในหน้าต่าง Varibles

7. เมื่อต้องการออกจากการจำลองการทำงาน ให้พิมพ์คำสั่ง

quit -force

3. การหาข้อผิดพลาดของไฟล์ VHDL

เพื่อให้เข้าใจง่ายขึ้น จะอธิบายโดยการยกตัวอย่างประกอบ

<u>การคอมไพล์ และโหลด Design</u>

1. สร้างไดเรกทอรี่ขึ้นใหม่ แล้วคัดลอกไฟล์ VHDL(.vhd) ต่อไปนี้ จาก \<install_dir>\ modeltech\example ไปยังไดเรกทอรี่ใหม่

- gates.vhd
- adder.vhd
- testadder.vhd

2. กำหนดให้ไดเรกทอรี่ที่สร้างขึ้นใหม่เป็นไดเรกทอรี่ที่ Modelsim ใช้ทำงาน โดยเลือก File > Change Directory จากหน้าต่างหลัก

3. พิมพ์คำสั่งต่อไปนี้ที่ MdelSim prompt เพื่อสร้าง Library ใหม่

vlib library_2

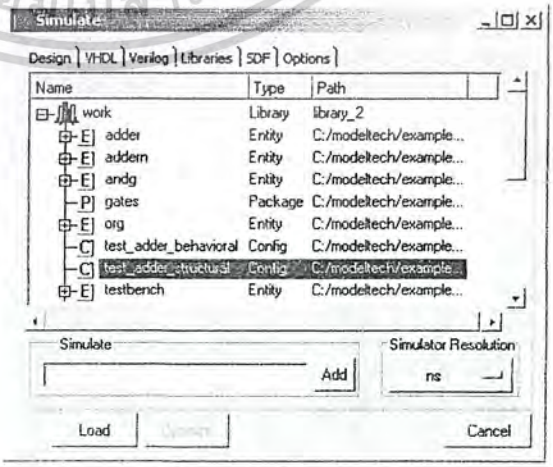
4. กำหนดให้ Library ที่สร้างใหม่เป็น work library โดยใช้คำสั่ง vmap ดังต่อไปนี้

vmap work library_2

5. คอมไพล์ไฟล์ที่ต้องการไปยัง Library ใหม่ โดยพิมพ์คำสั่งต่อไปนี้ที่ ModelSim prompt :Vcom -work library_2 gates.vhd adder.vhd testadder.vhd

6. เปิดหน้าต่าง Simulate โดยเลือก Simulate > Simulate

7. ตั้งค่า Simulator Resolution ให้อยู่ในระดับ ns จากนั้นเลือก test_adder_structural แล้วคลิก Load



รูปที่ 14 หน้าต่าง Simulate

การจำลองการทำงาน

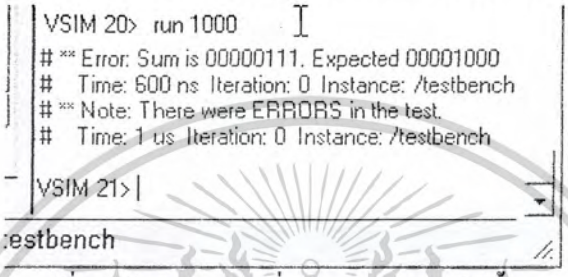
1. เริ่มต้นโดยการเปิดหน้าต่าง Variables และ Signals โดยใช้คำสั่งข้างล่างนี้

