การเข้ารหัสสัญญาณโทรศัพท์

TELEPHONE SIGNAL ENCRYPTION

นายอาคม ทองทวีผล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมการวัดคุม
ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549

TELEPHONE SIGNAL ENCRYPTION

AKOM THONGTHAWEEPHON

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2006

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ การเข้ารหัสสัญญาณโทรศัพท์

TELEPHONE SIGNAL ENCRYPTION

นักศึกษาผู้จัดทำ นายอาคม ทองทวีผล **รหัสนักศึกษา** 46010957

ปริญญา วิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมการวัดคุม

ปีการศึกษา 2549

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
รศ.ดร. ฟูศักดิ์ ชีวสุวิทย์	

ภาควิชารับรองแล้ว

(รศ.ประภาษ อุดคกิมาพันธุ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์ การเข้ารหัสสัญญาณโทรศัพท์

TELEPHONE SIGNAL ENCRYPTION

นักศึกษาผู้จัดทำ นายอาคม ทองทวีผล **รหัสนักศึกษา** 46010957

อาจารย์ที่ปรึกษา รศ.ดร. ฟูศักดิ์ ชีวสุวิทย์

ปีการศึกษา 2549

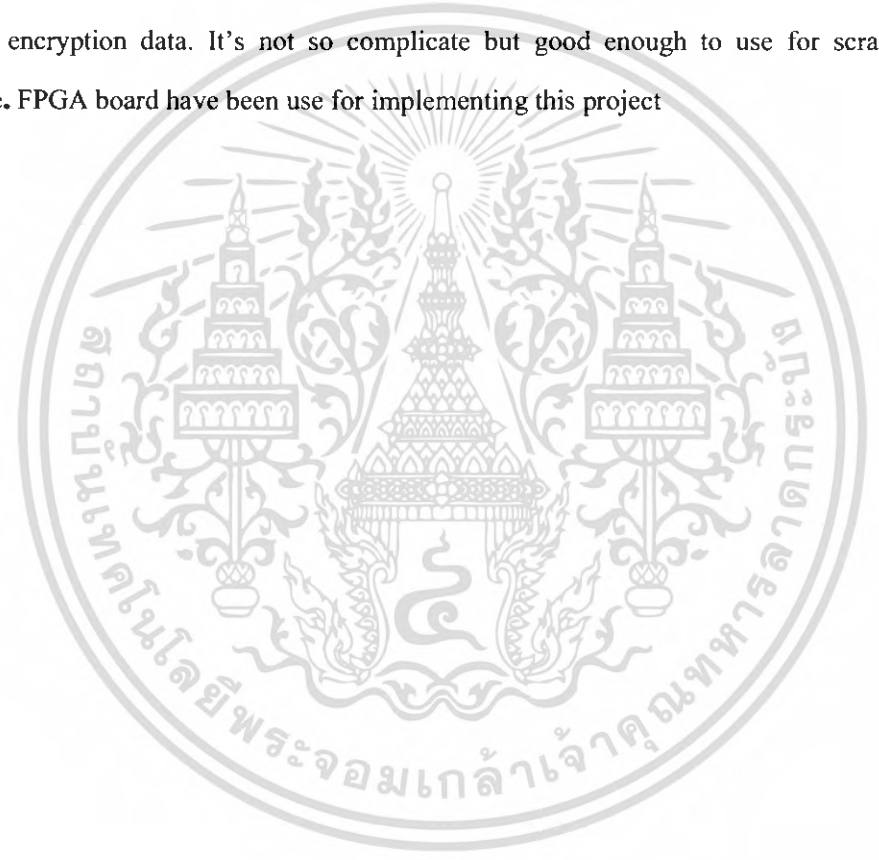
บทคัดย่อ

บทความนี้ ได้นำเสนอการเข้ารหัสข้อมูล 8 บิท ด้วยวิธีแบบ Linear Block Code เพื่อใช้ในการป้องกันข้อมูลไม่ให้ผู้ที่ไม่ได้รับอนุญาตที่จะรับข้อมูลนั้นสามารถเข้าถึงข้อมูลได้ซึ่งการเข้ารหัสแบบ Linear Block Code นั้น เป็นรูปแบบในการเข้ารหัสที่น่าสนใจ ไม่ยุ่งยากมาก แต่มีความสามารถเพียงพอที่จะใช้เป็นตัวเข้ารหัสข้อมูล และใช้ถอดรหัสข้อมูล โดยได้ออกแบบบนบอร์ด FPGA

Thesis Title	Telephone Signal Eneryption
Authors	Mr. Akom Thongthaweephon
Thesis Advisor	Assoc. Prof. Dr. Fusak Cheevasuvit
Year	2006

ABSTRACT

This article performs the 8 bits encryption data by Linear Block Code for protecting data from someone who have no permission to receive this data. Linear Block Code is very interested to use for encryption data. It's not so complicate but good enough to use for scramble and descramble. FPGA board have been use for implementing this project



กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงลงได้ ด้วยความช่วยเหลือและการให้คำปรึกษา อีกทั้งคำแนะนำรวมทั้งชี้แนะแนวทางเป็นอย่างดีตลอดระยะเวลาจากอาจารย์ที่ปรึกษา คณะผู้ทำการทดลองขอขอบคุณ รศ.ดร.ฟูศักดิ์ ชีวสุวิทย์ ที่ได้ให้คำแนะนำมาในการทำวิจัยมาโดยตลอดและ ผู้ให้คำแนะนำหลายๆท่านที่ไม่ได้มีส่วนได้ส่วนเสียใด ๆ แต่ก็ได้ให้การช่วยเหลือมาด้วยดีโดยตลอด ซึ่งเป็นประโยชน์อย่างมากในการทำปริญญานิพนธ์ ขอขอบคุณพี่ ๆ และเพื่อน ๆ ในห้องโปรเจคท์ทุกคนที่คอยช่วยเหลือในทุกๆ ด้านตลอดมา

ขอขอบคุณพระคุณ อาจารย์ภาควิชาวิศวกรรมการวัดคุมทุกท่าน ที่ให้คำแนะนำอันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

และที่ลืมเสียมิได้คือ ขอกราบขอบพระคุณคุณพ่อ คุณแม่ อันเป็นที่รักยิ่ง ที่สนับสนุนและเป็นแรงบันดาลใจในการทำปริญญานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องจากปัจจุบัน การติดต่อสื่อสารไม่ว่าจะเป็นรูปแบบทางโทรทัศน์ โทรศัพท์ คอมพิวเตอร์ และ วิทยุ ได้มีบทบาทอย่างมากในสังคมปัจจุบัน มีกลุ่มผู้ใช้บริการการสื่อสารในรูปแบบต่างๆในการธรุกิจ การทหาร หรือข้อมูลส่วนตัว ผ่านตัวกลางการสื่อสารในรูปแบบต่างๆ ซึ่งความสำคัญของข่าวสารขึ้นอยู่กับผู้ส่งสารจะเป็นผู้ให้ความสำคัญของผู้ส่งสารนั้น เช่นข่าวทางการทหาร หรือด้านธุรกิจที่มีความสำคัญมาก ทำให้ผู้ส่งสารต้องหาวิธีในการป้องกันข้อมูลข่าวสารของตนเองให้รอดพ้นจากการจารกรรม วิธีการหนึ่งที่นำมาใช้ก็คือ การเข้ารหัสข้อมูล (Scramble) ก่อนทำการส่งผ่านในตัวกลางของการสื่อสาร และทำการถอดรหัส (Descramble) ข้อมูลที่ได้รับกลับคืนมาจากผู้ได้รับข่าวสารนั้น ๆ ซึ่งทำให้ข้อมูลข่าวสารรอดพ้น จากการถูกจารกรรมข้อมูล ถ้าผู้ทำการจารกรรมไม่สามารถทำการถอดรหัสข้อมูลเหล่านั้นได้ ทำให้สามารถรักษาข้อมูลที่ต้องการส่งไว้ได้ ในการวิจัยนี้ได้นำเสนอวิธีการเข้ารหัสโดยใช้หลักการของบล็อคโค้ดเชิงเส้นมาใช้ในการเข้ารหัสลับและการถอดรหัสโดยอาศัยซินโดรมของบล็อคโค้ดเชิงเส้นสำหรับการป้องกันข้อมูลข่าวสารให้ได้รับความปลอดภัย

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1 เพื่อเป็นการศึกษาหาแนวทางในการออกแบบโปรแกรมตัวเข้ารหัสลับ และ โปรแกรมตัวถอดรหัสด้วยภาษา VHDL นำมาสร้างเป็นแผงวงจร สำหรับทำงานตามโปรแกรมที่ทำการออกแบบ

2 เพื่อเป็นแนวทางสำหรับการพัฒนาในการออกแบบวิธีการสร้างการเข้ารหัสลับและวิธีการถอดรหัสลับ โดยเป็นแนวทางสำหรับผู้สนใจสร้างอุปกรณ์ทางด้านการป้องกันข้อมูลข่าวสารให้ปลอดภัย และเป็นการช่วยประหยัดเวลา ในการเริ่มศึกษาทางด้านนี้

3 สามารถนำแผงวงจรและโปรแกรมที่ออกแบบในการเข้ารหัสและถอดรหัส ไปใช้งานสำหรับป้องกันข้อมูลให้มีความปลอดภัย

1.3 ขอบเขตของปริญญานิพนธ์

ขอบเขตของการวิจัยจะเป็นการสร้างวงจรและการเขียนโปรแกรม VHDL ในการเข้ารหัสและถอดรหัสข้อมูลโดยอาศัยหลักการของบล็อคโค้ดเชิงเส้น และการหาซีนโดรม ซึ่งเป็นหลักวิธีในการนำมาใช้ในการออกแบบวิธีการเข้ารหัสลับและวิธีการถอดรหัสลับและนำวิธีการที่ได้ มาทำการออกแบบสร้างเป็นแผงวงจรเข้ารหัสลับและถอดรหัสลับ สำหรับนำไปใช้งาน

บทที่ 1 บทนำ กล่าวถึงความเป็นมาและความสำคัญของงานวิจัย วัตถุประสงค์ในการทำวิจัยและรายละเอียดเนื้อหาในบทต่างๆ

บทที่ 2 ระบบการเข้ารหัสลับและทฤษฎี ในรายละเอียดจะเป็นการกล่าวถึงพื้นฐานของการเข้ารหัสแบบต่างๆ และทฤษฎีของบล็อคโค้ดเชิงเส้น

บทที่ 3 หลักการของตัวเข้ารหัสลับและตัวถอดรหัสลับ ในรายละเอียดจะกล่าวถึง การนำวิธีการของบล็อคโค้ดเชิงเส้น มาใช้ในการออกแบบวิธีการเข้ารหัสลับและการวิธีการถอดรหัสลับและขั้นตอนต่างๆของการเข้ารหัส

บทที่ 4 การนำอุปกรณ์ FPGA มาใช้งาน ในรายละเอียดเป็นการกล่าวถึงการนำอุปกรณ์ FPGA มาใช้ในการสร้างวงจรเข้ารหัสลับและถอดรหัสลับ

บทที่ 5 แสดงผลการทดลอง โดยทำการเข้ารหัสและถอดรหัสสัญญาณเสียงพูด

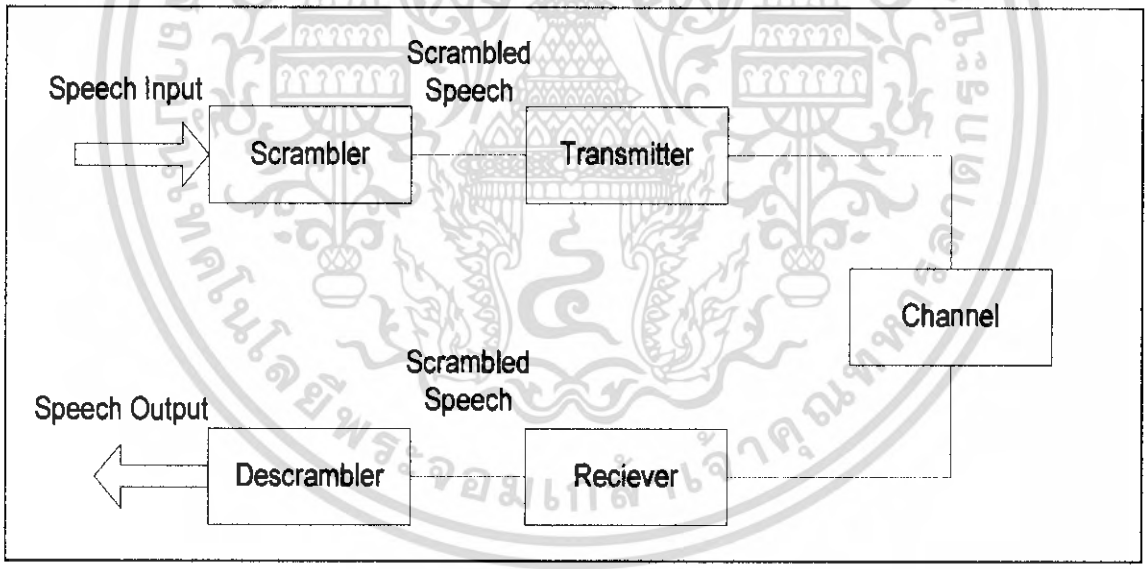
บทที่ 6 บทสรุปผลและแนวทางการพัฒนา กล่าวสรุปเนื้อหาของงานวิจัยที่ทำมา รวมถึงปัญหาที่ประสบในการทำวิจัย

บทที่ 2
หลักการของการเข้ารหัสและบล็อคโค้ดเชิงเส้น

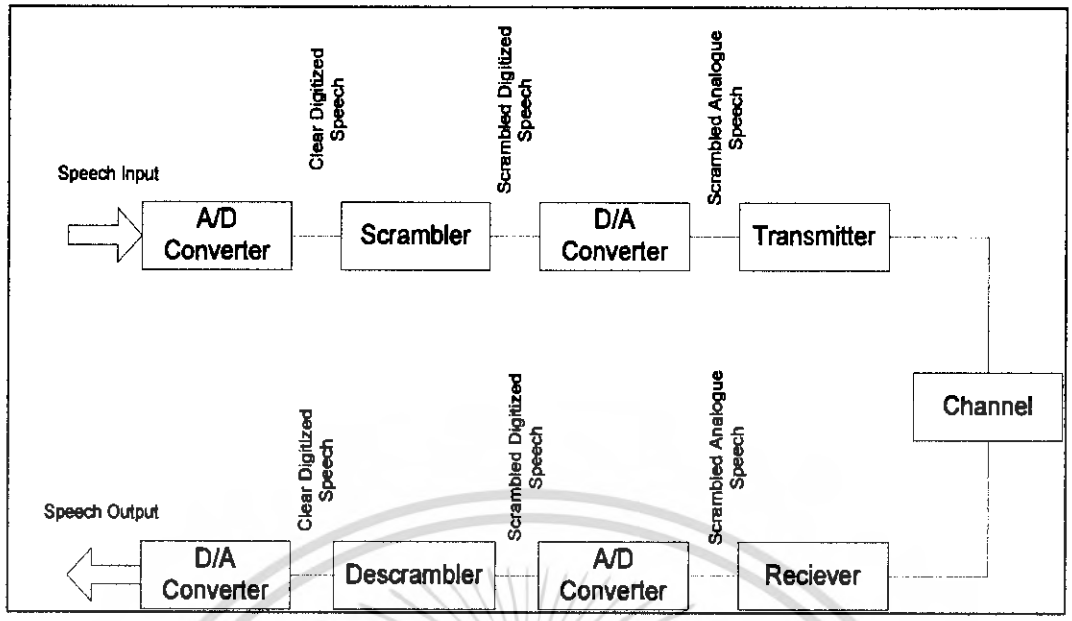
2.1 วิธีการสร้างรหัสลับ

โดยทั่วไปมักมีการกล่าวถึงการเข้ารหัสลับแบบพื้นฐาน 2 วิธี คือแบบ อะนาลอก และแบบดิจิตอล ส่วนใหญ่เครื่องเข้ารหัสสัญญาณที่มีความซับซ้อนมากๆ มักใช้การประมวลผลสัญญาณดิจิตอล กระบวนการนี้ทำงานโดยการแปลงสัญญาณ อะนาลอกไปเป็นสัญญาณดิจิตอลก่อนแล้วจึงทำการเข้ารหัส ส่วนในระบบที่เป็นแบบอะนาลอก โดยส่วนใหญ่จะมีความปลอดภัยค่อนข้างน้อย

ข้อแตกต่างที่ชัดเจนระหว่างการเข้ารหัสแบบอะนาลอกและแบบดิจิตอลคือ รูปแบบที่เกี่ยวกับตัวส่งผ่านซึ่งเป็นอุปกรณ์ที่ใช้ในการส่งสัญญาณที่เข้ารหัสแล้ว วัตถุประสงค์ของระบบที่มีการเข้ารหัสลับแบบอะนาลอก คือ ส่งผ่านข่าวสารที่เปลี่ยนแปลงอย่างต่อเนื่อง ในทางตรงข้ามการเข้ารหัสลับแบบดิจิตอลจะส่งผ่านด้วยสัญญาณที่สามารถนำไปใช้ในจำนวนที่จำกัด

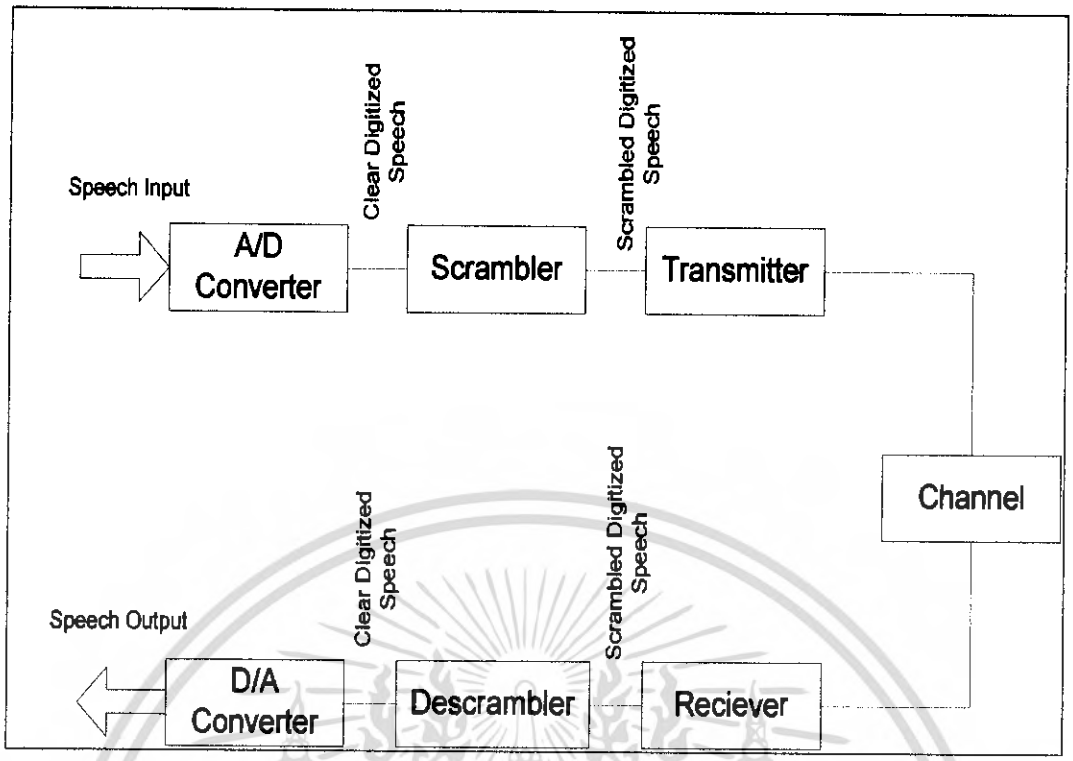


รูปที่ 2.1 แสดงการเข้ารหัสเสียงแบบอะนาลอกชนิดไม่มีการประมวลผลสัญญาณดิจิตอล



รูปที่ 2.2 แสดงการเข้ารหัสแบบอะนาลอกชนิดมีการประมวลผลสัญญาณแบบดิจิตอล

ในรูปที่ 2.1 และ 2.2 แสดงให้เห็นถึงการเข้ารหัสสัญญาณแบบอะนาลอก ความแตกต่างระหว่างรูปแบบการเข้ารหัสทั้ง 2 นี้คือ รูปแบบของสัญญาณที่ผ่านการเข้ารหัสแล้ว เช่นในรูปภาพที่ 1 ยังคงมีสัญญาณแบบอะนาลอก ทั้งกระบวนการเข้ารหัส และในรูปที่ 2.2 สัญญาณถูกเปลี่ยนไปอยู่ในรูปของสัญญาณดิจิตอลก่อนจะทำการเข้ารหัส แล้วถูกเปลี่ยนกลับให้ไปอยู่ในรูปของสัญญาณอะนาลอกก่อนทำการส่งผ่านและหลังทำการส่งผ่านจะถูกเปลี่ยนกลับให้เป็นสัญญาณดิจิตอลอีกครั้งหนึ่ง



