การปกปิดข้อมูลสัญญาณเสียงโดยใช้ชิพ FPGA

DATA SCRAMBLE VOICE SIGNAL BY USE FPGA CHIP

โดย

นายรณชิต สีไว เลขประจำตัว 48012185
นายวิชิต ชอบจิตต์ เลขประจำตัว 48012186
นายสุทธิธรรม ทิพวรนิก เลขประจำตัว 48012189

อาจารย์ที่ปรึกษา

ดร. สมศักดิ์ ชุมช่วย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2551

ปริญญานิพนธ์ปีการศึกษา 2551

ภาค วิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การปกปิดข้อมูลสัญญาณเสียงโดยใช้ชิพ FPGA

(Data Scramble Voice Signal By Use FPGA Chip)

ผู้จัดทำ

1. นาย รณชิต สีไว รหัสประจำตัว 48012185
2. นาย วิชิต ชอบจิตต์ รหัสประจำตัว 48012186
3. นาย สุทธิธรรม ทิพวรนิก รหัสประจำตัว 48012189

.....อาจารย์ที่ปรึกษา

(ดร. สมศักดิ์ ชุมช่วย)

การปกปิดข้อมูลสัญญาณเสียงโดยใช้ชิพ FPGA

นายรณชิต สีไว รหัส 48012185
นายวิชิต ชอบจิตต์ รหัส 48012186
นายสุทธิธรรม ทิพวรนิก รหัส 48012189
ดร. สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา
ปีการศึกษา 2551

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ ได้นำเสนอวิธีการที่จะปกปิดข้อมูลสัญญาณเสียง โดยการนำสัญญาณเสียงมาทำการเข้ารหัส โดยใช้ชิพ FPGA เป็นอุปกรณ์ที่ใช้ในการเข้ารหัสสัญญาณ เพื่อให้สัญญาณที่ออกมาไม่สามารถรับฟังได้ ทำให้ได้ข้อมูลที่ผิดเพี้ยนไปจากเดิม แล้วจึงถูกส่งออกไปในช่องทางการสื่อสาร ในส่วนของภาครับจะเป็นการย้อนกระบวนการของภาคส่ง โดยจะเป็นการนำสัญญาณที่ได้รับเข้ามา มาทำการถอดรหัสสัญญาณ เพื่อให้ได้ข้อมูลที่เหมือนกับต้นทาง ระบบปกปิดข้อมูลนี้ถูกสร้างบนชิป FPGA ที่มีขนาดเล็ก ปลอดภัยจากการคัดลอกวงจรต้นแบบ

DATA SCRAMBLE VOICE SIGNAL BY USE FPGA CHIP

Mr.Ronnachit Seewai ID. 48012185

Mr.Wichit Chobjit ID. 48012186

Mr.Sutitam Tipwaranik ID. 48012189

Assoc. Prof. Dr. Somsak Choomchuay Advisor

Education Year 2008

ABSTRACT

This thesis proposes a method of data scramble by coding method. The FPGA chip be the equipment that use in signal encode, for output signal can not listen, make get wrong data from originally. Then signal go out in transmission channel. In the part of receiver, will lead a input signal, do decode signal. For get the data at such as a part send. The encoder and decoder have been implemented onto the FPGA chip, that have small-sized for safe from circuit original duplication.

สารบัญ

สารบัญรูป

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบัน มีการสื่อสารด้วยเสียงพูด โทรสาร หรือข้อมูลข่าวสารต่างๆ ผ่านข่ายสายโทรศัพท์สาธารณะเพิ่มมาขึ้น ในจำนวนเหล่านั้น มีช่องการสื่อสารจำนวนไม่น้อยที่เสี่ยงต่อการถูกจารกรรมดักฟังและถูกทำลายโดยผู้ประสงค์ร้ายต่อข้อมูลหรือผู้ต้องการลักลอบนำข่าวสารไปใช้ประโยชน์โดยไม่ได้ขออนุญาต ทำให้เกิดความเสียหายต่อธุรกิจหรือความเป็นส่วนตัว

มีการนำเทคโนโลยีสมัยใหม่ ทั้งด้านการประมวลข้อมูลและการเข้ารหัสลับ มาช่วยในการรักษาความปลอดภัยของข่าวสารที่ถูกส่งผ่านสื่อสารธารณะ โดยระบบต่างๆเหล่านั้นถูกพัฒนาขึ้นบนพื้นฐานของความประหยัด สะดวกต่อการใช้งาน มีขนาดเล็ก และมีหลักการที่เชื่อถือได้รองรับเพื่อให้เกิดความถูกต้องและปลอดภัยของข้อมูลสูงสุด ระบบการสื่อสารที่ใช้ระบบปกปิดข้อมูลในปัจจุบัน ได้แก่ ระบบสื่อสารของธนาคาร การสื่อสารข้อมูลของทางทหาร การส่งข้อมูลผ่านเส้นใยแก้วนำแสง การให้บริการเคเบิลทีวี เป็นต้น

1.2 วัตถุประสงค์ของการศึกษา

1) เพื่อศึกษาทฤษฎีและหลักการ Finite Field Fourier Transform และการออกแบบวงจรรวมบนชิพ FPGA

2) เพื่อออกแบบระบบปกปิดข้อมูลโดยใช้หลักการ Finite Field Fourier Transform เพื่อสร้างวงจรรวมบน FPGA

3) เพื่อทดลองสร้างระบบปกปิดข้อมูลโดยใช้หลักการ Finite Field Fourier Transform บน FPGA

4) ทดลองและพัฒนาระบบปกปิดข้อมูล โดยใช้หลักการ Finite Field Fourier Transform บน FPGA เพื่อสร้างต้นแบบที่ใกล้เคียงกับระบบที่ใช้งานได้จริง

1.3 สมมติฐานของการศึกษา

การพัฒนาระบบปกปิดข้อมูลในระบบดิจิตอลมีข้อดีกว่าระบบอนาล็อก เนื่องจากมีขนาดเล็กและให้คุณภาพของสัญญาณที่ดีกว่า อุปกรณ์โปรแกรมได้ที่มีความจุสูงและแก้ไขโปรแกรมได้ง่าย เช่น FPGA มีความเหมาะสมที่จะนำมาทดลองสร้างเป็นระบบปกปิดข้อมูล เพราะจะทำให้ประหยัดเวลาและค่าใช้จ่ายสำหรับตัวอุปกรณ์ในการประกอบวงจร

1.4 ทฤษฎีหรือแนวคิดที่ใช้ในการวิจัย

เมื่อข่าวสารเดินทางผ่านสื่อที่มีสัญญาณรบกวนสูง ข่าวสารที่รับได้ที่ปลายทาง อาจมีรูปแบบที่ต่างไปจากเดิม ทำให้ผู้รับไม่สามารถนำข้อมูลที่ถูกต้องไปใช้งานได้จริง การแก้รหัสที่ผิดได้มีส่วนช่วยในการขจัดข่าวสารที่ผิดอันเกิดจากสัญญาณรบกวนนั้น และส่งข่าวสารที่ถูกต้องออกมา

ระบบการสื่อสารในปัจจุบัน ได้พัฒนาขึ้นมาก จนทำให้มีความผิดพลาดของข่าวสารที่เกิดจากช่องสัญญาณน้อยลงมาก อีกทั้งอุปกรณ์ปลายทาง (Terminal) ในปัจจุบันได้รวมระบบการแก้รหัสที่ผิดไว้ในตัว ซึ่งถือเป็นระบบที่มีความถูกต้องของข้อมูลสูงมาก (Error free channel)

งานวิจัยนี้ ได้รวมข้อดีของการแก้รหัสที่ผิดและคุณสมบัติที่ปราศจากการผิดพลาดของข้อมูลของช่องสัญญาณในปัจจุบัน มาสร้างระบบปกปิดข้อมูล โดยทำการเพิ่มสัญญาณรบกวนเข้าไปที่ต้นทาง ส่งผ่านระบบสื่อสาร (ที่ไม่ทำให้ข่าวสารผิดไปจากเดิม) และขจัดสัญญาณรบกวนนั้นออกที่ปลายทาง ทำให้ผู้ประสงค์ร้าย ที่ดักจับข้อมูลในช่องสื่อสาร ได้รับข่าวสารที่ผิดไปเนื่องจากมีสัญญาณรบกวนรวมอยู่ด้วย และไม่สามารถนำไปใช้งานได้

1.5 ขอบเขตการวิจัย

งานวิจัยนี้ ต้องการพัฒนาและทดสอบระบบปกปิดข้อมูล โดยอาศัยหลักการ Finite Field Fourier Transform บนชิพ FPGA ซึ่งเดิมถูกสร้างอยู่บนไอซีแบบแยกส่วน เพื่อเปรียบเทียบความแตกต่าง ระหว่างข่าวสารที่ส่งผ่านช่องสัญญาณโดยตรง กับข่าวสารที่ผ่านการปกปิดข้อมูลและข่าวสารที่ได้รับการถอดรหัสที่ปกปิดแล้ว

1.6 เนื้อหาภายในวิทยานิพนธ์

เนื้อหาภายในวิทยานิพนธ์ฉบับนี้ ถูกแบ่งออกเป็นส่วนๆ เพื่อให้ผู้ที่สนใจมีความสะดวกแก่การศึกษาและทำความเข้าใจ

บทที่ 2
หลักการและทฤษฎี

2.1 การเข้ารหัสลับ

โครงข่ายการสื่อสารสาธารณะในปัจจุบันเช่น โครงข่ายโทรศัพท์พื้นฐาน (Public Switched Telephone Network) เป็นที่นิยมใช้งานกันมาก ด้วยเหตุผลต่างๆ หลายประการ อาทิ มีคุณภาพของเสียงพูดสูง ใช้งานง่าย อัตราค่าบริการต่ำ แต่ในทางกลับกัน ระบบสื่อสารสาธารณะ ขาดความปลอดภัยในการรักษาความลับของข้อมูล ซึ่งการดักฟังทำได้ง่าย วิทยานิพนธ์ฉบับนี้จึงศึกษาระบบเข้ารหัสลับที่จะนำมาใช้กับระบบการสื่อสารดังกล่าว

การสร้างระบบเข้ารหัสลับเสียงพูดหรือข้อมูลข่าวสาร เพื่อนำมาใช้ร่วมกับช่องสื่อสารที่มีอยู่ในปัจจุบัน หรือการสร้างเพื่อรวมเข้าไว้ในช่องการสื่อสารที่จะมีในอนาคตนั้น จะต้องพิจารณาถึงส่วนประกอบหลายประการด้วยกัน ได้แก่

- ขอบเขตความต้องการของผู้ใช้
- ขอบเขตของอุปกรณ์และวงจรที่มีอยู่ในปัจจุบัน
- ความสามารถของผู้ผลิตหรือผู้ออกแบบระบบเข้ารหัสลับ
- ความยากง่ายในการติดตั้งและโอกาสที่จะเกิดการทำงานที่ผิดพลาดของระบบ

2.1.1 ขอบเขตความต้องการของผู้ใช้

โดยทั่วไปมักจะมีความขัดแย้งระหว่างระดับของความปลอดภัย คุณภาพของสัญญาณเสียง และราคาของระบบ ผู้ใช้มักต้องการระบบที่มีความปลอดภัยของเสียงพูดหรือข่าวสารระดับสูง ในราคาที่ประหยัดซึ่งเป็นไปได้ค่อนข้างยาก สามารถจำแนกประเภทของความต้องการเหล่านั้นได้เป็นระดับของความปลอดภัย

ในการเข้ารหัสลับเพื่อรักษาความปลอดภัยสำหรับระบบสื่อสารในระดับต่างๆ เช่นระดับผู้นำประเทศ กิจการทหาร หรือใช้ส่วนบุคคล จะมีระดับความยากง่ายต่างกันออกไปในการสื่อสารส่วนบุคคล อาจใช้วิธีการเข้ารหัสลับแบบง่ายๆ ในขณะที่ผู้นำประเทศแล้วจะต้องใช้ ระบบที่แทบจะไม่มีวิธีการดักฟังได้เลย ในการออกแบบระบบเข้ารหัสลับเพื่อให้รองรับกับความต้องการดังกล่าว มีสิ่งที่จะต้องพิจารณา คือ

ผู้ดักฟัง

เป็นเรื่องยากที่จะบอกได้ว่าสามารถจะป้องกันการดักฟังจากผู้ใด เช่น ในกรณีกิจการทหารในสงคราม จะต้องมีการป้องกันการดักฟังจากข้าศึก ส่วนในกรณีของตำรวจ อาจใช้ป้องกันการดักฟังจากผู้กระทำผิดกฎหมายที่มีเครื่องมือสื่อสารชนิดเดียวกัน ซึ่งสามารถรู้ตัวก่อนและหลบเลี่ยงการ

จับกุมได้ทัน แต่ถ้าเป็นกรณีบุคคลทั่วไปที่ไม่ประสงค์ทำผิดกฎหมาย การดักฟังข่าวสารของตำรวจก็ถือเป็นเรื่องที่ไม่มีความผิดร้ายแรง

ความสามารถของผู้ดักฟัง

ระบบเข้ารหัสลับจะมีประสิทธิภาพเพียงใด ขึ้นอยู่กับผู้ออกแบบและผู้ดักฟัง ถ้าผู้ดักฟังเป็นผู้ที่ได้รับการศึกษาและฝึกฝนเป็นอย่างดี ไม่สามารถป้องกันการดักฟังได้ ตัวแปรสำคัญที่ทำให้การดักฟังทำได้สำเร็จมีอยู่หลายประการ เช่น ทรัพยากร อุปกรณ์ในการดักฟัง เวลา และปริมาณข้อมูลที่ผู้ดักฟังรับได้ (ผู้ดักฟังอาจบันทึกเสียงไว้เพื่อฟังหลายๆครั้ง จนจับประเด็นได้)

ปริมาณข่าวสารที่ผู้ดักฟังได้รับ

ระยะเวลาในการส่งข่าวสารอย่างต่อเนื่อง มีผลโดยตรงต่อระดับความปลอดภัยของระบบ ถ้าผู้ดักฟังได้รับข่าวสารต่อเนื่องเป็นเวลานานพอ อาจทำให้สามารถคิดหาวิธีการถอดรหัสลับได้

ระยะเวลาในการได้รับข่าวสารของผู้ดักฟัง

ระบบสื่อสารที่ไม่มีการเปลี่ยนแปลงรูปแบบของการเข้ารหัสลับ ทำให้เกิดข้อเสียคือ ผู้ดักฟังมีเวลาในการคิดหาวิธีการถอดรหัสลับ ข่าวสารจะปลอดภัยอยู่ตราบเท่าที่ผู้ดักฟังยังไม่รู้วิธีการเข้ารหัสลับ ดังนั้นผู้ใช้ระบบ ไม่ควรมั่นใจในความปลอดภัยของระบบมากเกินไป และต้องเปลี่ยนรูปแบบของการเข้ารหัสลับให้บ่อยที่สุดเท่าที่จะทำได้ เหตุผลสำคัญอีกประการคือ เมื่อผู้ใช้มั่นใจว่าระบบสื่อสารนั้นไม่มีการเข้ารหัส ก็มักจะพยายามใช้คำพูดที่เข้ารหัสเพื่อปกปิดข่าวสารกันเอง แต่ถ้าผู้ใช้มีความไว้วางใจว่าระบบนั้นถูกเข้ารหัสลับแล้ว ก็จะมีการส่งข่าวสารถึงกัน โดยเปิดเผย จึงเป็นจุดอ่อนสำคัญที่ทำให้ผู้ดักฟังสามารถถอดรหัสลับได้ในที่สุด

2.1.2 ประสิทธิภาพและคุณภาพของสัญญาณ

ระบบการสื่อสารสาธารณะส่วนใหญ่ เป็นช่องสัญญาณที่มีความปลอดภัยของข้อมูลต่ำ หากมีการเพิ่มความปลอดภัยให้กับช่องสัญญาณด้วยการติดตั้งระบบเข้ารหัสลับ จะทำให้คุณภาพของสัญญาณลดต่ำลง ในทางปฏิบัติไม่สามารถบอก ได้ว่าตัวเข้ารหัสแต่ละตัวมีประสิทธิภาพและคุณภาพของสัญญาณดีหรือไม่ดี เนื่องจากความปลอดภัยของข้อมูลในระบบที่เข้ารหัสลับจะเป็นส่วนกลับกับคุณภาพของสัญญาณเสมอ

ในการสื่อสารผ่านช่องสัญญาณที่ไม่มีการเข้ารหัสลับ ผู้ใช้จะคิดหาวิธีการเข้ารหัสคำพูดของตนเอง เช่น การใช้รหัสวิทยุสื่อสาร ผู้รับจะได้ยินเสียงที่ชัดเจน และถอดรหัสตามความหมายที่ได้ตกลงกันไว้ล่วงหน้า แต่ในกรณีที่ติดตั้งตัวเข้ารหัสลับ คุณภาพของสัญญาณจะลดลงจนทำให้ไม่สามารถสื่อสารกันได้เข้าใจ ประสิทธิภาพในการสื่อสารจะขึ้นอยู่กับคุณภาพของสัญญาณเสียง

คุณภาพของช่องการสื่อสาร และการรู้จำของผู้ฟัง โดยทั่วไปแล้ว การเพิ่มความซับซ้อนของตัวเข้ารหัสลับ จะทำให้คุณภาพของสัญญาณเสียงที่ได้ลดลง

2.1.3 การใช้งานอุปกรณ์สื่อสารที่ติดตั้งระบบเข้ารหัสลับ

เหตุผลที่ใช้พิจารณาระบบหรืออุปกรณ์สื่อสารที่ต้องการเข้ารหัสลับ ได้แก่

1) ระบบนั้นถูกนำไปใช้ทางยุทธวิธีหรือในสถานะสงครามหรือไม่ ในการใช้งานระบบสื่อสารส่วนบุคคลอาจไม่ต้องการรักษาความลับของข้อมูล ในขณะที่การติดต่อสื่อสารในสนามรบต้องการความปลอดภัยของข่าวสารที่สูงมาก ซึ่งระบบที่นำมาใช้จะต้องคำนึงถึง

ความลับของอุปกรณ์หรือตัววงจร ในกรณีที่เครื่องมือสื่อสารถูกจับกุมได้โดยฝ่ายตรงข้าม ฝ่ายตรงข้ามสามารถศึกษากลวิธีในการเข้ารหัสลับและสามารถถอดรหัสลับได้ทั้งเครือข่ายการสื่อสาร

-น้ำหนัก

-ความทนทาน

-ความยากง่ายในการใช้งาน

-ความสะดวกในการเชื่อมต่ออุปกรณ์เข้ารหัสลับกับเครื่องรับ-ส่ง

-การบำรุงรักษา

2) การนำอุปกรณ์เข้ารหัสลับไปใช้งาน ในบริเวณที่มีสัญญาณรบกวนสูง อาจทำให้ไม่สามารถถอดรหัสลับได้

3) ในแต่ละเครื่องส่ง อาจมีตัวเข้ารหัสลับได้เป็นจำนวนมาก เพื่อประสิทธิภาพในการเข้ารหัสลับ หรือในกรณีที่มีการสื่อสารหลายช่อง (multi hop) อาจจำเป็นต้องเข้ารหัสลับในแต่ละช่วงให้แตกต่างกัน

4) การใช้กุญแจรหัสลับแบบปลายทางถึงปลายทาง (End-to-End) หรือแบบเป็นช่วงๆ (Link by Link) อาจใช้วิธีการเข้ารหัสลับได้ 2 ลักษณะ คือ จะใช้การเข้ารหัสลับด้วยกุญแจแบบปลายทางถึงปลายทาง เมื่อผู้ใช้ทั้งสองด้าน ไม่ไว้วางในวงจรสื่อสารช่วงกลางหรือต้องการปกปิดกุญแจรหัสแต่ถ้าผู้ใช้ทั้งหมดในระบบมีความไว้วาใจกัน หรือมีกุญแจรหัสจำนวนมากพอ ก็อาจใช้ระบบรหัสกุญแจสาธารณะได้

5) ปริมาณการสื่อสารในวงจร

6) ระบบสื่อสารเป็นแบบทางเดียว (Simplex) สองทาง (Duplex) หรือสองทางในเวลาเดียวกัน (Full duplex)

2.1.4 การบริการและสนับสนุนการใช้งาน

ในการใช้งานระบบเข้ารหัสลับให้มีประสิทธิภาพ มีองค์ประกอบต่างๆ ที่ต้องพิจารณา เช่น ความรู้ทางด้านคณิตศาสตร์และวิศวกรรมเกี่ยวกับระบบเข้ารหัสลับ วิธีการติดตั้ง การซ่อมและบำรุงรักษา การอบรมการใช้งาน เอกสารประกอบการใช้และวิธีการเปลี่ยนกุญแจรหัส

โดยทั่วไป ระบบเข้ารหัสลับจะถูกออกแบบมาเป็นอย่างดี โดยผู้เชี่ยวชาญด้านการเข้ารหัส สิ่งที่ต้องพิจารณาคือ การติดตั้ง เนื่องจากผู้ใช้ส่วนใหญ่ต้องการให้ติดตั้งในบริเวณที่เป็นความลับจริงๆ จึงอาจส่งผลถึงการซ่อมบำรุงรักษา ตลอดจนกรณีที่อุปกรณ์ในระบบอาจต้องถูกซ่อม หรือถอดเปลี่ยนโดยผู้ที่ไม่มีความรู้ด้านการซ่อมบำรุง เช่น ในสนามรบ เป็นต้น

ระบบเข้ารหัสส่วนมากจะใช้กุญแจรหัส ซึ่งเป็นหัวใจสำคัญของระบบ ผู้ใช้จะต้องจดจำและดูแลรักษากุญแจรหัส รวมถึงการสร้าง การแจกจ่ายและการบริหารกุญแจรหัส และต้องระลึกเสมอว่าผู้ดักฟังก็อาจมีกุญแจรหัสที่ใช้งานได้และกำลังใช้งานอยู่ด้วย

เนื่องจากกุญแจรหัสเป็นสิ่งที่สำคัญ จึงต้องพิจารณาถึงความปลอดภัยในการสร้าง และการใช้งานกุญแจรหัส

การสร้างกุญแจรหัส

ปัญหาประการแรกที่พบคือ ผู้ใช้จะกำหนดกุญแจรหัสได้อย่างไร โดยปกติในระบบที่เข้ารหัสลับ จะอนุญาตให้ผู้ใช้กำหนดรหัสกุญแจเอง แต่ผู้ใช้ส่วนใหญ่มักตั้งรหัสกุญแจตามชื่อของสิ่งของ หรือบุคคลที่มีความสำคัญสำหรับตน ทำให้ผู้ดักฟังสามารถคาดเดาได้โดยง่าย อีกวิธีหนึ่งที่ใช้กันก็คือ การสร้างกุญแจรหัสโดยวิธีการมาตรฐาน ซึ่งมีอันตรายในกรณีที่ผู้ดักฟังรู้กรรมวิธีในการสร้างกุญแจรหัส อันจะทำให้เขารู้กุญแจรหัสของทั้งระบบเลยทีเดียว

วิธีที่ดีในการสร้างกุญแจรหัส คือการกำเนิดรหัสกุญแจแบบสุ่ม ซึ่งทำให้ผู้ดักฟังสามารถคาดเดารูปแบบของรหัสกุญแจได้ยากขึ้น ดังนั้น ความปลอดภัยของระบบจึงขึ้นอยู่กับการทำให้ผู้ดักฟังไม่สามารถทราบกุญแจรหัสได้นั่นเอง

การแจกจ่ายกุญแจรหัส

เมื่อมีการสร้างกุญแจรหัส จะต้องแจกจ่ายไปยังผู้ใช้ อาจส่งโดยบุคคลหรือส่งผ่านทางระบบสื่อสาร ถ้าระบบสื่อสารมีขนาดใหญ่มาก หรือมีการเปลี่ยนแปลงกุญแจรหัสบ่อยครั้ง ก็จะทำให้เกิดปัญหายุ่งยากไม่น้อย โดยเฉพาะอย่างยิ่งต้องมั่นว่ากุญแจรหัสส่งไปถูกที่และถูกเวลา ในทางปฏิบัตินั้น ไม่อาจคาดหวังได้ว่าผู้ใช้จะมีความรู้เป็นอย่างดีในการใช้งานระบบสื่อสารที่มีการเข้ารหัสลับ จึงทำให้เกิดข้อผิดพลาดจากผู้ใช้ (human error) ขึ้นบ่อยครั้ง และเป็นจุดอ่อนที่ทำให้ผู้

ดักฟังสามารถถอดรหัสลับได้ การออกแบบระบบและวิธีการสร้างกุญแจรหัสเป็นงานที่ทำได้ยากมาก การแจกจ่ายรหัสกุญแจเป็นงานที่มีต้นทุนสูง ต้องใช้บุคลากรหรือทรัพยากรในระบบสื่อสารจำนวนมาก ต้องเชื่อมั่นและไว้วางใจในตัวสื่อนำกุญแจรหัสไปแจกจ่ายด้วย ซึ่งอาจแก้ปัญหาได้โดยการเข้ารหัสลับให้กับรหัสกุญแจนั้นอีกชั้นหนึ่ง

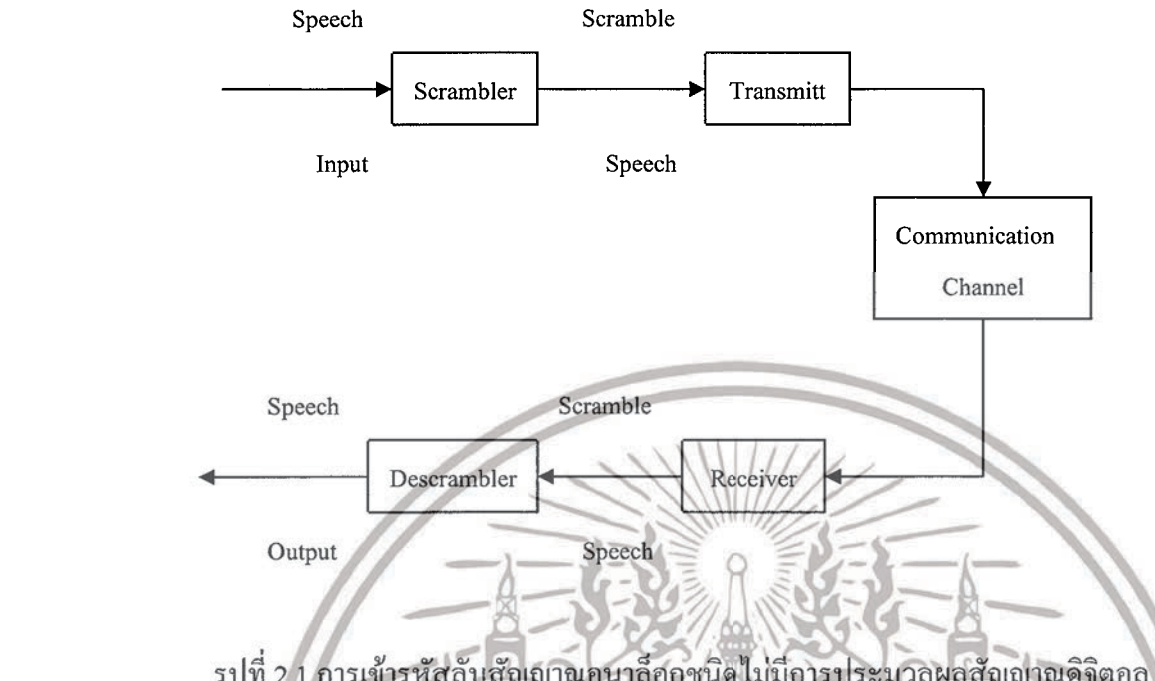
การบริหารกุญแจรหัส

การบริหารกุญแจรหัส หมายถึง กระบวนการในการเก็บรักษากุญแจรหัสทั้งหมดในระบบและต้องสามารถแจกจ่ายในเวลาที่ผู้ใช้ต้องการได้ โดยมีความปลอดภัยสูง รวมทั้งการสร้างกุญแจรหัสขึ้นใหม่ ในกรณีที่ของเดิมถูกทำลาย หรือคาดว่ากุญแจจะถูกขโมยจากผู้ดักฟัง

