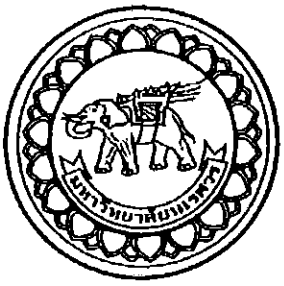


การจำลองการเพิ่มสมรรถนะของการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด โดยการนำเอาอินเทอร์ลีฟวิงมาใช้ร่วม

SIMULATION THE AUGMENTATION TO INCREASE APTITUDE OF CHANNEL CODING BY HAMMING CODE AND INTERLEAVING

นายมโนรัตน์ จันทร์คำ รหัส 47364013

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร
ปีการศึกษา 2550

ใบรับรองโครงงานวิศวกรรม

หัวข้อโครงงาน	**การจำลองการเพิ่มสมรรถนะของการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดโดยการนำเอาอินเทอร์ลีฟวิงมาใช้ร่วม**
ผู้ดำเนินโครงงาน	นายมโนรัตน์ จันทร์คำ รหัส 47364013
อาจารย์ที่ปรึกษา	ดร.อัครพันธ์ วงศ์กังแห
สาขาวิชา	วิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์
ปีการศึกษา	2550

.....

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร อนุมัติให้โครงงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า

คณะกรรมการสอบโครงงานวิศวกรรม

.....ประธานกรรมการ

(ดร.อัครพันธ์ วงศ์กังแห)

.....กรรมการ

(ดร.ชัยรัตน์ พิมทอง)

.....กรรมการ

(อาจารย์ ปิยคนัย ภาชนะพรรณ์)

หัวข้อโครงงาน	การจำลองการเพิ่มสมรรถนะของการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดโดยการนำเอาอินเทอร์ลีฟวิงมาใช้ร่วม
ผู้ดำเนินโครงงาน	นายมโนรัตน์ จันทร์คำ รหัส 47364013
อาจารย์ที่ปรึกษา	ดร.อัครพันธ์ วงศ์กังแห
สาขาวิชา	วิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์
ปีการศึกษา	2550

บทคัดย่อ

โครงงานนี้เป็นการศึกษาและพัฒนาโปรแกรมการจำลองการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดโดยการนำเอาอินเทอร์ลีฟวิงมาใช้ร่วมสำหรับการรันบนโปรแกรม MATLAB 7.0 โดยโปรแกรมสามารถแก้ไขข้อมูลที่มีการผิดพลาดหลายบิตติดต่อกันไม่เกิน 7 บิตติดต่อกัน โดยนำข้อมูลที่ได้จากผู้ใช้ หรือข้อมูลที่ได้เก็บไว้ในโปรแกรม มาทำการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดแล้วนำไปผ่านการทำอินเทอร์ลีฟวิง แล้วทำให้เกิดความผิดพลาด 7 บิตติดต่อกันก่อนที่จะถอดรหัสช่องสัญญาณ และแก้ไขข้อมูลที่ผิดพลาดให้กลับมาเป็นข้อมูลต้นฉบับ คือเหมือนกับก่อนที่จะทำการเข้ารหัสช่องสัญญาณ

ผลที่ได้จากการทำโครงงานนี้ คือ ได้โปรแกรมที่สามารถเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด, อินเทอร์ลีฟวิง, โปรแกรมที่ทำให้ข้อมูลเกิดความผิดพลาด, ทำอินเทอร์ลีฟวิงกลับ, ถอดรหัสแฮมมิงโค้ด ที่ทำให้ได้ผลของข้อมูลที่ถูกต้อง และแบบจำลองสัญญาณพัลส์

Project Title	Simulation the augmentation to increase aptitude of channel coding by Hamming code and Interleaving
Name	Mr. Manorat Chankham ID. 47364013
Project Advisor	Dr. Akaraphunt Vongkunghae
Major	Electrical Engineering
Department	Electrical and Computer Engineering
Academic Year	2007

.....

ABSTRACT

This project is studied and developed a program simulate the Hamming code channel coding and Interleaving for Matlab 7.0. Program that can 7 bits burst error-correction. Program received data from user or memory in program. Data into channel coding, make 7 bits error, error-correction

The result of this project is the program that can Hamming code channel coding, Interleaving, make 7 bits error, De-Interleaving, De-Hamming code influence burst error - correction and Pulse signal model

กิตติกรรมประกาศ

การทำโครงงานฉบับนี้ สำเร็จลุล่วงได้ด้วยดีเพราะได้รับความกรุณาจาก ดร.อัครพันธ์ วงศ์กังแห อาจารย์ที่ปรึกษา และคณะกรรมการทุกท่าน ที่ได้กรุณาให้คำแนะนำปรึกษา ตลอดจนผู้ที่ได้ช่วยตรวจสอบแก้ไข และให้ความรู้แก่ผู้ดำเนินโครงงานจนโครงงานนี้เสร็จสมบูรณ์ และผู้ดำเนินโครงงาน ขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้

ขอกราบขอบพระคุณ คณาจารย์ผู้สอน ประจำภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ ทุกท่านที่ให้ความรู้ หลักการทำงานอย่างเป็นระบบ แนวคิด แนวทางแก้ไขปัญหา อย่างมีหลักการและเป็นระบบในแต่ละวิชาตลอดหลักสูตรของการศึกษา

ผลการทำโครงงานฉบับนี้ คงเป็นประโยชน์กับผู้ที่สนใจศึกษา หากมีความผิดพลาดประการใด ผู้ศึกษาขออภัยมา ณ ที่นี้