รูปที่ 2.3 แสดงการเข้ารหัสเสียงแบบดิจิตอล

ในรูปที่ 2.3 แสดงให้เห็นถึงระบบดิจิตอลอีกประเภทหนึ่ง ความแตกต่างระหว่างระบบนี้และระบบในรูปที่ 2 คือ ไม่มีตัวแปลงสัญญาณจากดิจิตอลไปเป็นอะนาลอก ในทันทีก่อนเข้าสู่เครื่องส่งผ่าน และไม่มีตัวแปลงสัญญาณจากอะนาลอกไปเป็นสัญญาณดิจิตอลในทันทีหลังผ่านเครื่องรับ

2.2 การคำนวณทางคณิตศาสตร์ของรหัสแบบไบนารี

สำหรับการเข้ารหัสของวิทยานิพนธ์ฉบับนี้ จะใช้คณิตศาสตร์ที่มีความซับซ้อนน้อย ทั้งนี้เพื่อให้สามารถนำมาสร้างวงจรทางฮาร์ดแวร์และเขียนโปรแกรมภาษา VHDL ได้ง่าย คณิตศาสตร์ที่ว่านี้รู้จักกันในชื่อของ Galois Field เนื่องจากคณิตศาสตร์ที่ใช้มีลักษณะเป็นลอจิก คือมีสัญลักษณ์เป็น 0 กับ 1 บางครั้งคณิตศาสตร์ดังกล่าวนี้ถูกเรียกว่า Binary Field ซึ่งเขียนย่อเป็น GF(2) การทำงานของคณิตศาสตร์ดังกล่าวแสดงผลตามตารางที่ 1 โดยการบวกจะมีลักษณะเป็นการทำเอกซ์คลูซีฟ ออร์ (Exclusive OR) ของลอจิก ส่วนการคูณจะมีลักษณะเป็นการ AND ของลอจิก

ในการเข้ารหัสนี้ จะมีการนำเอาเวคเตอร์ของข้อมูลข่าวสาร (Message) คูณกับเมทริกซ์ที่ทำการออกแบบไว้ ผลลัพธ์ที่ได้จะเกิดจากการ AND และ EX-OR ของข้อมูลที่เป็นลอจิก จากเวคเตอร์ข้อมูลและเมทริกซ์ที่กำหนด

ตารางที่ 2.1 รูปแบบการคำนวณทางคณิตศาสตร์ของรหัส

0 + 0 = 0	0 x 0 = 0
0 + 1 = 1	0 x 1 = 0
1 + 0 = 1	1 x 0 = 0
1 + 1 = 1	1 x 1 = 1

2.3 การตรวจสอบพาริตี้ (Parity Check)

เพื่อให้เข้าใจถึงการสร้างรหัสเชิงเส้น ให้ดูตัวอย่างง่ายๆ จากการสร้างรหัสคำ (Code Word) เริ่มจากข้อมูลโดยปล่อยให้ผ่านไปในรหัสคำโดยตรง และจะมีบิทตามหลังแบบบิทเดียวซึ่งเป็นการคำนวณจากบิทข้อมูลทั้งหมด มีวิธีการกำหนดบิทสุดท้ายที่ตามหลังมา 2 วิธี คือ

1 กำหนดบิทสุดท้ายเพื่อให้ผลรวมแบบโมดูโล 2 ของบิททั้งหมดในรหัสคำ มีค่าเท่ากับ 1

2 กำหนดบิทสุดท้ายเพื่อให้ผลรวมแบบโมดูโล 2 ของบิททั้งหมดในรหัสคำ มีค่าเท่ากับ 0

ในกรณีแรก รหัสคำมีพาริตี้แบบคี่ (Odd Parity) กล่าวคือ จำนวนบิทที่เป็นหนึ่งในรหัสคำเป็นจำนวนคี่ ส่วนในกรณีที่ 2 จำนวนบิทที่เป็น 1 ในรหัสคำเป็นจำนวนคู่ (Even Parity) บิทที่เพิ่มขึ้นนอกเหนือจากบิทข้อมูลเรียกว่า พาริตี้เช็ค และอาจเรียกได้ว่าเป็นการตรวจสอบแบบพาริตี้คู่หรือพาริตี้คี่

รหัสแบบพาริตี้คู่หรือพาริตี้คี่จะแสดงในตารางที่ 2.2 และ 2.3 ตามลำดับ สำหรับกรณีที่มีบิทข้อมูล 3 ตำแหน่ง จะสังเกตุเห็นว่ารหัสของตารางที่ 2.2 ไม่มีแถวที่เป็น 0 ทั้งหมด ซึ่งต้องเป็นส่วนหนึ่งของรหัสที่เป็นเชิงเส้น

ตารางที่ 2.2 รหัสพาริตี้แบบคี่

ข้อมูลข่าวสาร	ข้อมูลเข้ารหัส
000	0001
001	0010
010	0100
011	0111
100	1000
101	1011
110	1101
111	1110

ดังนั้นการตรวจสอบพาริตี้แบบคี่ทำให้เกิดรหัสแบบไม่เป็นเชิงเส้น ในทางตรงข้ามรหัสที่ใช้พาริตี้ในตารางที่ 2.3 จะเป็นรหัสเชิงเส้น ระบบที่มีการผลิตการตรวจสอบพาริตี้แบบคู่โดยการเพิ่มบิทที่เป็นพาริตี้นั้น ได้จากการบวกแบบโมดูโล 2 ของบิทในรหัสของข้อมูลข่าวสาร ตัวอย่างเช่น รหัสข้อมูลข่าวสาร 101 เมื่อทำการบวกแบบโมดูโล 2 ของบิทที่มีค่าเป็น 1 , 0 , 1 จะได้ผลลัพธ์คือ 0 ซึ่งจะเป็นค่าของบิทที่จะนำมาทำการต่อข้างท้ายของรหัสข้อมูลข่าวสาร กลายเป็น 1010

ในการเพิ่มพาริตี้อีก 1 บิท ให้รหัสข้อมูลข่าวสารนั้น ปกติมักจะนำมาตรวจสอบได้ถ้าบิทผิดไปเพียงบิทเดียวเท่านั้นแต่ไม่สามารถนำมาทำการแก้ไขบิทที่ผิดได้ ในการแก้ไขบิทที่ผิดไปจะต้องใช้วิธีการเข้ารหัสที่เพิ่มพาริตี้บิทให้มากขึ้น อย่างเช่นการเข้ารหัสแบบบล็อคโค้ดเชิงเส้น ที่จะได้กล่าวถึงต่อไป

ตารางที่ 2.3 รหัสพาริตี้แบบคู่

ข้อมูลข่าวสาร	ข้อมูลเข้ารหัส
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

2.4 ความสามารถในการตรวจแก้บิทที่ผิดในรหัสเชิงเส้น

ในส่วนนี้จะได้กล่าวถึงคำศัพท์พื้นฐานที่ใช้ในการแก้บิทที่ผิดของรหัสเชิงเส้น

เวท (Weight) ของแฮมมิ่งสำหรับเวคเตอร์ v n-ทูเปิ้ลส์ คือ $\omega(v)$ ซึ่งหมายถึงผลรวมของจำนวนบิท ของรหัสของ v ที่ไม่เป็นศูนย์ ตัวอย่างเช่น ถ้า v = [1 0 1 0 1 1 0 0 0 1]จะได้ $\omega(v) = 5$

ให้ u และ v เป็นเวคเตอร์ n-ทูเปิ้ลส์ ค่าระยะห่างระหว่าง u และ v เขียนได้เป็น d(u,v) ระยะห่างของสองเวคเตอร์ใดๆ คือจำนวนบิทรหัส "1" ที่แตกต่างกันของเวคเตอร์ทั้ง 2 เช่น

u = [1 0 0 1 0 1 1 0 0 0 1]

v = [1 1 0 0 1 0 1 0 1 0 1]

จะได้ d(u,v) = 5

ถ้านำเวคเตอร์ u และเวคเตอร์ v มาบวกกัน โดยการบวกไบนารีเป็นการทำ Ex-OR ดังนั้น

$$u \oplus v = [0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0]$$

เวคเตอร์รวมที่ได้ถ้านำมาหาเวทของแฮมมิ่งจะได้ว่า $\omega(u \oplus v) = 5$ ดังนั้น พอสรุปได้ว่าระยะห่างแบบแฮมมิ่งระหว่างเวคเตอร์ u และ v จะเท่ากับเวทของแฮมมิ่งจากเวคเตอร์รวม กล่าวคือ

$$d(u,v) = \omega(u \oplus v) \tag{2.1}$$

สำหรับรหัสเชิงเส้นในการหาระยะห่างของแต่ละคู่ของรหัสคำ ระยะห่างที่น้อยที่สุดเขียนย่อเป็น $d_{\min}$ ถ้า u และ v เป็นโค้ดเวคเตอร์ 2 ชุดของรหัสเชิงเส้น โดย u + v ก็ยังเป็นโค้ดเวคเตอร์ เพราะเซทของทุกโค้ดเวคเตอร์ เป็นซับสเปซของทุก n-ทูเปิ้ลส์ ดังนั้นจากคำนิยามที่ว่า ระยะห่างระหว่างโค้ดเวคเตอร์ทั้ง 2 คือ เวทของโค้ดเวคเตอร์ที่ 3 ก็จะได้ระยะห่างน้อยสุดของรหัสเชิงเส้นเท่ากับเวทต่ำสุดของโค้ดเวคเตอร์ที่ไม่เป็นศูนย์ ค่าระยะห่างน้อยสุด และเวทต่ำสุดจะเป็นตัวกำหนดความสามารถในการแก้รหัวบิทที่ผิดของรหัสเชิงเส้น

พิจารณาจากรหัสที่ใช้ส่งโดยให้ $v = (v_1, v_2, ...v_n)$ เป็นโค้ดเวคเตอร์ที่ใช้ส่งและให้ $r = (r_1, r_2, ...r_n)$ เป็นเวคเดอร์ที่ได้รับจากการส่ง แต่เนื่องจากเวคเตอร์ที่รับได้จะเป็นอะไรก็ได้ใน 2^n เวคเตอร์ของ n-ทูเปิ้ลส์ ความแตกต่างระหว่าง r และ v คือ e

$$\begin{aligned} e &= (e_1, e_2, ...e_n) \\ &= r \oplus v \end{aligned}$$

$$\begin{aligned} e &= (r_1, r_2, ...r_n) + (v_1, v_2, ...v_n) \\ &= (r_1 + v_1, r_2 + v_2, ...r_n + v_n) \end{aligned}$$

ซึ่ง e เป็นรูปแบบของรหัสที่ผิด (Error Pattern , Error Vector) เมื่อ $e_i = r_i + v_i = 1$ นั่นก็หมายความว่า โค้ดเวคเตอร์เกิดความผิดพลาดคำแหน่งบิทที่ i^{th} แต่เนื่องจากในหนึ่งโค้ดเวคเตอร์ มีรหัสอยู่ n บิท จึงทำให้เกิดความผิดพลาด $2^n - 1$ รูปแบบที่แตกต่างกัน ไม่รับรูปแบบที่มีทุกบิทเป็นศูนย์

ทางด้านรับตัวถอดรหัสมีหน้าที่ในการตรวจหาโค้ดเวคเตอร์ที่ส่งมาจากโค้ดเวคเตอร์ r ที่รับได้ สำหรับการถอดรหัสดดยใช้วิธีแม็กซิมั่มไลค์รีฮู้ดนั้น ตัวถอดรหัสจะตรวจสอบว่า v เป็นเวคเตอร์ที่ใช้ในการส่ง ซึ่งจะมีค่าเข้าใกล้เวคเตอร์ r โดยอาศัยการดูจากระยะห่างของแฮมมิ่ง ตัวถอดรหัสสามารถทำการแก้ไขรหัสที่ผิดจำนวน t บิท ในโค้ดเวคเตอร์ที่รับเข้ามา โดยที่ $2t + 2 \geq d_{\min} \geq 2t + 1$ ตัวถอดรหัสสามารถทำการแก้ทุกรูปแบบที่ผิดไป t บิท จากเวคเตอร์ r ที่รับได้ ซึ่งแสดงให้เห็นได้ดังนี้ ให้ v เป็นโค้ดเวคเตอร์ที่ต้องการส่งและ u เป็นโค้ดเวคเตอร์ใดๆ ระยะห่างของแฮมมิ่งระหว่าง u , v , r จะต้องเป็นไปตามสมการนี้

$$d(v,r)+d(u,r)\geq d(u,v) \tag{2.2}$$

ถ้าสมมติว่าเกิดรหัสผิดไป t' บิท $(t'\leq t)$ ดังนั้นระยะห่างของแฮมมิ่งระหว่างโค้ดเวคเตอร์ที่ส่ง v กับโค้ดเวคเตอร์ที่รับ r คือ $d(v,r)=t'$ แต่ $d(u,v)\geq d_{\min}\geq 2t+1$ สมการที่ 2.16 จะให้

$$\begin{aligned} d(u,r) &\geq 2t+1-t' \\ d(u,r) &\geq t+ \\ d(u,r) &\geq t' \end{aligned} \tag{2.3}$$

จากสมการที่ 2.3 แสดงหเห็นว่ารูปแบบของรหัสที่ผิดไป t บิท หรือน้อยกว่า เวคเตอร์ r ที่รับได้จะเข้าใกล้โค้ดเวคเตอร์ v กว่าโค้ดเวคเดอร์ u ดังนั้นตัวถอดรหัสจะสามารถแก้ไขรหัสผิดได้ถูกต้องตามความผิดพลาดของบิทที่ผิดไปตัวถอดรหัสไม่สามารถจะแก้ทุกรูปแบบที่ผิดไป L บิท เมื่อ $L\geq t+1$ โดยปกติแล้วการแก้รหัสผิดของโค้ดเชิงเส้นจะทำได้เมื่อ

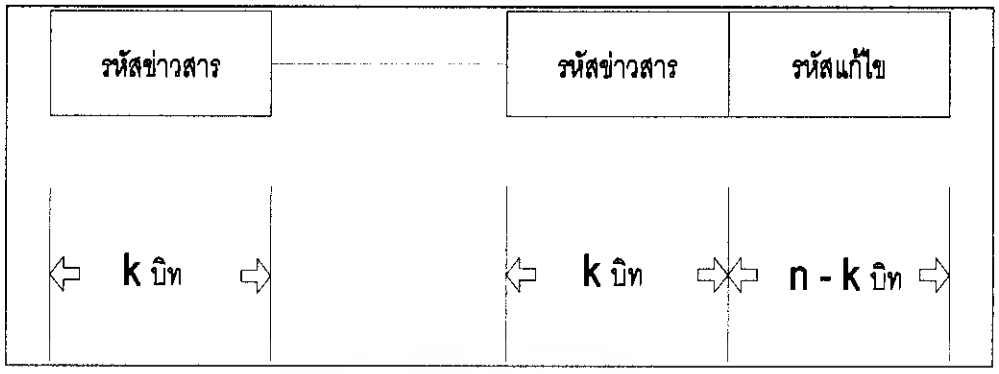
$$t=(d_{\min}-1)/2 \tag{2.4}$$

โดย $t=(d_{\min}-1)/2$ เป็นค่าจำนวนเต็มไม่คิดทศนิยม และสามารถตรวจสอบรหัสผิดที่เกิดขึ้นถึง $(d_{\min}-1)$ บิทในแต่ละรหัสคำ

2.5 บล็อคโค้ดเชิงเส้น

จากรหัสข่าวสาร ที่แต่ละบล็อคมีขนาด k บิท ซึ่งพบว่าจะให้ข่าวสารที่แตกต่างกันได้ถึง 2^k-1 ข่าวสาร (ยกเว้นบล็อคที่มีรหัสข่าวสารเป็นศูนย์หมดจะไม่มีการนำมาใช้) แต่ละบล็อคข่าวสารจะถูกรำมาเข้ารหัสเป็นบล็อคขนาด n บิทโดยจะมี n – k บิท ที่เพิ่มเข้าไปให้รหัสข่าวสาร บิทเหล่านี้ที่เพิ่มเข้าไปในต่ละบล็อคจะเป็นพาริตี้หรือบางทีเรียกว่ารหัสแก้ไข ที่จะถูกนำมาใช้ในการตรวจสอบบิทที่ผิดไป และค่าของ n – k บิท จะขึ้นกับรหัสข่าวสารโดยตรง

บล็อคข่าวสารขนาด n บิท ที่ได้จากการเข้ารหัสคำนี้จะเรียกว่า รหัสคำ ถ้าหากว่ารหัสคำมีบิทของข่าวสารเดิมปรากฏอยู่ใน k บิทเริ่มต้น รหัสนั้นจะเรียกว่าซิสเต็มเมติกซ์ ยิ่งไปกว่านั้นถ้าหากแต่ละรหัสคำจากจำนวน 2^k รหัสคำที่เข้ารหัสไว้ เกิดจากการรวมกันของ k เวคเตอร์ของรหัสแบบอิสระเชิงเส้น รหัสดังกล่าวจะเรียกว่ารหัสบล็อคโค้ดเชิงเส้น
รูปแบบของบล็อคโค้ดเชิงเส้นแสดงได้ดังรูปที่ 2.4



รูปที่ 2.4 แสดงรูปรหัสคำของบล็อคโค้ดเชิงเส้น

2.6 เมทริกซ์ตัวกำเนิด (Generator Matrix)

สำหรับซับสเปซ S ของ V_n และแต่ละ n-ทูเปิ้ลส์ ของ S เป็นการรวมแบบเชิงเส้นของ $V_1, V_2, \ldots V_k$ กล่าวคือ

$$u = m_1 v_1 + m_2 v_2 + \ldots + m_k v_k \tag{2.5}$$

เมื่อ $m_i = 0$ สำหรับ i = 1,2,...,k ซับสเปซนี้มีขนาด k มิติของ V_n ซึ่งประกอบด้วย 2^k ของ n-ทูเปิ้ลส์จากข้อกำหนดที่กล่าวมา ซึ่งสามารถอธิบายได้ถึงรหัสเชิงเส้นของ 2^k รหัสคำโดยเซ็ทของ k โค้ดเวคเตอร์ที่เป็นข้อกำหนดอิสระเชิงเส้น ถ้าจัด k รหัสคำเป็นอิสระต่อกันได้เมทริกซ์ k x n

$$G = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & & & \\ v_{k1} & v_{k2} & \cdots & v_{kn} \end{bmatrix} \tag{2.6}$$