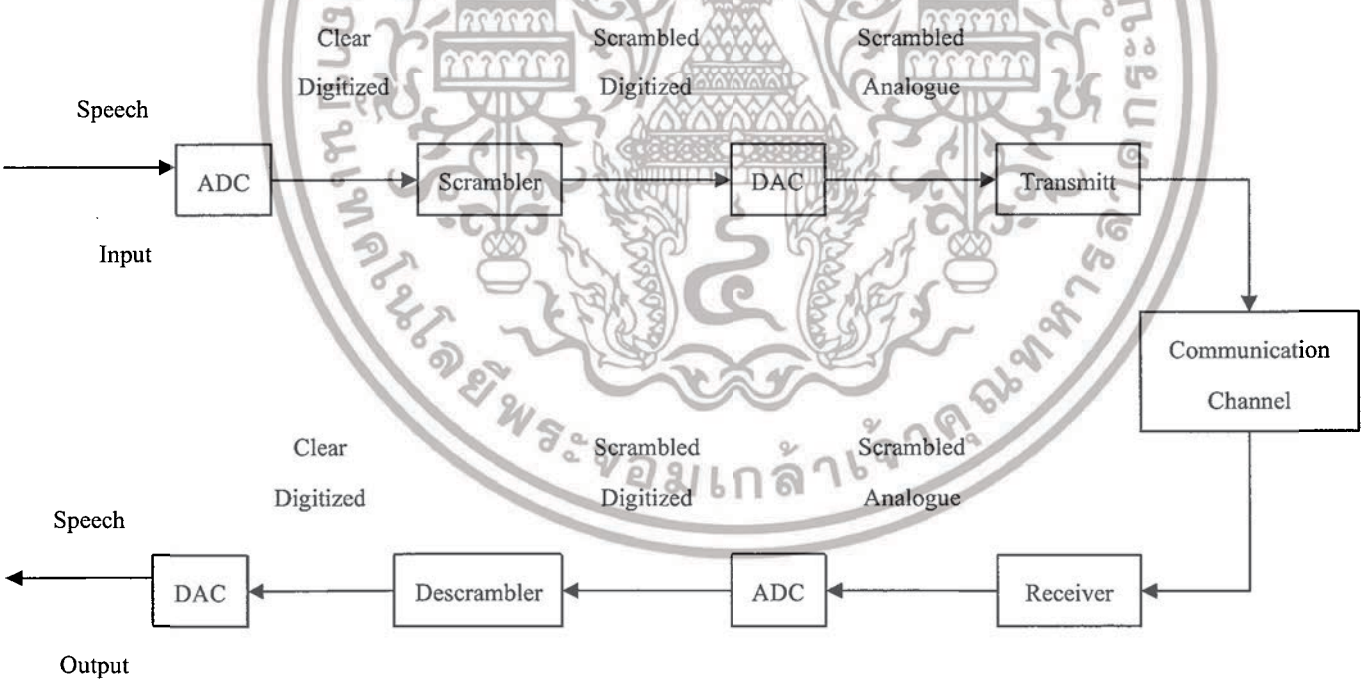
2.2 รูปแบบการเข้ารหัสลับ

ในการพิจารณาสร้างหรือนำระบบเข้ารหัสลับมาใช้งาน จะต้องพิจารณาถึงอุปกรณ์ที่จะนำมาสร้างเป็นระบบ โดยปกติจะพูดถึงตัวเข้ารหัสลับใน 2 ระบบใหญ่ๆ คือ ระบบอนาล็อกและระบบดิจิตอล ในปัจจุบันมีการนำการประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) มาใช้อย่างกว้างขวาง โดยอาศัยการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล ทำให้สามารถเปลี่ยนแปลงรูปแบบในการเข้ารหัสได้ง่าย ส่วนวงจรเข้ารหัสแบบอนาล็อกนั้น นอกจากจะออกแบบวงจรยุ่งยากซับซ้อนแล้ว ยังมีระดับความปลอดภัยต่ำ

ข้อแตกต่างที่เห็นได้ชัดระหว่างการเข้ารหัสลับแบบดิจิตอลและแบบอนาล็อกคือ ตัวส่งสัญญาณที่เข้ารหัสลับแล้ว ในวงจรอนาล็อก จะต้องส่งสัญญาณอย่างต่อเนื่อง ส่วนในวงจรดิจิตอลจะส่งสัญญาณได้จำนวนจำกัด ขึ้นอยู่กับปริมาณข่าวสารที่บรรจุได้ในรหัสดิจิตอลเท่านั้น โดยรูปที่ 2.1 ถึง 2.3 แสดงให้เห็นถึงผังวงจรของการเข้ารหัสลับแบบต่างๆ



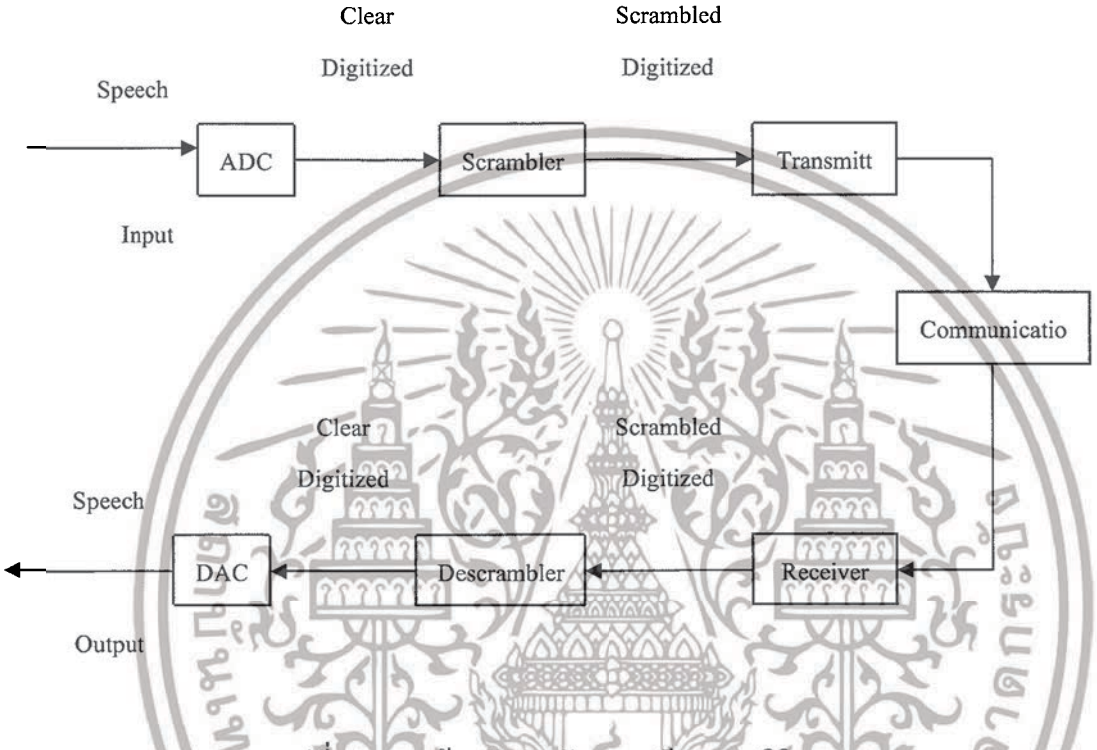
รูปที่ 2.1 การเข้ารหัสลับสัญญาณอนาล็อกชนิดไม่มีการประมวลผลสัญญาณดิจิตอล



รูปที่ 2.2 การเข้ารหัสสัญญาณอนาล็อกชนิดมีการประมวลผลสัญญาณดิจิตอล

รูปที่ 2.1 และ 2.2 เป็นการเข้ารหัสลับแบบอนาล็อก ความแตกต่างระหว่างรูปทั้งสอง คือ รูปแบบของสัญญาณที่ทำการเข้ารหัส ในรูปที่ 2.1 สัญญาณจะเป็นอนาล็อกตลอดกระบวนการส่วน

ระบบในรูปที่ 2.2 สัญญาณจะถูกเปลี่ยนให้อยู่ในรูปแบบของดิจิตอลก่อนเข้ารหัสลับ แล้วจึงแปลงกลับเป็นอนาล็อก เพื่อส่งในช่องสัญญาณที่เป็นอนาล็อก หลังจากนั้นจึงย้อนกระบวนการโดยแปลงสัญญาณที่รับเข้ามาเป็นดิจิตอล ทำการถอดรหัสลับในรูปแบบดิจิตอล แล้วจึงแปลงเป็นสัญญาณเสียงพูดอนาล็อกอีกครั้ง



รูปที่ 2.3 การเข้ารหัสลับสัญญาณเสียงแบบดิจิตอล

ผังวงจรในรูปที่ 2.3 เป็นรูปแบบที่ใช้ในวิทยานิพนธ์ฉบับนี้ จัดเป็นการเข้ารหัสลับแบบดิจิตอลโดยเสียงพูดจะถูกแปลง ให้อยู่ในรูปแบบดิจิตอล หลังจากนั้นจะถูกเข้ารหัสลับและส่งในรูปแบบของดิจิตอลไปยังภาครับ ที่ภาครับจะทำการถอดรหัสลับที่รับเข้ามาได้ แล้วจึงแปลงกลับเป็นสัญญาณอนาล็อกในขั้นสุดท้าย ระบบนี้จัดเป็นระบบที่ง่ายที่สุดในสามรูปแบบ จากที่ยกตัวอย่างมานั้น การพิจารณานำระบบใดไปใช้นั้นขึ้นอยู่กับว่าเสียงพูดที่ส่งในช่องการสื่อสารมีความสำคัญมากน้อยเพียงใด และช่องสัญญาณที่ใช้อยู่เป็นชนิดใด

2.3 ระบบเข้ารหัสสัญญาณเสียง (Audio Encryption System)

การเข้ารหัสเสียงพูด (Encryption Speech) แบ่งตามเทคนิคการสร้างได้ 2 วิธี คือ ทางด้านอนาล็อก (Analog) และทางด้านดิจิตอล (Digital)

2.3.1 การเข้ารหัสเสียงพูดด้วยวิธีทางอนาล็อก (Analog Encrypton Speech)

การเข้ารหัสเสียงพูดด้วยวิธีทางอนาล็อก (Analog Encryption Speech) มีหลายระบบด้วยกัน

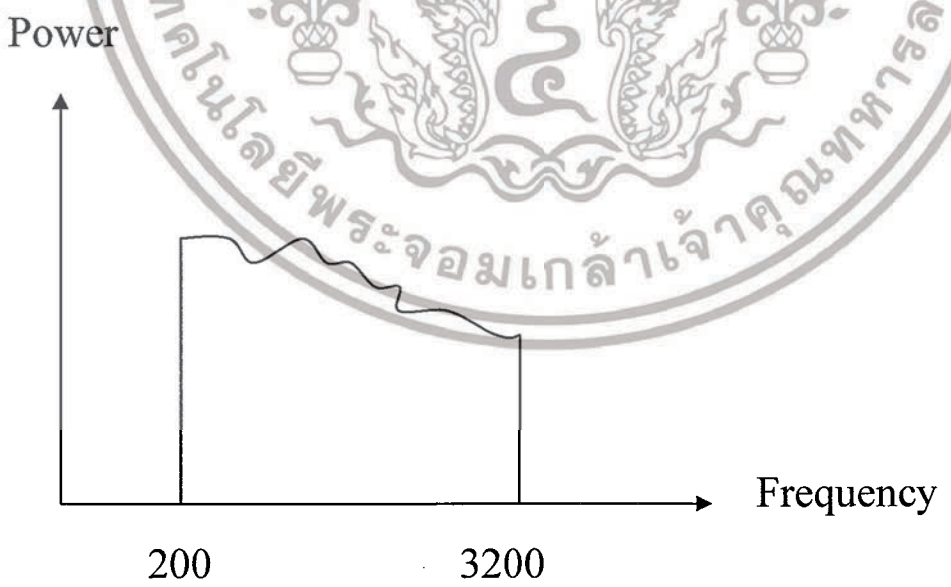
- Speech Inversion
- Band Shift Inversion
- Band Scramble or Band Splitter
- Time Element Scrambler

1) การเข้ารหัสเสียงพูดด้วยวิธี Speech Invarsion

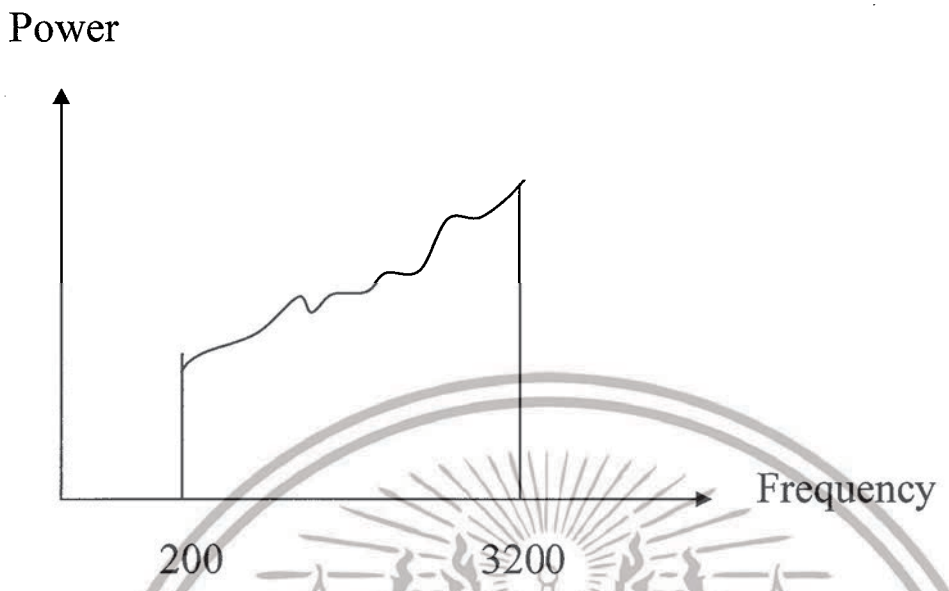
Speech Inversion เป็นการสแกรมบลิ่ง (ScramBling) โดยใช้การเปลี่ยน (Invert) ย่านความถี่ทั้งหมดจากย่านความถี่สูงให้กลายเป็นความถี่ต่ำ และจากย่านความถี่ต่ำให้กลายเป็นย่านความถี่สูง

การสแกรมบลิ่งแบบนี้เป็นวิธีที่ง่าย แต่ไม่ปลอดภัยมากนัก เพราะสามารถที่จะดิสแกรมบลิ่ง (Discrambing) ได้ง่ายเช่นกัน โดยการรีอินเวิร์ท (Reinvert) แบบลองผิดลองถูก (Trial and Error) ก็จะได้สัญญาณเดิมกลับมา

การเพิ่ม Conbitional ของการสแกรมบลิ่ง ทำได้โดยการใช้ความถี่พาหะ (Carrier Frequency) ในการ Invert ต่างกัน



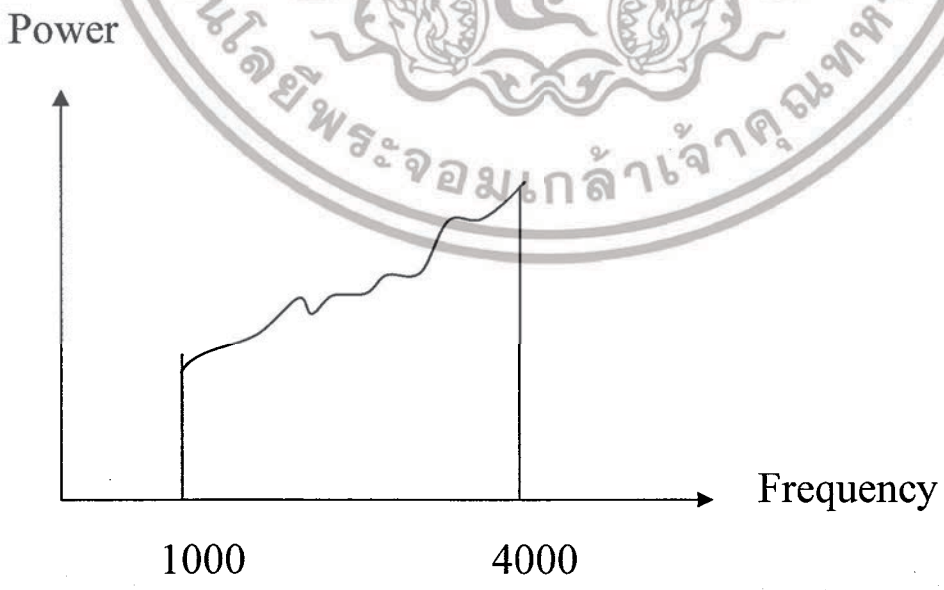
รูปที่ 2.4 แสดง Power Spectrum Density ของสัญญาณเสียงก่อนการสแกรมบลิ่ง



รูปที่ 2.5 แสดง Power Spectrum Density ของสัญญาณเสียงหลังการสแกรมบลิ่ง

2) การเข้ารหัสเสียงพูดด้วยวิธี Band-Shift Inversion

วิธีนี้เป็นการปรับปรุง Speech Inversion โดยการเปลี่ยน (Invert) ย่านความถี่ทั้งหมาด จากความสูงให้กลายเป็นความถี่ต่ำ และจากความถี่ต่ำให้กลายเป็นความถี่สูง จากนั้นก็จะเลื่อน (Shift) ย่านความถี่บางช่วงจากตำแหน่งเดิมที่หลายด้านหนึ่ง ไปไว้ที่ตำแหน่งที่ปลายอีกด้านหนึ่ง โดย Bandwidth ทั้งหมดของสัญญาณจะยังคงเท่าเดิม



รูปที่ 2.6 แสดงหลักการเข้ารหัสเสียงพดด้วยวิธี Band Shift Inversion

เนื่องจากวิธีนี้ปรับปรุงมาจาก Speech Inversion ดังนั้น Combitional ที่เป็นไปได้ยังมีจำนวนจำกัด วิธีที่จะเพิ่ม Combitional อาจทำได้โดยใช้ Pseudo Random Generator เป็นตัวเลือกตำแหน่งการเลื่อนที่แตกต่างกัน และแต่ละแบบให้มีช่วงเวลา (Time Interval) ประมาณ 10-20 ms. หลังจัดลำดับให้สลับเปลี่ยนกันไปในลักษณะเป็นวงกลม (Cyclical) หลักการนี้เราเรียกว่า Cyclical Band Shift Inversion ทำให้ Combitional ที่เป็นไปได้มากขึ้น

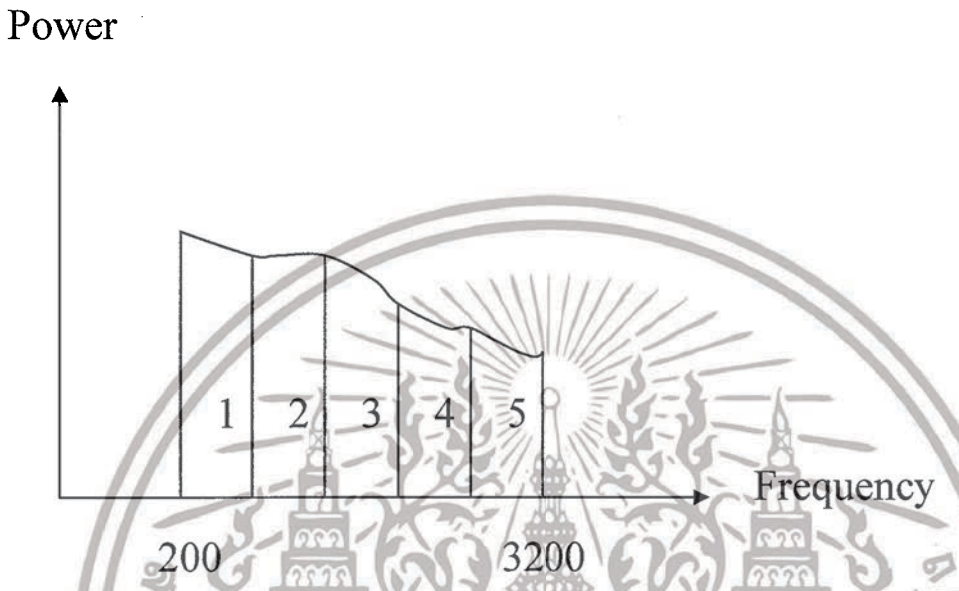
ข้อเสียของระบบเข้ารหัสเสียงพูดด้วยวิธี Band-Shift Inversion มีอยู่ 2 ข้อ คือ ข้อแรกเนื่องจาก Combitional ที่เป็นไปได้ยังไม่มากพอ ทำให้การดีสแกรมบลิ่งยังทำได้โดยง่ายด้วยวิธีลองผิดลองถูก (Trial and Error) ข้อที่สอง คือ จะยังมีบางส่วนของสัญญาณที่ถูกสแกรมบลิ่งแล้ว แต่จะถูกเปลี่ยนแปลงไปไม่มากนัก ทำให้ผู้ที่ชำนาญและคุ้นเคยกับเสียงที่ถูกสแกรมบลิ่งแล้ว สามารถที่จะเอาข้อมูลออกได้

3) การเข้ารหัสเสียงพูดด้วยวิธี Band Scramble or Band Spitter

วิธีการนี้แบนด์วิดธ์ของสัญญาณเสียงพูด (Speech Bandwidth) จะถูกแบ่งออกเป็นส่วนๆ เท่ากัน ซึ่งแต่ละส่วนเรียกว่า แบนด์ย่อย (Sub Band) แต่ละแบนด์ย่อยจะถูกสแกรมบลิ่ง โดยการสลับลำดับ (Permutation) แบนด์ย่อยเสียงใหม่ ในบางระบบอาจมีการ Invert แบนด์ย่อยด้วย

รูปที่ 2.7 แสดงตัวอย่างของ Band Scramble แบบง่าย โดยแบ่งออกเป็น 5 แบนด์ย่อย จากรูปแบนด์ย่อยที่ 1, 2 และ 5 ถูก Invert และถูกสลับตำแหน่ง สำหรับตัวอย่างดังกล่าวมีการจัดลำดับที่เป็นไปได้ (Possible Reordering) เท่ากับ 5! และมี Combitional สำหรับการ Reinvert เท่ากับ 2^5 นั่นหมายถึงว่าสามารถจะมี Combitional ได้มากถึง 5! X 2^5 เท่ากับ 3840 แบบ แต่ไม่ใช่ทั้งหมดที่สามารถนำไปใช้ได้ มีบางส่วนที่สัญญาณหลังจากการสแกรมบลิ่งแล้วที่มีการเปลี่ยนแปลงสมบูรณ์ (คือไม่สามารถเข้าใจรายละเอียดได้) ส่วนที่เหลือซึ่งเป็นส่วนใหญ่ ไม่สามารถนำมาใช้งานได้ เพราะยังสามารถที่จะเข้าใจความหมายได้แบบคลุมเครือสาเหตุที่เป็นเช่นนั้นก็เพราะว่า 40% ของ Power Spectrum Density ส่วนใหญ่ของเสียงพูดของคนเราจะอยู่ที่ 2 แบนด์ย่อยแรก (200-170 Hz) นั่นหมายความว่ามีเพียง 2 แบนด์ย่อยแรกก็สามารถพอที่จะเข้าใจความหมายได้ วิธีแก้ก็คือต้องแบ่ง 2 แบนด์ย่อยแรกให้มากขึ้น แล้วใช้ Pseudo Random Generator เป็นตัวกำหนดการจัดลำดับที่แตกต่างกันทุกๆ 100-200 ms. โดยการเรียกใช้จากการ Addressing ที่ตำแหน่งใดตำแหน่งหนึ่ง แต่การทำวิธีนี้จะมีข้อจำกัดคือ การเพิ่มจำนวนแบนด์ย่อยมากๆ อาจจะดูเหมือนเป็นการเพิ่มจำนวนการจัดลำดับและความปลอดภัยของข่าวสาร แต่ถ้ามากเกินไปจะทำให้ยุ่งยากในทางปฏิบัติ เพราะการเพิ่มจำนวนแบนด์ย่อยจะต้องใช้ฟิลเตอร์และส่วนประกอบอื่นๆ เพิ่มเติม ทำให้สัญญาณรบกวนใน

ระบบมีมากขึ้น ทำให้การปรับปรุงแก้ไข (Modification) การสแกรมบลิ่งมากเกินไปไม่ได้ทำให้คุณภาพเสียงดีขึ้น

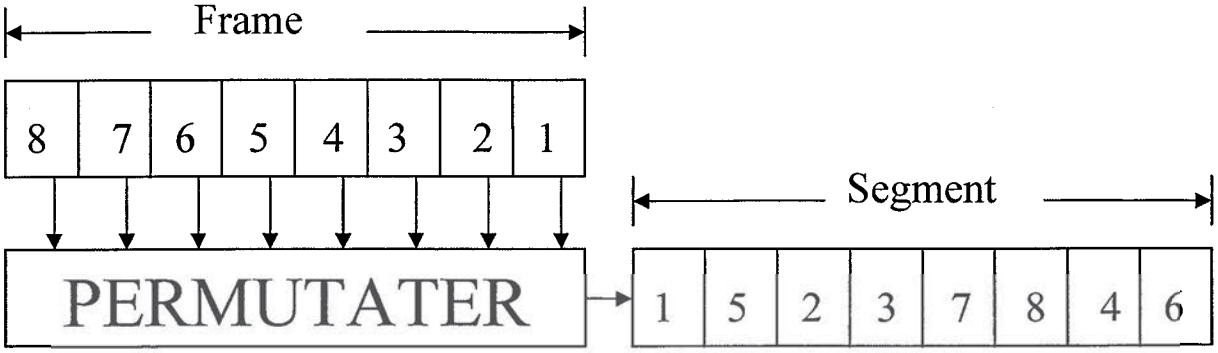


รูปที่ 2.7 แสดงถึงหลักการเข้ารหัสเสียงพูดด้วยวิธี Band Scramble

4) การเข้ารหัสสัญญาณเสียงแบบ Time Element Scrambler (T.E.S.)