มโนรัตน์ จันทร์คำ

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

รูปที่ หน้า

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

ในการสื่อสารดิจิตอล [1] การส่งสัญญาณดิจิตอลโดยทั่วไป มักจะเกิดปัญหาความผิดเพี้ยนของ รูปสัญญาณทำให้เกิดความผิดพลาดขึ้นทำให้ภาครับได้รับข้อมูลที่ไม่ถูกต้องเนื่องจากคุณสมบัติที่ไม่เป็นอุดมคติของช่องสัญญาณเองหรือเนื่องจากผลกระทบของสัญญาณรบกวนภายนอกในรูปแบบต่างๆไม่ว่าจะเป็น การรบกวนกันระหว่างสายส่ง การรบกวนจากคลื่นจากโทรทัศน์ การลบกวนจากคลื่นวิทยุ เป็นต้น ปัญหาเหล่านี้อาจส่งผลให้ข้อมูลดิจิตอล(digital) ที่ได้รับ มีความผิดพลาดเกิดขึ้น ด้วยเหตุนี้ระบบสื่อสารที่ใช้ในปัจจุบันต้องการให้เกิดความผิดพลาดในการส่งข้อมูลต่ำ โดยทั่วไปมักจะนำข้อมูลดิจิตอลที่ต้องการจะส่งไปผ่านกระบวนการเข้ารหัสช่องสัญญาณ(channel coding) [1] ก่อนที่จะส่งออกไป เนื่องจากต้องการให้มีความผิดพลาดที่จะเกิดขึ้นน้อยลง และอยู่ในระดับที่ยอมรับได้ ในการเข้ารหัสช่องสัญญาณนั้นจะช่วยให้ภาครับมีความสามารถในการตรวจจับความผิดพลาดที่เกิดขึ้นได้(error detection) [2-5] และอาจจะรวมไปถึงการที่ภาครับจะสามารถแก้ไขบิตข้อมูลที่ผิดพลาดได้(error correction) [2-5] ซึ่งความผิดพลาดที่จะเกิดขึ้นนั้นมีหลายแบบ เช่น ข้อมูลเกิดความผิดพลาดหนึ่งบิต(Single bit error) [5] ผิดหลายบิต และผิดหลายบิตติดต่อกัน(burst errors) [6-7, 11] ในที่นี้เราจะกล่าวถึงกระบวนการในการที่จะแก้ไขความผิดพลาดที่เกิดขึ้นแบบหลายบิตติดต่อกันซึ่งมักจะเกิดขึ้นในระบบโทรศัพท์เคลื่อนที่เนื่องจากการผิดพลาดแบบหลายบิตติดต่อกันนั้นเป็นรูปแบบที่มักจะเกิดขึ้นบ่อยครั้งในการสื่อสารในระบบโทรศัพท์เคลื่อนที่ เช่น การที่สัญญาณถูกบดบังด้วยวัตถุที่อยู่ใกล้เคียงไปชั่วขณะ ทำให้บิตข้อมูลที่ได้รับเกิดความผิดพลาดอย่างต่อเนื่อง

การผิดพลาดหลายบิตติดต่อกันนั้นเราจะนำวิธีการอินเทอร์ลีฟวิง(Interleaving) [1, 7-8] มาใช้ซึ่งเป็นวิธีการหนึ่งที่ได้มีการนำมาใช้งานกันเพื่อใช้แก้ปัญหาของการผิดพลาดของข้อมูลแบบหลายบิตติดต่อกัน และเพื่อให้สามารถแก้ไขบิตผิดพลาดด้วยเราได้มีการนำเอาการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด(Hamming code) [1-5, 9] มาใช้ร่วมกันซึ่งแฮมมิงโค้ดมีความสามารถในการตรวจจับและแก้ไขข้อมูลที่ผิดพลาดขนาดหนึ่งบิต(Single bit error) [5] ให้มีความถูกต้องได้

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อเพิ่มสมรรถนะของการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดด้วยอินเทอร์ลีฟวิง ให้สามารถแก้ไขความผิดพลาดแบบหลายบิตติดต่อกัน(burst errors) ไม่เกิน 7 บิตได้
2. เพื่อพัฒนาฐานความรู้ เกี่ยวกับการสื่อสารข้อมูลดิจิตอล ในหัวข้อการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code) และอินเทอร์ลีฟวิง(interleaving)
3. เพื่อพัฒนาทักษะ และลอจิก ในการเขียนโปรแกรมโดยใช้โปรแกรมแมทแลบ

1.3 ขอบเขตโครงงาน

1. โปรแกรมมีความสามารถในการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code)ได้
2. โปรแกรมมีความสามารถในการอินเทอร์ลีฟวิง(interleaving) ได้
3. โปรแกรมมีความสามารถในการใช้งานร่วมกันระหว่างอินเทอร์ลีฟวิง(interleaving) กับการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code)
4. โปรแกรมมีความสามารถในการแก้ไขข้อมูลแบบหลายบิตติดต่อกัน(burst errors)
5. โปรแกรมสามารถแสดงผลการเข้ารหัสช่องสัญญาณ และทำอินเทอร์ลีฟวิงได้

1.4 ขั้นตอนการดำเนินงาน

1.4.1 ศึกษาหลักการ

1. ศึกษาหลักการสื่อสารข้อมูลดิจิตอล
2. ศึกษาวิธีการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด(hamming code)
3. ศึกษาหลักการของอินเทอร์ลีฟวิง(interleaving)
4. ศึกษาการใช้โปรแกรมแมทแลบ

1.4.2 การออกแบบ

1. ออกแบบโฟชาร์สของการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code)
2. ออกแบบโฟชาร์สของอินเทอร์ลีฟวิง(interleaving)
3. ออกแบบโฟชาร์สในการใช้งานร่วมกันระหว่างอินเทอร์ลีฟวิง(interleaving)กับการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code)

1.4.3 การเขียนโปรแกรม

1. เขียนโปรแกรมการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code)
2. เขียนโปรแกรมอินเทอร์ลีฟวิง(interleaving)
3. เขียนโปรแกรมใช้งานร่วมกันระหว่างอินเทอร์ลีฟวิง(interleaving)กับการเข้ารหัสช่องสัญญาณแบบแฮมมิง(hamming code)