เมื่อ $v_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ สำหรับ i = 1,2,3.....,k ให้ m = $(m_1, m_2, \ldots, m_k)$ เป็นบล็อคของข่าวสาร รหัสคำจะได้จาก

$$
\begin{aligned}
u &= mG \\
&= (m_1, m_2, \ldots, m_k)\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix} \\
&= m_1 v_1 + m_2 v_2 + \ldots + m_k v_k
\end{aligned} \tag{2.7}
$$

ดังนั้นรหัสที่สอดคล้องกับชุดข่าวสาร $(m_1, m_2, \ldots, m_k)$ เกิดจากการรวมแบบเชิงเส้น ของแถวใน G กลุ่มแถวต่างๆของเมทริกซ์ G จะเป็นตัวผลิตรหัสเชิงเส้น และเราเรียกเมทริกซ์ G ว่า เมทริกซ์ตัวกำเนิดของรหัส รหัสเชิงเส้นที่กล่าวนี้ เรียกว่า รหัส (n , k) โดยในแต่ละบล็อคจะมีข่าวสารอยู่ k บิท ที่ถูกเข้ารหัสเป็นรหัสคำที่มีความยาวขนาด k บิท

ลักษณะของเมทริกซ์ตัวกำเนิดขนาด k x n ที่ใช้สร้างรหัสเชิงเส้น (n , k) แสดงได้ดังสมการที่ (2.8)

โดย $P_{ij} = 1$ หรือ 0 ให้ I_k เป็นเมทริกซ์เอกลักษณ์ ขนาด k x k และให้ P เป็นเมทริกซ์ขนาด k x (n - k) ที่มีอิลิเมนท์เป็น P_{ij} ดังนั้นเมทริกซ์ตัวกำเนิดของรหัสระบบเขียนใหม่ได้เป็น

$$G = [I_k : P]$$

พิจารณาถึงบล็อคของข่าวสาร m = $(m_1, m_2, \ldots, m_k)$ เมื่อได้เมทริกซ์ตัวกำเนิดของสมการ (2.8) จะได้โค้ดเวคเตอร์เป็น

$$
\begin{aligned}
u &= (u_1, u_2, u_3, \ldots, u_n \\
&= (m_1, m_2, m_3, \ldots, m_k)G
\end{aligned}
$$

$$
= (m_1, m_2, \ldots, m_k)\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{11} & p_{12} & \cdots & p_{1k} \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & p_{21} & p_{22} & \cdots & p_{2k} \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \vdots & & & \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \vdots & & & \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & p_{k1} & p_{k2} & \cdots & p_{k,n-k}
\end{bmatrix} \tag{2.9}
$$

จากการคูณเมทริกซ์จะได้

$$u_i = m_i \quad \text{สำหรับ i = 1,2,3,...,k} \tag{2.10}$$

และ

$$u_{k+j} = p_{1j}m_1 + p_{2j}m_2 + ... + p_{kj}m_k \tag{2.11}$$

สำหรับ j = 1,2,3,....., n – k จากสมการที่ 2.10 และ 2.11 จะพบว่ารหัส k บิทแรกของรหัสคำ คือ รหัสของข่าวสาร ส่วน (n - k) บิทของ u หรือ รหัสตรวจสอบพาริตี้ของรหัสคำ สมการที่ 2.11 จะเรียกว่าสมการพาริตี้ของรหัส

2.7 เมทริกซ์ในการตรวจสอบพาริตี้

จากที่กล่าวว่าเมทริกซ์ G ขนาด k x n จะมีเมทริกซ์ H ขนาด (n – k) x n ซึ่งโรว์สเปซของ G จะตั้งฉากอยู่กับ H อินเนอร์โปรดักส์ของเวคเตอร์ในโรว์สเปซของ G กับแถวของ H จะเป็น 0

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ \vdots & & & \\ h_{k1} & h_{k2} & \cdots & h_{n-k,n} \end{bmatrix}$$

และให้ $u = u_1, u_2, ..., u_n$ เป็นเวคเตอร์ในโรว์สเปซของ G จะได้

$$uH^T = (0\,0\,0\,\ldots\,0) \tag{2.13}$$

หรือ

$$uh_i = u_i h_{i1} + u_2 h_{i2} + ... + u_n h_{im} = 0 \tag{2.14}$$

สำหรับ $u = u_1, u_2, ..., u_n$ จึงสรุปได้ว่า u จะเป็นรหัสคำที่ได้จาก G ถ้าเพียงแต่ $uH^T = 0$ เมทริกซ์ H นี้เรียกว่า เมทริกซ์ในการตรวจสอบพาริตี้ หรือเรียกย่อๆว่า พาริตี้เมทริกซ์ ถ้าเมทริกซ์ตัวกำเนิดของรหัส ได้มาจากสมการที่ 2.8 พาริตี้เมทริกซ์ของรหัสคือ

$$H = \begin{bmatrix} p_{11} & p_{21} & \cdots & p_{k1} & 1 & 0 & 0 & 0 \\ p_{12} & p_{22} & \cdots & p_{k2} & 0 & 1 & 0 & 0 \\ p_{13} & & \cdots & p_{k3} & 0 & 0 & 1 & 0 \\ p_{14} & & \cdots & p_{k4} & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

$$= \left[P^T : I_{n-k} \right]$$

P^T เป็นทรานสโพสของเมทริกซ์ P สมการพาริตี้ 2.11 ได้จากเมทริกซ์ H นั่นคือ $u = u_1, u_2, ..., u_n$ เป็นรหัสคำของรหัสข้อมูล $m = m_1, m_2, ..., m_k$ เมื่อ $u_i = m_i$ สำหรับ I = 1,2,,,…,k แต่

$$uH^T = 0$$

จะได้ว่า

$$u_{k+j} = p_{1j}u_1 + p_{2j}u_2 + ... + p_{kj}u_k$$

$$= p_{1j}m_1 + p_{2j}m_2 + ... + p_{kj}m_k \tag{2.16}$$

เมื่อ j = 1,2,3,…., n – k ซึ่งสมการข้างบนเป็นสมการเดียวกับสมการที่ 2.11 ในการออกแบบรหัสเชิงเส้นนั้น เมทริกซ์ P จะถูกเลือกเพื่อให้มีคุณสมบัติในการแก้บิทที่ผิด

2.8 ซีนโดรม

ซีนโดรมเป็นบล็อคของรหัสคำขนาด n บิท เมื่อทำการส่งออกไปในตัวกลางจะเกิดสัญญาณรบกวน ทำให้บางบิทของข้อมูลผิดไป ซึ่งรูปแบบของบิทที่ผิดไปบางครั้งเรียกว่าโคเซ็ท (Coset) มีหลายรูปแบบ ถ้าหาก u เป็นรหัสคำที่ต้องการส่ง โดย u มีระยะห่างต่ำสุด ตามเงื่อนไขสมการ 2.4 และ e_i เป็นรูปแบบของบิทที่ผิดไปในระหว่างการติดต่อสื่อสาร ทางด้านรับจะได้รหัสคำเป็น r กล่าวคือ

$$r = e_i + u \tag{2.17}$$

การคำนวณหาซีนโดรมของรหัสคำที่รับได้ทำได้โดย

$$S = uH^T$$

$$= (e_i + u)H^T = e_i H^T + uH^T \qquad (2.18)$$

ถ้าหาก u เป็นรหัสคำที่ถูกต้อง จะพบว่า

$$uH^T = 0 \qquad (2.19)$$

ดังนั้น ซินโดรมคือ $S = e_i H^T$

โดยปกติแล้วแต่ละรูปแบบของบิทที่ผิดไปถ้าไม่ซ้ำกันจะให้ค่าซินโดรมที่ไม่เท่ากัน ในการพิสูจน์ทำได้โดยถ้าสมมติว่ารูปแบบของบิทที่ผิดไปคนละรูปแบบแต่ให้ซินโดรมเท่ากัน อย่างเช่น รูปแบบของบิทที่ผิดไป e_i กับ e_t เมื่อซินโดรมคือ

$$S_1 = e_i H^T$$

$$S_2 = e_t H^T$$

แต่ $S_1 = S_2$ จะให้

$$e_i H^T = e_t H^T$$

หรือ

$$(e_i - e_t)H^T = 0$$

เนื่องจาก H^T ไม่เป็น 0 ดังนั้น $(e_i - e_t) = 0$

ผลต่างของ e_i กับ e_t จะเป็น 0 ก็ต่อเมื่อทุกบิทใน e_i กับ e_t จะเหมือนกันแบบบิทต่อบิท จึงสามารถสรุปได้ว่า

$$e_i = e_t$$

ซึ่งขัดกับสมมติฐานที่ว่า e_i กับ e_t เป็นคนละรูปแบบ

ดังนั้นพอสรุปได้ว่า ถ้าหากรูปแบบบิทที่ผิดไปคนละรูปแบบ จะให้ค่าซินโดรมที่แตกต่างกันออกไป กล่าวคือ

$$e_i H^T \neq e_t H^T$$

ถ้าหากรหัสคำ (n , k) ถูกนำมาคำนวณหาซินโดรม จะได้ซินโดรมขนาด n – k บิท ดังนั้น ซินโดรมที่แตกต่างกันจะมีจำนวน 2^{n-k} ค่า จะพบว่ารูปแบบของบิทที่ผิดไปหนึ่งรูปแบบกับซินโด

รมที่สอดคล้องจะเป็นแบบหนึ่งต่อหนึ่ง ปกติแล้วทางด้านรับจะมีตารางของซีนโดรมและรูปแบบของบิทที่ผิดที่สอดคล้องกันกับซีนโดรมเก็บเอาไว้ ดังนั้น ทางด้านรับจะมีขั้นตอนการทำงาน 4 ขั้นตอน กล่าวคือ

1 คำนวณซีนโดรมของรหัสคำ r ที่ได้รับจาก $S = rH^T$

2 เปิดตารางของซีนโดรม เพื่อดึงเอารูปแบบที่ผิดให้ซีนโดรมเหมือนกับที่คำนวณได้ ถ้าหากเป็นรูปแบบของ e_l

3 รหัสที่ถูกต้องคำนวณได้จาก $u = r \oplus e$

4 ดึง k บิทแรกจากรหัสคำของ u ซึ่งจะเป็นรหัสข่าวสาร m ที่ส่งมา

ตัวอย่างเช่น ถ้าหากมีเมทริกซ์ตัวกำเนิด G เป็น

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

สามารถแปลงเป็นเมทริกซ์ตรวจสอบพาริตี้ H ได้เป็น

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

ถ้าสมมติว่ารูปแบบรหัสที่ผิดไปเพียงหนึ่งบิทเป็น $e_l = [0,0,0,0,0,1]$ จะให้ซีนโดรมเป็น

$$S = e_l H^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

สำหรับรูปแบบต่างๆของรหัสที่ผิดไปเพียงบิทเดียวกับซีนโดรมที่สอดคล้องพอสรุปได้ดังตารางที่จะแสดงต่อไปนี้

ตารางที่ 2.4 แสดงซีนโดรมที่ได้จากรหัสที่ผิดไป 1 บิท

ซีนโดรม	รูปแบบรหัสที่ผิดไปเพียงบิทเดียว
001	000001
010	000010
100	000100
110	001000
101	010000
011	100000

ถ้าสมมติว่าทางด้านส่งต้องการส่งรหัสข่าวสาร m = [1 0 1] จะได้รหัสคำ u

$$u = mG = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

เมื่อทำการส่งรหัส u ผ่านช่องส่งสัญญาณที่มีการรบกวน จะพบว่าทางด้านรับจะได้รับรหัสคำเป็น [1 0 0 1 0 1] ทางด้านรับจะทำการคำนวณหาซีนโดรมจากรหัสคำ r ถ้าซีนโดรมเป็น 0 หมดทุกบิท แสดงว่ารหัสคำ r ที่รับกับรหัสคำ u ที่ส่ง เป็นรหัสเดียวกัน แต่ถ้าหากค่าซีนโดรมไม่เป็น 0 ก็ต้องทำการเปิดตารางที่ 4 เพื่อหารูปแบบรหัสที่ผิดไปที่สอดคล้อง

การคำนวณหาซีนโดรมทำได้โดย

$$S = rH^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

ซีนโดรมที่ได้จะนำไปตรวจสอบกับตาราง พบว่ารูปแบบของรหัสที่ผิดไปคือ [0 , 0 , 1 , 0 , 0 , 0] ดังนั้น การแก้ไขรหัส r เพื่อให้ได้รหัสคำ u ที่ส่งมาทำได้โดย

$$\begin{aligned} u &= r \oplus e \\ &= [\,1\,0\,0\,1\,0\,1\,] + [\,0\,0\,1\,0\,0\,0\,] \\ &= [\,1\,0\,1\,1\,0\,1\,] \end{aligned}$$

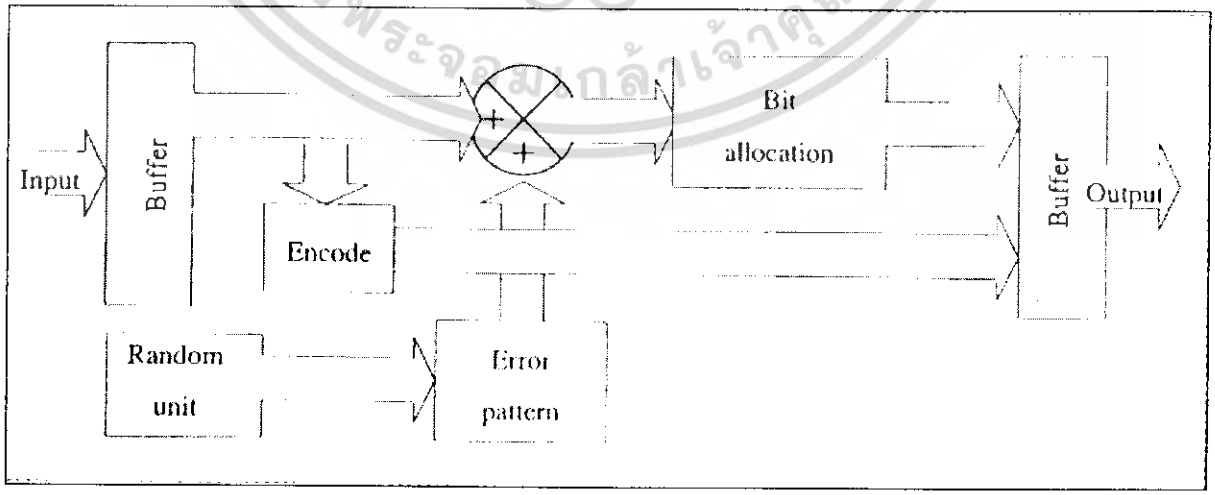
จากวิธีการดึงรหัสคำที่ถูกต้องกลับคืนมาจากรหัสคำที่ผิด จึงได้ถูกนำมาใช้เป็นกลักการเข้ารหัสลับให้กับข้อมูลในวิทยานิพนธ์ฉบับนี้ หลักการทำงานของการแก้รหัสลับคือ จากรหัสคำที่มีอยู่จะถูกนำมาบวกกับรูปแบบรหัสที่ผิดต่างๆ ที่กำหนดไว้ โดยแต่ละรูปแบบของรหัสที่ผิดจะมีซีนโดรมที่สอดคล้องเก็บเป็นตารางข้อมูลเอาไว้ ซึ่งรหัสคำที่ผิดมีหลายรูปแบบ ดังนั้นการบวกรหัสคำกับรูปแบบรหัสที่ผิด จะทำการเลือกรูปแบบของรหัสที่ผิดแบบสุ่มเทียม (Pseudo Random) วิธีการนี้ ทางด้านรับจะมีรหัสคำที่ผิดอยู่ตลอดเวลา แต่จะสามารถนำเอารหัสที่ถูกต้องกลับคืนมาได้ ถ้าหากรู้ว่าเมทริกซ์ G คืออะไร ในการเพิ่มความซับซ้อนของการเข้ารหัสลับ จะมีการเรียงลำดับตำแหน่งบิท ในแต่ละรหัสคำที่ผ่านการบวกรูปแบบที่ผิดไปเรียบร้อยแล้ว รูปแบบของการเรียงสลับบิทจะใช้ค่าของซีนโดรมเป็นตัวเลือก สำหรับรายละเอียดในการเข้ารหัสลับ จะได้กล่าวในบทถัดไป

บทที่ 3
การออกแบบตัวเข้ารหัสและตัวถอดรหัส

ในการออกแบบตัวเข้ารหัสและถอดรหัสลับแบบดิจิตอลนี้ สัญญาณที่จะนำมาป้อนเข้าทางอินพุตสำหรับทำการเข้ารหัสจะต้องเป็นสัญญาณที่อยู่ในรูปของสัญญาณดิจิตอล แต่ถ้าอินพุตที่ถูกป้อนเข้ามาอยู่ในรูปของสัญญาณอะนาลอก จะต้องทำการแปลงสัญญาณเหล่านั้น ให้อยู่ในรูปของสัญญาณดิจิคอล ก่อนที่จะทำการส่งเข้าทางด้านอินพุตของตัวเข้ารหัส ส่วนทางด้านตัวถอดรหัสลับก็จะได้รับข้อมูลที่ถูกส่งผ่านเข้ามาทางอินพุตที่อยู่ในรูปสัญญาณดิจิตอลเช่นกัน และทำการถอดรหัสกลับ แต่ถ้ารูปสัญญาณเดิมเป็นสัญญาณอะนาลอก ก็จะต้องใช้ตัวแปลงสัญญาณดิจิตอลกลับเป็นสัญญาณอะนาลอกกลับคืนมาเช่นเดิม ในส่วนนี้จะกล่าวถึงวิธีการเฉพาะส่วนของตัวเข้ารหัสลับและตัวถอดรหัสลับเท่านั้น

3.1 การออกแบบตัวเข้ารหัสลับ

ในการออกแบบตัวสร้างรหัสลับจะเป็นการนำข่าวสารข้อมูลที่เข้ามาทางด้านอินพุตมาทำการแปลงให้มีรูปแบบของข้อมูลแตกต่างไปจากข้อมูลเดิม โดยทำการแปลงข้อมูลข่าวสารที่เข้ามาทางด้านอินพุตที่มีขนาด k บิท (ในที่นี้ใช้ 8 บิท) ให้เป็นรหัสคำที่มีขนาด n บิท (ในที่นี้ใช้ 12 บิท) แล้วทำการบวกรหัสคำที่ได้กับรูปแบบผิดพลาดที่ทำการสร้างขึ้นจำนวน 15 รูปแบบ หลังจากนั้นจะทำการสลับตำแหน่งของรหัสคำซึ่งมีรูปแบบการสลับตำแหน่ง 16 รูปแบบ ที่ทำการบวกรูปแบบผิดพลาดแล้ว ก็จะทำให้ได้รหัสข้อมูลลับที่ทำการเข้ารหัสแล้ว ซึ่งสามารถแบ่งเป็นส่วนๆของการสร้างรหัสลับ ดังแสดงในรูปที่ 3.1 ได้ดังนี้