การทำงานของ Time Element Scrambler (T.E.S.) อาศัยหลักการพื้นฐาน โดยขั้นแรกแบ่งสัญญาณอนาล็อกเป็นคาบเวลา (Time period) เท่าๆ กัน โดยแต่ละส่วนเรียกว่า เฟรม แล้วแต่ละเฟรมจะถูกแบ่งย่อยออกเป็นคาบเวลาเล็กๆ เรียกว่า เซ็กเม้นท์ (Segment) และในทุกๆ เฟรมของอินพุทจะสแกรมบลิ่งเซ็กเม้นท์เหล่านั้นด้วยวิธี การสลับลำดับ (Permutation)

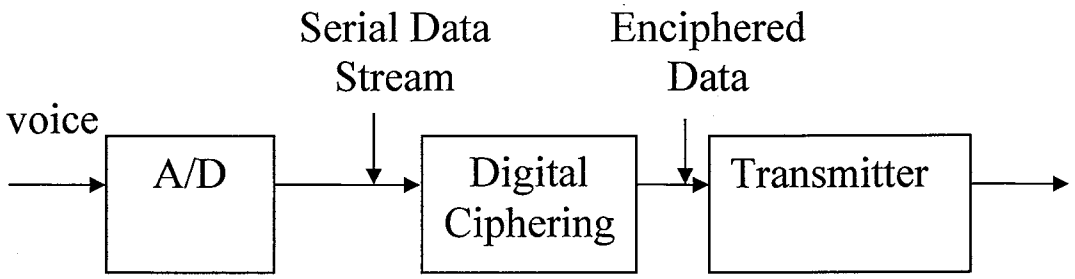
วิธีการดังกล่าวอธิบายด้วยแผนการทำงาน (Block diagram) ดังแสดงในรูปที่ 1.5 ซึ่งในที่นี้แต่ละเฟรมจะถูกแบ่งออกเป็น 8 เซ็กเม้นท์

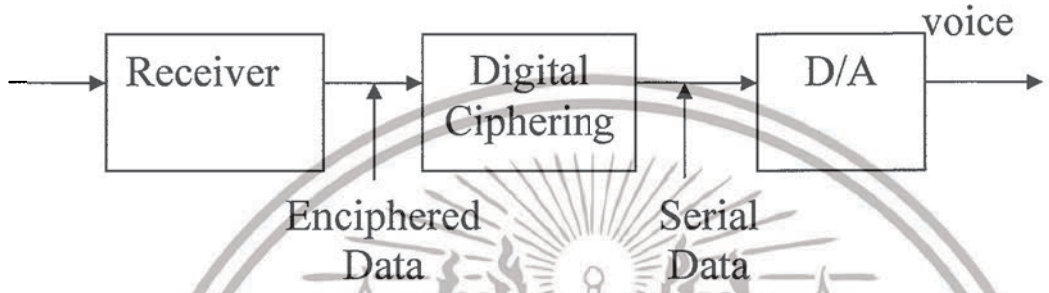


รูปที่ 2.8 Time Element Scrambler

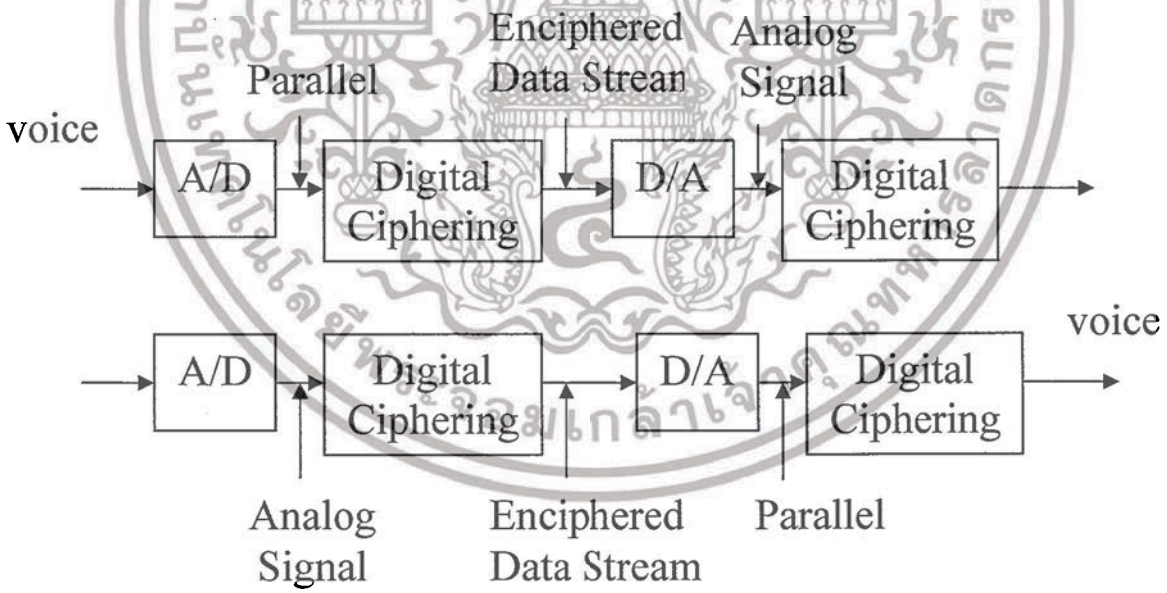
2.3.2 การเข้ารหัสเสียงพูดด้วยวิธีทางดิจิตอล (Digital Encryption Speech)

การเข้ารหัสเสียงพูดด้วยวิธีทางดิจิตอล เป็นอีกวิธีหนึ่งที่ได้ประสิทธิภาพในการป้องกันข้อมูลอย่างสูงแบบหนึ่ง จากรูปที่ 2.6 เสียงที่ถูกเปลี่ยน (Convert) เป็นสัญญาณดิจิตอล จะอยู่ในรูปของ Serial Data Stream ซึ่งอาจจะเป็น 64 Kbps, 4.8 Kbps หรือ 2.4 Kbps ของอัตราความเร็วของบิท (bit rate) ขนาด 9.6 Kbps และที่สูงกว่าจะเป็นการเพิ่มแบนด์วิดธ์ของสัญญาณ ทำให้เพิ่มความยุ่งยากในการนำไปใช้งาน โดยจะต้องพิจารณาถึงระบบเชื่อมโยงการส่งผ่าน (Transmission link) ที่สามารถตอบสนองต่อสัญญาณที่ได้ใช้สำหรับอัตราความเร็วของบิทที่ต่ำกว่า 9.6 Kbps สามารถที่จะนำไปใช้ได้จริง แต่ทั้งนี้ประสิทธิภาพสัมพันธ์กับการลดรูปของการจำรูปแบบของเสียง (Reduction in Voice Recognition) ซึ่งมีขบวนการที่ยุ่งยากและซับซ้อน ต้องใช้อุปกรณ์จำนวนมากและมีราคาแพง





รูปที่ 2.9 แสดงการเข้ารหัสเสียงพูดด้วยวิธีทางดิจิตอล



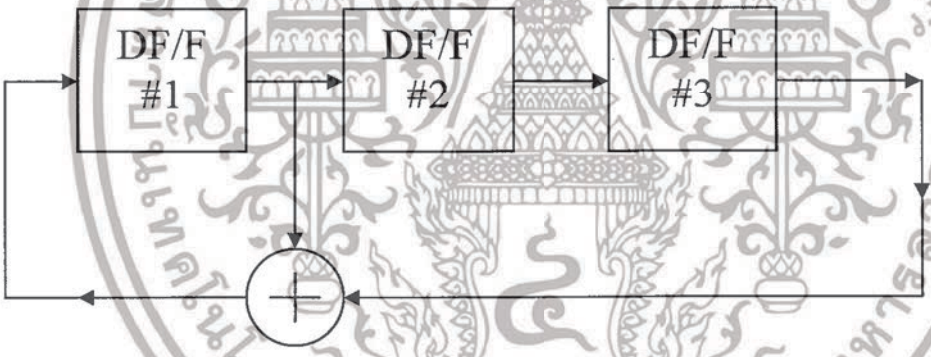
รูปที่ 2.10 แสดงการดัดแปลงการเข้ารหัสเสียงพูดด้วยวิธีทางดิจิตอลให้ดีขึ้น

1) การสแกรมเบลอและดีสแกรมเบลอ (Scrambler and Descrambler)

หลักการสแกรมเบลอและการดีสแกรมเบลอ

จากหลักการทำงานของข้อมูลสำหรับลำดับแบบกึ่งสุ่ม (Pseudo Random Sequences) ซึ่งสามารถสร้างได้โดยใช้ชิพรีจิสเตอร์ (Shift register) ต่อให้มีการป้อนกลับ (feedback) แบบ Modulo 2 adder ชิพรีจิสเตอร์นี้ประกอบขึ้นจากฟลิบฟลอบต่ออนุกรมกัน เมื่อชิพรีจิสเตอร์ได้รับ clock สภาวะของฟลิบฟลอบแต่ละตัวจะถูกส่งไปที่ฟลิบฟลอบตัวถัดไป สัญญาณที่ถูก tap ออกมาจะผ่าน Modulo 2 adder และป้อนกลับไปที่ฟลิบฟลอบตัวแรก

จากรูปที่ 2.11 พิจารณาที่ตัวกำเนิด 3 Sequence สมมุติให้สภาวะเริ่มแรกของฟลิบฟลอบเป็น "1" ทั้งหมด เมื่อมี clock เข้ามา ข้อมูลของฟลิบฟลอบตัวที่ 1 และ 3 จะถูกรวมกัน ข้อมูลของฟลิบฟลอบตัวที่ 1 และ 2 จะถูกเลื่อนไปที่ตัวที่ 2 และ 3 เอาท์พุทที่ Mod 2 จะถูกส่งกลับไปที่ฟลิบฟลอบตัวที่ 1 ข้อมูลของชิพรีจิสเตอร์จะวนเป็นรอบ มีสภาวะต่างกัน 7 สภาวะ แล้วจึงจะวนกลับไปซ้ำที่เดิม เอาท์พุทของ three state sequence นี้จะเริ่มจากฟลิบฟลอบตัวสุดท้ายของรีจิสเตอร์จะผลิตลำดับ 110100 ลำดับ 7 บิทนี้จะเป็นแบบสุ่ม



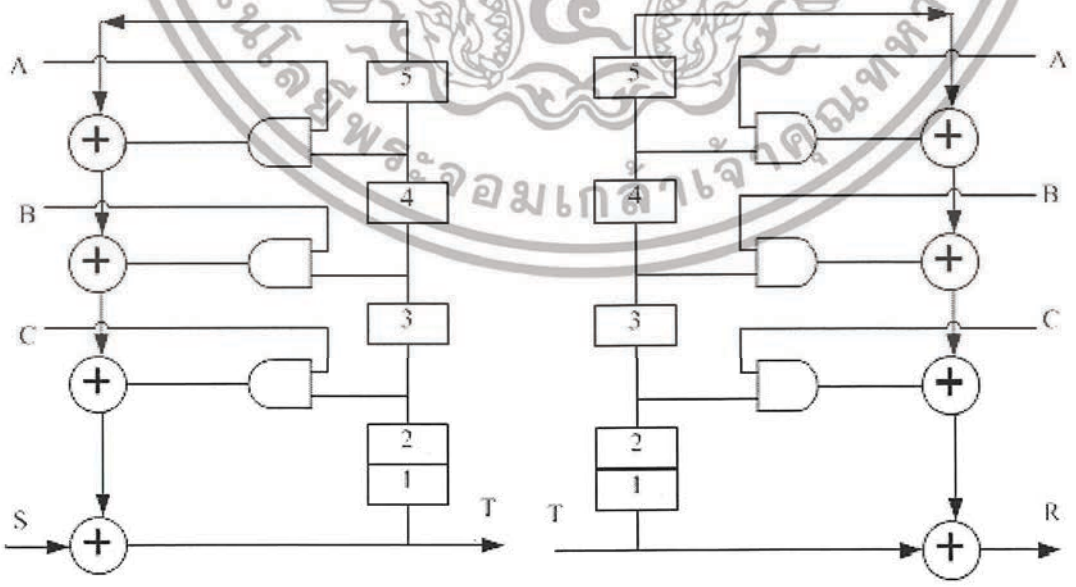
รูปที่ 2.11 ชิพรีจิสเตอร์ 3 ภาค ต่อให้มีการป้อนกลับ

ความยาวของ Pseudo random นี้จะคำนวณได้จากจำนวนของชิพรีจิสเตอร์ feedback tap และสภาพเริ่มแรกของฟลิบฟลอบ สังเกตจากรูปที่ 2.9 สภาวะเริ่มแรกที่เป็น "0" ทั้งหมดจะไม่ใช้ในการกำเนิดลำดับนี้ค่าความยาวของลำดับสูงสุดหาได้จาก n-state shift register คือ 2 x n-1 ตัวอย่างของค่าลำดับดับสูงสุดตามตารางที่ 2.1 สำหรับจำนวนชิพรีจิสเตอร์ต่างๆ

จำนวน shift register	Feed back TAP	Period of sequence
3	1,3	7
4	1,4	15
5	2,5	31
6	1,6	63
7	1,7	127
8	1,6,7,8	255
9	4,9	511
10	3,1,0	1023
11	2,1,1	2047

ตารางที่ 2.1 ความยาวสูงสุดของ Pseudo Random Sequence

Pseudo random จะนำมาใช้เป็นสแกรมเบลอข้อมูล โดย Modulo 2 adders ของข้อมูลกับลำดับ Pseudo random ตามรูปที่ 2.8 เป็น block diagram ของสแกรมเบลอและดิสแกรมเบลอ ซึ่งจะใช้ prescribed pseudo random (PR) sequence สังเกตว่า PR sequence generator ที่ใช้จะเหมือนกันทั้งสแกรมเบลอและดิสแกรมเบลอ ซึ่งสแกรมเบลอจะใช้ป้อนกลับ ส่วนดีสแกรมเบลอจะใช้ป้อนตาม (feed forword)

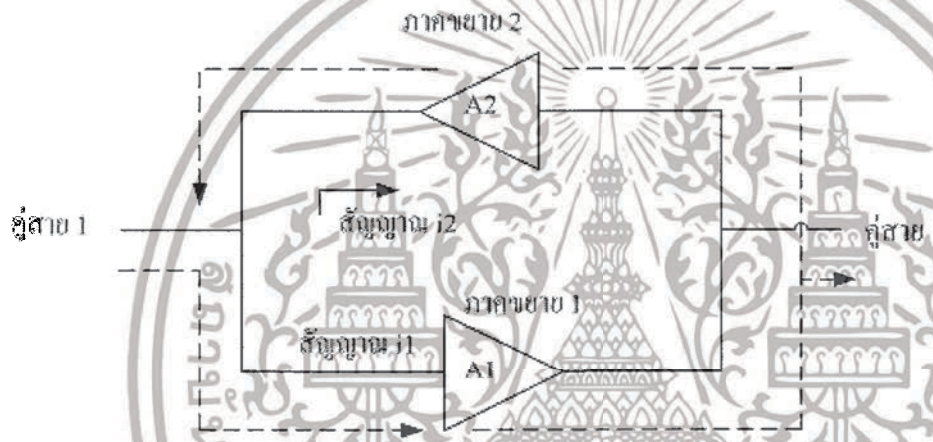


รูปที่ 2.12 แสดงให้เห็นสแกรมเบลอและดีสแกรมเบลอ

2.3.3 การส่งสัญญาณแบบสวนทิศทางกันได้

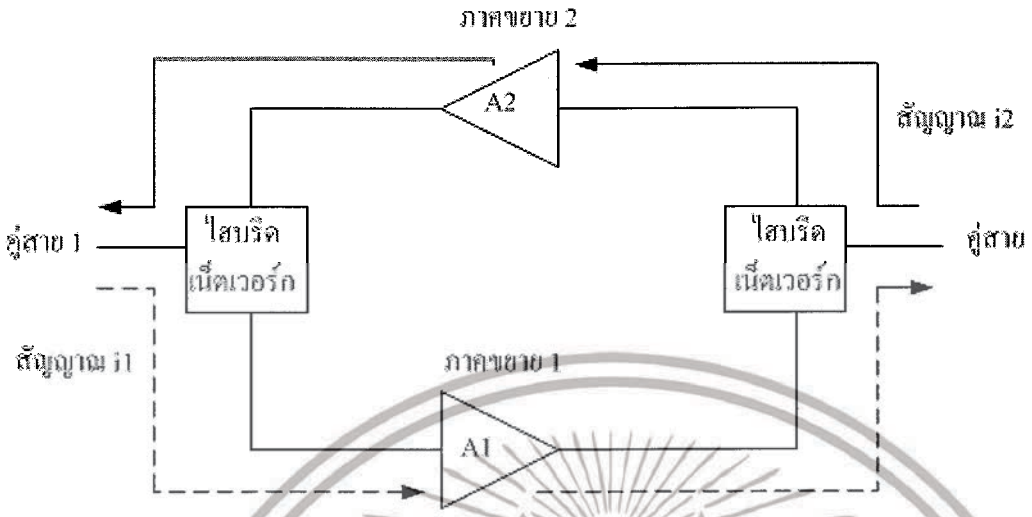
ปัจจุบันการติดต่อสื่อสาร โดยเฉพาะระบบโทรศัพท์มีความสำคัญมาก นอกจากเราจะใช้พูดคุยกันแล้ว ยังช่วยในการเชื่อมต่อข้อมูลในระบบคอมพิวเตอร์ เนื่องจาก Scramble ของเราต้องการออกแบบให้ใช้กับระบบการสื่อสารแบบ Full duplex ดังนั้นในบทนี้ เราจะมีการศึกษาถึงการส่งสัญญาณแบบสวนทิศทางกันในคู่สาย 2 เส้น

ในระบบโทรศัพท์ที่ใช้ 2 สายอย่างในประเทศไทยเรา สายสัญญาณ 2 เส้นนี้จะเป็นตัวผ่านสัญญาณทั้ง 2 ทิศทางในขณะเดียวกัน ปัญหาก็คือ ทำอย่างไรจึงจะส่งสัญญาณสวนทางกันได้บนสายสัญญาณ 2 เส้น และ หากมีการขยายสัญญาณด้วย จะใช้ด้านใดเป็นอินพุท ด้านใดเป็นเอาท์พุท



รูปที่ 2.13 การเกิดลูปย้อนกลับเมื่อไม่มีวงจร ไฮบริด

จากรูปที่ 2.13 เป็นการแก้ปัญหาลักษณะหนึ่ง โดยใช้ภาคขยายสัญญาณวางสลับทิศทางกัน มี A1 และ A2 เป็นอุปกรณ์ทวนสัญญาณ โดยมีการขยายในทิศทางตรงกันข้าม ถ้าสมมุติว่า คู่สาย 1 มีการส่งสัญญาณขึ้น ก็จะมีสัญญาณ I วิ่งเข้าที่จุด X แล้วแยกเป็น i1 และ i2 สัญญาณ i2 ไม่สามารถผ่านวงจรขยาย A2 ได้เพราะผิดทิศทาง แต่สัญญาณ i1 จะถูกขยายโดยวงจรขยาย A1 แล้วไหลเข้าที่จุด Y สัญญาณ i1 บางส่วนจะถูกขยายโดยวงจรขยาย A2 อีกครั้ง ทำให้สัญญาณ i1 เดินทางครบ loop เกิดการป้อนกลับแบบบวก ทำให้เกิดการ Oscillate ขึ้น การแก้ปัญหานี้ก็ต้องนำวงจรที่สามารถกันสัญญาณ i1 ไม่ให้ไหลย้อนกลับเข้าไปในวงจรขยาย A2 ได้ ซึ่งวงจรนั้นก็คือ วงจร Hi bridge ซึ่งจะเป็นดังรูปที่ 2.14



รูปที่ 2.14 การแก้การเกิดสภาพออสซิลเลตโดยเพิ่มวงจรไฮบริด

2.4 Finite Field Fourier Transform

2.4.1 Finite Fields

การบวกและการคูณของจำนวนเลขที่เป็นแบบโมดูโล-2 (Modulo-2) โดยถือว่า 2 เท่ากับ 0 ดังนั้นจะได้ 1 + 1 = 0 และ 1 = -1 สัญลักษณ์ 0 กับ 1 พร้อมด้วยการบวกและคูณนี้ รวมกันเข้าเป็นฟิลด์ (Filed) ซึ่งจะ เรียกว่า ไบนารีฟิลด์ (Binary Filed) เขียนแทน GF(2) เป็นฟิลด์ย่อยของฟิลด์ขยาย $GF(2^m)$ ในเส้นทางที่เป็นฟิลด์จำนวนจริงเป็นซับฟิลด์ของฟิลด์จำนวนเชิงซ้อน นอกจากนี้ จำนวน "0"และ "1" ที่เป็นการบวกที่เป็นส่วนประกอบหนึ่งในการขยายฟิลด์ จะเป็นการแสดงด้วยสัญลักษณ์ใหม่เป็น "α" แต่ละส่วนประกอบที่ไม่เป็นศูนย์ใน $GF(2^m)$ สามารถแสดงโดยกำลังของ α แต่ละส่วนประกอบ Infinite Set "F" เป็นรูปแบบโดยเริ่มต้นกับส่วนประกอบ $\{0, 1, \alpha\}$ และการสร้างส่วนประกอบเพิ่มโดยการคูณแบบก้าวหน้าเข้ามาหลังสุด โดย α ซึ่งได้ผลตามสมการ

$$F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^j, \ldots\} = \{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^j, \ldots\} \quad (1)$$

ไปถึง Finite Set ของส่วนประกอบของ $GF(2^m)$ จาก F เงื่อนไขต้องกำหนดบน F ดังที่มันกำหนดบน 2^m ส่วนประกอบและมันเป็นเงื่อนไขการปิดเซตของฟิลด์ส่วนประกอบการคูณเป็นคุณสมบัติโดยโพลิโนเมียลที่ลดรูปไม่ได้ (Irreducible Polynomial) แสดงดังสมการ

$$\alpha^{(2^m - 1)} + 1 = 0 \quad (2)$$

หรือ สมการที่สมมูล

$$\alpha^{(2m-1)} = 1 = \alpha^{0} \tag{3}$$

ใช้โพลิโนเมียลบังคับ ทุกๆส่วนประกอบมีกำลังเท่ากับหรือมากว่า 2^m -1 สามารถลดรูปส่วนประกอบกับกำลังที่ต่ำกว่า 2^m -1 ตามสมการ

$$\alpha^{(2m+n)} = \alpha^{(2m-1)}\alpha^{n+1} = \alpha^{n+1} \tag{4}$$

ดังนั้น จากสมการ(4) สามารถใช้รูปแบบลำดับ Finite F* จาก Infinite Sequence ตามสมการ

$$F^*=\{0,1, \alpha, \alpha^2,\ldots, \alpha^{2m-2}, \alpha^{2m-1}, \alpha^{2m},\ldots\}$$

$$F^*=\{0,, \alpha^0, \alpha^1, \alpha^2,\ldots, \alpha^{2m-2}, \alpha^0, \alpha^1, \alpha^2,\ldots\} \tag{5}$$

ก่อนจะสามารถเห็นรูปแบบสมการ ส่วนประกอบของ finite field $GF(2^m)$ เป็นตาม

$$GF(2^m) = \{0, \alpha^0, \alpha^1, \alpha^2,\ldots, \alpha^{2m-2}\} \tag{6}$$

2.4.2 Addition in the Extension Field $GF(2^m)$

ในแต่ละส่วนประกอบ 2^m ของ Finite Field $GF(2^m)$ สามารถแสดงได้ด้วยความแตกต่างโพลิโนเมียล เป็นค่าของเลขชี้กำลังลำดับที่สูงกว่า เราแสดงบางส่วนประกอบที่ไม่เป็นศูนย์ของ $GF(2^m)$ ที่โพลิโนเมียล $a_i(X)$ ที่ต่ำกว่าหนึ่งของ m สัมประสิทธิ์ ของ $a_i(X)$ เป็นเลขไม่เป็นศูนย์ สำหรับ i = 0, 1, 2, …,2^{m-2},

$$\alpha^{i} = a_i(X) = a_{i,0} + a_{i,1}X + a_{i,2}X^2 + \ldots + a_{i,m-1}X^{m-1} \tag{7}$$

กำหนดให้กรณีของ m = 3 ที่ Finite Field แสดงส่วนประกอบ $GF(2^3)$ ที่ทำให้เข้ากันของ 7 ส่วนประกอบ $\{\alpha^i\}$ และส่วนประกอบที่เป็นศูนย์ ในเทอมของส่วนประกอบพื้นฐาน $\{X^0, X^1, X^2\}$ รายละเอียดโดยสมการ (7) ตั้งแต่สมการ (3) แสดง $\alpha^0 = \alpha^7$ ที่มันเป็น 7 ส่วนประกอบที่ไม่เป็นศูนย์ ผลรวมของ 8 องค์ประกอบในฟิลด์นี้ แต่ละแถวเทียบกับลำดับของค่าไบนารีที่แสดงสัมประสิทธิ์ $a_{i,0}$, $a_{i,1}$, และ $a_{i,2}$ ในสมการ (7) ประโยชน์ของการใช้การขยายส่วนประกอบ $\{\alpha^i\}$

ในการวางของส่วนประกอบไบนารีเป็นเครื่องหมาย คณิตศาสตร์อำนวยความสะดวกการแสดงของเลขไม่เป็นไบนารี กระบวนการเข้ารหัสและถอดรหัส การเพิ่มของ 2 ส่วนประกอบของ Finite Field คือ กำหนด โมดูโล-2 รวมของแต่ละสัมประสิทธิ์โพลิโนเมียลของกำลังตามสมการ

$$\alpha^{i} + \alpha^{j} = (a_{i,0} + a_{j,0}) + (a_{i,1} + a_{j,1})X + \ldots + (a_{i,m-1} + a_{j,m-1})X^{m-1} \quad (8)$$