View v si

2. ทำการจำลองการทำงานเป็นเวลา 1000 ns โดยใช้คำสั่ง

Run 1000

ข้อความในหน้าต่างหลักจะบอกให้เราทราบ เมื่อมีความผิดพลาดเกิดขึ้น

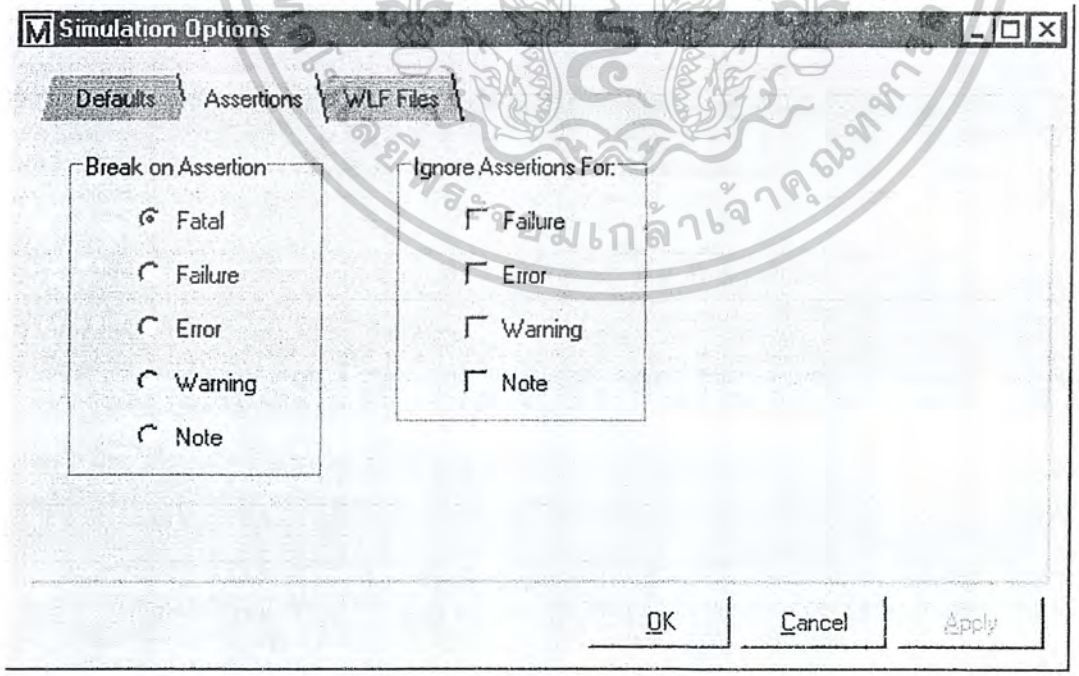


รูปที่ 15 ข้อความแสดงเมื่อมีความผิดพลาดเกิดขึ้น

การหาข้อผิดพลาดของกาจำลองการทำงาน

ต่อไปจะต้องหาที่ผิดให้พบ โดยทำตามขั้นตอนต่อไปนี้

1. เปลี่ยน Simulation assertion option โดยเลือก Simulate >Simulation option จากเมนูในหน้าต่างหลัก



รูปที่ 16 หน้าต่าง Simulation Option ใช้ในการเปลี่ยน Simulation assertion option

2. เลือกแท็บ Assertion แล้วเลือกในช่อง Brake on Assertion เป็น Error แล้วคลิก OK จะทำให้ simulator หยุดอยู่ที่ตำแหน่งที่เกิดความผิดพลาด

3. เริ่มทำการจำลองการทำงานใหม่อีกครั้ง โดยใช้คำสั่งต่อไปนี้

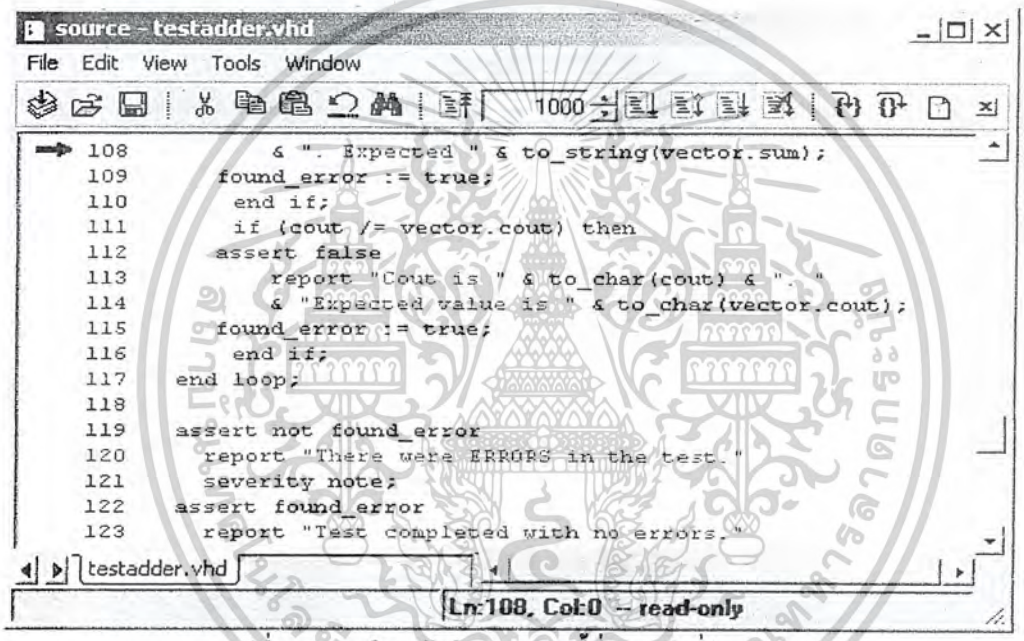
Restart -f

-f จะทำให้ ModelSim เริ่มต้นจำลองการทำงานใหม่โดยไม่ต้องมีหน้าต่างยืนยัน

4. จำลองการทำงานอีกครั้งเป็นเวลา 1000 ns โดยใช้คำสั่ง

Run 1000

จะสังเกตเห็นลูกศรในหน้าต่าง Source ชี้ที่ตำแหน่งที่ผิดพลาด

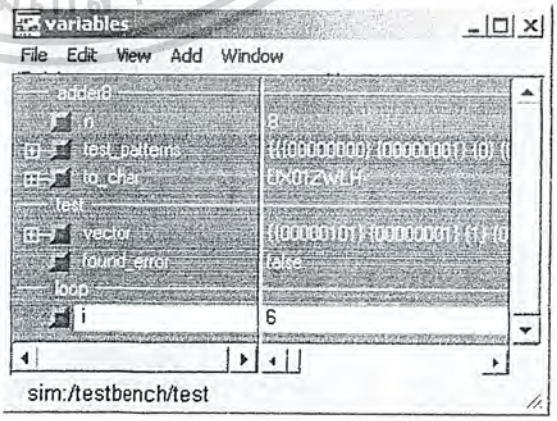


รูปที่ 17 ลูกศรในหน้าต่าง Source ชี้ที่ตำแหน่งที่ผิดพลาด

5. เมื่อดูที่หน้าต่าง Variables จะเห็นตัวแปร i = 6 ซึ่งหมายความว่าการจำลองการทำงานหยุดเมื่อทำไป 6 รอบ