รูปที่ 3.1 แสดงแผนภาพการเข้ารหัสลับ

3.2 ส่วนของการเข้ารหัส

เป็นส่วนที่ทำหน้าที่ในการแปลงรหัสข้อมูลข่าวสารที่เข้ามาทางอินพุตที่มีขนาดของข้อมูล k บิท มาทำการแปลงให้เป็นรหัสคำ ในการแปลงนี้จะใช้วิธีของบล็อคโค้ดเชิงเส้น โดยจะเป็นการนำรหัสข่าวสาร m ที่มีขนาด k บิท ไปทำการคูณกับเมทริกซ์ตัวกำเนิด G จะได้ผลลัพธ์เป็นรหัสคำที่มีขนาด n บิท จะมีจำนวนของบิทที่เพิ่มขึ้นมาจากรหัสของข้อมูลข่าวสาร (n – k) บิท (4 บิท) โดยรหัสที่เพิ่มขึ้นมานี้จะถูกนำมาใช้ในการแก้รหัสที่ผิด ซึ่งจะกล่าวถึงในหัวข้อถัดไป

ภายในเมทริกซ์ตัวกำเนิด G จะประกอบไปด้วยเมทริกซ์เอกลักษณ์ I_k ที่มีขนาด k x k และเมทริกซ์พาริตี้ P ที่มีขนาด k x (n – k) จะแสดงได้ในสมการที่ 3.1

$$G = [I_k : P] \tag{3.1}$$

เททริกซ์ตัวกำเนิด G ที่ใช้แสดงในรูปที่ 3.2

$$G = \left[\begin{array}{cccccccc:cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right]$$

รูปที่ 3.2 แสดงเมทริกซ์ตัวกำเนิด

จากรหัสข้อมูลข่าวสาร m ที่มีขนาด 8 บิท นำมาทำการคูณกับเมทริกซ์ตัวกำเนิด G จะได้รหัสคำที่มีขนาด 12 บิท ดังแสดงในสมการที่ 3.2

$$u = mG \tag{3.2}$$

ทำให้สามารถแสดงวิธีการสร้างรหัสคำ u ได้ดังรูปที่ 3.3

$$u = [m_1, m_2, m_3, m_4, m_5, m_6, m_7] \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & : & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & : & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & : & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & : & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & : & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & : & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & : & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & : & 1 & 1 & 0 & 1 \end{bmatrix}$$

รูปที่ 3.3 แสดงการสร้างรหัสคำ

3.3 ส่วนของรูปแบบผิดพลาด

สำหรับเมทริกซ์ตัวกำเนิดในรูปที่ 3.2 นั้น จะแก้รหัสผิดได้เพียงบิทเดียว แต่ในการเข้ารหัสลับนี้จะใช้รูปแบบของบิทที่ผิดไป 2 บิท ใน 8 บิทแรกของรหัสคำ ซึ่งเป็นบิทของข่าวสาร ดังนั้น จึงไม่เป็นไปตามกฎของซีนโดรม กล่าวคืออาจมีรูปแบบของบิทที่ผิดไปมากกว่าหนึ่งรูปแบบที่ให้ซีนโดรมเหมือนกัน ดังแสดงในตารางที่ 3.1 เนื่องจากการเข้ารหัสบล็อคโค้ดเชิงเส้นแบบ เข้า 8 บิท ออก 12 บิท จะมีซีนโดรมได้ 4 บิท หรือมีซีนโดรมได้ 15 รูปแบบที่แตกต่างกัน (ไม่นับซีนโดรมที่ทุกบิทเป็น 0 หมด) เนื่องจากรหัสที่ผิดไป 2 บิทใน 8 บิทแรก จะมีรูปแบบที่ไม่เหมือนกันได้เป็น 28 รูปแบบดังแสดงในคอลัมน์ที่ 2 ของตารางที่ 3.1 ในกรณีที่รูปแบบบิทที่ผิดมีหลายรูปแบบ จะเลือกเอารูปแบบบิทที่ผิดที่ให้ค่าสูงที่สุด อย่างเช่นรูปแบบ (3 , 5)ซึ่งมีบิทที่ผิด คือบิทที่ 3 กับ 5 จาก 8 บิทแรกของรหัสคำ กับรูปแบบ (2 , 7) ซึ่งมีบิทที่ผิดคือบิทที่ 2 กับ 7 ใน 8 บิทแรกของรหัสคำ จะพบว่ารูปแบบ (2 , 7) มีค่าสูงกว่า (3 , 5) จึงเลือกรูปแบบ (2 , 7) เนื่องจากการเข้ารหัสลับสัญญาณเสียง ถ้าเลือกบิทที่ผิดมีค่าสูง จะทำให้แอมปลิจูดของสัญญาณเสียงที่รวมกับรหัสที่ผิด แล้วเกิดการเปลี่ยนแปลงสูง ดังนั้น รูปแบบบิทที่ผิดที่ได้เลือกไว้ และซีนโดรมที่สอดคล้องพอสรุปได้ดังตารางที่ 3.1

ตารางที่ 3.1 รูปแบบผิดพลาด

ค่า Syndrome	ตำแหน่งบิทที่ผิด	เลือกรูปแบบ	รูปแบบบิทที่ผิด
0001	(3,5)(2,7)	(2,7)	001000010000
0010	(1,6)(2,4)	(1,6)	010000100000
0011	(4,7)	(4,7)	000010010000
0100	(0,5)(1,7)	(0,5)	100001000000
0101	(0,3)(1,2)(4,6)	(4,6)	000010100000
0110	(6,7)	(6,7)	000000110000
0111	(1,4)(2,6)	(1,4)	010010000000
1000	(0,6)(3,4)	(3,4)	000110000000
1001	(4,5)	(4,5)	000011000000
1010	(0,1)(2,3)(5,7)	(5,7)	000001010000
1011	(2,5)(3,7)	(2,5)	001001000000
1100	(5,6)	(5,6)	000001100000
1101	(0,4)(3,6)	(0,4)	100010000000
1110	(0,7)(1,5)	(0,7)	100000010000
1111	(0,2)(1,3)	(1,3)	010100000000

3.4 ส่วนของการสร้างสัญญาณสุ่มโดยใช้วงจรนับ LFSR

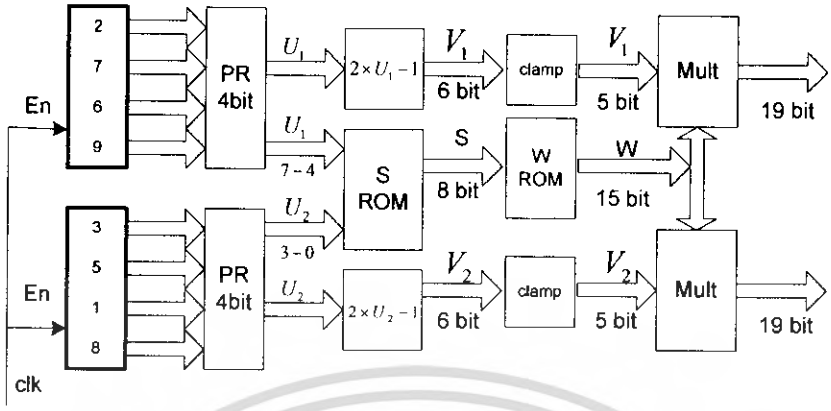
LFSR (Linear feedback shift register) เป็นทางเลือกหนึ่งในการสร้างวงจรนับไบนารี ซึ่งสามารถลดจำนวนวงจรลอจิกให้น้อยลงได้ดีกว่าวิธีอื่นๆ ผลของวงจรนับไบนารีของ LFSRโดยปกติแล้วจะมีลำดับของการนับเป็น $2^m - 1$ สภาวะ และมีลักษณะเป็นไบนารีแบบสุ่ม ทำให้ Register เหล่านี้มีชื่อเรียกอื่นๆ เช่น pseudo-random sequence generator (PRSG)

วัตถุประสงค์หลักๆ ในการใช้ LFSR ก็เพื่อให้มีการนับจำนวนใกล้เคียงกับสภาวะที่เป็นไปได้ $\left(2^m - 1\right)$ ให้มากที่สุด และมีการประยุกต์ใช้งานในเรื่องการเข้ารหัสแบบดิจิตอล (digital coding)

คุณสมบัติของ LFSR ในการออกแบบ

- LFSR ที่มีฟลิปฟลอบ (flip - flop) m ตัว สามารถที่จะสร้างจำนวนสภาวะ (state) เป็น $2^m - 1$ โดยที่สภาวะที่เป็นศูนย์จะไม่ยอมให้เกิดขึ้นเนื่องจาก LFSR จะไม่ทำงานได้ (หรือ counter locks up)
- เลือกใช้โพลิโนเมียลที่มั่นใจแล้วว่า $2^m - 1$ สภาวะจะต้องไม่ซ้ำกัน โดยจำเป็นจะต้องรู้ต้นแบบของโพลิโนเมียล
- ในการสร้างวงจรนับต้องคำนึงถึงโพลิโนเมียลที่เหมาะสม เช่น วงจรนับ 35 ต้องใช้โพลิโนเมียลที่มีอันดับเท่ากับ 6 ผลลัพธ์ที่เป็นไปได้คือ $2^6 - 1 = 63$

3.5 วิธีโพลาร์



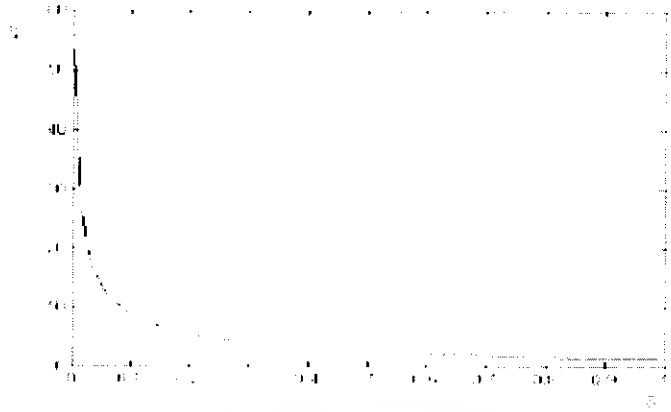
รูปที่ 3.4 แสดงแผนผังการทำงานของวิธีโพลาร์

1. ใช้ 8 LFSRs ที่เป็นอิสระเพื่อสร้างค่าจากการสุ่มอิสระ 2 ตัว คือ U_1 และ U_2 ซึ่งแต่ละตัวประกอบด้วย 4 บิท โดยบิททั้งหมดนี้จะอยู่ในส่วนที่เป็นเศษ ดังนั้นค่าที่เกิดขึ้นของ U_1 และ U_2 จะอยู่ในช่วง [0, 0.9375] ([.0000 , .1111] เมื่อเป็นระบบฐานสอง) ส่วนค่า V_1 และ V_2 จะต้องขึ้นอยู่กับค่า U_1 และ U_2 จากสมการข้างต้น และในการคิดค่า S จะได้ว่า

$$S = (2*U_1 - 1)^2 + (2*U_2 - 1)^2 \tag{3.3}$$

โดยค่า S จะถูกเก็บข้อมูลใน ROM ถ้าค่า $S \geq 1$ จะถูกเซตเป็นศูนย์

2. ในส่วนของค่า W ที่ถูกเก็บใน ROM จากรูปที่ 3.4 แสดงกราฟความสัมพันธ์ระหว่าง W กับ S โดยเมื่อค่า S อยู่ในช่วง [0.00390625, 0.99609375] และ ค่า W จะอยู่ในช่วง [54.2835, 0.0886] ในระบบฐานสองเมื่อเราใช้ W แทนด้วย 13 บิท จะได้ว่า 5 บิทสำหรับส่วนจำนวนเต็ม และ 8 บิท สำหรับส่วนเศษ



รูปที่ 3.5 กราฟความสัมพันธ์ระหว่าง W กับ S

3.6 ส่วนของการสลับตำแหน่งบิท

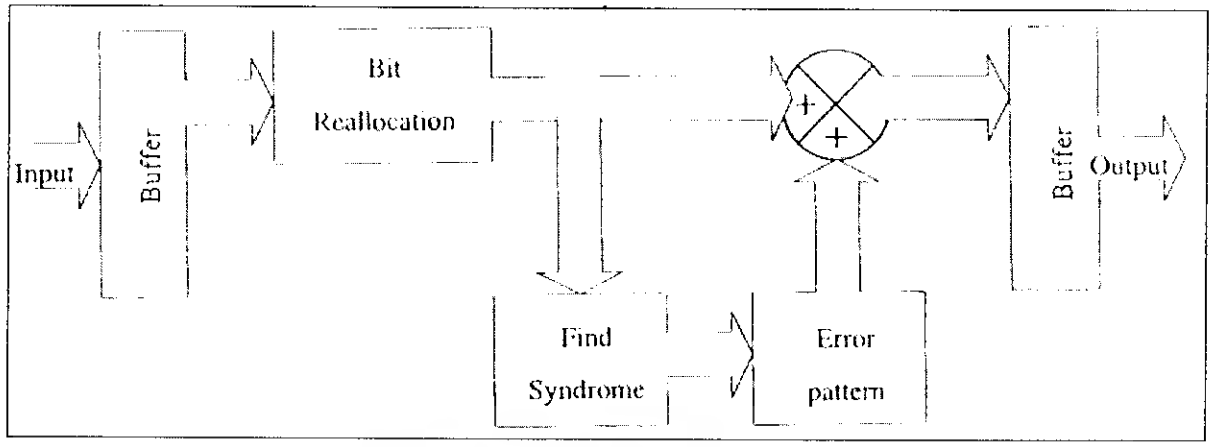
เป็นการสลับตำแหน่งของข้อมูลหลังจากที่รหัสคำที่ทำการบวกกับรูปแบบผิดพลาดแล้ว โดยจะทำการสลับเฉพาะ 8 บิทแรกเท่านั้น เพื่อเพิ่มความซับซ้อนในการป้องกันข้อมูลให้มีความปลอดภัยยิ่งขึ้น โดยการนำ 4 บิทหลังที่ได้จากรหัสคำมาเป็นตัวเลือกรูปแบบของการสลับตำแหน่งบิทข้อมูลซึ่งมีรูปแบบการสลับตำแหน่งบิท 16 รูปแบบ ดังแสดงในตารางที่ 3.2

ตารางที่ 3.2 การสลับตำแหน่งบิท

ตำแหน่งเดิม	0 LSB	1	2	3	4	5	6	7 MSB
0000	0	2	3	4	7	6	1	5
0001	2	4	6	1	0	3	7	5
0010	5	7	0	4	2	6	1	3
0011	6	4	0	3	7	5	1	2
0100	4	0	6	3	7	1	2	5
0101	5	1	6	2	4	3	7	0
0110	7	0	1	2	6	4	5	3
0111	6	2	1	0	7	3	4	5
1000	3	1	4	0	6	2	5	7
1001	2	5	4	1	6	3	0	7
1010	1	7	3	4	5	2	0	6
1011	5	1	6	2	7	0	3	4
1100	6	4	0	3	1	2	7	5
1101	7	0	1	2	3	4	5	6
1110	4	0	7	3	2	1	5	6
1111	5	0	4	1	2	3	7	6

3.7 การออกแบบตัวถอดรหัสลับ

ในการออกแบบตัวถอดรหัสลับเป็นการนำข้อมูลข่าวสารที่ผ่านการเข้ารหัสลับแล้วกลับคืนมาให้ได้ข้อมูลเดิม โดยทำการถอดรหัสข้อมูลข่าวสาร จากข่าวสารที่ทำการเข้ารหัสแล้วมีขนาด n บิท เข้ามาทางอินพุต โดยเริ่มจากการสลับตำแหน่งบิทของข้อมูลกลับ ให้ข้อมูลที่ได้มีตำแหน่งเหมือนเดิม โดยใช้ข้อมูล 4 บิทที่เพิ่มเข้ามาเป็นตัวเลือกรูปแบบที่จะทำการสลับตำแหน่งบิทกลับแล้วทำการคำนวณหาค่าซินโดรมที่จะมาเป็นตัวเลือกรูปแบบที่ผิดพลาดดังแสดงในตารางที่ 3.1 มาทำการบวกแบบเอกซคลูซีฟออร์กับข้อมูลข่าวสารนั้น ทำให้ได้ข้อมูลข่าวสารเดิมกลับคืนมา แบ่งเป็นขั้นตอนดังแสดงในรูปที่ 3.6



รูปที่ 3.6 แสดงแผนภาพการถอดรหัสลับ

3.8 ส่วนของการสลับตำแหน่งบิทกลับ

เป็นส่วนที่ทำหน้าที่ในการสลับตำแหน่งของบิทข้อมูล ที่เป็นรหัสคำที่ผิดกลับคืนมาให้ได้ตำแหน่งที่ถูกต้องตรงตามตำแหน่งเดิม โดยจะทำการสลับตำแหน่งข้อมูลกลับเฉพาะ 8 บิทแรกที่ถูกสลับไว้เท่านั้น และ 4 บิทที่เหลือจะเป็นตัวเลือกรูปแบบของการสลับตำแหน่งข้อมูลกลับ ซึ่งจะมีรูปแบบการสลับตำแหน่งบิทข้อมูล ดังแสดงในตารางที่ 3.3

ตารางที่ 3.3 การสลับตำแหน่งบิทกลับ

ตำแหน่งที่เปลี่ยนไป	0 LSB	1	2	3	4	5	6	7 MSB
0000	0	6	1	2	3	7	5	4
0001	4	3	0	5	1	7	2	6
0010	2	6	4	7	3	0	5	1
0011	2	6	7	3	1	5	0	4
0100	1	5	6	3	0	7	2	4
0101	7	1	3	5	4	0	2	6
0110	1	2	3	7	5	6	4	0
0111	3	2	1	5	6	7	0	4
1000	3	1	5	0	2	6	4	7
1001	6	3	0	5	2	1	4	7
1010	6	0	5	2	3	4	7	1
1011	5	1	3	6	7	0	2	4
1100	2	4	5	3	1	7	0	6
1101	1	2	3	4	5	6	7	0
1110	1	5	4	3	0	6	7	2
1111	1	3	4	5	2	0	7	6

3.9 ส่วนของการคำนวณหาซีนโดรม

ในส่วนของการคำนวณหาซีนโดรมประกอบไปด้วยเมทริกซ์ตรวจสอบพาริตี้ H ที่ถูกนำมาใช้ในการคำนวณหาค่าซีนโดรม สำหรับนำมาใช้ในการถอดรหัสที่ผิดไป โดยเมทริกซ์ตรวจสอบพาริตี้ H ประกอบขึ้นจากเมทริกซ์ทรานสโพสของเมทริกซ์พาริตี้ P ที่มีขนาด (n – k) x k และเมทริกซ์เอกลักษณ์ I_{n-k} คือ

$$H = [P^T : I_{4x4}] \tag{3.4}$$

P^T = Transpose ของ Parity

I_{4x4} = Matrix เฉียง ขนาด 4x4

เมทริกซ์ตรวจสอบพาริตี้ H แสดงในรูปที่ 3.7

$$H = \left[\begin{array}{cccccccc:cccc} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

รูปที่ 3.7 แสดงเมทริกซ์พาริตี้

ในการคำนวณหาซินโดรม S ได้จากการนำรหัสคำที่ได้รับ มาทำการคูณกับเมทริกซ์ทรานสโพสของ H ซึ่ง H^T เป็นเมทริกซ์โดย 4 แถวสุดท้ายของ H^T จะเป็นเมทริกซ์เอกลักษณ์ โดยใน 8 แถวแรก ได้จากการทำทรานสโพสเมทริกซ์ P ของเมทริกซ์ตัวกำเนิดโดยทั้ง 8 แถวจะแตกต่างกัน และ r เป็นรหัสที่ได้รับเข้ามาซึ่งสามารถทำการคำนวณได้ดังแสดงในสมการที่ 3.5

$$\text{Syndrome} = \text{r}\text{H}^{\text{T}} \tag{3.5}$$

ซินโดรมสามารถทำการหาได้ดังรูปที่ 3.8

$$S = [r_0, r_1, r_2, ..., r_{10}, r_{11}] \left[\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]$$