2.4.3 A Primitive Polynomial Is Used To Define The Finite Field

ชนิดของโพลิโนเมียล เรียก Primitive Polynomial มันน่าสนใจ เพราะว่า เป็นฟังก์ชั่นกำหนด Finite Field GF(2^m) ในรอบที่ต้องการในการกำหนดรหัส RS Codes ตามเงื่อนไขที่สำคัญและเพียงพอในการประกันโพลิโนเมียลเป็น Primitive แต่ละการลดรูปไม่ได้ของโพลิโนเมียล f(X) ของกำลัง m พูดได้ว่าเป็น Primitive ถ้า n เป็นจำนวนจริงที่เป็นบวกเล็กๆ สำหรับ f(X) หาร $X^n + 1$ เป็น $n = 2^m - 1$ แสดงสถานะ A /B หมายถึง A หารเข้าไปใน B ผลลัพธ์ที่ได้ไม่เป็นศูนย์และส่วนที่เหลือเป็นศูนย์ โพลิโนเมียลจะเป็นปกติแสดงลำดับต่ำถึงสูง บางเวลามันจะปรับเปลี่ยนตามรูปแบบที่กลับกัน

2.4.4 วงรอบพหุนาม(Polynomial Ring)

พหุนามบน GF(q) ซึ่งแสดงโดย

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \cdots \ldots + f_2x^2 + f_1x + f_0$$

ซึ่งจะเรียก x ว่า indeterminate และสัมประสิทธิ์ $f_{n-1}, f_{n-2}, \ldots, f_1, f_0$ เป็นสมาชิกใน GF(q) และ ถ้าหาก f_{n-1}=1 ก็จะเรียกพหุนามนั้นว่า พหุนาม โมนิค (Monic polynomial)

2.4.5 พหุนามปฐมภูมิ (Primitive Polynomial)

หากพหุนาม p(x)ซึ่งมีสัมประสิทธิ์ใน GF(q) = Zp จะเป็นพหุนามปฐมภูมิ หากว่าพหุนามนั้นมีราก α ใน GF(p^m) หรือการขยายเชิงเส้น (linear span) ของ $\langle 1, \alpha, \alpha^2.\alpha^3, \ldots, \alpha^{p^m - 2} \rangle$ ครอบคลุม GF(p^m) นอกจากนี้ p(x) จะเป็นพหุนามที่ดีกรีต่ำที่สุดที่มี α เป็นราก และพหุนามปฐมภูมิก็จะเป็นพหุนามที่ลดรูปไม่ได้ (irreducible polynomial)พหุนามปฐมภูมิจะมีจำนวนเทอมเป็นเลขคี่เสมอ (ไม่เช่นนั้น จะสามารถหารด้วย (x+1) ได้ลงตัว) หรืออีกนัยหนึ่งพหุนามปฐมภูมิ หากปรากฏจำนวนเต็มบวก n ที่ทำให้ p(x) หาร (X^n-1)ได้ โดยที่ค่า n ก็คือ $n = p^m - 1$และรากของพหุนามปฐมภูมิจะมีขนาด (order) เท่ากับ $p^m - 1$

ตัวอย่างเช่น เมื่อ p=2 และ m=4 โดยที่ α เป็นราก (สมาชิกใน GF(2^4) หรือ GF(16) เป็นการขยายเชิงเส้นของ $\langle 1, \alpha, \alpha^2, ..., \alpha^{14} \rangle$ จะเห็นว่า $n=p^m - 1 = 15$ หรือ $\alpha^{15} = 1$ พหุนามลดรูปไม่ได้ $p(x) = x^4 + x + 1$ ซึ่งมีดีกรี 4 เป็นพหุนามปฐมภูมิซึ่ง $x^4 + x + 1$หาร $(x^{15} - 1)$ได้ลงตัว (ผลหารคือ $x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$)

m	Primitive polynomial	m	Primitive polynomial
3	$1 + x + x^3$	14	$1 + x + x^6 + x^{10} + x^{14}$
4	$1 + x + x^4$	15	$1 + x + x^{15}$
5	$1 + x^2 + x^5$	16	$1 + x + x^3 + x^{12} + x^{16}$
6	$1 + x + x^6$	17	$1 + x^3 + x^{17}$
7	$1 + x^3 + x^7$	18	$1 + x^7 + x^{18}$
8	$1 + x^2 + x^3 + x^4 + x^8$	19	$1 + x + x^2 + x^5 + x^{19}$
9	$1 + x^4 + x^9$	20	$1 + x^3 + x^{20}$
10	$1 + x^3 + x^{10}$	21	$1 + x^2 + x^{21}$
11	$1 + x^2 + x^{11}$	22	$1 + x + x^{22}$
12	$1 + x + x^4 + x^6 + x^{12}$	23	$1 + x^5 + x^{23}$
13	$1 + x + x^3 + x^4 + x^{13}$	24	$1 + x + x^2 + x^7 + x^{24}$

ตารางที่ 2.2 แสดงค่า Primitive polynomial

2.4.6 สมาชิกของสนามจำกัด (Field Element)

ถ้าหาก α ซึ่งเป็นสมาชิกในสนามจำกัด GF(p^m) เป็นรากของพหุนามปฐมภูมิ p(x)แล้ว สมาชิกทั้งหมดของ GF(p^m) สามารถที่จะแสดงได้ ด้วยรากยกกำลังอย่างต่อเนื่อง คือ $\langle 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^{p^m - 2} \rangle$ ซึ่งสมาชิกเหล่านี้มาจากการมอดดูโลด้วย p(x)

ตัวอย่าง สมาชิกใน $GF(2^4)$ ซึ่ง $p(z) = z^4 + z + 1$

รากยกกำลัง	พหุนาม (z)	ข้อมูลไบนารี	ค่าฐานสิบ
α	z	0010	2
α^2	z^2	0100	4
α^3	z^3	1000	8
α^4	$z^4 = z + 1$	0011	3
α^5	$z^5 = z^2 + z$	0110	6
α^6	$z^6 = z^3 + z^2$	1100	12
α^7	$z^7 = z^3 + z + 1$	1011	11
α^8	$z^8 = z^2 + 1$	0101	5
α^9	$z^9 = z^3 + z$	1010	10
α^{10}	$z^{10} = z^2 + z + 1$	0111	7
α^{11}	$z^{11} = z^3 + z^2 + z$	1110	14
α^{12}	$z^{12} = z^3 + z^2 + z + 1$	1111	15
α^{13}	$z^{13} = z^3 + z^2 + 1$	1101	13
α^{14}	$z^{14} = z^3 + 1$	1001	9
α^{15}	$z^{15} = 1$	0001	1

ตารางที่ 2.3 ตารางแสดงสมาชิกใน $GF(2^4)$ ซึ่ง $p(z) = z^4 + z + 1$

การบวกกันของตัวเลข หรือสมาชิกในสนามจำนวนนี้ กระทำโดยการ XOR เช่น 10+14 = 4

ส่วนการคูณนั้นจะใช้การบวกของกำลัง เช่น 10 x 14 = 6

$$10 \times 14 = \alpha^9 \cdot \alpha^{11} = \alpha^{<20>_{15}} = \alpha^5 = 6$$

การหาค่าส่วนกลับก็ใช้การบวกของกำลังเช่น $(10)^{-1} = 12$

$$1/10 = \alpha^{15} / \alpha^9 = \alpha^{(15-9)} = \alpha^6 = 12$$

2.5 ภาษา VHDL

2.5.1 ประวัติความเป็นมาของภาษา VHDL

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

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่อง คอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

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

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

DoD ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ในเดือนมกราคมปี ค.ศ.1983 ไว้ดังนี้

1) ลักษณะทั่วไป

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

2) สนับสนุนการออกแบบแบบลำดับขั้น

การออกแบบแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบระบบที่มีหลายๆ ระดับ โดยในการ ออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อ และส่วนการบรรยายหน้าที่การทำงาน ซึ่งหน้าที่การทำงาน ของระบบสามารถกำหนดได้ด้วยตัวเอง หรืออาจถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆ ลง ไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเอง และไม่สามารถกำหนด การทำงานโดยลักษณะแบบโครงสร้างได้

3) ไลบรารี

VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของ อุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูก ต้องควรจัดเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไป ใช้ได้ด้วย

4) ลำดับคำสั่ง

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการ โดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษา เองก็ยังมีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบ ที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายใน ของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if - then - else และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้ สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

5) การกำหนดคุณสมบัติ

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

6) ชนิดของข้อมูล

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

7) โปรแกรมย่อย

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

8) การควบคุมเวลา

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอย เหตุการณ์ (Event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

9) การกำหนดแบบโครงสร้าง

การกำหนดโครงสร้างขององค์ประกอบต่างๆ สามารถกระทำได้ในทุกระดับของการออกแบบ โดยการกำหนดโครง สร้างขององค์ประกอบร่วมที่เกิดจากองค์ประกอบย่อยซึ่งแตกต่างกันหรือ เหมือนกันก็เป็นข้อกำหนดอย่างหนึ่งของ VHDL เช่นกัน

2.5.2 องค์ประกอบพื้นฐานของ VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบของ VHDL จะประกอบไปด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 2.15 โดยในการบรรยายการเชื่อมต่อจะขึ้น ต้นด้วยคำว่า ENTITY แล้วตามด้วยชื่อขององค์ประกอบจากนั้นตามด้วยคำว่า IS และถัดมาจะเป็นการบรรยายถึงพอร์ต การติดต่อ อินพุท - เอาท์พุท ขององค์ประกอบ ส่วนลักษณะภายนอกอื่น ๆ เช่น เวลา อุณหภูมิก็สามารถรวมเข้าไปในส่วนนี้ ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้ บรรยายหน้าที่การทำงานขององค์ประกอบ โดยหน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท เอาท์พุทและพารามิเตอร์ อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 2.15 และสำหรับการบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังจาก คำว่า BEGIN เป็นต้นไป

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

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

รูปที่ 2.15 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

1) การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ โดยในระดับนี้ต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบ ภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 2.16 ซึ่งเป็นบล็อคไดอะแกรม และการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับตัวจ่าย สัญญาณนาฬิกา ในบรรทัดแรกของการบรรยายการเชื่อมต่อเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดเป็น clock_component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็บ ส่วน IN และ OUT เป็นการกำหนด โหนดของสัญญาณให้เป็นอินพุทหรือเอาท์พุท และ BIT เป็นการแสดงชนิดของข้อมูล


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

รูปที่ 2.16 บล็อคไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_ component

2) การกำหนดรูปแบบการบรรยาย

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ ซึ่งในการบรรยายสามารถกำหนดค่าของสัญญาณ เอาท์พุทในเทอมของอินพุทหรือในรูปขององค์ประกอบอื่นๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 2.17 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมโดยมี en เป็นอินพุทและ ck เป็นเอาท์พุท PROCESS เป็นคำที่ใช้ในการเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม และภายในโปรเซสกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น "0" ถ้าสัญญาณ en มีค่าเป็น "1"' จะทำให้ตัวแปร periodic ถูกคอมพลีเมนท์ (complement) และส่งค่าให้กับ ck ซึ่งเป็นสัญญาณเอาท์พุท และสำหรับคำสั่ง WAIT จะเป็นการกำหนดให้สัญญาณมีคาบเวลาเท่ากับ 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS

BEGIN
   PEOCESS
      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 2.17 การบรรยายเชิงพฤติกรรมของ clock _component

3) หน่วยการออกแบบแพ็กเก็ต

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบการบรรยายระบบดิจิตอล สามารถเก็บไว้ใน ส่วนของแพ็กเก็ต ซึ่งหน่วยการออกแบบต่างๆ เช่น หน่วยการออกแบบ Entity หน่วยการออกแบบสถาปัตยกรรมหรือ หน่วยการออกแบบแพ็กเกจอื่นๆ สามารถเรียกข้อมูลเหล่านี้ไปใช้ได้ นอกจากนั้นสิ่งที่นิยมทำกันมากคือการนำรูปแบบ มาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐาน (เช่น ไอซีตระกูล 74XX เป็นต้น) มาเก็บไว้ในรูปของแพ็กเกจ ที่ทุกคนสามารถเข้าถึงได้ ตามปกติแล้วแพ็กเกจจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็กเกจ (Package declaration)และ ส่วนของบอดี้แพ็กเกจ (Package body) เนื่องจาก แพ็กเกจถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่ กำลังเขียนอยู่ ฉะนั้นการที่นำแพ็กเกจไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถ กระทำได้ด้วยชุดคำสั่ง USE

(3.1) PACKAGE DECLARATION

ส่วนที่มีความสำคัญที่สุดของแพ็กเกจ (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ ส่วนการประกาศแพ็กเกจ เนื่องจากเป็นส่วนที่ใช้กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็กเกจ สำหรับนำไปใช้ภายนอกตัวของแพ็กเกจเอง ถ้ามี การประกาศสิ่งใดๆ ในส่วนของส่วนบอดี้แพ็กเกจ แต่ไม่ถูกประกาศในส่วนการประกาศแพ็กเกจจะทำให้ค่าและพฤติ กรรมไม่สามารถนำไปใช้งานในส่วนนอกได้ซึ่งเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศ Entity คือ จุดเชื่อมต่อหรือ พอร์ต ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเกจสามารถสร้างขึ้นได้โดยไม่จำเป็น ต้องมีส่วนบอดี้ และยังสามารถนำไปใช้งานจากรูปแบบภายนอกได้เช่น ใช้สำหรับประกาศ ชนิด (Type) หรือสัญญาณ เช่นเดียวกับ ส่วนบอดี้แพ็กเกจที่ไม่จำเป็นต้องมี ส่วนของการประกาศแพ็กเกจ แต่แพ็กเกจนั้นจะไม่สามารถนำไปใช้จาก รูปแบบอื่นได้

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

รูปที่ 2.18 โครงสร้างทั่วไปของส่วนการประกาศแพ็กเกจ

(3.2) PACKAGE BODY

โครงสร้างซึ่งประกอบด้วยลำดับคำสั่งที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อยทั้งหลาย ซึ่งชื่อของโปรแกรมย่อยนั้นๆ ได้ถูกประกาศไปแล้วในส่วนของการประกาศแพ็กเกจ จะถูกเก็บไว้ในส่วนของบอดี้แพ็กเกจ ทั้งนี้รวมถึง การกำหนดค่าคงที่ต่างๆ อันได้แก่ค่าคงที่ที่ถูกประกาศชื่อไว้ก่อนในส่วนของการประกาศแพ็กเกจ และถูกกำหนดค่าใน ส่วนของบอดี้แพ็กเกจ ฉะนั้นในส่วนของบอดี้แพ็กเกจจึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็กเกจไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อย หรือค่าคงที่ การเขียนบอดี้แพ็กเกจนั้นจะเป็นไปตามกฎเกณฑ์ดังแสดงในรูปที่ 2.19

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

รูปที่ 2.19 โครงสร้างของบอดี้แพ็กเกจ

4) หน่วยการออกแบบ Configuration

ดังที่ทราบกันแล้วว่าระบบดิจิตอลรูปแบบหนึ่งไม่ว่าจะเป็นอะไรก็ตาม จะสามารถมีหน่วยการออกแบบ Entity ได้ เพียงหนึ่งเดียวเท่านั้น ซึ่งในหน่วยการออกแบบ Entity หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลาย หน่วย ดังนั้นจะต้องมีหน่วยการออกแบบ Configuration มาเพื่อกำหนดการใช้ Configuration ของการประกอบ Entity กับหน่วยการออกแบบสถาปัตยกรรมหน่วยใดๆ เข้าด้วยกัน

```
COMFIGURTRION identifier OF entiry_name IS
    Configuretion_declarative_part
END ;
```

รูปที่ 2.20 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

5) โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเดอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆ ไปค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าใช้ฟังก์ชัน แทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรลอจิกจริงๆ ในขณะที่ถ้าใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูล หรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่ 2.21 แสดงการใช้โพรซีเดอร์ เพื่อเปลี่ยนข้อมูลชนิด 8 บิตเป็นค่าจำนวนเต็ม และรูปที่ 2.22 แสดงการใช้ฟังก์ชันโดยกำหนดให้ X เป็นตัวแปรชนิด บิตแทนการกระทำในสมการบูลีน

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

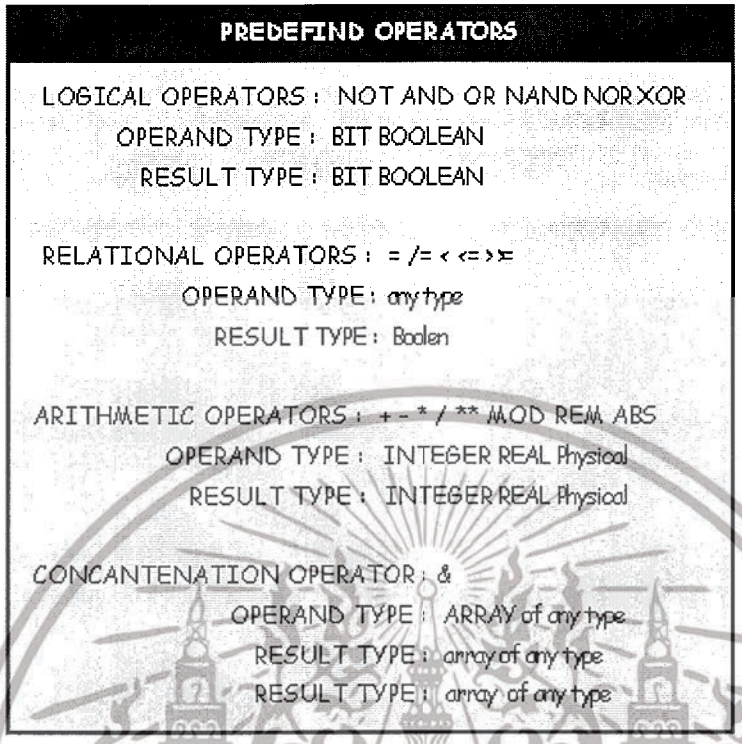
รูปที่ 2.21 การใช้โพรซีเดอร์

```
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 2.22 การใช้ฟังก์ชัน

6) โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา VHDL มีตัวดำเนินการหรือโอเปอร์เรเตอร์ทางลอจิกและคณิตศาสตร์เช่นเดียว กับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 2.23



รูปที่ 2.23 ตัวดำเนินการใน VHDL

7) เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์อุปกรณ์ทุกๆ ตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลา เข้ามาเกี่ยวข้องในทุกๆเหตุการณ์ที่เกิดขึ้นเสมอ VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อให้สามารถบรรยายรูป แบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วน ของการบรรยายสถาปัตยกรรม จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โปรเซสซึ่งมีการทำงานภายในเป็น แบบลำดับคำสั่งก็ตาม ซึ่งหากมีหลายๆ โปรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆ โปรเซสก็จะทำงานไปพร้อมๆ กัน ด้วย

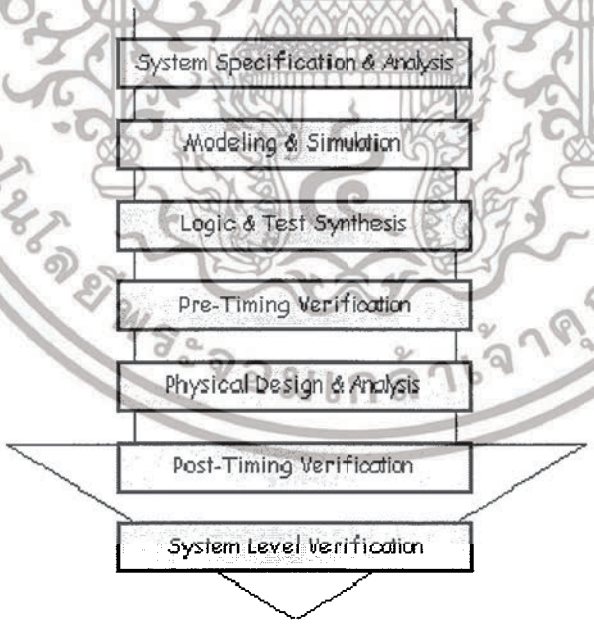
8) สัญญาณและตัวแปร

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยการ กำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการ ส่งผ่านค่าของสัญญาณ เช่น w <= a AFTER 12 NS หมายถึงการกำหนดค่าสัญญาณ a ให้กับ w หลังจากเวลา ผ่านไป 12 นาโนวินาที ในทางตรงข้ามตัวแปรมีลักษณะเป็นเสมือนตัวกลางที่ใช้ในการส่งผ่านข้อมูลและไม่มีเรื่องของ เวลาเข้ามาเกี่ยวข้อง

ด้วย ซึ่งตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชัน โพรซีเดอร์ และโปรเซส สำหรับการกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ :=

2.5.3 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของ บล็อกไดอะแกรมก่อนที่ทำวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายและวิเคราะห์การ ทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการ นอกจากนี้ยัง สามารถเพิ่มเติมในรายละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่าง (Top - Down Design) นั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom - Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ ก่อนแล้วจึงทำการจำลองการทำงาน และตรวจ สอบความถูกต้อง ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถ ออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.24 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.24 แสดงถึงขั้นตอนของการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้าง เล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

1) สร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ใน การแก้ปัญหา

2) เขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL หรือ ภาษา HDL อื่น ๆ สำหรับบรรยายพฤติกรรมการ ทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3) หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมา เป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจริง หรือสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับ วงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของ วงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือ ไม่ก็อยู่ในรูปของ Netlist ที่สามารถนำไปผลิตในอุปกรณ์อื่นได้

4) หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือ Netlist แล้ว ข้อมูลนี้จะถูกใช้สำหรับจำลองการทำงานในเรื่อง ความถูกต้องของฟังก์ชัน พร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาประกอบการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ ทางอิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็น เวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถทำ งานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

5) ผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจ จะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวม ASIC

6) ทำการตรวจสอบการทำงานและตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

7) นำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบร่วมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

บทที่ 3

การทดสอบสมการ Finite Field Fourier Transform บน Matlab

ในการออกแบบนั้นในส่วนแรกเราจะทำการทดลองบนโปรแกรม Matlab ก่อนเพื่อทดสอบดูว่าสมการ Finite Fourier Transform จะสามารถทำให้สัญญาณเสียงผิดเพี้ยนไปจากต้นฉบับจนไม่สามารถรับฟังได้จริงหรือไม่ มีขั้นตอนการทดลองดังนี้

3.1 ขั้นตอนการทดลอง

1. เราจะทำการเขียน Code บนโปรแกรม Matlab ขึ้นมาเพื่อใช้ในการเรียกเสียงที่เราอัดไว้แล้วเข้ามาในโปรแกรม
2. ทำการ quantization เป็น 15 ระดับตามแกน y
3. นำสัญญาณที่เราทำการ quantization แล้วมาทำการยกระดับค่าแกน y ขึ้น เพื่อให้จำนวนของข้อมูลมีค่าอยู่ในช่วงที่เราต้องการ เพื่อนำไปเข้ารหัส
4. นำสัญญาณมาแบ่ง block โดย 1 block จะมีข้อมูล 15 ค่า
5. นำสัญญาณ ไปแปลงค่าเป็นเลขชี้กำลังของ alfa ตามค่าในตาราง
6. นำสัญญาณ ไปเข้ารหัส
7. ฟังเสียงที่ได้จากสัญญาณการเข้ารหัส
8. นำสัญญาณที่เข้ารหัสมามาแปลงค่าเป็นเลขชี้กำลังของ alfa ตามค่าในตาราง
9. นำสัญญาณมาถอดรหัส
10. ฟังเสียงที่ได้จากสัญญาณที่ถอดรหัส

วิธีการที่เราจะใช้ในการปกปิดสัญญาณอินพุทที่เรารับเข้ามานั้น เราจะใช้สมการ Finite Field Fourier Transform มาใช้ในการทำให้สัญญาณอินพุทที่เรารับเข้ามานั้นมีความผิดเพี้ยนไป โดยเราจะนำอินพุทที่เข้ามากำหนดให้เป็น v_i จากนั้นนำเข้าไปแทนในสมการเพื่อนำไปคูณกับค่า α ซึ่งเป็นค่าคงที่ เราจะได้เอาท์พุทออกมานั้นก็คือ v_k นี้คือกระบวนการในการนำสัญญาณมาเข้ารหัส สามารถดูได้จาก block diagram

3.2 สมการ Finite Field Fourier Transform ที่ใช้ในการทดสอบ

3.2.1 สมการที่ใช้ในการทดสอบวงจรเข้ารหัส

$$V_k = \sum_{i=0}^{N-1} v_i \alpha^{<ik>_N} \qquad \text{for k} = 0,1,2,3,\ldots N\text{-}1$$

3.2.1 สมการที่ใช้ในการทดสอบวงจรถอดรหัส

$$V_i = \sum_{k=0}^{N-1} v_k \alpha^{<-ik>_N} \qquad \text{for i} = 0,1,2,3,\ldots N\text{-}1$$

โดยในการทดสอบนั้นเราจะทำการเขียนโปรแกรมแยกไว้เป็นส่วนต่าง โดยโปรแกรมที่ใช้ทดสอบที่เขียนแยกไว้เป็นส่วนๆมีดังนี้

3.3 Code โปรแกรมในส่วนต่างๆ

3.3.1 โปรแกรมส่วนรับเสียงเข้ามา

```
clear;
[y fs] = wavread('sound1.wav');
figure(1)
plot(y);
```

3.3.2 โปรแกรมส่วน quantization

```
c=real(y);
sig = c;
mi=min(sig);
mx=max(sig);
partition = [mi:.14:mx];
codebook = [mi-.14:.14:mx];
[index,quants] = quantiz(sig,partition,codebook);
figure(2);
plot(quants,'.');
```

3.3.3 โปรแกรมส่วนการยกระดับแกน y

```
quants1= (quants +1)*7 ;
quants2=round(quants1);
figure(3);
plot(quants2,'.');
```

3.3.4 โปรแกรมส่วนการแบ่ง block

```
d=1;
for e= 1:15:25980
  vk(d,:) = quants2(e:e+14);
  d=d+1;
end
```

3.3.5 โปรแกรมส่วนแปลงค่า input เป็นเลขชี้กำลัง alfa ตามตาราง

```
for j = 1:1:1732
  vk(j,:);
  for m = 1:1:15
      if (vk(j,m) == 1)
      vk1(m) = 0;
      elseif (vk(j,m)==2)
      vk1(m) = 1;
      elseif (vk(j,m)==3)
      vk1(m) = 4;
      elseif (vk(j,m)==4)
      vk1(m) = 2;
      elseif (vk(j,m)==5)
      vk1(m) = 8;
      elseif (vk(j,m)==6)
      vk1(m) = 5;
      elseif (vk(j,m)==7)
```