1.4.4 การทดสอบการทำงานจริง

1. ทดสอบการทำงานของโปรแกรมที่ใช้เข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด (hamming code)
2. ทดสอบการทำงานโปรแกรมอินเทอร์ลีฟวิง(interleaving)
3. ทดสอบการทำงานโปรแกรมใช้งานร่วมกันระหว่างอินเทอร์ลีฟวิง(interleaving)กับการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด(hamming code)

1.4.5 ตรวจสอบและแก้ไขข้อผิดพลาดของโปรแกรม

1.4.6 รวบรวมข้อมูลทั้งหมดเข้ารูปเล่มพร้อมรายงาน

ตารางที่ 1.1 ตารางแสดงระยะเวลาของขั้นตอนการดำเนินงาน

กิจกรรม	ปี 2550				ปี 2551	
	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.
1. ศึกษาหลักการและทฤษฎีพื้นฐานของระบบดิจิตอล	←	—	→			
2. ศึกษาการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด และวิธีการทำอินเตอร์ลีฟวิง		←	—	→		
3. การออกแบบโปรแกรม		←	—	→		
4. เขียนโปรแกรม		←	—	→		
5. การทดสอบการทำงานของโปรแกรม			←	—	→	
6. ตรวจสอบและแก้ไขโปรแกรม				←	—	→
7. รวบรวมข้อมูลทั้งหมดเข้ารูปเล่มพร้อมรายงาน					←	— →

1.5 ผลที่คาดว่าจะได้รับ

1. การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดที่ใช้อินเทอร์ลีฟวิงร่วม สามารถแก้ไขการผิดพลาดหลายบิตติดต่อกัน 7 บิตได้
2. มีการพัฒนาฐานความรู้ และสร้างความรู้ใหม่ เกี่ยวกับการสื่อสารข้อมูลดิจิตอล ในหัวข้อการเข้ารหัสช่องสัญญาณแบบแฮมมิง อินเทอร์ลีฟวิง และการเข้ารหัสช่องสัญญาณแบบต่างๆ
3. เกิดการพัฒนาทักษะในการใช้โปรแกรมแมทแลบ และลอจิกในการเขียนโปรแกรม

1.6 งบประมาณ

1. ค่าเอกสารประกอบการทำโครงงาน เป็นเงิน 600 บาท
2. ค่าถ่ายเอกสารเข้าเล่ม เป็นเงิน 400 บาท

รวมเป็นเงินทั้งสิ้น 1,000 บาท (หนึ่งพันบาทถ้วน)

บทที่ 2
หลักการและทฤษฎี

2.1 การเข้ารหัสช่องสัญญาณ

2.1.1 ประเภทของการเข้ารหัสช่องสัญญาณ

1. การเข้ารหัสแบบบล็อก (Block Codes)

การเข้ารหัสแบบบล็อก(Block Codes) [1] จะแบ่งบิตข้อมูลที่จะทำการเข้ารหัสออกเป็นกลุ่มหรือที่เรียกว่าบล็อกขนาด k บิต จากนั้นบิตข้อมูลแต่ละบล็อกก็จะถูกแปลงให้กลายตำเป็นรหัส (code word) ที่มีความยาวเท่ากับ n บิต โดยที่ n > k ดังนั้นจึงมักเรียกการเข้ารหัสนี้ว่า (n,k) โดยปกติแล้วชุดรหัสที่ได้จากการเข้ารหัสนั้นจะยังคงประกอบด้วยส่วนของบิตข้อมูลเดิม k บิต และส่วนของบิตพิเศษที่เพิ่มเข้าไปอีก n-k บิตหรือที่เรียกว่าบิตเช็ก (check bit)เพื่อใช้สำหรับตรวจสอบว่ามีความผิดพลาดในบิตข้อมูลในระหว่างที่ส่งผ่านช่องสัญญาณหรือไม่ ณ ที่ภาครับก็จะมีวงจรที่ทำหน้าที่ถอดรหัสเพื่อดึงบิตข้อมูลเดิมออกมา พร้อมกันนั้นก็จะให้ค่าที่เรียกว่า ซินโดรม (syndrome) ออกมาด้วย โดยค่าซินโดรมนั้นมีไว้สำหรับบ่งบอกว่ามีความผิดพลาดเกิดขึ้นในข้อมูลหรือไม่ หรืออาจใช้ในการบ่งบอกถึงตำแหน่งของบิตด้วย

2. การเข้ารหัสแบบคอนโวลูชั่น (Convolution Codes)

การเข้ารหัสคอนโวลูชั่น(Convolution Codes) [1, 8] แตกต่างจากรหัสแบบบล็อกตรงที่ ข้อมูลที่จะเข้ารหัสไม่จำเป็นต้องนำมาแบ่งออกเป็นบล็อกความยาวตายตัวก่อนที่จะนำไปผ่านกระบวนการเข้ารหัสจะดำเนินต่อไปจนกว่าจะหยุดการป้อนข้อมูลเข้าไป ฉะนั้นจุดแตกต่างที่สำคัญคือ รหัสคอนโวลูชั่นจะไม่นิยามชุดรหัสในรูปของ (n,k) เนื่องจากไม่มีการระบุขอบเขตความยาวของข้อมูลที่จะทำการเข้ารหัสที่แน่นอน การนิยามคุณสมบัติของรหัสคอนโวลูชั่นจึงแสดงในรูปของอัตราส่วนการเข้ารหัส เช่น 1/n แทนกล่าวคือเมื่อเราป้อนข้อมูล 1 บิตเข้าสู่วงจรเข้ารหัส จะได้เป็นรหัสที่มีความยาวเพิ่มขึ้น n เท่า อัตราส่วนการเข้ารหัสอาจมีค่าที่ต่างไปจากนี้ได้ เช่น การป้อนข้อมูล 2 บิตและให้ผลเป็นรหัสที่มีความยาว 3 บิต อัตราส่วนการเข้ารหัสในกรณีนี้มีค่าเท่ากับ 2/3