รูปที่ 3.8 แสดงการหาค่าซินโดรม

แต่เนื่องจากแต่ละแถวของเมทริกซ์ G และเมทริกซ์ H ตั้งฉากกัน จะให้อินเนอร์โปรดักส์เป็น 0 ก็หมายความว่า $uH^T = 0$ นั่นเอง ทำการแทนค่า r ในสมการที่ 3.5 ได้ว่า

$$Syndrome = (u \oplus e)H^T$$

$$S = uH^T \oplus eH^T \quad , \quad uH^T = 0 \tag{3.6}$$

$$S = eH^T$$

จากสมการที่ 3.5 และ 3.6 เป็นตัวบอกให้ทราบว่าค่าซีนโดรมที่คำนวณได้จากรหัสคำที่ผิดไปเนื่องจากรูปแบบผิดพลาด e ที่คูณกับ H^T นั้นให้ค่าซีนโดรมเหมือนกันกับการนำรูปแบบผิดพลาด e ไปคูณโดยตรงกับ H^T ด้วยเหตุนี้เอง ค่าของซีนโดรมจึงเป็นตัวบ่งว่ารหัส r ที่ได้รับ มีรูปแบบผิดพลาด e รูปแบบใดปรากฏอยู่ ดังนั้นรูปแบบการดึงรหัสคำที่ถูกส่ง u กลับคืนมา ทำได้โดยการนำ e ไปบวกแบบเอกซคลูซีฟออร์ กับรหัส r อีกครั้ง ดังแสดงในสมการที่ 3.7 ซึ่งทำให้สามารถทำการถอดรหัสได้ และได้ข้อมูลข่าวสารที่ถูกต้องกลับคืนมา.

$$r \oplus e = (u \oplus e) \oplus e$$

$$= u \oplus (e \oplus e)$$

$$= u$$

จากนั้น นำ u ไปถอด Generator Matrix ออก จะได้กลับมาเป็นสัญญาณ m เมื่อนำ m 8 บิต ไปเข้า D/A จะได้กลับมาเป็นสัญญาณ Analog ที่เราส่งมานั่นเอง

บทที่ 4
ภาษา VHDL และส่วนประกอบต่างๆของภาษา

ปัจจุบันในการออกแบบระบบดิจิตอลที่เรารู้จักกันจะเป็นการออกแบบโดยใช้การวาดวงจร (Capture Schematic) โดยใช้โปรแกรมช่วยในการวาด (Schematic entry tools) ซึ่งผู้ออกแบบจะต้องมีทักษะสูงในการออกแบบ และต้องใช้เวลามาก ในการออกแบบระบบจำลองการทำงาน (Simulation) ตลอดจนถึงการแก้ไขความถูกต้องของระบบ (Debugging) ซึ่งในการออกแบบจะต้องอ้างอิงเทคโนโลยีที่ใช้การออกแบบระบบดิจิตอล (Technology Dependent) ถ้าต้องการเปลี่ยนเทคโนโลยีของระบบที่ออกแบบค่อนข้างยากและใช้เวลามาก และเมื่อต้องการออกแบบระบบดิจิตอลที่มีความซับซ้อนสูง ยิ่งทำได้ยากหรือทำไม่ได้ โดยใช้การออกแบบเก่าๆ แต่ในการออกแบบระบบดิจิตอลในปัจจุบันได้มีกระบวนการออกแบบระบบรูปแบบใหม่ที่มีประสิทธิภาพสูง รวดเร็ว และไม่ยึดติดกับเทคโนโลยีที่ใช้ออกแบบ (Technology Independent) กระบวนการดังกล่าวคือ Top-down design ซึ่งกระบวนการดังกล่าว จะใช้ภาษาบรรยายฮาร์ดแวร์ HDL (Hardware Description Language) ในการออกแบบจำลองการทำงานสังเคราะห์วงจร (Synthesis) ในรูปแบบของเทคโนโลยีที่เราต้องการ และสามารถทดสอบวงจรที่ออกแบบได้บนฮาร์ดแวร์จำพวกชิป FPGA (Field Programmable Gate Arrays) or ASICs (Application Specific Integrated Circuits) ดังนั้นการออกแบบสามารถทำได้ง่ายและมีความสะดวกมากยิ่งขึ้น

4.1 VHDL

VHDL ย่อมาจาก VHSIC Hardware Description Language เป็นภาษาบรรยายฮาร์ดแวร์ประเภทหนึ่ง โดยภาษาเกิดขึ้นจากโครงการที่มีชื่อ VHSIC (Very High Speed Integrated Circuits) ที่ถูกพัฒนาขึ้นโดยกระทรวงกลาโหมของสหรัฐอเมริกา โดยเป้าหมายของโครงการนี้ก็เพื่อพัฒนาขีดความสามารถในการออกแบบวงจรรวมให้สูงขึ้นและสามารถทำได้ง่ายมากยิ่งขึ้นโดยมีเป้าหมายหลัก 2 ประการคือ

- ต้องการภาษาที่สามารถรองรับการออกแบบวงจรที่มีความซับซ้อนได้
- ต้องการภาษาที่เป็นมาตรฐานหรือเป็นภาษากลางที่ทำให้สามารถเผยแพร่ผลงานการออกแบบกัน
  ในกลุ่มนักออกแบบด้วยกันได้

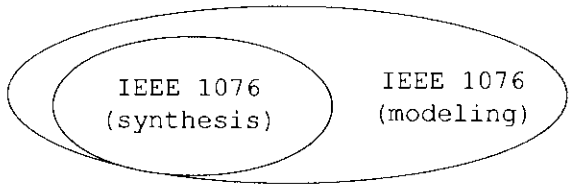
4.2 ประโยชน์ของภาษา VHDL

- เป็นภาษามาตรฐานสากลโดยรับรองจากสถาบัน IEEE ทำให้มีโปรแกรมและเครื่องมือต่างๆ และบริษัทที่สนับสนุนการทำงานมากมาย นอกจากนี้วงจรที่ออกแบบโดยภาษา VHDL ก็จะใช้งานได้นานเนื่องจากมีความเข้ากันได้ของภาษากับวงจรที่ได้รับการออกแบบใหม่
- ได้รับการสนับสนุนจากรัฐบาล เนื่องจากภาษา VHDL ได้รับการพัฒนาโดยกระทรวงกลาโหม ของสหรัฐอเมริกา
- เป็นภาษาที่ใช้งานจริงในอุตสาหกรรม เนื่องจากภาษา VHDL เป็นภาษาที่เป็นมาตรฐานจากสถาบัน IEEE จึงมีอุตสาหกรรมจำนวนมากที่นำภาษานี้ไปใช้ออกแบบ
- เป็นภาษาที่สามารถใช้ได้หลายระบบ การออกแบบโดยใช้ภาษา VHDL สามารถนำไปจำลองการทำงานหรือสังเคราะห์โดยซอฟต์แวร์ตัวใดก็ได้ที่รองรับภาษา VHDL จึงเป็นการออกแบบที่ไม่ยึดติดกับซอฟต์แวร์ที่ใช้ในการออกแบบ
- เป็นภาษาที่สามารถใช้จำลองรูปแบบการทำงานของวงจร ผู้ออกแบบวงจรสามารถออกแบบวงจรโดยใช้ภาษา VHDL ได้หลายระดับ ตั้งแต่ระดับมาโครบล็อก (Macro block) จนถึงระดับเกต (Gate) และสามารถออกแบบวงจรที่มีความซับซ้อนสูงและมีขนาดใหญ่ได้
- เป็นภาษาที่สามารถนำกลับมาใช้ใหม่ได้ วงจรที่ออกแบบโดยภาษา VHDL สารถนำกลับมาใช้ใหม่ได้ง่าย เนื่องจากสามารถเปลี่ยนแปลงแก้ไขวงจรได้ง่าย คล้ายคลึงกับโปรแกรมของซอฟต์แวร์
- เป็นภาษาที่สามารถนำไปใช้เป็นเอกสารประกอบได้ เป็นภาษาในรูปแบบพฤติกรรม ทำให้เราสามารถอธิบายการทำงานของวงจรภายในการออกแบบได้ทันที

4.3 ลักษณะการใช้งานภาษา VHDL

การใช้งานภาษา VHDL อาจจำแนกออกเป็น 5 ประเภทดังนี้คือ

- Document Language : เป็นภาษาที่ใช้บรรยายรายละเอียดการทำงานของวงจรที่ออกแบบ
- Design Language : เป็นภาษาที่ใช้สำหรับออกแบบวงจรที่มีความซับซ้อนสูง
- Verification Language : เป็นภาษาที่ใช้ตรวจสอบความถูกต้องของวงจรที่ออกแบบ
- Test Language : เป็นภาษาที่ใช้สำหรับการทำงานของวงจรที่ออกแบบ
- Synthesis Language : เป็นภาษาที่ใช้สำหรับสังเคราะห์วงจร (Synthesis) จริงแต่รูปแบบดังกล่าวนี้ไม่ได้ครอบคลุมทุกรูปแบบทั้งหมดของการเขียนในภาษา VHDL แต่มีเพียงบางรูปแบบเท่านั้นที่สามารถเขียนสามารถนำไปสังเคราะห์เป็นวงจรได้หรือเรียกว่าเป็นรูปแบบการเขียนสำหรับการสังเคราะห์ (VHDL for Synthesis)



รูปที่ 4.1 ขอบเขตของรูปแบบการเขียนสำหรับการสังเคราะห์วงจร

4.4 ส่วนประกอบของภาษา

ส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียนมี 4 หน่วยคือ

- หน่วยการออกแบบเอนทิตี (Entity Design unit)
- หน่วยการออกแบบสถาปัตยกรรม (Architecture Design unit)
- หน่วยการออกแบบแพกเกจ (Package Design unit)
- หน่วยการออกแบบโครงแบบ (Configuration Design unit)

4.4.1 หน่วยการออกแบบเอนทิตี (Entity Design unit)

หน่วยการออกแบบนี้เป็นส่วนที่ใช้สำหรับดิดต่อระหว่างภายนอกกับรูปแบบที่เขียนขึ้นโดยเป็นการกำหนดจุดเชื่อมต่อของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น รูปที่ 4.2 แสดงให้เห็นถึงโครงสร้างของหน่วยการออกแบบเอนทิตี

```
Entity component name
is
  Input and Output
ports
  Physical and other
parameter
End [component_name];
```

รูปที่ 4.2 โครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี

ส่วนนี้จะขึ้นต้นด้วยคำว่า Entity ... is ระหว่างคำทั้งสองคำเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component name) หลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและข้อมูล (input-output) รวมทั้งพารามิเตอร์อื่นๆ และที่สำคัญคือ หน่วยการออกแบบเอนทิตีจะต้องปิดท้ายด้วยคำว่า End และเครื่องหมายอัฒภาคเสมอ (;)

4.4.2 หน่วยการออกแบบสถาปัตยกรรม (Architecture Design unit)

หน่วยการออกแบบสถาปัตยกรรม คือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบในมุมมองของการจำลองการทำงาน พฤติกรรมต่างๆที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออกตรงช่องทาง ตลอดจนพารามิเตอร์ต่างๆที่กำหน่วยการออกแบบเอนทิตี รูปที่ 4.3 แสดงให้เห็นถึงโครงสร้างของหน่วยการออกแบบสถาปัตยกรรม

```
Architecture identifier of
company_name is
[declaration]
Begin
Specification of the functionality
Of the component in term of its
input lines and as influenced by
physical and other parameter
End [identifier];
```

รูปที่ 4.3 โครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม

ส่วนของหน่วยการออกแบบสถาปัตยกรรมเริ่มต้นด้วยคำว่า Architecture และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า Architecture นั้นใช้บรรยายการออกแบบเอนทิตีใดๆ (of <entity design unit> is) ส่วนที่อยู่ระหว่าง Architecture และ Begin เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (Architecture declaration area) ที่เป็นส่วนเผื่อเลือก (Option) ในบริเวณนี้ สามารถใช้เขียนประกาศกำหนดค่าต่างๆที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้าและไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง port) นั้นจะถูกบรรยายในบริเวณของเนื้อที่ระหว่างคำว่า Begin กับ End ของหน่วยการออกแบบสถาปัตยกรรมและนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (Concurrent statement) เท่านั้น คือทุก ๆ statement จะทำงานพร้อมกัน ลำดับก่อนหลังจะไม่มีผลต่อการทำงานของรูปแบบ หน่วยการออกแบบสถาปัตยกรรมจะต้องปิดท้ายด้วยคำสั่ง End และชื่อของสถาปัตยกรรมนั้น ๆ โดยทั่วไปการเขียนรูปแบบระบบเชิงเลขด้วยภาษา VHDL สามารถเขียนได้ในลักษณะต่าง ๆ ดังนี้

- ลักษณะการไหลของข้อมูล (Dataflow style)
- ลักษณะพฤติกรรม (Behavioral style)
- ลักษณะผสม (Mixed Model stylc)

- ลักษณะโครงสร้าง (Structural style)

ถ้าผู้ออกแบบต้องการออกแบบวงจรดิจิตอลใดๆจะต้องออกแบบหน่วยการออกแบบพื้นฐานก็คือส่วนของ Entity และส่วนของ Architecture ที่มีความสัมพันธ์กัน ซึ่งมีด้วยกันหลายรูปแบบสำหรับการออกแบบวงจรเดียวกัน และความเข้าใจของผู้ออกแบบเอง

4.4.3 หน่วยการออกแบบแพกเกจ

แพกเกจเป็นหน่วยการออกแบบที่ใช้เก็บข้อมูลต่างๆตลอดจนโปรแกรมย่อย (Subprogram) ที่เป็นประโยชน์ในการเขียนรูปแบบบรรยายวงจรดิจิตอล โดยข้อมูลใน Package สามารถถูกเรียกใช้ได้โดย Entity, Architecture หรือด้วย Package อื่นๆด้วยคำสั่ง USE Statement นอกจากนั้นสิ่งที่นิยมทำกันมากคือรูปแบบมาตรฐานต่างๆเช่น Standard components ต่างๆจะถูกเก็บไว้ใน Package ที่ทุกคนเรียกใช้งานได้ โดยสิ่งที่สามารถสร้างไว้ใน Package ได้แก่

- Subprogram
- Types
- Constants
- Signal
- Aliases
- Attributes
- Component
- Disconnection Specification

การเรียกใช้งาน Package จะใช้คำสั่ง USE Package โดยมีรูปแบบดังนี้

```
USE libary-name.package-name.item;
```

รูปที่ 4.4 โครงสร้างโดยทั่วไปของส่วน USE Package

แพกเกจจะแบ่งออกเป็น 2 ส่วน

1. Package Declaration

ส่วนที่มีความสำคัญที่สุดในแพกเกจ (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ ส่วนการประกาศแพกเกจ เพราะจะเป็นส่วนที่กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพกเกจสำหรับไปใช้ภายนอกของตัวแพกเกจเอง สิ่งใดๆที่ถูกประกาศไว้ในส่วนของบอดี้ แต่ไม่ได้ถูกประกาศไว้ในส่วนการประกาศแพกเกจจะมาสามารถถูกนำค่าและพฤติกรรมไปใช้ส่วนนอกบอดี้ ซึ่งสามารถเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของประกาศเอนทิตี คือจุดเชื่อมต่อ หรือพอร์ท ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพกเกจสามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้ และสามารถถูกนำไปใช้จากรูปแบบภายนอกได้ เช่น ใช้สำหรับประกาศชนิด (Type) หรือสัญญาณ เช่นเดียวกับส่วนบอดี้ แพกเกจที่ไม่จำเป็นต้องมีส่วนประกาศแพกเกจ แต่แพกเกจนั้นจะไม่สามารถนำไปจากรูปแบบภายนอกได้

```
PACKAGE package_name IS
    constant_declarations
    type_declarations
    signal_declarations
    component_declarations
END package_name;
```

รูปที่ 4.5 โครงสร้างโดยทั่วไปของส่วนประกาศแพกเกจ

2. Package Body

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆในรูปของคำสั่งลำดับ (Sequence) ที่ใช้บรรยายฟังก์ชั่นการทำงานของโปรแกรมย่อย (Subprogram) ทั้งหลาย ที่ชื่อของโปรแกรมย่อยนั้นที่ถูกประกาศไปในส่วนของการประกาศแพกเกจแล้วจะถูกเก็บไว้ในส่วนบอดี้แพกเกจ ทั้งร่วมทั้งการประกาศค่าคงที่ต่างๆอันได้แก่ค่าคงที่ที่ถูกประกาศชื่อก่อนในส่วนของประกาศแพกเกจ แต่ถูกกำหนดค่าในส่วนของบอดี้แพกเกจ ฉะนั้นส่วนของบอดี้แพกเกจจึงไม่จำเป็นต้องมีถ้าในส่วนประกาศแพกเกจไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อยหรือค่าคงที่ การเขียนบอดี้แพกเกจนั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 4.6

```
PACKAGE package_name IS

constant_declarations
      type_declarations
      subprogram_body
END package_name;
```

รูปที่ 4.6 โครงสร้างโดยทั่วไปของบอดี้แพกเกจ

4.4.4 หน่วยการออกแบบโครงแบบ

ดังที่ทราบกันแล้วว่ารูปแบบหนึ่งของระบบเชิงตัวเลขไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนทิตีได้เพียงหนึ่งเดียวเท่านั้น แต่หน่วยการออกแบบเอนทิตีหนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมีหน่วยการออกแบบโครงแบบมาเพื่อกำหนดการใช้โครงแบบ (Configuration) ประกอบเอนทิตีกับหน่วยการออกแบบสถาปัตยกรรมหน่วยไหนเข้าด้วยกัน

```
Configuration identifier of entity_name is
      Configuration_declarative_part
End;
```

รูปที่ 4.7 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

4.5 การเขียนแบบลำดับชั้น (Hierarchical Model)

ในการออกแบบวงจรดิจิตอลบางครั้งจำเป็นต้องมีการแบ่งวงจรเป็นบล็อกย่อยๆตามหน้าที่การทำงาน เนื่องจากระบบที่ต้องการออกแบบมีความซับซ้อนสูงไม่สามารถที่จะออกแบบได้เพียงบล็อกเดียว ทำให้จำเป็นต้องแบ่งวงจรออกเป็นบล็อกย่อยๆ ซึ่งภาษา VHDL ก็มีความสามารถในการเขียนแบบลำดับชั้น (Hierarchy) ได้ โดยเขียนบรรยายวงจรตั้งแต่ระดับบนสุด (Top model) ที่มีแต่ความสำพันธ์ของการเชื่อมต่อในแต่ละบล็อกย่อย โดยยังไม่มีรายละเอียดของวงจรในแต่ละบล็อก และในระดับล่างลงไปถึงจะมีรายละเอียดของวงจร

4.6 ไลบรารีในภาษา VHDL

เป็นส่วนที่เก็บข้อมูลต่างๆของ Design unit ที่ถูกทำการวิเคราะห์ตรวจสอบความถูกต้องตามการทำงานแล้ว (Compiled and simulated) เพื่อให้พร้อมที่จะนำไปใช้ได้กับ Design unit อื่นๆ โดยการอ้างถึงชื่อของ Library นั้นๆที่อ้างถึงได้และข้อมูลที่เก็บไว้ใน Library ได้จะแบ่งเป็น