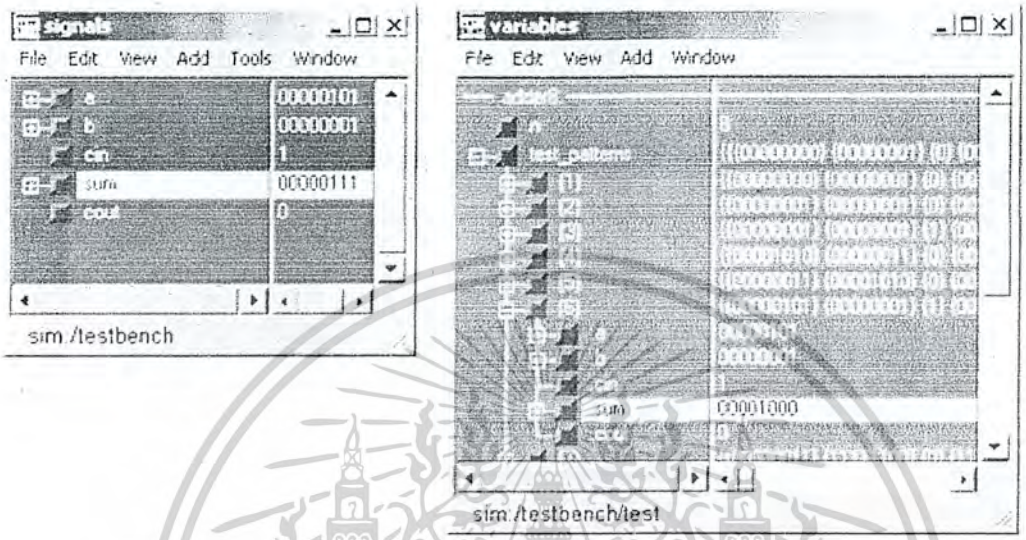
6. ขยายตัวแปรที่ชื่อ test_pattern โดยการคลิกที่เครื่องหมาย "+"

7. ขยายตัวแปรที่เป็นอาเรย์ที่ 6 ของ test_pattern คือ test_pattern(6) โดยการคลิกที่เครื่องหมาย "+" จะพบว่าสัญญาณ sum



รูปที่ 18 หน้าต่าง Variable แสดงตัวแปร i = 6

มีค่าไม่เท่ากับ sum ในหน้าต่าง Variables ซึ่งปกติแล้วจะต้องมีค่าเท่ากัน นั่นคือเกิดความผิดพลาดขึ้นที่จุดนี้ เพื่อแก้ไขให้ถูกต้อง ให้จำลองการทำงานใหม่อีกครั้ง และปรับปรุงค่าเริ่มต้นของตัวแปร test



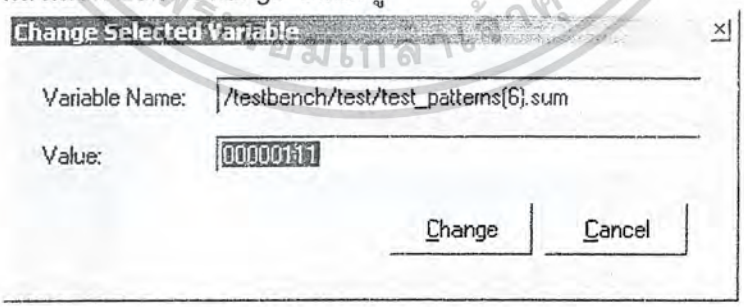
รูปที่ 19 แสดงการเปรียบเทียบ sum ในหน้าต่าง Signals และ Variables

8. เริ่มจำลองการทำงานใหม่อีกครั้งด้วยคำสั่ง

Restart -f

9. อัพเดทหน้าต่าง Variables โดยเลือก test process ในหน้าต่าง process

10. ขยายตัวแปร test_patterns และ test_pattern (6) ในหน้าต่าง Variables อีกครั้ง และคลิกที่ชื่อ .sum แล้วเลือก Edit >Change จากเมนู



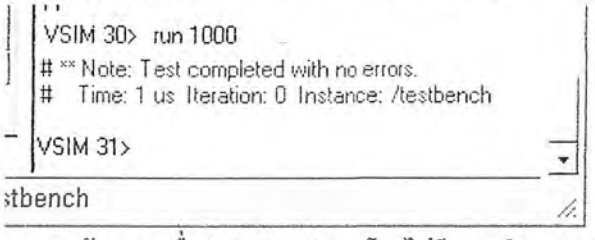
รูปที่ 20 หน้าต่าง Change Selected Variable ใช้ในการเปลี่ยนค่าตัวแปรชั่วคราว

11. เปลี่ยนค่าในช่อง Value เป็น 00000111 แล้วคลิก Change (การเปลี่ยนแปลงค่าของตัวแปรในลักษณะนี้เป็นการเปลี่ยนแบบชั่วคราวเท่านั้น ถ้าต้องการเปลี่ยนแบบถาวรจะต้องเปลี่ยนใน source code ที่เขียนขึ้น)

12. จำลองการทำงานเป็นเวลา 1000 ns อีกครั้งหนึ่งด้วยคำสั่ง

Run 1000

จะพบว่าการจำลองการทำงานในครั้งนี้ไม่มีความผิดพลาดเกิดขึ้น



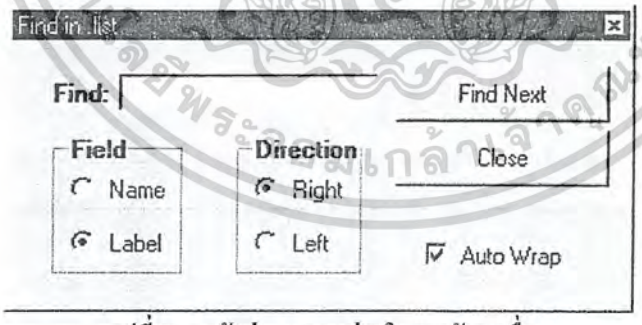
รูปที่ 21 แสดงข้อความเมื่อจำลองการทำงานโดยไม่มีความผิดพลาดเกิดขึ้น