```
            vk1(m) = 10;
            elseif (vk(j,m)==8)
            vk1(m) = 3;
            elseif (vk(j,m)==9)
            vk1(m) = 14;
            elseif (vk(j,m)==10)
            vk1(m) = 9;
            elseif (vk(j,m)==11)
            vk1(m) = 7;
            elseif (vk(j,m)==12)
            vk1(m) = 6;
            elseif (vk(j,m)==13)
            vk1(m) = 13;
            elseif (vk(j,m)==14)
            vk1(m) = 11;
            elseif (vk(j,m)== 15)
            vk1(m) = 12;
            else
                vk1(m) = 99;
            end
        end
        vh(j,:) = vk1;
    end
```

3.3.6 โปรแกรมส่วนเข้ารหัส

```
a1 = [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ];
vl = 0;
e = 0;
for z = 1:1:1732
   for k=1:1:15
```

```
for m=1:1:15
        vn = e * a1
        vk2(z,m) = mod((vh(z,m) + vn(m)),15);
        if (vk2(z,m) == 0)
        vk3(m) = 1;
        elseif (vk2(z,m)==1)
        vk3(m) = 2;
        elseif (vk2(z,m)==2)
        vk3(m) = 4;
        elseif (vk2(z,m)==3)
        vk3(m) = 8;
        elseif (vk2(z,m)==4)
        vk3(m) = 3;
        elseif (vk2(z,m)==5)
        vk3(m) = 6;
        elseif (vk2(z,m)==6)
        vk3(m) = 12;
        elseif (vk2(z,m)==7)
        vk3(m) = 11;
        elseif (vk2(z,m)==8)
        vk3(m) = 5;
        elseif (vk2(z,m)==9)
        vk3(m) = 10;
        elseif (vk2(z,m)==10)
        vk3(m) = 7;
        elseif (vk2(z,m)==11)
        vk3(m) = 14;
        elseif (vk2(z,m)==12)
        vk3(m) = 15;
        elseif (vk2(z,m)==13)
```

```
vk3(m) = 13;
elseif (vk2(z,m)==14)
vk3(m) = 9;
else
        vk3(m) = 1000;
end
vl = bitxor(vl, vk3(m));
end
vo4(z,k) = vl;
vl = vl*0
e = e+1;
end
end
```

3.3.7 โปรแกรมส่วนเรียงข้อมูลต่อกันให้เป็น 1 แถว

```
vo2 = vo4(1,:);
for n=2:1732
  vo2 = cat(2,vo2,vo4(n,:))
end
figure(5)
plot(vo2,'.');
```

3.3.8 โปรแกรมส่วนการเล่นเสียงและบันทึกเสียงที่เข้ารหัสแล้ว

```
wavplay(vo2);
wavwrite(vo2,fs,'work1.wav')
```

3.3.9 โปรแกรมส่วนการแบ่ง block ข้อมูลที่เข้ารหัสแล้ว

```
d=1;
for e= 1:15:25980
   vk(d,:) = vo2(e:e+14);
   d=d+1;
```

```
end
```

3.3.10 โปรแกรมส่วนแปลงค่าข้อมูลที่เข้ารหัสแล้วเป็นเลขชี้กำลัง alfa

```
for j = 1:1:1732
  vo(j,:);
  for m = 1:1:15
      if (vo(j,m) == 1)
          vo1(m) = 0;
      elseif (vo(j,m)==2)
          vo1(m) = 1;
      elseif (vo(j,m)==3)
          vo1(m) = 4;
      elseif (vo(j,m)==4)
          vo1(m) = 2;
      elseif (vo(j,m)==5)
          vo1(m) = 8;
      elseif (vo(j,m)==6)
          vo1(m) = 5;
      elseif (vo(j,m)==7)
          vo1(m) = 10;
      elseif (vo(j,m)==8)
          vo1(m) = 3;
      elseif (vo(j,m)==9)
          vo1(m) = 14;
      elseif (vo(j,m)==10)
          vo1(m) = 9;
      elseif (vo(j,m)==11)
          vo1(m) = 7;
      elseif (vo(j,m)==12)
          vo1(m) = 6;
```

```
elseif (vo(j,m)==13)
        vo1(m) = 13;
elseif (vo(j,m)==14)
        vo1(m) = 11;
elseif (vo(j,m)== 15)
        vo1(m) = 12;
else
        vo1(m) = 99;
end
end
vh1(j,:) = vo1;
end
```

3.3.11 โปรแกรมส่วนถอดรหัส

```
a2 = [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ];
vl = 0;
e = 0;
for z = 1:1:1732
 for k=1:1:15
   for m=1:1:15
     if (vh1(z,m) == 99)
        vk2(z,m) = 99;
     else
        vn = e * a2
     vk2(z,m) = mod((vh1(z,m) + vn(m)),15);
     end
     if (vk2(z,m) == 0)
        vk3(m) = 1;
     elseif (vk2(z,m)==1)
        vk3(m) = 2;
```

```
elseif (vk2(z,m)==2)
        vk3(m) = 4;
elseif (vk2(z,m)==3)
        vk3(m) = 8;
elseif (vk2(z,m)==4)
        vk3(m) = 3;
elseif (vk2(z,m)==5)
        vk3(m) = 6;
elseif (vk2(z,m)==6)
        vk3(m) = 12;
elseif (vk2(z,m)==7)
        vk3(m) = 11;
elseif (vk2(z,m)==8)
        vk3(m) = 5;
elseif (vk2(z,m)==9)
        vk3(m) = 10;
elseif (vk2(z,m)==10)
        vk3(m) = 7;
elseif (vk2(z,m)==11)
        vk3(m) = 14;
elseif (vk2(z,m)==12)
        vk3(m) = 15;
elseif (vk2(z,m)==13)
        vk3(m) = 13;
elseif (vk2(z,m)==14)
        vk3(m) = 9;
else
        vk3(m) = 0;
end
vl = bitxor(vl, vk3(m));
```

```
            end
            vo3(z,k) = vl;
            vl = vl*0
            e = e-1;
        end
    end
```

3.3.12 โปรแกรมส่วนเรียงข้อมูลต่อกันให้เป็น 1 แถว

