การออกแบบระบบถอดรหัสรีด-โซโลมอน ในโดเมนเวลาที่สามารถปรับเปลี่ยนความเร็วได้

A DESIGN OF RECONFIGURABLE SPEED TIME-DOMAIN REED-SOLOMON DECODER

ปรเมตต์ ปราสาร
PARAMATE PRASARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2558

การออกแบบระบบถอดรหัสรีด – โซโลมอน ในโดเมนเวลาที่สามารถปรับเปลี่ยนความเร็วได้

A DESIGN OF RECONFIGURABLE SPEED TIME-DOMAIN REED-SOLOMON DECODER

ปรเมตต์ ปราสาร
PARAMATE PRASARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2558

การออกแบบระบบถอดรหัสรีด – โซโลมอน ในโดเมนเวลาที่สามารถปรับเปลี่ยนความเร็วได้

A DESIGN OF RECONFIGURABLE SPEED TIME-DOMAIN REED-SOLOMON DECODER

ปรเมตต์ ปราสาร
PARAMATE PRASARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2558

A DESIGN OF RECONFIGURABLE SPEED TIME-DOMAIN REED-SOLOMON DECODER

PARAMATE PRASARN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN ELECTRONICS ENGINEERING
FACULTY OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2015

คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การออกแบบระบบถอดรหัสรีด-โซโลมอนในโดเมนเวลาที่สามารถปรับเปลี่ยนความเร็วได้
Thesis Title A Design of Reconfigurable Speed Time-Domain Reed - Solomon Decoder
นักศึกษา นายปรเมตต์ ปราสาร
รหัสประจำตัว 53610801
ปริญญา วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์
อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.ดร.สมศักดิ์ ชุมช่วย
หมายเลขวิทยานิพนธ์ KMITL-2015-EN-M-040-024

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.เมธา	คงพูน	
ผศ.ดร.พินิจ	กำหอม	
รศ.ดร.พรชัย	ทรัพย์นิธิ	
รศ.ดร.สมศักดิ์	ชุมช่วย	

วัน / เดือน/ ปี ที่สอบ วันอังคารที่ 17 กุมภาพันธ์ พ.ศ. 2558 เวลา 13.00-15.00 น.
สถานที่สอบ ณ อาคาร A ชั้น 5 ห้องประชุม 1

คณะวิศวกรรมศาสตร์ รับรองแล้ว

(รองศาสตราจารย์ ดร. คมสัน มาลีสี)

คณบดี คณะวิศวกรรมศาสตร์

วันที่ 17 กุมภาพันธ์ พ.ศ. 2558

หัวข้อวิทยานิพนธ์	การออกแบบระบบถอดรหัสรีด – โซโลมอน ซึ่งปรับความเร็วในการทำงานได้
นักศึกษา	นายปรเมตต์ ปราสาร
รหัสประจำตัว	53610801
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมอิเล็กทรอนิกส์
พ.ศ.	2558
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.สมศักดิ์ ชุมช่วย

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ นำเสนอระบบถอดรหัสรีด – โซโลมอนบนโดเมนเวลา ซึ่งสามารถถูกปรับความเร็วในการทำงานได้ โดยในขั้นแรกเริ่มจากการศึกษาระบบการเข้า และถอดรหัสรีด – โซโลมอน โดยละเอียด ทั้งในด้านอัลกอริทึม และการอิมพลิเมนต์บนฮาร์ดแวร์ ในขั้นตอนต่อมา นำอัลกอริทึมที่ศึกษาทั้งการเข้า และถอดรหัสมาอิมพลิเมนต์บนฮาร์ดแวร์ โดยการเข้ารหัสนั้นได้ทำการเข้ารหัสโดยใช้รีจิสเตอร์เป็นวงจรมอดูโล ส่วนการถอดรหัส ได้ทำการอิมพลิเมนต์ระบบถอดรหัสบนโดเมนเวลาแบบไปป์ไลน์ ด้วยการปรับจำนวนของบล็อกที่ใช้คำนวณรีจิสเตอร์ของการถอดรหัสในแต่ละรอบ โดยปกตินั้นในการคำนวณเพื่อถอดรหัสในระบบโดเมนเวลาจะมีการคำนวณทั้งหมด 2t รอบ เมื่อ t คือระยะต่ำสุดของรหัส งานวิจัยนี้จึงได้นำเสนอวิธีการพัฒนาระบบการถอดรหัส ซึ่งช่วยในการปรับจำนวนของบล็อกย่อยในการคำนวณแต่ละรอบ และความเร็วในการถอดรหัสได้ พร้อมทั้งนำเสนอ และวิเคราะห์จำนวนเกตที่ใช้ในระบบ เวลาในการคำนวณ ความเร็วสูงสุด และพลังงาน เพื่อใช้งานในแต่ละแอพลิเคชัน ที่มีความจำกัดในด้านทรัพยากรต่าง ๆ กันไป

Thesis	A Design of Reconfigurable Speed Time-Domain Reed-Solomon Decoder
Student	Mr.Paramate Prasarn
Student ID.	53610801
Degree	Master of Engineering
Program	Electronics Engineering
Year	2015
Thesis Advisor	Assoc.Prof.Dr.Somsak Choomchuay

ABSTRACT

This thesis proposes a time-domain Reed-Solomon decoding system with reconfigurable speed. In the first step, we research about RS codes. At the first step acquire a knowledge of RS codes intensively in both algorithm and hardware implementation. Next step is to implement both encoding and decoding algorithms in hardware by utilizing registers as modulus circuit for encoding. The decoder on time-domain which the speed of decoding can be adjust. The process of time domain pipeline decoding development starts with adjust number of block that use as compute the decoder registers in each round. In general, time domain decoder takes 2t rounds calculation when t is minimum distance of the codes.

The research presents the decoder that capable to adjust speed of decoder and number of calculation block. Also diagnose amount of logic gate, calculation time, maximum speed and power for each limit resource application.

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จได้ด้วยความกรุณาจากท่านอาจารย์ที่ปรึกษา รศ.ดร.สมศักดิ์ ชุมช่วย ที่ให้ความช่วยเหลือ ให้คำชี้แนะช่วยแก้ไขปัญหาตลอดจนให้ความรู้ ประสบการณ์ และแนวคิดในการทำงานที่ดีแก่ข้าพเจ้า

สำหรับคุณงามความดีอันใดที่เกิดจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดา ซึ่งเป็นที่รักและเคารพยิ่ง ตลอดจนครูอาจารย์ที่เคารพทุกท่านที่ได้ประสิทประสาทวิชาความรู้ และถ่ายทอดประสบการณ์ และแนวคิดที่ดีให้แก่ข้าพเจ้า

ปรเมตต์ ปราสาร

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

รหัสรีด – โซโลมอน (RS – codes) เป็นวิธีการที่สำคัญอย่างหนึ่ง ซึ่งใช้กันอย่างแพร่หลายในวิธีการแก้ไขข้อมูลผิดพลาดสำหรับข้อมูลที่ไม่ใช่เลขฐานสอง (non-binary) ซึ่งได้ถูกนำมาประยุกต์ใช้งานอย่างแพร่หลายมาเป็นเวลานานหลายปี ไม่ว่าจะเป็น เทคโนโลยีการบันทึกข้อมูลต่างๆ[1] แผ่นบันทึกเสียง[2] ลายน้ำภาพดิจิตอล[3] การกระจายสัญญาณวิดีโอ[4] การสื่อสารแบบไร้สาย[5] การสื่อสารผ่านดาวเทียม[6] ต่างก็ใช้ รหัสรีด – โซโลมอน สำหรับต้นแบบระบบนั้น ในด้านฮาร์ดแวร์ส่วนมากได้ถูกนำเสนอให้ออกแบบด้วยเอฟพีจีเอ (FPGA)

การออกแบบเอฟพีจีเอ ในแต่ละการใช้งานนั้น ก็มีความต้องการ และข้อจำกัดต่างๆกันไป โดยที่บางงานนั้น ต้องการความเร็วสูง และบางงานนั้น มีข้อจำกัดด้านทรัพยากร การออกแบบวงจรให้เหมาะสมกับความต้องการในแต่ละการใช้งาน การออกแบบเอฟพีจีเอ จึงมีความสำคัญเป็นอย่างมาก เพื่อให้ได้ตรงตามความต้องการของผู้ใช้งาน หรือผู้ผลิตอุปกรณ์ต่างๆ ที่ต้องการระบบแก้ไขข้อมูลผิดพลาด ไม่ว่าจะเป็นงานที่ต้องการความเร็วสูง หรืองานที่ต้องการทรัพยากรด้านฮาร์ดแวร์ที่มีจำกัด การถอดรหัสรีด – โซโลมอน ใช้บล็อคสัญญาณข้อมูลดิจิตอลในการเติมบิตสำรอง ค่าผิดพลาดปรากฏระหว่างการส่งผ่าน หรือ การเก็บรักษาข้อมูลด้วยเหตุผลหลายประการ (ยกตัวอย่างเช่น สัญญาณรบกวน หรือสัญญาณสอดแทรก รอยขูดขีดบนแผ่นซีด เป็นต้น) การถอดรหัสจะประมวลผลแต่ละบล็อค และพยายามแก้ไขค่าผิดพลาด และกู้ข้อมูลเดิมจำนวน และชนิดของข้อผิดพลาดที่สามารถแก้ไขได้ ขึ้นอยู่กับคุณสมบัติของรหัสรีด – โซโลมอน

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1) เพื่อศึกษาการอัลกอริทึมการเข้ารหัส และถอดรหัสรีด – โซโลมอน
2) เพื่อศึกษาประสิทธิภาพการเข้ารหัส และถอดรหัสรีด – โซโลมอน
3) เพื่อออกแบบระบบการเข้ารหัสรีด – โซโลมอนบน FPGA
4) เพื่อออกแบบระบบการถอดรหัสรีด – โซโลมอนแบบปรับแต่งความเร็วได้บน FPGA

1.3 ขอบเขตการวิจัย

1) ศึกษา และพัฒนาระบบการเข้ารหัสรีด – โซโลมอน
2) ศึกษา และพัฒนาระบบถอดรหัสรีด – โซโลมอน
3) ระบบถอดรหัสรีด – โซโลมอน ปรับความเร็วได้ 2 ระดับ

1.4 ส่วนประกอบของรายงาน

ในบทที่ 1 กล่าวถึงความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ สมมติฐานของการศึกษา ขอบเขตของงานวิจัย และส่วนประกอบต่างๆของวิทยานิพนธ์ฉบับนี้

เนื้อหาบทที่ 2 กล่าวถึง ทฤษฎีที่เกี่ยวข้อง และจำเป็นกับการศึกษาวิจัย

เนื้อหาบทที่ 3 กล่าวถึงการพัฒนาอัลกอริทึมการถอดรหัสบนโดเมนเวลา

เนื้อหาบทที่ 4 กล่าวถึงการอิมพลิเมนต์ระบบการถอดรหัสบนโดเมนเวลา

เนื้อหาบทที่ 5 เป็นการสรุปงานวิจัยทั้งหมด

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

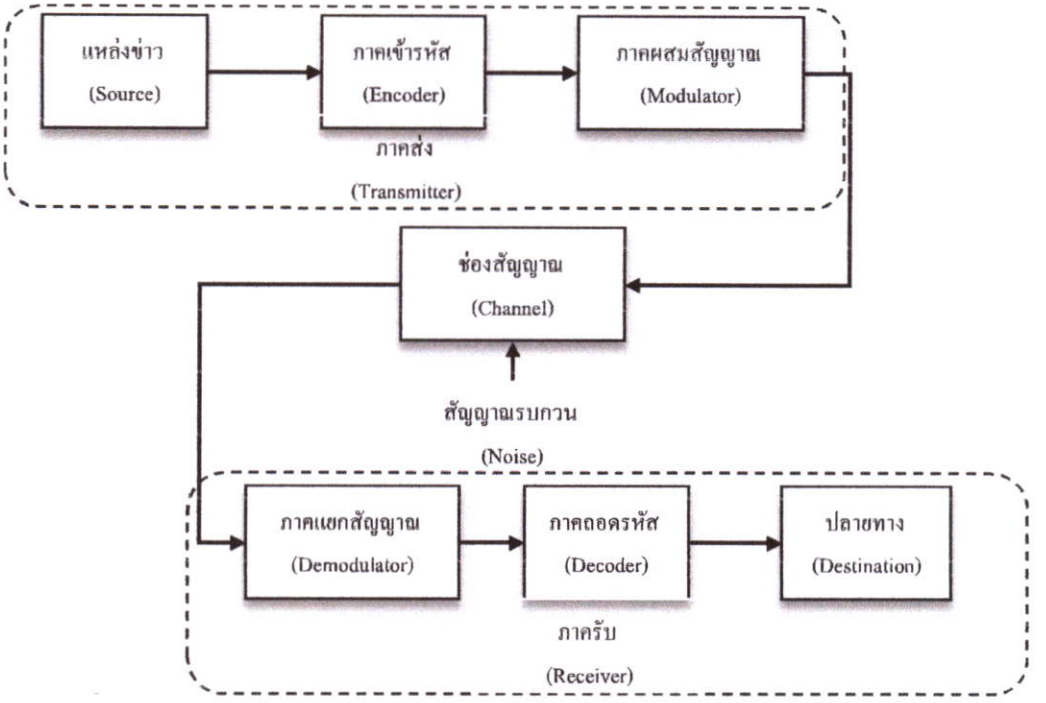
2.1 ที่มา และความสำคัญของการแก้ความผิดพลาดของข้อมูล

ระบบสื่อสารเป็นระบบสำหรับส่งข่าวสารโดยใช้สัญญาณทางไฟฟ้าเป็นสื่อ เพื่อส่งถ่ายข่าวสารระหว่างภาคส่ง และภาครับ รูปแบบของการส่งข่าวสารมีหลายรูปแบบ เช่น การส่งสัญญาณในรูปของคลื่นแม่เหล็กไฟฟ้าผ่านอากาศ การส่งสัญญาณเสียงผ่านสายโทรศัพท์ และการส่งข้อมูลผ่านสายโคแอค เนื่องจากการเดินทางของสัญญาณผ่านตัวกลางหรือที่เรียกว่าช่องสัญญาณ (Channel) สัญญาณข่าวสารมีโอกาสเกิดการเปลี่ยนแปลงองค์ประกอบทั้งทางขนาด (Amplitude) และมุม (Phase) ได้เพราะว่าสัญญาณมีโอกาสถูกรบกวนจากสัญญาณรบกวน (Noise) รวมถึงเกิดความผิดเพี้ยนอันเนื่องมาจากคุณสมบัติของตัวกลางและอุปกรณ์ที่ใช้งาน ความผิดเพี้ยนนี้ส่งผลให้ข่าวสารที่ถูกส่งไปที่ภาครับมีโอกาสเกิดความคลาดเคลื่อนไปจากข่าวสารต้นฉบับได้ การลดความผิดพลาดของข่าวสารสามารถทำได้หลายรูปแบบ เช่น การเพิ่มกำลังงาน (Power) สำหรับส่งสัญญาณเพื่อเพิ่มความแตกต่างระหว่างสัญญาณข่าวสาร และสัญญาณรบกวน การปรับปรุงอุปกรณ์ในระบบสื่อสารให้มีความทนทานต่อสัญญาณรบกวน และการใช้รหัสควบคุมความผิดพลาด (Error Control Coding: ECC) เป็นต้น

สำหรับรหัสควบคุมความผิดพลาดเป็นเทคโนโลยีลดความผิดพลาดของข่าวสารสำหรับระบบสื่อสารแบบดิจิตอลซึ่งได้รับการนำไปใช้ในระบบสื่อสารหลายประเภท เช่น ระบบโทรศัพท์เคลื่อนที่ ระบบสื่อสารผ่านดาวเทียม และระบบแพร่สัญญาณวีดีทัศน์ดิจิตอล นอกจากนั้นรหัสควบคุมความผิดพลาด ได้ถูกนำไปใช้ในระบบจัดเก็บข้อมูล เช่น ฮาร์ดดิสก์ และแผ่นคอมแพ็คดิสก์ เป็นต้น การนำรหัสควบคุมความผิดพลาดมาใช้งานสามารถลดความต้องการกำลังงานสำหรับส่งสัญญาณซึ่งเป็นประโยชน์อย่างยิ่งแก่ระบบสื่อสารบางประเภทที่จำเป็นต้องส่งข่าวสารในระยะทางไกลหรือมีข้อจำกัดเป็นพิเศษทางด้านกำลังงานของอุปกรณ์ส่งสัญญาณ การส่งข้อมูลในอวกาศที่มีระยะทางไกลเป็นตัวอย่างหนึ่งของระบบสื่อสารที่นำรหัสควบคุมความผิดพลาดมาใช้งาน การปรับปรุงระบบ สื่อสารประเภทนี้ให้มีกำลังส่งสัญญาณเพิ่มขึ้นแม้เพียงเล็กน้อย ก็จำเป็นต้องใช้เงินทุนมหาศาลนอกเหนือจากการใช้งาน ในระบบสื่อสารแล้ว รหัสสำหรับควบคุมความผิดพลาด ถูกใช้เพื่อปกป้องข่าวสาร ในคอมพิวเตอร์และอุปกรณ์บันทึกข้อมูล ด้วยที่คำรหัสที่กำเนิดขึ้นมาจากกระบวนการเข้ารหัสในแต่ละครั้ง มีปริมาณมากกว่าบิตข่าวสารที่ปราศจากการเข้ารหัสเสมอดังนั้นระบบสื่อสารที่นำรหัสควบคุมความผิดพลาดมาใช้งานจำเป็นต้องส่งข้อมูลผ่านตัวกลางด้วยอัตราเร็วที่เพิ่มขึ้นหรือต้องการแบนด์วิด (Bandwidth) หรือช่วงความถี่ (Frequency) ที่ใช้เพื่อการส่งสัญญาณเพิ่มขึ้น สำหรับระบบจัดเก็บข้อมูลการนำรหัสควบคุมความผิดพลาดมาใช้งานส่งผลให้ข้อมูลที่จำเป็นต้องบันทึกลงในอุปกรณ์มีขนาดเพิ่มขึ้น

พื้นฐานของรหัสควบคุมความผิดพลาดมีการทำงานดังรูปที่ 1 เป็นการนำบิตข่าวสารมาผ่านการเข้ารหัส เพื่อแปรสภาพให้อยู่ในรูปของคำรหัสก่อนการนำไปส่งผ่านระบบสื่อสาร หรือบันทึกลงสื่อตามปกติ คำรหัสเป็นข้อมูลที่มีคุณสมบัติพิเศษโดยสามารถนำมาใช้ตรวจจับ หรือแก้ไขความ

ผิดพลาดได้ เมื่อต้องการนำบิตข่าวสารกลับคืนมาก็จะนำคำรหัสที่รับได้มาผ่านการถอดรหัสเพื่อตรวจจับหรือแก้ไขความผิดพลาด



รูปที่ 2.1 แผนภาพบล็อกรหัสควบคุมความผิดพลาดในระบบสื่อสาร

พื้นฐานของรหัสควบคุมความผิดพลาด มีความเกี่ยวข้องกับทฤษฎีข่าวสารของแชนนอน (Shannon's Information theory) ซึ่งเป็นทฤษฎีเกี่ยวกับคณิตศาสตร์สำหรับระบบสื่อสารที่ได้รับการนำเสนอใน พ.ศ. 2491 โดย โคลด เอลวูด แชนนอน (Claude Elwood Shannon) [11] ภายหลังจากการนำ เสนอทฤษฎีข่าวสารมีการพัฒนารหัสสำหรับควบคุมความผิดพลาดในลักษณะต่าง ๆ เช่น รหัสแฮมมิ่ง (Hamming code) รหัสบีซีเอช (BCH Code) รหัสรีด – โซโลมอน (Reed Solomon: RS) เป็นต้น

2.2 สนามจำกัดกาลัวส์

เนื่องจากทฤษฎีการเข้า และถอดรหัสรีด – โซโลมอน เป็นการกระทำบนพื้นฐานของคณิตศาสตร์สนามกาลัวส์เป็นหลัก ดังนั้นในบทนี้จึงต้องอธิบายนิยามต่างๆ และวิธีการคำนวณเสียก่อน โดยสนามจำกัดกาลัวส์นั้น มีทฤษฎีที่จำเป็นต่อการเข้า และถอดรหัสรีด – โซโลมอน ดังนี้

2.2.1 วงล้อ

วงล้อ คือ ออปเจ็คทางคณิตศาสตร์ ที่สามารถจะบวก ลบ และคูณกันได้ มีคุณสมบัติดังนี้

1) เป็น Abelian Group ภายใต้เครื่องหมายบวก

2) Closure สำหรับ a และ b ที่อยู่ในเซ็ต R ค่าของ c= a*b ก็จะอยู่ในเซ็ตด้วย

3) Associative law สำหรับ a,b และ c ที่อยู่ในเซ็ต คือ
a(bc) = (ab)c

4) Distributive law
a(b + c) = ab + ac
(b + c) = ba + ca

2.2.2 สนาม

สนาม คือ ออปเจ็คท์ทางคณิตศาสตร์ที่สามารถบวก ลบ คูณ และหารกันได้ โดยสนาม F จะมี 2 การกระทำที่นิยามคือ การบวก และการคูณ และคุณสมบัติของสนามมีดังนี้

1) เซ็ตจะเกิดจาก Abelian Group ภายใต้เครื่องหมายการบวก

2) สนามจะเป็นสนามปิดภายใต้การคูณ และเซ็ตของอิลิเมนต์ที่ไม่เป็นศูนย์จะเป็น Abelian Group ภายใต้การคูณ

3) ใช้กฎการกระจายได้สำหรับ a,b และ c ที่อยู่ในสนาม
$a(b + c) = ab + ac$

4) สนามอาจมีจำนวนมากมายเช่น สนามจำนวนจริง (R), สนามจำนวนเชิงซ้อน (C) หรือสนามของจำนวนสัดส่วน (Q) เป็นต้น

2.2.3 วงล้อพหุนาม

สำหรับจำนวนเต็มบวก q พหุนามบนสนาม GF(q) สามารถเขียนได้โดยสมการคณิตศาสตร์คือ

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

โดยที่สัมประสิทธิ์ $f_{n-1}, f_{n-2}, \ldots, f_2, f_1, f_0$ เป็นสมาชิกใน GF(q)

1) Monic polynomial ก็คือพหุนามที่ $f_{n-1} = 1$ และพหุนามสองพหุนามจะเท่ากันเมื่อสัมประสิทธิ์ f_i ของแต่ละพหุนามมีค่าเท่ากันสำหรับทุกๆ ค่าของ i

2) ดีกรีของพหุนาม ก็คือ $n - 1$

3) เซ็ตของพหุนามใน GF(q) สามารถที่จะต่อกันเป็นวงล้อได้หาก การคูณและการบวกนั้น ก็นิยามเช่นเดียวกับพหุนามปกติโดยใช้ GF(q)[x] แทนวงล้อของพหุนาม

4) การบวกกันของสองพหุนาม f(x) และ g(x) ก็จะทำให้ได้พหุนามใหม่ ซึ่งทำได้คือ $f(x) + g(x) = \sum_{i=0}^{n-1}(f_i + g_i)x^i$ โดยที่ดีกรีสูงสุดของผลลัพท์จะเท่ากับดีกรีของพหุนามเดิม

5) การคูณกันของสองพหุนาม $f(x)$ และ $g(x)$ ก็จะทำให้ได้พหุนามใหม่ ซึ่งทำได้โดย $f(x)g(x) = \sum_{i=0}^{i=\infty}(\sum_{j=0}^{i} f_i g_i)\, x^i$ โดยที่ดีกรีสูงสุดของผลลัพท์จะเท่ากับดีกรีของสองพหุนามเดิมบวกกัน

2.2.4 สนามจำกัดบนพื้นฐานของวงล้อพหุนาม

ทำนองเดียวกันกับสนามจำนวนเต็ม สนามจำกัดของพหุนาม ก็สามารถที่จะสร้างจากวงล้อพหุนาม โดยหาจากผลหารเต็ม และพหุนามปฐม ซึ่งสนามจำกัดนี้ โดยทั่วไปจะเรียกว่า สนามจำกัดกาลัวส์ (Galois Field: GF) ซึ่งมีคุณสมบัติต่างๆดังนี้

1) ในสนามจำกัดกาลัวส์ใดๆ จำนวนสมาชิกทั้งหมดจะเป็นกำลังของจำนวนปฐม (เช่น p^m)
2) หากว่า p เป็นจำนวนปฐม และ m เป็นจำนวนเต็มบวก สนามย่อยที่สุดของ $GF(p^m)$ ก็คือ $GF(p)$ ซึ่งสมาชิกของ $GF(p)$ จะเรียกว่าเป็นจำนวนเต็มของ $GF(p^m)$ และจะเรียก p ว่าค่าคาแร็กเตอริสติก
3) สนามจำกัดกาลัวส์ซึ่งมีคาแร็กเตอริสติก 2 ค่า $\beta = -\beta$ สำหรับทุกๆค่าของ β ในสนามนั้น
4) หากว่า p เป็นจำนวนปฐม และ m เป็นจำนวนเต็มบวก ก็จะปรากฏมีสนามจำกัดกาลัวส์ซึ่งมีสมาชิกทั้งหมด p^m สมาชิก
5) สนามจำกัด $GF(q^m)$ จะมีสนามย่อย $GF(q)$ หรือในทางกลับกันเรียก $GF(q^m)$ ว่าเป็นสนามขยายของสนามย่อย $GF(q)$
6) สนามของจำนวนเชิงซ้อนย่อยของสนามจำกัด $GF(p^m)$ หรือ $GF(q^m)$ เป็นสนามขยายของสนามย่อย $GF(q)$
7) สนามจำกัด $GF(q^n)$ จะไม่ใช่สนามย่อยของ $GF(q^m)$ หาก n ไม่สามารถหาร m ได้ลงตัว

2.2.5 อิเลเมนต์ปฐมภูมิ

อิเลเมนต์ปฐมภูมิ (Primitive Element) ของสนามจำกัด $GF(q)$ คือ α ซึ่งอิเลเมนต์อื่นๆ ทั้งหมดสามารถแสดงได้ในรูปของกำลังของ α เช่น ใน $GF(5)$ จะมี 2 เป็นอิเลเมนต์ปฐมภูมิ และจะเห็นได้ว่า อิเลเมนต์อื่นจะสามารถสร้างจาก 2 ได้ทั้งหมด เช่น $2^1 = 2, 2^2 = 4, 2^3 = 3, 2^4 = 1$ และในที่นี้ก็คือ α=2

ในทุกๆ สนามจำกัด จะต้องมีอิเลเมนต์ปฐมภูมิเสมอ เช่นใน หรือ อิเลเมนต์ต่างๆ จะมีออร์เดอร์ที่หาร 15 ลงตัว คือ 1 3 5 และ 15 ซึ่งอิเลเมนต์ที่มีออร์เดอร์ 15 ก็จะเป็นอิเลเมนต์ปฐมภูมิ

เช่นเราสามารถที่จะสร้างอิเลมเนต์ใน $GF(2^4)$ ได้โดยใช้ พหุนามปฐม $p(x) = x^4 + x + 1$ แสดงโดยตารางที่ 1 ซึ่งการแทนอิเลเมนต์นั้นสามารถที่จะเขียนแทนได้หลายรูปแบบ

ตารางที่ 2.1 อิเลเมนต์ต่างๆใน $GF(2^4)$ โดยใช้พหุนามปฐม $p(x) = x^4 + x + 1$

i	$\beta_i = \alpha^i$	Polynomial	Binary	Decimal	Hexadecimal	Minimal Polynomial
0	α^0	1	0001	1	1	$x + 1$
1	α^1	x	0010	2	2	$x^4 + x + 1$
2	α^2	x^2	0100	4	4	$x^4 + x + 1$
3	α^3	x^3	1000	8	8	$x^4 + x^3 + x^2 + x + 1$
4	α^4	$x + 1$	0011	3	3	$x^4 + x + 1$
5	α^5	$x^2 + x$	0110	6	6	$x^2 + x + 1$
6	α^6	$x^3 + x^2$	1100	12	C	$x^4 + x^3 + x^2 + x + 1$
7	α^7	$x^3 + x + 1$	1011	11	B	$x^4 + x^3 + 1$
8	α^8	$x^2 + 1$	0101	5	5	$x^4 + x + 1$
9	α^9	$x^3 + x$	1010	10	A	$x^4 + x^3 + x^2 + x + 1$
10	α^{10}	$x^2 + x + 1$	0111	7	7	$x^2 + x + 1$
11	α^{11}	$x^3 + x^2 + 1$	1110	14	E	$x^4 + x^3 + 1$
12	α^{12}	$x^3 + x^2 + x + 1$	1111	15	F	$x^4 + x^3 + x^2 + x + 1$
13	α^{13}	$x^3 + x^2 + 1$	1101	13	D	$x^4 + x^3 + 1$
14	α^{14}	$x^3 + x$	1001	9	9	$x^4 + x^3 + 1$

แต่ละอิเลเมนต์จะมีพหุนามต่ำสุด (minimal polynomial, m(x)) เสมอ เช่น α^5 เป็นรากของ
$x^2 + x + 1$ เพราะ $\alpha^{10} + \alpha^5 + 1 = 7 + 6 + 1 = 0$ หรือจะกล่าวได้ว่า α^i เป็นรากของ $m_i(x)$

2.2.6 การดำเนินการของอิเลเมนต์ในสนามจำกัด

1) การกระทำที่สำคัญ คือ การบวก การลบ การคูณ การหาร และการหาส่วนกลับ

2) ในสนามจำกัดกาลัวส์ การบวก และการลบเป็นการกระทำเดียวกัน คือ เอ็กซ์คลูซีฟออร์ (XOR)

3) หาก a และ b เป็นอิเลเมนต์ในสนามจำกัดเดียวกันแล้ว โดยที่ b เป็นส่วนกลับของ a หรือ $b=a^{-1}$ ก็จะได้ $ab=1$ ดังนั้น การหารก็คือการคูณด้วยส่วนกลับ เช่น $\frac{a}{c} = ac^{-1}$

ตัวอย่าง แสดงการหาค่า 10+7; 10x7; 7/10 และ 10/7

1) 10+7

เนื่องจาก $(10)_{10} = (1010)_2$ และ $(7)_{10} = (0111)_2$ ดังนั้น 1010⊕0111=1101 หรือ 10+7=13

2) 10x7

เนื่องจาก 10 = α^9 และ 10 = ดังนั้น 10x7=$\alpha^9 \times \alpha^{10} = \alpha^{15+4} = \alpha^4 = 3$ หรือ 10x7=3 หรือ 10 -> $x^3 + x$, 7 -> $x^2 + x + 1$ ดังนั้น

$$10x7 = (x^3 + x) \times (x^2 + x + 1) \mathrm{mod}(x^4 + x + 1)$$

3) 7/10 และ 10/7

เนื่องจาก $7 = \alpha^9$ และ $10 = \alpha^{10}$ ดังนั้น $\frac{7}{10} = \alpha^9\alpha^{-10} = 2$ หรือ $\frac{10}{7} = \alpha^{10}\alpha^{-9} = 9$

จะเห็นว่า การบวก และการคูณเป็นสิ่งสำคัญ และเพื่อความสะดวก จึงมักมีการสร้างตารางเอาไว้ ดังตารางที่ 2 และ ตารางที่ 3

ตารางที่ 2.2 ตารางการบวกใน GF(16)

α^i			α^0	α^1	α^4	α^2	α^8	α^5	α^{10}	α^3	α^{14}	α^9	α^7	α^6	α^{13}	α^{11}	α^{12}
	+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
α^0	**1**	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
α^1	**2**	2	3	0	1	6	7	4	5	10	7	8	9	14	15	12	13
α^4	**3**	3	2	1	0	7	6	5	4	11	6	13	8	15	14	13	12
α^2	**4**	4	5	6	7	0	1	2	3	12	5	14	15	8	9	10	11
α^8	**5**	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
α^5	**6**	6	7	4	5	2	3	0	1	14	15	12	13	10	9	8	11
α^{10}	**7**	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
α^3	**8**	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
α^{14}	**9**	9	8	7	6	5	12	15	14	1	0	3	2	5	4	7	6
α^9	**10**	10	11	8	13	14	15	12	13	2	3	0	2	6	7	4	5
α^7	**11**	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
α^6	**12**	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3

α^{13}	**13**	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
α^{11}	**14**	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
α^{12}	**15**	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ตารางที่ 2.3 ตารางการคูณใน GF(16) ซึ่ง $p(x) = x^4 + x + 1$

α^i			α^0	α^1	α^4	α^2	α^8	α^5	α^{10}	α^3	α^{14}	α^9	α^7	α^6	α^{13}	α^{11}	α^{12}
	+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
α^0	**1**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
α^1	**2**	0	2	4	6	8	10	12	14	3	1	7	5	11	9	15	13
α^4	**3**	0	3	6	9	12	15	10	9	11	8	13	14	7	4	1	2
α^2	**4**	0	4	8	12	3	7	11	15	6	2	14	10	5	1	13	9
α^8	**5**	0	5	10	15	7	2	13	8	14	11	4	1	9	12	3	6
α^5	**6**	0	6	12	10	11	13	7	1	5	3	9	15	14	8	2	4
α^{10}	**7**	0	7	14	5	15	8	1	6	13	10	3	4	2	5	12	11
α^3	**8**	0	8	3	11	6	14	5	13	12	4	15	7	10	8	9	1
α^{14}	**9**	0	9	1	8	2	11	3	10	4	13	5	12	6	15	7	14
α^9	**10**	0	10	7	13	14	4	9	3	15	5	12	2	1	11	6	12
α^7	**11**	0	11	5	14	10	1	15	4	7	12	2	9	13	6	8	3
α^6	**12**	0	12	11	7	5	9	14	2	10	6	1	13	15	3	4	8
α^{13}	**13**	0	13	9	4	1	12	8	5	2	15	11	6	3	14	10	7
α^{11}	**14**	0	14	15	1	13	3	2	12	9	7	6	8	4	10	11	5
α^{12}	**15**	0	15	13	2	9	6	4	11	1	14	12	3	8	7	5	10

2.3 รหัสบล็อก และรหัสคอนโวลูชัน

2.3.1 รหัสบล็อก (Block Code)

เป็นการเข้า และถอดรหัสข้อมูลแบบทีละบล็อก บล็อกละหลายๆ บิต หรือหลายๆ สัญลักษณ์ ตามข้อกำหนดของแต่ละรหัส โดยอาศัยความรู้ด้านสนามจำกัด (Finite Field) และพีชคณิตนามธรรม (Abstract Algebra) ในการเข้า และถอดรหัส เช่น รหัสบล็อกเชิงเส้น (Linear Block Code) รหัสวน (Cyclic Code) รหัสรีด – โซโลมอน (RS Codes) รหัสบีซีเอช (Bose-Chaudhuri-Hocquenghen Code: BCH Code) รหัสโกเลย์ (Golay Code) และรหัสแฮมมิง (Hamming Code) เป็นต้น

2.3.2 รหัสคอนโวลูชัน (Convolution Code)

เป็นการเข้า และถอดรหัสทีละบิตต่อเนื่องกันไปเรื่อย ๆ ในรูปของกระแสบิต (Bit Stream) จึงทำให้สามารถทำงานแบบเวลาจริง (Real – Time) ได้ ซึ่งรหัสคอนโวลูชันนี้ นิยมใช้ในหลายงานประยุกต์ เช่น วิดีทัศน์แบบดิจิตอล การสื่สารแบบไร้สาย และการสื่อสารดาวเทียม เป็นต้น

2.4 รหัสบีซีเอช

รหัสบีซีเอช (Bose-Chaudhuri-Hocquenghen Code: BCH Code) ได้มีการใช้งานอย่างแพร่หลายในเรื่องของการสื่อสารโดยรหัสนี้ยังคงเป็นรหัสวนรอบ (หมายถึงการเลื่อนวนไปทางขวาของอิลิเมนต์ในคำรหัสยังคงได้คำรหัสเช่นกัน) ข้อมูลของรหัสบีซีเอชจะอยู่ในสนามพื้นฐาน (Ground Field) แต่ตำแหน่งความผิดพลาดนั้นจะอยู่ในสนามขยาย (Extension Field) รหัสรีด – โซโลมอนก็จัดอยู่ในรหัสบีซีเอชเช่นกัน

2.4.1 รหัสวนรอบ (Cyclic Code)

กำหนดให้ $c(x)$, $d(x)$ และ $g(x)$ เป็นพหุนามแทนรหัส (Codeword) ข้อมูล (Data) และพหุนามกำเนิด (Generator Polynomial) ตามลำดับดังนี้

$$c(x) = \sum_{i=0}^{n-1} c_i x^i = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \tag{2.1}$$

$$d(x) = \sum_{i=0}^{n-1} d_i x^i = d_0 + d_1 x + \cdots + d_{n-1} x^{n-1} \tag{2.2}$$

$$g(x) = \sum_{i=0}^{n-1} d_i x^i = d_0 + d_1 x + \cdots + d_{n-1} x^{n-1} \tag{2.3}$$

ในรูปที่ไม่ใช่เชิงระบบ (Non-Systematic Form) พหุนาม $c(x)$ จะเขียนได้ ดังนี้

$$\begin{aligned} c(x) &= g(x)d(x) \\ &= c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \end{aligned} \tag{2.4}$$

ในรูปที่เป็นเชิงระบบ (Systematic Form) พหุนาม $c(x)$ จะเขียนได้เป็น

$$c(x) = r_0 + r_1x + \cdots + r_{n-k-1}x^{n-k-1} + d_0x^{n-k} + d_1x^{n-k+1} + \cdots + c_{k-1}x^{n-1}$$