13. ถ้าต้องการออกจากการจำลองการทำงาน ทำได้โดยพิมพ์คำสั่งต่อไปนี้ที่ VSIM prompt: quit –f

4. การค้นหาชื่อและค่าต่าง ๆ

<u>การค้นหาชื่อในหน้าต่างแผนภูมิต้นไม้</u>

เราสามารถหาชื่อของสิ่งที่เกี่ยวกับ HDL โดยเลือก Edit >Find ได้จากเมนูในหน้าต่าง Dataflow, List, Process, Signals, Source, Structure, Variables, และ Wave จะปรากฏหน้าต่าง Find ขึ้นมา

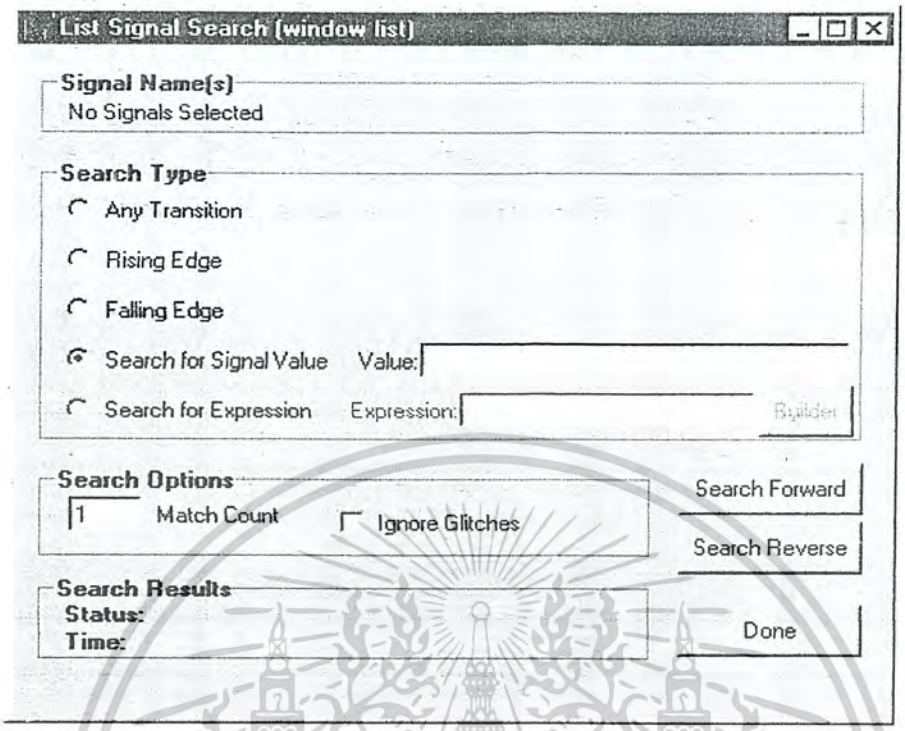


รูปที่ 22 หน้าต่าง Find ช่วยในการค้นหาชื่อ

พิมพ์ชื่อที่ต้องการค้นหา และคลิก Find Next เพื่อค้นหา

<u>การค้นหาค่าในหน้าต่าง List และ Wave</u>

เราสามารถค้นหาค่าในหน้าต่าง List และ Wave โดยเลือก Edit > Search จากเมนูของหน้าต่าง จะปรากฏหน้าต่าง Signal Search ขึ้นมา



รูปที่ 23 หน้าต่าง Signal Search

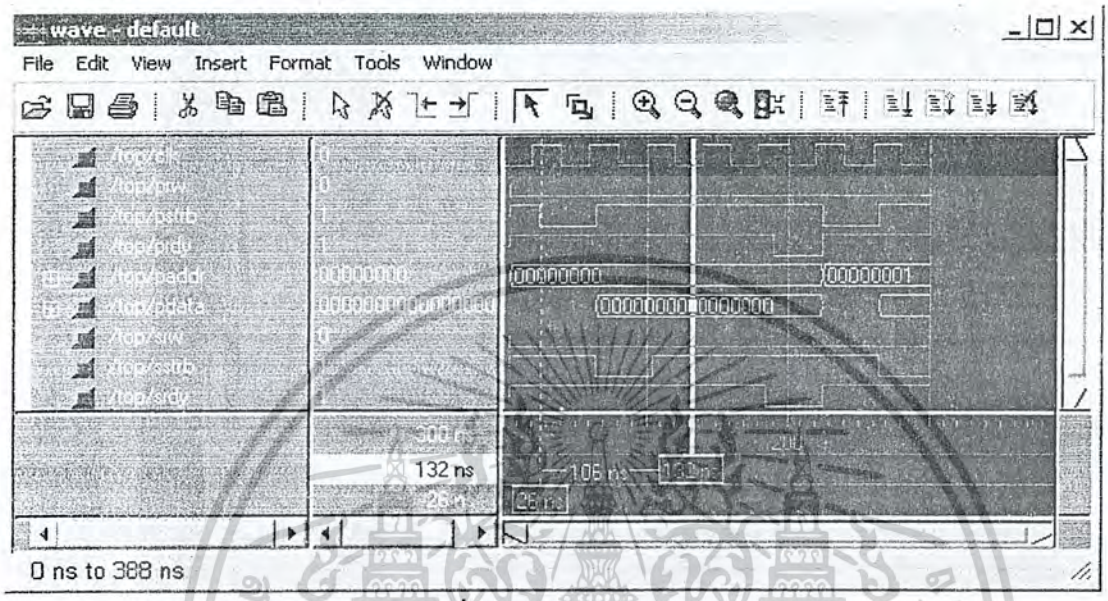
เราสามารถหาค่าของสัญญาณที่มีชื่อปรากฏอยู่ด้านบนของหน้าต่างได้หลายแบบดังต่อไปนี้