- Primary units
  - Entity declarations
  - Package declarations
  - Configuration specification
- Secondary units
  - Architecture bodies
  - Package bodies

ส่วน Secondary units เป็นส่วนที่ต้องอ้างถึง Primary units ดังนั้นจะต้องคอมไพล์ในส่วน Primary units ก่อนเสมอ เช่น คอมไพล์ Package declarations ก่อนคอมไพล์ Package bodies หรือ คอมไพล์ Entity ก่อนคอมไพล์ Architecture โดยปกติแล้วจะมี Library อยู่ 2 ประเภทคือ

- Working Library หมายถึงไดเรกทอรี (Directory) ที่กำลังใช้งานอยู่ปกติจะถูกกำหนดให้เป็น Work เสมอ
- Resource Library หมายถึงไลบรารีเพิ่มเติมโดยสามารถตั้งชื่ออะไรก็ได้ ซึ่งในภาษา VHDL ได้สงวนชื่อของไลบรารีไว้ 2 ชื่อด้วยกัน STD และ WORK โดยใน STD ประกอบไปด้วย 2 package คือ
  - Package STANDARD กำหนดโดย IEEE 1076 โดยที่ภานในจะประกอบด้วยการประกาศ (Declaration) ต่างๆ เช่น VHDL type (integer, real, time, bit และ Boolean เป็นต้น)
  - Package TEXTIO ภายในประกอบด้วยโปรแกรมย่อยต่างๆที่สำหรับแก้ไขดัดแปลงข้อมูลจำพวก ASCII ไฟล์

การเรียกใช้งาน Library มีรูปแบบดังนี้

```
LIBRARY library_name, another_library_name;
```

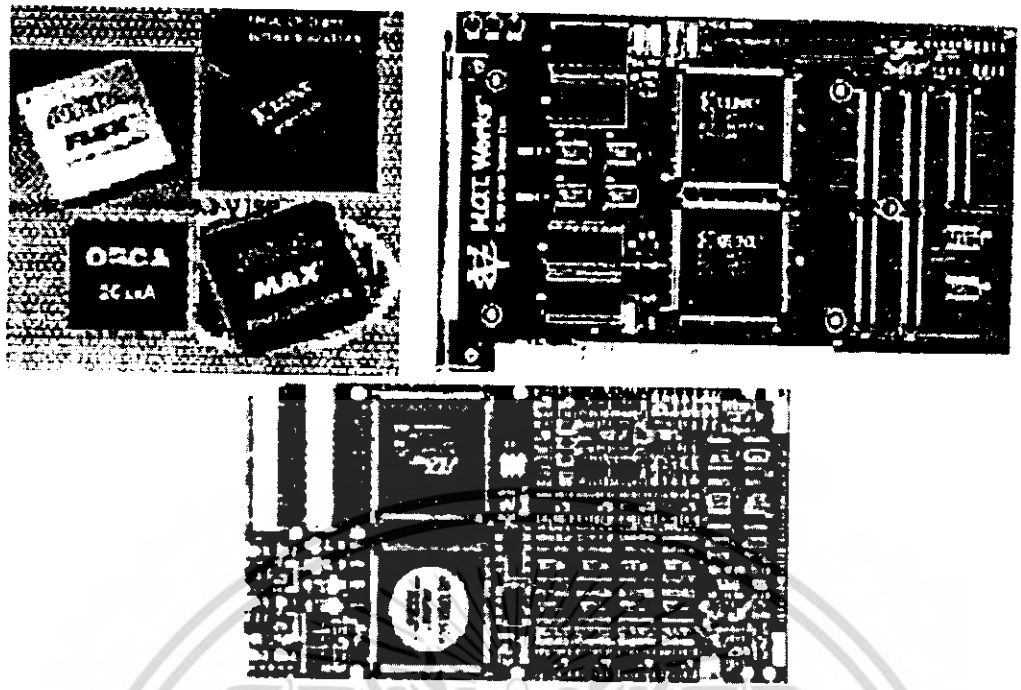
รูปที่ 4.8 โครงสร้างโดยทั่วไปของไลบรารี

4.7 สถาปัตยกรรมและการออกแบบบน FPGA

4.7.1 การออกแบบวงจรเชิงเฉพาะด้วยอุปกรณ์ FPGA

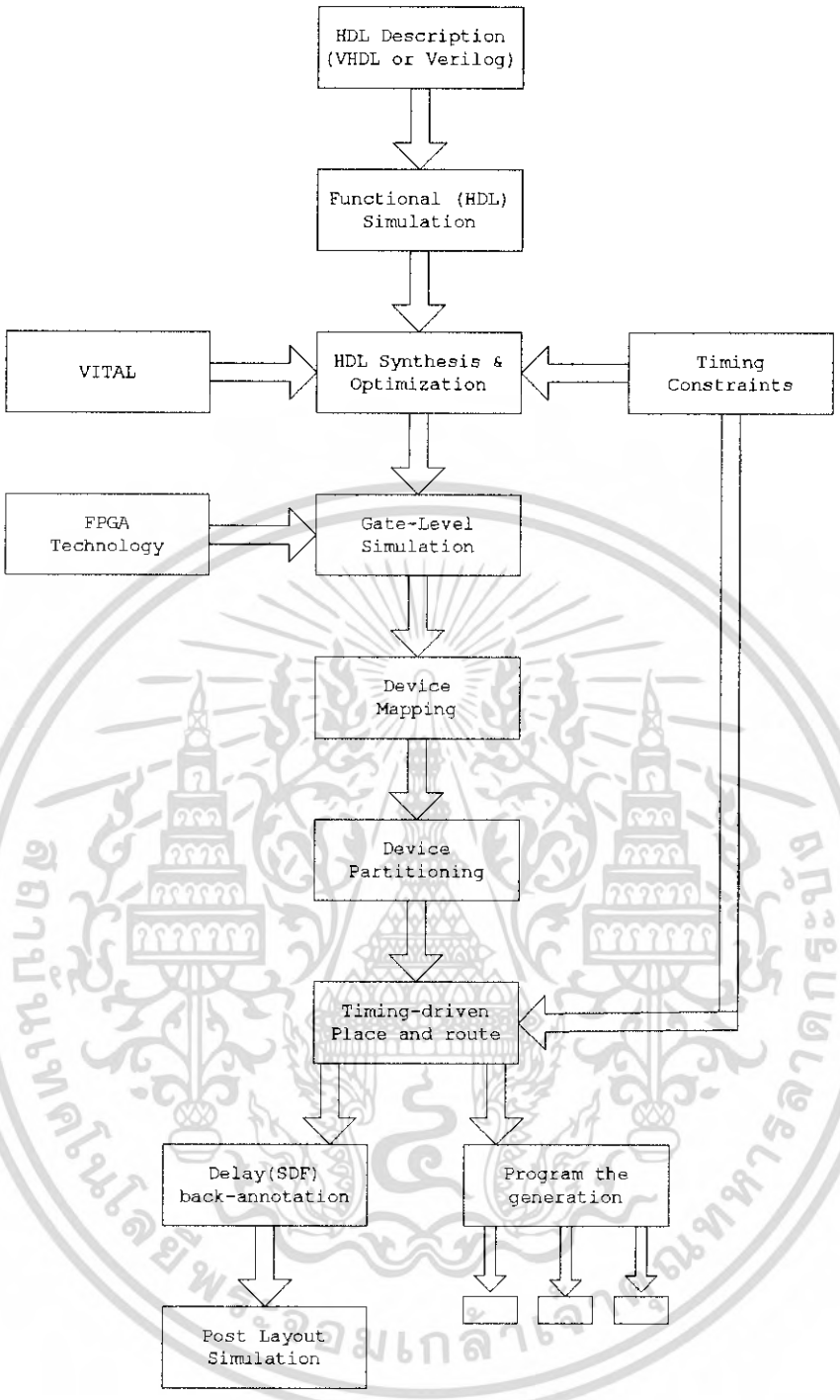
อุปกรณ์ FPGA (Field Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการทำงานตามที่ออกแบบไว้ ในการทำ FPGA ซึ่งเป็นวิธีการออกแบบ IC (Integrated Circuit) แบบ Semi-custom อีกวิธีหนึ่ง เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuits) แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือการทำ FPGA จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในอุปกรณ์ FPGA จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัด และการทำ FPGA ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดนั้นน้อยกว่าการทำ ASIC มากและการตรวจสอบหรือแก้ไขการออกแบบก็ทำได้สะดวก

การทำ FPGA ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิต อุปกรณ์ FPGA ได้เพิ่มความสามารถของอุปกรณ์ FPGA โดยเพิ่มจำนวนองค์ประกอบภายในหรือปรับปรุงโครงสร้างสถาปัตยกรรมภายใน และยังได้เพิ่มประสิทธิภาพของซอฟแวร์ที่ใช้ทำ PPR (Partitioning Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย ลักษณะของตัว FPGA และการนำไปใช้งานแสดง ดังรูปที่ 4.9



รูปที่ 4.9 ลักษณะของตัว FPGA และการนำไปใช้งาน

สำหรับตัวอุปกรณ์ FPGA นั้นก็มีโครงสร้างพื้นฐาน เทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นอุปกรณ์ FPGA ของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นอุปกรณ์ FPGA สามารถนำไปประยุกต์ใช้งานได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP: Digital Signal Processing) การออกแบบไมโครคอนโทรเลอร์ เป็นต้น โดยมีขั้นตอนในการออกแบบ ดังแสดงในรูปที่ 4.10



รูปที่ 4.10 ขั้นตอนการออกแบบโดยใช้อุปกรณ์ FPGA

4.7.1.1.1.1 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์

ในการออกแบบวงจรเชิงเลขนั้นทำได้โดยการวาดวงจร หรือใช้ภาษาฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASIC ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะแตกต่างกัน โดยที่การทำวิธีนี้จะต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไปจะเห็นได้ว่าการออกแบบโดยใช้ภาษาอธิบาย

ฮาร์ดแวร์ทำได้สะดวกว่า เพราะการทำด้วยวิธีนี้ไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ และที่สำคัญการออกแบบด้วยวิธีนี้สามารถที่จะแก้ไขโมเดล หรือ เปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่าเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

ในการเขียนโค้ด สิ่งที่ต้องคำนึงถึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และให้คุณสมบัติของวงจรตามที่กำหนด เพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับวงจรที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้นซอร์ฟแวร์สังเคราะห์วงจร (Synthesis Tools) จะทำการสังเคราะห์ตามโค้ดที่เขียน ถ้าอธิบายการทำงานของวงจรเดียวกัน แต่เขียนโค้ดในลักษณะที่ต่างกันเมื่อสังเคราะห์แล้วจะได้วงจรที่ต่างกัน และจากวงจรที่ต่างกัน เมื่อนำไปทำต้นแบบด้วย FPGA หรือการทำ ASIC แล้วจะได้ไอซีที่มีคุณสมบัติต่างกันทั้งในด้านของขนาดหรือความเร็ว (area and time) ส่วนการเขียนโค้ดลักษณะใดเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ในการออกแบบ

4.7.1.2 การจำลองการทำงานของวงจร (Simulation)

ขั้นตอนนี้เป็นขั้นตอนที่สำคัญเพราะเป็นขั้นตอนที่ใช้ตรวจสอบฟังก์ชั่น การทำงานของวงจรว่าถูก ต้องหรือไม่ มีข้อผิดพลาดตรงไหน เพื่อที่จะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สำหรับการทำการจำลองการทำงานของวงจร เช่น V-System และ MODELSIM ของบริษัท Model Technology

4.7.1.3 การสังเคราะห์วงจร

ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สังเคราะห์วงจร ทำการสังเคราะห์โค้ดเพื่อให้ได้เป็นวงจรขึ้นมา แต่ต้องตรวจสอบด้วยว่าซอร์ฟแวร์นั้นๆสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการใช้หรือไม่ โดย FPGA ที่นิยมใช้งานเช่นของบริษัท XILINX ตระกูล XC4000 และของบริษัท ALTERA ตระกูล FLEX 10 K ซอร์ฟแวร์สังเคราะห์วงจรที่นิยมใช้เช่นโปรแกรม Leonardo Speetrum ของบริษัท Exemplar Logic ซึ่งในขั้นตอนนี้ซอร์ฟแวร์สังเคราะห์วงจรจะแปลงโค้ดและทำการออปติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ นอกจากนี้ยังสามารถกำหนดข้อบังคับสำหรับวงจรได้เช่น ข้อบังคับในเรื่องของเวลา หรือข้อบังคับในเรื่องของพื้นที่ ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอนการออปติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการการออปติไมซ์คือการเทียบ (mapping) วงจรให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์ FPGA ในกรณีของ XILINX ตระกูล XC4000 และบริษัท ALTERA ตระกูล FLEX 10 K จะเทียบโดยใช้วิธี LUT (Look Up Table) เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอร์ฟแวร์สังเคราะห์วงจร ก็จะมีการรายงานผลว่าวงจรที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วงเท่าไร ใช้ทรัพยากรต่างใน FPGA อะไรบ้างเป็นต้น

4.7.1.4 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ให้เป็นส่วนย่อย สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือ ให้แต่ละส่วนที่จะแยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อช่วยลดความหนาแน่นในการทำเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอร์ฟแวร์ทำ โดยซอร์ฟแวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate) , ฟลิบฟลอบ (flip flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ภายในอุปกรณ์ FPGA (CLBs, IOBs, BUFT และ edge decoder) หลังจากทำขั้นตอนนี้เสร็จแล้วสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ที่ใช้งาน เช่น FPGA ของบริษัท XILINX จะใช้ XILINX Foundation Series 2.1i ซึ่งซอร์ฟแวร์นี้ตัวนี้จะรวมเอาซอร์ฟแวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement and Routing) เป็นไปอย่างต่อเนื่อง ส่วน FPGA ของบริษัท ALTERA จะใช้ ALTERA MAX + II

4.7.1.5 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Placement) มาแล้วว่าควรจะอยู่ในตำแหน่งใดในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทางได้ง่าย หรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือตัว Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

4.7.1.6 การเชื่อมต่อสัญญาณ (Routing)

ขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA เช่นระหว่าง CLBs หรือระหว่าง CLBs กับ IOBs ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอร์ฟแวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด หรือเกิดความหน่วงเกินค่าที่กำหนดในข้อบังคับ โดยสามารถทำขั้นตอนนี้ได้โดยใช้ซอร์ฟแวร์ เช่นกัน หรือจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (manual layout) ก็ได้ แต่ทางที่ดีควรใช้ซอร์ฟแวร์ทำดีกว่าโดยให้ทำการค้นหาเส้นทางหลายๆ ครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา (time constraints) จะช่วยให้ผลที่ได้จากการทำการเชื่อมต่อสัญญาณดีขึ้นได้

4.7.1.7 การโปรแกรมอุปกรณ์ FPGA (Configuration)

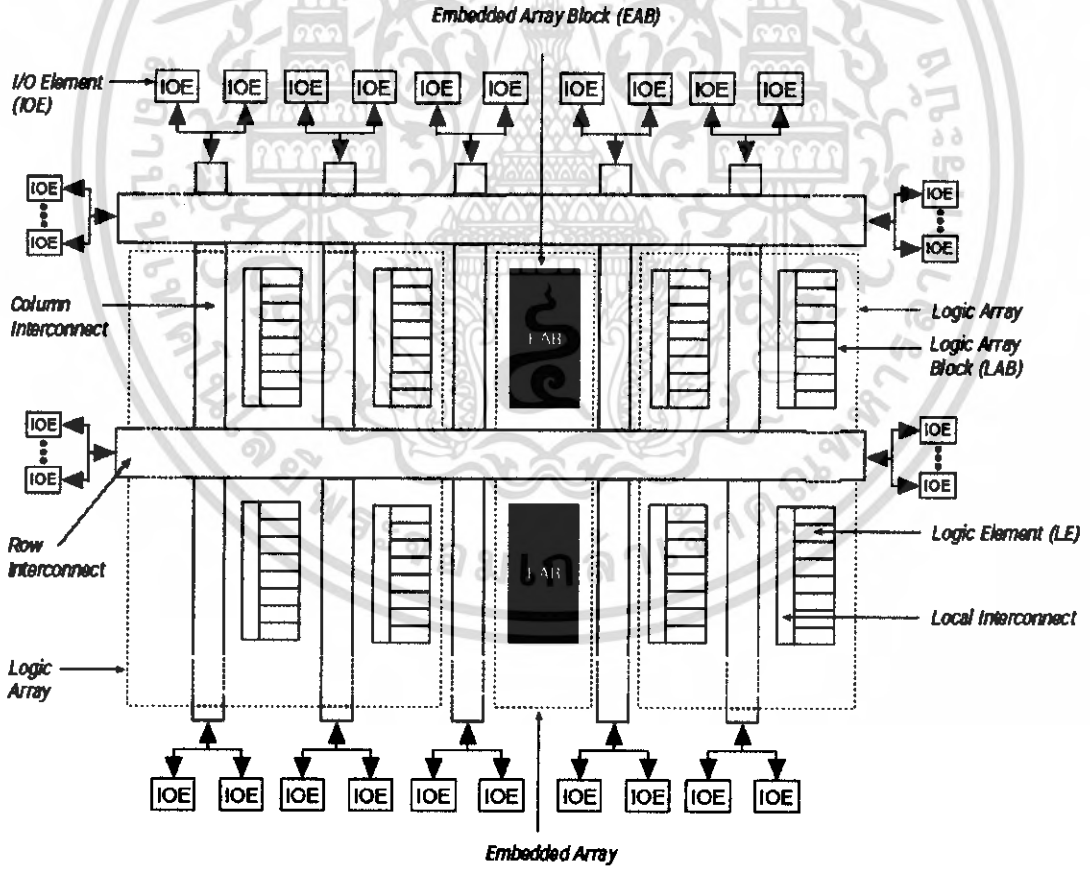
หลังจากที่วงจรผ่านขั้นตอนต่างๆจนกระทั่งผ่านการทำ PPR (Partitioning Placement and Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลข่าวสารวงจร (Configuration data) ซึ่งอยู่ในรูปของบิทสตรีม (Bit-stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการทำงานตามวงจรที่ออกแบบไว้

จากที่อธิบายมาทั้งหมดจะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้น ทำได้สะดวกกว่าการทำ ASIC มากเพราะใช้เวลาน้อยกว่ามาก ส่วนสำคัญที่ใช้ในการทำ FPGA คือซอร์ฟแวร์ที่ใช้ตั้งแต่การเขียนได้คอธิบายฮาร์ดแวร์ จนกระทั่งดาวน์โหลดลงในอุปกรณ์ FPGA ซึ่งซอร์ฟแวร์ที่ใช้ต้องเป็นซอร์ฟแวร์ที่ใช้ทำงานต่อเนื่องกัน

4.8 สถาปัตยกรรมภายในของ FPGA

4.8.1 รายละเอียดของ POWER ACEX1K SERIES

FPGA ตระกูล ACEX1K ของบริษัท ALTERA เป็น FPGA ที่มีโครงสร้างภายในเป็นแบบ SRAM-Based FPGA ใช้เทคโนโลยีในการโปรแกรมเหมือนกับหน่วยความจำแบบ SRAM (Static RAM) ทำให้การโปรแกรมสามารถทำซ้ำได้โดยไม่จำกัดจำนวนครั้ง และใช้เวลาในการโปรแกรมชิป FPGA น้อยมาก (ระดับ nsec) การโปรแกรมทำได้ง่ายเช่นเดียวกับการเขียน SRAM ทั่วไป แต่ข้อเสียคือ มันต้องการไฟเลี้ยงในการเก็บวงจรที่ได้ออกแบบไว้