พหุนามที่ต่อท้าย $r(x)$ หาได้จากการหาเศษเหลือของ $\{x^{n-k}d(x)\}\mathrm{mod}g(x)$ หรือก็คือ $x^{n-k}d(x) = Q(x)g(x) + r(x)$ สำหรับพหุนาม $Q(x)$ ใด ๆ หรือในรูปของพหุนาม $c(x)$ จะเขียนได้เป็น

$$\mathbb{C} = \{r_0, r_1, r_2, \ldots, r_{n-k-1}, d_0, d_1, \ldots, d_{k-1}\} \quad (2.5)$$

2.4.2 รหัสบีชีเอช (BCH Code)

รหัสบีชีเอช ได้ถูกเสนอครั้งแรกในปี ค.ศ. 1959 โดย Hocquenghem และเพิ่มเติมโดย Bose และ Chausdhuri ในปี ค.ศ. 1960

นิยาม: พหุนามกำเนิด (Generator Polynomial) ของรหัสบีชีเอชความยาวบล็อก $n = 2^m - 1$ ที่สามารถแก้ไขจำนวนบิตผิดพลาด t บิต $f_i(x)$ คือผลคูณของพหุนามต่ำสุด (Minimal Polynomial, *m(x)*) ที่ต่างกัน $f_i(x); 1 \le i \le 2t$ ของ $\alpha, \alpha^2, \ldots, \alpha^{2t}$ เมื่อ $\alpha \in GF(2^m)$ เป็นรากของพหุนามปฐม *p(x)* ดังนั้น

$$g(x) = LCM\{f_1(x), f_2(x), \ldots, f_{2t}(x)\} \quad (2.6)$$

แต่เนื่องจาก α ที่มีกำลังเป็นจำนวนเท่าของกันจะเป็นรากของพหุนามต่ำสุดเดียวกันเสมอ (เช่น $x^4 + x + 1$ เป็นพหุนามต่ำสุดของ $\alpha, \alpha^2, \alpha^4, \alpha^8$) สมการของพหุนามกำเนิดจึงเขียนได้เป็น

$$g(x) = LCM\{f_1(x), f_3(x), \ldots, f_{2t-1}(x)\} \quad (2.7)$$

จะเห็นได้ว่าโดยวิธีการนี้เมื่อพหุนามกำเนิด *g(x)* สร้างจากการคูณกันของพหุนามต่ำคำรหัส $c(x) = g(x)d(x)$ ก็จะมี $\alpha, \alpha^2, \ldots, \alpha^{2t}$ เป็นรากด้วย และเนื่องจากดีกรีของ *f(x)* นั้นมีค่าสูงสุดคือ *m* ดังนั้น กำลังสูงสุดของพหุนามกำเนิดจะเป็น *mt*

2.4.3 การเข้ารหัสบีชีเอช

ในการเข้ารหัสนั้น จำเป็นจะต้องรู้พหุนามกำเนิด จากนั้นจึงตัดสินใจที่จะเลือกว่าจะเข้ารหัสเชิงระบบหรือไม่เป็นเชิงระบบก็ได้

1) กำหนดสนาม หรือความยาวบล็อก
2) กำหนดจำนวนบิตผิดพลาดที่รหัสบีชีเอชจะยังคงแก้ไขได้ (หาค่า *t*)
3) หาพหุนามกำเนิด $g(x) = LCM\{f_1(x), f_2(x), \ldots, f_{2t}(x)\}$
4) จากดีกรีของ *g(x)* คือ *n-k* จะทำให้ได้รหัส *BCH(n,k,t)*
5) สำหรับการเข้ารหัสที่ไม่เป็นเชิงระบบ *c(x) = g(x)d(x)*
6) สำหรับการเข้ารหัสที่เป็นเชิงระบบ $\{x^{n-k}d(x)\}\mathrm{mod}g(x)$ และ
$$c(x) = r_0 + r_1x + \cdots + r_{n-k-1}x^{n-k-1} + d_0x^{n-k} + d_1x^{n-k+1} + \cdots + c_{k-1}x^{n-1}$$

2.4.4 การถอดรหัสบีซีเอช

2.4.4.1 ซินโดรม

สมมติว่าคำรหัส $c(x)$ ถูกส่งออกไปผ่านช่องทางสัญญาณที่มีการรบกวนทำให้ข้อมูลที่ส่งไปเกิดความผิดพลาด เมื่อให้คำรหัสที่ได้รับคือ $v(x)$ จะได้ว่า

$$v(x) = c(x) + e(x) \tag{2.8}$$

เมื่อ e(x) คือพหุนามความผิดพลาด หรือรูปแบบความผิดพลาด

$$e(x) = \sum_{i=0}^{n-1} e_i x^i = e_0 + e_1 x + e_2 x^2 + \cdots + e_{n-1} x^{n-1} \tag{2.9}$$

ความผิดพลาดที่เกิดขึ้นนั้นจะถูกแก้ไขได้หากพหุนามนี้มีจำนวนเทอมซึ่งสัมประสิทธิ์ไม่เป็นศูนย์ (คือมีค่าเป็น 1) ไม่มากกว่า t สมมติว่ามีจำนวนบิตผิดพลาดทั้งหมด v ซึ่ง $1 \leq v \leq t$ เกิดขึ้นที่ตำแหน่ง i ใดๆ ซึ่ง $0 \leq i \leq n-1$ พหุนามความผิดพลาดอาจเขียนใหม่ได้เป็น

$$\begin{aligned} e(x) &= \sum_{i=0}^{n-1} e_{i_\lambda} x^{i_\lambda} \\ &= e_{i_1} x^{i_1} + e_{i_2} x^{i_2} + \cdots + e_{i_v} x^{i_v} \end{aligned} \tag{2.10}$$

โดยที่ e_{i_λ} เป็นขนาดของความผิดพลาด $1 \leq \lambda \leq v$ และเนื่องจากคำรหัสกำหนดให้ $\alpha, \alpha^2, \ldots, \alpha^{2t}$ เป็นรากของพหุนามกำเนิด ดังนั้น $c(\alpha^k) = 0$ สำหรับ $1 \leq k \leq 2t$ การประเมินพหุนามที่ค่ารากต่างๆจะได้ผลลัพท์ที่เรียกว่าซินโดรม (Syndrome)

$$S_k = v(\alpha^k) = c(\alpha^k) + e(\alpha^k) = \sum_{i=0}^{n-1} v_i \alpha^{ik} \tag{2.11}$$

สำหรับราก α^1 จึงเขียนได้เป็น

$$S_1 = v(\alpha) = c(\alpha) + e(\alpha) = e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \cdots + e_{i_v}\alpha^{i_v} \tag{2.12}$$

เพื่อความกระชับหากให้ $Y_i = e_{i_1}$ คือขนาดของความผิดพลาด และ$X_i = \alpha^{i_\lambda}$ คือตำแหน่งของความผิดพลาดสมการนี้จึงเขียนใหม่ได้เป็น

$$S_1 = Y_1 X_1 + Y_2 X_2 + \cdots + Y_v X_v \tag{2.13}$$

เนื่องจากสมการมีรากทั้งหมดคือ $\alpha, \alpha^2, \ldots, \alpha^{2t}$ ดังนั้นจึงได้

$$\begin{aligned} S_1 &= Y_1 X_1 + Y_2 X_2 + \cdots + Y_v X_v \\ S_2 &= Y_1 X_1^2 + Y_2 X_2^2 + \cdots + Y_v X_v^2 \\ &\vdots \\ S_{2t} &= Y_1 X_1^{2t} + Y_2 X_2^{2t} + \cdots + Y_v X_v^{2t} \end{aligned} \quad \text{หรือ} \quad \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & \cdots & X_v \\ X_1^2 & X_2^2 & \cdots & X_v^2 \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{2t} & X_2^{2t} & \cdots & X_v^{2t} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_v \end{bmatrix} \tag{2.14}$$

สมการเหล่านี้จะเรียกว่า power sum symmetric function

เนื่องจากรหัสบีซีเอชมีขนาดของข้อมูลเป็นไบนารี (หรือ $d_j \in (0,1)$ สำหรับ $0 \leq j \leq k-1$) การถอดรหัสจึงมีความจำเป็นเพียงเพื่อหาตำแหน่งของความผิดพลาดเท่านั้น หน้าที่ของเราคือการแก้สมการที่เป็นสมการไม่เป็นเชิงเส้นเหล่านี้

2.4.4.2 สมการซินโดรม

เพื่อหลีกเลี่ยงความยุ่งยากในการแก้สมการที่เป็นสมการไม่เป็นเชิงเส้นเหล่านั้นโดยตรง เราจะนิยามพหุนามตำแหน่งความผิดพลาด (Error Locator Polynomial) ดังนี้

□□=1+□1□+□2□2+...+□□□□ (2.15)

ซึ่งพหุนามนี้จะมีรากเป็นส่วนกลับของตำแหน่งความผิดพลาด (X_λ^{-1} สำหรับ $1 \leq \lambda \leq v$) ดังนั้น

$$\Lambda(x) = (1 - xX_1)(1 - xX_2) \dots (1 - xX_v) \tag{2.16}$$

สมการไม่เป็นเชิงเส้นเหล่านั้นสามารถที่จะจัดการให้อยู่ในรูปแบบที่เป็นสมการเชิงเส้นได้ โดยสามารถเขียนในรูปของเมตริกซ์ ดังนี้

$$\begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_v \\ S_2 & S_3 & S_4 & \cdots & S_{v+1} \\ S_3 & S_4 & S_5 & \cdots & S_{v+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_v & S_{v+1} & S_{v+2} & \cdots & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \Lambda_{v-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ -S_{v+3} \\ \vdots \\ -S_{2v} \end{bmatrix} \tag{2.17}$$

สมการนี้สามารถที่จะแก้ได้โดยการหาค่าส่วนกลับของเมตริกซ์ และเรียกว่าสมการซินโดรม (Syndrome Equations) มีหลายวิธีที่จะแก้สมการซินโดรม เช่น วิธีการหาส่วนกลับเมตริกซ์ (Peterson-Gorestein-Zeirler), วิธีการสังเคราะห์รีจิสเตอร์ (Berlekamp-Massey), วิธีการยูคลิดส์ (ใช้การหา ค.ร.น.) เป็นต้น

2.4.4.3 วิธีการสังเคราะห์รีจิสเตอร์ (Peterson Gorenstein-Zeirler Algorithm)

สามารถสรุปได้ดังนี้

1) หาค่าซินโดรมจาก $S_k = v(\alpha^{j_0+k+1}); k = 1,2, \dots, 2t$

2) กำหนดเมตริกซ์ซินโดรม

$$M = \begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_v \\ S_2 & S_3 & S_4 & \cdots & S_{v+1} \\ S_3 & S_4 & S_5 & \cdots & S_{v+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_v & S_{v+1} & S_{v+2} & \cdots & S_{2v-1} \end{bmatrix}$$

3) เริ่มต้นที่ $v=t$

4) ตรวจสอบดีเทอร์มิแนนท์ของเมตริกซ์ หากดีเทอร์มิแนนท์เป็นศูนย์ ก็ให้ลดขนาดเมตริกซ์ลงแล้วหาดีเทอร์มิแนนท์ใหม่

5) แก้สมการ

$$\begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \Lambda_{v-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \frac{1}{M} \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ -S_{v+3} \\ \vdots \\ -S_{2v} \end{bmatrix}$$

6) หาค่ารากของสมการ $\Lambda(x) = 1 + \Lambda_1 x + \Lambda_2 x^2 + \cdots + \Lambda_v x^v$

7) ส่วนกลับของรากคือตำแหน่งที่ข้อมูลผิดพลาด กลับบิตของข้อมูลที่ตำแหน่งนั้น หรือคำนวณ

8) ขนาดของความผิดพลาดจาก

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_v \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & \cdots & X_v \\ X_1^2 & X_2^2 & \cdots & X_v^2 \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{2t} & X_2^{2t} & \cdots & X_v^{2t} \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix}$$

วิธีการ Peterson-Gorenstein-Zeirler เป็นวิธีที่ง่ายสำหรับการทำความเข้าใจเรื่องการถอดรหัสบีซีเอช แต่จะเห็นว่าวิธีการนี้มีการคำนวณการหาส่วนกลับขนาด $t \times t$ ถึง 2 ครั้ง แม้ว่าการคำนวณในสนามจำกัดจะไม่มีความผิดพลาดในเรื่องของการปัดเศษ แต่การคำนวณการหาส่วนกลับของเมตริกซ์เมื่อ t มีขนาดใหญ่ขึ้นจะเป็นการเสียเวลามาก การหาค่าส่วนกลับของเมตริกซ์ครั้งแรก เพื่อหาพหุนามระบุตำแหน่งนั้น สามารถที่จะแทนได้ด้วยวิธีการสังเคราะห์รีจิสเตอร์ด้วยวิธีการของเบอร์ลิแคมป์ – แมสซีย์ (Berlekamp-Massey Algorithm) ส่วนการหาส่วนกลับเมตริกซ์ครั้งที่สองเพื่อหาขนาดของความผิดพลาดนั้นจะแทนด้วยวิธีการของฟอร์เนย์ (Forney's Procedure) ดังจะได้กล่าวในรายละเอียดในเรื่องการถอดรหัสรีด – โซโลมอน

2.4.4.4 วิธีการประเมินพหุนามโดยวิธีการฮอร์เนอร์ (Horner's Scheme)

พิจารณาพหุนาม

$$A(x) = \sum_{i=0}^{n-1} a_i x^i = \mathrm{a}_0 + \mathrm{a}_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \qquad (2.18)$$

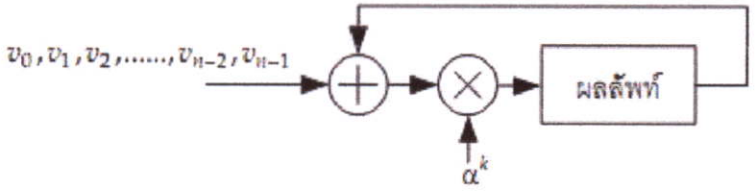
จะเขียนใหม่ในรูปเอกนามได้เป็น

$$A(x) = \Big(\big((\ldots((\mathrm{a}_{n-1}x) + \mathrm{a}_{n-2})x + \cdots + a_3\big)x + a_2\Big)x + a_1\Big)x + a_0 \qquad (2.19)$$

เช่น $A(x) = 7x^3 + 5x^2 - 4x + 2 = \big((7x + 5)x - 4\big)x + 2$

หรือ $A(x) = v_{n-1}x^{n-1} + v_{n-2}x^{n-2} + \cdots + v_2x^2 + v_1x + v_0 = \Big(\big((\ldots((\mathrm{v}_{n-1}x) + \mathrm{v}_{n-2})x + \cdots + v_3\big)x + v_2\Big)x + v_1\Big)x + v_0$ หากต้องการประเมนพหุนามนี้ที่ $x = \alpha^k$ ก็จะได้เป็น

$A(x) = \Big(\big((\ldots((\mathrm{v}_{n-1}\alpha^k) + \mathrm{v}_{n-2})\alpha^k + \cdots + v_3\big)\alpha^k + v_2\Big)\alpha^k + v_1\Big)\alpha^k + v_0$ ซึ่งง่ายต่อการคำนวณในรูปแบบของการวนลูป ดังรูปที่ 2



รูปที่ 2.2 การประเมินพหุนามโดยวิธีการฮอร์เนอร์

จะเห็นได้ว่าค่าซินโดรมก็คือ $S_k = v(\alpha^k) = \sum_{i=0}^{n-1} v_i\alpha^{ik}$ เป็นการประเมินคำรหัสที่ได้รับมาที่รากของพหุนามต่ำสุดในแต่ละการประเมินจะใช้ n รอบการคำนวณ (n ครั้งการคูณ และ n ครั้งการบวก) เนื่องจากซินโดรมมีทั้งหมด 2t ตัว จะต้องเสียรอบการทำงานถึง 2nt อย่างไรก็ตามสามารถปรับปรุงให้สั้นลงได้โดยการขนานส่วนประเมินพหุนาม 2t ชุดเข้าด้วยกัน

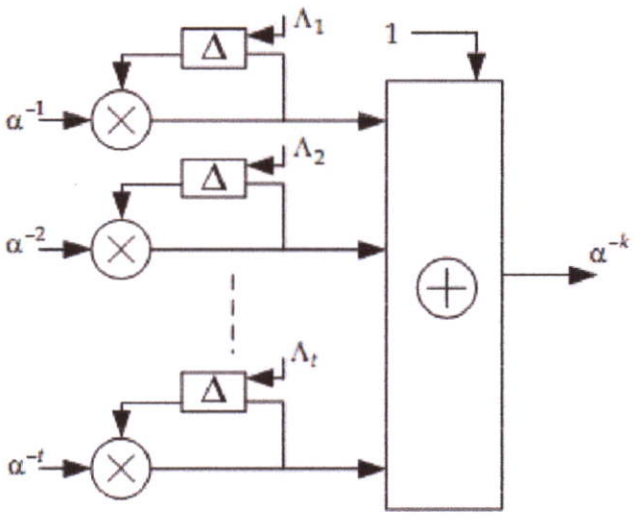
2.4.4.5 การหารากด้วยวิธีการเฉียน (Chien's Search)

พหุนามระบุตำแหน่ง (Error Locator Polynomial) จากสมการที่ (2.15) สามารถจะเปลี่ยนรูปโดยใช้กฎของฮอร์เนอร์ได้เช่นกัน ซึ่งจะได้ดังนี้

$$\Lambda(x) = \Big(\big((\ldots((\mathrm{v}_{n-1}\alpha^k) + \mathrm{v}_{n-2})\alpha^k + \cdots + v_3\big)\alpha^k + v_2\Big)\alpha^k + v_1\Big)\alpha^k + v_0 \qquad (2.20)$$

ซึ่งจะต้องทำการประมเนที่ทุกค่าอิเลเมนต์ α^k เพื่อหาค่า ที่ทำให้ $\Lambda(\alpha^k) = 0$ การกระทำเช่นนี้จะต้องเสียเวลาทั้งหมดในการวนรอบไป nt ครั้ง

เพื่อปรับปรุงให้การคำนวณเร็วขึ้นโดยเสียเวลาวนรอบเพียงประมาณ n ครั้ง และเมื่อ k เป็นตำแหน่งข้อมูลที่เกิดจากการผิดพลาด การประเมินจะกระทำได้ที่ $x = \alpha^{-k}$



รูปที่ 2.3 การออกแบบฮาร์ดแวร์วิธีการเฉียน

จากรูปที่ 3 รีจิสเตอร์จะต้องพรีโหลดด้วย $\Lambda_1, \Lambda_2, \Lambda_3, \ldots, \Lambda_t$ ผลการประเมินจะได้มาในทุกๆรอบ และต้องประเมินทั้งหมด n-1 รอบ โดย

$$
\begin{aligned}
\mathrm{k}=0, \quad & \Lambda(0) && = 1+\Lambda_1+\Lambda_2+\cdots+\Lambda_t \\
\mathrm{k}=1, \quad & \Lambda(\alpha^{-1}) && = 1+\Lambda_1\alpha^{-1}+\Lambda_2\alpha^{-2}+\cdots+\Lambda_t\alpha^{-t} \\
\mathrm{k}=2, \quad & \Lambda(\alpha^{-2}) && = 1+\Lambda_1\alpha^{-2}+\Lambda_2\alpha^{-4}+\cdots+\Lambda_t\alpha^{-2t} \\
& \vdots && \vdots \\
\mathrm{k}=\mathrm{n}-1, \quad & \Lambda\left(\alpha^{-(n-1)}\right) && = 1+\Lambda_1\alpha^{-(n-1)}+\Lambda_2\alpha^{-2(n-1)}+\cdots+\Lambda_t\alpha^{-t(n-1)}
\end{aligned}
$$

2.5 การแทนค่ารหัสรีด – โซโลมอน ด้วยพหุนาม

2.5.1 รหัสวนรอบ

รหัสรีด – โซโลมอน จัดอยู่ในกลุ่มรหัสแบบวนรอบไม่เป็นไบนารี่ (Non-Binary Cyclic Code) ดังนั้น การแทนค่าสัญลักษณ์ในรหัสรีด-โซโลมอน สามารถอธิบายแบบรหัสวนรอบได้

รหัสวนรอบนั้น เป็นรหัสบล็อกเชิงเส้นชนิดหนึ่ง ซึ่งง่ายต่อการอิมพลิเมนต์ วงจรตรรกซีเควนเชียล หรือรีจิสเตอร์เลื่อน

กำหนดให้ C เป็นคำรหัสเมื่อ $C=(c_0, c_1, \ldots, c_{n-1})$ การเลื่อนครั้งที่ i ของคำรหัสนี้เขียนได้ ดังนี้

$$C^i = (c_{n-i}, c_{n-i+1}, \ldots, c_{n-i}, c_0, c_1, \ldots, c_{n-i-1}) \tag{2.21}$$

รหัสบล็อกเชิงเส้น ที่เป็นรหัสวนรอบ จะต้องมีคุณสมบัติ 2 ข้อ ดังต่อไปนี้

- คุณสมบัติรหัสบล็อกเชิงเส้นถูกต้อง
- การวนรอบใด ๆ ของคำรหัสที่ถูกต้อง จะเป็นคำรหัสที่ถูกต้องด้วยเสมอ

2.5.1.2 คำรหัสในรูปแบบของพหุนาม

คำรหัสของรหัสวนรอบนั้นสามารถเขียนแทนด้วยพหุนาม C(X) ซึ่งเป็นฟังก์ชันของ x โดยที่ค่าสัมประสิทธิ์ c_i เป็นสมาชิกของ GF(2^m) เมื่อ i คือจำนวนเต็มซึ่งแสดงถึงตำแหน่งของสัมประสิทธิ์นั้นๆ

กำหนดให้ $C = (c_0, c_1, \dots, c_{n-1})$ เป็นเวคเตอร์ของคำรหัส ดังนั้น พหุนามที่เขียนแทนเวคเตอร์คำรหัสนี้คือ C(X) ได้ดังนี้

$$C(X) = c_0 + c_1X + \cdots + c_{n-1}X^{n-1} \tag{2.22}$$

2.5.2 รหัสรีด – โซโลมอน

รหัสรีด – โซโลมอน (Reed – Solomon Codes: RS Codes) เป็นรหัสแก้ไขความผิดพลาดที่ถูกพัฒนาขึ้นในปี 1960 โดย Irving S. Reed และ Gustave Solomon จากบทความ [15] ในช่วงแรกนั้น อัลกอริทึมถอดรหัสนี้ยังไม่ค่อยดีนัก ในเวลาต่อมาในปี 1969 ได้มีการพัฒนาวิธีถอดรหัสที่ได้รับการยอมรับขึ้น โดย Elwyn Blerlekamp และ James Massey ซึ่งอัลกอริทึมถอดรหัสนี้ ก็ได้ถูกเรียกว่า วิธีการของเบอร์ลิแคมป์ – แมสซีย์ (Berlekamp – Massey Decoding Algorithm) ในปี 1982 รหัสรีด – โซโลมอน ได้ถูกนำมาใช้ครั้งแรกกับแผ่นซีดี (Compact Disc: CD) และได้ถูกนำมาใช้อย่างแพร่หลายในเวลาต่อมา

รหัสรีด – โซโลมอนเป็นรหัสวนรอบแบบไม่เป็นไบนารี่ (Non-Binary Cyclic Codes) เขียนแทนด้วย *RS(n,k)* และมีสัญลักษณ์ m บิต ซึ่งรหัส RS(n,k) บนสัญลักษณ์ที่มีความยาว m บิตนั้น มีขนาดซึ่งเขียนในรูปของ m,n และ k ได้ดังนี้

$$0 < k < n \leq 2^m - 1 \tag{2.23}$$

เมื่อ k คือจำนวนสัญลักษณ์ของข้อมูล ที่ถูกนำมาเข้ารหัส และ n คือจำนวนของสัญลักษณ์ทั้งหมดในคำรหัส 1 บล็อกหลังเข้ารหัสแล้ว แสดงให้เห็นว่า ส่วนการเข้ารหัสรีด – โซโลมอนนั้น จะประกอบไปด้วยข้อมูลทั้งหมด k สัญลักษณ์ และมีสัญลักษณ์พาริตีอีก n-k สัญลักษณ์ เพื่อเป็นคำรหัสยาว n สัญลักษณ์ดังที่กล่าวไปข้างต้น

สำหรับรหัสที่ใช้กันอย่างแพร่หลายของ RS(n,k) คือ

$$(n, k) = (2^m - 1, (2^m - 1) - 2t) \tag{2.24}$$

เมื่อ t คือจำนวนของสัญลักษณ์ที่รหัสสามารถแก้ไขได้ในรหัสนั้นๆ และกำหนดให้

$$t = \frac{(n-k)}{2} \tag{2.25}$$

ดังนั้นการเขียนแทนรหัสรีด – โซโลมอนด้วยพหุนาม จึงเขียนได้ดังนี้

$$c(x) = x^{2t}m(x) + p(x) \tag{2.26}$$

เมื่อ m(x) คือพหุนามกำหนดข้อมูลที่ต้องการนำมาเข้ารหัส เขียนแทนได้โดย

$$m(x) = m_0 + m_1x + m_2x^2 + \cdots + m_{k-1}x^{k-1} \tag{2.27}$$

และ p(x) คือพหุนามในส่วนของพาริตี เขียนได้ดังนี้

$$p(x) = p_0 + p_1x + p_2x^2 + \cdots + p_{2t-1}x^{2t-1} \tag{2.28}$$

และจากสมการที่ (2.26) (2.27) และ (2.28) สามารถนำมาจัดรูป และเขียนใหม่ได้ดังนี้

$$\begin{aligned} c(x) = {} & m_{k-1}x^{2t+k-1} + m_{k-2}x^{2t+k-2} + \cdots + m_1x^{2t+1} + m_1x^{2t} + p_{2t-1}x^{2t-1} \\ & + p_{2t-2}x^{2t-2} + \cdots + p_1x + p_0 \end{aligned} \tag{2.29}$$

2.6 การเข้า และถอดรหัสรีด-โซโลมอนที่เป็นที่นิยม

2.6.1 การเข้ารหัส รีด – โซโลมอน

การเข้ารหัสมีขั้นตอนหลัก ๆ อยู่ 2 ขั้นตอนหลัก คือ 1. คำนวณพหุนามกำเนิด 2. เข้ารหัสข้อมูล

2.6.1.1 คำนวณพหุนามกำเนิด

พหุนามกำเนิด (Generator Polynomial) ของรหัสรีด – โซโลมอน จะกำหนดจากรากที่ต่อเนื่องจำนวน 2t ราก หรือเมื่อเขียนเป็นรูปทั่วไป คือ

$$g(x) = \prod_{i=j_0}^{j_0+2t-1} (x - \alpha^i) \tag{2.30}$$

หรือ

$$g(x) = \sum_{i=0}^{2t} g_ix^i = g_0 + g_1x + g_2x^2 + \cdots + g_{2t}x^{2t} \tag{2.31}$$

เมื่อ $g_i \epsilon GF(2^m)$ ดีกรีของพหุนามจำนวนพหุนามกำเนิดคือ 2t และ $g_{2t} = 1$ เสมอ ซึ่งปกติแล้วจะให้ $j_0 = 1$ แต่บางกรณี $j_0 \neq 1$ ก็ได้ เช่น $2(j_0 + t) = 2^m$ ในกรณีต้องการสัมประสิทธิ์ $g(x)$ แบบสมมาตร

2.6.1.2 เข้ารหัสข้อมูล

1. การเข้ารหัสเชิงระบบ

เนื่องจากว่ารหัสรีด – โซโลมอน เป็นรหัสวนรอบ การเลื่อนขวาของคำรหัสก็ยังคงเป็นคำรหัสอยู่ เช่นนี้เมื่อเลื่อน k สัญลักษณ์ไปจนขวาสุด แล้วเติมส่วน 2t ตำแหน่งด้านหน้า ก็จะได้รหัสรีด – โซโลมอน เชิงระบบ การกระทำดังกลาวในเชิงคณิตศาสตร์ คือ เรากำลังทำ $x^{(n-k)}m(x)$ หลังจากนั้น จึงหาร $x^{(n-k)}m(x)$ ด้วย $g(x)$ จะได้เศษเหลือซึ่งก็คือการ พาริตีนั่นเอง ดังนั้น

$$x^{(n-k)}m(x) = Q(x)g(x) + p(x) \tag{2.32}$$

เมื่อ $Q(x)$ คือค่าเต็มผลหารที่ได้ (Quotient) และ $p(x)$ คือเศษเหลือ (reminder) และเมื่อการเข้ารหัสรีด – โซโลมอนนั้น สนใจเฉพาะเศษเหลือจึงเขียนใหม่ได้เป็น

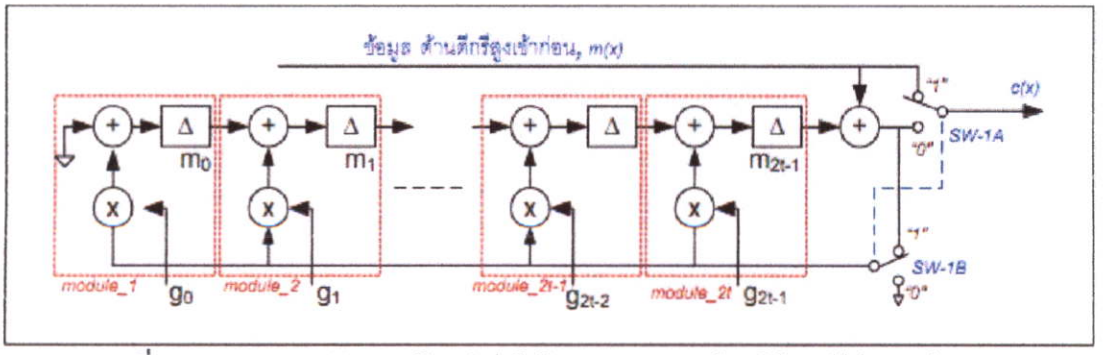
$$p(x) = x^{(n-k)}m(x)\ mod\ g(x) \tag{2.33}$$

ดังนั้นคำรหัสเชิงระบบที่ได้คือ

$$c(x) = x^{2t}m(x) + p(x) \tag{2.34}$$

2. การเข้ารหัสโดยใช้รีจิสเตอร์

รีจีสเตอร์ป้อนกลับเลื่อนเชิงเส้น (Linear Feedback Shift Register: LFSR) ซึ่งใช้ในการคำนวณการหารเศษเหลือ หรือมอดูโลสามารถนำมาใช้ในการเข้ารหัสได้เป็นอย่างดี ข้อมูลซึ่งดีกรีสูงเข้ามาก่อน จะถูกส่งออกไปพร้อมๆกับการโหลดเข้าสู่วงจรหารเศษเหลือทีเป็นรีจิสเตอร์ป้อนกลับเลื่อนเชิงเส้น เนื่องจากข้อมูลทั้งหมดมีจำนวน k ข้อมูล และความยาวของรีจีสเตอร์ป้อนกลับนี้มีขนาด 2t สเตท ช่วงแรกนี้จึงเป็นการกระทำสมการ $x^{(n-k)}m(x)\ \mathbf{mod}\ g(x)$ เมื่อครบจำนวน k ข้อมูล (หรือ k สัญลักษณ์) แล้ว สวิทช์ SW-1 จะเปลี่ยนไปทางตรงข้าม เพื่อให้ผลเศษหารเลื่อนออกไป ซึ่งช่วงนี้จะไม่มีการป้อนกลับใด ๆ



รูปที่ 2.4 วงจรสำหรับการเข้ารหัสรีดโซโลมอนเชิงระบบโดยใช้ชิฟต์รีจีสเตอร์

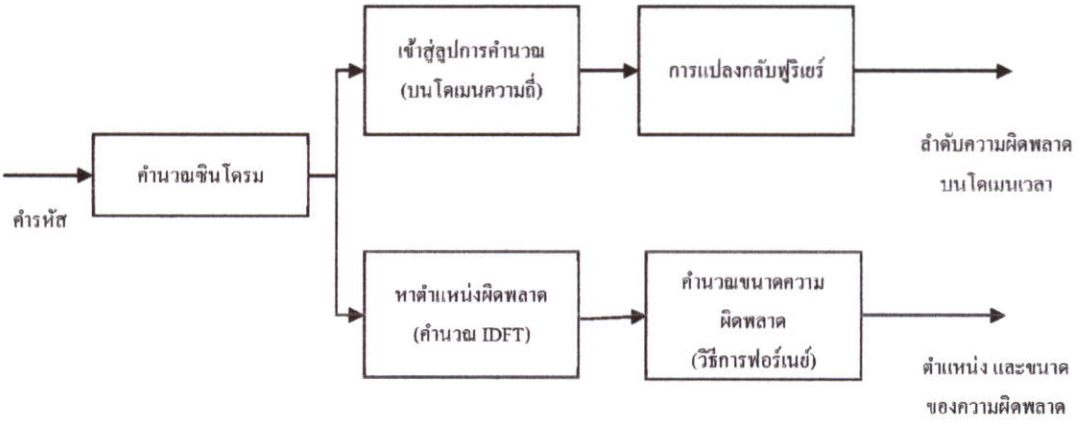
ดังนั้นคำรหัสที่ส่งออกไปจะเขียนเป็นพหุนามได้ คือ

$$\begin{aligned} c(x) &= m_{2t+k-1}x^{2t+k-1} + m_{2t+k-2}x^{2t+k-2} + \cdots \\ &\quad + m_0 x^{2t} \\ &\quad + p_{2t-1}x^{2t-1} + \cdots + p_2 x^2 + p_1 x + p_0 \\ &= c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_k x^k + \\ &\quad c_{2t-1}x^{2t-1} + \cdots + c_2 x^2 + c_1 x + c_0 \end{aligned} \tag{2.35}$$

2.6.2การถอดรหัสรีด – โซโลมอน

2.6.2.1 การถอดรหัสรีด – โซโลมอนบนโดเมนความถี่

การถอดรหัสบนโดเมนความถี่นั้น ในขั้นตอนแรกจะต้องมีการคำนวณซินโดรมก่อน ซึ่งการคำนวณนี้ จะทำได้ในรูปของพหุนามที่รากกำเนิด และสามารถเปลี่ยนกลับมาเป็นข้อมูลในโดเมนฟูริเยร์ สมการที่เป็นหัวใจหลักนั้นจะถูกคำนวณในโดเมนความถี่ จากนั้นการแปลงกลับฟูริเยร์ของลำดับค่าผิดพลาดจะถูกนำมาคำนวณในตอนท้ายบนโดเมนเวลา ซึ่งการถอดรหัสบนโดเมนความถี่ที่เป็นที่รู้จัก และใช้กันอย่างแพร่หลายนั้น แสดงดังรูป



รูปที่ 2.5 การถอดรหัสบนโดเมนความถี่ทั้งสองแบบ

ในแบบแรกนั้น ลำดับของความผิดพลาดบนโดเมนความถี่จะได้รับจากลูปการคำนวณ หลังจากนั้นการแปลงกลับฟูริเยร์แบบไม่ต่อเนื่อง จำนวน N จุดจะถูกนำมาใช้เพื่อเปลี่ยนกลับเป็นโดเมนเวลา แล้วข้อความที่ผิดพลาดไปนั้น จะถูกกลับมาแก้ในโดเมนเวลา

ตัวอย่างของกระบวนการถอดรหัสแบบวิธีหนึ่งคือ [12] แทนที่จะคำนวณแค่ 2t รอบด้วยการใช้การแปลงฟูริเยร์แบบไม่ต่อเนื่อง ก็ใช้การคำนวณข้อมูลจำนวน N สัญลักษณ์นั้นแทน ดังนั้น คำรหัสที่ได้มา จะถูกเปลี่ยนไปอยู่บนโดเมนความถี่ เมื่อได้ลำดับผิดพลาดมาจากโดเมนความถี่แล้ว ก็เอาข้อมูลที่ได้มาใช้การแปลงกลับฟูริเยร์ เพื่อคำนวณข้อมูลจริงได้เลย

ในแบบที่สองที่แสดงในรูปที่ 5 นั้นตำแหน่งของความผิดพลาดนั้นหาได้จากรากของพหุนามบอกตำแหน่งโดยใช้กระบวนคำนวณหาตำแหน่งผิดพลาด กระบวนการนี้มีค่าเท่ากับการแปลงโดเมนกลับ ของลำดับความผิดพลาด หลังจากนั้นก็มาคำนวณขนาดความผิดพลาดจากวิธีการของฟอร์เนย์ (Forney Technique)

2.6.2.2 วิธีการเบอร์ลิแคมป์ – แมสซีย์

เป็นการมองสมการซินโดรมเป็นการแก้ปัญหาแบบชิฟต์รีจิสเตอร์ แต่ในวิธีการนี้จะหารีจิสเตอร์ที่มีความยาวต่ำที่สุด ที่ทำให้สมการยังคงเป็นจริง ที่สำคัญคือเราต้องหาพหุนามบอกตำแหน่งผิดพลาด (Error Locator Polynomial: $\Lambda(x)$) ซึ่งการสังเคราะห์รีจิสเตอร์นี้ จะเริ่มจากรีจิสเตอร์ที่มีความยาวน้อยๆก่อน แล้วลองคำนวณค่าความต่างของซินโดรมซึ่งเรียกว่า Discrepancy (Δ_r) แล้วนำค่าดังกล่าวไปปรับค่าสัมประสิทธิ์ และเพิ่มจำนวนแท็ปของฟิลเตอร์ แล้วมีการเก็บค่าในรีจีสเตอร์สัมประสิทธิ์เอาไว้เพื่อปรับปรุงค่าสัมประสิทธิ์ใหม่ ซึ่งจะแสดงวิธีการคำนวณอย่างละเอียดในบทถัดไป