- Search Type: Any Transition เป็นการค้นหาค่าที่ตำแหน่งใด ๆ ของสัญญาณ
- Search Type: Rising Edge เป็นการค้นหาขอบขาขึ้นของสัญญาณ
- Search Type: Falling Edge เป็นการค้นหาขอบขาลงของสัญญาณ
- Search Type: Search for Signal Value เป็นการค้นหาค่าที่ระบุไว้ในช่อง Value ของสัญญาณ ค่าที่ระบุในช่อง Value ควรอยู่ในรูปแบบที่ภาษา HDL สามารถตีความได้
- Search Type: Search for Expression เป็นการค้นหาค่าที่ระบุไว้ในช่อง Expression เราสามารถใช้ Expression Builder โดยการคลิกที่ปุ่ม Builder สิ่งที่ระบุไว้ในช่อง Expression อาจจะเกี่ยวข้องกับสัญญาณมากกว่า 1 สัญญาณ แต่สัญญาณที่เกี่ยวข้องนั้นต้องอยู่ในหน้าต่าง List หรือ Wave และอาจรวมไปถึงค่าคงที่ (Constants), ตัวแปร (Variables) และ Tcl macros ถ้าไม่มี Expressions ที่ถูกระบุไว้ การค้นหาจะมีข้อผิดพลาดเกิดขึ้น

5. การใช้หน้าต่าง Wave

เปิดหน้าต่าง Wave ขึ้นมา โดยเลือก View > Wave จากหน้าต่างหลัก

การใช้ Time cursor ในหน้าต่าง Wave



รูปที่ 24 หน้าต่าง Wave

เมื่อหน้าต่าง Wave ปรากฏขึ้นในครั้งแรกจะมี cursor 1 อันที่เวลา 0 โดยการคลิกที่ตำแหน่งใด ๆ ในส่วนที่แสดงรูปคลื่น cursor ก็จะเคลื่อนที่ไปอยู่ที่ตำแหน่งที่เมาส์คลิกนั้น เราสามารถเพิ่ม cursor อีกได้โดยเลือก Insert >Cursor (หรือใช้ปุ่ม Add Cursor ที่แสดงด้านล่าง) cursor ที่ถูกเลือกจะเป็นเส้นสีเข้ม และ cursor ที่ไม่ถูกเลือกจะเป็นเส้นประ ส่วนการลบ cursor ออกทำได้โดยเลือก Edit > Delete Cursor (หรือใช้ปุ่ม Delete Cursor ที่แสดงดังรูปที่ 25)

Add Cursor เพิ่ม cursor เข้าไปยังหน้าต่าง Wave	Delete Cursor ลบ cursor ที่ถูกเลือกออกจากหน้าต่างWave

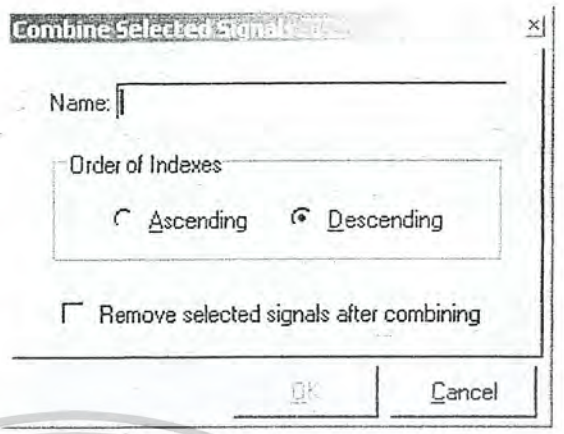
รูปที่ 25 ปุ่มที่ใช้เพิ่ม และลบ Cursor

Cursor แต่ละอันจะแสดงพร้อมกับเวลาที่ cursor นั้นอยู่ เมื่อมี cursor มากกว่า 1 อัน ModelSim จะแสดงผลต่างของเวลาของ cursor ที่อยู่ติดกันด้วย เมื่อคลิกที่ส่วนแสดงรูปคลื่น cursor ที่อยู่ใกล้ตำแหน่งที่คลิกจะถูกเลือกและเคลื่อนที่ไปยังตำแหน่งที่ถูกคลิกนั้น