```
tk = vo3(1,:);
for n=2:1732
  tk = cat(2,tk,vo3(n,:))
end
```

3.3.13 โปรแกรมส่วนการลดระดับแกน y กลับเป็นค่าเดิม

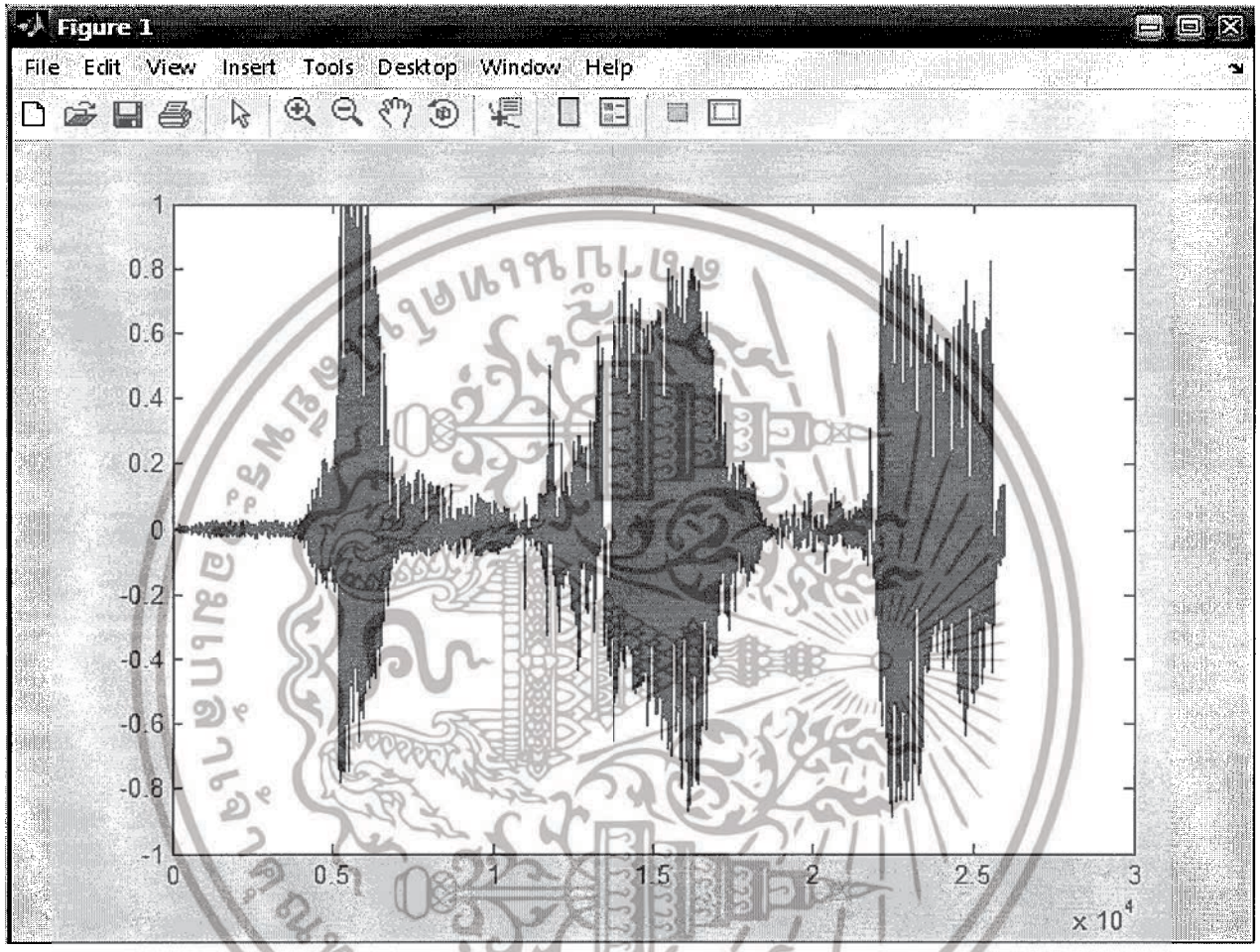
```
i4 = (tk/7)-1;
figure(6);
plot(i4,'.');
```

3.3.14 โปรแกรมส่วนการเล่นเสียงและบันทึกเสียงที่ถอดรหัสแล้ว

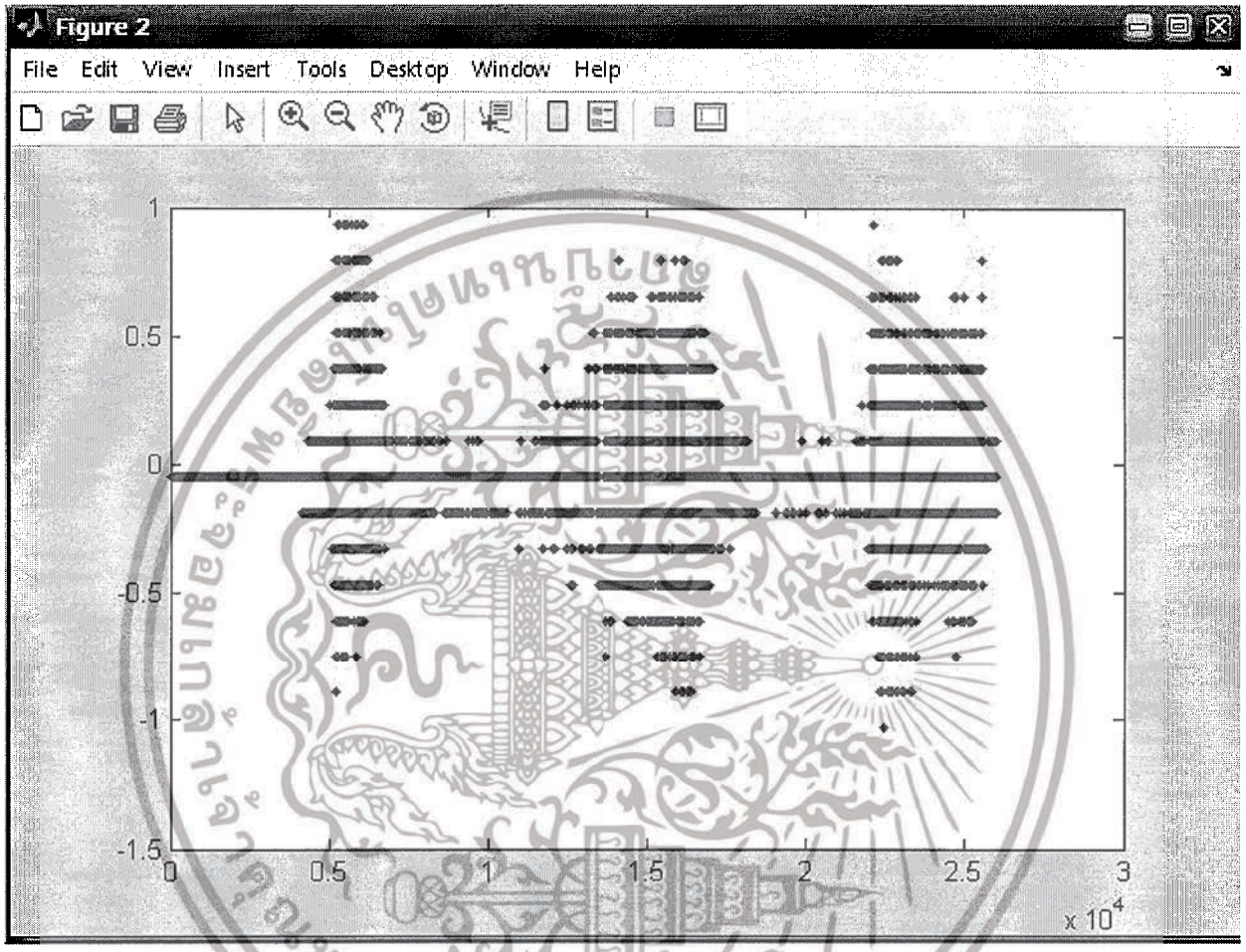
```
wavplay(i4);
wavwrite(i4,fs,'work2.wav'
```

3.4 ผลการทดสอบ

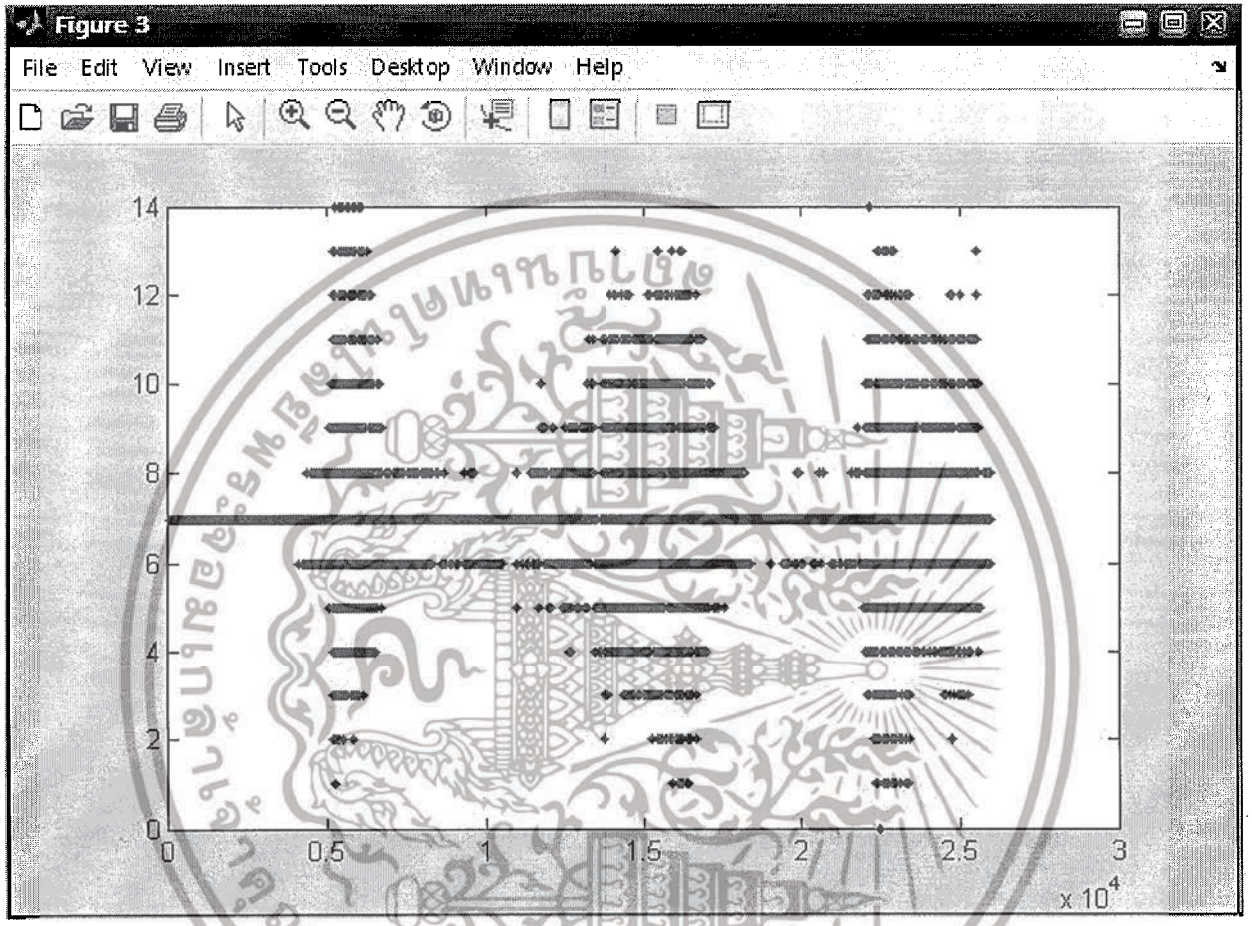
ในผลการทดสอบที่ได้เราจะทำการพร็อตกราฟในแต่ละจุดเพื่อดูว่า ลักษณะของสัญญาณที่จุดต่างๆนั้นเป็นอย่างไร



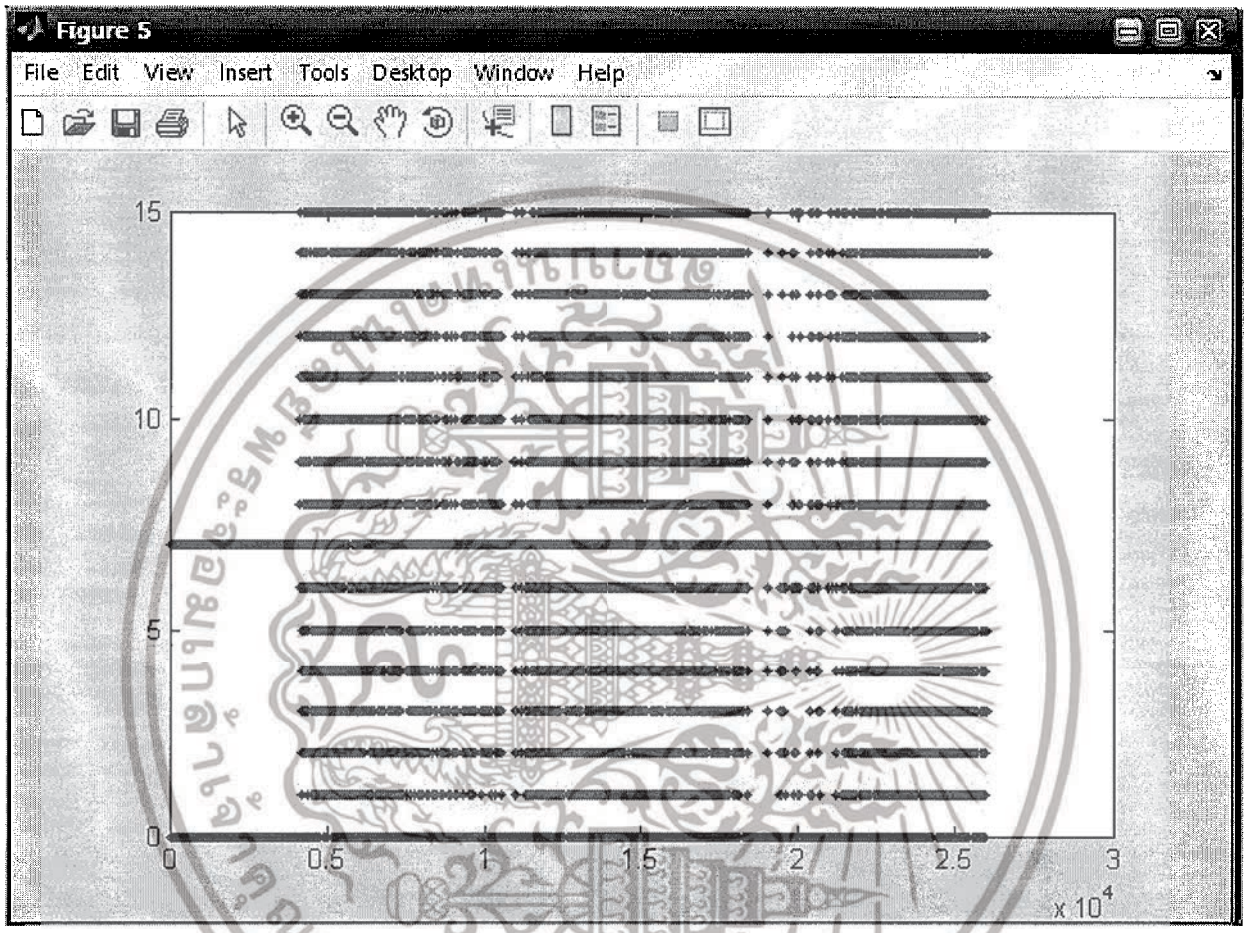
รูปที่ 3.1 สัญญาณเสียงอินพุทที่รับเข้ามา



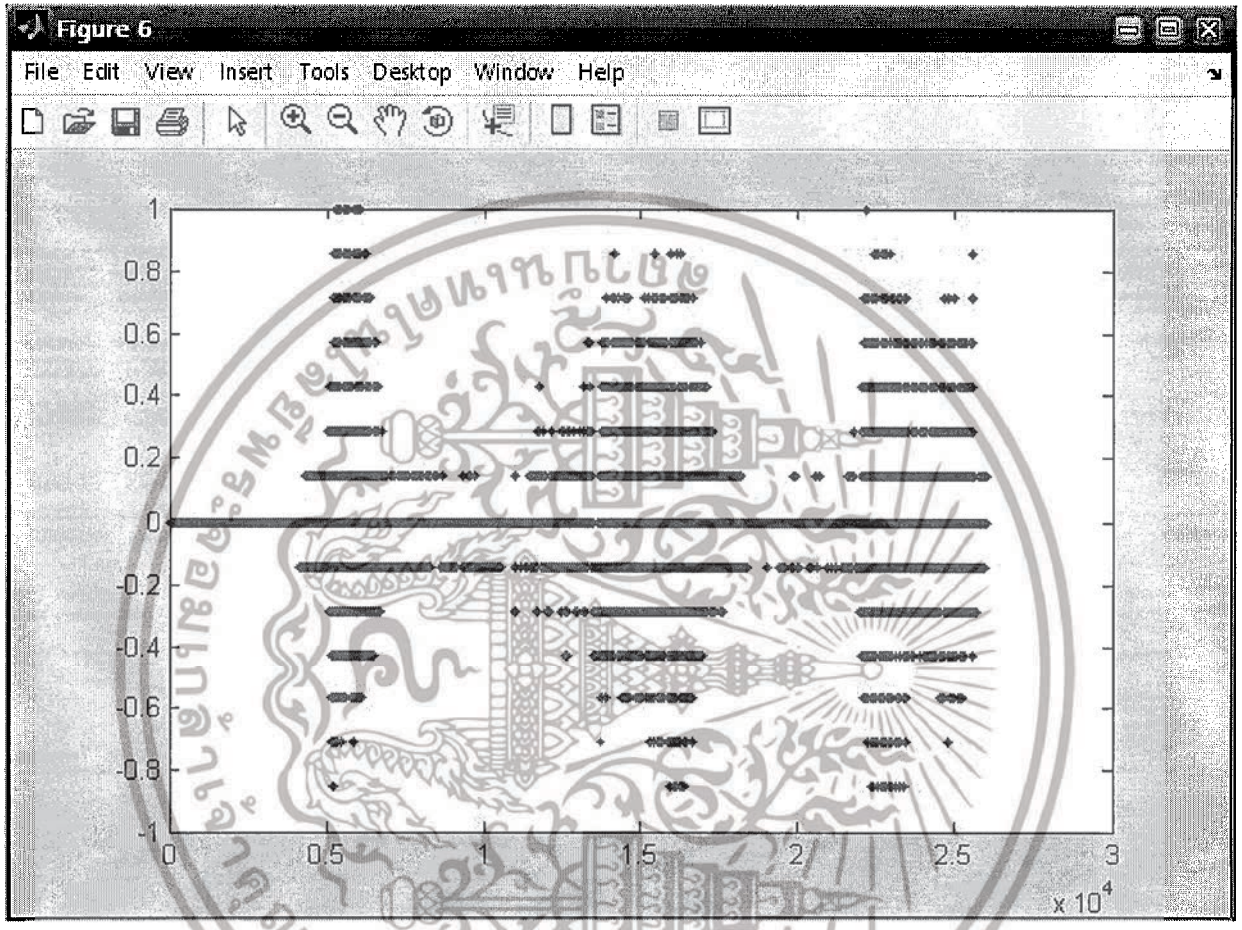
รูปที่ 3.2 สัญญาณเสียงที่ทำการ quantization แล้วให้มี 15 ระดับ



รูปที่ 3.3 สัญญาณที่ทำการยกระดับแกน y แล้ว



รูปที่ 3.4 สัญญาณที่ได้จากการเข้ารหัส



รูปที่ 3.5 สัญญาณที่ได้จากการถอดรหัส

บทที่ 4
การออกแบบวงจรปกปิดเสียงด้วยภาษา VHDL

4.1 Finite Field Fourier Transform

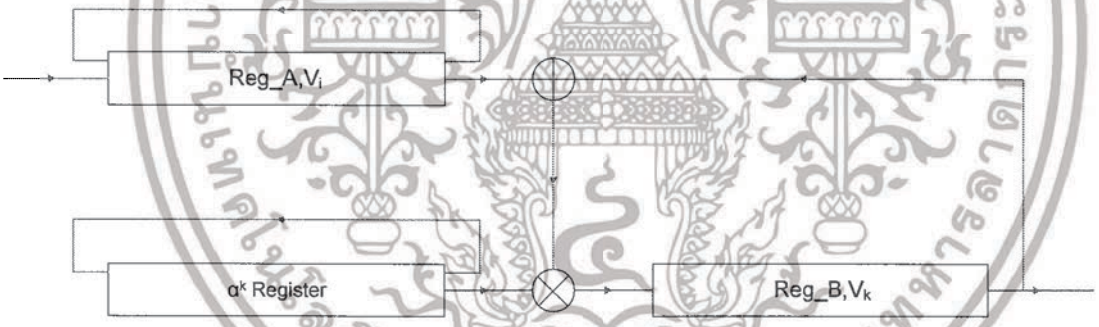
4.1.1 สมการที่ใช้ในการเข้ารหัส

วิธีการที่เราจะใช้ในการปกปิดสัญญาณอินพุทที่เรารับเข้ามานั้น เราจะใช้สมการ Finite Field Fourier Transform มาใช้ในการทำให้สัญญาณอินพุทที่เรารับเข้ามานั้นมีความผิดเพี้ยนไป โดยเราจะนำอินพุทที่เข้ามากำหนดให้เป็น v_i จากนั้นนำเข้าไปแทนในสมการเพื่อนำไปคูณกับค่า α ซึ่งเป็นค่าคงที่ เราจะได้เอาท์พุทออกมานั้นก็คือ v_k นี้คือกระบวนการในการนำสัญญาณมาเข้ารหัส สามารถดูได้จาก block diagram

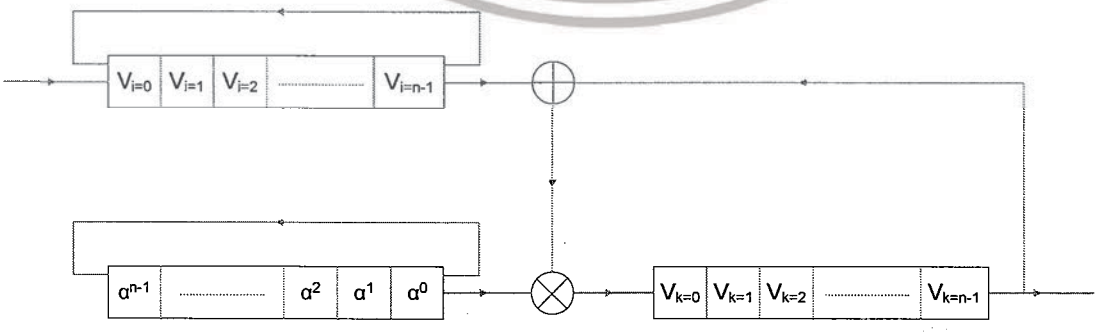
$$V_k = \sum_{i=0}^{N-1} v_i \alpha^{<ik>_N} \qquad \text{for } k = 0,1,2,3,\ldots N\text{-}1$$

$$= v_0\alpha^{0k} + v_1\alpha^{1k} + v_2\alpha^{2k} + v_3\alpha^{3k} + \ldots\ldots\ldots\ldots\ldots + v_{(n-1)}\alpha^{(n-1)k}$$

$$= (((..(((v_{(n-1)}+0)\,\alpha^{k} + v_{(n-2)})\,\alpha^{k} + v_{(n-3)})\,\alpha^{k} + \ldots\ldots + v_2)\,\alpha^{k} + v_1)\,\alpha^{k} + v_0)$$



รูปที่ 4.1 block diagram สมการ Finite Field Fourier Transform



รูปที่ 4.2 block diagram สมการ Finite Field Fourier Transform

4.1.2 สมการที่ใช้ในการถอดรหัส

เมื่อเราได้ค่า v_k แล้วเราสามารถที่จะนำค่า v_k มาแปลงกลับให้เป็น v_i เหมือนเดิมได้จากสมการ inverse ด้านล่างโดยค่า α ที่นำมาคูณกลับ vk นั้นจะเป็นค่า α ที่เป็นค่า inverse

$$V_i = \sum_{k=0}^{N-1} v_k \alpha^{<-ik>_N} \quad \text{for i = 0,1,2,3,...N-1}$$

$$= v_0 \alpha^{-(0k)} + v_1 \alpha^{-(1k)} + v_2 \alpha^{-(2k)} + v_3 \alpha^{-(3k)} + \ldots\ldots\ldots\ldots + v_{(n-1)} \alpha^{-(n-1)k}$$

$$= (((..(((v_{(n-1)}+0)\ \alpha^{-k} + v_{(n-2)})\ \alpha^{-k} + v_{(n-3)})\ \alpha^{-k} + \ldots\ldots + v_2)\ \alpha^{-k} + v_1)\ \alpha^{-k} + v_0)$$

โดยในการออกแบบนั้นเราจะทำการเขียนโปรแกรมที่เป็นภาษา VHDL แยกไว้เป็นส่วนหรือ แยกไว้เป็นโปรแกรมย่อยๆก่อน หลังจากนั้นถึงค่อยนำมารวมให้เป็นโปรแกรมที่สมบูรณ์อีกทีหนึ่ง โดยโปรแกรมย่อยๆที่เราเขียนแยกไว้จะมีหน้าที่ในการทำงานที่แตกต่างกันไป

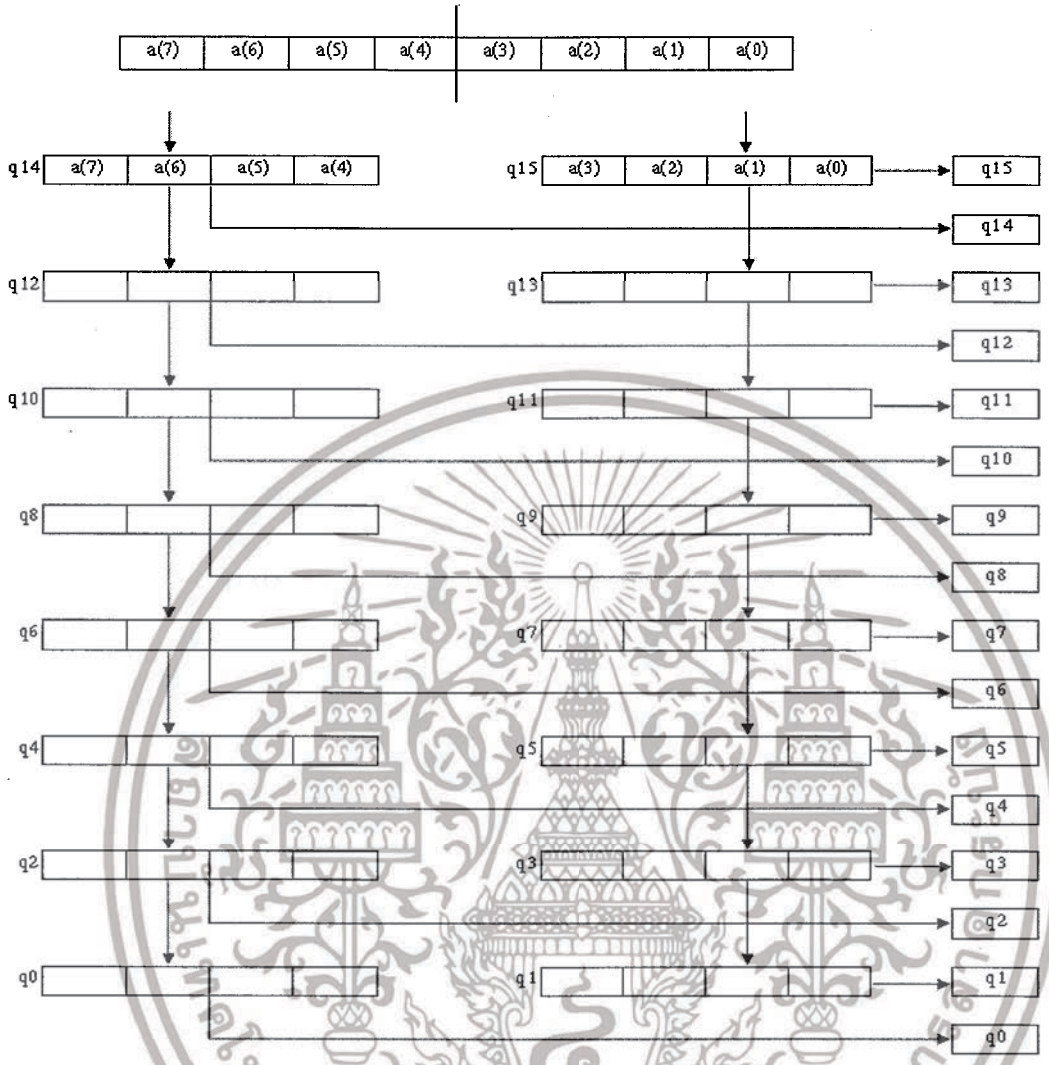
ในการออกแบบนั้นเราจะต้องทำการออกแบบแยกเป็น 2 ส่วนด้วยกันคือ 1.ส่วนของวงจรเข้ารหัส และ 2.ส่วนของวงจรถอดรหัส โดยมีวิธีการออกแบบดังนี้

4.2 วงจรเข้ารหัสสัญญาณเสียง

ในการออกแบบวงจรเข้ารหัสเราจะทำการออกแบบเป็นวงจรย่อยๆไว้ก่อนแล้วถึงจะนำวงจรที่เราออกแบบไว้มารวมเป็นวงจรที่สมบูรณ์ในภายหลัง โดยวงจรย่อยที่ได้ทำการออกแบบไว้มีดังนี้

4.2.1 วงจรแยก 8 bit เป็น 4 bit

วงจรนี้จะทำรับค่าอินพุทที่มีขนาด 8 bit และเมื่อนำมาเข้าวงจรเราจะทำการแยกบิตที่ 7-4 มาเก็บเป็นค่า 4 บิต 1 ค่า และบิตที่ 3-0 มาเก็บเป็นค่า 4 บิต อีกหนึ่งค่า หลังจากนั้นจะทำการ Shift ค่าข้อมูล 4 บิตทั้ง 2 ค่าไปเก็บไว้ แล้วก็จะทำการรับค่าอินพุทค่าใหม่เข้ามาทำอย่างนี้จนได้ค่า 4 บิต ครบทั้ง 16 ค่า จากนั้นนำค่าที่ได้มาทำการเรียงลำดับตั้งแต่ค่าสุดท้ายไปจนถึงค่าแรกสุด สามารถดูได้จากรูปด้านล่าง



รูปที่ 4.3 แสดงการทำงานของวงจร แยก 8 bit เป็น 4 bit

4.2.2 วงจรชี้ค่าอินพุท

โดยวงจรนี้จะรับค่าอินพุทที่ได้แยกจาก 8 บิตเป็น 4 บิตแล้วโดยจะรับค่าเข้ามาและจะส่งสัญญาณเอาต์พุทออกไปทีล่ะค่าโดนค่าเอาต์พุทที่ส่งออกไปนั้นจะเป็นการวน loop โดยค่าต่อไปที่จะส่งออกไปจะขึ้นกับค่าสัญญาณ ctr ที่เข้ามา

4.2.3 วงจร ROM ชี้ค่าเลขชี้กำลัง α

โดยเราจะทำการเขียนโปรแกรมขึ้นมาโดยจะรับค่า input ที่เป็นข้อมูลแบบ binary ที่มีขนาด 4 bit เมื่อนำมาผ่านวงจรนี้ เราจะนำข้อมูลที่รับเข้ามามาทำการชี้ให้เป็นเลขชี้กำลังของ α

โดยที่เราเขียนเป็นค่าคงที่เอาไว้แล้วว่าข้อมูลไบนารีแต่ละตัวมีค่าเท่ากับเลขชี้กำลัง α ค่าใด โดยสามารถดูได้จากตาราง

รากยกกำลัง	ข้อมูลไบนารี
α	0010
α^2	0100
α^3	1000
α^4	0011
α^5	0110
α^6	1100
α^7	1011
α^8	0101
α^9	1010
α^{10}	0111
α^{11}	1110
α^{12}	1111
α^{13}	1101
α^{14}	1001
α^{15}	0001

ตารางที่ 4.1 ตารางแสดงค่าเลขชี้กำลัง α ค่าต่างๆว่าเท่ากับค่าไบนารีเท่าไร

4.2.4 วงจร ROM ชี้ค่าข้อมูลที่เป็นไบนารี

โดยเราจะทำการเขียนโปรแกรมขึ้นมาโดยจะรับค่า input ที่เป็นเลขชี้กำลังของ α เมื่อนำมาผ่านวงจรนี้ เราจะนำข้อมูลที่รับเข้ามามาทำการชี้ให้เป็นข้อมูลที่เป็นไบนารี โดยที่เราเขียนเป็นค่าคงที่เอาไว้แล้วว่าเลขชี้กำลัง α แต่ล่ะตัวมีค่าเท่ากับข้อมูลที่เป็นไบนารีค่าใด โดยสามารถดูได้จากตาราง

ข้อมูลไบนารี	รากยกกำลัง
0010	α
0100	α^2
1000	α^3
0011	α^4
0110	α^5
1100	α^6
1011	α^7
0101	α^8
1010	α^9
0111	α^{10}
1110	α^{11}
1111	α^{12}
1101	α^{13}
1001	α^{14}
0001	α^{15}

ตารางที่ 4.2 ตารางแสดงค่าไบนารีต่างๆว่ามีค่าเท่ากับเท่าไรของเลขชี้กำลัง α

4.2.5 วงจร OR Gate

เป็นวงจรที่เราสร้างขึ้นมาเพื่อใช้ในการ OR กันระหว่าง input ที่รับเข้ามา 2 ค่าซึ่งวงจรในส่วนนี้จะไปเป็นส่วนประกอบของวงจร Full adder อีกทีหนึ่ง นิยามของ OR Gate คือ Gate ที่ให้ Output เป็น 0 เมื่อ Input ทุกตัวเป็น 0 และจะให้ Output เป็น 1 เมื่อมี input ตัวใดตัวหนึ่งเป็น 1

OR Gate มีตัวดำเนินการ คือ +

ผลลัพธ์ ของ OR Gate สามารถเขียนได้ดังสมการ Y = A + B

ตารางค่าความจริง (Truth Table) ของ OR Gate เป็นดังนี้

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

ตารางที่ 4.3 ตารางค่าความจริง (Truth Table) ของ OR Gate

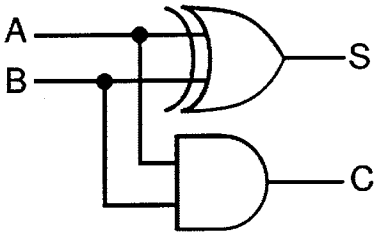
รูปที่ 4.4 สัญลักษณ์ OR Gate

4.2.6 วงจร Half adder

วงจรบวกหนึ่งบิตครึ่งอัตรา(Single bit Half-Adder)เป็นวงจรบวกมีอินพุทหนึ่งบิตสองค่า วงจรจะบวกเลขและให้ค่าสองค่าคือ ค่าผลรวม(Sum) และตัวทด(Carry) โดยวงจรนี้จะได้ค่าผลรวมและตัวทดตามตารางต่อไปนี้

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

ตารางที่ 4.4 ตารางแสดงค่าของ Half-Adder



รูปที่ 4.5 เป็นการต่อวงจร Half-Adder แบบการใช้ XOR-Gate

ลักษณะของสมการ Half-Adder คือ

$$Sum = A \oplus B$$
$$Carry = A \cdot B$$

4.2.7 วงจร Full adder 1 bit

วงจรบวกหนึ่งบิตเต็มอัตรา (Single bit Full-Adder) เป็นวงจรที่ใช้ในการบวกเลขฐานสองเช่นกันแต่จะสามารถบวกได้มากกว่า Half-Adder โดยจะมีอินพุทหนึ่งบิตสามค่า คือสองค่าที่จะบวก และ ตัวทดเข้า(Carry in) จากภายนอก เพื่อเชื่อมตัวกับวงจรตัวอื่น วงจรจะบวกเลขและให้ค่าสองค่าคือ ค่าผลรวม(Sum) และตัวทด(Carry) เช่นเดียวกับ Half-Adder ซึ่งแสดงได้ตามตารางค่าความจริงต่อไปนี้

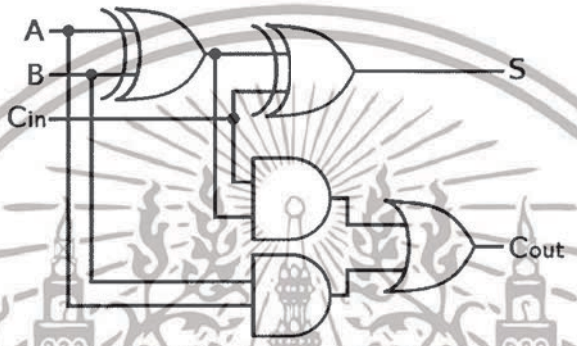
A	B	Carry in	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ตารางที่ 4.5 ตารางแสดงค่าของ Full-Adder

ลักษณะสมการของวงจร Full-Adder คือ

$$Sum = A \oplus B \oplus C_{in}$$
$$Carry_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$$

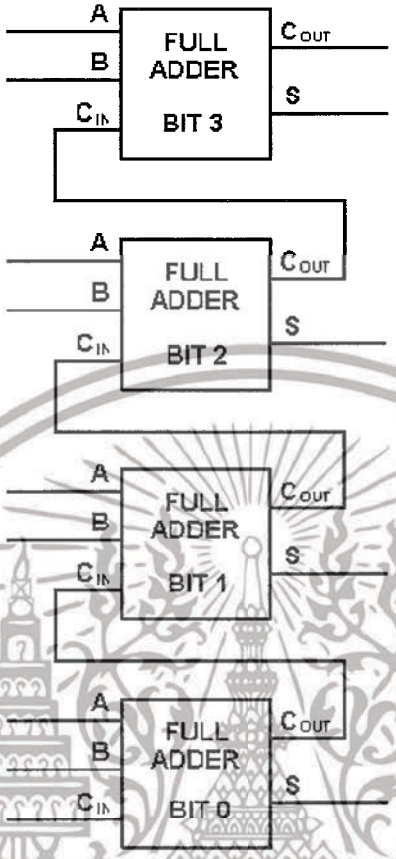
ลักษณะของวงจร Full-Adder สามารถต่อได้หลายแบบ คือการนำ Half-Adder 2 ตัว และ OR-Gate 1 ตัวมาต่อรวมกัน



รูปที่ 4.6 การต่อวงจร Full-Adder แบบใช้ Half-Adder 2 ชุดกับ OR-Gate 1 ตัว มาต่อรวมกัน

4.2.8 วงจร Full adder 4 bit

วงจรบวกหลายบิต(Multiple-bit Adder) คือการสร้างวงจรบวกให้สามารถรับอินพุทได้มากขึ้น กล่าวคือบวกเลขได้หลายบิตนั้นเอง หลักการสร้างก็ง่ายๆคือการนำ Full-Adder หลายๆตัวมาต่อรวมๆกันเป็นวงจรใหญ่ๆ เพื่อที่จะได้คำนวณให้ได้หลายบิตมากขึ้น โดยเมื่อมีการต่อวงจรแบบนี้แล้วจะมีการคำนวณการทด โดยในวงจรของเราจะใช้การบวกกันแบบ 4 bit นั่นเอง และใช้การทดแบบริปเปอร์ การทดแบบริปเปอร์(Ripple Carry Adder) คือการทด โดย เมื่อมีการต่อวงจรบวกหลายๆตัวจะต้องมีการส่งตัวทด(Carry) ไปให้กับวงจรบวกตัวต่อไปเพื่อคำนวณด้วย (โดยคำว่า Ripple นั้นแปลว่า ระลอก) เพราะฉะนั้น Ripple Carry Adder คือวงจรบวกที่มีการส่งตัวทดเป็นระลอกนั้นเอง ลักษณะวงจรคือ

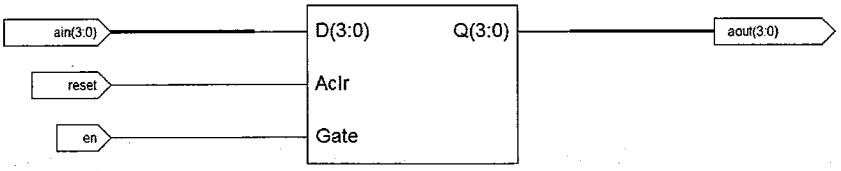


รูปที่ 4.7 รูปการต่อวงจร Full adder แบบ 4 bit

การต่อวงจรลักษณะนี้มีข้อเสียคือ จะทำให้วงจรมีการทำงานที่ช้าลงเมื่อมีการต่อลอจิกเกตมากขึ้นเพราะว่าจากสมการ Carry ของ Full-Adder จะเห็นได้ว่าจำเป็นต้องมีการรอ Carry-in ก่อนที่จะคำนวณต่อไปได้

4.2.9 วงจร Latch

Latch เป็นอุปกรณ์ที่มีลักษณะการทำงาน เมื่อสัญญาณen (enable) มีค่าลอจิกเป็น 1 จะอนุญาตให้ข้อมูลอินพุทไหลผ่านไปยังเอาต์พุท แต่ถ้าสัญญาณ en (enable) มีค่าลอจิกเป็น 0 จะคงค่าเดิมของเอาต์พุทเอาไว้



รูปที่ 4.8 ลักษณะวงจร Latch

4.2.10 วงจร XOR Gate

ในวงจรหลักเราจะใช้วงจร XOR ในการบวกกันของตัวเลข หรือสมาชิกในสนามจำกัด (Field Element) นิยามของ XOR Gate(Exclusive OR Gate) คือ Gate ที่ให้ Output เป็น 1 เมื่อ Input ต่างกัน และจะให้ Output เป็น 0 เมื่อมี input เหมือนกัน

XOR Gate มีตัวดำเนินการ คือ $\oplus$

ผลลัพธ์ ของ NAND Gate สามารถเขียนได้ดังสมการ

$$Y = A \oplus B = (A + B).(\overline{A} + \overline{B}) = \overline{A}B + A\overline{B}$$

ตารางค่าความจริง (Truth Table) ของ XOR Gate เป็นดังนี้

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

ตารางที่ 4.6 ตารางค่าความจริง (Truth Table) ของ XOR Gate

รูปที่ 4.9 สัญลักษณ์ XOR Gate

4.2.11 วงจรรวมทั้งหมด

วงจรนี้คือวงจรที่จะนำวงจรย่อยๆทุกๆส่วนมาต่อเข้าด้วยกันเพื่อให้ได้วงจรที่มีความสมบูรณ์และสามารถทำงานได้ตามที่เราต้องการ โดยในส่วนนี้จะเป็นส่วนที่ใหญ่ที่สุด โดยจะมีการกำหนดทั้งอินพุทและเอาต์พุททั้งหมดที่เข้าและออกจากวงจร สามารถดูโค้ดและรูปวงจรได้จากด้านล่าง

4.3 วงจรถอดรหัส

ในส่วนของวงจรถอดรหัสจะอาศัยหลักการทำงานที่คล้ายกับวงจรเข้ารหัสจะแตกต่างกันก็ตรงที่ค่า α ที่เรานำมาใช้ในการคูณกับสัญญาณอินพุทที่เรารับเข้ามาซึ่งก็คือสัญญาณที่ผ่านการเข้ารหัสแล้วนั่นเอง โดยค่า α ที่เราใช้ในการถอดรหัสนั้นก็คือค่า α ที่มีการ inverse แล้ว

แต่ในการเขียนเป็น Code VHDL นั้นเราจะต้องทำการมอดค่า α ที่เป็นกำลังลบให้เป็นบวกเสียก่อนจึงจะนำไปเข้าวงจร Full adder ได้ โดยเราก็ทำการสร้างตารางที่เก็บค่า α ที่มีการมอดแล้วเอาไว้ก่อนก่อนที่เราจะนำไปเข้าสมการ โดยสมการ inverse ค่า v_k กลับมาเป็นค่า v_i สามารถดูได้จากด้านล่าง

$$V_i = \sum_{k=0}^{N-1} v_k \alpha^{<-ik>_N} \qquad \text{for i} = 0,1,2,3,\ldots N\text{-}1$$

รากยกกำลังลบ	รากยกกำลังที่มอดให้เป็นบวก	ข้อมูลไบนารี
α^{-1}	α^{14}	1001
α^{-2}	α^{13}	1101
α^{-3}	α^{12}	1111
α^{-4}	α^{11}	1110
α^{-5}	α^{10}	0111
α^{-6}	α^{9}	1010
α^{-7}	α^{8}	0101
α^{-8}	α^{7}	1011
α^{-9}	α^{6}	1100
α^{-10}	α^{5}	0110
α^{-11}	α^{4}	0011
α^{-12}	α^{3}	1000
α^{-13}	α^{2}	0100
α^{-14}	α	0010
α^{-15}	α^{0}	0001

ตารางที่ 4.7 ตารางแสดงค่า α ที่มอดจากกำลังลบเป็นบวก

4.4 Code วงจรส่วนต่างๆ

4.4.1 Code วงจรแยก 8 bit เป็น 4 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reg_2 is
        Port (  a   : in  std_logic_vector(7 downto 0);
                clk   : in  std_logic;
                q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14,q15 : out  std_logic
_vector(3 downto 0);
                tr    : out std_logic);
end reg_2;

architecture Behavioral of reg_2 is
signal a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15 : std_logic_vector(3 downto 0);
signal v0,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15 : std_logic_vector(3 downto 0);
signal trF:std_logic;
type state_type is (s0,s1,s2,s3,s4,s5,s6,s7);
signal state : state_type;
begin
process(clk)
begin
if (clk'event and clk = '1') then

-- แบ่ง 8bit เป็น 4bit
        a1 <= a(3) & a(2) & a(1)  & a(0);
        a0 <= a(7) & a(6) & a(5) & a(4);
```

```
-- เลื่อน 4bit low
        a2  <= a0;
        a4  <= a2;
        a6  <= a4;
        a8  <= a6;
        a10 <= a8;
        a12 <= a10;
        a14 <= a12;

-- เลื่อน 4bit high
        a3  <= a1;
        a5  <= a3;
        a7  <= a5;
        a9  <= a7;
        a11 <= a9;
        a13 <= a11;
        a15 <= a13;
-- look output
case state is
        when s0 =>
                state <= s1;
                trF<='1';
        when s1 =>
                state <= s2;
                trF<='1';
        when s2 =>
                state <= s3;
                trF<='1';
        when s3 =>
                state <= s4;
```

```
            trF<='1';
        when s4 =>
            state <= s5;
            trF<='1';
        when s5 =>
            state <= s6;
            trF<='1';
        when s6 =>
            state <= s7;
            trF<='1';
        when s7 =>
            state <= s0;
            trF<='0';
            v0 <= a(7) & a(6) & a(5)  & a(4);
            v1 <= a(3) & a(2) & a(1)  & a(0);
            v2<=a0;
            v3<=a1;
            v4<=a2;
            v5<=a3;
            v6<=a4;
            v7<=a5;
            v8<=a6;
            v9<=a7;
            v10<=a8;
            v11<=a9;
            v12<=a10;
            v13<=a11;
            v14<=a12;
            v15<=a13;
end case;
```

```
end if;
end process;
        tr <= not((not trf) and clk);
        q0 <=v0;
        q1 <=v1;
        q2 <=v2;
        q3 <=v3;
        q4 <=v4;
        q5 <=v5;
        q6 <=v6;
        q7 <=v7;
        q8 <=v8;
        q9 <=v9;
        q10<=v10;
        q11<=v11;
        q12<=v12;
        q13<=v13;
        q14<=v14;
        q15<=v15;
end Behavioral;
```