2.6.2.3 วิธีการยูคลิด

ในระบบเลขจำนวนเต็มเราจะเห็นได้ว่า gcd{*u,v*} = gcd{*u,(v)* mod *u*} เช่น gcd{6,9} = gcd{6,9 mod 6} ซึ่ง gcd{6,9} = 3 และ gcd{6,9 mod 6} = 3 เป็นต้น และยังแสดงได้ว่า gcd{*u,v*} = *au* + *bv* ซึ่ง 3 = *a*6 + *b*9 = (-1)(6) + 1(9) ในทำนองเดียวกันหากเป็นนพหุนาม ขั้นตอนวิธียูคลิดแสดงได้ว่า gcd{*w(x),t(x)*} = gcd{*w(x),t(x)* mod *w(x)*} ซึ่งคำตอบที่ได้รับจะสอดคล้องกับสมการ gcd{*w(x),t(x)*} = *w(x)u(x)* + *t(x)v(x)* สำหรับพหุนาม *u(x)* และ *v(x)* ใดๆ โดยขั้นตอนการคำนวณวนซ้ำสามารถจะเขียนได้ว่า

$$t^r(x) = A_{21}^r(x)w(x) + A_{22}^r(x)t(x) \tag{2.36}$$

สำหรับพหุนาม A_{21}^r และ A_{22}^r ใดๆ จากสมการนี้หากเราให้ $w(x) = x^{2t}$ และ $t(x) = S(x)$ แล้วทำการมอดูโลสมการข้างบนนี้ด้วย $w(x)$ ก็จะได้

$$t^r(x) = A_{22}^r(x)S(x) mod x^{2t} \tag{2.37}$$

ซึ่งเมื่อเทียบฟอร์มแล้วจะเหมือนกรณีของรหัสรีด – โซโลมอนโดยวิธีการฟอร์เนย์ ซึ่ง $\Gamma(x) = \Lambda(x)S(x) \mathrm{mod} x^{2t}$ และที่บางเฉพาะรอบการคำนวณซึ่ง $\deg. A_{22}^r(x) \leq t$ และ $\deg. t^r(x) \leq t - 1$ ก็จะทำให้ได้ $t^r(x) = \Gamma(x)$ และ $A_{22}^r(x) = \Lambda(x)$ ซึ่งในขั้นตอนต่อไปนี้เป็นการแก้สมการซินโดรม

กำหนดค่าเริ่มต้น ดังนี้

$w^0(x) = x^{2t}, \Gamma(x) = S(x), u^0(x) = 0, \Lambda^0(x) = 1$

ในทุกรอบการคำนวณ

$$d = deg.w^{(r-1)}(x) - deg.\Gamma^{(r-1)}(x) \tag{2.38}$$

q = อัตราส่วนของสัมประสิทธิ์นำของ $w^{(r-1)}(x)$ และ $\Gamma^{(r-1)}(x)$

$$\begin{bmatrix} w^r(x) \\ \Gamma^r(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -qx^d \end{bmatrix} \begin{bmatrix} w^{(r-1)}(x) \\ \Gamma^{(r-1)}(x) \end{bmatrix} \tag{2.39}$$

$$\begin{bmatrix} u^r(x) \\ \Lambda^r(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -qx^d \end{bmatrix} \begin{bmatrix} u^{(r-1)}(x) \\ \Lambda^{(r-1)}(x) \end{bmatrix} \tag{2.40}$$

โดยการคำนวณจะหยุดเมื่อ $\Gamma(x) < t$

ในแต่ละรอบการคำนวณนั้น ดีกรีของพหุนาม $\Gamma(x)$ จะลดลงด้วยผลต่างของดีกรีของ $\Gamma(x)$ และ $w(x)$ ในขณะเดียวกันดีกรีของ $\Lambda(x)$ จะเพิ่มขึ้นอย่างต่อเนื่อง ในขณะเดียวกันพหุนาม $w(x)$ และ $\Gamma(x)$ ก็จะสลับกันในทุกรอบการคำนวณด้วย การคำนวณทั้งหมด $2t$ รอบ สำหรับรหัสที่แก้ไขได้ t สัญลักษณ์ผิดพลาด

$$w^r(x) = \Gamma^{(r-1)}(x) \tag{2.41}$$

$$\Gamma^{(r)}(x) = w^{r-1}(x) - qx^d\Lambda^{(r-1)}(x) \tag{2.42}$$

$$u^r(x) = \Lambda^{(r-1)}(x) \tag{2.43}$$

$$\Lambda^r(x) = u^{(r-1)}(x) - qx^d\Lambda^{(r-1)}(x) \tag{2.44}$$

2.7 สมการซินโดรม และการแก้สมการซินโดรมสำหรับรหัสรีด-โซโลมอน

การเขียนสมการแทนด้วยพหุนามนั้น เราจะได้รับพหุนามซึ่งประกอบไปด้วยข้อมูลจริง และข้อมูลผิดพลาดของสัญญาณ ซึ่งเขียนแทนได้ดังนี้

$$v(x) = c(x) + e(x) \tag{2.45}$$

ซึ่งพหุนามซินโดรม S(x) นั้น ก็คือสมการที่ใช้ตรวจสอบความถูกต้องของลำดับ v{i} ในแต่ละรากของ
พหุนามกำเนิด ที่คำนวณได้ในขั้นตอนการเข้ารหัส ดังนั้น

$$\begin{aligned} S_j &= v(\alpha^k) = v_0(\alpha^k)^0 + v_1(\alpha^k)^1 + v_2(\alpha^k)^2 + \cdots + v_{N-1}(\alpha^k)^{N-1} \\ &= c(\alpha^k) + e(\alpha^k) \\ &= e(\alpha^k) \end{aligned} \quad (2.46)$$

เมื่อ $j = 1,2,\ldots,2t$

เนื่องจาก $\{c_k\} = 0$, เมื่อ $k = t_0, t_0+1, t_0+2, .., t_0+2t$ ซึ่งในแต่ละอิเลเมนต์ของซินโดรม S_j นั้นคือการเขียนแทนรูปแบบของพหุนามข้อมูลผิดพาด $e(x)$ อย่างง่ายที่ $x = \alpha^k$ ซึ่งแต่ละค่านั้นเป็นสมาชิกใน GF[2^m] จากสมการ (2.46) พหุนามนี้ก็คือการแปลงฟูริเยร์ ของลำดับ $\{v_i\}$ บนสนามจำกัดซึ่งค่า α แต่ละค่าก็คือรากปฐมของสนามจำกัดนั่นเอง เพราะว่าในขั้นตอนการเข้ารหัสนั้นได้บังคับให้มีการบังคับให้ $\{e_i\} = 0$ ทั้ง $2t$ ค่า ดังนั้น ถ้าหากไม่มีค่าผิดพลาดใดๆ S_j จะต้องมีค่าเป็น 0 แต่หากมีความผิดพลาด $S_j = E_j$

กำหนดให้พหุนามระบุตำแหน่งผิดพลาด (Error Locator Polynomial) $\Lambda(x)$ ดังนี้

$$\Lambda(x) = \prod_{i \in P}(1 - x\alpha^i) \quad (2.47)$$

โดยที่ $P = \{1,2,3,\ldots,t\}$

จากวิธีการดังกล่าว $\Lambda(x)$ สามารถหาตำแหน่งของข้อมูลที่ผิดพลาดได้

กำหนด $\{\lambda_i\}$ เป็นค่าการแปลงกลับฟูริเยร์ของ $\{\Lambda_k\}$ ดังนั้น $\lambda_i = 0$ เมื่อ $e_i \neq 0$ ผลคูณในแต่ละตำแหน่ง $\lambda_i e_i = 0$ ในโดเมนเวลานั้น สามารถเปลี่ยนรูปให้อยู่ในรูปแบบการคอนโวลูชันแบบวนรอบบนโดเมนความถี่ และกำหนดค่าของตำแหน่งด้วยการมอดูโลด้วย N

$$\sum_{i=0}^{N-1} \Lambda_i E_{k-i} = 0, \text{เมื่อ } k = 0,.., N\text{-}1 \quad (2.48)$$

$\Lambda_0 = 1$ และ $\Lambda(x)$ จะเป็นได้สูงสุดถึงดีกรีสูงสุด v แต่ว่าตั้งแต่ดีกรี $v+1$ ขึ้นไปของ $\{\Lambda_i\}$ นั้นสามารถมีค่าไม่เป็นศูนย์ได้ และสามารถเขียนเป็นสมการได้ดังนี้

$$E_k = -\sum_{i=1}^{N-1} \Lambda_i E_{k-i} \quad \text{สำหรับ } k = 0,.., N\text{-}1 \quad (2.49)$$

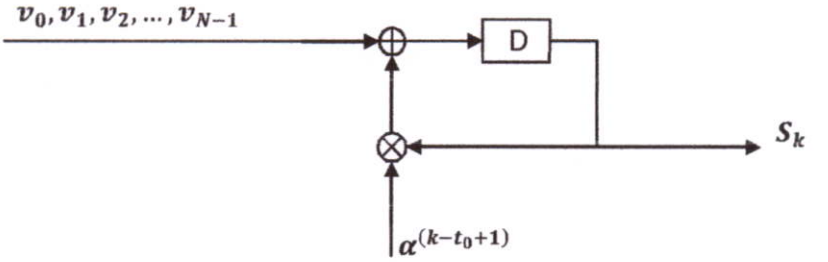
หรือเขียนแทนด้วยสมการซินโดรม ได้ดังนี้

$$S_k = -\sum_{i=1}^{N-1} \Lambda_i S_{k-i} \quad \text{สำหรับ } k = v+1 - 2t \quad (2.50)$$

สมการที่ (2.49) มักจะกล่าวถึงสมการซินโดรม ซึ่งควรจะบอกได้ว่า กระบวนการถอดรหัสเหล่านี้นั้นมีความน่าเชื่อถือโดยคุณสมบัติของส่วนเข้ารหัส ถ้าหาก ส่วนเข้ารหัสนั้น ไม่บังคับให้ค่าซินโดรมทั้ง $2t$ ค่ามีค่าเป็นศูนย์แล้ว ก็จะไม่สามารถได้ค่า $2t - v$ ค่าที่ถูกต้องได้ด้วยสมการ (2.50) และอีกทั้งยังไม่สามารถหาตำแหน่งข้อมูลผิดพลาดที่ถูกต้องได้อีกด้วย

สำหรับ t ที่มีค่าต่ำ ส่วนกลับของเมตริกซ์สามารถนำมาใช้เพื่อแก้ปัญหาได้ด้วยสมการที่กำหนดให้ซึ่งจะมีค่าของการคำนวณเท่ากับ $O(t^3)$ ดังนั้น สมการนี้จึงเหมาะสำหรับที่จะใช้กับการคำนวณที่มีค่า t ต่ำๆ ส่วนในรหัสที่มีการแก้ข้อมูลผิดพลาดได้มากๆนั้น (ค่า t สูง) จะใช้วิธีการของเบอร์ลิแคมป์ – แมสซีย์ หรือ วิธีการยูคลิดจะดีกว่า ซึ่งใช้การคำนวณเพียงแค่ $O(t^2)$ เท่านั้น

การคำนวณค่าซินโดรม สามารถใช้ได้ทั้งวิธีการแปลงฟูริเยร์แบบไม่ต่อเนื่อง (DFT) และการแปลงฟูริเยร์แบบไม่ต่อเนื่องแบบเร็ว (FDFT) ในการคำนวณ DFT นั้นสามารถใช้วิธีการฮอร์เนอร์ (Horner's Scheme) ดังแสดงในรูป เพื่อการถอดรหัสความเร็วสูง ข้อมูลทั้ง $2t$ ค่า จะถูกนำมาประมวลผลแบบขนาน คือ 1 ค่า ต่อ 1 คอมโพเนนต์ซินโดรม ซึ่งการคำนวณในทุกๆคอมโพเนนต์นั้น ซึ่งในการคำนวณคอมโพเนนต์เหล่านี้นั้น แต่ละค่าของ $\alpha^{(k-t_0-1)}$ สำหรับ $k = 1 - 2t$ จำเป็นจะต้องกำหนดค่าคงที่ไว้ ดังนั้นในการทำให้ฮาร์ดแวร์ทำงานง่าย จำเป็นจะต้องกำหนดค่าตัวคูณแต่ละค่าไว้อย่างตายตัว



รูปที่ 2.6 สถาปัตยกรรมแสดงการคำนวณซินโดรมด้วยวิธี DFT โดยตรง

จำนวณการคูณสูงสุดที่ต้องการสำหรับการคำนวณซินโดรมด้วยวิธีนี้มีค่าเท่ากับ 2t(N-1) เมื่อ N คือความยาวบล็อกของคำรหัส และวิธีนี้ใช้เวลาน้อยที่สุด เนื่องจาก ซินโดรมทุกค่านั้นถูกคำนวณเสร็จพร้อมกันหมดหลังจากได้รับคำรหัสสัญลักษณ์สุดท้ายแล้ว ส่วนวิธีการ FDFT นั้นจะใช้เวลานานกว่า เพราะว่าจำเป็นจะต้องคำนวณ DFT เสร็จครบทุกค่าก่อน สำหรับทุกค่าในบล็อกที่ประกอบกันขึ้นมานั้น การแปลงแบบเร็วอาจจะได้ผลดีกว่าการแปลงโดยตรง อย่างไรก็ตาม วิธีการ FDFT นั้นไม่สามารถใช้ได้กับคำรหัสแบบสั้น ในการถอดรหัสคำรหัสสั้นได้

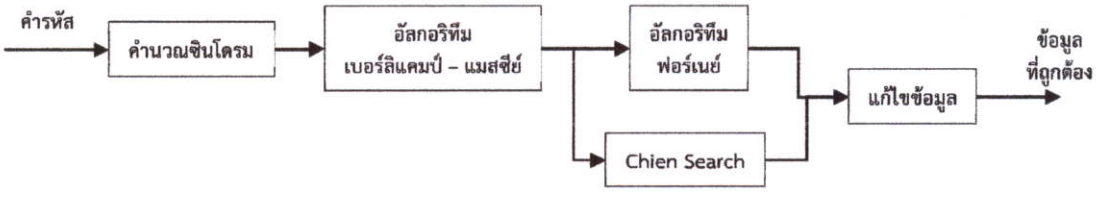
2.8 รหัสรีด – โซโลมอนบนโดเมนความถี่ และโดเมนเวลา

การถอดรหัสรีด – โซโลมอนนั้น กระทำได้ทั้งสองโดเมน คือ โดเมนความถี่คือการนำคำรหัส (Codeword) มาทำการแปลงฟูริเยร์เพื่อทำการถอดรหัสโดยใช้ซินโดรม แล้วทำการแปลงกลับฟูริเยร์เพื่อให้ได้ข้อมูลที่ถูกต้อง และโดเมนเวลา ซึ่งสามารถนำคำรหัสมาถอดรหัสได้โดยตรงโดยไม่ต้องทำการคำนวณซินโดรมก่อน ซึ่งแต่ละวิธี มีกระบวนการถอดรหัสคร่าว ๆ ดังนี้

2.8.1 รหัสรีด – โซโลมอน บนโดเมนความถี่

การถอดรหัสรีด – โซโลมอนนั้น สามารถทำได้ทั้งด้วยวิธีถอดจากข้อมูลโดยตรง หรือการเปลี่ยนด้วยวิธีการแปลงฟูริเยร์แบบไม่ต่อเนื่องบนสนามจำกัดกาลัวส์ (Galois Field Discrete Fourier Transform) ซึ่งวิธีที่สองนี้ ถูกเรียกว่าการถอดรหัสบนโดเมนความถี่ ซึ่งได้ถูกใช้กันอย่างแพร่หลายในการออกแบบระบบถอดรหัสรีด – โซโลมอน วิธีการถอดรหัสบนโดเมนความถี่นี้ทำได้โดยการคำนวณซินโดรมซึ่งก็คือการแปลงฟูริเยร์แบบไม่ต่อเนื่อง แยกในแต่ละซินโดรม นำซินโดรมที่ได้มาแก้สมการเพื่อหาสมการระบุตำแหน่งผิดพลาด (Error Locator Polynomial) เมื่อได้ผลลัพธ์แล้วก็มาทำการแปลงกลับฟูริเยร์แบบไม่ต่อเนื่อง (IDFT) หรือการแปลงกลับฟูริเยร์แบบไม่ต่อเนื่องอย่างเร็ว (Fast IDFT) เพื่อนำค่า และตำแหน่งผิดพลาดมาหาค่าที่ถูกต้อง

กระบวนการหลักของวิธีการนี้ก็คือการแก้สมการหลักเพื่อที่จะได้พหุนามระบุตำแหน่ง รากของพหุนามระบุตำแหน่งที่ได้นี้ จะบอกตำแหน่งของค่าที่มีความผิดพลาดของคำรหัส สำหรับการถอดรหัสในคำรหัสที่มีความยาวไม่มาก จะใช้ค่าส่วนกลับของเมตริกซ์เพราะสามารถหาได้ไม่ยาก อย่างไรก็ตาม สำหรับคำรหัสที่มีความยาวปานกลางขึ้นไป การคำนวณในลักษณะนี้จะไม่นิยมเท่าใดนัก เนื่องด้วยการที่ต้องอาศัยการคำนวณถึง $O(t^3)$ เมื่อ t คือจำนวนของสัญลักษณ์ผิดพลาดสูงสุดที่สามารถแก้ไขได้ เพราะฉะนั้นจึงมีการใช้เทคนิควิธีการอื่นเพื่อแก้สมการซินโดรม โดยวิธีที่เป็นที่นิยมนำมาใช้เพื่อพัฒนา และอิมพลิเมนต์ลงฮาร์ดแวร์ คือวิธีการของเบอร์ลิแคมป์ – แมสซีย์ และวิธีการยูคลิด และเพื่อที่จะให้ได้ความเร็วสูงสุดวิธีการไปป์ไลน์จึงเป็นที่นิยมนำมาใช้งาน แสดงการทำงานของอัลกอริทึมการถอดรหัสเบอร์ลิแคมป์ – แมสซีย์ ได้ดังนี้

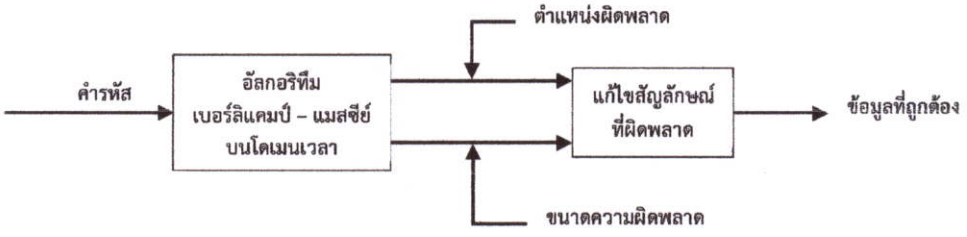


รูปที่ 2.7 แผนภาพการถอดรหัสเบอร์ลิแคมป์ – แมสซีย์บนโดเมนความถี่

2.8.2 รหัสรีด – โซโลมอน บนโดเมนเวลา

จากการถอดรหัสบนโดเมนความถี่นั้น จะเห็นว่าอัลกอริทึมของเบอร์ลิแคมป์ – แมสซีย์ ใช้ในการคำนวณเพื่อหาพหุนามระบุตำแหน่งผิดพลาด และพหุนามประเมินค่าผิดพลาด หลังจากการแปลงข้อมูลครั้งแรกด้วยวิธีการแปลงฟูริเยร์แบบไม่ต่อเนื่องบนสนามกาลัวส์ หรือการคำนวณซิน

โดรม วิธีที่เป็นที่นิยมอีกอย่างหนึ่งก็คือ การถอดรหัสบนโดเมนเวลา ซึ่งพัฒนาขึ้นมาโดยการใช้การแปลงกลับฟูริเยร์แบบไม่ต่อเนื่อง (IDFT) ลงไปในทุกสมการ ในวิธีการของเบอร์ลิแคมป์ – แมสซีย์ของการถอดรหัสบนโดเมนความถี่ ซึ่งอัลกอริทึมนี้จะคำนวณกับคำรหัสโดยตรงโดยไม่ต้องทำการแปลงฟูริเยร์ และจะได้ลำดับของความผิดพลาดมา แล้วนำมาคำนวณกับข้อมูลโดยไม่ต้องมีกระบวนการแปลงกลับฟูริเยร์เลยในกระบวนการถอดรหัส เขียนเป็นแผนภาพอัลกอริทึมคร่าวๆได้ดังนี้



รูปที่ 2.8 แผนภาพอัลกอริทึมการถอดรหัสบนโดเมนเวลา

2.9 งานวิจัยที่เกี่ยวข้อง

การถอดรหัสรีด – โซโลมอนนั้นมีงานวิจัยที่เกี่ยวข้องจำนวนมาก เราจึงเลือกศึกษาเฉพาะงานวิจัยที่เกี่ยวข้องกับเป้าหมายของงานวิจัยนี้ โดยแบ่งวรรณกรรมที่เกี่ยวข้องเป็นสองส่วน คืออัลกอริทึมเกี่ยวกับการพัฒนาการถอดรหัสบนโดเมนเวลา สถาปัตยกรรมของการถอดรหัสบนฮาร์ดแวร์ ซึ่งในแต่ละงานวิจัยนั้นมีวัตถุประสงค์ซึ่งเน้นไปในทางการพัฒนาอัลกอริทึมเพื่อลดเวลา และทรัพยากรฮาร์ดแวร์ เพื่อการทำงานที่รวดเร็ว และประหยัดต้นทุนในการสร้างฮาร์ดแวร์

2.9.1 การพัฒนาอัลกอริทึมเพื่อลดการคำนวณ

ใน [12] ได้กล่าวถึงการพัฒนาอัลกอริทึมการถอดรหัสจากโดเมนความถี่ มาเพื่อเป็นอัลกอริทึมการถอดรหัสบนโดเมนเวลา เพื่อให้ง่ายต่อการออกแบบบนฮาร์ดแวร์ ด้วยการแปลงกลับฟูริเยร์เต็มหน่วย (Inverse Discrete Fourier Transform: IDFT) ในทุกๆ สมการในขั้นตอนเบอร์ลิแคมป์ - แมสซีย์ ซึ่งอัลกอริทึมบนโดเมนนี้ จะกระทำการถอดรหัสต่อคำรหัสโดยตรง ดังนั้น อัลกอริทึมนี้ต้องการทั้งการ แปลงฟูริเยร์ และการแปลงกลับ ในหลายๆขั้นตอน เช่น คำนวณซินโดรม วิธีการค้นหาของเฉียน เป็นต้น ด้วยเหตุนี้ อัลกอริทึมนี้สามารถนำไปใช้ได้โดยการนำกระบวนการเดียวไปใช้ซ้ำ ๆ ในหลาย ๆ ที่ได้ ซึ่งเป็นสิ่งที่สำคัญเพื่อนำไปใช้ออกแบบวงจรรวมความจุสูง (Very Large Scale Integration Design: VLSI Design) ได้

ใน [14] ได้นำเสนอการใช้เทคนิคการแปลงฟูริเยร์อย่างเร็ว (Fast Transform Technique: FTT) สำหรับคำนวณบนสนามจำกัดด้วยวิธีการแปลงโดยใช้ทฤษฎี NTT (Number-Theoretic Transform) จากสมการของ NTT

$$A_k = \sum_{i=0}^{n} \alpha_i \beta^{ik} \tag{2.51}$$

ซึ่งการใช้ซินโดรมนั้นก็คือการแปลงคำรหัสให้อยู่ในรูปของโดเมนความถี่นั่นเอง จึงได้นำวิธีการ NTT มาแปลงให้อยู่ในรูปของการคำนวณซินโดรมได้ดังนี้

$$S_k = \sum_{i=0}^{n} v_i \alpha^{ik} \tag{2.52}$$

การใช้สมการ ทำให้ไม่ต้องคำนวณเมตริกซ์ขนาดใหญ่ ซึ่งสามารถคำนวณซินโดรมแต่ละค่าตั้งแต่ 1 – 2t ได้แบบขนานทำให้ประหยัดเวลา และทรัพยากรลงไปด้วย

อย่างไรก็ตาม ข้อด้อยของอัลกอริทึมบนโดเมนเวลานี้ก็ยังคงมีอยู่ นั่นก็คือการที่ต้องใช้ปริมาณการคำนวณสูงมาก ดังนั้น ใน [12] ด้านงานวิจัยสำหรับโดเมนเวลาจึงเน้นไปที่การแก้ปัญหาด้านปริมาณการคำนวณที่สูง ซึ่งจะต้องกระทำการคำนวณกับข้อมูลโดยตรงคือบล็อกละ N สัญลักษณ์ ในขณะที่การถอดรหัสบนโดเมนความถี่นั้นจะทำการคำนวณแค่ซินโดรมซึ่งมีความยาวเพียงแค่ N-k เท่านั้น ดังนั้น เพื่อที่จะให้อัลกอริทึมบนโดเมนเวลาน่าศึกษาวิจัย จำเป็นจะต้องศึกษาในเรื่องของการลดจำนวนของการคำนวณ

หลังจากศึกษาการพัฒนาจาก [12] แล้ว ใน [16] จึงได้ทำการวิจัยพัฒนาสถาปัตยกรรมฮาร์ดแวร์ สำหรับถอดรหัสรีด – โซโลมอน บนโดเมนเวลา ซึ่งได้นำเสนอสิ่งที่เป็นประโยชน์ในงานวิจัยไว้ คือ อย่างแรกอัลกอริทึมที่ใช้การคำนวณน้อยลงซึ่งสามารถลดจำนวนการคูณเหลือเพียงแค่ 60% สำหรับการถอดรหัสบนโดเมนเวลา อย่างที่สอง อัลกอริทึมนั้นสามารถอิมพลิเมนต์เป็นสถาปัตยกรรมแบบอาเรย์เดี่ยวได้ ซึ่งในแต่ละอาเรย์นั้นมี ตัวกำหนดตำแหน่งผิดพลาด (Error Locator) และ อีกอาเรย์ไว้สำหรับคำนวณการวนรอบ อย่างที่สาม อัลกอริทึมที่ได้จากการวิจัยนี้ ง่ายต่อการนำไปประยุกต์ใช้กับรหัสที่สั้นลง (Truncated Codes) ด้วยการลดลำดับการคำนวณของอัลกอริทึม หรือสถาปัตยกรรมฮาร์ดแวร์ซึ่งจะกล่าวถึงในหัวข้อถัดไป

จากอัลกอริทึมบนโดเมนความถี่

กำหนดค่าเริ่มต้น $\Lambda^0(x) = \Gamma^0(x) = 1$;
$L_0 = 0$;
$K_0 = u_0 = 1$;

$$\Delta_r = \sum_{j=0}^{t} \Lambda_j^{r-1} S_{r-j} \tag{2.53}$$

$$L_r = \delta_r(r - L_{r-1}) + (1 - \delta_r)L_{r-1} \tag{2.54}$$

$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r)K_{r-1} \tag{2.55}$$

$$u_r = \delta_r + (1 - \delta_r)(u_{r-1} + 1) \tag{2.56}$$

$$\begin{bmatrix}\Lambda^r(x)\\ \Gamma^r(x)\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r K_{r-1}x^{\mu_r}\\ \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\Lambda^{r-1}(x)\\ \Gamma^{r-1}(x)\end{bmatrix} \tag{2.57}$$

$$Z^r(x)=x^{2-t_0}\frac{d\Lambda^r(x)}{dx} \tag{2.58}$$

$$\Xi^r=x^{2-t_0}\frac{d\gamma^r(x)}{dx} \tag{2.59}$$

สามารถขยายอัลกอริทึมด้วยการเพิ่มสมการสำหรับพหุนามระบุตำแหน่ง และประเมินตำแหน่งความผิดพลาดจะได้สมการดังนี้

กำหนดค่าเริ่มต้น $\Lambda^0(x)=\Gamma^0(x)=\Omega^0(x)=1;$
$Z^0(x)=\Xi^0(x)=\mathrm{M}^0(x)=0;$
$L_0=0;$
$K_o=u_0=1;$

$$\Delta_r=\sum_{j=0}^{t}\Lambda_j^{r-1}S_{r-j} \tag{2.60}$$

$$L_r=\delta_r(r-L_{r-1})+(1-\delta_r)L_{r-1} \tag{2.61}$$

$$K_r=\delta_r\Delta_r^{-1}+(1-\delta_r)K_{r-1} \tag{2.62}$$

$$\begin{bmatrix}\Lambda^r(x)\\ \Gamma^r(x)\\ \mathrm{Z}^r(x)\\ \Xi^r(x)\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r\mathrm{K}_{r-1}x^{\mu_r} & 0 & 0\\ \delta_r & (1-\delta_r) & 0 & 0\\ 0 & -\Delta_r\mathrm{K}_{r-1}u_r x^{\mu_r+1-t_0} & 1 & -\Delta_r\mathrm{K}_{r-1}x^{\mu_r}\\ 0 & 0 & \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\Lambda^{r-1}(x)\\ \Gamma^{r-1}(x)\\ \mathrm{Z}^{r-1}(x)\\ \Xi^{r-1}(x)\end{bmatrix} \tag{2.63}$$

$$\begin{bmatrix}\Omega^r(x)\\ \mathrm{M}^r(x)\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r K_{r-1}x^{\mu_r}\\ \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\Omega^{r-1}(x)\\ \mathrm{M}^{r-1}(x)\end{bmatrix} \tag{2.64}$$

For $r=1,2\ldots 2t$. *Here* $\delta_r=1$ *if both* $\Delta_r\neq 0$ *and* $2L_{r-1}\leq r-1$; *otherwise* $\delta_r=0$. *If* $\delta_r=0,\mu_r=\mu_{r-1}+1$; *otherwise* $\mu_r=1$. *Then*

$$\boldsymbol{e_i=\frac{\Omega^{2t}(x)}{\mathrm{Z}^{2t}(x)}}\quad \text{เมื่อ } \boldsymbol{x=\alpha^{-i}} \text{ เป็นรากของ } \boldsymbol{\Lambda^{2t}(x)} \tag{2.65}$$

เมื่อได้อัลกอริทึมที่ลดการคำนวณลงได้แล้วทำการเปลี่ยนให้อยู่ในรูปของโดเมนเวลาโดยการแปลงฟูริเยร์อย่างเร็วแบบไม่ต่อเนื่อง (IDFT) เข้าไปในทุกสมการจะได้

กำหนดค่าเริ่มต้น $\lambda_i^0=\gamma_i^0=\omega_i^0=1;$
$\zeta_i^0=\xi_i^0=\mu_i^0=0;$
$L_0=0;$
$\mathrm{K}_0=u_0=1;$

$for\ i=0,1,\ldots,N-1$

$$\Delta_r=\sum_{i=0}^{N-1}v_i\lambda_i^{r-1}\alpha^{i(r+t_0-1)} \tag{2.66}$$

$$L_r=\delta_r(r-L_{r-1})+(1-\delta_r)L_{r-1} \tag{2.67}$$

$$\mathrm{K}_r=\delta_r\Delta_r^{-1}+(1-\delta_r)\mathrm{K}_{r-1} \tag{2.68}$$

$$\begin{bmatrix}\lambda_i^r\\ \gamma_i^r\\ \zeta_i^r\\ \xi_i^r\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r \mathrm{K}_{r-1}\alpha^{-iu_r} & 0 & 0\\ \delta_r & (1-\delta_r) & 0 & 0\\ 0 & -\Delta_r \mathrm{K}_{r-1}u_r\alpha^{-i(u_r+1-t_0)} & 1 & -\Delta_r K_{r-1}\alpha^{-iu_r}\\ 0 & 0 & \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\lambda_i^{r-1}\\ \gamma_i^{r-1}\\ \zeta_i^{r-1}\\ \xi_i^{r-1}\end{bmatrix} \tag{2.69}$$

$$\begin{bmatrix}\omega_i^r\\ \mu_i^r\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r K_{r-1}\alpha^{-iu_r}\\ \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\omega_i^{r-1}\\ \mu_i^{r-1}\end{bmatrix} \tag{2.70}$$

สำหรับ $r = 1,2,\dots,2t.$

$\delta_r = 1$ ถ้าหาก $\Delta_r \neq 0$ และ $2L_{r-1} \leq r-1$; นอกจากนั้น $\delta_r = 0. If\ \delta_r = 0,$

$$u_r = u_{r-1} + 1; \text{นอกจากนั้น } u_r = 1. \tag{2.71}$$

$$e_i = \frac{\omega_i^{2t}}{\zeta_i^{2t}} \text{ เมื่อ } \lambda_i = 0 \tag{2.72}$$

จากอัลกอริทึมที่บนโดเมนเวลาที่ได้นั้น สามารถพัฒนาเป็นอัลกอริทึมสำหรับไปป์ไลน์เต็มรูปแบบได้ดังนี้

กำหนดค่าเริ่มต้น $\lambda_i^0 = \gamma_i^0 = \omega_i^0 = 1;$

$\mu_i^0 = 0;$

$L_0 = 0;$

$K_0 = u_0 = 1;$

$$\Delta_r = \sum_{i=0}^{N-1} v_i \lambda_i^{r-1}\alpha^{i(r+t_0-1)} \tag{2.73}$$

$$L_r = \delta_r(r - L_{r-1}) + (1-\delta_r)L_{r-1} \tag{2.74}$$

$$\mathrm{K}_r = \delta_r \Delta_r^{-1} + (1-\delta_r)\mathrm{K}_{r-1} \tag{2.75}$$

$$\begin{bmatrix}\lambda_i^r\\ \gamma_i^r\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r K_{r-1}\alpha^{-iu_r}\\ \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\lambda_i^{r-1}\\ \gamma_i^{r-1}\end{bmatrix} \tag{2.76}$$

$$\begin{bmatrix}\omega_i^r\\ \mu_i^r\end{bmatrix}=\begin{bmatrix}1 & -\Delta_r K_{r-1}\alpha^{-iu_r}\\ \delta_r & (1-\delta_r)\end{bmatrix}\begin{bmatrix}\omega_i^{r-1}\\ \mu_i^{r-1}\end{bmatrix} \tag{2.77}$$

สำหรับ $r = 1,2,\dots,2t.$

$\delta_r = 1$ ถ้า $\Delta_r \neq 0$ และ $2\mathrm{L}_{r-1} \leq \mathrm{r}-1$; นอกจากนั้น $\delta_\mathrm{r} = 0. \mathrm{If}\ \delta_\mathrm{r} = 0,$

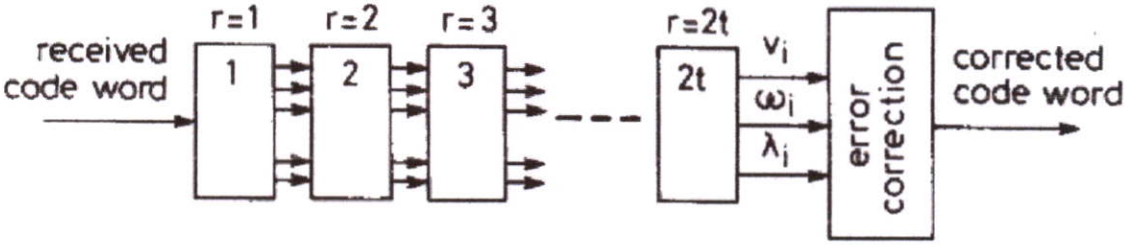
$$u_r = u_{r-1} + 1; \text{นอกจากนั้น } u_r = 1.$$

$$\zeta_i = \alpha^{-i(t_o-1)} \textstyle\sum_{k=0}^{N-1} h_k \lambda_{\langle i-k\rangle N} \text{ เมื่อ } \lambda_i = 0$$

$$\langle\cdot\rangle_N \textit{ denotes modulo } N, \textit{and } h_k = \alpha^{-k} + \alpha^{-3k} + \cdots + \alpha^{-k(N-2)}. \textit{Then } e_i = \frac{\omega_i^{2t}}{\zeta_i}$$

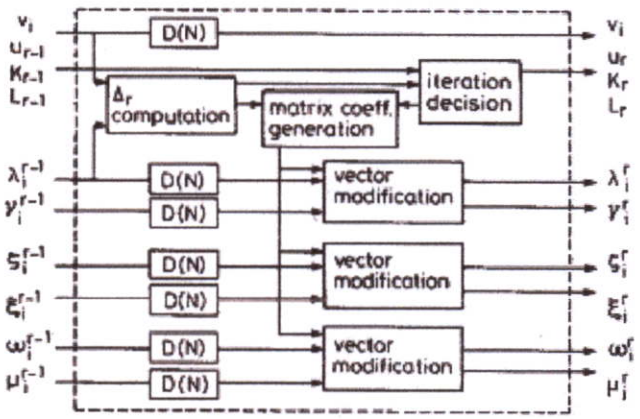
2.9.2การออกแบบสถาปัตยกรรมฮาร์ดแวร์

ใน [16] ได้นำเสนอสถาปัตยกรรมฮาร์ดแวร์ ของการถอดรหัสบนโดเมนเวลาแบบไปป์ไลน์ไว้ดังนี้



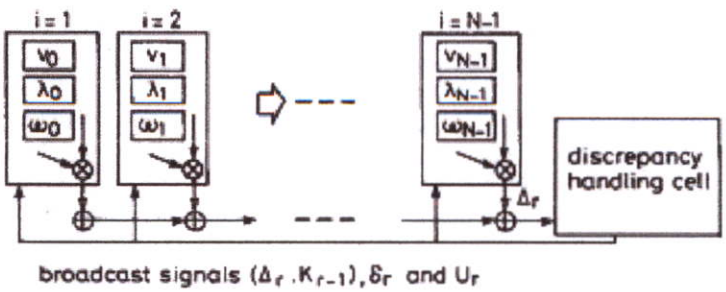
รูปที่ 2.9 แผนภาพแสดงสถาปัตยกรรมการถอดรหัสแบบไปป์ไลน์[16]

จากแผนภาพในรูปที่ 9 จะมีการรับอินพุตทุกอย่างเข้ามาจากทางซ้ายมือ และในทุกรอบการประมวลผลนั้นจะเริ่มจากการคำนวณค่า Δ_r จากสมการ (2.73) ซึ่งต้องการการคูณทั้งหมด 2N ครั้ง ถ้าตัวคูณ 2 ตัวถูกใช้งานแล้ว ค่า Δ_r จะถูกต้องเมื่อทำงานครบ N รอบ ดังนั้นค่า Δ_r จะต้องถูกเปลี่ยนไปเรื่อยๆจนครบ N รอบจึงจะได้ค่าที่ถูกต้อง ดังนั้นทุกอินพุต จะต้องถูกป้อนเค้าไปในรีจิสเตอร์เพื่อถ่วงเวลาให้ Δ_r คำนวณเสร็จก่อนจำนวน N สัญญาณนาฬิกา ซึ่งก็คือ D(N) ในรูปที่ 10