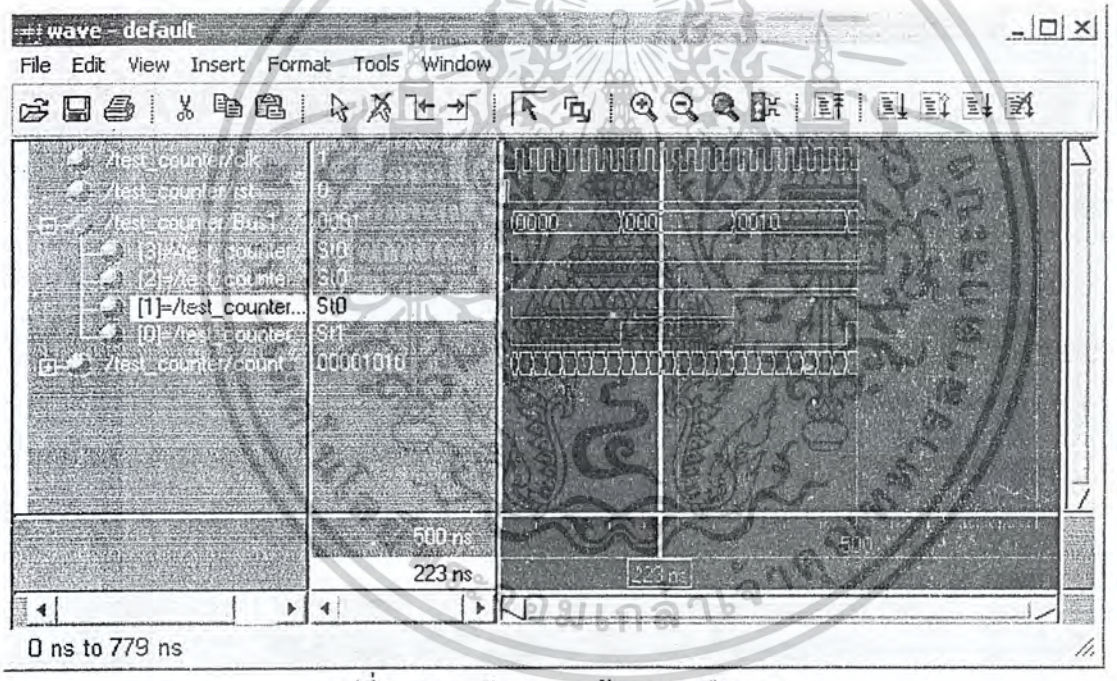
Cursor ถูกออกแบบมาให้อยู่ที่ขอบขาของสัญญาณที่ใกล้ที่สุดทางด้านซ้ายของเมาส์ที่ชี้อยู่ และเราสามารถเคลื่อนย้ายตำแหน่ง cursor ไปยังจุดที่มีการเปลี่ยนแปลงของสัญญาณจุดถัดไปได้โดยใช้ปุ่มข้างล่างนี้

การรวมสัญญาณในหน้าต่าง Wave

เราสามารถที่จะรวมสัญญาณให้เป็น Bus ได้โดยเลือก Tools > Combine Signals เพื่อเปิดหน้าต่าง Combine Selected Signals รูปด้านล่างแสดงให้เห็นสัญญาณ 4 เส้นถูกรวมเป็น Bus โดยเรียกว่า Bus1 สังเกตว่า Bus ดังกล่าวมีค่าที่สร้างจากการรวมสัญญาณเข้าด้วยกัน



รูปที่ 28 หน้าต่าง Combine Selected Signals



รูปที่ 29 แสดงสัญญาณ 4 เส้นถูกรวมเป็น Bus

การสร้างและเรียกดู Datasets

Datasets ทำให้เราสามารถดูผลการจำลองการทำงานก่อนหน้าหรือเพื่อเปรียบเทียบผลการจำลองการทำงานได้ ในการเรียกดู Datasets ก่อนอื่นจะต้องบันทึกผลการจำลองการทำงานในรูปของ WLF file (โดยการใช้คำสั่ง vsim – wlf) เมื่อเราได้บันทึกเป็น WLF file แล้ว เราสามารถเปิดขึ้นมาได้ในลักษณะของ view – mode dataset

ในหัวข้อนี้เราจะทำการเปรียบเทียบ Design ภาษา Verilog 2 แบบคือ แบบ structural description และแบบ RTL description ของ counter ขนาด 4 บิต เริ่มต้นโดยการจำลองการทำงานของแบบ structural description และบันทึกเป็น WLF file จากนั้นเราจะจำลองการทำงานของแบบ RTL description สุดท้ายเราจะเปิด WLF file ในลักษณะของ Dataset และเปรียบเทียบผลการจำลองการทำงานในหน้าต่าง Wave

- *การจำลองการทำงานแบบ Structural*
  1. คอมไพล์รูปแบบ Structural ของ counter โดยใช้คำสั่ง
     Vlog cntr_struct.v
  2. โหลด Design และบันทึกการจำลองการทำงานเป็น WLF file ชื่อ struct.wlf ด้วยคำสั่ง
     Vsim –wlf struct.wlf work.cntr_struct
  3. ต่อไปเราจะ Run DO file ซึ่งจะเป็นการจำลองการทำงานและใส่รูปคลื่นไปยังหน้าต่าง Wave ด้วยคำสั่ง
     Do stimulus.do
     รูปคลื่นจะปรากฏในหน้าต่าง Wave และถูกบันทึกอย่างอัตโนมัติไปยังไฟล์ชื่อ struct.wlf
  4. ออกจากการจำลองการทำงานด้วยคำสั่ง
     Quit – sim
- *การจำลองการทำงานแบบ RTL*
  1. คอมไพล์รูปแบบ RTL ของ counter โดยใช้คำสั่ง
     Vlog cntr_rtl.v
  2. จำลองการทำงานด้วยคำสั่ง
     Vsim work.cntr_rtl
  3. ทำการ Run DO file ด้วยคำสั่ง
     Do stimulus.do
- *การเปรียบเทียบ Design 2 แบบ*

เพื่อเปรียบเทียบการจำลองการทำงานของทั้ง 2 แบบ เราจะสร้างส่วนแสดงผลในหน้าต่าง Wave ขึ้นมาอีก 1 ส่วน เปิดไฟล์ชื่อ struct.wlf และใส่สัญญาณจาก Dataset ไปยังส่วนแสดงผลที่สร้างขึ้นใหม่

1. สร้างส่วนแสดงผลในหน้าต่าง Wave เพิ่มขึ้นอีกหนึ่งส่วนโดยเลือก Insert > Window Pane
2. เปิดไฟล์ struct.wlf ขึ้นมาด้วยคำสั่ง

   Dataset open struct.wlf
3. ใส่สัญญาณสำหรับ Dataset ของ "struct" ด้วยคำสั่ง

   Add wave *

สังเกตว่าชื่อของสัญญาณที่เราเพิ่มเข้าไปคือ "struct" ส่วนชื่อของสัญญาณการจำลองการทำงานที่แอคทีฟอยู่คือ "sim"