2.2 การตรวจสอบและแก้ไขความผิดพลาดของข้อมูล

ในการรับส่งข้อมูลผ่านช่องสัญญาณหรือเครือข่ายนั้น จำเป็นอย่างยิ่งที่ข้อมูลจากต้นทางที่ถูกส่งไปยังปลายทางนั้นจะต้องมีความถูกต้องสมบูรณ์เหมือนเดิมทุกประการ แต่การเดินทางของข้อมูลไปยังปลายทางนั้น ยังมีปัจจัยอื่นๆ อีกหลายประการที่สามารถส่งผลให้ข้อมูลที่ถูกส่งออกไป

นั้นเกิดความผิดพลาด(error) หรือผิดเพี้ยนไปจากเดิม ดังนั้นจึงต้องมีการตรวจสอบข้อผิดพลาด (error detection) [2-5] และทำการแก้ไขข้อมูลในส่วนที่เกิดความผิดพลาด(correction) [2-5] ด้วย

2.2.1 แหล่งกำเนิดของความผิดพลาด และวิธีป้องกัน (Sources of errors and prevention)

เหตุผลหลักๆที่ทำให้เกิดความผิดพลาดขึ้นนั้นจะเกิดจากการที่มีสัญญาณเข้ามารบกวนสัญญาณที่เราได้ส่งออกไป ซึ่งมีผลทำให้สัญญาณข้อมูลนั้นเกิดความผิดเพี้ยนไปจากรูปสัญญาณเดิม โดยส่วนที่สัญญาณรบกวนสามารถเข้าไปกวนสัญญาณข้อมูลได้นั้น เช่น สื่อกลางของอุปกรณ์ไฟฟ้า สัญญาณไฟฟ้าที่ไม่ปราถนา อุปกรณ์ไฟฟ้าต่าง ประสิทธิภาพที่ลดลงของวงจรไฟฟ้า เป็นต้น ซึ่งสิ่งที่ได้กล่าวมานี้จะเป็นการรบกวนที่หลีกเลี่ยงไม่ได้เนื่องจากสิ่งเหล่านี้มีอยู่ในธรรมชาติตลอดเวลา

ข้อสังเกตเบื้องต้นที่จะทำให้เราสามารถบ่งบอกได้ว่ามีการเกิดความผิดพลาดขึ้น เช่น มีการเพิ่มจำนวนของบิตข้อมูล การเปลี่ยนแปลงของบิตข้อมูล บิตข้อมูลมีเกิดการสูญหาย หรือจำนวนบิตของข้อมูลลดลง

ตารางที่ 2.1 แหล่งกำเนิดของความผิดพลาด และวิธีการป้องกัน (Sources of errors and prevention)

แหล่งกำเนิดความผิดพลาด (Sources of errors)	เหตุผลที่ทำให้เกิดความผิดพลาด(What causes it)	วิธีการป้องกัน (prevention)
Line outages	ความผิดพลาดของอุปกรณ์, เกิดพายุ, อุบัติเหตุต่างๆที่ทำให้วงจรไฟฟ้าตก หรือดับ	-
White noise (Gaussian niose)	เกิดจากการเคลื่อนที่ของอิเล็กตรอน(เกิดพลังงานความร้อน)	เพิ่มกำลังของสัญญาณ (เพิ่ม Signal to noise ratio (SNR))
Impules noise (spike)	การเพิ่มกระแสไฟฟ้าอย่างทันทีทันใด เช่น เกิดฟ้าผ่าขึ้นที่สายส่ง เกิดการแกว่งของกำลังไฟฟ้า เป็นต้น	ใช้สายส่งที่มีฉนวนหุ้มสายส่ง หรือย้ายตำแหน่งของสายส่ง
Cross-talk	Guard band ของ มัลติเพล็กเซอร์น้อยเกินไป, เกิดการรบกวนกันเกิดขึ้นระหว่างสายส่งที่อยู่ใกล้เคียง	เพิ่มขนาดของ Guard band หรือใช้สายส่งที่มีฉนวนห่อหุ้ม

ตารางที่ 2.1 (ต่อ) แหล่งกำเนิดของความผิดพลาด และวิธีการป้องกัน (Sources of errors and prevention)

Echo	เกิดจากการเชื่อมต่อกันทำได้ไม่ดี(สัญญาณที่ถูกส่งออกไปบางส่วนได้ถูกสะท้อนกลับมายังแหล่งกำเนิดสัญญาณ)	ซ่อมแซมการเชื่อมต่อ หรือปรับปรุงอุปกรณ์ที่ใช้เชื่อมต่อกัน
Attenuation	เกิดการลดทอนของระดับสัญญาณขึ้นระหว่าทางที่สัญญาณกำลังเดินทางไปยังปลายทาง	ติดตั้งอุปกรณ์ทวนสัญญาณหรืออุปกรณ์ขยายระดับสัญญาณ
Intermodulation noise	สัญญาณจากวงจรไฟฟ้าทั่วๆไปได้เกิดการรวมตัวของสัญญาณขึ้น	เคลื่อนย้าย หรือใช้สายส่งที่มีฉนวนห่อหุ้ม
Jitter	เกิดจากการเปลี่ยนแปลงสัญญาณอนาลอก เช่น เกิดการเปลี่ยนแปลงขนาดสัญญาณ มุมเฟสของสัญญาณ หรือความถี่ของสัญญาณ	ปรับแต่งอุปกรณ์ให้มีความเสถียรมากขึ้น
Harmonic distortion	อุปกรณ์ขยายสัญญาณทำให้เกิดการเปลี่ยนเฟสของสัญญาณ	ปรับแต่งอุปกรณ์ให้มีความเสถียรมากขึ้น