รูปที่ 2.10 แผนภาพแสดงฟังก์ชันการทำงานในแต่ละรอบการคำนวณ[16]

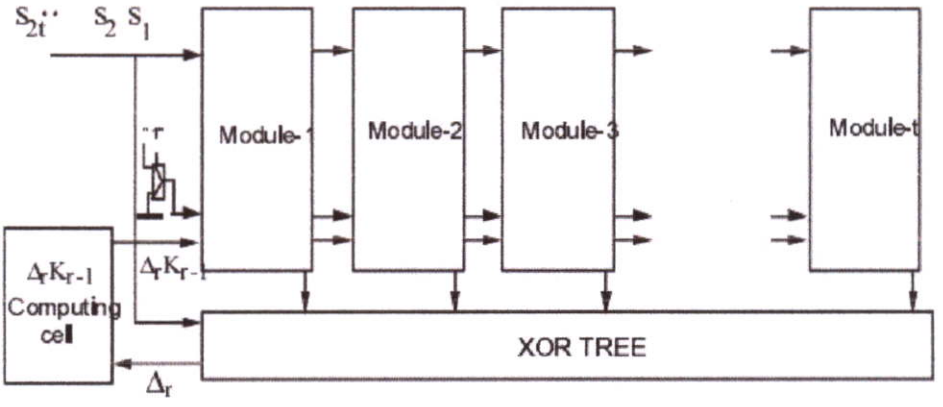
นอกจากไปป์ไลน์แล้ว ยังมีการนำเสนอสถาปัตยกรรมแบบขนาน ซึ่งมีลักษณะดังนี้



รูปที่ 2.11 แผนภาพสถาปัตยกรรมการถอดรหัสแบบขนาน[16]

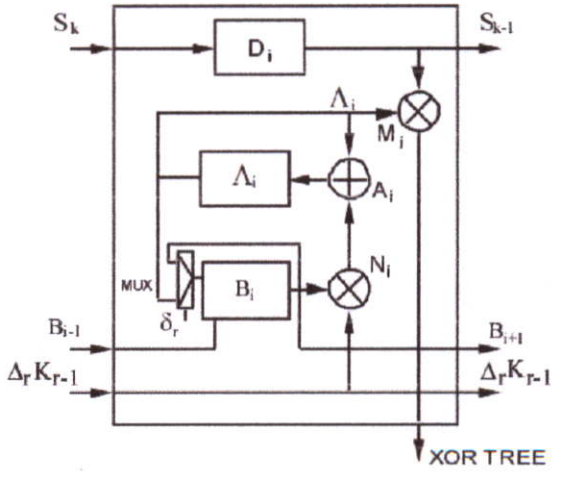
จากภาพจะเห็นว่ามีเซลล์ประมวลผลทั้งหมด N เซลล์ ซึ่งในแต่ละเซลล์นั้นอิมพลิเมนต์มาจากสมการของเบอร์ลิแคมป์ – แมสซีย์ ซึ่งจะทำการอัพเดตค่า L_r, K_r, μ_r และ δ_r ซึ่งจะทำงานทั้งหมด 2t รอบ ต่อคำรหัส 1 บล็อกอย่างไรก็ตาม การที่ข้อมูลจะต้องมีการ XOR ทั้งหมดนั้น จำเป็นจะต้องใช้ความยาวสัญญาณนาฬิกาที่มากกว่าการถอดรหัสแบบไปป์ไลน์ แต่ใช้หน่วยความจำน้อยกว่าน้อยกว่าแบบไปป์ไลน์

ใน [16] ได้นำเสนอการพัฒนาออกแบบสถาปัตยกรรมขนาน สำหรับการถอดรหัสบนโดเมนความถี่ ไว้ดังนี้



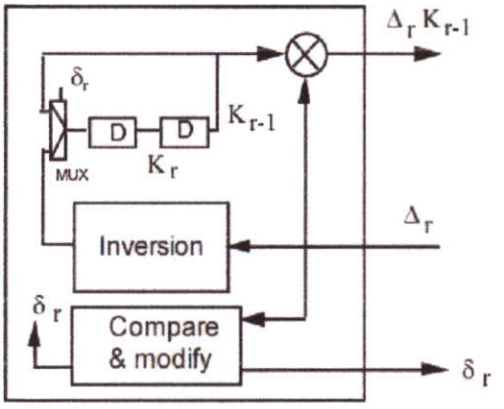
รูปที่ 2.12 แผนภาพแสดงการคำนวนอัลกอริทึมเบอร์ลิแคมป์ – แมสซีย์แบบไปป์ไลน์[16]

จากรูปจะเห็นได้ว่าโมดูลการคำนวณนี้มีทั้งหมด t โมดูลซึ่งขนานกันอยู่ เมื่อในแต่ละบล็อกคำนวณเสร็จแล้ว จะส่งเอาท์พุตออกมาที่ XOR TREE ทำให้ได้ค่า Δ_r แล้วส่งไปเข้าโมดูลคำนวณ $\Delta_r K_{r-1}$ เพื่อคำนวณในรอบถัดไป ซึ่งแต่ละโมดูลมีโครงสร้างภายในดังนี้



รูปที่ 2.13 โครงสร้างภายในของโมดูลคำนวณสำหรับสถาปัตยกรรมขนาน[16]

และเซลล์ที่ใช้คำนวณค่า $\Delta_r K_{r-1}$ มีโครงสร้างภายในดังนี้



รูปที่ 2.14 โครงสร้างภายในโมดูลคำนวณ $\Delta_r K_{r-1}$[16]

อีกทั้งยังได้สรุปเปรียบเทียบ จำนวนเกต ความยาวสัญญาณนาฬิกา ระหว่างการถอดรหัสแบบไปป์ไลน์ และขนาน ในรหัสที่ต่างกัน ดังตารางที่ 4

ตารางที่ 2.4 การเปรียบเทียบจำนวนเกต ความยาวสัญญาณ และรอบคำนวณ[16]

รหัส	สถาปัตยกรรมไปป์ไลน์			สถาปัตยกรรมขนาน		
	จำนวนเกต	ความยาวสัญญาณ	รอบคำนวณ	จำนวนเกต	ความยาวสัญญาณ	รอบคำนวณ
RS(15,11)	3,584	4.5	2	828	5.35	12
RS(15,9)	6,216	4.5	3	1,112	6.20	18
RS(31,15)	35,520	5.14	8	3,390	7.69	48
RS(255,239)	60,672	7.06	8	6,504	9.61	48
RS(255,223)	193,024	7.06	16	12,328	10.46	96

บทที่ 3

อัลกอริทึมบนโดเมนเวลา

ในบทนี้จะกล่าวถึงอัลกอริทึมการถอดรหัสบนโดเมนความถี่โดยวิธีการของเบอร์ลิแคมป์ – แมสซีย์ และอธิบายการแปลงกลับฟูริเยร์ เพื่อเข้าสู่สมการของอัลกอริทึมบนโดเมนเวลา และอธิบายในรูปแบบของแผนภาพกระแสข้อมูล

3.1 อัลกอริทึมบนโดเมนความถี่

สำหรับอัลกอริทึมเบอร์ลิแคมป์ – แมสซีย์ บนโดเมนความถี่นั้นสามารถเลือกกระทำได้ 2 แบบ คือแบบมาตรฐาน และแบบสมบูรณ์ กระทำได้ดังนี้

3.1.1 วิธีการเบอร์ลิแคมป์ – แมสซีย์ฉบับมาตรฐาน

1) คำนวณซินโดรม จากสมการที่ (2.50)

2) กำหนดค่าเริ่มต้น : $\Lambda^{(0)}(x) = B^{(0)}(x) = 1$ และ $L_0 = 0$ (ในที่นี้ $\Lambda^{(r)}(x)$ คือ รีจิสเตอร์พหุนามบอกตำแหน่ง (Error Locator Polynomial), $B^{(r)}(x)$ คือ รีจิสเตอร์สำรอง (Auxiliary Register), r คือ รอบการคำนวณ และ L คือ ความยาวของรีจิสเตอร์) ซึ่งการคำนวณนี้จะดำเนินไป $2t$ รอบ แล้วจะทำให้ได้พหุนามบอกตำแหน่งที่มี $\Lambda_0^{2t} = 1$ โดยแต่ละรอบการคำนวณ

3) ทดสอบว่า สำหรับค่าของพหุนามในปัจจุบันให้ค่าการคำนวณซินโดรมค่าต่อไปที่ถูกต้องหรือไม่ โดยการหาส่วนต่าง (Discrepancy: Δ_r) ดังสมการ

$$\Delta_r = \sum_{j=0}^{L} \Lambda_j^{(r-1)} S_{r-j}$$

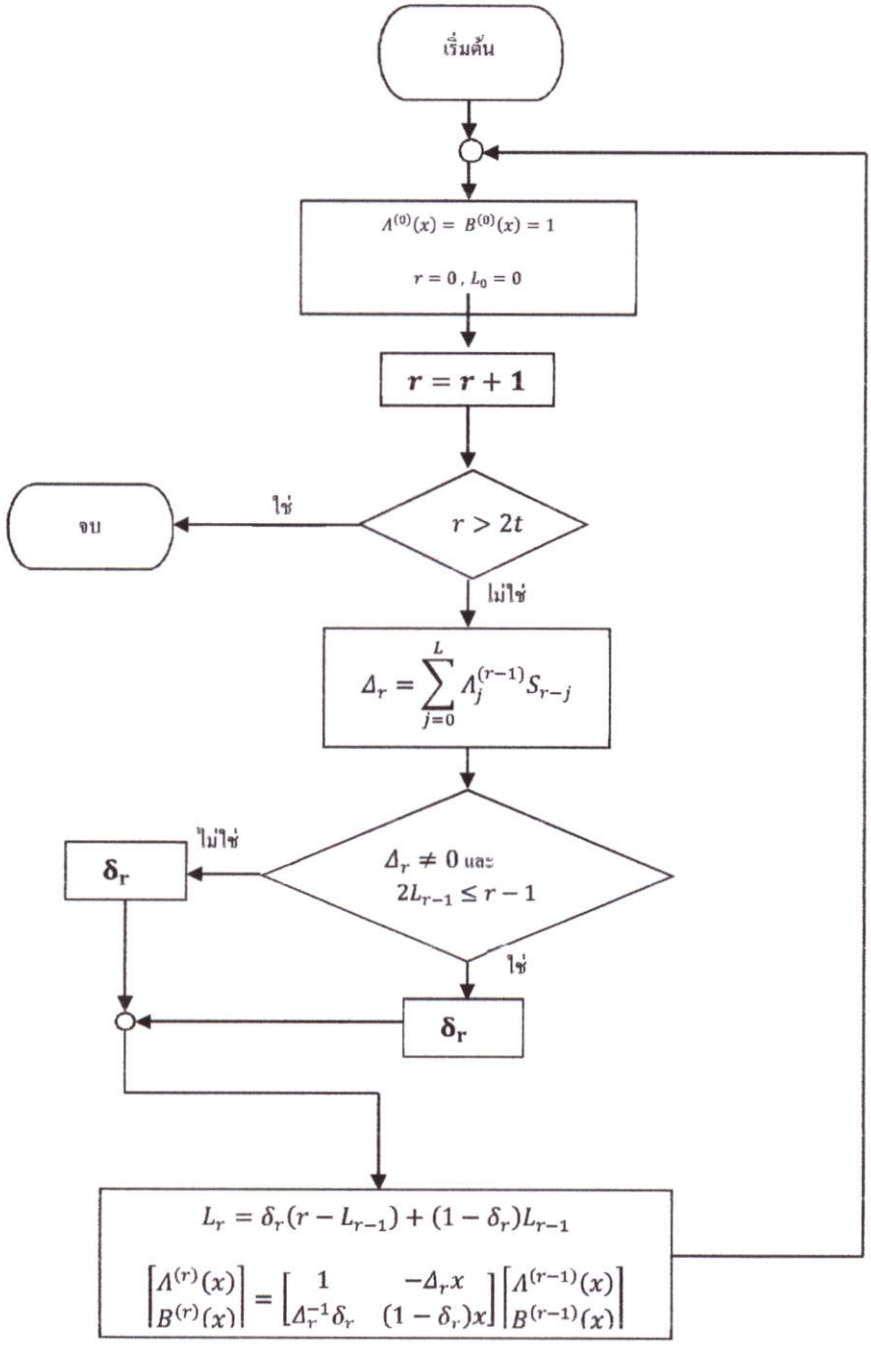
4) ถ้าหากยังมีความคลาดเคลื่อน คือ $\Delta_r \neq 0$ หรือความยาวของรีจิสเตอร์ไม่เหมาะสม คือ $2L_{r-1} \leq r-1$ ก็จะมีการเพิ่มความยาวรีจิสเตอร์ขึ้น และถ้าหากไม่มีความคลาดเคลื่อน ก็จะไม่มีการปรับค่าใดๆ เพื่อความสะดวกจะมีการกำหนดตัวทำการปรับปรุงคือ δ_r โดยให้ $\delta_r = 1$ ถ้าหาก $\Delta_r \neq 0$ และ $2L_{r-1} \leq r-1$ นอกจากนั้น $\delta_r = 0$ ดังนั้น ความยาวของรีจิสเตอร์จึงเขียนได้เป็น

$$L_r = \delta_r(r - L_{r-1}) + (1 - \delta_r)L_{r-1} \tag{3.1}$$

5) ทำการปรับปรุงรีจิสเตอร์หลัง และรีจิสเตอร์สำรอง ดังสมการ

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix} \tag{3.2}$$

6) ทำ 2) – 5) ซ้ำทั้งหมด 2t รอบ (r = 2t) จะได้ค่าใน รีจิเตอร์ Λ ซึ่งก็คือค่าสัมประสิทธิ์ของ $\Lambda(x)$ นั่นเอง ซึ่งสามารถเขียนเป็นผังงานได้ดังนี้



รูปที่ 3.1 ผังงานแสดงการทำงานของ อัลกอริทึมเบอร์ลิแคม – แมสเซย์ มาตรฐาน

7) คำนวณราก (root) ของพหุนามบอกตำแหน่งผิดพลาด ด้วยวิธีการ Chein Search ซึ่งจะหารากของ $\Lambda^{(2t)}(x)$ ตั้งแต่ $\Lambda(\alpha)$ - $\Lambda(\alpha^{m-2})$ แล้วหาส่วนกลับของราก เลขยกกำลังของส่วนกลับของรากนั้นจะบอกตำแหน่งที่ผิดพลาด

8) คำนวณหาขนาดความผิดพลาดโดยวิธีการฟอร์เนย์ (Forney Algorithm)[13] เมื่อทราบตำแหน่งที่ผิดพลาดแล้ว เหลือเพียงการหาขนาด โดยการเพิ่มพหุนามประเมินความผิดพลาด (Error Evaluator Polynomial: $\Gamma(x)$) เข้ามา ซึ่งคำนวณได้จาก

$$\bar{\Gamma}(x) = \bar{\Gamma}(x)S(x) \; mod \; x^{2t} \tag{3.3}$$

9) จากพหุนามบอกตำแหน่ง และพหุนามประเมินความผิดพลาด ขนาดของความผิดพลาดจะหาได้จาก

$$e_i = \frac{\Gamma(\alpha^{-i})}{\Lambda'(\alpha^{-i})} \tag{3.4}$$

เมื่อ e_i คือ ขนาดของความผิดพลาด ณ ตำแหน่งที่ i , $\Lambda'(\alpha^{-i})$ เป็นอนุพันธ์ของ $\Lambda(x)$ และ

$$e(x) = \sum_{i=0}^{N-1} e_i x^i \tag{3.5}$$

10) นำตำแหน่งและขนาดที่ได้ ไปคำนวณข้อมูลที่ถูกต้อง ดังสมการ

$$m(x) = r(x) + e(x) \tag{3.6}$$

วิธีดังกล่าว จะเห็นว่า เมื่อคำนวณค่า Δ_r แล้ว จะต้องคำนวณค่า Δ_r^{-1} ในทันที หากต้องการถอดรหัสที่ต้องการความเร็วสูง อัลกอริทึมนี้ จึงไม่เหมาะที่จะทำมาใช้ เพราะต้องรอเวลาในการคำนวณ Δ_r^{-1} ดังนั้น เพื่อไม่ต้องรีบร้อนในการหา Δ_r^{-1} อัลกอริทึมนี้จะสามารถถูกดัดแปลง พร้อมกับการถอดรหัสที่สัญลักษณ์มีการลบเลือน (Erasure) ได้ดังนี้

3.1.2 การถอดรหัสด้วยวิธีเบอร์ลิแคมป์ – แมสเซย์ ฉบับสมบูรณ์

รหัสลบเลือน (Erasure) เป็นรูปแบบที่ข้อมูลสัญลักษณ์ถูกลบไป หรือกลายเป็นคำอื่น แต่ข้อมูลนั้นเราทราบตำแหน่งแน่ชัด ว่าลบเลือนที่ตำแหน่งใด กรณีเช่นนี้เราไม่ต้องหาตำแหน่งของความผิดพลาด และก็จะหาเพียงขนาดเท่านั้น รหัสรีด – โซโลมอน จะยังคงแก้ไขความผิดพลาด และความลบเลือนได้ หาก $\rho + 2v \leq n - k$ เมื่อ ρ และ v คือจำนวนสัญลักษณ์ลบเลือน และสัญลักษณ์ผิดพลาด ตามลำดับ ในกรณีเช่นนี้ รหัสรีด – โซโลมอน จะได้รับการผนวกพหุนามระบุตำแหน่งข้อมูลลบเลือน (Erasure Locator Polynomial: $\Psi(x)$) เข้าไป ซึ่งนิยามโดย

$$\Psi(x) = \prod_{l=1}^{\rho} (1 - x\alpha^{i_l}) \tag{3.7}$$

ขั้นตอนการถอดรหัส

1) คำนวนซินโดรม และกำหนดพหุนามซินโดรม

2) คำนวณพหุนามระบุตำแหน่งข้อมูลเลือน

3) หาค่าซินโดรมใหม่

4) หาค่าพหุนามระบุตำแหน่งผิดพลาด และลบเลือน (Error-Erasure Locator Polynomial: $\bar{\Lambda}(x)$) โดยอัลกอริทึมเบอร์ลิแคมป์ – แมสเซย์ ฉบับสมบูรณ์ ซึ่งเพิ่มเติมจากอัลกอริทึม เบอร์ลิแคมป์ – แมสเซย์มาตรฐาน ดังนี้

5) กำหนดค่าเริ่มต้น $\Lambda^{(0)}(x) = B^{(0)}(x) = 1$, $\Gamma^{(0)}(x) = M^{(0)}(x) = S(x)$ และ $K_\rho = \rho, L_\rho = \rho$

6) สำหรับรอบคำนวณซึ่ง $r \leq \rho$

$$\boldsymbol{\Delta_r = \alpha^{i_l}} \tag{3.8}$$

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - \Delta_r x B^{(r-1)}(x) \tag{3.9}$$

$$B^{(r)}(x) = \Lambda^{(r)}(x) \tag{3.10}$$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - \Delta_r x M^{(r-1)}(x) \tag{3.11}$$

$$M^{(r)}(x) = \Gamma^{(r)}(x) \tag{3.12}$$

7) สำหรับรอบคำนวณซึ่ง $\rho < r \leq 2t$

$$\boldsymbol{\Delta_r = \sum_{j=0}^{L} \Lambda_j^{(r-1)} S_{r-j}} \tag{3.13}$$

$\boldsymbol{\delta_r = 1}$ ถ้าหาก $\boldsymbol{\Delta_r \neq 0}$ และ $\boldsymbol{2L_{r-1} \leq r-1}$ นอกจากนั้น $\boldsymbol{\delta_r = 0}$

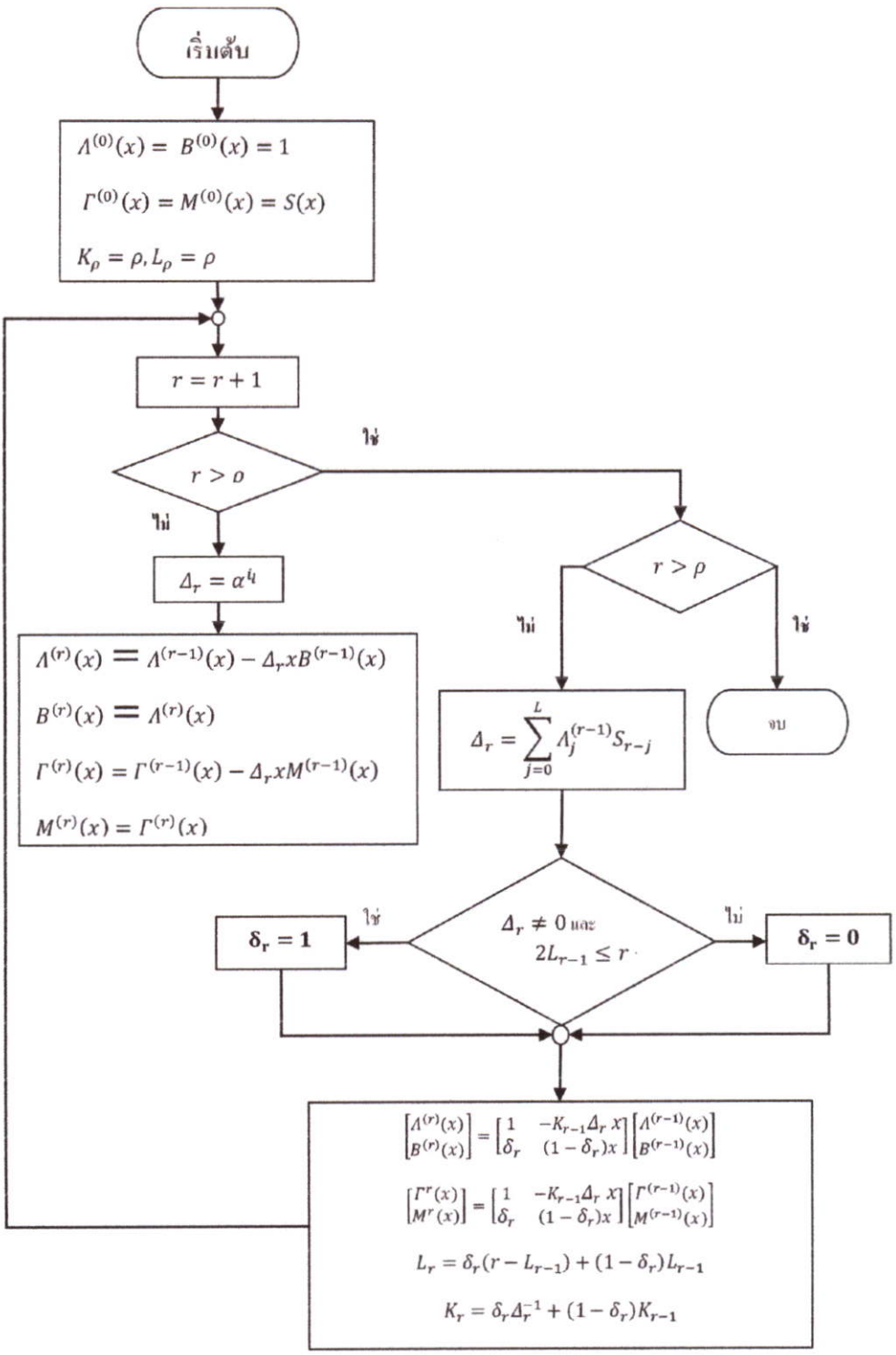
$$L_r = \delta_r(r - L_{r-1}) + (1 - \delta_r)L_{r-1} \tag{3.14}$$

$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r)K_{r-1} \tag{3.15}$$

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -K_{r-1}\Delta_r\, x \\ \delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix} \tag{3.16}$$

$$\begin{bmatrix} \Gamma^{r}(x) \\ M^{r}(x) \end{bmatrix} = \begin{bmatrix} 1 & -K_{r-1}\Delta_r\, x \\ \delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} \Gamma^{(r-1)}(x) \\ M^{(r-1)}(x) \end{bmatrix} \tag{3.17}$$

อัลกอริทึมดังกล่าว เขียนเป็นผังงานได้ดังนี้



รูปที่ 3.2 ผังงานแสดงอัลกอริทึมเบอร์ลิแคมป์ – แมสเซย์ฉบับสมบูรณ์

8) การอัลกอริทึมดังกลาวจะเห็นว่าการคำนวณนั้น ไม่รีบร้อนในการหาค่าส่วนกลับของผลต่างซินโดรม Δ_r^{-1} ดังที่กล่าวไว้ใน 1) โดยการเพิ่ม K เป็นตัวเก็บค่าดังกล่าวไว้ก่อนเพื่อนำไปใช้ในการคำนวณรอบหน้า

9) หาขนาดของความผิดพลาดด้วยอัลกอริทึมฟอร์เนย์

$$\bar{\Gamma}(x) = \bar{\Gamma}(x)S(x) \; mod \; x^{2t} \tag{3.18}$$

$$e_i = \frac{\bar{\Gamma}(x)}{\Lambda'(\alpha^{-i})} \tag{3.19}$$

3.2 อัลกอรึทึมบนโดเมนเวลา

การถอดรหัสบนโดเมนเวลานั้น สามารถแปลงจากสมการจากโดเมนความถี่ เพื่อให้เป็นโดเมนเวลาได้ จากสมการ (2.98) ซึ่ง Δ_r คือ ค่าที่เกิดจากการคอนโวลูชัน (Convolution) ระหว่าง S_k และ Λ_k รอบที่ r ซึ่งจะสามารถแปลงให้อยู่ในรูปของการคูณ ระหว่าง v_i และ λ_i สำหรับโดเมนเวลาได้ โดยที่ S_1 คือ ส่วนประกอบการแปลงฟูริเยร์เต็มหน่วย (DFT) ที่ t_0 ของ v_i และ Δ_r คือ ส่วนประกอบการแปลงฟูริเยร์เต็มหน่วย ที่ $(r + t_0 - 1)$ ของลำดับผลคูณระหว่าง v_i และ λ_i ซึ่งจะสามารถพิสูจน์ Δ_r บนโดเมนเวลา ได้ดังนี้

$$\begin{aligned}
\Delta_r &= \sum_{j=0}^{N-1} \Lambda_j^{(r-1)} S_{r-j} \\
&= \sum_{j=0}^{N-1} \Lambda_j^{(r-1)} \sum_{j=0}^{N-1} v_i \alpha^{i(r-j+t_0-1)} \\
&= \sum_{j=0}^{N-1} v_i \alpha^{i(r-j+t_0-1)} \sum_{j=0}^{N-1} \Lambda_j^{(r-1)} \alpha^{ij} \\
&= \sum_{j=0}^{N-1} \lambda_{r-1} v_i \alpha^{i(r-j+t_0-1)}
\end{aligned} \tag{3.20}$$

และจากอัลกอริทึมการถอดรหัสในโดเมนความถี่นั้น จะเห็นได้ว่าขนาดของความผิดพลาดนั้นจะถูกนำมาคำนวณแยกจากขั้นตอนการคำนวณพหุนามระบุตำแหน่ง จากสมการที่ (3.2) สามารถนำมาเขียนในอีกรูปแบบหนึ่งได้ดังนี้

$$e_i = \frac{\Omega(x)}{x\Lambda'(x)}, \qquad at \; x = \alpha^{-i} \tag{3.21}$$

เมื่อ

$$\Omega(x) = (S_1 x + S_2 x^2 + \cdots + s_{2t} x^{2t}) \Lambda(x) mod\ x^{2t} \tag{3.22}$$

และ

$$S_k = \sum_{i=0}^{N-1} v_i \alpha^{i(k+t_0-1)}, \quad for\ k = 1,2,\ldots,2t \tag{3.23}$$

และจากสมการ (3.2) นั้น เราสามารถเขียนใหม่คือ Z(x) แทน xΛ′(x) และ $\Xi(x)$ แทน $xB'(x)$ ดังนั้น อัลกอริทึมการถอดรหัส สามารถนำมาเขียนใหม่ ได้ดังนี้

Initializations: $\Lambda^{(0)}(x) = B^{(0)}(x) = \Omega^0(x) = 1;\ Z^{(0)}(x) = \Xi^{(0)}(x) = M^{(0)}(x) = 0;$

$L_0 = 0;\ K_0 = u_0 = 1;$

$$\Delta_r = \sum_{j=0}^{N-1} \Lambda_j^{(r-1)} S_{r-j} \tag{3.24}$$

$$L_r = \delta_r (r - L_{r-1}) + (1 - \delta_r) L_{r-1} \tag{3.25}$$

$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r) K_{r-1} \tag{3.26}$$

$$\begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \\ Z^{(r)}(x) \\ \Xi^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r K_{r-1} x^{u_r} & 0 & 0 \\ \delta_r & (1-\delta_r) & 0 & 0 \\ 0 & -\Delta_r K_{r-1} x^{(u_r+1-t_0)} & 1 & -\Delta_r K_{r-1} x^{u_r} \\ 0 & 0 & \delta_r & (1-\delta_r) \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \\ Z^{(r-1)}(x) \\ \Xi^{(r-1)}(x) \end{bmatrix} \tag{3.27}$$

$$\begin{bmatrix} \Omega^{(r)}(x) \\ M^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r K_{r-1} x^{u_r} \\ \delta_r & (1-\delta_r) \end{bmatrix} \begin{bmatrix} \Omega^{(r)}(x) \\ M^{(r)}(x) \end{bmatrix} \tag{3.28}$$

เมื่อ $r = 1, 2, \cdots, 2t$

$\delta_r = 1$ ถ้า $\Delta_r \neq 0$ และ $2L_{r-1} \leq r - 1$ นอกจากนั้น $\delta_r = 0$

$u_r = u_{r-1} + 1$ ถ้า $\delta_r = 0$ นอกจากนั้น $u_r = 1$

และเมื่อครบ 2t รอบแล้วนั้น สามารถนำมาหาขนาดความผิดพลาดได้ดังนี้

$$e_i = \frac{\Omega^{(2t)}(x)}{Z^{(2t)}(x)}, \tag{3.29}$$

ที่ $x = \alpha^{-i}$ที่เป็นรากของ $\Lambda^{(2t)}(x)$

ดังนั้นจากสมการจากโดเมนความถี่นั้น สามารถแปลงได้เป็นสมการสำหรับโดเมนเวลาได้ดังนี้

$$\Delta_r = \sum_{j=0}^{N-1} \lambda_{r-1} v_i \alpha^{i(r-j+t_0-1)} \tag{3.30}$$

$$L_r = \delta_r(r - L_{r-1}) + (1-\delta_r)L_{r-1} \tag{3.31}$$

$$K_r = \delta_r \Delta_r^{-1} + (1-\delta_r)K_{r-1} \tag{3.32}$$

$$\begin{bmatrix} \lambda_i^{(r)} \\ \beta_i^{(r)} \\ \zeta_i^{(r)} \\ \xi_i^{(r)} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r K_{r-1}\alpha^{-iu_r} & 0 & 0 \\ \delta_r & (1-\delta_r) & 0 & 0 \\ 0 & -\Delta_r K_{r-1} u_r \alpha^{-i(u_r+1-t_0)} & 1 & -\Delta_r K_{r-1}\alpha^{-iu_r} \\ 0 & 0 & \delta_r & (1-\delta_r) \end{bmatrix} \begin{bmatrix} \lambda_i^{(r-1)} \\ \beta_i^{(r-1)} \\ \zeta_i^{(r-1)} \\ \xi_i^{(r-1)} \end{bmatrix} \tag{3.33}$$

$$\begin{bmatrix} \omega_i^{(r)} \\ \mu_i^{(r)} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r K_{r-1}\alpha^{-iu_r} \\ \delta_r & (1-\delta_r) \end{bmatrix} \begin{bmatrix} \omega_i^{(r-1)} \\ \mu_i^{(r-1)} \end{bmatrix} \tag{3.34}$$

เมื่อ $r = 1, 2, \cdots, 2t$

$\delta_r = 1$ ถ้า $\Delta_r \neq 0$ และ $2L_{r-1} \leq r-1$ นอกจากนั้น $\delta_r = 0$

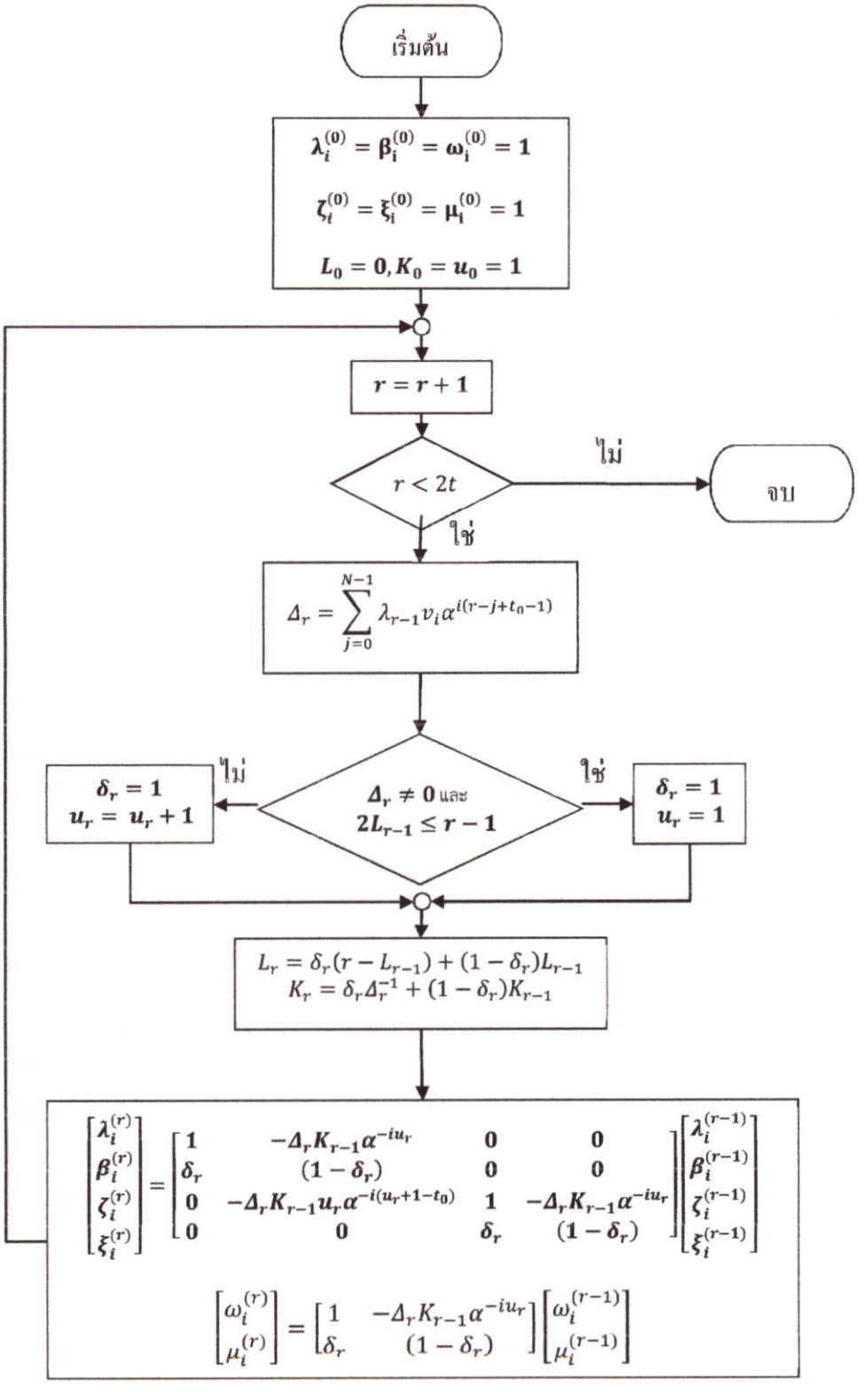
$u_r = u_{r-1} + 1$ ถ้า $\delta_r = 0$ นอกจากนั้น $u_r = 1$

จากนั้นนำ $\omega_i^{(r)}$และ $\zeta_i^{(r)}$ มาคำนวณหาขนาดของข้อมูลผิดพลาดได้ด้วยอัลกอริทึมฟอร์เนย์ ดังสมการ

$$e_i = \frac{\omega_i^{(2t)}}{\zeta_i^{2t)}} \tag{3.35}$$

ซึ่งสมการที่ (3.18) นี้จะใช้คำนวณเฉพาะตำแหน่งที่ $\lambda_i = 0$

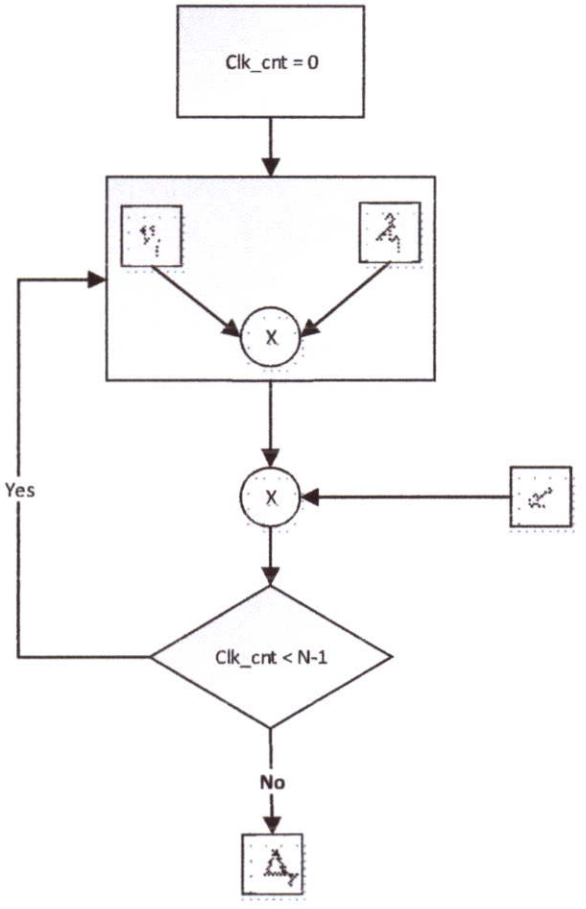
ดังนั้น การถอดรหัสบนโดเมนเวลา สามารถเขียนเป็นผังงานได้ ดังต่อไปนี้