ผลการจำลองการทำงานของแต่ละแบบควรจะเหมือนกัน เราสามารถที่จะทำการทดสอบการจำลองการทำงานของทั้ง 2 แบบต่อไปหรือถ้าต้องการออกจากการจำลองการทำงานจะใช้คำสั่ง

Quit - sim

ภาคผนวก ง
ตัวอย่างการเข้าและถอดรหัส*

หัวข้อนี้จะแสดงตัวอย่างการเข้ารหัสและถอดรหัส ซึ่งได้ใช้ในการอ้างอิงกับการจำลองการทำงานในบทที่ 5 ด้วย

input: ข้อมูลอินพุทของการเข้ารหัส
start: stateเริ่มต้นของรอบ
s_box: stateผลลัพธ์จากการแปลงแบบ SubBytes()
s_row: state ผลลัพธ์จากการแปลงแบบ ShiftRows()
m_col: state ผลลัพธ์จากการแปลงแบบ MixColumns()
k_sch: ค่าของ Key สำหรับ รอบที่ r (round[r])
output: ข้อมูลเอาท์พุท

iinput: ข้อมูลอินพุทของการถอดรหัส
istart: stateเริ่มต้นของรอบ
is_box: state ผลลัพธ์จากการแปลงแบบ InvSubBytes()
is_row: state ผลลัพธ์จากการแปลงแบบ InvShiftRows()
im_col: state ผลลัพธ์จากการแปลงแบบ InvMixColumns()
ik_sch: ค่าของ Key สำหรับ รอบที่ r (round[r])
ioutput: ข้อมูลเอาท์พุทจากการถอดรหัส

```
PLAINTEXT    : 00112233445566778899aabbccddeeff
KEY          : 000102030405060708090a0b0c0d0e0f

CIPHER (ENCRYPT):
round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1].start 00102030405060708090a0b0c0d0e0f0
round[ 1].s_box 63cab7040953d051cd60e0e7ba70e1..
round[ 1].s_row 6353e08c0960e104cd70b751ba..d0..
round[ 1].m_col 5f72641557f5bc92f7be3b291d..01..
round[ 1].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
```