รูปที่ 4.11 โครงสร้างภายในของ FPGA ตระกูล ACEX1K

ตารางที่ 4.1 คุณลักษณะต่างๆของชิป FPGA ที่ใช้กับบอร์ดในกลุ่ม POWER ACEX1K SERIES

Feature	EP1K10	EP1K30	EP1K50	EP1K100
Typical gates	10,000	30,000	50,000	100,000
Maximum system gates	56,000	119,000	199,000	257,000
Logic elements (LEs)	576	1,728	2,880	4,992
EABs	3	6	10	12
Total RAM bits	12,288	24,576	40,960	49,152
Maximum user I/O pins	136	171	249	333

4.8.2 JTAG CONNECTOR

JTAG Connector เป็น Header ตัวผู้แบบ 5×2 ขา ใช้สำหรับต่อกับสาย Byte Blaster MV เพื่อดาวน์โหลด ข้อมูลทางลอจิกของวงจรที่ได้ออกแบบไว้จากคอมพิวเตอร์ลงสู่ชิปเอฟพีจีเอ

ในการโปรแกรมจะมี Jumper ให้เลือกว่าจะโปรแกรมลงในชิป FPGA หรือโปรแกรมลง Configuration Device

ในกรณีที่ต้องการโปรแกรมลงชิป FPGA ACEX 1K สามารถดูวิธีการ Complier และโปรแกรมลง Configuration Device ด้วยโปรแกรม MAX+PLUS II ให้ดูวิธีการใช้ MAX+PLUS II

4.8.3 Configuration Device Socket

Socket แบบ PLCC 20 ขา สำหรับใส่ Configuration Device ซึ่งเป็นอุปกรณ์แบบ Flash Memory ใช้สำหรับเก็บข้อมูลวงจรที่เราได้ออกแบบเอาไว้ เมื่อเริ่มต้นจ่ายไฟให้แก่บอร์ด POWER ACEX 1K SERIES วงจรที่อยู่ภายใน Configuration Device จะถูกโหลดลงไปเก็บไว้ในชิป FPGA โดยอัตโนมัติ Configuration Device ที่ใช้ร่วมกับ FPGA แบบ SRAM-BASE FPGA สำหรับเบอร์ของ Configuration Device ที่ใช้กับบอร์ด POWER ACEX 1K SERIES จะเป็นเบอร์ EPC2LI20 หรือ EPC2LC20 มีขนาด 1,695,680 ×1 Bit สามารถใช้กับแรงดัน 5.0V หรือ 3.3V ก็ได้ ในกรณีที่ไม่ต้องการให้ FPGA โหลดวงจรจาก Configuration Device ให้เลือก JUMPER J10 มาที่ตำแหน่ง OFF ลักษณะและตำแหน่งขาของ EPC2LI20 และ EPC2LC20

4.8.4 พอร์ตขยายช่องสัญญาณแบบ Header 13×2 ขา จำนวน 4 ตัว

บอร์ด POWER ACEX 1K SERIES แต่ละรุ่นจะมีพอร์ตขยายช่องสัญญาณ J6, J7, J8 และ J9 ให้ใช้งานได้อย่างอิสระ 4 พอร์ต แต่ละพอร์ตจะมีขนาด 13×2 ขา ต่อกับขา I/O ของ FPGA โดยตรง

ตารางที่ 4.2 ความสัมพันธ์ของ J6 กับตำแหน่ง I/O ของ ACEX1K FPGA

J6 Pin Number	ACEX1K Pin Number	J6 Pin Number	ACEX1K Pin Number
1	7(I/O)	2	8(I/O)
3	9(I/O)	4	10(I/O)
5	11(I/O)	6	12(I/O)
7	13(I/O)	8	14(I/O)
9	17(I/O)	10	18(I/O)
11	19(I/O)	12	20(I/O)
13	21(I/O)	14	22(I/O)
15	23(I/O)	16	26(I/O)
17	27(I/O)	18	28(I/O)
19	29(I/O)	20	30(I/O)
21	31(I/O)	22	32(I/O)
23	33(I/O)	24	36(I/O)
25	37(I/O)	26	GND

ตารางที่ 4.3 ความสัมพันธ์ของ J7 กับตำแหน่ง I/O ของ ACEX1K FPGA

J7 Pin Number	ACEX1K Pin Number	J7 Pin Number	ACEX1K Pin Number
1	38(I/O)	2	39(I/O)
3	41(I/O)	4	42(I/O)
5	43(I/O)	6	44(I/O)
7	46(I/O)	8	47(I/O)
9	48(I/O)	10	49(I/O)
11	51(I/O)	12	54(Ded.Input)
13	56(Ded.Input)	14	59(I/O)
15	60(I/O)	16	62(I/O)
17	63(I/O)	18	64(I/O)
19	65(I/O)	20	67(I/O)
21	68(I/O)	22	69(I/O)
23	77(I/O)	24	72(I/O)
25	73(I/O)	26	GND

ตารางที่ 4.4 ความสัมพันธ์ของ J8 กับตำแหน่ง I/O ของ ACEX1K FPGA

J8 Pin Number	ACEX1K Pin Number	J8 Pin Number	ACEX1K Pin Number
1	78(I/O)	2	79(I/O)
3	80(I/O)	4	81(I/O)
5	82(I/O)	6	83(I/O)
7	86(I/O)	8	87(I/O)
9	88(I/O)	10	89(I/O)
11	90(I/O)	12	91(I/O)
13	92(I/O)	14	95(I/O)
15	96(I/O)	16	97(I/O)
17	98(I/O)	18	99(I/O)
19	100(I/O)	20	101(I/O)
21	102(I/O)	22	109(I/O)
23	110(I/O)	24	111(I/O)
25	112(I/O)	26	GND

ตารางที่ 4.5 ความสัมพันธ์ของ J9 กับตำแหน่ง I/O ของ ACEX1K FPGA

J9 Pin Number	ACEX1K Pin Number	J9 Pin Number	ACEX1K Pin Number
1	113(I/O)	2	114(I/O)
3	116(I/O)	4	117(I/O)
5	118(I/O)	6	119(I/O)
7	120(I/O)	8	121(I/O)
9	122(I/O)	10	124(Ded.Input)
11	126(Ded.Input)	12	128(I/O)
13	130(I/O)	14	131(I/O)
15	132(I/O)	16	133(I/O)
17	135(I/O)	18	136(I/O)
19	137(I/O)	20	138(I/O)
21	140(I/O)	22	141(I/O)
23	142(I/O)	24	143(I/O)
25	144(I/O)	26	GND

4.8.5 Module Oscillator

บอร์ด POWER ACEX 1K SERIES มีโมดูลออสซิลเลเตอร์ความถี่ 25.175 MHz (สามารถถอดและเปลี่ยนเป็นค่าความถี่ที่ต้องการได้) เป็นแหล่งกำเนิดความถี่สำหรับใช้ในการออกแบบวงจรที่ต้องการอ้างอิงกับสัญญาณนาฬิกา โดยการป้อนเข้าที่ขา 55 (GCLK1) ของ FPGA

ตารางที่ 4.6 ความสัมพันธ์ระหว่างสัญญาณ Oscillator กับตำแหน่งขาของ ACEX1K FPGA

Module Oscillator Pin Number	ACEX1K Pin Number
CLK	55(GCLK1)

4.8.6 Terminal แบบ 2 ขั้ว สำหรับ GCLK1 และ GCLK2

Terminal ตัวผู้แบบ 2 ขั้ว ใช้สำหรับต่อกับสัญญาณนาฬิกาจากภายนอกเพื่อป้อนให้แก่ FPGA ทางขา Global Clock 1 (GCLK1) หรือขา Global Clock 2 (GCLK2)

ตารางที่ 4.7 ความสัมพันธ์ของ T4, T5 กับตำแหน่งขาของ ACEX1K FPGA

T4 Pin Number	ACEX1K Pin Number	T5 Pin Number	ACEX1K Pin Number
1(+)	125(GCLK2)	1(+)	55(GCLK1)
2(-)	GND	2(-)	GND

4.8.7 Terminal แบบ 2 ขั้ว สำหรับแรงดัน 2.5 โวลท์, 3.3 โวลท์ และ 5 โวลท์

Terminal ตัวผู้แบบ 2 ขั้วสำหรับต่อกับแหล่งจ่ายแรงดันภายในบอร์ด POWER ACEX1K SERIES เพื่อเป็นแหล่งจ่ายแรงดันไฟตรงให้กับวงจรภายนอก โดยมีแรงดันให้เลือกใช้งาน 3 ระดับคือ 2.5 โวลท์, 3.3 โวลท์ และ 5.0 โวลท์

ตารางที่ 4.8 ความสัมพันธ์ของ T1, T2 และ T3 กับแหล่งจ่ายแรงดันภายในบอร์ด ACEX1K SERIES

T1 Pin Number	Volt	T2 Pin Number	Volt	T3 Pin Number	Volt
1(+)	5 Volt	1(+)	3.3 Volt	1(+)	2.5 Volt
2 (-)	GND	2 (-)	GND	2 (-)	GND

4.8.8 คอนเนคเตอร์แหล่งจ่ายไฟ

บอร์ด POWER ACEX1K SERIES ถูกออกแบบมาให้ใช้กับไฟกระแสตรงแรงดัน 9 โวลท์ ถึง 12 โวลท์ โดยแกนในของคอนเนคเตอร์ J1 เป็นขั้วบวกและแกนนอกเป็นขั้วลบ นอกจากนี้ยังมีวงจรกลับขั้วแรงดันสำหรับในกรณีที่มีการป้อนแรงดันผิดขั้วให้แก่บอร์ดอีกด้วย

บทที่ 5

การทดลองและผลการทดลอง

5.1 ผลการทดลอง

การทดลองตัวเข้ารหัสและถอดรหัส สามารถทำได้โดยการนำเอาสัญญาณที่อยู่ในรูปแบบต่างๆ มาทำการเข้ารหัสและถอดรหัส ทำได้โดยการส่งสัญญาณดิจิตอล 8 บิท เข้าไปทำการเข้ารหัสในบอร์ดประมวลผล FPGA และส่งผลของการเข้ารหัสนั้นมาในรูปแบบสัญญาณดิจิตอล 12 บิท และยังมีสัญญาณดิจิดอล 8 บิท ที่ผ่านการถอดรหัสแล้ว ส่งมาทาง RS-232 ด้วยเหมือนกัน ซึ่งการส่งข้อมูลและการแสดงผลต่างๆได้กระทำบนโปรแกรม Matlab

ในการทดลองนั้น ได้ส่งข้อมูลเป็นดิจิตอลที่มีลักษณะเป็นตัวอักษรภาษาอังกฤษ เข้าไปทำการเข้ารหัสและแสดงผลของการเข้ารหัสและผลของการถอดรหัสบนหน้าจอของโปรแกรม Matlab ซึ่งได้ผลของการทดลองออกมาดังนี้

INSTRUMENTATION ENGINEERING

รูปที่ 5.1 ข้อมูลต้นแบบที่ทำการส่งเข้าไป

Ë¹éÒÍ¡ãËèá¹è¹ ÅÕÅÒÊØ´ÂÍ´ äÁèà«ç¹à«ÍÃì¡éÍ§«×éÍÊ

รูปที่ 5.2 ข้อมูลต้นแบบที่ถูกทำการเข้ารหัส

INSTRUMENTATION ENGINEERING

รูปที่ 5.3 ข้อมูลที่ทำการถอดรหัสแล้ว

จะเห็นได้ว่าการนำข้อมูลที่เป็นสัญญาณดิจิตอลไปทำการเข้ารหัสแล้ว จะทำให้ข้อมูลมีลักษณะผิดเพี้ยนไปจากต้นแบบ และเมื่อข้อมูลที่ผิดเพี้ยนนั้น ได้ถูกนำไปเข้าสู่กระบวนการถอดรหัสแล้ว ก็สามารถทำให้ข้อมูลนั้น กลับมาเป็นเหมือนกับข้อมูลต้นแบบที่ส่งมา และทำให้ผู้รับสามารถเข้าใจถึงความหมายที่ต้องการจะสื่อถึงของข้อมูลนั้นๆ

บทที่ 6
สรุปผลและแนวทางการพัฒนา

6.1 สรุปผล

จากการทำงานที่ผ่านมา เป็นการนำเสนอรูปแบบทางอัลกอริทึมและการนำมาสร้างเป็นแผงวงจรตัวเข้ารหัสลับและตัวชุดถอดรหัสลับ โดยอาศัยหลักการของบล็อคโค้ดเชิงเส้น ในการสร้างความปลอดภัยให้กับข้อมูลข่าวสาร ด้วยการนำรูปแบบผิดพลาดที่ถูกกำหนดรูปแบบไว้แล้วมาทำการบวกแบบ Exclusive – OR เข้ากับรหัสคำ ในส่วนที่เป็นข้อมูลแล้วนำรหัสคำที่ได้มาทำการสลับบิทในส่วนของข้อมูล โดยมีรหัสแก้ไขเป็นตัวเลือกรูปแบบการสลับทางด้านการถอดรหัสใช้วิธีการคำนวณหาค่าซินโดรมมาทำการถอดรหัสกลับคืนมาโดยวิธีการทั้งหมด ได้กล่าวละเอียดในบทที่ 3

ในการทำวิจัยนี้ ได้ศึกษาและนำมาสร้างชุดเข้ารหัส และ ชุดถอดรหัส โดยการเขียนโปรแกรม VHDL ลงบนบอร์ด FPGA

6.2 ปัญหาและข้อเสนอแนะ

การเข้ารหัสด้วยวิธีของ Linear Block Code นั้น สามารถถูกถอดรหัสได้ หากผู้รับมีตัวถอดรหัสอยู่ ดังนั้น ในการส่งข้อมูลที่เข้ารหัสด้วยวิธีการนี้ หากว่าผู้ที่เราไม่ต้องการให้ได้รับข้อมูลมีตัวถอดรหัสข้อมูลที่เป็นตัวถอดรหัสของ Linear Block Code อยู่ในมือ จะทำให้บุคคลนั้นสามารถถอดรหัสและล่วงรู้ถึงข้อมูลที่เราส่งไปได้ ดังนั้น ผู้ที่ทำการเข้ารหัสด้วยวิธีการนี้ จึงควรเก็บรักษาโปรแกรม VHDL ที่ใช้ ไว้เป็นอย่างดี ควรเก็บไว้ให้มีเฉพาะผู้ส่งและผู้รับเท่านั้น

บรรณานุกรม

[1] ชำนาญ ปัญญาใส, วัชรากร หนูทอง, "ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล", ซีเอ็ดยูเคชั่น: กรุงเทพฯ, 2547

[2] พรชัย ภววงษ์ศักดิ์, "การประมวลผลสัญญาณดิจิตอลเบื้องต้น", มหาวิทยาลัยเทคโนโลยีมหานครhttp://www.ee.mut.ac.th/home/pornchai, 2543

[3] มนัส สังวรศิลป์, วรรัตน์ ภัทรอมรกุล, "คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์". กรุงเทพฯ : สำนักพิมพ์อินโฟเพรส, 2543.

[4] วิวัฒน์ กิรานนท์, "วิศวกรรมการสื่อสาร", พิมพ์ครั้งที่ 3. กรุงเทพฯ : อักษรสยามการพิมพ์, 2544.

[5] Y. Fan and Z. Zilic, "A novel scheme of implementing high speed AWGN eommunication channel emulators in FPGAS," IEEE Circuits and Systems, 2004. ISCAS'04., Vol.2, pp. II 877-II 880, May 2004

[6] Algebraic Codes for Data Transmission by Richard Blahut

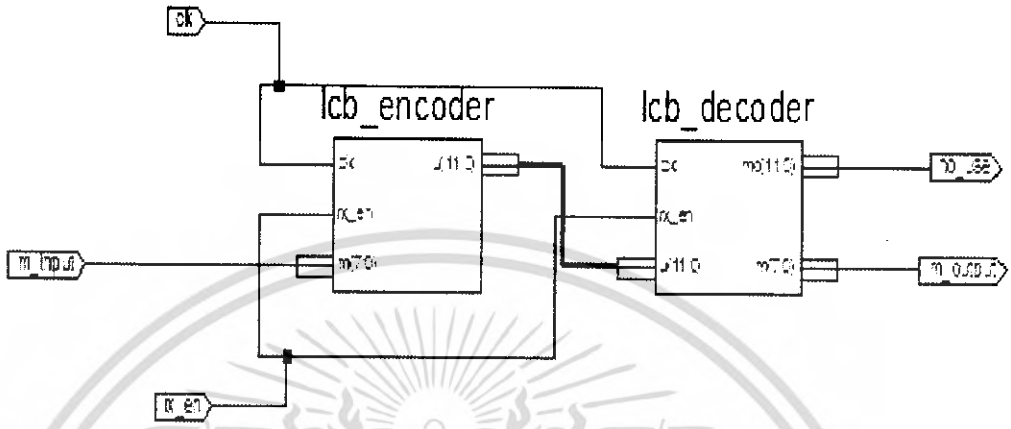
[7] การแก้รหัสที่ผิด รศ.ดร. ฟูศักดิ์ ชีวสุวิทย์

[8] Stanford University Lecture By John Gill

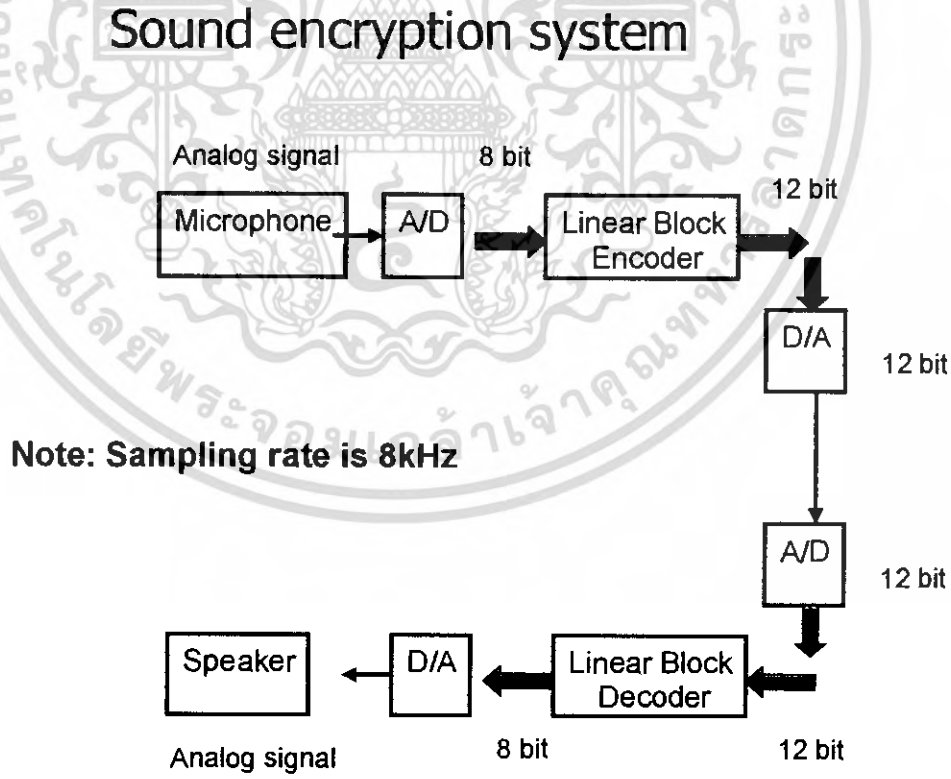
[9] Data Communication & Networking by Behrouz A. Forouzan