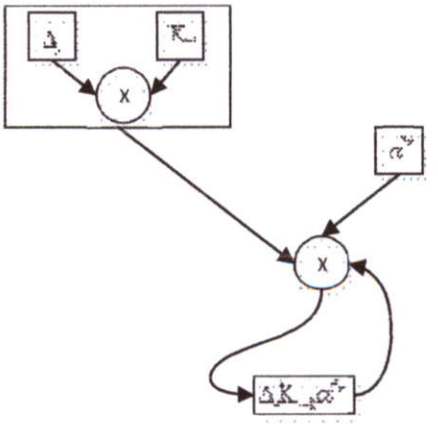
รูปที่ 3.3 ผังงานแสดงอัลกอริทึมการถอดรหัสบนโดเมนเวลา

3.3 แผนภาพกระแสข้อมูลของอัลกอริทึม

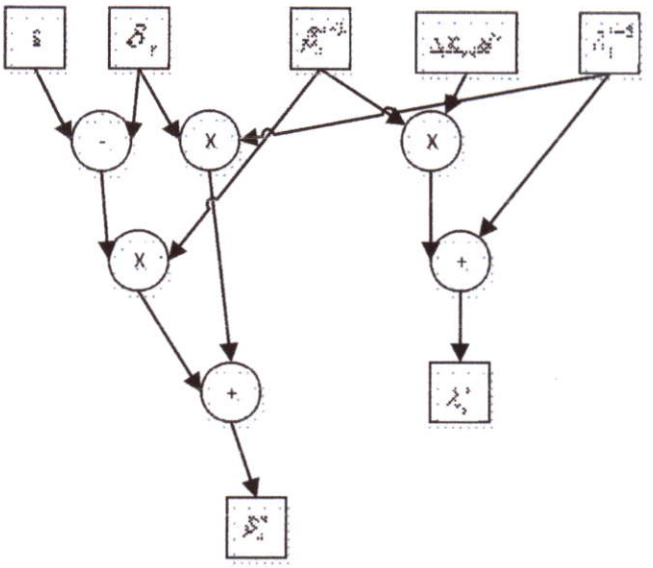
จากอัลกอริทึมการถอดรหัสบนโดเมนเวลา สามารถมาเขียนแผนภาพแสดงการคำนวณค่าต่าง ๆ ได้ดังนี้



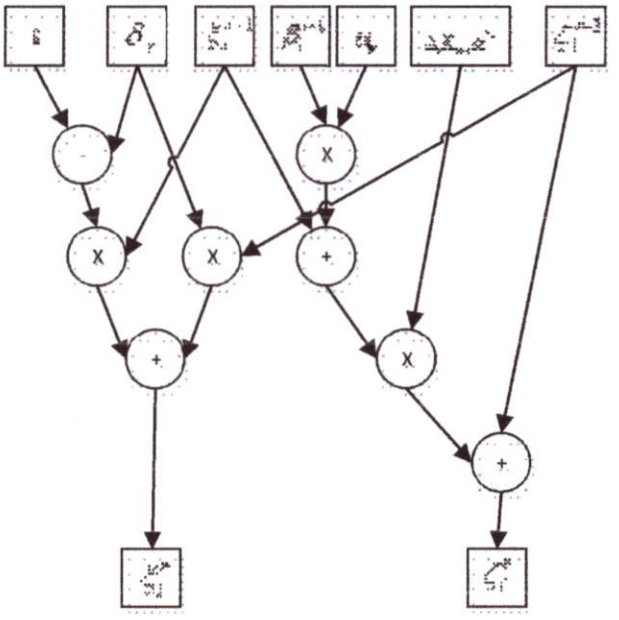
รูปที่ 3.4 แผนภาพแสดงการคำนวณ Δ_r



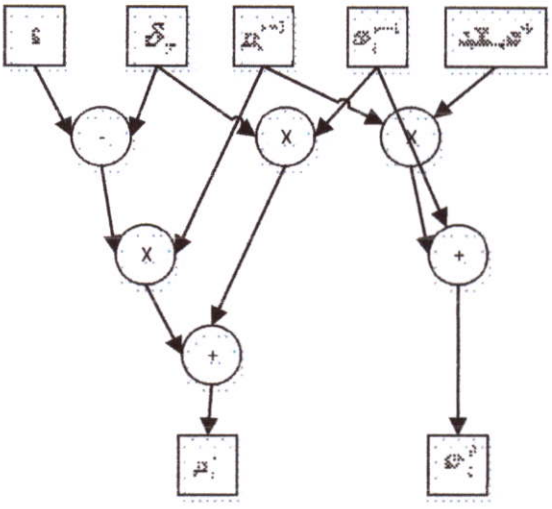
รูปที่ 3.5 แผนภาพแสดงการคำนวณค่า $\Delta_r \mathrm{K}_{r-1}\alpha^{-iu_r}$



รูปที่ 3.6 แผนภาพแสดงการคำนวณ λ_r และ β_r



รูปที่ 3.7 แผนภาพแสดงการคำนวณ ζ_r และ ξ_r



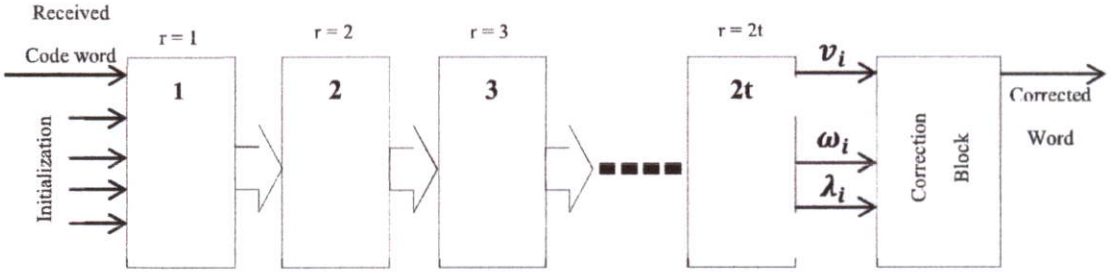
รูปที่ 3.8 แผนภาพแสดงการคำนวณค่า ω_r และ μ_r

บทที่ 4
การอิมพลิเมนต์อัลกอริทึมการถอดรหัสบนโดเมนเวลา

ในบทนี้ จะเป็นการออกแบบฮาร์ดแวร์ถอดรหัสรีด – โซโลมอนบนโดเมนเวลา หลังจากศึกษาอัลกอริทึมการถอดรหัสบนโดเมนเวลาแล้ว นำมาออกแบบฮาร์ดแวร์ จากนั้นทำการจำลองพฤติกรรมแล้วอิมพลิเมนต์ลงฮาร์ดแวร์ เพื่อทดสอบการถอดรหัส

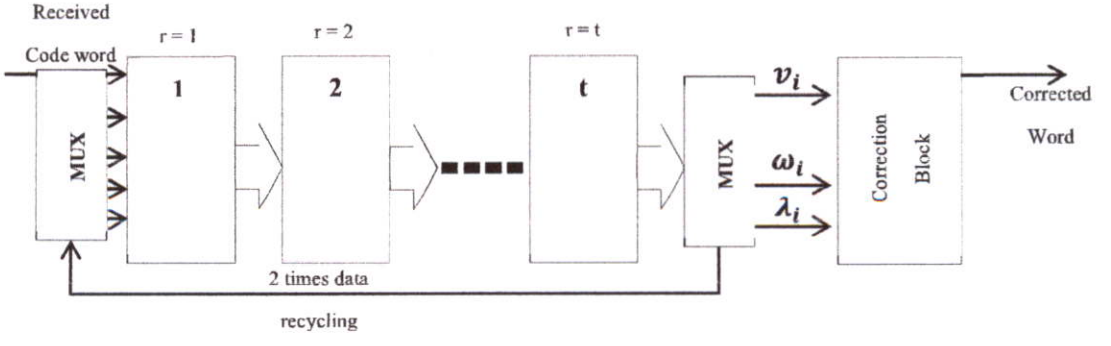
4.1 การแปลงอัลกอริทึมเป็นฮาร์ดแวร์

ระบบการถอดรหัสนั้น ได้เลือกการถอดรหัสบนโดเมนเวลา เพื่อความสะดวกในการปรับเปลี่ยนความเร็วในการทำงาน และการถอดรหัสแบบการทำงานแบบไปป์ไลน์ (Pipeline) นั้น สามารถเขียนเป็นแผนผังบล็อก ได้ดังนี้

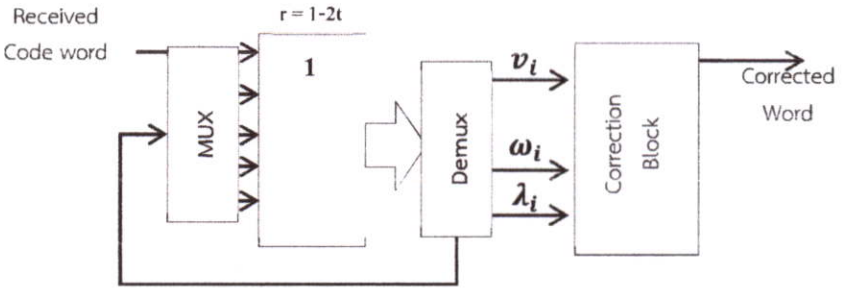


รูปที่ 4.1 แผนภาพบล็อกแสดงการถอดรหัสแบบไปป์ไลน์เป็นลำดับ 2t รอบ

นอกจากการทำงานแบบไปป์ไลน์แล้ว ระบบการถอดรหัสยังสามารถออกแบบให้เป็นแบบ t บล็อกแล้ววนรอบคำนวณบล็อกละสองรอบ หรือบล็อกเดียว แล้วคำนวณ 2t รอบ ซึ่งออกแบบระบบได้ดังนี้

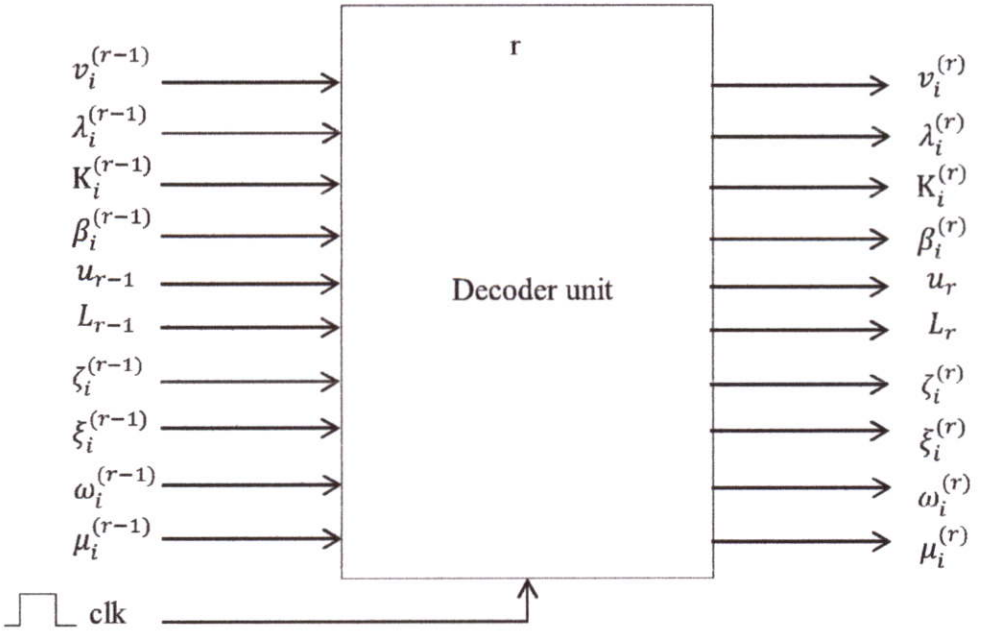


รูปที่ 4.2 แผนภาพบล็อกแสดงการถอดรหัสแบบไปป์ไลน์ 2t บล็อก และวนรอบ 2 รอบ



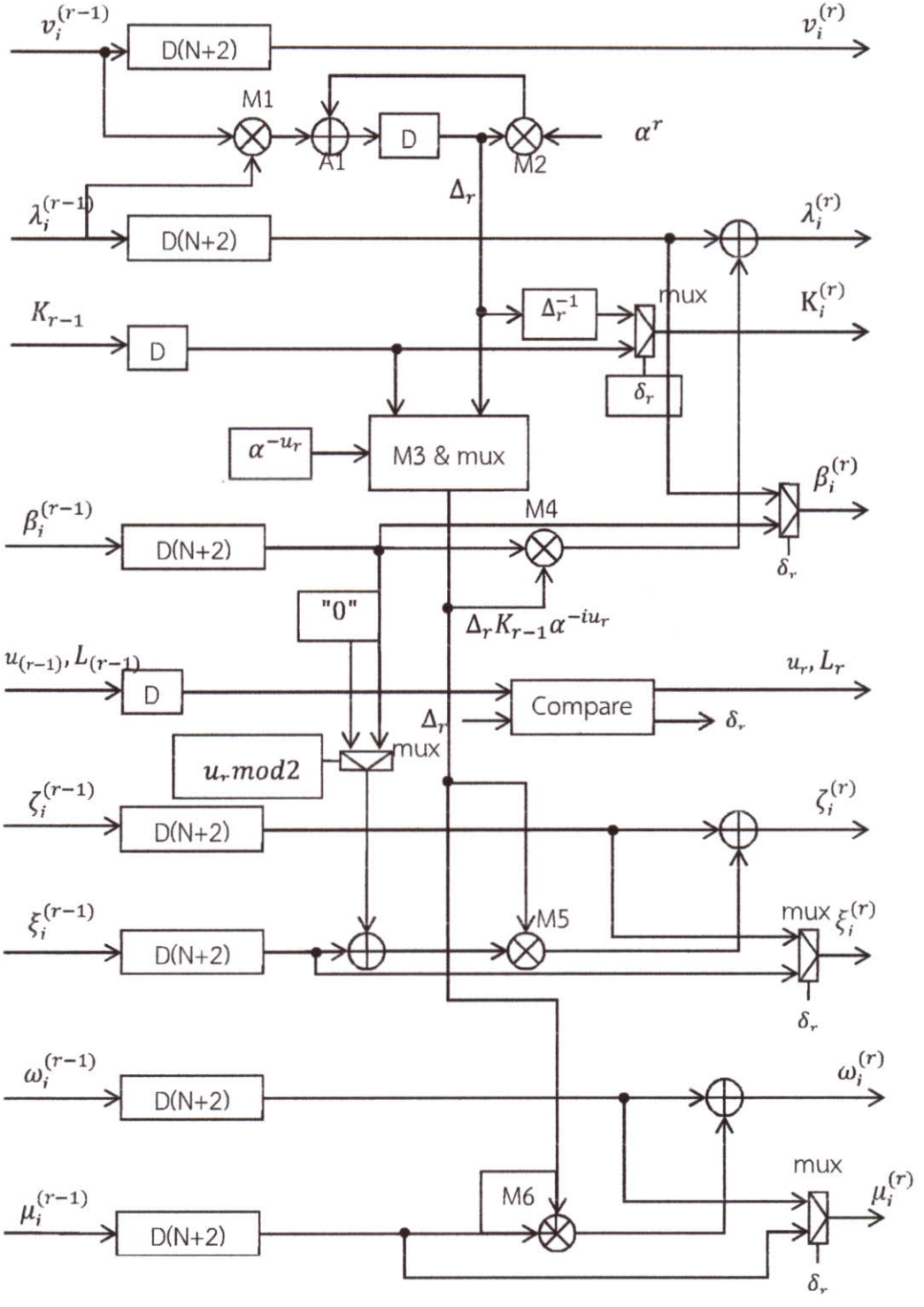
รูปที่ 4.3 แผนภาพบล็อกแสดงแบบบล็อกเดียววน 2t รอบ

จากรูปที่ 23 - 25 จะเห็นว่า มีบล็อกย่อยสำหรับถอดรหัสซึ่งจะมีการคำนวณเพื่อถอดรหัสทั้งหมด 2t รอบเท่ากัน แต่จะต่างกันในเรื่องของการใช้ทรัพยากร และในแต่ละบล็อกนั้น มีอินพุตและเอาท์พุตดังรูป



รูปที่ 4.4 อินพุต และเอาท์พุตสำหรับบล็อกย่อยการถอดรหัส

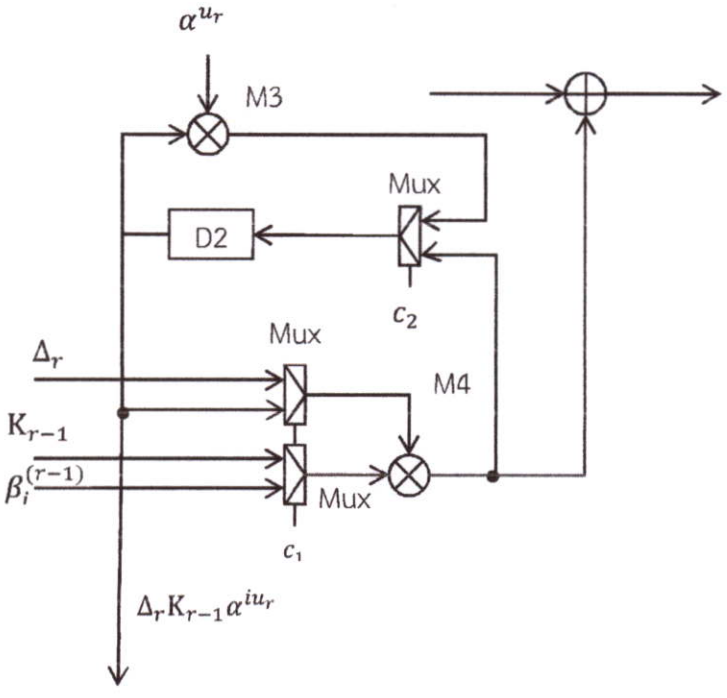
จากรูปที่ 23 - 25 แสดงให้เห็นว่าเมื่ออินพุตเข้ามาในแต่ละบล็อกย่อยแล้ว เมื่อประมวลผลเสร็จแล้ว เอาท์พุตที่ออกมานั้น จะส่งต่อเป็นอินพุตให้กับบล็อกต่อไป ในขณะเดียวกันก็จะรับชุดข้อมูลอินพุตชุดถัดไปเข้ามาเพื่อประมวลผลในทันที ซึ่งก็คือไปป์ไลน์นั่นเอง เมื่อครบ 2t รอบแล้ว จะนำข้อมูลที่ได้จากรอบที่ 2t คือ v_i^{2t}, λ_i^{2t} และ v_i^{2t} เข้าไปในบล็อกสุดท้าย (Correction Block) เพื่อแก้ไขข้อมูลให้ถูกต้อง ซึ่งสถาปัตยกรรมภายในของบล็อกถอดรหัสนั้น แสดงได้ดังนี้



รูปที่ 4.5 โครงสร้างภายในแต่ละบล็อกการถอดรหัส

กระบวนการถอดรหัสในแต่ละรอบ r นั้น เริ่มต้นทำงานโดยรับค่า v_i เข้ามา แล้วทำการคูณกับ λ_i ที่ M1 ในขณะที่ M2 ทำการคูณค่าที่ได้จาก ฟลิปฟล็อป แล้วนำผลคูณทั้งสองค่า ป้อนกลับไปยังฟลิปฟล็อป เพื่อจะนำผลไปคูณในสัญญาณนาฬิกาลูกต่อไป ทำซ้ำทั้งหมด N รอบ ก็จะได้ค่า Δ_r

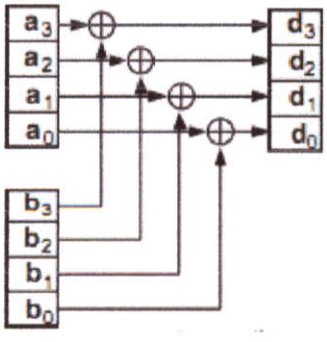
สำหรับวงจรที่จะใช้สร้างสัญญาณ $\Delta_r \mathrm{K}_{r-1}\alpha^{-iu_r}$ นั้น ซึ่งก็คือบล็อก M3 & mux ในรูปที่ 28 ประกอบไปด้วยวงจรคุณ M3 และมัลติเพล็กเซอร์ 3 หน่วยอยู่ภายในดังรูป



รูปที่ 4.6 แผนภาพแสดงวงจรการคำนวณ $\Delta_r \mathrm{K}_{r-1}\alpha^{-iu_r}$

จากรูปที่ 28 ในสัญญาณนาฬิกาสัญญาณแรกหลังจากได้ค่า Δ_r มาแล้ว จะถูกนำมาคำนวณค่า $\Delta_r \mathrm{K}_{r-1}$ ด้วย M4 แล้วจะถูกเก็บไว้ใน D2 ซึ่งจะถูกควบคุมการทำงานโดย Mux 1 – 3 และในสัญญาณนาฬิกาถัดๆมา จะนำค่า $\Delta_r \mathrm{K}_{r-1}$ ที่ได้แล้วมาคำนวณเป็นค่า $\Delta_r \mathrm{K}_{r-1}\alpha^{iu_r}$ ด้วย M3 และเพื่อนำไปใช้ในการคำนวณค่าอื่น ๆ ในวงจร ซึ่งในช่วงที่คำนวณตอนแรกนั้น จะปิดกั้นการรับข้อมูลอื่น ๆ ทั้งหมด เพื่อจะได้ข้อมูลที่ถูกต้องมาใช้ในการคำนวณต่อไป

จากรูปที่ 27 และ 28 จะเห็นได้ว่าในบล็อกคำนวณนั้น จะมีวงจรคำนวณสำคัญสองอย่างคือ วงจรบวก และวงจรคูณ วงจรบวกนั้นเป็นวงจร XOR อย่างง่ายดังรูปที่ 29



รูปที่ 4.7 วงจรบวก 4 บิต

สำหรับวงจรคูณนั้น จะได้จากการพิจารณาการคูณพหุนามบนสนามจำกัดกาลัวส์ ดังนี้

กำหนดพหุนามใน GF(16) ที่มีพหุนาม $I(x) = x^4 + x + 1$ ซึ่ง $A(x) = a_3x^3 + a_2x^2 + a_1x + a_0$, $B(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ และ $C(x) = c_3x^3 + c_2x^2 + c_1x + c_0$ โดยการคูณคือ

$$C(x) = \langle A(x) \cdot B(x) \rangle_{\mathrm{mod} I(x)}$$

$$= (a_3b_3 + a_0b_3 + a_1b_2 + a_3b_0)x^3 + (a_2b_3 + a_3b_2 + a_0b_2 + a_1b_1 + a_2b_0 + a_3b_3)x^2 +$$

$$(a_1b_3 + a_2b_2 + a_3b_1 + a_0b_1 + a_1b_0 + a_2b_3 + a_3b_2)x + (a_1b_3 + a_2b_2 + a_3b_1 + a_0b_0)$$

จะเห็นว่า

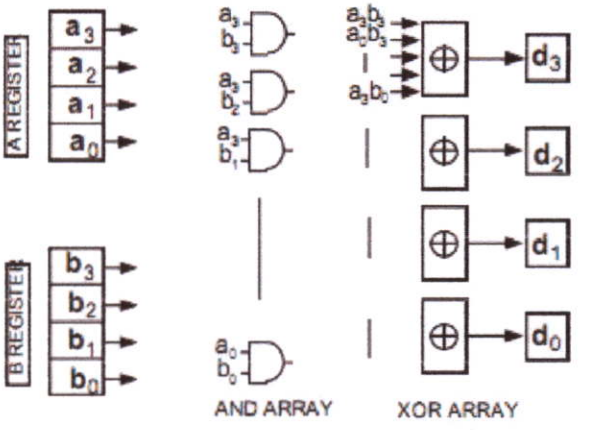
$c_3 = a_3b_3 + a_0b_3 + a_1b_2 + a_3b_0$

$c_2 = a_2b_3 + a_3b_2 + a_0b_2 + a_1b_1 + a_2b_0 + a_3b_3$

$c_1 = a_1b_3 + a_2b_2 + a_3b_1 + a_0b_1 + a_1b_0 + a_2b_3 + a_3b_2$

$c_0 = a_1b_3 + a_2b_2 + a_3b_1 + a_0b_0$

ซึ่งวงจรที่จะคำนวณผลคูณนี้จะประกอบด้วยอาเรย์ของแอนด์เกท และอาเรย์ของ XOR เกท



รูปที่ 4.8 วงจรคูณบน GF(16) ซึ่งมี $I(x) = x^4 + x + 1$

สำหรับวงจรย่อยต่อไปคือ วงจรคำนวณ Δ_r^{-1} ซึ่งจะกระทำได้โดยการเปิดตาราง (Lookup Table) เพื่อตรวจสอบค่าของ Δ_r^{-1} ในรอบการคำนวณนั้นๆ ซึ่งจะได้ตารางดังนี้

ตารางที่ 4.1 ค่า Δ_r^{-1} ที่ Δ_r ค่าต่างๆ

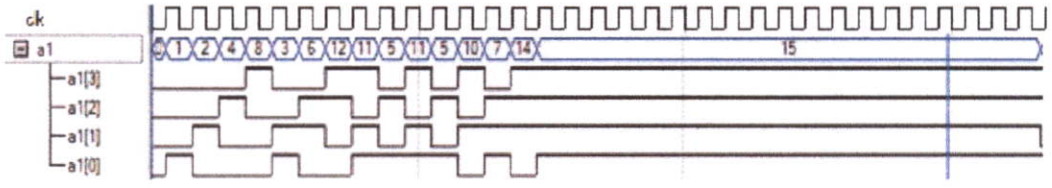
Δ_r	Δ_r^{-1}
0	0
1	1
2	9
3	14
4	13
5	11
6	7
7	6
8	15
9	2
10	12
11	5
12	10
13	4
14	3
15	8

4.2 การอิมพลิเมนต์ และตรวจสอบความถูกต้อง

4.2.1 ระบบถอดรหัส

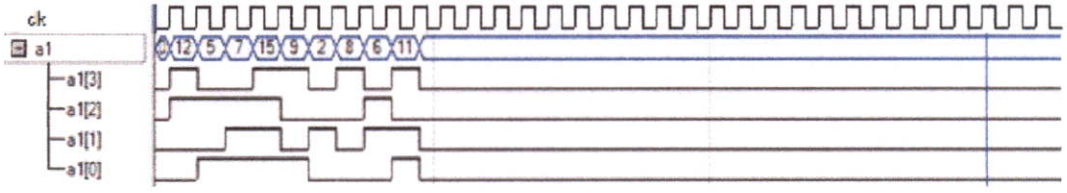
4.2.1.1 ทดสอบการคำนวณ Δ_r

จากการทดสอบการคำนวณในขั้นตอนก่อนหน้านี้ โดยใช้ชุดข้อมูลจากการเข้ารหัสคือ สำหรับ RS(15,11) คือ {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1} ซึ่งการคำนวณในแต่ละรอบนั้น จากการคำนวณด้วยโปรแกรมแมทแล็บได้ค่า $\Delta_r = 9$



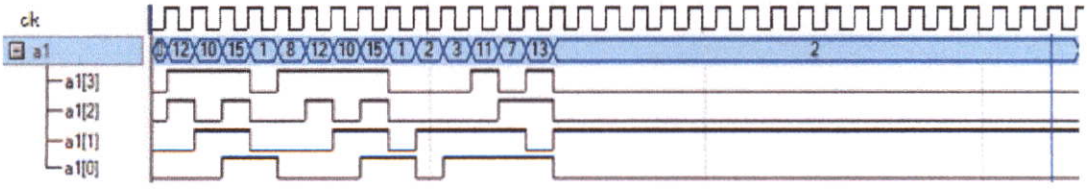
รูปที่ 4.9 ผลการทดสอบโมดูลคำนวณ Δ_r ในรอบที่ r = 1 สำหรับ RS(15,11)

สัญญาณ a1 ในรูปที่ 31 คือรีจิสเตอร์ที่ใช้เก็บค่าของ Δ_r เมื่อรับคำรหัสครบ N สัญลักษณ์ซึ่งในที่นี้ N = 15 แล้วการคำนวณ Δ_r นั้นได้ผลลัพธ์เท่ากับที่คำนวณในขั้นตอนการคำนวณด้วยโปรแกรมแมทแล็บ คือ 15



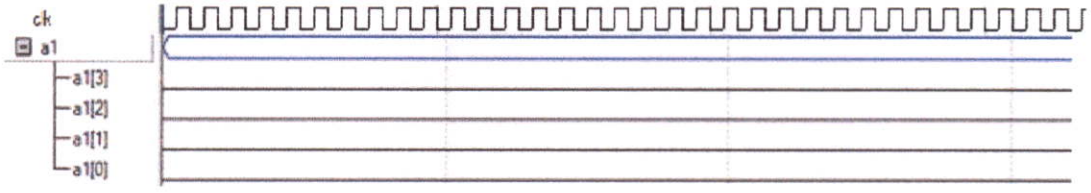
รูปที่ 4.10 ผลการทดสอบโมดูลคำนวณ Δ_r ในรอบที่ r = 2 สำหรับ RS(15,11)

สำหรับการคำนวณนั้นเมื่อนำไปเปรียบเทียบกับการคำนวณ จะเห็นว่าคำตอบที่ได้คือ 0 จะเห็นได้ว่า ผลการคำนวณนั้นถูกต้อง



รูปที่ 4.11 ผลการทดสอบโมดูลคำนวณ Δ_r ในรอบที่ r = 3 สำหรับ RS(15,11)

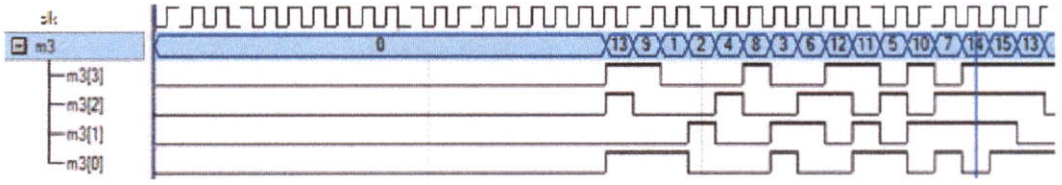
สำหรับการคำนวณนั้นเมื่อนำไปเปรียบเทียบกับการคำนวณ จะเห็นว่าคำตอบที่ได้คือ 2 จะเห็นได้ว่า ผลการคำนวณนั้นถูกต้อง



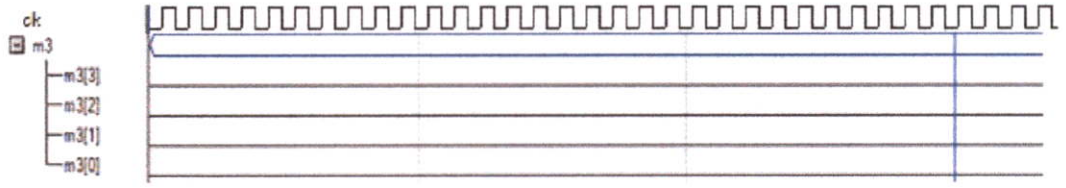
รูปที่ 4.12 ผลการทดสอบโมดูลคำนวณ Δ_r ในรอบที่ r = 4 สำหรับ RS(15,11)

สำหรับการคำนวณนั้นเมื่อนำไปเปรียบเทียบกับการคำนวณ จะเห็นว่าคำตอบที่ได้คือ 0 จะเห็นได้ว่า ผลการคำนวณนั้นถูกต้อง

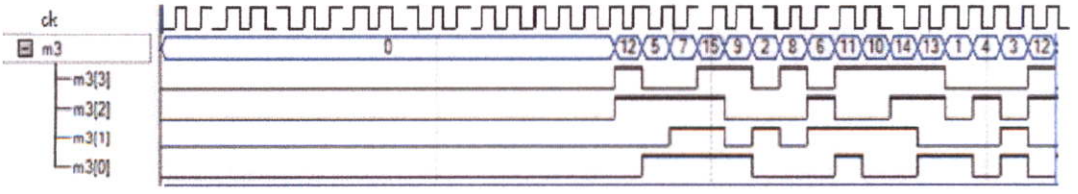
4.2.1.2 ทดสอบการคำนวณ $\Delta_r K_{r-1}\alpha^{-iu_r}$



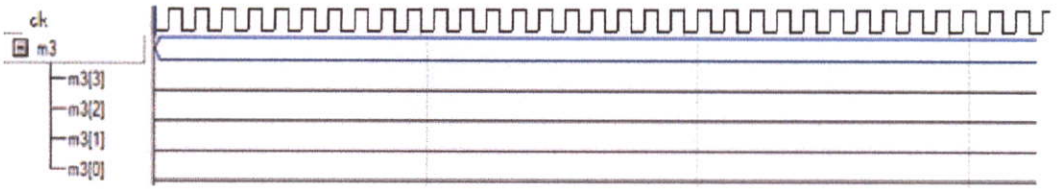
รูปที่ 4.13 ผลการทดสอบโมดูลคำนวณ $\Delta_r K_{r-1}\alpha^{-iu_r}$ ที่ r = 1



รูปที่ 4.14 ผลการทดสอบโมดูลคำนวณ $\Delta_r K_{r-1}\alpha^{-iu_r}$ ที่ r = 2



รูปที่ 4.15 ผลการทดสอบโมดูลคำนวณ $\Delta_r K_{r-1}\alpha^{-iu_r}$ ที่ r = 3



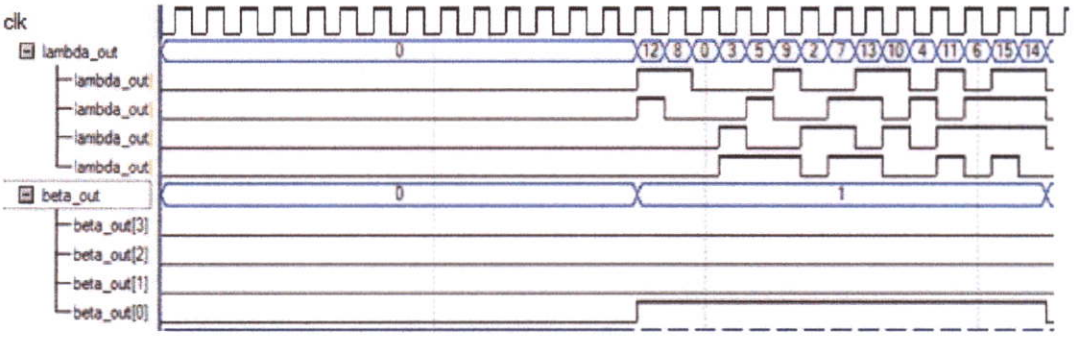
รูปที่ 4.16 ผลการทดสอบโมดูลคำนวณ $\Delta_r K_{r-1}\alpha^{-iu_r}$ ที่ r = 4

จากรูปที่ 33 - 36 ผลการทดสอบโมดูลคำนวณสัญญาณกลาง $\Delta_r K_{r-1}\alpha^{-iu_r}$ เริ่มที่สัญญาณนาฬิกาที่ N + 2 นั้นจะเป็นค่าของ $\Delta_r K_{r-1}$ เนื่องจากยังไม่มีการคูณค่า α^{-iu_r} เพิ่มเข้าไป หลังจากนั้นจึงจะเริ่มการคำนวณค่า $\Delta_r K_{r-1}\alpha^{-iu_r}$ โดยที่ i = N-1, N-2,...., 0

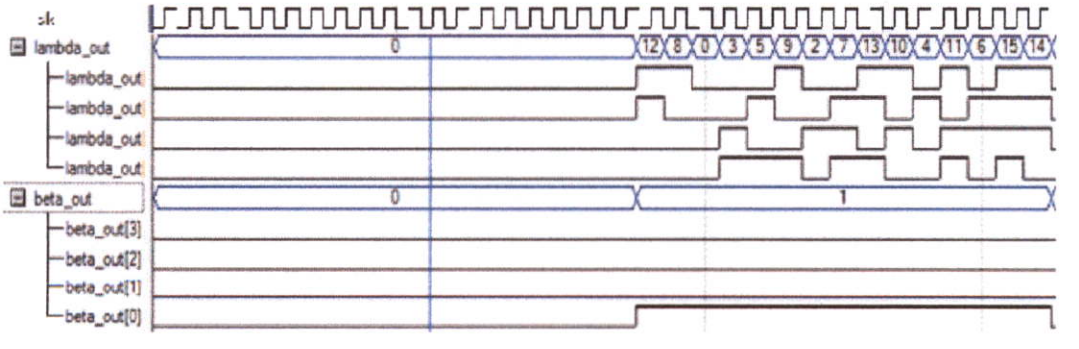
เมื่อได้สัญญาณกลาง $\Delta_r K_{r-1}\alpha^{-iu_r}$ แล้ว สามารถนำไปคำนวณหาค่า $\lambda, \beta, \zeta, \xi, \omega$ และ μ ได้ผลการคำนวณ ดังต่อไปนี้