จากตารางที่ 2.1 สาเหตุที่สำคัญหลักๆที่ทำให้เกิดความผิดพลาดของข้อมูลที่ถูกส่งออกไป คือ Line outages, White noise(Gaussian niose), Impules noise (spike), Cross-talk, Echo, Attenuation ซึ่งเป็นการรบกวนที่มีอยู่ในธรรมชาติ ส่วน Internodulation noise, Jitter, Harmonic distortion จะเกิดขึ้นบ่อยกับการส่งสัญญาณแบบอนาลอก

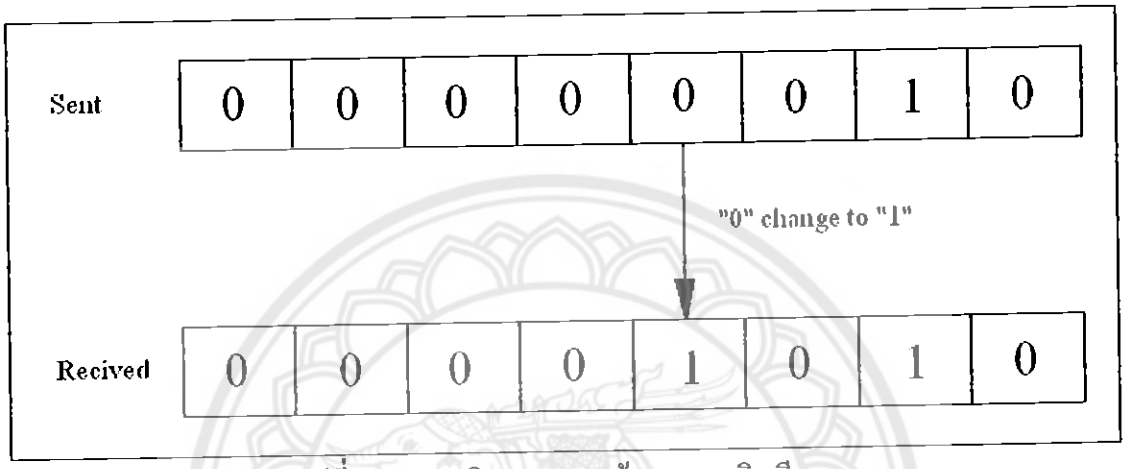
2.2.2 ประเภทความผิดพลาดของข้อมูล

สัญญาณรบกวน(noise) ต่างๆ ที่เกิดขึ้นนั้นทำให้สัญญาณที่ถูกส่งออกไปในระบบสื่อสารเกิดการเปลี่ยนแปลงรูปแบบได้ เช่น เปลี่ยนจาก "0" เป็น "1" หรือเปลี่ยนจาก "1" เป็น "0" ได้ ซึ่งเหตุการณ์

เช่นนี้จะทำให้ข้อมูลที่ได้รับที่ปลายทางเกิดความผิดพลาดไปจากเดิม เราจึงสามารถแบ่งประเภทความผิดพลาดของข้อมูล(Types of errors) [5] แบ่งได้เป็น 2 ประเภทด้วยกันคือ

1. ความผิดพลาดบิตเดียว (single-bit error)

ความผิดพลาดของข้อมูลแบบบิตเดียว(Single bit error) [5] หมายความว่า ข้อมูลแต่ละชุดที่ถูกส่งออกไปจะมีความผิดพลาดเกิดขึ้นเพียงบิตเดียวเท่านั้น โดยปกติแล้วความผิดพลาดแบบบิตเดียวนั้นมักจะเกิดขึ้นกับการส่งข้อมูลแบบขนาน(parallel transmission)



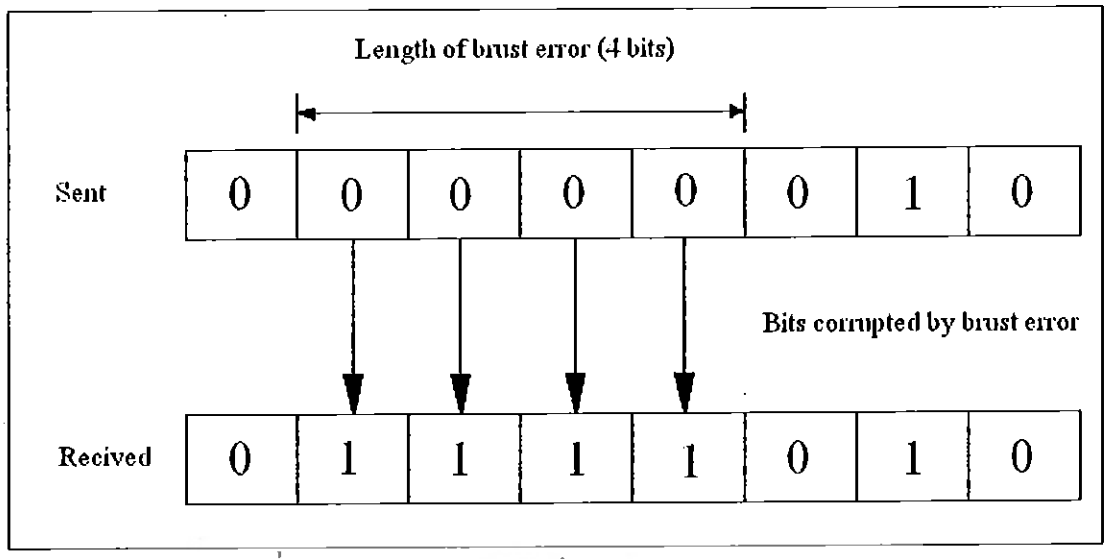
รูปที่ 2.1 ความผิดพลาดของข้อมูลแบบบิตเดียว

จากรูปที่ 2.1 จะเห็นได้ว่าข้อมูลที่ได้ถูกส่งออกไปมีจำนวน 8 บิต โดยที่ข้อมูลที่ถูกส่งออกไปจากตัวส่งคือ 00000010 แต่ได้มีสัญญาณรบกวนระหว่างที่ข้อมูลกำลังเดินทางไปยังภาครับส่งผลให้ข้อมูลชุดนี้เปลี่ยนแปลงกลายเป็น 00001010 ซึ่งเมื่อรับข้อมูลนี้ไป ถ้าที่ภาครับไม่มีการตรวจสอบข้อผิดลาดและแก้ไขข้อผิดพลาดที่เกิดขึ้นนั้นจะทำให้ข้อมูลที่ได้รับเป็นข้อมูลที่มีความผิดพลาด