4.4.2 Code วงจรชี้ค่าอินพุท

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity v1tov15_new is
        Port (    a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15 : in  std_logic _vector(3
downto 0);
```

```
        q   : out  std_logic _vector(3 downto 0);
        tr  : in  std_logic;
        ctr : in  std_logic _vector( 3 downto 0));
end v1tov15_new;

architecture Behavioral of v1tov15_new is
signal v0,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,v13,v14,v15 : std_logic_vector(3 downto 0);
begin
process(tr,ctr)
        begin
        if (tr<='0') then
                v0 <=a0;
                v1 <=a1;
                v2 <=a2;
                v3 <=a3;
                v4 <=a4;
                v5 <=a5;
                v6 <=a6;
                v7 <=a7;
                v8 <=a8;
                v9 <=a9;
                v10<=a10;
                v11<=a11;
                v12<=a12;
                v13<=a13;
                v14<=a14;
                v15<=a15;
        end if;

case ctr is
```

```
        when "0000" => q <= v0;
        when "0001" => q <= v1;
        when "0010" => q <= v2;
        when "0011" => q <= v3;
        when "0100" => q <= v4;
        when "0101" => q <= v5;
        when "0110" => q <= v6;
        when "0111" => q <= v7;
        when "1000" => q <= v8;
        when "1001" => q <= v9;
        when "1010" => q <= v10;
        when "1011" => q <= v11;
        when "1100" => q <= v12;
        when "1101" => q <= v13;
        when "1110" => q <= v14;
        when "1111" => q <= v15;
        when others => q <= "0000";
end case;
end process ;
end Behavioral;
```

4.4.3 Code วงจร ROM ชี้ค่าเลขชี้กำลัง α

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity rom_3 is
        Port (  address : in std_logic_vector(3 downto 0);
                data : out std_logic_vector(3 downto 0) );
```

```
end entity rom_3;

architecture behavioral of rom_3 is
type mem is array ( 0 to 2**4 - 2) of std_logic_vector(3 downto 0);
constant my_rom : mem := (
        0 => "1111",
        1 => "0001",
        2 => "0100",
        3 => "0010",
        4 => "1000",
        5 => "0101",
        6 => "1010",
        7 => "0011",
        8 => "1110",
        9 => "1001",
        10 => "0111",
        11 => "0110",
        12 => "1101",
        13 => "1011",
        14 => "1100");
begin
process (address)
        begin
        case address is
                when "0001" => data <= my_rom(0);
                when "0010" => data <= my_rom(1);
                when "0011" => data <= my_rom(2);
                when "0100" => data <= my_rom(3);
                when "0101" => data <= my_rom(4);
                when "0110" => data <= my_rom(5);
```

```
            when "0111" => data <= my_rom(6);
            when "1000" => data <= my_rom(7);
            when "1001" => data <= my_rom(8);
            when "1010" => data <= my_rom(9);
            when "1011" => data <= my_rom(10);
            when "1100" => data <= my_rom(11);
            when "1101" => data <= my_rom(12);
            when "1110" => data <= my_rom(13);
            when "1111" => data <= my_rom(14);
            when others => data <= "0000";
        end case;
end process;
end architecture behavioral;
```

4.4.4 Code วงจร ROM ชี้ค่าข้อมูลที่เป็นไบนารี

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity rom_2 is
        Port (  address : in std_logic_vector(3 downto 0);
                data : out std_logic_vector(3 downto 0) );
end entity rom_2;

architecture behavioral of rom_2 is
type mem is array ( 0 to 2**4 - 1) of std_logic_vector(3 downto 0);
constant my_Rom : mem := (
        0  => "0001",
        1  => "0010",
```

```
        2  => "0100",
        3  => "1000",
        4  => "0011",
        5  => "0110",
        6  => "1100",
        7  => "1011",
        8  => "0101",
        9  => "1010",
        10 => "0111",
        11 => "1110",
        12 => "1111",
        13 => "1101",
        14 => "1001",
        15 => "0001");
begin
process (address)
        begin
        case address is
                when "0000" => data <= my_rom(0);
                when "0001" => data <= my_rom(1);
                when "0010" => data <= my_rom(2);
                when "0011" => data <= my_rom(3);
                when "0100" => data <= my_rom(4);
                when "0101" => data <= my_rom(5);
                when "0110" => data <= my_rom(6);
                when "0111" => data <= my_rom(7);
                when "1000" => data <= my_rom(8);
                when "1001" => data <= my_rom(9);
                when "1010" => data <= my_rom(10);
                when "1011" => data <= my_rom(11);
```

```
            when "1100" => data <= my_rom(12);
            when "1101" => data <= my_rom(13);
            when "1110" => data <= my_rom(14);
            when "1111" => data <= my_rom(15);
            when others => data <= "0000";
        end case;
end process;
end architecture behavioral;
```

4.4.5 Code วงจร OR Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity or_gate is
        Port (  in1 : in  std_logic;
                in2 : in  std_logic;
                outq : out  std_logic);
end or_gate;

architecture Behavioral of or_gate is
begin
        outq <= in1 or in2;
end Behavioral;
```

4.4.6 Code วงจร Half adder

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

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity half_adder is
        Port (  a : in  std_logic;
                b : in  std_logic;
                sum : out  std_logic;
                carry : out  std_logic);
end half_adder;

architecture Behavioral of half_adder is
begin
process(a,b)
variable x,y : std_logic;
begin
        x := a;
        y := b;
        sum <= a xor b ;
        carry <= a and b ;
end process;
end Behavioral;
```

4.4.7 Code วงจร Full adder 1 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity full_adder is
        Port (  ain : in std_logic;
                bin : in std_logic;
```

```
                cin : in std_logic;
                sum_out : out std_logic;
                arry_out : out std_logic);
end full_adder;

architecture structural of full_adder is
--component declaration
component half_adder
        generic(gate_delay : time);
        port(   a: in std_logic;
                b: in std_logic;
                sum: out std_logic;
                carry : out std_logic);
end component;
component or_gate
        port(   in1 : in std_logic;
                in2 : in std_logic;
                outq: out std_logic);
end component;

-- signal declarations
signal n_sum,n_carry1,n_carry2 :std_logic;
begin

-- component instantiation
u1 : half_adder
        generic map (gate_delay => 2 ns)
        port map (      a => ain,
                        b => bin,
                        sum => n_sum,
```

```
                    carry => n_carry1);
u2 : half_adder
        generic map (gate_delay => 2 ns)
        port map (      a => n_sum,
                        b => cin,
                        sum => sum_out,
                        carry => n_carry2);
u3 : or_gate
        port map (      in1 =>n_carry1,
                        in2 => n_carry2,
                        outq => carry_out);
end structural;
```

4.4.8 Code ของ Full adder 4 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity add4par is
        Port(   cin : in std_logic;
                a : in std_logic_vector(3 downto 0);
                b : in std_logic_vector(3 downto 0);
                sum : out std_logic_vector(3 downto 0));
end add4par;

architecture arch1 of add4par is

 -- component declaration
component full_adder
```

```
        port (  ain : in std_logic;
                bin : in std_logic;
                cin : in std_logic;
                sum_out : out std_logic;
                carry_out : out std_logic);
 end component;
 -- define signal for internal carry bit
signal c1,c2,c3,c4: std_logic;

begin
 -- Four Component Instantiation Statements
adder1: full_adder
        port map (      ain => a(0),
                        bin => b(0),
                        cin => c4,
                        carry_out => c1,
                        sum_out => sum(0));
adder2: full_adder
        port map (      ain => a(1),
                        bin => b(1),
                        cin => c1,
                        carry_out => c2,
                        sum_out => sum(1));
adder3: full_adder
        port map (      ain => a(2),
                        bin => b(2),
                        cin => c2,
                        carry_out => c3,
                        sum_out => sum(2));
adder4: full_adder
```

```
        port map (      ain => a(3),
                        bin => b(3),
                        cin => c3,
                        carry_out => c4,
                        sum_out => sum(3));
                        end arch1;
```

4.4.9 Code วงจร Latch

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity latch is
        Port (  ain   : in  std_logic_vector(3 downto 0);
                en    : in  std_logic;
                aout  : out std_logic_vector (3 downto 0);
                reset : in  std_logic);
end latch;

architecture Behavioral of latch is
begin
process(en,reset)
        begin
        if   (reset = '1') then
                aout<="0000";
                elsif  ( en='1') then
                aout <= ain;
        end if;
end process;
```

```
end Behavioral;
```

4.4.10 Code วงจร XOR Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity plus is
        Port (  ain : in  std_logic_vector (3 downto 0);
                bin : in  std_logic_vector (3 downto 0);
                out_q : out  std_logic_vector (3 downto 0));
end plus;

architecture Behavioral of plus is
begin
process (ain,bin)
        begin
                out_q <= ain xor bin ;
end process;
end Behavioral;
```

4.4.11 Code วงจรรวมทั้งหมด

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity gaga is
        Port (  address1 : in  std_logic_vector (7 downto 0);
```

```
                address2 : in  std_logic_vector (3 downto 0);
                address3 : in  std_logic_vector (3 downto 0);
                ctr      : in  std_logic_vector (3 downto 0);--เพิ่ม
                clk_2    : in  std_logic;--ลองทำเพิ่มเอง
                clk1     : in  std_logic;
                reset1   : in  std_logic;--ลองทำเพิ่มเอง
                sum_out  : inout  std_logic_vector (3 downto 0));
end gaga;

architecture Behavioral of gaga is
--component declaration
component rom_3
        port (  address : in std_logic_vector(3 downto 0);
                data    : out std_logic_vector(3 downto 0));
end component;

component add4par
        port(   a : in std_logic_vector(3 downto 0);
                b : in std_logic_vector(3 downto 0);
                sum : out std_logic_vector(3 downto 0));
end component ;

--เพิ่มรอมใหม่ชี้อัลฟากลับ
component rom_2
        port (  address : in std_logic_vector(3 downto 0);
                data : out std_logic_vector(3 downto 0) );--signal declaration
end component;

--เพิ่ม latch ----------------------------------------
component latch
```

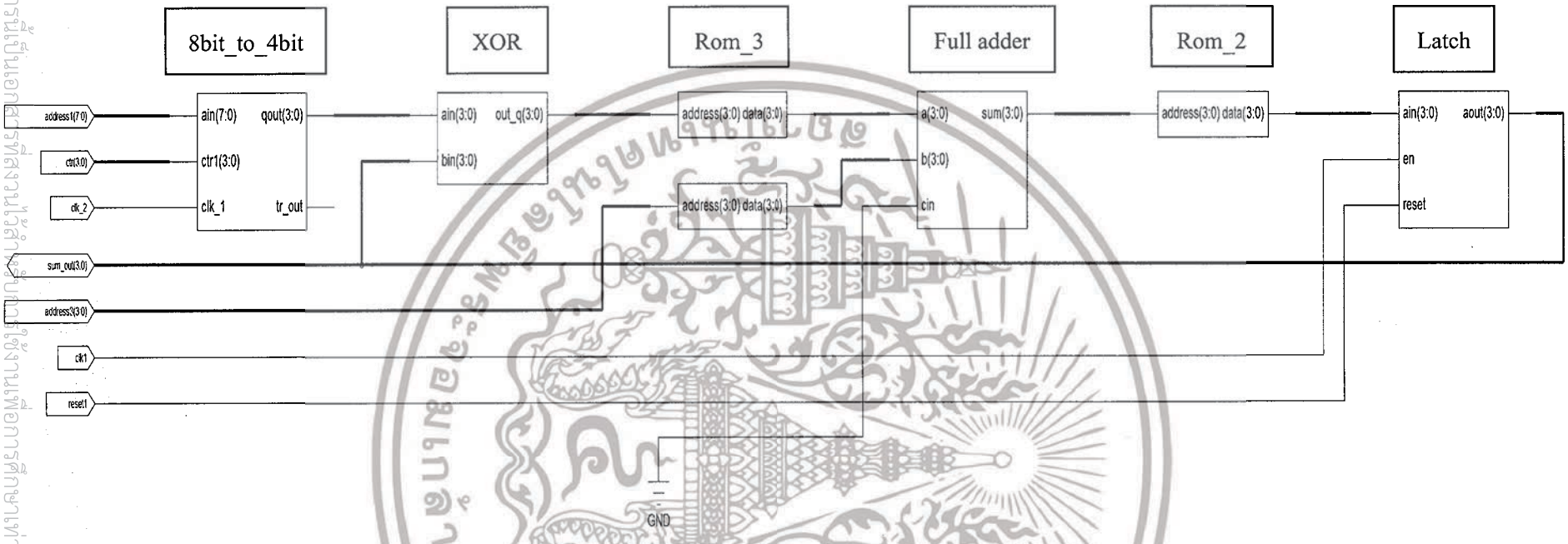
```
        Port (    ain   : in  std_logic_vector (3 downto 0);
                  en    : in  std_logic;
                  aout  : out  std_logic_vector (3 downto 0);
                  reset : in  std_logic);
end component;
-----------------------------------------------
--เพิ่ม xor
component plus
        Port (    ain   : in  std_logic_vector (3 downto 0);
                  bin   : in  std_logic_vector (3 downto 0);
                  out_q : out  std_logic_vector (3 downto 0));
end component;
-----------------------------------------------
--เพิ่ม 8bit to 4bit
component yoyo_new
        Port      (ain  : in  std_logic_vector (7 downto 0);
                  clk_1: in std_logic;
                  ctr1  : in std_logic_vector (3 downto 0);
                  qout : out  std_logic_vector (3 downto 0));
end component;
-----------------------------------------------
signal data1,data2,data3,data4,data5,data6 : std_logic_vector(3 downto 0);
begin
-- component instantiation

u1 : rom_3
        port map (        address => data2,
                          data    => data3);
u2 : rom_3
        port map (        address => address3,
```

```
                    data    => data4);
u3 : add4par
        port map (  a   => data3,
                    b   => data4,
                    sum     => data5);
u4 : rom_2
        port map (  address => data5,
                    data    => data6);
u5 : latch
        port map (  ain   => data6,
                    en    => clk1,
                    aout  => sum_out,
                    reset =>  reset1 );
u6 : plus
        port map (  ain   => data1,
                    bin   => sum_out,
                    out_q =>  data2);
u7 : yoyo_new
        port map (  ain   => address1,
                    clk_1 => clk_2,
                    ctrl  => ctr,
                    qout  => data1  );
end Behavioral;
```

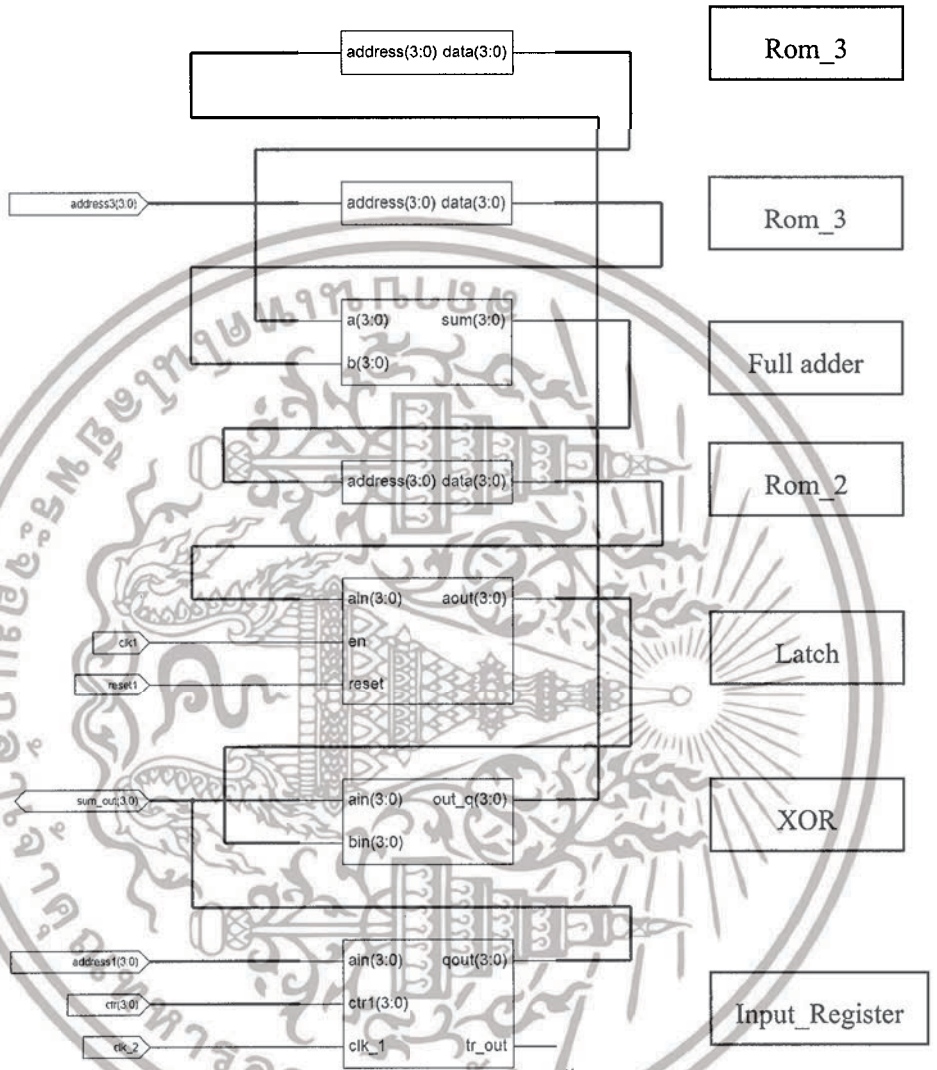
บทที่ 5

ผลการ Simulation วงจรในส่วนต่างๆ

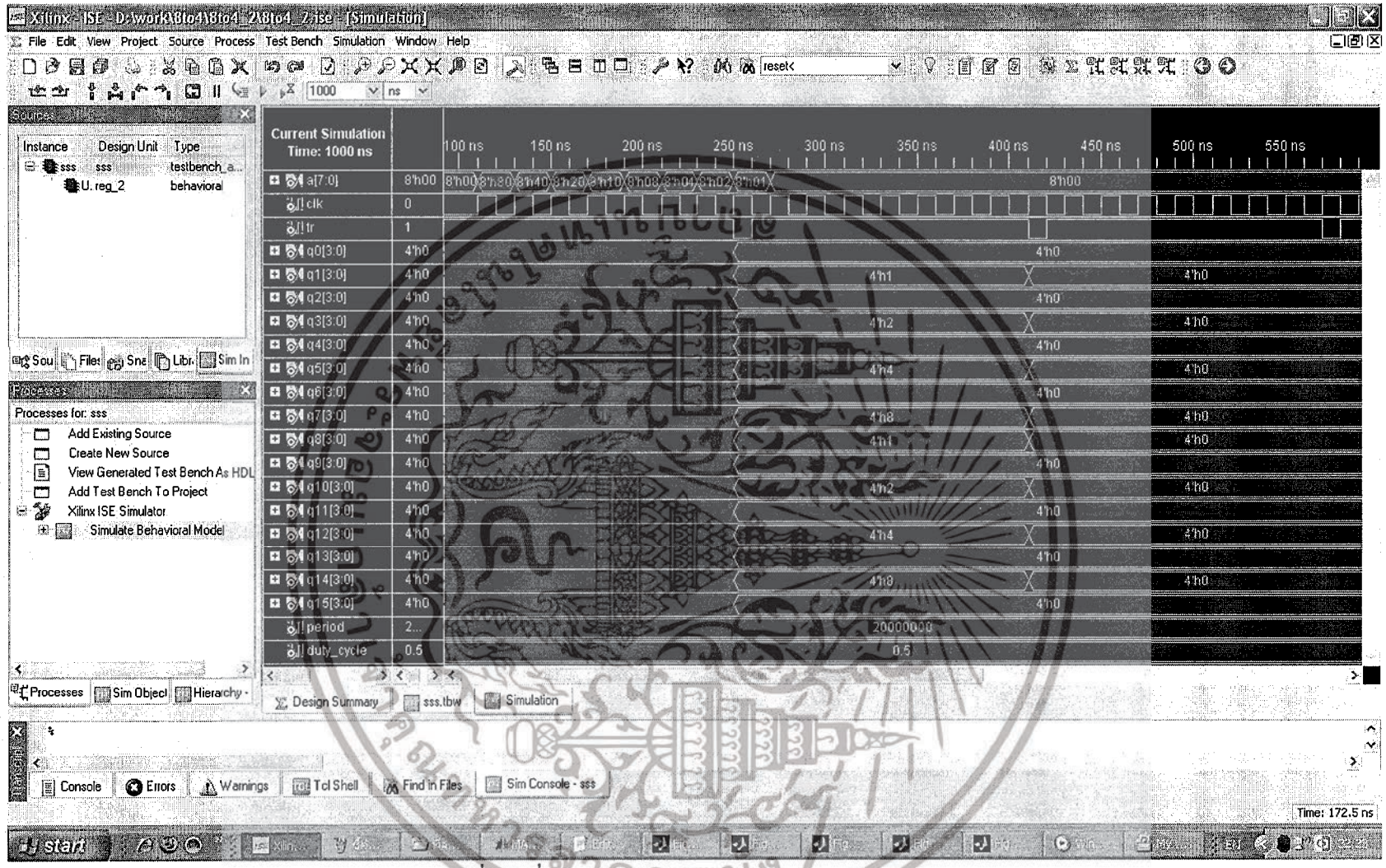
เมื่อเขียน Source Code ของภาษา VHDL ได้แล้ว จะต้องนำมา Compile ให้ผ่านก่อน จึงสามารถทำการ simulation ได้ ในการ simulation วงจรในส่วนต่างๆนั้นก็เพื่อจะดูว่าวงจรในแต่ละส่วนสามารถทำงานได้ถูกต้องตามที่เราคิดเอาไว้หรือไม่