4.2.1.3 ผลการทดสอบการคำนวณ λ และ β

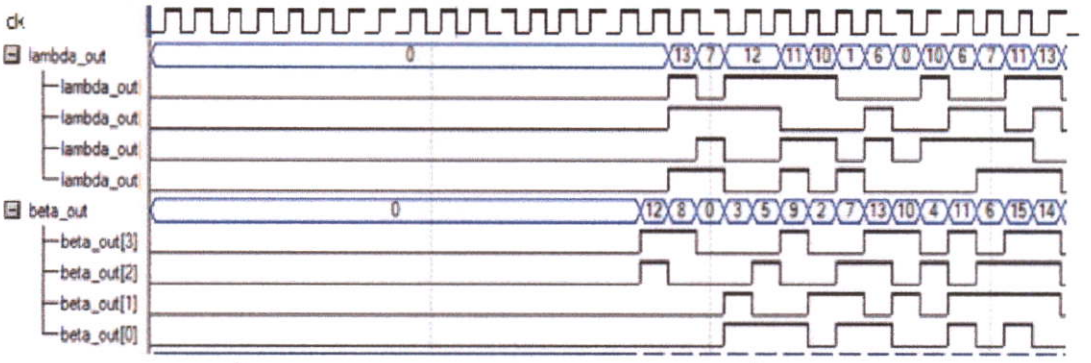
ในการถอดรหัสนั้น จำเป็นจะต้องมีพหุนามบอกตำแหน่งของข้อมูลผิดพลาดและรีจิสเตอร์สำรองของพหุนามบอกตำแหน่งนี้ ซึ่งก็คือค่าของ λ และ β ตามลำดับ



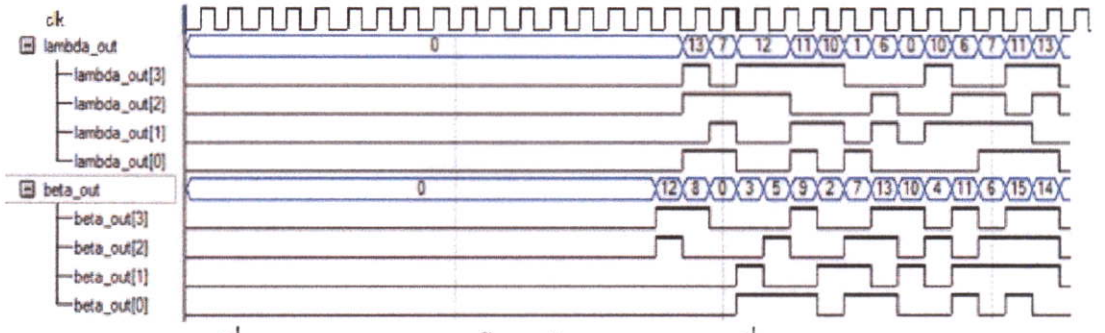
รูปที่ 4.17 ผลการทดสอบโมดูลคำนวณ λ และ β ที่ r = 1



รูปที่ 4.18 ผลการทดสอบโมดูลคำนวณ λ และ β ที่ r = 2



รูปที่ 4.19 ผลการทดสอบโมดูลคำนวณ λ และ β ที่ r = 3



รูปที่ 4.20 ผลการทดสอบโมดูลคำนวณ λ และ β ที่ r = 4

จากรูปที่ 39 – 42 แสดงให้เห็นผลการคำนวณ λ และ β จากการป้อนคำรหัสตัวอย่าง RS(15,11) คือ {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1} และผลลัพธ์ที่ได้ คือ

1) r = 1

λ = {12,8,0,3,5,9,2,7,13,10,4,11,6,15,14}

β = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

2) r = 2

λ = {12,8,0,3,5,9,2,7,13,10,4,11,6,15,14}

β = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

3) r = 3

λ = {0,13,7,12,12,11,10,1,6,0,10,6,7,11,13}

β = {12,8,0,3,5,9,2,7,13,10,4,11,6,15,14}

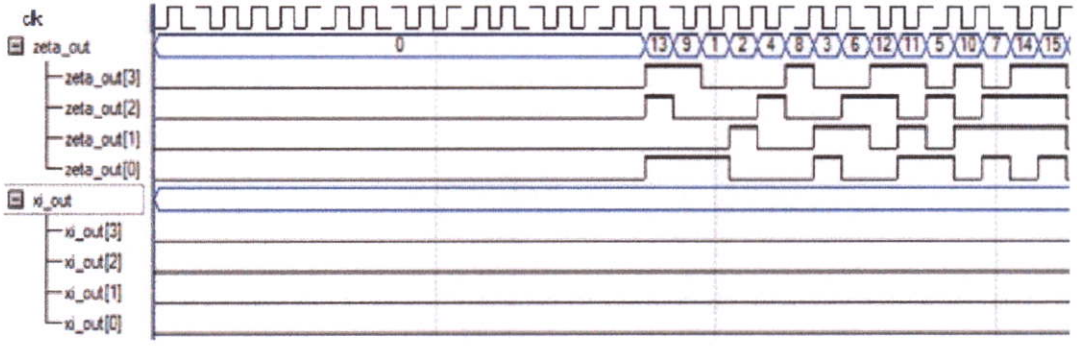
4) r = 4

λ = {0,13,7,12,12,11,10,1,6,0,10,6,7,11,13}

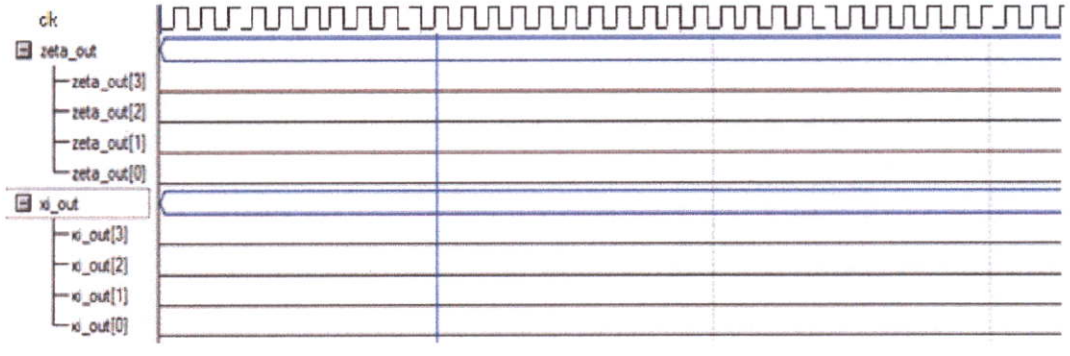
β = {12,8,0,3,5,9,2,7,13,10,4,11,6,15,14}

4.2.1.4 ผลการทดสอบการคำนวณ ζ และ ξ

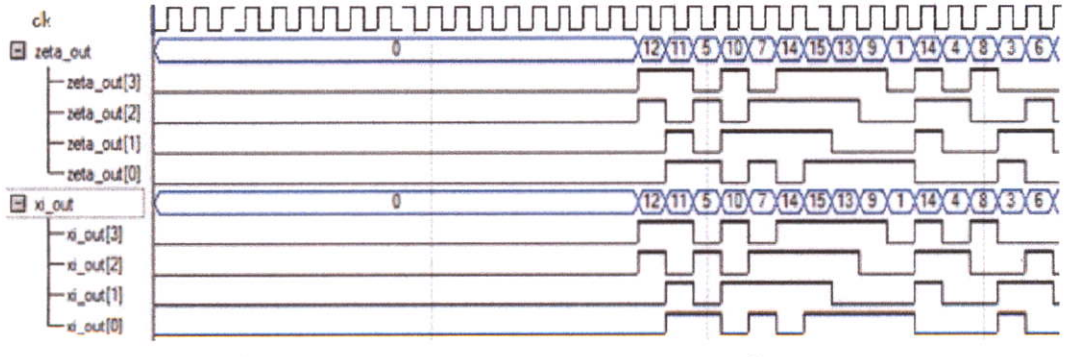
ζ และ ξ คือพหุนามบอกขนาดผิดพลาด และรีจีสเตอร์สำรองตามลำดับ เพื่อในตอนท้ายสุด จะได้นำไปคำนวณเพื่อแก้ไขข้อผิดพลาด



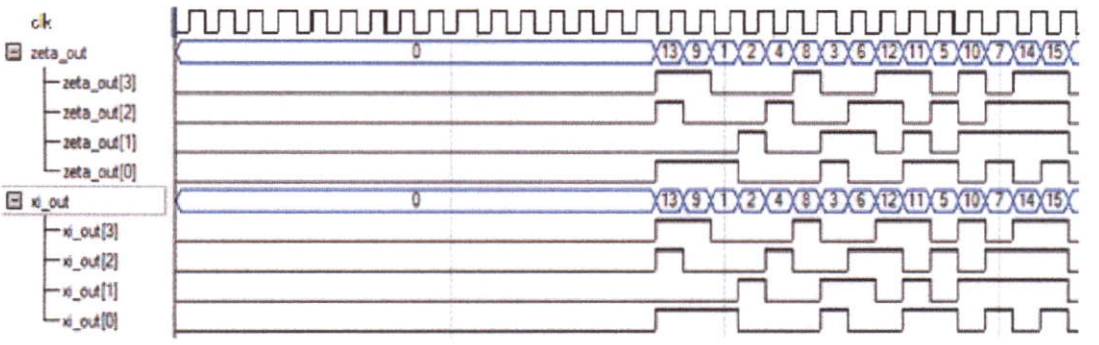
รูปที่ 4.21 ผลการทดสอบโมดูลคำนวณ ζ และ ξ ที่ r = 1



รูปที่ 4.22 ผลการทดสอบโมดูลคำนวณ ζ และ ξ ที่ r = 2



รูปที่ 4.23 ผลการทดสอบโมดูลคำนวณ ζ และ ξ ที่ r = 3



รูปที่ 4.24 ผลการทดสอบโมดูลคำนวณ **ζ** และ **ξ** ที่ r = 4

จากรูปที่ 43 – 46 แสดงให้เห็นผลการคำนวณ ζ และ ξ จากการป้อนคำรหัสตัวอย่าง RS(15,11) คือ {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1} และผลลัพธ์ที่ได้ คือ

1) r = 1

ζ = {13,9,1,24,8,3,6,12,11,5,10,7,14,15}

ξ = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}

2) r = 2

ζ = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}

ξ = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}

3) r = 3

ζ = {12,11,5,10,7,14,15,13,9,1,14,4,8,3,6}

ξ = {12,11,5,10,7,14,15,13,9,1,14,4,8,3,6}

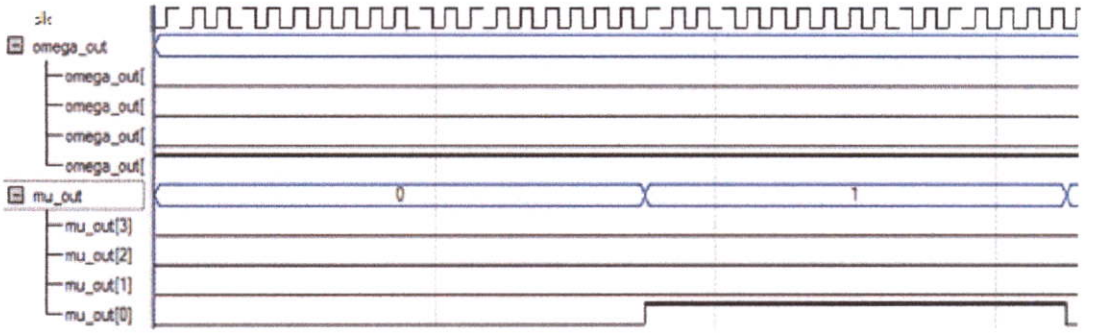
4) r = 4

ζ = {13,9,1,2,4,8,3,6,12,11,5,10,7,14,15}

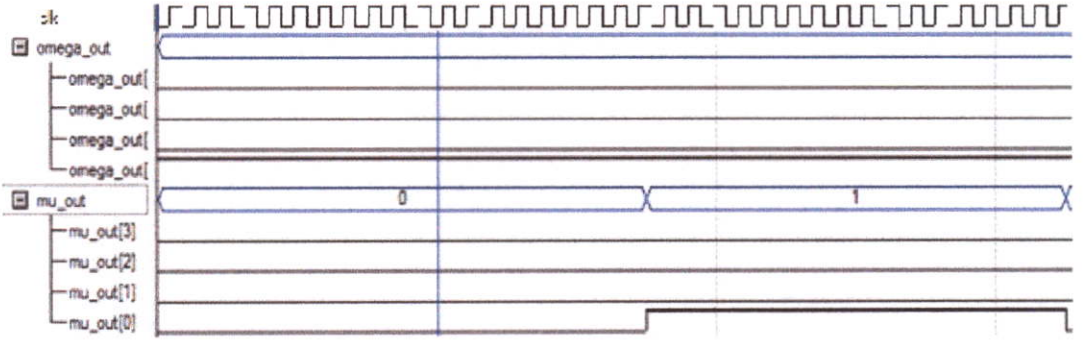
ξ = {13,9,1,2,4,8,3,6,12,11,5,10,7,14,15}

4.2.1.5 ผลการทดสอบการคำนวณ ω และ μ

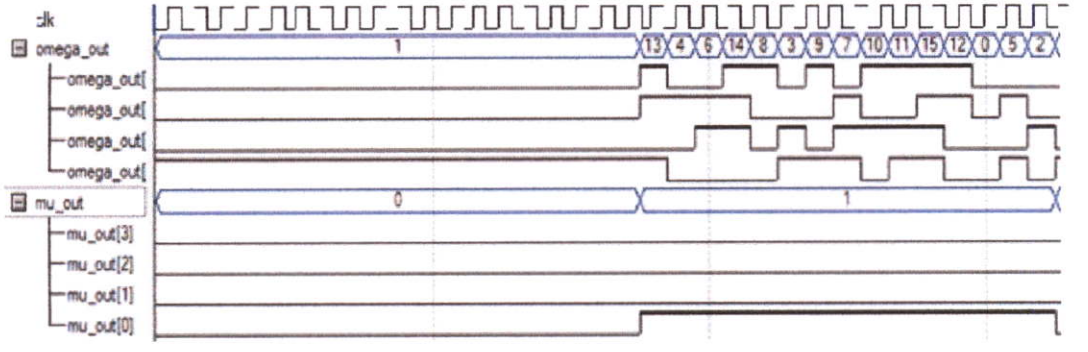
ω และ μ คือพหุนามคำนวณขนาดที่ซึ่งต้องนำมาคำนวณร่วมกับ ζ และ ξ เพื่อทราบขนาดของข้อมูลผิดพลาด



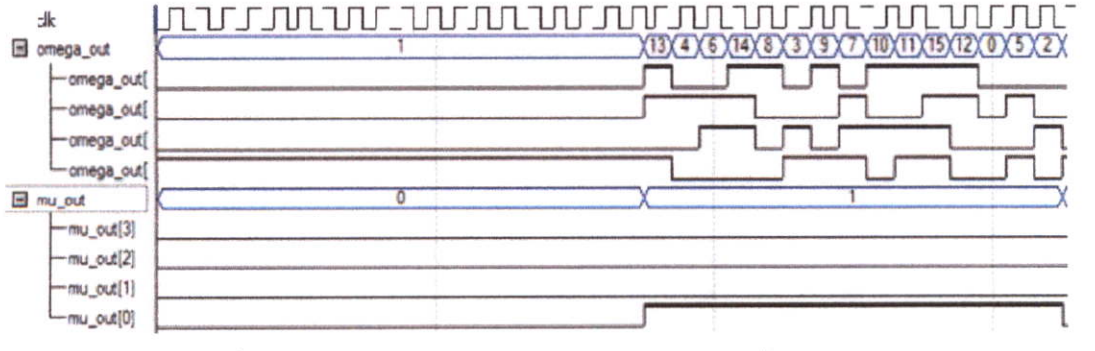
รูปที่ 4.25 ผลการทดสอบโมดูลคำนวณ ω และ μ ที่ r = 1



รูปที่ 4.26 ผลการทดสอบโมดูลคำนวณ ω และ μ ที่ r = 2



รูปที่ 4.27 ผลการทดสอบโมดูลคำนวณ ω และ μ ที่ r = 3



รูปที่ 4.28 ผลการทดสอบโมดูลคำนวณ ω และ μ ที่ r = 4

จากรูปที่ 47 – 50 แสดงให้เห็นผลการคำนวณ ω และ μ จากการป้อนคำรหัสตัวอย่าง RS(15,11) คือ {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1} และผลลัพธ์ที่ได้ คือ

1) r = 1

ω = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

μ = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

2) r = 2

ω = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

μ = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

3) r = 3

ζ = {13,4,6,14,8,3,9,7,10,11,15,12,0,5,2}

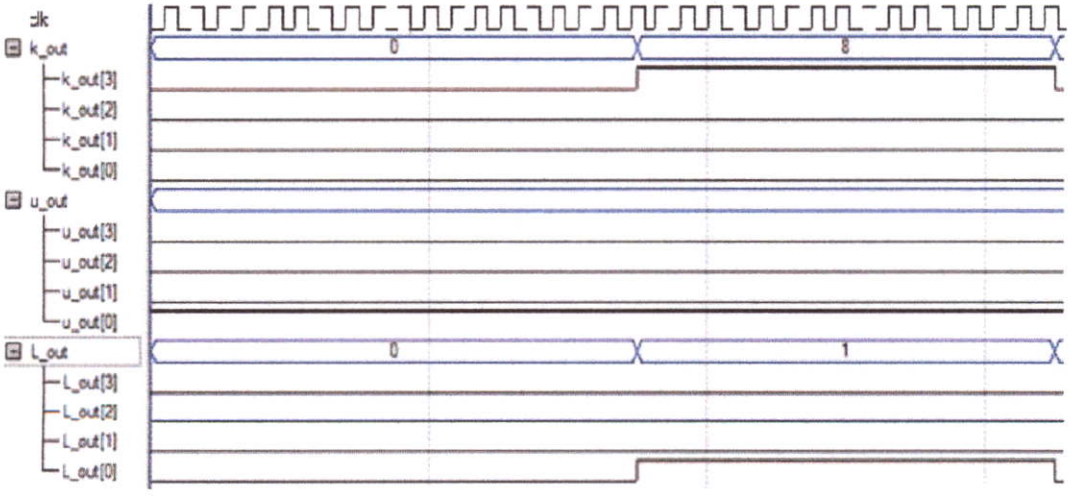
μ = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

4) r = 4

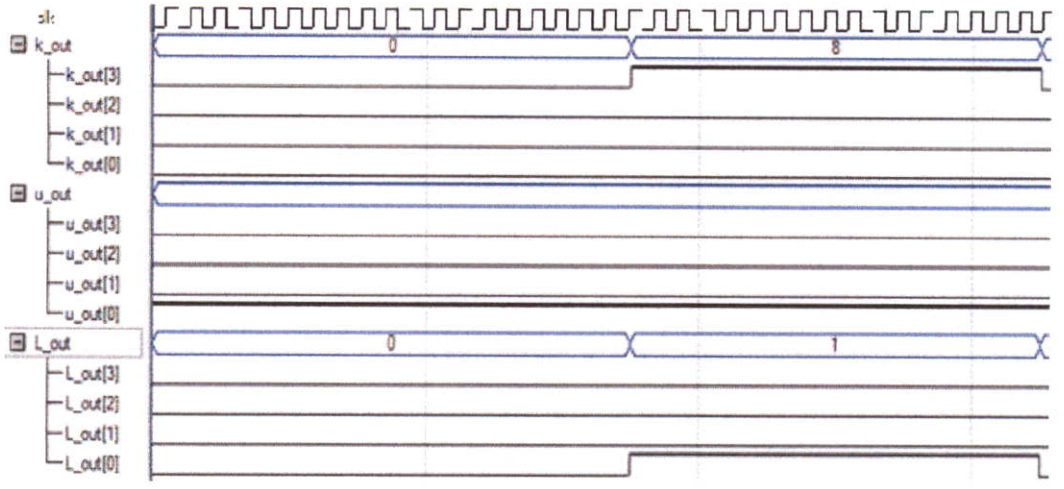
ω = {13,4,6,14,8,3,9,7,10,11,15,12,0,5,2}

μ = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

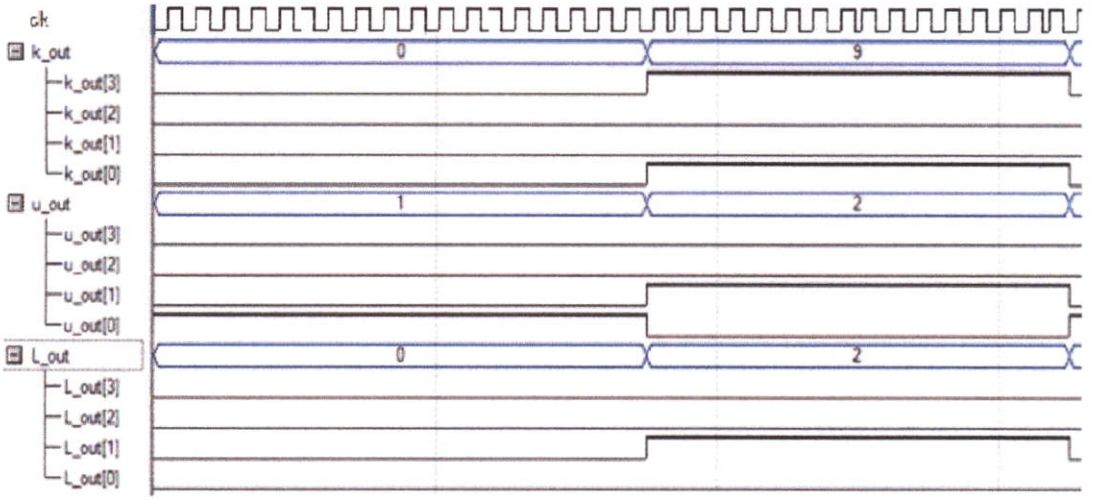
4.2.1.6 ผลการทดสอบการคำนวณ K, L และ u



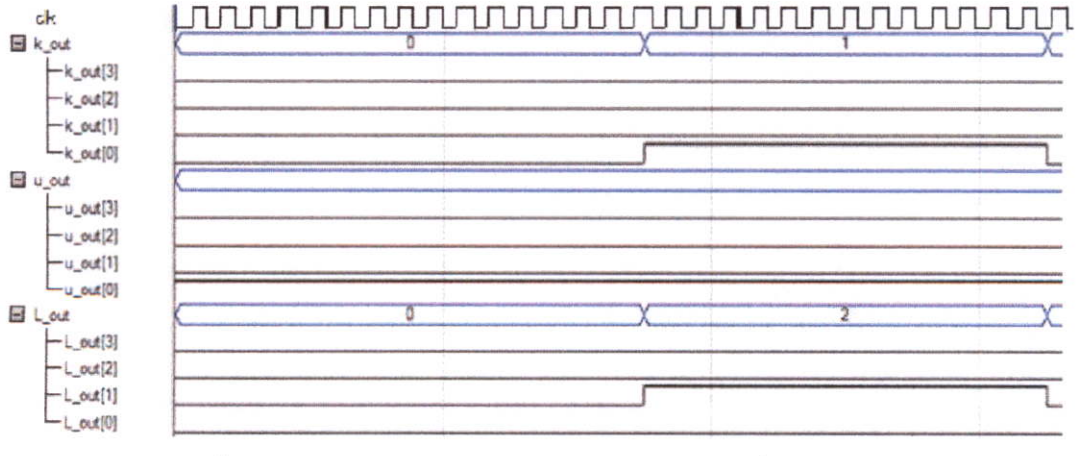
รูปที่ 4.29 ผลการทดสอบโมดูลคำนวณ K, L และ u ที่ r = 1



รูปที่ 4.30 ผลการทดสอบโมดูลคำนวณ K, L และ u ที่ r = 2



รูปที่ 4.31 ผลการทดสอบโมดูลคำนวณ K, L และ u ที่ r = 3



รูปที่ 4.32 ผลการทดสอบโมดูลคำนวณ K, L และ u ที่ r = 4

จากรูปที่ 4.29 – 4.32 แสดงให้เห็นผลการคำนวณ r, K, L, และ u จากการป้อนคำรหัสตัวอย่าง RS(15,11) คือ {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1} และผลลัพท์ที่ได้ คือ

1) r = 1

   K = 8

   L = 0

   u = 1

2) r = 2

   K = 8

   L = 0

   u = 1

3) r = 3

   K = 9

   L = 2

   u = 2

4) r = 4

   K = 8

   L = 0

   u = 1

เมื่อทำการทดลองการถอดรหัสโดยแยกส่วน แต่ละตัวแปรจนเสร็จสมบูรณ์แล้ว นำมารวมเข้าเป็นระบบ โดยจะแบ่งเป็น 2 ระบบ คือ แบบไปป์ไลน์ และแบบวนรอบ

4.3 การสังเคราะห์วงจร

จากการออกแบบใน 4.1 และผลการจำลองใน 4.2 นำวงจรมาสังเคราะห์ ได้ผลการสังเคราะห์วงจร ในด้านต่างๆ ทั้งทรัพยากร ความถี่สูงสุดที่ใช้ได้ และพลังงานที่ใช้ ดังตารางที่ 4.2

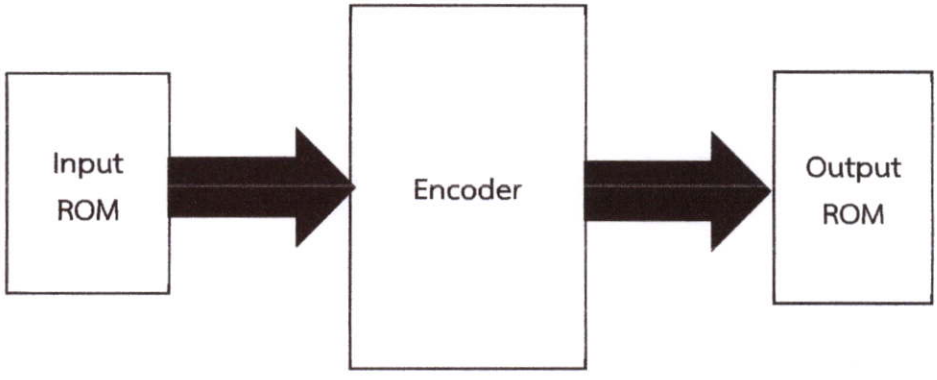
ตารางที่ 4.2 เปรียบเทียบทรัพยากรสำหรับ RS(15,11) และ RS(255,223)

		RS(15,11)		RS(255,223)	
		ไปป์ไลน์	วนรอบ	ไปป์ไลน์	วนรอบ
จำนวนลอจิกบล็อก	Area Optimize	266	1,109	29,487	2,224
	Speed Optimize	462	1,472	47,673	2,970
จำนวนรีจิสเตอร์	Area Optimize	743	948	7,383	1,856
	Speed Optimize	880	1,317	45,076	2,609
จำนวนหน่วยความจำ (บิต)	Area Optimize	2,053	523	33,779	2,737
	Speed Optimize	2,233	0	61,440	2,048
Data delay between blocks (Clock Cycle)		2	68	2	8,224
ความกว้างพัลซ์ต่ำสุด (นาโนเมตร)	Area Optimize	4.201	4.231	3.714	3.152
	Speed Optimize	2.000	2.857	3.174	3.057
ความถี่สูงสุด (MHz)	Area Optimize	238.04	236.50	315.06	317.26
	Speed Optimize	500.00	350.02	269.25	327.12
พลังงานรวม (mW)	Area Optimize	76.97	76.52	135.12	297.07
	Speed Optimize	77.01	76.53	137.57	297.10

4.4 การทดสอบกับฮาร์ดแวร์

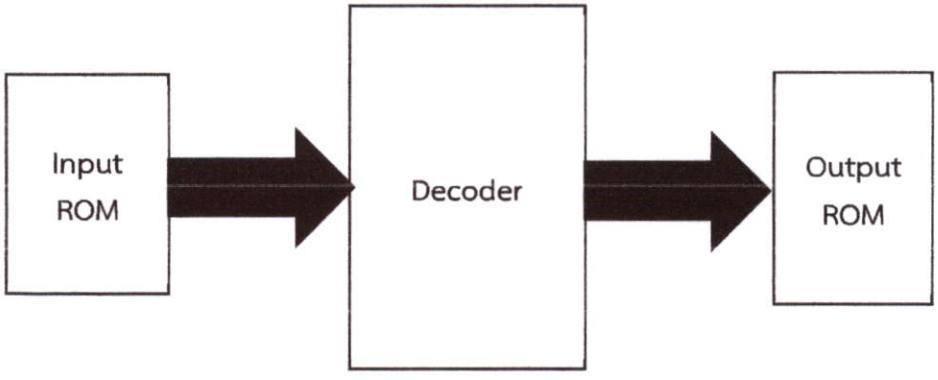
4.4.1 การออกแบบการทดสอบระบบบนฮาร์ดแวร์

การทดสอบระบบบนฮาร์ดแวร์นั้น งานวิจัยนี้ได้ทดสอบระบบเข้ารหัส และถอดรหัส เพื่อทดลองการทำงานจริงของระบบที่ออกแบบ โดยจะเขียน ข้อมูลอินพุตลงในระบบ และหลังถอดรหัส จะให้ระบบเขียนเอาท์พุตลงหน่วยความจำประเภทรอม แล้วอ่านข้อมูลออกมาว่าได้ตามที่คำนวณไว้หรือไม่ โดยจะเลือกเข้ารหัส RS(15,11) มีแผนภาพระบบดังนี้



รูปที่ 4.33 การออกแบบการทดสอบระบบเข้ารหัสบนฮาร์ดแวร์

สำหรับการถอดรหัส เลือกใช้เวกเตอร์ λ และ β ในการทดสอบ เพื่อทดสอบความถูกต้องในการคำนวณ โดยออกแบบระบบคล้ายกับการทดสอบระบบเข้ารหัส คือเขียนอินพุตลงในรอม แล้วอ่านข้อมูลที่คำนวณได้ จากรอม โดยมีแผนภาพระบบดังนี้



รูปที่ 4.34 การออกแบบการทดสอบระบบถอดรหัสบนฮาร์ดแวร์

4.4.2โปรแกรม Quartus II 8.1 Web Edition

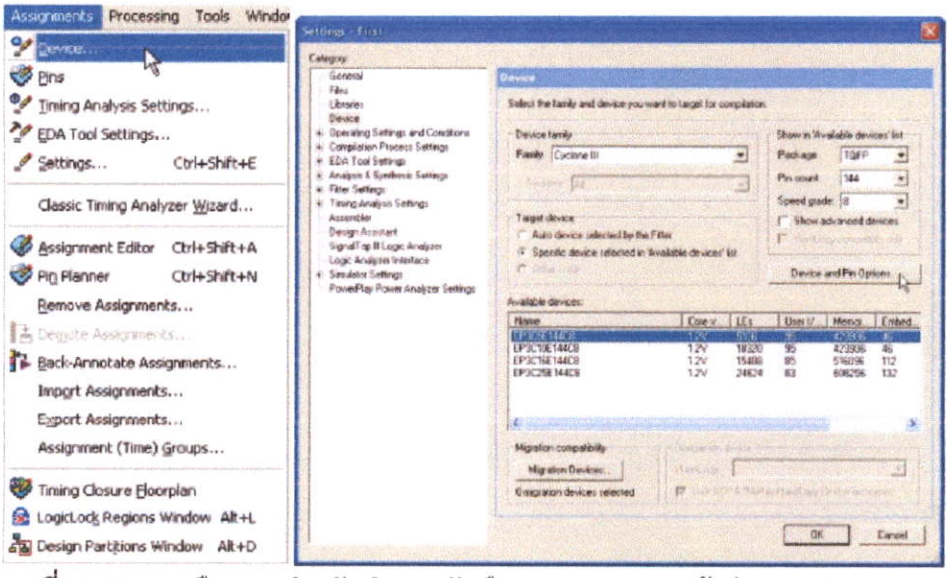
Quartus II เป็นซอฟต์แวร์ที่ใช้สำหรับออกแบบวงจรดิจิตอลโดยเขียนวงจรเป็นแบบบล็อกไดอะแกรม (Block Diagram File), แผนผังวงจร (Schematic File), ภาษาสำหรับอธิบายการทำงานของวงจร (Hardware Description Language) เช่น AHDL, VHDL, Verilog เป็นต้น นอกจากนี้ยังสนับสนุนการออกแบบระบบบนชิบ (System on a Programmable Chip: SoPC) อีกด้วย ตัวอย่างชิปที่ Quartus II 8.1 สนับสนุน ได้แก่

- CPLD Device เช่น MAX II, CPLD MAX3000A, MAX7000AE, MAX7000B, MAX7000S
- FPGA Device เช่น Cyclone, Cyclone II, Cyclone III, Stratix, Stratix II, Stratix II, Stratix IV, FLEX10KE, FLEX10K, FLEX 10KA, ACEX, FLEX 6000

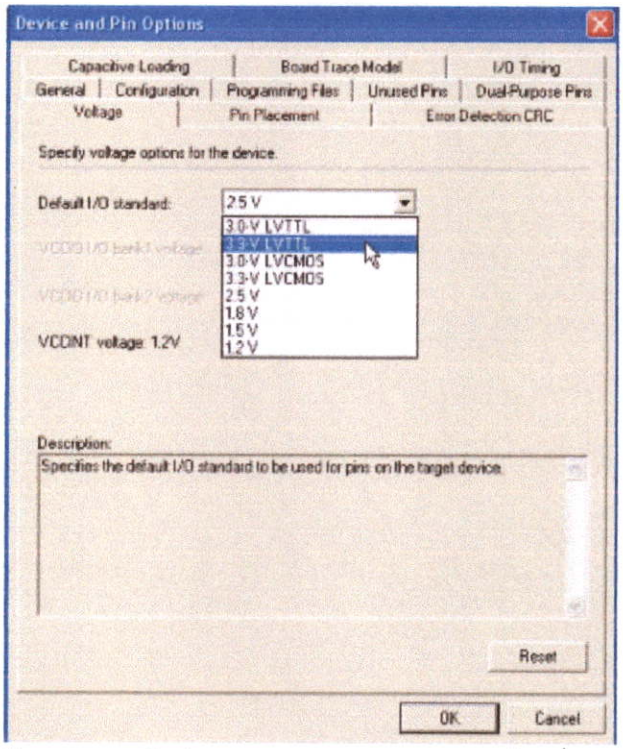
4.4.3การทดสอบวงจรบนบอร์ด Cyclone III EP3C10E144C8

หลังจากเขียนโค้ด และจำลองวงจรเรียบร้อยแล้วการทดลองในขั้นต่อไปคือการนำวงจรที่ออกแบบ มาทดสอบระบบถอดรหัสบนฮาร์ดแวร์จริง โดยงานวิจัยนี้ จะใช้บอร์ด Cyclone III EP3C10E144C8 เพื่อทดสอบระบบ โดยมีกระบวนการดำเนินงานดังนี้

1) กำหนดค่าแรงดัน IO ให้แก่วงจร โดยคลิกที่เมนู Assignments -> Device... จะปรากฏหน้าต่าง Settings ขึ้นมา ให้คลิกที่ปุ่ม Device and Pin Options... ซึ่งจะมีหน้าต่าง Device and Pin Options ปรากฏขึ้นมาให้คลิกที่แท็บ Voltage ในช่อง Defalut I/O standard คลิกเลือกที่ 3.3-V LVTTL แล้วคลิก OK เพื่อปิดหน้าต่าง



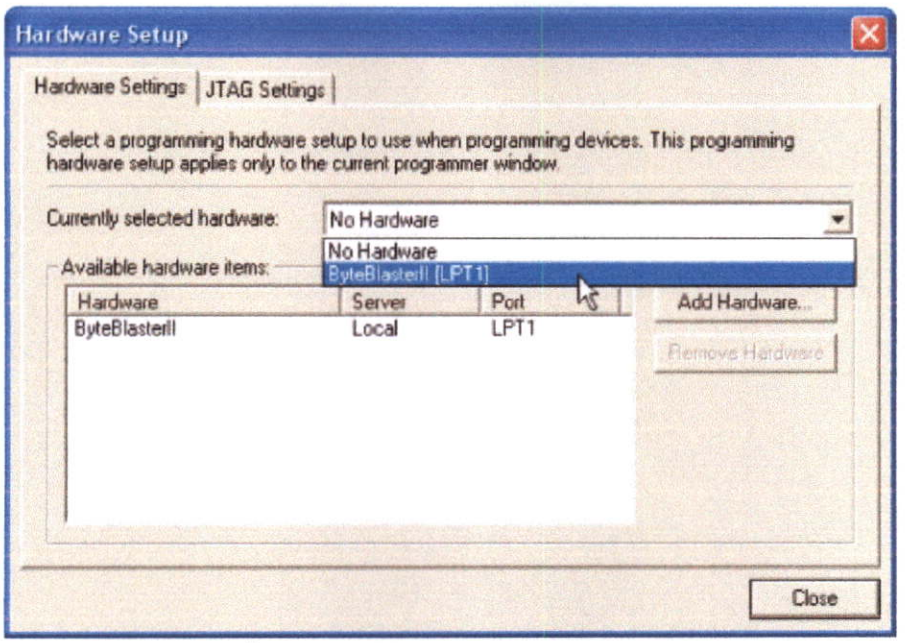
รูปที่ 4.35 การเลือกเมนูสำหรับกำหนดตัวเลือก Device และหน้าต่าง Settings