ตัวอย่าง สมมติว่าข้อมูลได้ถูกส่งออกไปด้วยอัตราเร็ว 1 Mbps (1 บิต ใช้เวลา 1 ไมโครวินาที) ดังนั้นถ้ามีสัญญาณรบกวนเกิดขึ้นด้วยเวลา 1 ไมโครวินาที จะทำให้ข้อมูลเกิดความผิดพลาด 1 บิตได้ แต่ในความเป็นจริงแล้วช่วงเวลาที่เกิดสัญญาณรบกวนจะใช้เวลามากกว่านี้ ดังนั้นจึงสามารถทำให้เกิดความผิดพลาดของข้อมูลที่ได้ถูกส่งไปมากกว่า 1 บิตได้ด้วย

2. ความผิดพลาดหลายบิต (Burst error)

ความผิดพลาดของข้อมูลหลายบิต(Burst error) [6-7, 11] นั้น เป็นความผิดพลาดที่เกิดขึ้นกับข้อมูลโดยมีความผิดพลาดตั้งแต่ 2 บิตขึ้นไป เนื่องจากสัญญาณที่เกิดขึ้นอาจจะมีช่วงเวลาที่นาน ดังนั้น จึงสามารถทำให้เกิดความผิดพลาดของข้อมูลได้มากกว่า 1 บิตขึ้นไปนั่นเอง



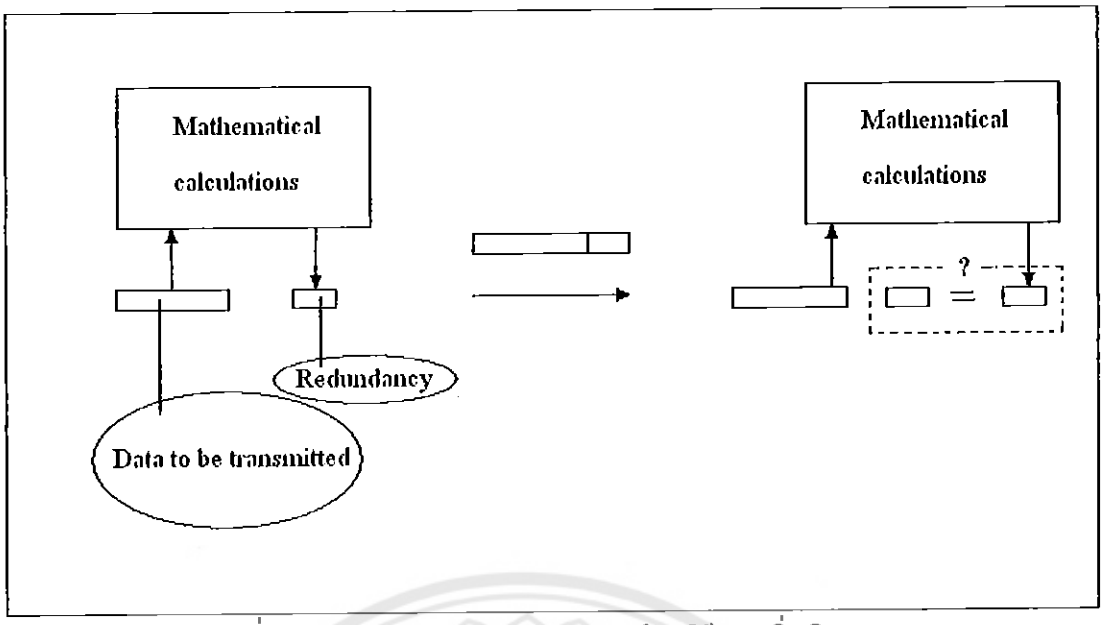
รูปที่ 2.2 ความผิดพลาดของข้อมูลแบบหลายบิต (4 บิต)

จากรูปที่ 2.2 จะเห็นได้ว่าข้อมูลที่ถูกส่งออกไปนั้นมีจำนวน 8 บิต และความผิดพลาดที่เกิดขึ้นนี้ไม่จำเป็นว่าข้อมูลที่ผิดเพี้ยนไปจากเดิมจะต้องอยู่ติดกันเสมอไป เนื่องจากสัญญาณรบกวนสามารถเกิดขึ้นได้เป็นช่วงๆ เช่นกัน และแต่ละช่วงอาจจะมีระยะเวลาที่เกิดการรบกวนไม่เท่ากันด้วย โดยทั่วไปแล้วจะพบว่าความผิดพลาดของข้อมูลแบบหลายบิตนี้ จะเกิดขึ้นกับการส่งข้อมูลแบบอนุกรม

ตัวอย่าง สมมติว่าข้อมูลได้ถูกส่งออกไปด้วยอัตราเร็ว 1 Mbps (1 บิต ใช้เวลา 1 ไมโครวินาที) ดังนั้นถ้ามีสัญญาณรบกวนเกิดขึ้นเป็นเวลา 0.01 วินาที จะส่งผลทำให้ข้อมูลที่ภาครับรับได้นั้นจะเกิดความผิดพลาดจะมีได้ทั้งหมด 10,000 บิต

2.2.3 การตรวจสอบความผิดพลาดของข้อมูล (error detection)