* คัดลอกมาจาก [3]

```
round[ 2].start 89d810e8855ace682d1843d8cb128fe4
round[ 2].s_box a761ca9b97be8b45d8ad1a611fc97369
round[ 2].s_row a7be1a6997ad739bd8c9ca451f618b61
round[ 2].m_col ff87968431d86a51645151fa773ad009
round[ 2].k_sch b692cf0b643dbdf1be9bc5006830b3fe
round[ 3].start 4915598f55e5d7a0daca94fa1f0a63f7
round[ 3].s_box 3b59cb73fcd90ee05774222dc067fb68
round[ 3].s_row 3bd92268fc74fb735767cbe0c0590e2d
round[ 3].m_col 4c9c1e66f771f0762c3f868e534df256
round[ 3].k_sch b6ff744ed2c2c9bf6c590cbf0469bf41
round[ 4].start fa636a2825b339c940668a3157244d17
round[ 4].s_box 2dfb02343f6d12dd09337ec75b36e3f0
round[ 4].s_row 2d6d7ef03f33e334093602dd5bfb12c7
round[ 4].m_col 6385b79ffc538df997be478e7547d691
round[ 4].k_sch 47f7f7bc95353e03f96c32bcfd058dfd
round[ 5].start 247240236966b3fa6ed2753288425b6c
round[ 5].s_box 36400926f9336d2d9fb59d23c42c3950
round[ 5].s_row 36339d50f9b539269f2c092dc4406d23
round[ 5].m_col f4bcd45432e554d075f1d6c51dd03b3c
round[ 5].k_sch 3caaa3e8a99f9deb50f3af57adf622aa
round[ 6].start c81677bc9b7ac93b25027992b0261996
round[ 6].s_box e847f56514dadde23f77b64fe7f7d490
round[ 6].s_row e8dab6901477d4653ff7f5e2e747dd4f
round[ 6].m_col 9816ee7400f87f556b2c049c8e5ad036
round[ 6].k_sch 5e390f7df7a69296a7553dc10aa31f6b
round[ 7].start c62fe109f75eedc3cc79395d84f9cf5d
round[ 7].s_box b415f8016858552e4bb6124c5f998a4c
round[ 7].s_row b458124c68b68a014b99f82e5f15554c
round[ 7].m_col c57e1c159a9bd286f05f4be098c63439
round[ 7].k_sch 14f9701ae35fe28c440adf4d4ea9c026
round[ 8].start d1876c0f79c4300ab45594add66ff41f
round[ 8].s_box 3e175076b61c04678dfc2295f6a8bfc0
round[ 8].s_row 3e1c22c0b6fcbf768da85067f6170495
round[ 8].m_col baa03de7a1f9b56ed5512cba5f414d23
round[ 8].k_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 9].start fde3bad205e5d0d73547964ef1fe37f1
round[ 9].s_box 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9].s_row 54d990a16ba09ab596bbf40ea111702f
round[ 9].m_col e9f74eec023020f61bf2ccf2353c21c7
round[ 9].k_sch 549932d1f08557681093ed9cbe2c974e
round[10].start bd6e7c3df2b5779e0b61216e8b10b689
round[10].s_box 7a9f102789d5f50b2beffd9f3dca4ea7
round[10].s_row 7ad5fda789ef4e272bca100b3d9ff59f
round[10].k_sch 13111d7fe3944a17f307a78b4d2b30c5
round[10].output 69c4e0d86a7b0430d8cdb78070b4c55a

INVERSE CIPHER (DECRYPT):
round[ 0].iinput 69c4e0d86a7b0430d8cdb78070b4c55a
round[ 0].ik_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 1].istart 7ad5fda789ef4e272bca100b3d9ff59f
round[ 1].is_box bdb52189f261b63d0b107c9e8b6e776e
round[ 1].is_row bd6e7c3df2b5779e0b61216e8b10b689
round[ 1].im_col 4773b91ff72f354361cb018ea1e6cf2c
round[ 1].ik_sch 13aa29be9c8faff6f770f58000f7bf03
```

```
round[ 2].istart 54d990a16ba09ab596bbf40ea111702f
round[ 2].is_box fde596f1054737d235febad7f1e3d04e
round[ 2].is_row fde3bad205e5d0d73547964ef1fe37f1
round[ 2].im_col 2d7e86a339d9393ee6570a1101904e16
round[ 2].ik_sch 1362a4638f2586486bff5a76f7874a83
round[ 3].istart 3e1c22c0b6fcbf768da85067f6170495
round[ 3].is_box d1c4941f7955f40fb46f6c0ad68730ad
round[ 3].is_row d1876c0f79c4300ab45594add66ff41f
round[ 3].im_col 39daee38f4f1a82aaf432410c36d45b9
round[ 3].ik_sch 8d82fc749c47222be4dadc3e9c7810f5
round[ 4].istart b458124c68b68a014b99f82e5f15554c
round[ 4].is_box c65e395df779cf09ccf9e1c3842fed5d
round[ 4].is_row c62fe109f75eedc3cc79395d84f9cf5d
round[ 4].im_col 9a39bf1d05b20a3a476a0bf79fe51184
round[ 4].ik_sch 72e3098d11c5de5f789dfe1578a2cccb
round[ 5].istart e8dab6901477d4653ff7f5e2e747dd4f
round[ 5].is_box c87a79969b0219bc2526773bb016c992
round[ 5].is_row c81677bc9b7ac93b25027992b0261996
round[ 5].im_col 18f78d779a93eef4f6742967c47f5ffd
round[ 5].ik_sch 2ec410276326d7d26958204a003f32de
round[ 6].istart 36339d50f9b539269f2c092dc4406d23
round[ 6].is_box 2466756c69d25b236e4240fa8872b332
round[ 6].is_row 247240236966b3fa6ed2753288425b6c
round[ 6].im_col 85cf8bf472d124c10348f545329c0053
round[ 6].ik_sch a8a2f5044de2c7f50a7ef79869671294
round[ 7].istart 2d6d7ef03f33e334093602dd5bfb12c7
round[ 7].is_box fab38a1725664d2840246ac957633931
round[ 7].is_row fa636a2825b339c940668a3157244d17
round[ 7].im_col fc1fc1f91934c98210fbfb8da340eb21
round[ 7].ik_sch c7c6e391e54032f1479c306d6319e50c
round[ 8].istart 3bd92268fc74fb735767cbe0c0590e2d
round[ 8].is_box 49e594f755ca638fda0a59a01f15d7fa
round[ 8].is_row 4915598f55e5d7a0daca94fa1f0a63f7
round[ 8].im_col 076518f0b52ba2fb7a15c8d93be45e00
round[ 8].ik_sch a0db02992286d160a2dc029c2485d561
round[ 9].istart a7be1a6997ad739bd8c9ca451f618b61
round[ 9].is_box 895a43e485188fe82d121068cbd8ced8
round[ 9].is_row 89d810e8855ace682d1843d8cb128fe4
round[ 9].im_col ef053f7c8b3d32fd4d2a64ad3c93071a
round[ 9].ik_sch 8c56dff0825dd3f9805ad3fc8659d7fd
round[10].istart 6353e08c0960e104cd70b751bacad0e7
round[10].is_box 0050a0f04090e03080d02070c01060b0
round[10].is_row 00102030405060708090a0b0c0d0e0f0
round[10].ik_sch 000102030405060708090a0b0c0d0e0f
round[10].ioutput 00112233445566778899aabbccddeeff
```

เอกสารอ้างอิง

1. ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ , *VHDL Workshop 2000*, 2000.
2. Douglas E. Ott and Thomas J. Wilderotter, *A DESIGNER'S GUIDE TO VHDL SYNTHESIS* , Kluwer Academic publisher,Boston,1994 .
3. Federal Information Processing Standards Publication 197, *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, November 26, 2001.
4. J. Daemen and V. Rijmen, *AES Proposal: Rijndael*, AES Algorithm Submission, September 3, 1999.
5. Model Technology Inc., *ModelSim SE Tutorial version 5.6*, Mar 7, 2002.
6. P. Noo – in, S. Chantarawong and S. Choomchuay, *An Architecture for S-Box Computation in the AES*, ReCCIT, KMITL, Bangkok.
7. P. Noo – in, S. Chantarawong and S. Choomchuay, *An Architecture for a Compact AES System*, ReCCIT, KMITL, Bangkok.
8. P. Noo – in, S. Chantarawong and S. Choomchuay, *An Architecture for MixColumn Transform in the AES*, ReCCIT, KMITL, Bangkok.
9. Roland Airiau, Jean Michel Berge and Vincent Olive, *Circuit Synthesis with VHDL*, Kluwer Academic publisher, Dordrecht, 1994 .
10. Stefan Sjoholm and Lennart Lindth , *VHDL for Designers* ,Prentice-Hall , London , 1997 .
11. Victor P. Nelson, H. Troy Nagle, Bill D. Carroll and J. David Irwin, *Digital Logic Circuit Analysis and Design*, Prentice-Hall, New Jersey,1995.