รูปที่ 4.36 ที่แท็บ Voltage ในช่อง Default I/O standard เลือกที่ 3.3-V LVTTL

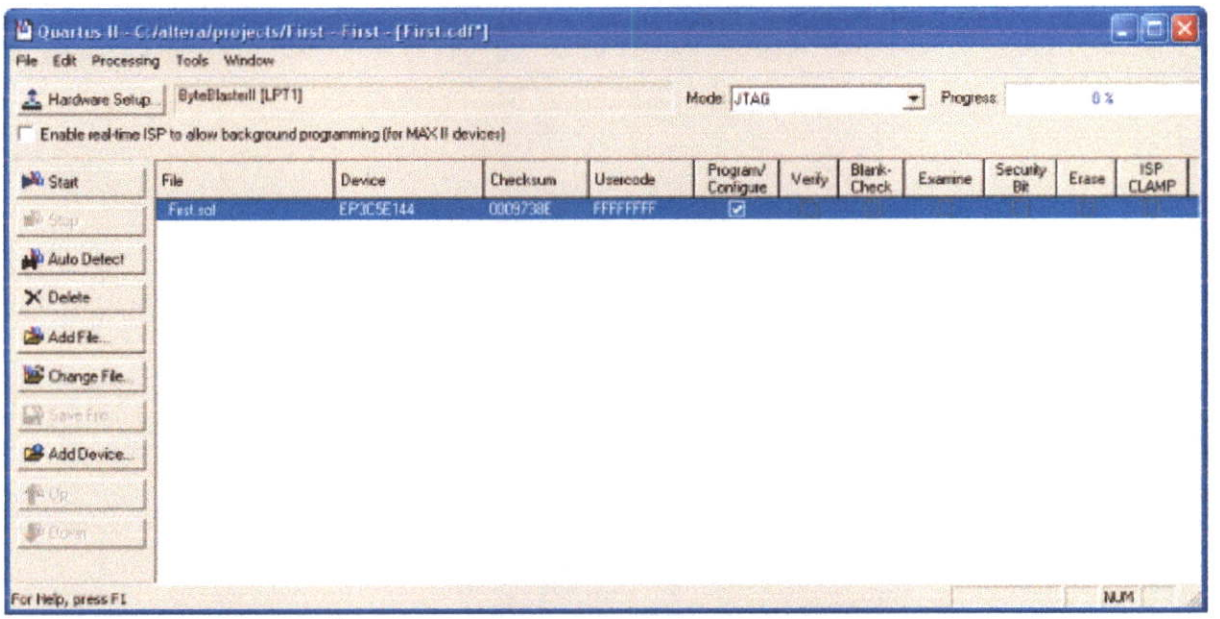
2) ทำการคอมไพล์วงจรใหม่อีกครั้ง โดยเลือกที่เมนู Processing -> Start Compilation

3) เมื่อคอมไพล์เสร็จเรียบร้อยแล้ว ทำการโปรแกรมลงบอร์ดทดลอง Cyclone III เพื่อทดสอบการทำงานจริง เลือกที่เมนู Tools -> Programmer จะปรากฏหน้าต่าง Programmer ขึ้นมา ในกรณีที่ยังไม่เคยเลือก Hardware สำหรับดาวน์โหลดให้ทำการคลิกปุ่ม Hardware Setup.. จะมีหน้าต่าง Hardware Setup ปรากฏขึ้นมาดังรูป ที่ 56 ในช่อง Currently Selected Hardware ให้เลือก ByteBlasterII หลังจากนั้นคลิกที่ปุ่ม Close



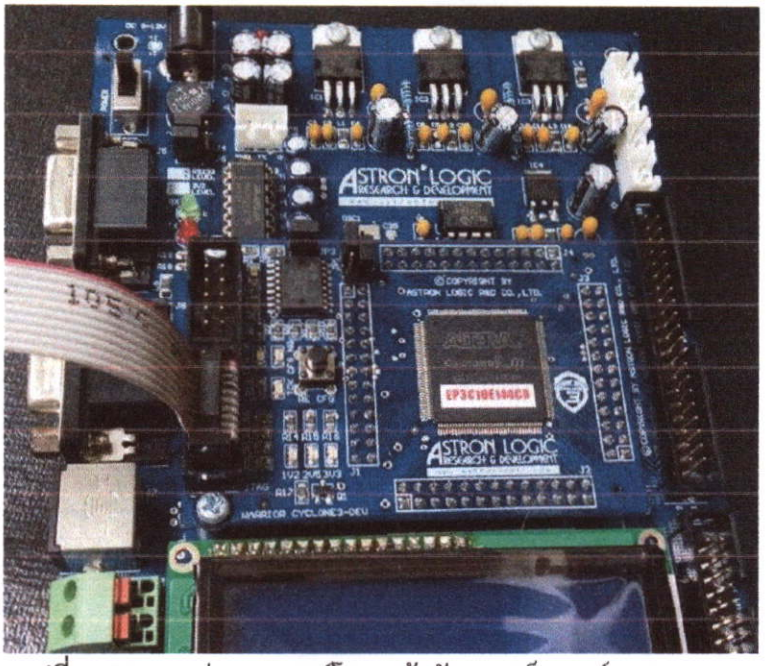
รูปที่ 4.37 หน้าต่าง Hardware Setup

4) จากนั้น คลิก Enable ในช่อง Program/Configure ในหน้าต่าง Programmer



รูปที่ 4.38 เลือก Program/Configure

5) เสียบคอนเนคเตอร์ ลงในช่อง JTAG บนโมดูล Warrior Cyclone III



รูปที่ 4.39 การต่อสายดาวน์โหลดเข้ากับคอนเน็คเตอร์ JTAG

6) คลิกเลือกที่ปุ่มสตาร์ท เป็นการโปรแกรมข้อมูลลงบนบอร์ด แล้วอ่านผลการทดสอบ

4.4.4 ผลการทดสอบระบบ

จากการทดสอบระบบที่ความถี่สัญญาณนาฬิกา 50 MHz ได้ผลการทดลองดังนี้

ตารางที่ 4.3 ผลการอ่านข้อมูลการเข้ารหัสจาก ROM

อินพุต	เอาท์พุต
{0,0,0,0,0,0,0,0,0,0,0}	{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
{1,1,1,1,1,1,1,1,1,1,1}	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
{2,2,2,2,2,2,2,2,2,2,2}	{2,2,2,2,2,2,2,2,2,2,2,2,2,2,2}
{3,3,3,3,3,3,3,3,3,3,3}	{3,3,3,3,3,3,3,3,3,3,3,3,3,3,3}
{0,1,2,3,4,5,6,7,8,9,10}	{0,1,2,3,4,5,6,7,8,9,10,12,14,8,3}

ตารางที่ 4.4 ผลการอ่านข้อมูลการถอดรหัสจาก ROM

อินพุต	เอาท์พุต	
	λ	β
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,1}	{1,5,10,8,5,14,3,12,10,12,14,7,7,3,8}	{12,8,0,3,5,9,2,7,13,10,4,11,6,15,}
{0,0,0,0,0,0,0,1,0,0,0,0,0,1,0}	{10,12,14,7,7,3,8,1,5,10,8,5,14,3}	{13,10,4,11,6,15,14,12,8,0,3,5,9,2}
{0,0,0,0,1,0,0,0,0,0,1,0,0,0,0}	{14,3,12,10,12,14,7,7,3,8,1,5,10,8}	{9,2,7,13,10,4,11,6,15,14,12,8,0,3}
{0,1,0,0,0,0,0,0,0,0,0,0,0,0,1}	{0,13,7,12,12,11,10,1,6,0,10,6,7,11}	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
{0,0,0,0,0,1,0,0,1,0,0,0,0,0,0}	{6,4,1,15,3,4,15,13,6,10,3,10,13,8}	{7,13,10,4,11,6,15,14,12,8,0,3,5,9}

จากการทดลองทั้งการถอดรหัส และเข้ารหัสนั้น ได้ทำการตรวจสอบโดยการเปรียบเทียบกับผลการเข้ารหัสด้วยโปรแกรมสำเร็จรูป MATLAB และพบว่าผลลัพธ์นั้นถูกต้อง

บทที่ 5
การวิจารณ์และสรุปผล

วิทยานิพนธ์ฉบับนี้ศึกษาวิจัยเกี่ยวกับการออกแบบ ระบบถอดรหัสรีด – โซโลมอน ซึ่งสามารถปรับเวลาได้บนโดเมนเวลา จากการศึกษางานวิจัยที่เกี่ยวข้องนั้น สามารถแบ่งงานวิจัยออกเป็นสองส่วน คือการพัฒนาอัลกอริทึมเพื่อลดการคำนวณ และสถาปัตยกรรมฮาร์ดแวร์เพื่อประหยัดต้นทุนในการสร้างฮาร์ดแวร์

สำหรับการลดการคำนวณนั้น ได้เริ่มจากการพัฒนาอัลกอริทึม จากโดเมนความถี่มาเป็นโดเมนเวลา โดยการใช้เทคนิคการแปลงกลับฟูริเยร์[12] จากนั้น [13] จึงได้ทำการวิจัยพัฒนาสถาปัตยกรรมฮาร์ดแวร์ สำหรับถอดรหัสรีด – โซโลมอน บนโดเมนเวลา ซึ่งได้นำเสนอสิ่งที่เป็นประโยชน์ในงานวิจัยไว้ คืออัลกอริทึมที่ใช้การคำนวณน้อยลงซึ่งสามารถลดจำนวนการคูณเหลือเพียงแค่ 60%

ในเรื่องของการพัฒนาสถาปัตยกรรมฮาร์ดแวร์นั้น ได้มีการนำเสนอการออกแบบฮาร์ดแวร์โดยมีสถาปัตยกรรมถอดรหัสแบบไปป์ไลน์ และแบบขนาน ซึ่งได้มีการเปรียบเทียบจำนวนเกต ความยาวสัญญาณนาฬิกา และจำนวนรอบในการคำนวณ สถาปัตยกรรมไปป์ไลน์ใช้จำนวนลอจิกเกตมากกว่า แต่ใช้ความยาวสัญญาณนาฬิกาสั้นกว่า และรอบคำนวณน้อยกว่า อีกทั้งสถาปัตยกรรมแบบไปป์ไลน์ ยังง่ายต่อการนำมาพัฒนาต่อ เพื่อทำเป็นแบบวนรอบได้ งานวิจัยนี้จึงได้เลือกสถาปัตยกรรมแบบไปป์ไลน์มาพัฒนาต่อ

สำหรับในงานวิจัย ได้ทำการศึกษาอัลกอริทึมการถอดรหัสบนโดเมนเวลาแต่ละสมการมาเขียนในรูปแบบของแผนภาพกระแสข้อมูล แล้วนำมาออกแบบเป็นฮาร์ดแวร์ เพื่อคำนวณในบล็อกคำนวณย่อย และนำมาสร้างเป็นระบบถอดรหัส 2 รูปแบบ คือ ไปป์ไลน์ และวนรอบ โดยระบบถอดรหัสไปป์ไลน์ จะใช้บล็อกคำนวณย่อย เท่ากับ 2t บล็อก สำหรับระบบถอดรหัสแบบวนรอบ จะใช้บล็อกคำนวณย่อยเพียงแค่ 1 บล็อก แล้วทำการวนข้อมูลเข้ามาคำนวณทั้งหมด 2t รอบ จากการจำลองโดยใช้ RS(15,11) ซึ่งมีค่า 2t = 4 มีดีเลย์ของข้อมูลระหว่างบล็อกของแบบไปป์ไลน์ 2 สัญญาณนาฬิกา และแบบวนรอบ 68 สัญญาณนาฬิกา และใน RS(255,223) แบบไปป์ไลน์ มีดีเลย์ของข้อมูล 2 สัญญาณนาฬิกา และแบบวนรอบ 8,224 สัญญาณนาฬิกา จะเห็นว่าแบบวนรอบนั้นจะมีการรอเพื่อให้มีการคำนวณข้อมูลในบล็อกก่อนหน้าเสร็จครบ 2t รอบเสียก่อนจึงทำให้เสียเวลาในการในการคำนวณ ในขณะที่แบบไปป์ไลน์ นั้นรอระหว่างบล็อกข้อมูลเพียงแค่ 2 สัญญาณนาฬิกา (เพื่อรอคำนวณ Δ_r) เท่านั้น

สำหรับการศึกษาวิจัยในเรื่องฮาร์ดแวร์นั้น ได้ทำการสังเคราะห์วงจรบนชิป Altera Cyclone III EP3C10E144C8 ได้ผลการสังเคราะห์วงจรสำหรับ RS(15,11) บนระบบไปป์ไลน์ เปรียบเทียบระหว่างการสังเคราะห์แบบจำกัดพื้นที่ (Area Optimize) กับแบบเน้นความเร็ว (Speed Optimize) จะมีจำนวนลอจิกบล็อก 266 และ 462 บล็อก ตามลำดับ มีจำนวนรีจิสเตอร์ 743 และ 880 หน่วย ตามลำดับ และใช้จำนวนหน่วยความจำ 2,053 และ 2,253 บิต ตามลำดับ ซึ่งจะเห็นได้ว่า จำนวนรีจิสเตอร์ และจำนวนหน่วยความจำมีปริมาณใกล้เคียงกัน แต่ว่าจำนวนลอจิกบล็อกนั้นการสังเคราะห์

แบบจำกัดพื้นที่จะใช้น้อยกว่าเกือบ 2 เท่า แต่หากพิจารณาที่ความถี่สูงสุดที่ได้นั้นการสังเคราะห์เพื่อต้องการความเร็วสูงสุด ก็จะได้ความเร็วมากเป็น 2 เท่าเช่นกัน

ในรหัส RS(15,11) บนระบบวนรอบ เปรียบเทียบระหว่างการสังเคราะห์แบบจำกัดพื้นที่ กับแบบเน้นความเร็ว จะมีจำนวนลอจิกบล็อก 1,109 และ 1,472 บล็อก ตามลำดับ มีจำนวนรีจิสเตอร์ 948 และ 1,317 หน่วยตามลำดับ และใช้จำนวนหน่วยความจำ 523 และ 0 บิต ตามลำดับ และได้ความเร็วสูงสุด ต่างกันประมาณ 1.5 เท่า จะเห็นว่าระบบวนรอบที่ออกแบบไว้นั้นใช้ทรัพยากรสูงกว่าระบบไปป์ไลน์ เพราะระบบวนรอบจะต้องมีบัฟเฟอร์ มัลติเพล็กเซอร์ และดีมัลติเพล็กเซอร์ในปริมาณมากนั่นเอง ส่วนในด้านพลังงานนั้น พบว่า การทดสอบกับระบบ RS(15,11) ทุกรูปแบบ มีพลังงานที่ใช้ไม่ต่างกันนัก คือต่างกันเพียงแค่ 0.5 mW เท่านั้น โดยระบบไปป์ไลน์ใช้พลังงานสูงกว่าระบบวนรอบ

ส่วนการสังเคราะห์ระบบ RS(255,233) ระบบไปป์ไลน์ เปรียบเทียบระหว่างการสังเคราะห์แบบจำกัดพื้นที่ กับแบบเน้นความเร็ว จะมีจำนวนลอจิกบล็อก 29,487 และ 47,673 บล็อก ตามลำดับ มีจำนวนรีจิสเตอร์ 7,383 และ 45,076 หน่วยตามลำดับ และใช้จำนวนหน่วยความจำ 33,779 และ 61,440 บิต ตามลำดับ และได้ความเร็วสูงสุด ต่างกันเพียงเล็กน้อย ส่วนการสังเคราะห์ระบบ RS(255,233) ระบบวนรอบ เปรียบเทียบระหว่างการสังเคราะห์แบบจำกัดพื้นที่ กับแบบเน้นความเร็ว จะมีจำนวนลอจิกบล็อก 2,224 และ 2,970 บล็อก ตามลำดับ มีจำนวนรีจิสเตอร์ 1,856 และ 2,609 หน่วยตามลำดับ และใช้จำนวนหน่วยความจำ 2,737 และ 2,048 บิต ตามลำดับ และได้ความเร็วสูงสุดต่างกันเพียงเล็กน้อย แต่หากเปรียบเทียบทรัพยากรที่ใช้ระหว่างระบบไปป์ไลน์และวนรอบนั้น พบว่าระบบวนรอบใช้ทรัพยากรน้อยกว่าประมาณ 18 – 20 เท่า แต่ก็ใช้พลังงานมากกว่า 2.2 เท่าเช่นกัน

เอกสารอ้างอิง

[1] D. Lee, S. Lee, and J. Kim. 2001. "A Reed-Solomon Decoder with efficient recursive cell architecture for DVD application," 184-185. IEEE International Conference on Consumer Electronics.

[2] Richard S. Colon, John J. Dwyer, Devid K. Godin, John J. Pawloski, Stephen Rothschild. Portable digital audio recorder with adaptive control configurations. USA, US 6038199A, March 2000.

[3] Joseph J.K., **ÓRaunaidh,** Thierry Pun. 1997, "Rotation, scale and translation invariant spread spectrum digital image watermarking," 303 – 317, ELSEVIER.

[4] B. Tiwari and R. Mehra, 2012 "FPGA Implementation of RS Codec for Digital Video Broadcasting," 68 – 77, VSRD IJEECE, Vol.2(2).

[5] Dakshi A., Vahid T., Ayman N. and Nambi S., 1998, "Space-Time Codded OFDM for High Data-Rate Wireless Communication Over Wideband Channels," 2232-2236, IEEE.

[6] B. Moision and J. Hamkins, 2003, "Deep-Space Optical Communications Downlink Budget: Modulation and Coding," IPN Progress Report 42-154.

[7] Mark S. Y., John D., Michael C. S. Method and apparatus for error detection and correction in systems comprising floppy and/or hard disk drives. USA. US4667326A, May 1987

[8] Hsie-Chia C., C. Bernard Shung, Chen-Yi Lee. 2001. "A Reed-Solomon Product-Code (RS-PC) Decoder Chip for DVD Applications," 229-238, IEEE Journal of Solid-State Circuits, Vol.36 No.2

[9] Lassalle R. and M. Alard, 1987, "Principles of modulation and channel coding for digital broadcasting for mobile receivers," 168-190, EBU Review – Technical.

[10]Dilip V. S. and Naresh R. S., 2001, "High-Speed Architectures for Reed-Solomon Decoders," 641 – 655, IEEE.

[11]Shannon C. E., 1948, "A Mathematical Theory of Communication," 379-423, The Bell System Technical Journal, Vol.27

[12]Blahut R. E., 1983, "Theory and practice of Error Control Codes", Addison-Wesley, Reading, MA

เอกสารอ้างอิง (ต่อ)

[13]Forney Jr. G., 1965, "On Decoding BCH Codes," IEEE Transactions on Information Theory, Vol. it-11, No.4, 549-557,

[14]Somsak C., "Fast Transform Techniques for RS Codes," Ladkraban Engineering Journal, Vol.12, No.1, June 1995, pp.32-41

[15]Reed Irving S., Solomon Gustave, "Polynomial Codes over Certain Finite Fields," Journal of the Society for Industrial and Applied Mathematics(SIAM) 8 (2), 1960, pp. 300-304

[16]Choomchuay S., Arambepola B., "Time Domain Algorithms and Architectures for Reed – Solomon Decoding," IEE Proceedings-I, Vol. 140, No. 3, June 1993, pp. 189 – 196

ภาคผนวก

ภาคผนวก ก

ตัวอย่างการเข้ารหัสรีด – โซโลมอน

กำหนดให้ m = {10,14,13,9,12,6,5,15,7,} ซึ่งเป็นรหัสบน GF(16)

1. คำนวณพุนามกำเนิด

$$g(x) = \prod_{i=1}^{2t}(x - \alpha^i)$$

เนื่องจาก m มีจำนวน 9 อิเลเมนต์ ดังนั้น ต้องเข้ารหัส RS(15,9) จะได้ 2t = 6 ดังนั้น

$$\begin{aligned} g(x) &= \prod_{i=0}^{6}(x - \alpha^i) \\ &= (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6) \\ &= \alpha^6 + \alpha^9 x + \alpha^6 x^2 + \alpha^4 x^3 + \alpha^{14} x^4 + \alpha^{10} x^5 + x^6 \\ &= 12 + 10x + 12x^2 + 3x^3 + 9x^4 + 7x^5 + x^6 \end{aligned}$$

จากโจทย์ จะได้

$$m(x) = 10 + 14x + 13x^2 + 9x^3 + 12x^4 + 6x^5 + 5x^6 + 15x^7 + 7x^8$$

$$\begin{aligned} p(x) &= x^6 m(x) \bmod g(x) \\ &= 5 + 15x + 12x^2 + 4x^3 + 2x^4 + x^5 \end{aligned}$$

ดังนั้น เมื่อนำมาเข้ารหัส จะได้ดังนี้

$$\begin{aligned} c(x) &= p(x) + x^6 m(x) \\ &= 5 + 15x + 12x^2 + 4x^3 + 2x^4 + x^5 + 10x^6 + 14x^7 + 13x^8 \\ &\quad + 9x^9 + 12x^{10} + 6x^{11} + 5x^{12} + 15x^{13} + 7x^{14} \end{aligned}$$

ดังนั้น

$$c_i = \{5,15,12,4,2,1,10,14,13,9,12,6,5,15,7\}$$

ตัวอย่างการถอดรหัสรีด – โซโลมอน

จากตัวอย่างที่ผ่านมา ได้คำรหัส $c_i = \{5,15,12,4,2,1,10,14,13,9,12,6,5,15,7\}$ ที่ถูกส่งผ่านช่องทางสื่อสาร โดยจะสมมติให้สัญญาณรบกวน ได้ทำให้เกิดสัญลักษณ์ลบเลือนที่ตำแหน่ง 3 และ 6 และเกิดสัญญาณรบกวนชุด (Burst Error) รวม 8 บิตต่อเนื่อง ที่สัญลักษณ์ที่ 11 และ 12

วิธีการ

1.) การเกิดสัญลักษณ์ลบเลือนตำแน่งที่ 3 และ 6 ทำให้มีค่าเป็น 0 (หรือ $\Psi(x) = 1 + 4x + 10x^2$)

2.) การเกิดสัญญาณรบกวน 8 บิต ต่อเนื่องที่สัญลักษณ์ที่ 11 และ 12 ทำให้สัญลักษณ์ดังกล่าวมีค่าเป็น 1111_2 หรือ 15

ดังนั้น คำรหัสที่ได้รับคือ $v_i = \{5,15,12,0,2,1,0,14,13,9,12,15,15,15,7\}$ ซึ่งการถอดรหัสทำได้ดังต่อไปนี้

ขั้นตอนที่ 1 หาค่าซินโดรม

$$S_k = v(\alpha^k) = \sum_{i=0}^{n-1} v_i \alpha^k$$

ซึ่งจะได้ $\{S_k\} = \{12,13,4,3,14,13\}$ หรือ $S(x) = 12 + 13x + 4x^2 + 3x^3 + 14x^4 + 13x^5$ จะเห็นได้ว่า

$$\begin{aligned} S'(x) &= S(x)\Psi(x) \mathrm{mod} x^6 \\ &= \langle (12 + 13x + 4x^2 + 3x^3 + 14x^4 + 13x^5)(1 + 4x + 10x^2) \rangle \mathrm{mod} x^6 \\ &= 12 + 8x + 4x^2 + 11x^3 + 12x^4 + 13x^5 \end{aligned}$$

ขั้นตอนที่ 2 วิธีการเบอริแคมป์ – แมสซีย์ (Berlekamp – Massey Algorithm)

ขั้นตอนที่ 2.1 : กำหนดค่าเริ่มต้น

$$\Lambda^{(0)}(x) = \mathrm{B}^{(0)}(x) = 1, \Gamma^{(0)}(x) = \mathrm{M}^{(0)}(x) = S(x) = L_0 = 0$$

รอบการคำนวณที่ 1:

$\Delta_1 = \alpha^{i_1} = \alpha^3 = 8$

$L_1 = L_0 + 1 = 1$

$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - \Delta_r \mathrm{x} \mathrm{B}^{(r-1)}(x)$

$\Lambda^{(1)}(x) = \Lambda^{(0)}(x) - \Delta_1 \mathrm{x} \mathrm{B}^{(0)}(x) = 1 + 8x$

$\mathrm{B}^{(r)}(x) = \Lambda^{(r)}(x)$

$\mathrm{B}^{(1)}(x) = \Lambda^{(1)}(x) = 1 + 8x$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - \Delta_r \mathrm{x} \mathrm{B}^{(r-1)}(x)$$

$$\Gamma^{(1)}(x) = \Gamma^{(0)}(x) - \Delta_1 \mathrm{x} \mathrm{B}^{(0)}(x)$$

$$= 12 + 7x + 6x^2 + 5x^2 + 5x^2 + 4x^2$$

$$\mathrm{M}^{(r)}(x) = \Gamma^{(r)}(x)$$

$$\mathrm{M}^{(1)}(x) = \Gamma^{(1)}(x) = 12 + 7x + 6x^2 + 5x^3 + 5x^4 + 4x^5$$

รอบการคำนวณที่ 2:

$$\Delta_2 = \alpha^{i_2} = \alpha^6 = 12$$

$$L_2 = L_1 + 1 = 2$$

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - \Delta_r \mathrm{x} \mathrm{B}^{(r-1)}(x)$$

$$\Lambda^{(2)}(x) = \Lambda^{(1)}(x) - \Delta_2 \mathrm{x} \mathrm{B}^{(1)}(x) = 1 + 8x + 12x(1 + 8x) = 1 + 4x + 10x^2$$

$$\mathrm{B}^{(r)}(x) = \Lambda^{(r)}(x)$$

$$\mathrm{B}^{(2)}(x) = \Lambda^{(2)}(x) = 1 + 4x + 10x^2$$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - \Delta_r \mathrm{x} \mathrm{M}^{(r-1)}(x)$$

$$\Gamma^{(2)}(x) = \Gamma^{(1)}(x) - \Delta_2 \mathrm{x} \mathrm{M}^{(1)}(x)$$

$$= 12 + 7x + 6x^2 + 5x^3 + 5x^4 + 4x^5 + 12x(12 + 7x + 6x^2 + 5x^2 + 5x^2 + 4x^2)$$

$$= 12 + 8x + 4x^2 + 11x^3 + 12x^4 + 13x^5$$

$$\mathrm{M}^{(r)}(x) = \Gamma^{(r)}(x)$$

$$\mathrm{M}^{(2)}(x) = \Gamma^{(2)}(x) = 12 + 8x + 4x^2 + 11x^3 + 12x^4 + 13x^5$$

รอบการคำนวณที่ 3:

$$\Delta_r = \sum_{j=0}^{t+\rho} \Lambda_j^{(r-1)} S_{r-j}$$

$$\Delta_3 = \sum_{j=0}^{5} \Lambda_j^{(2)} S_{3-j} = \Lambda_0^{(2)} S_3 + \Lambda_1^{(2)} S_3 + \Lambda_2^{(2)} S_3$$

$$= 1 + 4 + 10 = 4 \times 1 + 4 \times 13 + 10 \times 12 = 4$$

$\delta_3 = 1$ (เพราะว่า $\Delta_3 \neq 0$ และ $2L_{2-1} = 2 \times 2 = 4$ ซึ่ง $r + \rho + 1 = 3 + 2 - 1 = 4$)

$$L_r = \delta_r(r - L_{r-1} + \rho) + (1 - \delta_r)L_{r-1}$$

$$L_3 = 3 - 2 + 2 = 3$$

$$K_{r-1} = K_\rho = 1$$
$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r) K_{r-1}$$
$$K_3 = \Delta_3^{-1} = \frac{1}{4} = 13$$

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{B}^{(r-1)}(x)$$
$$\Lambda^{(3)}(x) = \Lambda^{(2)}(x) - K_2 \Delta_3 x \mathrm{B}^{(2)}(x)$$
$$= 1 + 4x + 10x^2 + 4x(1 + 4x + 10x^2)$$
$$= 1 + 9x^2 + 14x^3$$

$$\mathrm{B}^{(r)}(x) = \delta_r \Lambda^{(r-1)}(x) - (1 - \delta_r) x \mathrm{B}^{(r-1)}(x)$$
$$\mathrm{B}^{(3)}(x) = \Lambda^{(2)}(x)$$
$$= 1 + 4x + 10x^2$$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{M}^{(r-1)}(x)$$
$$\Gamma^{(3)}(x) = \Gamma^{(2)}(x) - K_2 \Delta_3 x \mathrm{M}^{(2)}(x)$$
$$= 12 + 8x + 4x^2 + 11x^3 + 12x^4 + 13x^5 - 4x(12 + 8x + 4x^2 + 11x^3 + 12x^4 + 13x^5)$$
$$= 12 + 13x + 2x^2 + 8x^3 + 6x^4 + 8x^5$$

$$\mathrm{M}^{(r)}(x) = \delta_r \Gamma^{(r-1)}(x) - (1 - \delta_r) x \mathrm{M}^{(r-1)}(x)$$
$$\mathrm{M}^{(3)}(x) = \delta_3 \Gamma^{(2)}(x) - (1 - \delta_3) x \mathrm{M}^{(3)}(x)$$
$$= 12 + 8x + 4x^2 + 11x^3 + 12x^4 + 13x^5$$

รอบการคำนวณที่ 4:

$$\Delta_r = \sum_{j=0}^{t+\rho} \Lambda_j^{(r-1)} S_{r-j}$$

$$\Delta_4 = \sum_{j=0}^{5} \Lambda_j^{(3)} S_{4-j} = \Lambda_0^{(3)} S_4 + \Lambda_1^{(3)} S_3 + \Lambda_2^{(3)} S_2 + \Lambda_3^{(3)} S_1$$
$$= 1 \times 3 + 0 \times 4 + 9 \times 13 + 14 \times 12 = 8$$

$$\delta_4 = 0$$

$$L_r = \delta_r (r - L_{r-1} + \rho) + (1 - \delta_r) L_{r-1}$$
$$L_4 = L_3 = 3$$

$$K_{r-1} = K_3 = 13$$
$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r) K_{r-1}$$
$$K_4 = K_3 = 13$$

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{B}^{(r-1)}(x)$$
$$\Lambda^{(4)}(x) = \Lambda^{(3)}(x) - K_3 \Delta_4 x \mathrm{B}^{(3)}(x)$$
$$= 1 + 9x^2 + 14x^3$$
$$\mathrm{B}^{(r)}(x) = \delta_r \Lambda^{(r-1)}(x) - (1 - \delta_r) x \mathrm{B}^{(r-1)}(x)$$
$$\mathrm{B}^{(4)}(x) = x\mathrm{B}^{(3)}(x)$$
$$= x + 4x^2 + 10x^3$$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{M}^{(r-1)}(x)$$
$$\Gamma^{(4)}(x) = \Gamma^{(3)}(x) - K_3 \Delta_4 x \mathrm{M}^{(3)}(x)$$
$$= 12 + 6x + x^2 + 3x^4 + 3x^5$$
$$\mathrm{M}^{(r)}(x) = \delta_r \Gamma^{(r-1)}(x) - (1 - \delta_r) x \mathrm{M}^{(r-1)}(x)$$
$$\mathrm{M}^{(4)}(x) = \delta_4 \Gamma^{(3)}(x) - (1 - \delta_4) x \mathrm{M}^{(3)}(x)$$
$$= 12x + 8x^2 + 11x^3 + 12x^4 + 13x^5$$

รอบการคำนวณที่ 5:

$$\Delta_r = \sum_{j=0}^{t+\rho} \Lambda_j^{(r-1)} S_{r-j}$$

$$\Delta_5 = \sum_{j=0}^{5} \Lambda_j^{(4)} S_{5-j} = \Lambda_0^{(4)} S_5 + \Lambda_1^{(4)} S_4 + \Lambda_2^{(4)} S_3 + \Lambda_3^{(4)} S_2$$
$$= 1 \times 14 + 2 \times 3 + 1 \times 4 + 9 \times 13 = 3$$

$$\delta_5 = 1$$

$$L_r = \delta_r (r - L_{r-1} + \rho) + (1 - \delta_r) L_{r-1}$$
$$L_5 = 4$$

$$K_{r-1}\Delta_r = K_4 \Delta_5 = 4$$
$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r) K_{r-1}$$
$$K_5 = \frac{1}{3} = 14$$

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{B}^{(r-1)}(x)$$

$$\Lambda^{(5)}(x) = \Lambda^{(4)}(x) - K_4\Delta_5 x \mathrm{B}^{(4)}(x)$$

$$= 1 + 2x + 5x^2 + 10x^3 + 14x^4$$

$$\mathrm{B}^{(r)}(x) = \delta_r \Lambda^{(r-1)}(x) - (1 - \delta_r) x \mathrm{B}^{(r-1)}(x)$$

$$\mathrm{B}^{(5)}(x) = \Lambda^{(4)}(x)$$

$$= 1 + 2x + x^2 + 9x^3$$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{M}^{(r-1)}(x)$$

$$\Gamma^{(5)}(x) = \Gamma^{(4)}(x) - K_4\Delta_5 x \mathrm{M}^{(4)}(x)$$

$$= 12 + 6x + 4x^2 + 6x^3 + 9x^5$$

$$\mathrm{M}^{(r)}(x) = \delta_r \Gamma^{(r-1)}(x) - (1 - \delta_r) x \mathrm{M}^{(r-1)}(x)$$

$$\mathrm{M}^{(5)}(x) = \delta_5 \Gamma^{(4)}(x) - (1 - \delta_5) x \mathrm{M}^{(4)}(x)$$

$$= 12 + 6x + x^2 + 3x^4 + 3x^5$$

รอบการคำนวณที่ 6:

$$\Delta_r = \sum_{j=0}^{t+\rho} \Lambda_j^{(r-1)} S_{r-j}$$

$$\Delta_6 = \sum_{j=0}^{5} \Lambda_j^{(5)} S_{6-j} = \Lambda_0^{(5)} S_6 + \Lambda_1^{(4)} S_5 + \Lambda_2^{(4)} S_4 + \Lambda_3^{(4)} S_3 + \Lambda_4^{(4)} S_2$$

$$= 1 \times 13 + 2 \times 14 + 5 \times 3 + 10 \times 4 + 14 \times 13 = 9$$

$$\delta_6 = 0$$

$$L_r = \delta_r (r - L_{r-1} + \rho) + (1 - \delta_r) L_{r-1}$$

$$L_6 = 4$$

$$K_{r-1}\Delta_r = K_4\Delta_5 = 4$$

$$K_r = \delta_r \Delta_r^{-1} + (1 - \delta_r) K_{r-1}$$

$$K_5 = \frac{1}{3} = 14$$

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - K_{r-1}\Delta_r x \mathrm{B}^{(r-1)}(x)$$

$$\Lambda^{(6)}(x) = \Lambda^{(5)}(x) - K_5\Delta_6 x\mathrm{B}^{(5)}(x)$$
$$= 1 + 5x + 11x^2 + 13x^3 + 14x^4$$

$$\Gamma^{(r)}(x) = \Gamma^{(r-1)}(x) - K_{r-1}\Delta_r x\mathrm{M}^{(r-1)}(x)$$
$$\Gamma^{(6)}(x) = \Gamma^{(5)}(x) - K_5\Delta_6 x\mathrm{M}^{(5)}(x)$$
$$= 12 + 4x + 5x^2 + x^3$$

หรือสุดท้ายแล้วจะได้

$\Lambda(x) = 1 + 5x + 11x^2 + 13x^3 + 14x^4 = 1 + \alpha^8 x + \alpha^7 x^2 + \alpha^{13} x^3 + \alpha^{14} x^4$

$\Gamma(x) = 12 + 4x + 5x^2 + x^3 = \alpha^6 + \alpha^2 x + \alpha^8 x^2 + x^3$

ขั้นตอนที่ 3 หารากของ $\Lambda(x)$ ซึ่ง $\Lambda(x) = \prod(1 - x\alpha^i)$ คือพหุนามระบุตำแหน่งโดยวิธีการ Chien Search รากของพหุนามระบุตำแหน่งก็คือส่วนกลับของค่า α^i เมื่อ i คือตำแหน่งที่เกิดการลบเลือน หรือเกิดความผิดพลาดขึ้น ดังนั้น ในขั้นตอนนี้จึงเพียงแสดงให้เห็นว่า เป็นรากของพหุนามบอกตำแหน่งนั้นจริง

$$\Lambda(\alpha^{-12}) = \Lambda(\alpha^3) = 1 + 5\alpha^3 + 11(\alpha^3)^2 + 13(\alpha^3)^3 + 14(\alpha^3)^4$$
$$= 1 + 14 + 13 + 11 + 9$$
$$= 0$$

$$\Lambda(\alpha^{-11}) = \Lambda(\alpha^4) = 1 + 5\alpha^4 + 11(\alpha^4)^2 + 13(\alpha^4)^3 + 14(\alpha^4)^4$$
$$= 1 + 15 + 1 + 7 + 8$$
$$= 0$$

$$\Lambda(\alpha^{-6}) = \Lambda(\alpha^9) = 1 + 5\alpha^9 + 11(\alpha^9)^2 + 13(\alpha^9)^3 + 14(\alpha^9)^4$$
$$= 1 + 4 + 5$$
$$= 0$$

$$\Lambda(\alpha^{-3}) = \Lambda(\alpha^{12}) = 1 + 5\alpha^{12} + 11(\alpha^{12})^2 + 13(\alpha^{12})^3 + 14(\alpha^{12})^4$$
$$= 1 + 6 + 2 + 3 + 6$$
$$= 0$$

ขั้นตอนที่ 4 หาขนาดความผิดพลาด โดยวิธีการฟอร์เนย์ ซึ่ง $\Gamma(x) = \alpha^6 + \alpha^2 x + \alpha^8 x^2 + x^3$ และ

$\Lambda(x) = 1 + \alpha^8 x + \alpha^7 x^2 + \alpha^{13} x^3 + \alpha^{14} x^4$ ดังนั้น จะได้อนุพันธ์ของ $\Lambda(x)$ ได้ดังนี้ $\Lambda'(x) = \alpha^8 + \alpha^{13} x^2$ ได้จากสมการ $e_i = \frac{\Gamma(\alpha^{-i})}{\Lambda'(\alpha^{-i})}$ ซึ่งจะได้

$$e_i = \frac{\Gamma(\alpha^{-3})}{\Lambda'(\alpha^{-3})} = \frac{\Gamma(\alpha^{12})}{\Lambda'(\alpha^{12})} = \frac{12+9+4+12}{5+11} = \frac{13}{14} = 4$$

$$e_i = \frac{\Gamma(\alpha^{-6})}{\Lambda'(\alpha^{-6})} = \frac{\Gamma(\alpha^{9})}{\Lambda'(\alpha^{9})} = \frac{12+14+14+15}{5+2} = \frac{3}{7} = 10$$

$$e_i = \frac{\Gamma(\alpha^{-11})}{\Lambda'(\alpha^{-11})} = \frac{\Gamma(\alpha^{4})}{\Lambda'(\alpha^{4})} = \frac{12+12+2+15}{9} = \frac{13}{9} = 9$$

$$e_i = \frac{\Gamma(\alpha^{-12})}{\Lambda'(\alpha^{-12})} = \frac{\Gamma(\alpha^{3})}{\Lambda'(\alpha^{3})} = \frac{12+6+9+10}{6} = \frac{9}{6} = 10$$