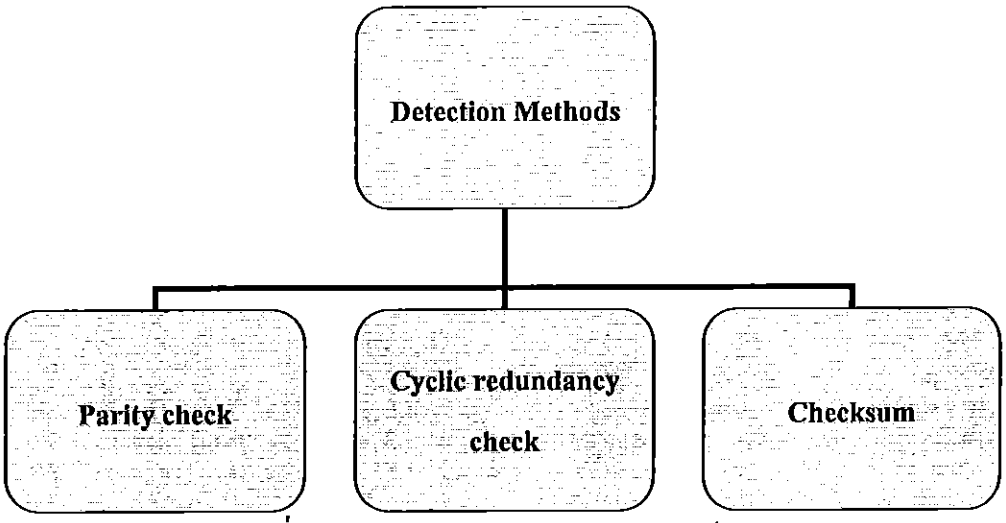
ก่อนที่จะสามารถแก้ไขความผิดพลาด(errorcorrection) [2-5] ของข้อมูลได้นั้น จะต้องทำการตรวจสอบข้อมูลเสียก่อนว่ามีความผิดพลาดเกิดขึ้นกับข้อมูลหรือไม่ วิธีการง่ายๆอย่างหนึ่งที่สามารถทำได้คือ ส่งบิตข้อมูลไป 2 ชุด เมื่อถึงภาครับก็ทำการเปรียบเทียบกันว่าบิตข้อมูลที่ได้รับนั้นมีบิตตรงกันหรือไม่ แต่วิธีการแบบนี้ไม่มีประสิทธิภาพมากนัก เนื่องจากจะต้องส่งข้อมูลเพิ่มขึ้นอีกเท่าตัว นอกจากนั้นแล้วยังต้องเสียเวลาในการเปรียบเทียบข้อมูลว่าเหมือนกันหรือไม่ด้วย ดังนั้นจึงต้องใช้วิธีการแบบใหม่ โดยทำการตรวจสอบ(Redundancy) [4] เข้าไปกับข้อมูลจริงถึงแม้ว่าวิธีการนี้จะลดประสิทธิภาพของการส่งลงไปบ้าง แต่ก็ยังน้อยกว่าวิธีการแบบแรกมาก



รูปที่ 2.3 การตรวจสอบความผิดพลาดโดยวิธีการเพิ่มบิตตรวจสอบ

จากรูปที่ 2.3 เป็นการตรวจสอบความผิดพลาดโดยวิธีการเพิ่มบิตตรวจสอบ เมื่อเรามีข้อมูลที่เราต้องการส่ง เราจะนำข้อมูลนั้นมาทำการคำนวนทางคณิตศาสตร์ตามหลักการคำนวนที่ออกแบบมาเพื่อหาค่าของบิตที่จะให้ในการตรวจสอบบิตข้อมูล เมื่อได้บิตตรวจสอบข้อมูลแล้วเราจะทำการส่งบิตข้อมูล และบิตที่ใช้ในการตรวจสอบข้อมูลออกไปเมื่อไปถึงยังภาครับ ถาครับก็จะนำข้อมูลที่ได้มาแยกเป็นส่วนของบิตข้อมูล และบิตตรวจสอบออก หลังจากนั้นจะนำส่วนที่เป็นบิตข้อมูลมาคำนวนหาค่าบิตตรวจสอบเหมือนกับที่ภาคส่งได้ทำการคำนวนเพื่อที่จะนำมาหาความผิดพลาดว่ามีความผิดพลาดเกิดขึ้นหรือไม่ ถ้าค่าบิตตรวจสอบที่ได้จากการคำนวนของภาครับมีค่าเท่ากับค่าบิตตรวจสอบที่ได้รับจากภาคส่งก็แสดงว่าข้อมูลที่ได้รับมานั้นมีความถูกต้อง แล้วจึงจะทำการทิ้งบิตตรวจสอบเหล่านั้นไป ในทางกลับกันถ้าค่าบิตที่ได้จากการคำนวนของภาครับนั้นมีค่าไม่เท่ากับค่าที่คำนวณได้จากภาคส่งแสดงว่าข้อมูลที่ได้รับนั้นเกิดความผิดพลาดเกิดขึ้นระหว่างการเดินทางมายังภาครับ

ในการตรวจสอบความผิดพลาดของข้อมูลนั้นโดยการใช้บิตตรวจสอบนั้นสามารถทำได้ 3 วิธีการ ดังรูปที่ 2.4



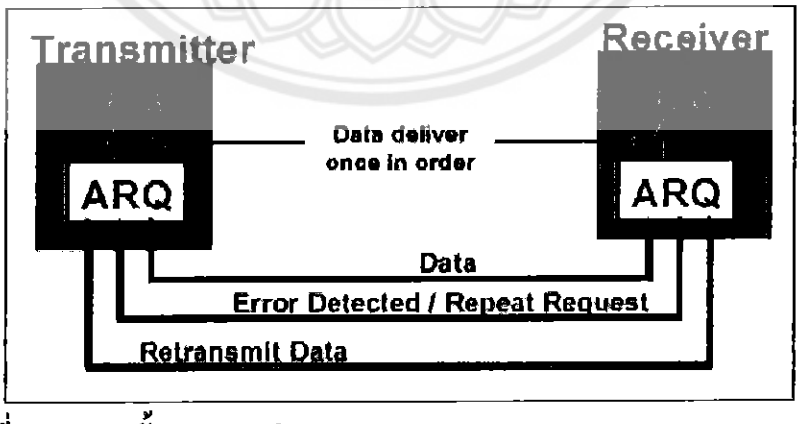
รูปที่ 2.4 วิธีตรวจสอบความผิดพลาดของข้อมูล