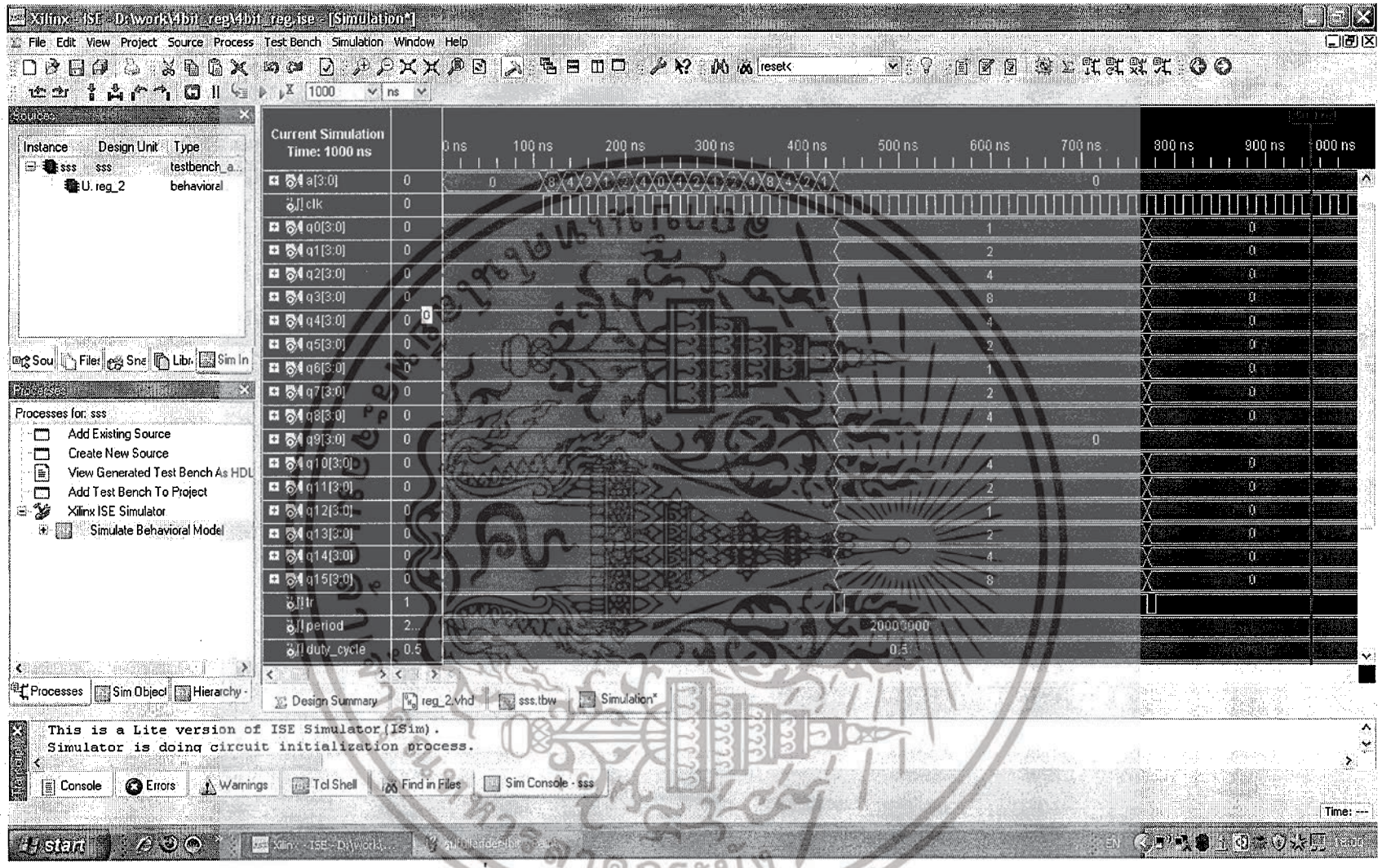
รูปที่ 5.1 วงจรเข้ารหัสโดยรวมทั้งหมด



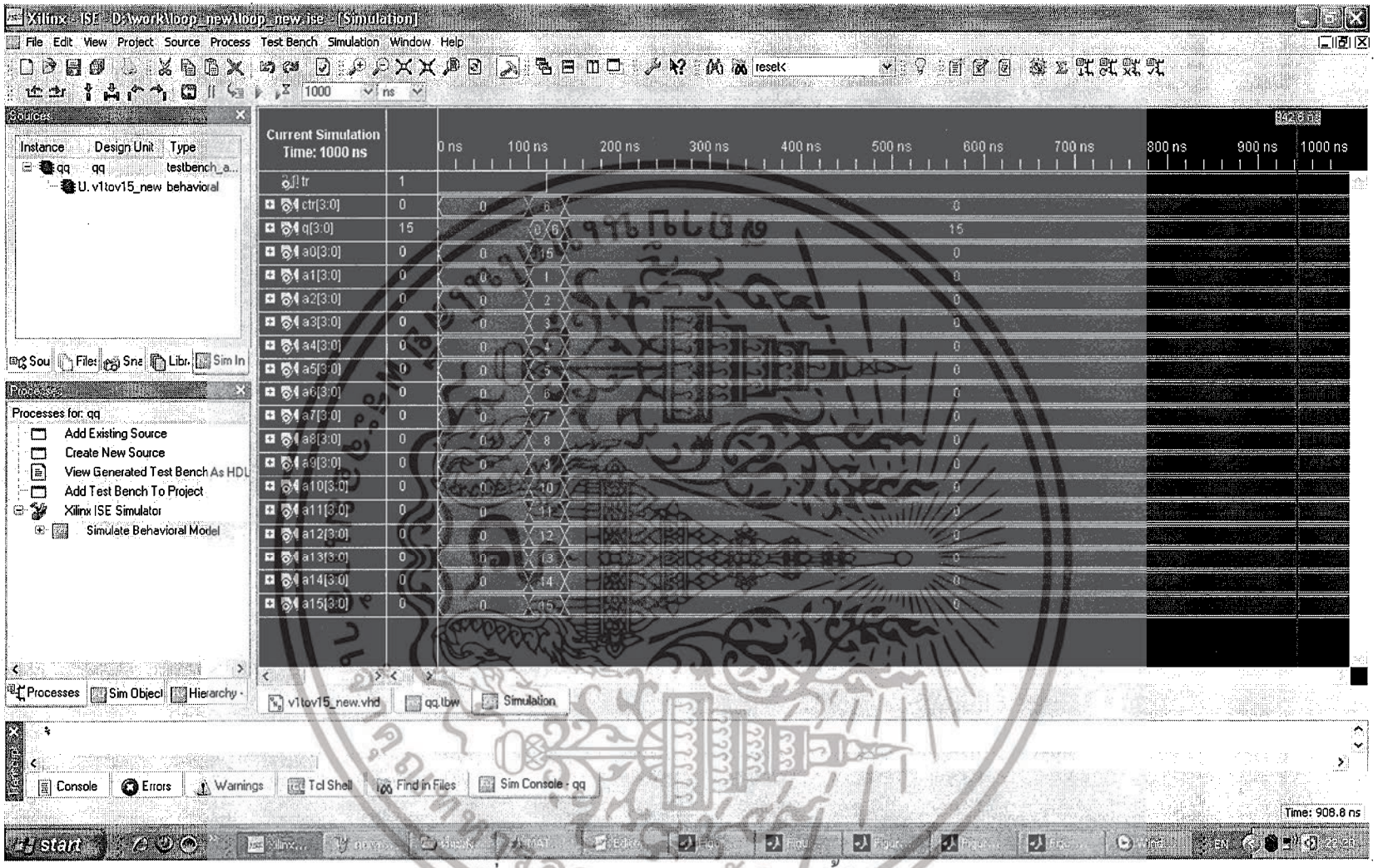
รูปที่ 5.2 วงจรถอดรหัสโดยรวมทั้งหมด



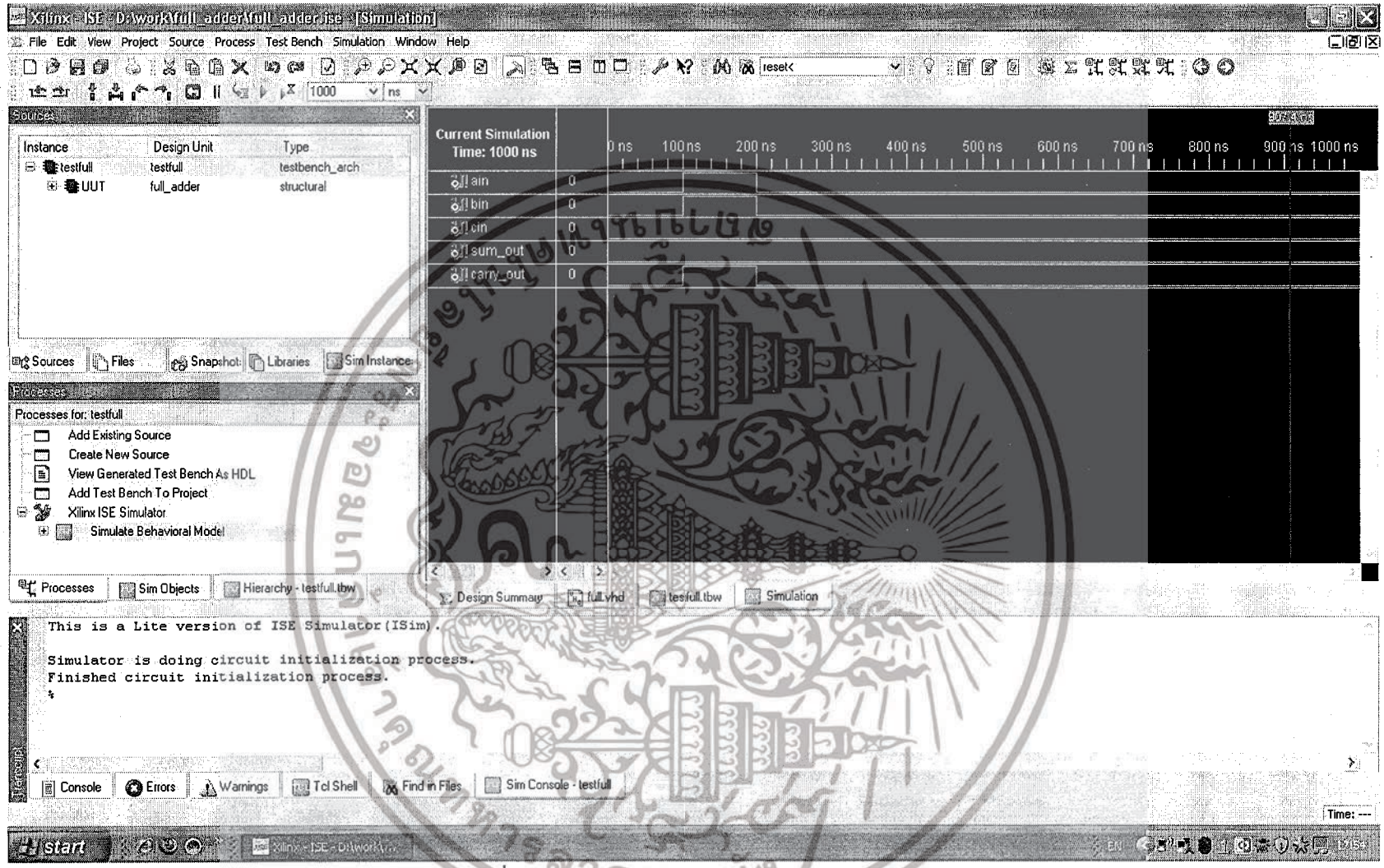
รูปที่ 5.3 ที่ สัญญาณ simulation ของวงจร 8bit_to_4bit



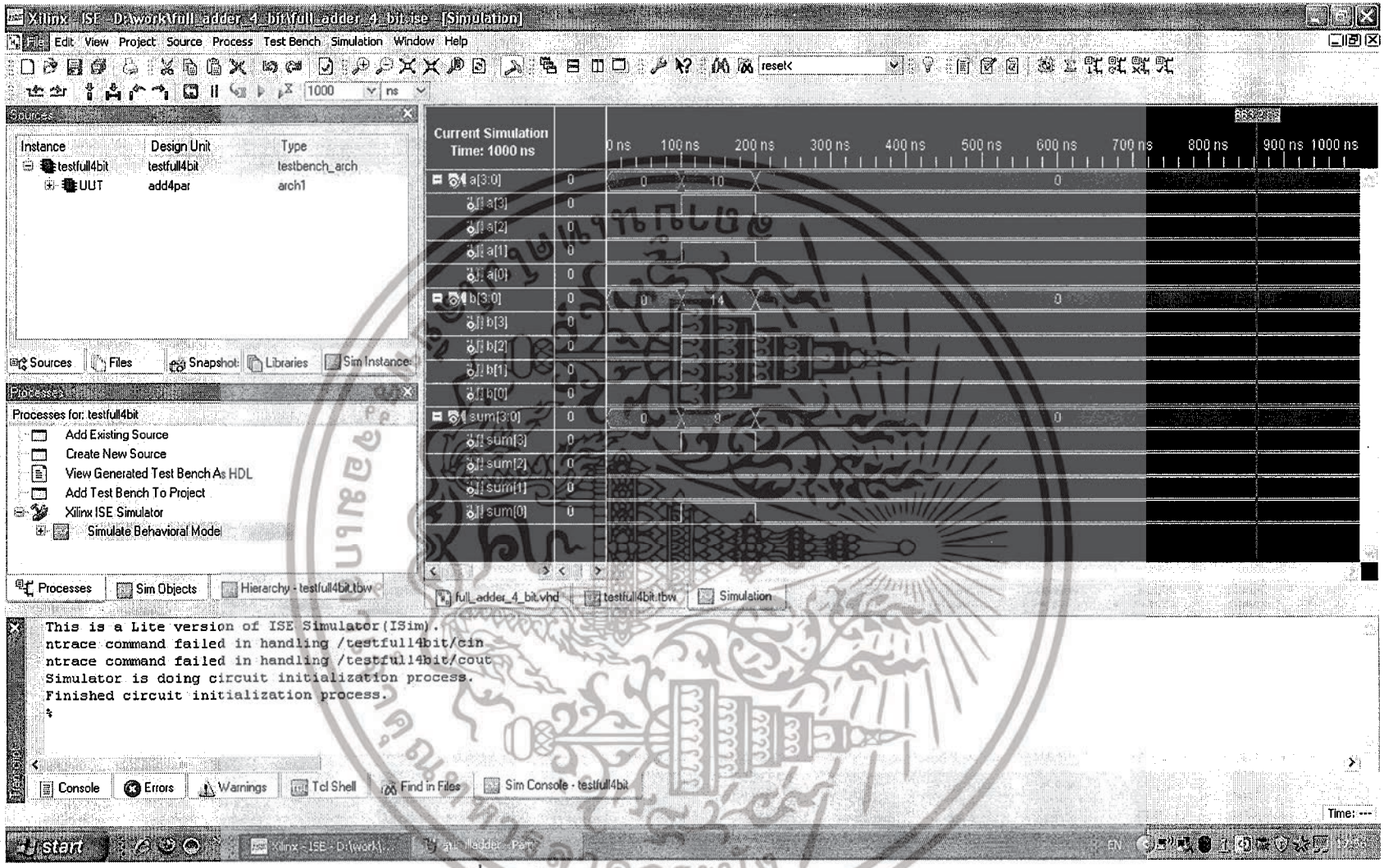
รูปที่ 5.4 สัญญาณ simulation ของวงจร Shift ค่า input



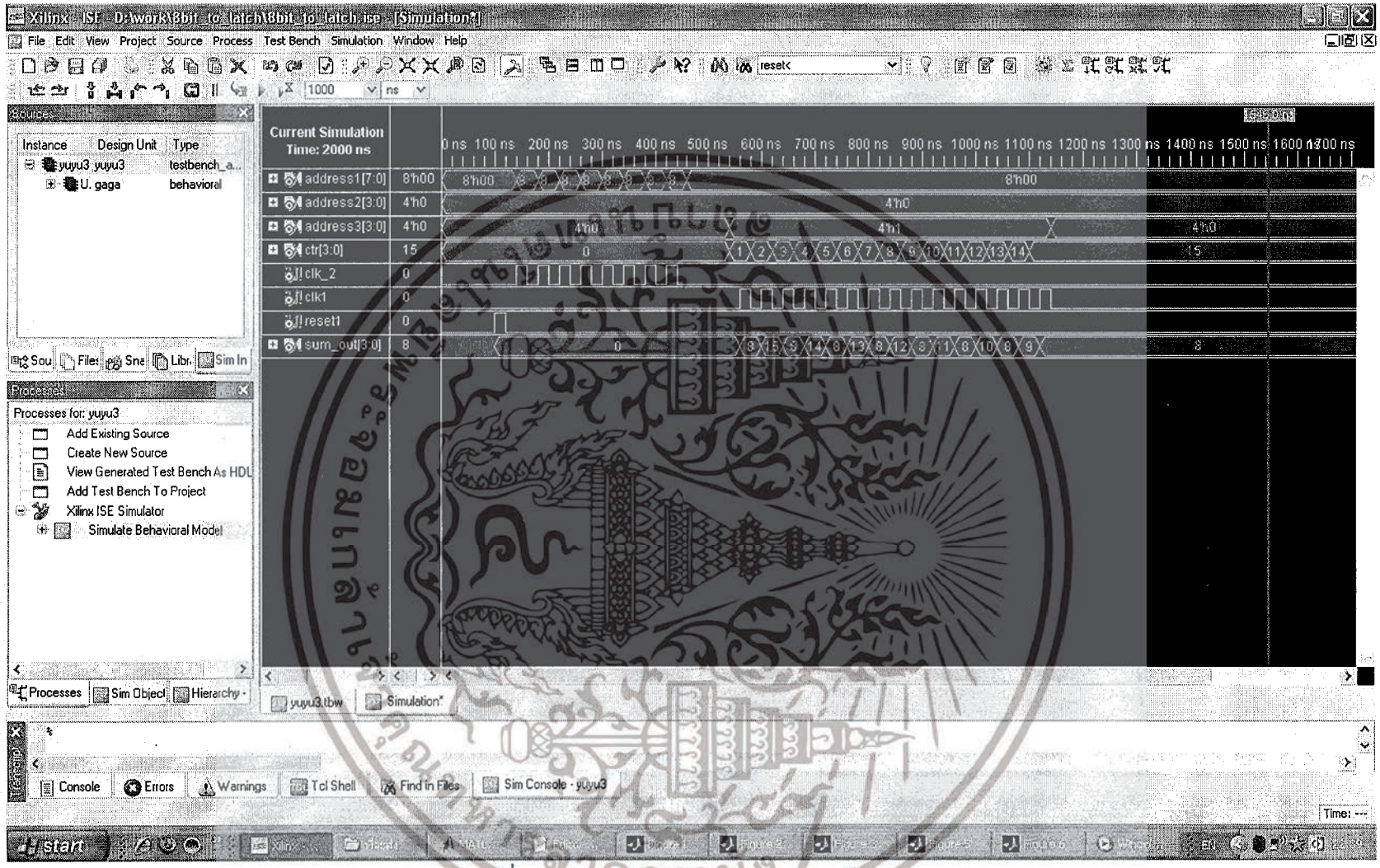
รูปที่ 5.5 สัญญาณ simulation ของวงจรชี้ค่า input



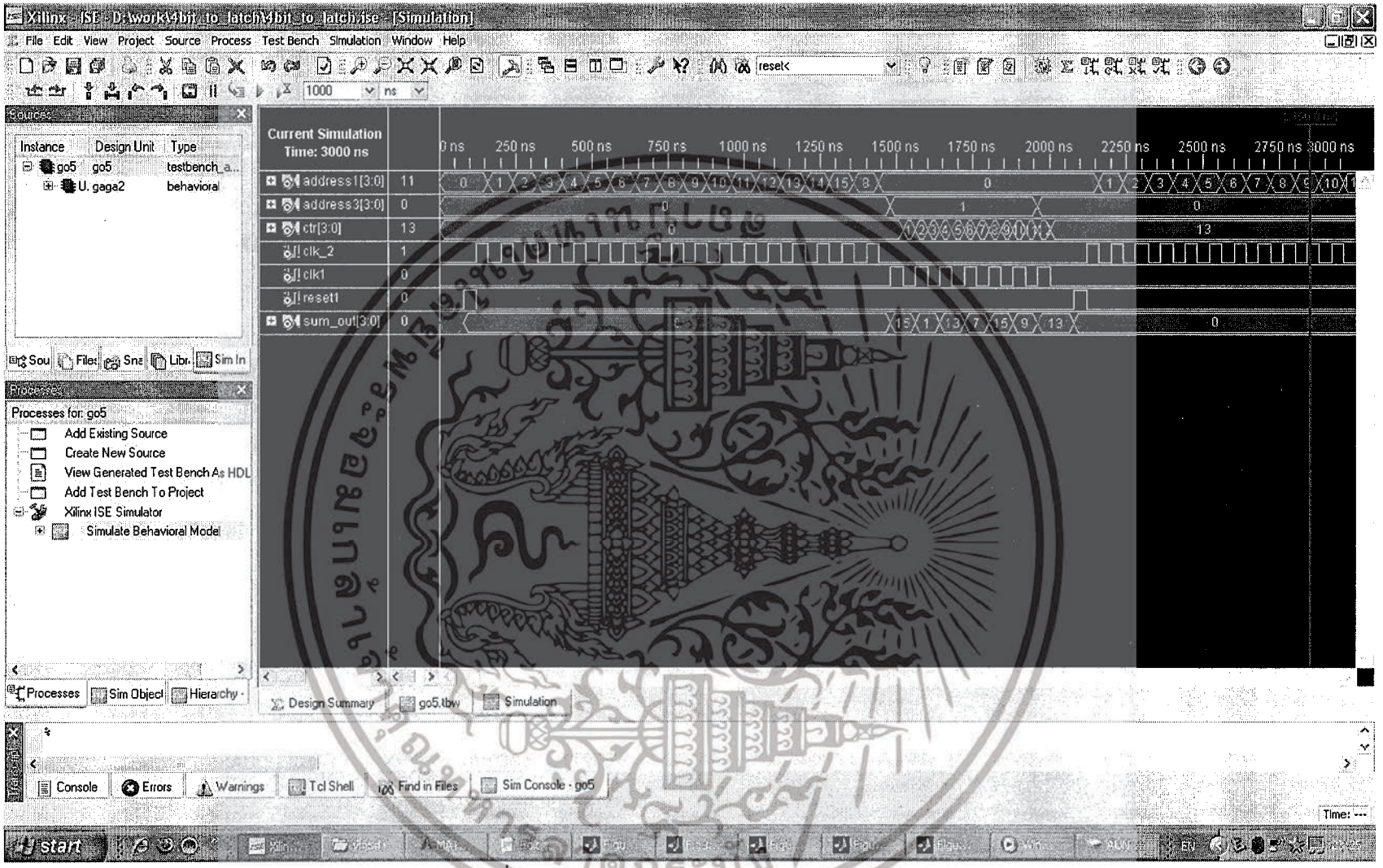
รูปที่ 5.6 สัญญาณ simulation ของวงจร full adder



รูปที่ 5.7 สัญญาณ simulation ของวงจร full adder 4 bit



รูปที่ 5.8 สัญญาณ simulation ของวงจรเข้ารหัส



รูปที่ 5.8 สัญญาณ simulation ของวงจรถอดรหัส

บทที่ 6
สรุปและวิจารณ์

6.1 ส่วนของ Matlab

จากการศึกษาและทดลองโดยใช้โปรแกรม matlab นั้น จะใช้ในการพิสูจน์สมการ Finite Field Fourier Transform ว่าเมื่อรับเสียงเข้ามา ผ่านสมการแล้ว จะสามารถเข้ารหัสและถอดรหัสได้จริงหรือไม่ จากผลการทดลองที่ได้จากการ Simulation โดยโปรแกรม matlab ดูแล้ว สรุปว่าสมการ Finite Field Fourier Transform สามารถนำไปใช้ในการเข้ารหัสและถอดรหัส สัญญาณเสียงได้จริง

6.2 ส่วนของ VHDL

จากการศึกษาและทดลองทำการออกแบบระบบดิจิตอลด้วย VHDL นั้น พบว่าขั้นตอนในการออกแบบมีดังนี้คือ

1. แนวความคิดในการออกแบบ
2. การออกแบบตัวบรรยายภาษา VHDL
3. การออกแบบทางวงจรตรรกะและการสังเคราะห์
4. การออกแบบทางกายภาพและการสร้างเครื่องต้นแบบ
5. การจำลองและการทดสอบ

ในแต่ละขั้นตอนจะให้ผลลัพธ์ที่เป็นข้อมูลให้กับขั้นต่อไป เพราะฉะนั้นการออกแบบไม่สามารถทำข้ามขั้นตอนได้ ในขั้นตอนการออกแบบตัวบรรยายภาษา VHDL นั้นเป็นขั้นตอนที่มีความยืดหยุ่นสูงพอสมควร มีรูปแบบต่างๆกันในการรออกแบบ ซึ่งสามารถแบ่งได้ดังนี้

1) รูปแบบ Top-Down Design รูปแบบนี้มีลักษณะคล้ายกับการออกแบบ Software โดยการแบ่งตัวระบบออกเป็นโมเดล ซึ่งจะทำการออกแบบ Component ที่ไม่ทราบการทำงานภายใน แต่ทราบว่ามี I/O เป็นอย่างไร จากนั้นจึงทำการออกแบบส่วนของโมเดล

2) รูปแบบ Bottom-Up Design รูปแบบนี้จะกลับกันกับแบบ Top-Down Design เพราะจะออกแบบทีละโมเดลก่อน จากนั้นนำโมเดลทั้งหมดมาประกอบเข้าด้วยกัน ผลลัพธ์จากการออกแบบจะเป็นรูปแบบ Flatten Design

3) รูปแบบ Flatten Design รูปแบบนี้จะไม่มีการแบ่งแยกการทำงานออกเป็นโมเดลจะมองระบบทั้งหมด แล้วทำการออกแบบ รูปแบบนี้เหมาะสำหรับการออกแบบระบบที่ไม่มีการทำงานซ้ำซ้อนกัน และเป็นระบบที่ไม่มีความซับซ้อนมากนัก

6.3 ปัญหาและวิธีแก้ไข

6.3.1 ปัญหาและวิธีแก้ไขในส่วน Matlab

ในส่วนของโปรแกรม matlab นั้น เมื่อเรานำเสียงเข้ามาแล้วความถี่นั้นจะมีหลายค่ามาก ขึ้นกับความถี่ sampling ที่เราใช้กับเวลาที่เราบันทึกเสียงด้วย ถ้าเราพิสูจน์เสียงเป็นจำนวนมากแล้ว ทำให้โปรแกรม matlab ใช้เวลานานมาก กว่าจะ Simulate แล้วเสร็จ ดังนั้นจึงแก้ไขโดย ลดจำนวนข้อมูลที่เข้ามาลงด้วยวิธีการตัดข้อมูลช่วยหนึ่งเข้ามาพิสูจน์

การเขียนโปรแกรมบางทีเมื่อไม่รู้จะเขียนต่อไปอย่างแล้วก็จะเสียเวลาเป็นอย่างมาก แก้ไขโดยการช่วยกันคิดทั้งกลุ่ม แต่ถ้ายังคิดไม่ออกก็จะไปถามอาจารย์ที่ปรึกษา

6.3.2 ปัญหาและวิธีแก้ไขในส่วน VHDL

ในส่วนของการเขียนโปรแกรมภาษา VHDL ในตอนเริ่มต้นเขียนเราค่อยข้างจะพบปัญหามากเนื่องจากเรายังไม่ค่อยเข้าใจโครงสร้างของภาษา ทำให้เมื่อเราเขียนโปรแกรมเสร็จแล้วทำการ Compile ไม่ผ่าน แต่เมื่อเราใช้ไปสักพักจะเริ่มคุ้นเคยมากขึ้นทำการเขียนได้ไวขึ้น

สำหรับโปรแกรมที่เราทำการเขียนขึ้นมานั้นจะยังคงมีปัญหาในการทำงานอยู่ ก็คือเมื่อเรารับค่าอินพุทที่เป็นข้อมูล “ 0000 ” เข้ามานั้นจะทำให้วงจรทำงานผิดพลาดเนื่องจากเราใช้เลขชี้กำลังอัลฟ่ามาทำการคำนวณแต่ข้อมูลที่เป็น “ 0000 ” นั้น ไม่มีเลขชี้กำลังอัลฟ่า เราจึงต้องกำหนดข้อยกเว้นพิเศษให้กับมัน โดยต้องให้ค่าเอาต์พุทออกมาเป็น “ 0000 ” ทันทีเมื่อเจอกรณีที่อินพุทเป็น “ 0000 ” วงจรจึงจะสามารถทำงานได้อย่างถูกต้อง

เอกสารอ้างอิง

1. ชำนาญ ปัญญาใส, วัชรากร หนูทอง, " ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล ", กรุงเทพฯ, ซีเอ็ดยูเคชั่น, 2547.

2. รัตนาพร แขชัย, วุฒิชัย สุภวัฒนากูล, " ระบบการป้องกันการลอบดักฟังสัญญาณเสียงแบบดิจิตอล voice scramble descramble ", 2539.

3. ลัญฉกร วุฒิสิทธิกุลกิจ, " MATLAB การประยุกต์ใช้งานทางวิศวกรรมไฟฟ้า ", กรุงเทพฯ, สำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย, 2547.

4. มนัส สังวรศิลป์, วรรัตน์ ภัทรอมรกุล, " คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์ ", อินโฟเพรส, 2543.

5. โกศล ตราชู, " การปกปิดข้อมูลด้วยการเข้ารหัสบล็อคโค้ดและสัญญาณรบกวนแบบลำดับสุ่มเทียมที่สร้างบน FPGA Data scramble based on block code and pseudo random sequence implemented on FPGA ", 2544.

6. Douglas L. Perry, San Ramon, " VHDL ", McGraw-Hill, 1991.

7. Jayaram Bhasker, " A VHDL Primer ", Prentice-hall, 1992.

8. Steve Carlson, " Introduction to HDL-Base Desing Using VHDL ", Synopsys Inc., 1991.

กิตติกรรมประกาศ

การศึกษารวบรวมข้อมูลและการทดลอง เพื่อทำรายงานเล่มนี้ให้สำเร็จนั้นลุล่วงไปได้ด้วยดี เพราะได้ความกรุณาจากคณะอาจารย์ โดยเฉพาะอย่างยิ่งอาจารย์ที่ปรึกษา คือ ดร. สมศักดิ์ ชุมช่วย ที่ให้ความรู้และคำแนะนำมาตลอด รวมทั้งต้องกล่าวขอบคุณรุ่นพี่และเพื่อนๆ ที่ร่วมงาน และให้ความช่วยเหลือด้วยดีเสมอมา และขอขอบคุณบิดามารดาที่ได้ให้กำเนิดและเลี้ยงดูอบรมสั่งสอนจนทำให้ผมมีวันนี้

นาย รณชิต สีไว

นาย วิชิต ชอบจิตต์

นาย สุทธิธรรม ทิพวรนิก

ผู้จัดทำ