ขั้นตอนที่ 5 ทำการแก้ไขสัญลักษณ์ที่ผิดพลาดในตำแหน่งที่สอดคล้อง ซึ่งจะได้

$m_3 = r_3 + e_3 = 0 + 4 = 4$

$m_6 = r_6 + c_6 = 0 + 10 = 10$

$m_{11} = r_{11} + e_{11} = 15 + 9 = 6$

$m_{12} = r_{12} + e_{12} = 15 + 10 = 5$

ภาคผนวก ข

คำอธิบายโมดูล VHDL

1) **delay_f.vhd** คือโมดูลที่ใช้เป็นดีเลย์ข้อมูล 1 สัญญาณนาฬิกา โดยมีเอ็นติตี้ดังนี้

```
entity delay_f is port (
        d_in : in std_logic_vector (3 downto 0);
        clk : in std_logic;
        d_out : out std_logic_vector (3 downto 0)
        );
end delay_f;

architecture rtl of delay_f is
begin
        process(clk)
        begin
                if (clk'event and clk ='1') then
                        d_out <= d_in;
                end if;
        end process;
end rtl;
```

2) **gf_arith.vhd** คือฟังก์ชัน การคำนวณเกี่ยวกับสนามกาลัวส์ ประกอบไปด้วยการคูณและส่วนกลับ โดยมีรูปแบบการส่งค่าฟังก์ชัน ดังนี้

2.1) function gfmul_ab (a : std_logic_vector(3 downto 0); b : std_logic_vector(3 downto 0)) return std_logic_vector is

```
variable t: std_logic_vector(6 downto 0) ;
variable gfmul_ab: std_logic_vector(3 downto 0) ;
begin
```

```
t(0) := (a(0) and b(0)) ;
t(1) := (a(0) and b(1)) xor (a(1) and b(0)) ;
t(2) := (a(0) and b(2)) xor (a(1) and b(1)) xor (a(2) and b(0)) ;
t(3) := (a(0) and b(3)) xor (a(1) and b(2)) xor (a(2) and b(1)) xor (a(3) and
  b(0)) ;
t(4) := (a(1) and b(3)) xor (a(2) and b(2)) xor (a(3) and b(1)) ;
t(5) := (a(2) and b(3)) xor (a(3) and b(2)) ;
t(6) := (a(3) and b(3)) ;
gfmul_ab(0) := t(0) xor t(4) ;
gfmul_ab(1) := t(1) xor t(4) xor t(5) ;
gfmul_ab(2) := t(2) xor t(5) xor t(6) ;
gfmul_ab(3) := t(3) xor t(6) ;
   return (gfmul_ab) ;
end gfmul_ab ;
```

2.2) function gf_inv(c :std_logic_vector(3 downto 0)) return std_logic_vector is variable gf_inv : std_logic_vector(3 downto 0);

```
begin
   case c is
    when "0001" => gf_inv := "0001";
    when "0010" => gf_inv := "1001";
    when "0100" => gf_inv := "1101";
    when "1000" => gf_inv := "1111";
    when "0011" => gf_inv := "1110";
    when "0110" => gf_inv := "0111";
    when "1100" => gf_inv := "1010";
    when "1011" => gf_inv := "0101";
    when "0101" => gf_inv := "1011";
    when "1010" => gf_inv := "1100";
    when "0111" => gf_inv := "0110";
```

```
        when "1110" => gf_inv := "0011";
        when "1111" => gf_inv := "1000";
        when "1101" => gf_inv := "0100";
        when "1001" => gf_inv := "0010";
        when others => gf_inv := "0000";
    end case;
    return gf_inv;
end gf_inv;
```

3) gf_square.vhd คือฟังก์ชันยกกำลังสองบนสนามกาลัวส์

```
entity gf_square is port
(
    d_in : in std_logic_vector(7 downto 0);
    d_out : out std_logic_vector(7 downto 0)
);
end entity ;
architecture rtl of gf_square is
begin
    d_out(0) <= d_in(2) xor d_in(0);
    d_out(1) <= d_in(2) ;
    d_out(2) <= d_in(1) xor d_in(3);
    d_out(3) <= d_in(3);
end rtl;
```

4) mux.vhd คือฟังก์ชันมัลติเพล็กเซอร์ 2 อินพุต 1 เอาท์พุต

```
entity mux is port (
    sel : in std_logic;          --เลือกอินพุตหากเป็น 0 จะใช้ in1 หากเป็น 1 จะใช้ in2
    in1 : in std_logic_vector(7 downto 0);
    in2 : in std_logic_vector(7 downto 0);
```

```
        output : out std_logic_vector(7 downto 0)
        );
end mux;
architecture rtl of mux is
begin
        process(sel)
        begin
         --if (clk'event and clk = '1') then
                if (sel = '0') then
                        output <= in1;
                else
                        output <= in2;
                end if;
         --end if;
        end process;
end rtl;
```

5) nomCorr.vhd คือฟังก์ชันการคำนวณขนาดความผิดพลาด และแก้ไขข้อผิดพลาดของข้อมูล มีเอ็นติตี้ดังนี้

```
entity nomCorr is
        port (
        rst_n : in std_logic;       --รีเซ็ตวงจร
        clk : in std_logic;         --สัญญาณนาฬิกา
        v_in : in std_logic_vector(7 downto 0);          -- v_i
        lambda_in : in std_logic_vector(7 downto 0);          -- λ_i
        zeta_in : in std_logic_vector(7 downto 0);       -- β_i
        omega_in : in std_logic_vector(7 downto 0);  -- ω_i
         -- output
         v_corr : out std_logic_vector(7 downto 0)     --
```

```
        );
end nomCorr;
architecture rtl of nomCorr is
--constant
        type rom is array (0 to 14) of std_logic_vector(3 downto 0);
        type idx is array (0 to 15) of std_logic_vector(3 downto 0);
        constant alpha : rom :=
("0001","0010","0100","1000","0011","0110","1100","1011","0101","1010","0111","11
10","1111","1101","1001");
        constant inv_alpha : rom :=
("0001","1001","1101","1111","1110","0111","1010","0101","1011","1100","0110","00
11","1000","0100","0010");
        constant inv_val : idx :=
("0000","0001","1001","1110","1101","1011","0111","0110","1111","0010","1100","01
01","1010","0100","0011","1000");
begin
process (rst_n,clk)
begin
        if (rst_n <= '0') then
         v_out <= "0000";
         v_corr <= "0000";
        elsif (clk'event and clk ='1') then
         if lambda_in = "0000" then
                v_corr <= v_in xor
gfmul_ab(omega_in,inv_val(to_integer(unsigned(zeta_in))));
         else
                v_corr <= v_in;
         end if;
        end if;
end process;
```

```
end rtl;
```

6) **TimeDomain.vhd** คือฟังก์ชันการคำนวณการถอดรหัสรีด – โซโลมอน บนโดเมนเวลาโดยมีเอ็นติตี้ดังนี้

```
entity TimeDomain is
        generic (N : integer := 15;
         r : integer := 1);
        port (
        rst_n : in std_logic;
        clk : in std_logic;
        v_in : in std_logic_vector(7 downto 0);
        lambda_in : in std_logic_vector(7 downto 0);
        beta_in : in std_logic_vector(7 downto 0);
        k_in : in std_logic_vector(7 downto 0);
        u_in : in std_logic_vector(7 downto 0);
        L_in : in std_logic_vector(7 downto 0);
        zeta_in : in std_logic_vector(7 downto 0);
        xi_in : in std_logic_vector(7 downto 0);
        omega_in : in std_logic_vector(7 downto 0);
        mu_in: in std_logic_vector(7 downto 0);
        -- output
         v_out : out std_logic_vector(7 downto 0);
        lambda_out : out std_logic_vector(7 downto 0);
        beta_out  : out std_logic_vector(7 downto 0);
        k_out : out std_logic_vector(7 downto 0);
        u_out : out std_logic_vector(7 downto 0);
        L_out : out std_logic_vector(7 downto 0);
        zeta_out : out std_logic_vector(7 downto 0);
        xi_out : out std_logic_vector(7 downto 0);
```

```
        omega_out : out std_logic_vector(7 downto 0);
        mu_out : out std_logic_vector(7 downto 0)
        );
end TimeDomain;
architecture rtl of TimeDomain is

--constant
        type rom is array (0 to 14) of std_logic_vector(3 downto 0);
        type idx is array (0 to 15) of std_logic_vector(3 downto 0);
        constant alpha : rom :=
("0001","0010","0100","1000","0011","0110","1100","1011","0101","1010","0111","11
10","1111","1101","1001");
        constant inv_alpha : rom :=
("0001","1001","1101","1111","1110","0111","1010","0101","1011","1100","0110","00
11","1000","0100","0010");
        constant inv_val : idx :=
("0000","0001","1001","1110","1101","1011","0111","0110","1111","0010","1100","10
10","1010","0100","0011","1000");

-- signal
        -- v_in delay
        signal v1 : std_logic_vector(3 downto 0);
        signal v2 : std_logic_vector(3 downto 0);
        signal v3 : std_logic_vector(3 downto 0);
        signal v4 : std_logic_vector(3 downto 0);
        signal v5 : std_logic_vector(3 downto 0);
        signal v6 : std_logic_vector(3 downto 0);
        signal v7 : std_logic_vector(3 downto 0);
        signal v8 : std_logic_vector(3 downto 0);
        signal v9 : std_logic_vector(3 downto 0);
```

```
signal v10 : std_logic_vector(3 downto 0);
signal v11 : std_logic_vector(3 downto 0);
signal v12 : std_logic_vector(3 downto 0);
signal v13 : std_logic_vector(3 downto 0);
signal v14 : std_logic_vector(3 downto 0);
signal v15 : std_logic_vector(3 downto 0);
signal v16 : std_logic_vector(3 downto 0);
signal v17 : std_logic_vector(3 downto 0);
signal v_carry : std_logic_vector(3 downto 0);
signal v_carrya : std_logic_vector(3 downto 0);

--lamda_delay
signal lamda1 : std_logic_vector(3 downto 0);
signal lamda2 : std_logic_vector(3 downto 0);
signal lamda3 : std_logic_vector(3 downto 0);
signal lamda4 : std_logic_vector(3 downto 0);
signal lamda5 : std_logic_vector(3 downto 0);
signal lamda6 : std_logic_vector(3 downto 0);
signal lamda7 : std_logic_vector(3 downto 0);
signal lamda8 : std_logic_vector(3 downto 0);
signal lamda9 : std_logic_vector(3 downto 0);
signal lamda10 : std_logic_vector(3 downto 0);
signal lamda11 : std_logic_vector(3 downto 0);
signal lamda12 : std_logic_vector(3 downto 0);
signal lamda13 : std_logic_vector(3 downto 0);
signal lamda14 : std_logic_vector(3 downto 0);
signal lamda15 : std_logic_vector(3 downto 0);
signal lamda16 : std_logic_vector(3 downto 0);
signal lamda17 : std_logic_vector(3 downto 0);
```

```
signal lambda_carry : std_logic_vector(3 downto 0);
signal lambda_carrya : std_logic_vector(3 downto 0);

--beta delay
signal beta1 : std_logic_vector(3 downto 0);
signal beta2 : std_logic_vector(3 downto 0);
signal beta3 : std_logic_vector(3 downto 0);
signal beta4 : std_logic_vector(3 downto 0);
signal beta5 : std_logic_vector(3 downto 0);
signal beta6 : std_logic_vector(3 downto 0);
signal beta7 : std_logic_vector(3 downto 0);
signal beta8 : std_logic_vector(3 downto 0);
signal beta9 : std_logic_vector(3 downto 0);
signal beta10 : std_logic_vector(3 downto 0);
signal beta11 : std_logic_vector(3 downto 0);
signal beta12 : std_logic_vector(3 downto 0);
signal beta13 : std_logic_vector(3 downto 0);
signal beta14 : std_logic_vector(3 downto 0);
signal beta15 : std_logic_vector(3 downto 0);
signal beta16 : std_logic_vector(3 downto 0);
signal beta17 : std_logic_vector(3 downto 0);
signal beta_carry : std_logic_vector(3 downto 0);
signal beta_carrya : std_logic_vector(3 downto 0);
--u delay
signal u1 : std_logic_vector(3 downto 0);
signal u2 : std_logic_vector(3 downto 0);
signal u3 : std_logic_vector(3 downto 0);
signal u4 : std_logic_vector(3 downto 0);
signal u5 : std_logic_vector(3 downto 0);
```

```
signal u6 : std_logic_vector(3 downto 0);
signal u7 : std_logic_vector(3 downto 0);
signal u8 : std_logic_vector(3 downto 0);
signal u9 : std_logic_vector(3 downto 0);
signal u10 : std_logic_vector(3 downto 0);
signal u11 : std_logic_vector(3 downto 0);
signal u12 : std_logic_vector(3 downto 0);
signal u13 : std_logic_vector(3 downto 0);
signal u14 : std_logic_vector(3 downto 0);
signal u15 : std_logic_vector(3 downto 0);
signal u16 : std_logic_vector(3 downto 0);
signal u17 : std_logic_vector(3 downto 0);
signal u_carry : std_logic_vector(3 downto 0);
signal u_carrya : std_logic_vector(3 downto 0);
--L delay
signal L1 : std_logic_vector(3 downto 0);
signal L2 : std_logic_vector(3 downto 0);
signal L3 : std_logic_vector(3 downto 0);
signal L4 : std_logic_vector(3 downto 0);
signal L5 : std_logic_vector(3 downto 0);
signal L6 : std_logic_vector(3 downto 0);
signal L7 : std_logic_vector(3 downto 0);
signal L8 : std_logic_vector(3 downto 0);
signal L9 : std_logic_vector(3 downto 0);
signal L10 : std_logic_vector(3 downto 0);
signal L11 : std_logic_vector(3 downto 0);
signal L12 : std_logic_vector(3 downto 0);
signal L13 : std_logic_vector(3 downto 0);
signal L14 : std_logic_vector(3 downto 0);
```

```
signal L15 : std_logic_vector(3 downto 0);
signal L16 : std_logic_vector(3 downto 0);
signal L17 : std_logic_vector(3 downto 0);
signal L_carry : std_logic_vector(3 downto 0);
signal L_carrya : std_logic_vector(3 downto 0);

--K delay
signal K1 : std_logic_vector(3 downto 0);
signal K2 : std_logic_vector(3 downto 0);
signal K3 : std_logic_vector(3 downto 0);
signal K4 : std_logic_vector(3 downto 0);
signal K5 : std_logic_vector(3 downto 0);
signal K6 : std_logic_vector(3 downto 0);
signal K7 : std_logic_vector(3 downto 0);
signal K8 : std_logic_vector(3 downto 0);
signal K9 : std_logic_vector(3 downto 0);
signal K10 : std_logic_vector(3 downto 0);
signal K11 : std_logic_vector(3 downto 0);
signal K12 : std_logic_vector(3 downto 0);
signal K13 : std_logic_vector(3 downto 0);
signal K14 : std_logic_vector(3 downto 0);
signal K15 : std_logic_vector(3 downto 0);
signal K16 : std_logic_vector(3 downto 0);
signal K17 : std_logic_vector(3 downto 0);
signal K_carry : std_logic_vector(3 downto 0);
signal K_carrya : std_logic_vector(3 downto 0);

--zeta delay
signal zeta1 : std_logic_vector(3 downto 0);
```

```
signal zeta2 : std_logic_vector(3 downto 0);
signal zeta3 : std_logic_vector(3 downto 0);
signal zeta4 : std_logic_vector(3 downto 0);
signal zeta5 : std_logic_vector(3 downto 0);
signal zeta6 : std_logic_vector(3 downto 0);
signal zeta7 : std_logic_vector(3 downto 0);
signal zeta8 : std_logic_vector(3 downto 0);
signal zeta9 : std_logic_vector(3 downto 0);
signal zeta10 : std_logic_vector(3 downto 0);
signal zeta11 : std_logic_vector(3 downto 0);
signal zeta12 : std_logic_vector(3 downto 0);
signal zeta13 : std_logic_vector(3 downto 0);
signal zeta14 : std_logic_vector(3 downto 0);
signal zeta15 : std_logic_vector(3 downto 0);
signal zeta16 : std_logic_vector(3 downto 0);
signal zeta17 : std_logic_vector(3 downto 0);
signal zeta_carry : std_logic_vector(3 downto 0);
signal zeta_carrya : std_logic_vector(3 downto 0);

signal ur : std_logic_vector(3 downto 0);
signal beta_ur : std_logic_vector(3 downto 0);

--xi delay
signal xi1 : std_logic_vector(3 downto 0);
signal xi2 : std_logic_vector(3 downto 0);
signal xi3 : std_logic_vector(3 downto 0);
signal xi4 : std_logic_vector(3 downto 0);
signal xi5 : std_logic_vector(3 downto 0);
signal xi6 : std_logic_vector(3 downto 0);
```

```
signal xi7 : std_logic_vector(3 downto 0);
signal xi8 : std_logic_vector(3 downto 0);
signal xi9 : std_logic_vector(3 downto 0);
signal xi10 : std_logic_vector(3 downto 0);
signal xi11 : std_logic_vector(3 downto 0);
signal xi12 : std_logic_vector(3 downto 0);
signal xi13 : std_logic_vector(3 downto 0);
signal xi14 : std_logic_vector(3 downto 0);
signal xi15 : std_logic_vector(3 downto 0);
signal xi16 : std_logic_vector(3 downto 0);
signal xi17 : std_logic_vector(3 downto 0);
signal xi_carry : std_logic_vector(3 downto 0);
signal xi_carrya : std_logic_vector(3 downto 0);

--omega delay
signal omega1 : std_logic_vector(3 downto 0);
signal omega2 : std_logic_vector(3 downto 0);
signal omega3 : std_logic_vector(3 downto 0);
signal omega4 : std_logic_vector(3 downto 0);
signal omega5 : std_logic_vector(3 downto 0);
signal omega6 : std_logic_vector(3 downto 0);
signal omega7 : std_logic_vector(3 downto 0);
signal omega8 : std_logic_vector(3 downto 0);
signal omega9 : std_logic_vector(3 downto 0);
signal omega10 : std_logic_vector(3 downto 0);
signal omega11 : std_logic_vector(3 downto 0);
signal omega12 : std_logic_vector(3 downto 0);
signal omega13 : std_logic_vector(3 downto 0);
signal omega14 : std_logic_vector(3 downto 0);
```

```
signal omega15 : std_logic_vector(3 downto 0);
signal omega16 : std_logic_vector(3 downto 0);
signal omega17 : std_logic_vector(3 downto 0);
signal omega_carry : std_logic_vector(3 downto 0);
signal omega_carrya : std_logic_vector(3 downto 0);

--mu delay
signal mu1 : std_logic_vector(3 downto 0);
signal mu2 : std_logic_vector(3 downto 0);
signal mu3 : std_logic_vector(3 downto 0);
signal mu4 : std_logic_vector(3 downto 0);
signal mu5 : std_logic_vector(3 downto 0);
signal mu6 : std_logic_vector(3 downto 0);
signal mu7 : std_logic_vector(3 downto 0);
signal mu8 : std_logic_vector(3 downto 0);
signal mu9 : std_logic_vector(3 downto 0);
signal mu10 : std_logic_vector(3 downto 0);
signal mu11 : std_logic_vector(3 downto 0);
signal mu12 : std_logic_vector(3 downto 0);
signal mu13 : std_logic_vector(3 downto 0);
signal mu14 : std_logic_vector(3 downto 0);
signal mu15 : std_logic_vector(3 downto 0);
signal mu16 : std_logic_vector(3 downto 0);
signal mu17 : std_logic_vector(3 downto 0);
signal mu_carry : std_logic_vector(3 downto 0);
signal mu_carrya : std_logic_vector(3 downto 0);

-- delta_r computation
signal m1 : std_logic_vector(3 downto 0);
```

```
        signal m2 : std_logic_vector(3 downto 0);
        signal a1 : std_logic_vector(3 downto 0);
        signal a2 : std_logic_vector(3 downto 0);

        -- delta.k.alpha computation
        signal delta_k_alpha : std_logic_vector(3 downto 0);
        signal alpha_k : std_logic_vector(3 downto 0);
        signal mux1out : std_logic_vector(3 downto 0);
        signal mux2out : std_logic_vector(3 downto 0);
        signal mux3out : std_logic_vector(3 downto 0);
        signal m3 : std_logic_vector(3 downto 0);
        signal m4 : std_logic_vector(3 downto 0);
        signal c1 : std_logic;
        signal c2 : std_logic;
        signal d2temp : std_logic_vector(3 downto 0);
        signal m5 : std_logic_vector(3 downto 0);
        signal m6 : std_logic_vector(3 downto 0);
        signal a5 : std_logic_vector(3 downto 0);

        --clock sync
        signal m3a : std_logic_vector(3 downto 0);
        signal m4a : std_logic_vector(3 downto 0);
        signal m5a : std_logic_vector(3 downto 0);
        signal m6a : std_logic_vector(3 downto 0);

        signal inv_del : std_logic_vector(3 downto 0);
--      signal temp_delta : std_logic_vector(3 downto 0);
        signal delta : std_logic;
--      signal inside_d : std_logic_vector(3 downto 0);
```

```
        -- variable

-- component
        component delay_f
         port (
                d_in : in std_logic_vector (3 downto 0);
                clk : in std_logic;
                d_out : out std_logic_vector (3 downto 0)
                );
        end component;

        component mux
         port (
                sel : in std_logic;
                in1 : in std_logic_vector(3 downto 0);
                in2 : in std_logic_vector(3 downto 0);
                output : out std_logic_vector(3 downto 0)
         );
        end component;

begin
--data delay

        vd1 : delay_f port map(v_in,clk,v1);
        vd2 : delay_f port map(v1,clk,v2);
        vd3 : delay_f port map(v2,clk,v3);
        vd4 : delay_f port map(v3,clk,v4);
```

```
        vd5 : delay_f port map(v4,clk,v5);
        vd6 : delay_f port map(v5,clk,v6);
        vd7 : delay_f port map(v6,clk,v7);
        vd8 : delay_f port map(v7,clk,v8);
        vd9 : delay_f port map(v8,clk,v9);
        vd10 : delay_f port map(v9,clk,v10);
        vd11 : delay_f port map(v10,clk,v11);
        vd12 : delay_f port map(v11,clk,v12);
        vd13 : delay_f port map(v12,clk,v13);
        vd14 : delay_f port map(v13,clk,v14);
        vd15 : delay_f port map(v14,clk,v15);
        vd16 : delay_f port map(v15,clk,v16);
        vd17 : delay_f port map(v16,clk,v17);
        vd18 : delay_f port map(v17,clk,v_carry);
--lamda delay
        lam1 : delay_f port map(lambda_in,clk,lamda1);
        lam2 : delay_f port map(lamda1,clk,lamda2);
        lam3 : delay_f port map(lamda2,clk,lamda3);
        lam4 : delay_f port map(lamda3,clk,lamda4);
        lam5 : delay_f port map(lamda4,clk,lamda5);
        lam6 : delay_f port map(lamda5,clk,lamda6);
        lam7 : delay_f port map(lamda6,clk,lamda7);
        lam8 : delay_f port map(lamda7,clk,lamda8);
        lam9 : delay_f port map(lamda8,clk,lamda9);
        lam10 : delay_f port map(lamda9,clk,lamda10);
        lam11 : delay_f port map(lamda10,clk,lamda11);
        lam12 : delay_f port map(lamda11,clk,lamda12);
        lam13 : delay_f port map(lamda12,clk,lamda13);
        lam14 : delay_f port map(lamda13,clk,lamda14);
```

```
        lam15 : delay_f port map(lamda14,clk,lamda15);
        lam16 : delay_f port map(lamda15,clk,lamda16);
        lam17 : delay_f port map(lamda16,clk,lamda17);
        lam18 : delay_f port map(lamda17,clk,lambda_carry);
--beta delay
        ubeta1 : delay_f port map(beta_in,clk,beta1);
        ubeta2 : delay_f port map(beta1,clk,beta2);
        ubeta3 : delay_f port map(beta2,clk,beta3);
        ubeta4 : delay_f port map(beta3,clk,beta4);
        ubeta5 : delay_f port map(beta4,clk,beta5);
        ubeta6 : delay_f port map(beta5,clk,beta6);
        ubeta7 : delay_f port map(beta6,clk,beta7);
        ubeta8 : delay_f port map(beta7,clk,beta8);
        ubeta9 : delay_f port map(beta8,clk,beta9);
        ubeta10 : delay_f port map(beta9,clk,beta10);
        ubeta11 : delay_f port map(beta10,clk,beta11);
        ubeta12 : delay_f port map(beta11,clk,beta12);
        ubeta13 : delay_f port map(beta12,clk,beta13);
        ubeta14 : delay_f port map(beta13,clk,beta14);
        ubeta15 : delay_f port map(beta14,clk,beta15);
        ubeta16 : delay_f port map(beta15,clk,beta16);
        ubeta17 : delay_f port map(beta16,clk,beta17);
        ubeta18 : delay_f port map(beta17,clk,beta_carry);
--u delay
        uu1 : delay_f port map(u_in,clk,u1);
        uu2 : delay_f port map(u1,clk,u2);
        uu3 : delay_f port map(u2,clk,u3);
        uu4 : delay_f port map(u3,clk,u4);
        uu5 : delay_f port map(u4,clk,u5);
```

```
        uu6 : delay_f port map(u5,clk,u6);
        uu7 : delay_f port map(u6,clk,u7);
        uu8 : delay_f port map(u7,clk,u8);
        uu9 : delay_f port map(u8,clk,u9);
        uu10 : delay_f port map(u9,clk,u10);
        uu11 : delay_f port map(u10,clk,u11);
        uu12 : delay_f port map(u11,clk,u12);
        uu13 : delay_f port map(u12,clk,u13);
        uu14 : delay_f port map(u13,clk,u14);
        uu15 : delay_f port map(u14,clk,u15);
        uu16 : delay_f port map(u15,clk,u16);
        uu17 : delay_f port map(u16,clk,u17);
        uu18 : delay_f port map(u17,clk,u_carry);
--L delay
        uL1 : delay_f port map(L_in,clk,L1);
        uL2 : delay_f port map(L1,clk,L2);
        uL3 : delay_f port map(L2,clk,L3);
        uL4 : delay_f port map(L3,clk,L4);
        uL5 : delay_f port map(L4,clk,L5);
        uL6 : delay_f port map(L5,clk,L6);
        uL7 : delay_f port map(L6,clk,L7);
        uL8 : delay_f port map(L7,clk,L8);
        uL9 : delay_f port map(L8,clk,L9);
        uL10 : delay_f port map(L9,clk,L10);
        uL11 : delay_f port map(L10,clk,L11);
        uL12 : delay_f port map(L11,clk,L12);
        uL13 : delay_f port map(L12,clk,L13);
        uL14 : delay_f port map(L13,clk,L14);
        uL15 : delay_f port map(L14,clk,L15);
```

```
uL16 : delay_f port map(L15,clk,L16);
uL17 : delay_f port map(L16,clk,L17);
uL18 : delay_f port map(L17,clk,L_carry);

--K delay
uK1 : delay_f port map(k_in,clk,K1);
uK2 : delay_f port map(K1,clk,K2);
uK3 : delay_f port map(K2,clk,K3);
uK4 : delay_f port map(K3,clk,K4);
uK5 : delay_f port map(K4,clk,K5);
uK6 : delay_f port map(K5,clk,K6);
uK7 : delay_f port map(K6,clk,K7);
uK8 : delay_f port map(K7,clk,K8);
uK9 : delay_f port map(K8,clk,K9);
uK10 : delay_f port map(K9,clk,K10);
uK11 : delay_f port map(K10,clk,K11);
uK12 : delay_f port map(K11,clk,K12);
uK13 : delay_f port map(K12,clk,K13);
uK14 : delay_f port map(K13,clk,K14);
uK15 : delay_f port map(K14,clk,K15);
uK16 : delay_f port map(K15,clk,K16);
uK17 : delay_f port map(K16,clk,K17);
uK18 : delay_f port map(K17,clk,K_carry);

--zeta delay
uzeta1 : delay_f port map(zeta_in,clk,zeta1);
uzeta2 : delay_f port map(zeta1,clk,zeta2);
uzeta3 : delay_f port map(zeta2,clk,zeta3);
uzeta4 : delay_f port map(zeta3,clk,zeta4);
```

```
        uzeta5 : delay_f port map(zeta4,clk,zeta5);
        uzeta6 : delay_f port map(zeta5,clk,zeta6);
        uzeta7 : delay_f port map(zeta6,clk,zeta7);
        uzeta8 : delay_f port map(zeta7,clk,zeta8);
        uzeta9 : delay_f port map(zeta8,clk,zeta9);
        uzeta10 : delay_f port map(zeta9,clk,zeta10);
        uzeta11 : delay_f port map(zeta10,clk,zeta11);
        uzeta12 : delay_f port map(zeta11,clk,zeta12);
        uzeta13 : delay_f port map(zeta12,clk,zeta13);
        uzeta14 : delay_f port map(zeta13,clk,zeta14);
        uzeta15 : delay_f port map(zeta14,clk,zeta15);
        uzeta16 : delay_f port map(zeta15,clk,zeta16);
        uzeta17 : delay_f port map(zeta16,clk,zeta17);
        uzeta18 : delay_f port map(zeta17,clk,zeta_carry);
--xi delay
        uxi1 : delay_f port map(xi_in,clk,xi1);
        uxi2 : delay_f port map(xi1,clk,xi2);
        uxi3 : delay_f port map(xi2,clk,xi3);
        uxi4 : delay_f port map(xi3,clk,xi4);
        uxi5 : delay_f port map(xi4,clk,xi5);
        uxi6 : delay_f port map(xi5,clk,xi6);
        uxi7 : delay_f port map(xi6,clk,xi7);
        uxi8 : delay_f port map(xi7,clk,xi8);
        uxi9 : delay_f port map(xi8,clk,xi9);
        uxi10 : delay_f port map(xi9,clk,xi10);
        uxi11 : delay_f port map(xi10,clk,xi11);
        uxi12 : delay_f port map(xi11,clk,xi12);
        uxi13 : delay_f port map(xi12,clk,xi13);
        uxi14 : delay_f port map(xi13,clk,xi14);
```

```
        uxi15 : delay_f port map(xi14,clk,xi15);
        uxi16 : delay_f port map(xi15,clk,xi16);
        uxi17 : delay_f port map(xi16,clk,xi17);
        uxi18 : delay_f port map(xi17,clk,xi_carry);
----omega delay
        uomega1 : delay_f port map(omega_in,clk,omega1);
        uomega2 : delay_f port map(omega1,clk,omega2);
        uomega3 : delay_f port map(omega2,clk,omega3);
        uomega4 : delay_f port map(omega3,clk,omega4);
        uomega5 : delay_f port map(omega4,clk,omega5);
        uomega6 : delay_f port map(omega5,clk,omega6);
        uomega7 : delay_f port map(omega6,clk,omega7);
        uomega8 : delay_f port map(omega7,clk,omega8);
        uomega9 : delay_f port map(omega8,clk,omega9);
        uomega10 : delay_f port map(omega9,clk,omega10);
        uomega11 : delay_f port map(omega10,clk,omega11);
        uomega12 : delay_f port map(omega11,clk,omega12);
        uomega13 : delay_f port map(omega12,clk,omega13);
        uomega14 : delay_f port map(omega13,clk,omega14);
        uomega15 : delay_f port map(omega14,clk,omega15);
        uomega16 : delay_f port map(omega15,clk,omega16);
        uomega17 : delay_f port map(omega16,clk,omega17);
        uomega18 : delay_f port map(omega17,clk,omega_carry);
----mu delay
        umu1 : delay_f port map(mu_in,clk,mu1);
        umu2 : delay_f port map(mu1,clk,mu2);
        umu3 : delay_f port map(mu2,clk,mu3);
        umu4 : delay_f port map(mu3,clk,mu4);
        umu5 : delay_f port map(mu4,clk,mu5);
```

```
        umu6 : delay_f port map(mu5,clk,mu6);
        umu7 : delay_f port map(mu6,clk,mu7);
        umu8 : delay_f port map(mu7,clk,mu8);
        umu9 : delay_f port map(mu8,clk,mu9);
        umu10 : delay_f port map(mu9,clk,mu10);
        umu11 : delay_f port map(mu10,clk,mu11);
        umu12 : delay_f port map(mu11,clk,mu12);
        umu13 : delay_f port map(mu12,clk,mu13);
        umu14 : delay_f port map(mu13,clk,mu14);
        umu15 : delay_f port map(mu14,clk,mu15);
        umu16 : delay_f port map(mu15,clk,mu16);
        umu17 : delay_f port map(mu16,clk,mu17);
        umu18 : delay_f port map(mu17,clk,mu_carry);
--mux and m3
        mux1 : mux port map (c2,m4,m3,delta_k_alpha);
        mux2 : mux port map (c1,m2,delta_k_alpha,mux2out);
        mux3 : mux port map (c1,k_in,beta17,mux3out);

        muxbeta_ur : mux port map (ur(0),"0000",beta17,beta_ur);
--other mux

-- process
delta_r_calc : process(rst_n,clk) -- delta r calculation
        variable count_clk : integer range 0 to 2*N+2 := 0;
        variable alpha_u : std_logic_vector(3 downto 0);
        --variable delta : std_logic;
begin
        a1 <= m2 xor m1;
        a5 <= xi17 xor beta_ur;
```

```
 if (rst_n <= '0') then
 lambda_out <= "0000";
 beta_out  <= "0000";
elsif (clk'event and clk ='1' and count_clk < N) then
 m1 <= gfmul_ab(v_in,lambda_in);
 m2 <= gfmul_ab(a2,alpha(r));
 c1 <= '0';
 c2 <= '0';
 count_clk := count_clk +1;
elsif (clk'event and clk ='0' and count_clk < N) then
 a2 <= a1;
elsif (clk'event and clk ='1' and count_clk = N ) then
 m4 <= gfmul_ab(mux2out,mux3out);
 inv_del <= gf_inv(m2);
 m5 <= gfmul_ab(delta_k_alpha,a5);
 m6 <= gfmul_ab(delta_k_alpha,mu_carry);
 count_clk := count_clk +1;
 if (a1 /=0) and (to_integer(unsigned(L15+L15)) <= r-1) then --delta == 1
          delta <= '1';
          ur <= u14 + "0001";
 else
          delta <= '0';
          ur <= "0001";
 end if;
elsif (clk'event and clk ='1' and count_clk > N) then --output
 c1 <= '1';
 c2 <= '1';
 m3 <= gfmul_ab(delta_k_alpha,alpha(to_integer(unsigned(ur))));
 m4 <= gfmul_ab(mux2out,mux3out);
```

```
 m5 <= gfmul_ab(delta_k_alpha,a5);
 m6 <= gfmul_ab(delta_k_alpha,mu17);
 if (count_clk = 2*N+3) then
        count_clk := 0;
 else
        count_clk := count_clk +1;
 end if;
elsif (clk'event and clk ='0') then --and count_clk > N) then --output
 m3a <= m3;
 m4a <= m4;
 m5a <= m5;
 m6a <= m6;
 v_carrya <= v_carry;
 lambda_carrya <= lambda_carry;
 beta_carrya <= beta_carry;
 zeta_carrya <= zeta_carry;
 xi_carrya <= xi_carry;
 omega_carrya <= omega_carry;
 mu_carrya <= mu_carry;
 K_carrya <= K_carry;
 L_carrya <= L_carry;
 u_carrya <= u_carry;
end if;
if (count_clk > N+2 ) then
 lambda_out <= lambda_carrya xor m4a;
 zeta_out <= m5a xor zeta_carrya;
 omega_out <= m6a xor omega_carrya;
 v_out <= v_carrya;
 if (delta = '1') then --delta == 1
```

```
                beta_out <= lambda_carrya;
                k_out <= inv_del;
                --ur <= "0001";
                u_out <= ur;
                L_out <= r-L1;
                xi_out <= zeta_carrya;
                mu_out <= omega_carrya;
            else
                beta_out <= beta_carrya;
                k_out <= k_carrya;
                --ur <= u_carry + "0001";
                u_out <= ur;
                L_out <= L_carrya;
                xi_out <= xi_carrya;
                mu_out <= mu_carrya;
            end if;
        else
            lambda_out <= "0000";
            beta_out <= "0000";
            k_out <= "0000";
            u_out <= "0001";
            L_out <= "0000";
            zeta_out <= "0000";
            xi_out <= "0000";
            mu_out <= "0000";
            omega_out <= "0001";
        end if;
end process;
end rtl;
```

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายปรเมตต์ ปราสาร
วัน เดือน ปีเกิด	25 สิงหาคม 2531
สถานที่เกิด	จังหวัดร้อยเอ็ด
ที่อยู่	45 หมู่ที่ 2 ตำบลน้ำคำใหญ่ อำเภอเมือง จังหวัดยโสธร 35000
โทรศัพท์	08-3365-8502
ประวัติการศึกษา	2546 – 2549 มัธยมศึกษาตอนปลาย โรงเรียนยโสธรพิทยาคม
	2549 – 2553 วิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ความชำนาญเฉพาะด้าน	1) การเขียนโปรแกรมภาษาซี
	2) การออกแบบระบบสมองกลฝังตัว

ประสบการณ์การทำงาน และผลงานวิจัย

พ.ศ. 2552	ฝึกงานที่บริษัท ไทยเทล เอ็นจิเนียริง จำกัด ในตำแหน่งวิศวกรออกแบบซอฟต์แวร์
พ.ศ. 2556	ได้รับการตีพิมพ์งานวิจัยในหัวข้อ An Application of ECC for Reliability Improvement of a HDD Testing System