ภาคผนวก

ภาคผนวก Schematic ที่โปรแกรมลงบน FPGA



ภาคผนวก การเข้ารหัสสัญญาณโทรศัพท์



ภาคผนวก โปรแกรม VHDL ของ Encoder

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;
Entity lcb_encoder is
Port( clk    : in  std_logic;
    rx_en  : in  std_logic;
    m      : in  std_logic_vector( 7 downto 0);
    u      : out std_logic_vector(11 downto 0));
End;

Architecture rtl of lcb_encoder is
    signal poly_2a  : std_logic_vector(1 downto 0);
    signal poly_3a  : std_logic_vector(2 downto 0);
    signal poly_5a  : std_logic_vector(4 downto 0);
    signal poly_6a  : std_logic_vector(5 downto 0);
 Begin
   Process (clk,m)
    variable state  : integer range 0 to 7;

    variable wait_acc  : integer range 0 to 15;

    variable acc_u,acu : std_logic_vector(11 downto 0);
    variable err       : std_logic_vector(11 downto 0);

    variable acc_lfsr  : std_logic_vector(3 downto 0);

    variable acc_a  : std_logic_vector(11 downto 0);
    variable acc_b  : std_logic_vector(11 downto 0);
    variable acc_c  : std_logic_vector(11 downto 0);
    variable acc_d  : std_logic_vector(11 downto 0);
    variable acc_e  : std_logic_vector(11 downto 0);
    variable acc_f  : std_logic_vector(11 downto 0);
    variable acc_g  : std_logic_vector(11 downto 0);
    variable acc_h  : std_logic_vector(11 downto 0);


     Begin

       if clk'event and clk='1' then
          case state is
                when 0      => if rx_en = '1' then
                             state:=1;
                           end if;
                             wait_acc:=0;
```

```
when 1     => state:=2;

            if   m(7)  = '1' then
                 acc_a :="100000000011";
            else acc_a :="000000000000";
            end if;

            if   m(6)  = '1' then
                 acc_b :="010000001001";
            else acc_b :="000000000000";
            end if;

             if   m(5)  = '1' then
                  acc_c  :="001000001100";
            else acc_c  :="000000000000";
            end if;

             if   m(4)  = '1' then
                  acc_d  :="000100000110";
            else acc_d  :="000000000000";
            end if;

            if   m(3)  = '1' then
                 acc_e  :="000010001110";
            else acc_e  :="000000000000";
            end if;

            if   m(2)  = '1' then
                 acc_f  :="000001000111";
            else acc_f  :="000000000000";
            end if;

            if   m(1)  = '1' then
                 acc_g  :="000000101011";
            else acc_g  :="000000000000";
            end if;

             if   m(0)  = '1' then
                  acc_h  :="000000011101";
            else acc_h  :="000000000000";
            end if;

            acc_u:=(acc_a xor acc_h xor
                  acc_b xor acc_g xor
                  acc_c xor acc_f xor
                  acc_d xor acc_e);
```

```
when 2      => state:=3;
        if poly_2a="00" then
          poly_2a(0)<='1';
        else
          poly_2a(0) <= poly_2a(1);
          poly_2a(1) <=(poly_2a(1) xor poly_2a(0));
          acc_lfsr(0):= poly_2a(1);
        end if;

        if poly_3a="000" then
          poly_3a(0)<='1';
        else
          poly_3a(1 downto 0) <= poly_3a(2 downto 1);
          poly_3a(2)          <=(poly_3a(2) xor poly_3a(0));
          acc_lfsr(1)         := poly_3a(2);
        end if;

        if poly_5a="00000" then
          poly_5a(0)<='1';
        else
          poly_5a(3 downto 0) <= poly_5a(4 downto 1);
          poly_5a(4)          <=(poly_5a(4) xor poly_5a(1));
          acc_lfsr(2)         := poly_5a(4);
        end if;

        if poly_6a="000000" then
          poly_6a(0)<='1';
        else
          poly_6a(4 downto 0) <= poly_6a(5 downto 1);
          poly_6a(5)          <=(poly_6a(5) xor poly_5a(0));
          acc_lfsr(3)         := poly_6a(5);
        end if;

when 3 =>     case acc_lfsr is
            when  "0000"  => err:="000000000000";
            when  "0001"  => err:="001000010000";
            when  "0010"  => err:="010000100000";
            when  "0011"  => err:="000010010000";
            when  "0100"  => err:="100001000000";
            when  "0101"  => err:="000010100000";
            when  "0110"  => err:="000000110000";
            when  "0111"  => err:="010010000000";
            when  "1000"  => err:="000110000000";
            when  "1001"  => err:="000011000000";
            when  "1010"  => err:="000001010000";
            when  "1011"  => err:="001001000000";
            when  "1100"  => err:="000001100000";
            when  "1101"  => err:="100010000000";
            when  "1110"  => err:="100000010000";
```

```
            when others    => err:="010100000000";
          end case;
            acc_u:=(err xor acc_u);
            state:=4;

when 4 =>      state:=5;
           case acc_u(3 downto 0) is
  when   "0000"  => acu(4):=acc_u(4);

           acu(5):=acc_u(6);

           acu(6):=acc_u(7);

           acu(7):=acc_u(8);

           acu(8):=acc_u(11);

           acu(9):=acc_u(10);

           acu(10):=acc_u(5);

           acu(11):=acc_u(9);
  when   "0001"  => acu(4):=acc_u(6);

           acu(5):=acc_u(8);

           acu(6):=acc_u(10);

           acu(7):=acc_u(5);

           acu(8):=acc_u(4);

           acu(9):=acc_u(7);

           acu(10):=acc_u(11);

           acu(11):=acc_u(9);
  when   "0010"  => acu(4):=acc_u(9);

           acu(5):=acc_u(11);

           acu(6):=acc_u(4);

           acu(7):=acc_u(8);

           acu(8):=acc_u(6);

           acu(9):=acc_u(10);

           acu(10):=acc_u(5);
```

```
            acu(11):=acc_u(7);
when   "0011"  => acu(4):=acc_u(10);

            acu(5):=acc_u(8);

            acu(6):=acc_u(4);

            acu(7):=acc_u(7);

            acu(8):=acc_u(11);

            acu(9):=acc_u(9);

            acu(10):=acc_u(5);

            acu(11):=acc_u(6);
when   "0100"  => acu(4):=acc_u(8);

            acu(5):=acc_u(4);

            acu(6):=acc_u(10);

           acu(7):=acc_u(7);

            acu(8):=acc_u(11);

            acu(9):=acc_u(5);

            acu(10):=acc_u(6);

            acu(11):=acc_u(9);
when   "0101"  => acu(4):=acc_u(9);

            acu(5):=acc_u(5);

            acu(6):=acc_u(10);

             acu(7):=acc_u(6);

            acu(8):=acc_u(8);

            acu(9):=acc_u(7);

            acu(10):=acc_u(11);

            acu(11):=acc_u(4);
```

```
when   "0110"  => acu(4):=acc_u(11);
          acu(5):=acc_u(4);
          acu(6):=acc_u(5);
          acu(7):=acc_u(6);
          acu(8):=acc_u(10);
          acu(9):=acc_u(8);
          acu(10):=acc_u(9);
          acu(11):=acc_u(7);
when   "0111"  => acu(4):=acc_u(10);
          acu(5):=acc_u(6);
          acu(6):=acc_u(5);
          acu(7):=acc_u(4);
          acu(8):=acc_u(11);
          acu(9):=acc_u(7);
          acu(10):=acc_u(8);
          acu(11):=acc_u(9);
when   "1000"  => acu(4):=acc_u(7);
          acu(5):=acc_u(5);
          acu(6):=acc_u(8);
          acu(7):=acc_u(4);
          acu(8):=acc_u(10);
          acu(9):=acc_u(6);
          acu(10):=acc_u(9);
          acu(11):=acc_u(11);
```

```
when  "1001"  => acu(4):=acc_u(6);
        acu(5):=acc_u(9);
        acu(6):=acc_u(8);
       acu(7):=acc_u(5);
        acu(8):=acc_u(10);
        acu(9):=acc_u(7);
        acu(10):=acc_u(4);
        acu(11):=acc_u(11);
 when  "1010"  => acu(4):=acc_u(5);
        acu(5):=acc_u(11);
        acu(6):=acc_u(7);
      acu(7):=acc_u(8);
        acu(8):=acc_u(9);
        acu(9):=acc_u(6);
        acu(10):=acc_u(4);
        acu(11):=acc_u(10);
when  "1011"  => acu(4):=acc_u(9);
        acu(5):=acc_u(5);
        acu(6):=acc_u(10);
      acu(7):=acc_u(6);
        acu(8):=acc_u(11);
        acu(9):=acc_u(4);
        acu(10):=acc_u(7);
        acu(11):=acc_u(8);
```

```
when  "1100"  => acu(4):=acc_u(10);
                 acu(5):=acc_u(8);
                 acu(6):=acc_u(4);
                acu(7):=acc_u(7);
                 acu(8):=acc_u(5);
                 acu(9):=acc_u(6);
                 acu(10):=acc_u(11);
                 acu(11):=acc_u(9);
when  "1101"  => acu(4):=acc_u(11);
                 acu(5):=acc_u(4);
                 acu(6):=acc_u(5);
                acu(7):=acc_u(6);
                 acu(8):=acc_u(7);
                 acu(9):=acc_u(8);
                 acu(10):=acc_u(9);
                 acu(11):=acc_u(10);
when  "1110"  => acu(4):=acc_u(8);
                 acu(5):=acc_u(4);
                 acu(6):=acc_u(11);
                acu(7):=acc_u(7);
                 acu(8):=acc_u(6);
                 acu(9):=acc_u(5);
                 acu(10):=acc_u(9);
                 acu(11):=acc_u(10);
```

```
                when others    => acu(4):=acc_u(9);

                                  acu(5):=acc_u(4);

                                  acu(6):=acc_u(8);

                                 acu(7):=acc_u(5);

                                  acu(8):=acc_u(6);

                                  acu(9):=acc_u(7);

                                  acu(10):=acc_u(11);

                                  acu(11):=acc_u(10);
                                 end case;
                                      acu(3 downto 0):=acc_u(3 downto 0);

                when others =>        u<=acu;
                                  if wait_acc = 15 then
                                    state:=0;
                                  end if;
                                    wait_acc:=wait_acc+1;


            end case;
          end if;
      End Process;
End;
```

ภาคผนวก โปรแกรม VHDL ของ Decoder

```
library ieee;
Use ieee.std_logic_1164.ALL;
Use ieee.std_logic_Unsigned.ALL;

Entity lcb_decoder is
Port( clk    : in  std_logic;
     rx_en  : in  std_logic;
     u      : in  std_logic_vector( 11 downto 0);
     mo     : out std_logic_vector( 11 downto 0);
     m      : out std_logic_vector(  7 downto 0));
End;

Architecture rtl of lcb_decoder is
 Begin

   Process (clk,u,rx_en)
    variable state  : integer range 0 to 7;
    variable sy       : std_logic_vector(3 downto 0);
    variable acu,acc_u,acc_uu,err : std_logic_vector(11 downto 0);
    variable Ha,Hb,Hc,Hd    : std_logic_vector(11 downto 0);

     Begin
                        Ha:="011010111000";
                        Hb:="001111010100";
                        Hc:="100111100010";
                        Hd:="110001110001";

       if clk'event and clk='1' then
          case state is
               when 0     => if rx_en = '1' then
                        state:=1;
                        acc_u:=u;
                       end if;

              when 1      => state:=2;

                    case acc_u(3 downto 0) is
                when   "0000"  => acu(4):=acc_u(4);

                        acu(6):=acc_u(5);

                        acu(7):=acc_u(6);

                        acu(8):=acc_u(7);

                        acu(11):=acc_u(8);
```

```
            acu(10):=acc_u(9);
            acu(5):=acc_u(10);
            acu(9):=acc_u(11);
        when   "0001"  => acu(6):=acc_u(4);
            acu(8):=acc_u(5);
            acu(10):=acc_u(6);
           acu(5):=acc_u(7);
            acu(4):=acc_u(8);
            acu(7):=acc_u(9);
            acu(11):=acc_u(10);
            acu(9):=acc_u(11);
        when   "0010"  => acu(9):=acc_u(4);
            acu(11):=acc_u(5);
            acu(4):=acc_u(6);
            acu(8):=acc_u(7);
            acu(6):=acc_u(8);
            acu(10):=acc_u(9);
            acu(5):=acc_u(10);
            acu(7):=acc_u(11);

          when   "0011"  => acu(10):=acc_u(4);
            acu(8):=acc_u(5);
            acu(4):=acc_u(6);
             acu(7):=acc_u(7);
            acu(11):=acc_u(8);
```

```
            acu(9):=acc_u(9);
            acu(5):=acc_u(10);
            acu(6):=acc_u(11);

        when  "0100"  => acu(8):=acc_u(4);
            acu(4):=acc_u(5);
            acu(10):=acc_u(6);
            acu(7):=acc_u(7);
            acu(11):=acc_u(8);
            acu(5):=acc_u(9);
            acu(6):=acc_u(10);
            acu(9):=acc_u(11);

        when  "0101"  => acu(9):=acc_u(4);
            acu(5):=acc_u(5);
            acu(10):=acc_u(6);
            acu(6):=acc_u(7);
            acu(8):=acc_u(8);
            acu(7):=acc_u(9);
            acu(11):=acc_u(10);
            acu(4):=acc_u(11);

        when  "0110"  => acu(11):=acc_u(4);
            acu(4):=acc_u(5);
            acu(5):=acc_u(6);
            acu(6):=acc_u(7);
```

```
            acu(10):=acc_u(8);
            acu(8):=acc_u(9);
            acu(9):=acc_u(10);
            acu(7):=acc_u(11);

        when  "0111"  => acu(10):=acc_u(4);
            acu(6):=acc_u(5);
            acu(5):=acc_u(6);
            acu(4):=acc_u(7);
            acu(11):=acc_u(8);
            acu(7):=acc_u(9);
            acu(8):=acc_u(10);
            acu(9):=acc_u(11);

        when  "1000"  => acu(7):=acc_u(4);
            acu(5):=acc_u(5);
            acu(8):=acc_u(6);
            acu(4):=acc_u(7);
            acu(10):=acc_u(8);
            acu(6):=acc_u(9);
            acu(9):=acc_u(10);
            acu(11):=acc_u(11);

        when  "1001"  => acu(6):=acc_u(4);
            acu(9):=acc_u(5);
            acu(8):=acc_u(6);
```

```
            acu(5):=acc_u(7);
            acu(10):=acc_u(8);
            acu(7):=acc_u(9);
            acu(4):=acc_u(10);
            acu(11):=acc_u(11);

        when  "1010"  => acu(5):=acc_u(4);
            acu(11):=acc_u(5);
            acu(7):=acc_u(6);
           acu(8):=acc_u(7);
            acu(9):=acc_u(8);
            acu(6):=acc_u(9);
            acu(4):=acc_u(10);
            acu(10):=acc_u(11);

       when  "1011"  => acu(9):=acc_u(4);
            acu(5):=acc_u(5);
            acu(10):=acc_u(6);
            acu(6):=acc_u(7);
            acu(11):=acc_u(8);
            acu(4):=acc_u(9);
            acu(7):=acc_u(10);
            acu(8):=acc_u(11);

      when  "1100"  => acu(10):=acc_u(4);
            acu(8):=acc_u(5);
```

```
            acu(4):=acc_u(6);
            acu(7):=acc_u(7);
            acu(5):=acc_u(8);
            acu(6):=acc_u(9);
            acu(11):=acc_u(10);
            acu(9):=acc_u(11);

when  "1101" => acu(11):=acc_u(4);
            acu(4):=acc_u(5);
            acu(5):=acc_u(6);
           acu(6):=acc_u(7);
            acu(7):=acc_u(8);
            acu(8):=acc_u(9);
            acu(9):=acc_u(10);
            acu(10):=acc_u(11);

 when  "1110" => acu(8):=acc_u(4);
            acu(4):=acc_u(5);
            acu(11):=acc_u(6);
           acu(7):=acc_u(7);
            acu(6):=acc_u(8);
            acu(5):=acc_u(9);
            acu(9):=acc_u(10);
            acu(10):=acc_u(11);
```

```
when others    => acu(9):=acc_u(4);
                  acu(4):=acc_u(5);
                  acu(8):=acc_u(6);
                  acu(5):=acc_u(7);
                  acu(6):=acc_u(8);
                  acu(7):=acc_u(9);
                  acu(11):=acc_u(10);
                  acu(10):=acc_u(11);
                   end case;
                              acu(3 downto 0):=acc_u(3 downto 0);

          when 2 =>     state:=3;
                        mo<=acu;
 sy(3):=((Ha(11) and acu(11))xor(Ha(10) and acu(10))xor(Ha(9) and acu(9))xor
(Ha(8) and acu(8))xor(Ha(7) and acu(7))xor(Ha(6) and acu(6))xor
 (Ha(5) and acu(5))xor(Ha(4) and acu(4))xor(Ha(3) and acu(3))xor
 (Ha(2) and acu(2))xor(Ha(1) and acu(1))xor(Ha(0) and acu(0)));

 sy(2):=((Hb(11) and acu(11))xor(Hb(10) and acu(10))xor(Hb(9) and acu(9))xor
  (Hb(8) and acu(8))xor(Hb(7) and acu(7))xor(Hb(6) and acu(6))xor
  (Hb(5) and acu(5))xor(Hb(4) and acu(4))xor(Hb(3) and acu(3))xor
  (Hb(2) and acu(2))xor(Hb(1) and acu(1))xor(Hb(0) and acu(0)));

 sy(1):=((Hc(11) and acu(11))xor(Hc(10) and acu(10))xor(Hc(9) and acu(9))xor
(Hc(8) and acu(8))xor(Hc(7) and acu(7))xor(Hc(6) and acu(6))xor
 (Hc(5) and acu(5))xor(Hc(4) and acu(4))xor(Hc(3) and acu(3))xor
```

```
(Hc(2) and acu(2))xor(Hc(1) and acu(1))xor(Hc(0) and acu(0)));


sy(0):=((Hd(11) and acu(11))xor(Hd(10) and acu(10))xor(Hd(9) and acu(9))xor
   (Hd(8) and acu(8))xor(Hd(7) and acu(7))xor(Hd(6) and acu(6))xor
   (Hd(5) and acu(5))xor(Hd(4) and acu(4))xor(Hd(3) and acu(3))xor
  (Hd(2) and acu(2))xor(Hd(1) and acu(1))xor(Hd(0) and acu(0)));

            when 3 =>     state:=4;
                      case sy is
                         when   "0000"  => err:="000000000000";
                         when   "0001"  => err:="001000010000";
                         when   "0010"  => err:="010000100000";
                         when   "0011"  => err:="000010010000";
                         when   "0100"  => err:="100001000000";
                         when   "0101"  => err:="000010100000";
                         when   "0110"  => err:="000000110000";
                         when   "0111"  => err:="010010000000";
                         when   "1000"  => err:="000110000000";
                         when   "1001"  => err:="000011000000";
                         when   "1010"  => err:="000001010000";
                         when   "1011"  => err:="001001000000";
                         when   "1100"  => err:="000001100000";
                         when   "1101"  => err:="100010000000";
                         when   "1110"  => err:="100000010000";
                         when others    => err:="010100000000";
                       end case;
                         acc_uu:=(err xor acu);

            when others =>     state:=0;
                         m(7 downto 0)<=acc_uu(11 downto 4);
            end case;
        end if;
     End Process;
End;
```