2.2.4 การแก้ไขความผิดพลาดของข้อมูล

จากหัวข้อ 2.2.3 ได้กล่าวถึงการตรวจสอบว่าข้อมูลที่ได้รับมานั้นเกิดความผิดพลาดขึ้นหรือไม่ และในหัวข้อนี้จะกล่าวถึงเทคนิคเพื่อแก้ไขความผิดพลาดของข้อมูลนั้นๆ ซึ่งแบ่งออกได้เป็น 2 วิธีคือ แก้ไขความผิดพลาดของข้อมูลด้วยการส่งข้อมูลใหม่(Automatic Repeat Request) [1, 6] และแก้ไขความผิดพลาดของข้อมูลด้วยวิธี Forward error correction [1, 4-6]

1. ARQ (Automatic Repeat Request)

เมื่อผู้รับตรวจสอบแล้วว่าข้อมูลที่ได้รับมานั้นมีความผิดพลาดของข้อมูล จะต้องทำการส่งข้อมูลไปบอกกับทางผู้ส่งว่าต้องการที่จะให้ส่งกลับมาใหม่อีกครั้งหนึ่ง ซึ่งเทคนิคในการควบคุมความผิดพลาดของข้อมูลด้วยการส่งข้อมูลกลับมาใหม่ ดังรูปที่ 2.5



รูปที่ 2.5 แสดงขั้นตอนการทำงานของ ARQ (Automatic Repeat Request) [11]

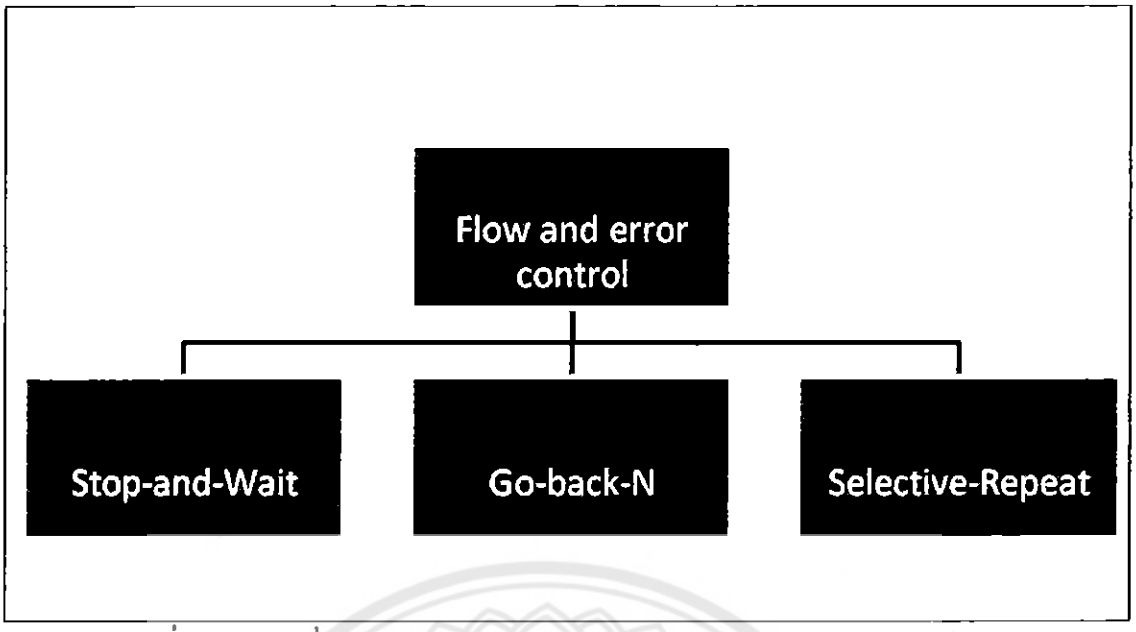
วิธีการแก้ไขความผิดพลาดของข้อมูลด้วยการส่งข้อมูลใหม่ ARQ (Automatic Repeat Request) [1, 6] นั้นจะต้องมีการควบคุมอัตราการไหล และควบคุมความผิดพลาดของข้อมูล (Flow and error control)

การควบคุมอัตราการไหลของข้อมูล(Flow control) [6] เป็นการกล่าวถึงขั้นตอนกระบวนการที่ควบคุมจำนวนของข้อมูลที่ถูกส่งออกไปให้อยู่ในปริมาณที่เหมาะสม ก่อนที่จะได้รับการตอบรับการยืนยันจากผู้รับข้อมูล นั่นหมายความว่าข้อมูลที่ถูกส่งออกไปนั้นจะต้องถูกจำกัดเอาไว้จำนวนหนึ่ง ถ้ายังไม่ได้รับการตอบรับจากผู้รับว่าได้รับข้อมูลชุดนั้นแล้ว ผู้ส่งจะไม่ส่งข้อมูลชุดถัดไป สาเหตุที่ต้องควบคุมการไหลของข้อมูล ก็เนื่องจากว่าผู้รับอาจมีความเร็วในการรับข้อมูลไม่เท่ากับผู้ส่ง หรือมีหน่วยความจำที่ใช้ในการเก็บข้อมูลอย่างจำกัด หรือมีความเร็วในการประมวลผลต่ำ ดังนั้นการควบคุมอัตราการไหลของข้อมูลจึงมีความจำเป็นอย่างมาก ถ้าผู้รับยังไม่พร้อมที่จะรับข้อมูลในขณะนั้น ผู้ส่งจะต้องทำการหยุดส่งข้อมูลชั่วคราวเสียก่อน หรือถ้าผู้รับมีความเร็วในการประมวลผลต่ำ ผู้ส่งจะต้องส่งข้อมูลด้วยอัตราเร็วที่เหมาะสมเพื่อให้เกิดความสมดุลกันในการรับส่งข้อมูล

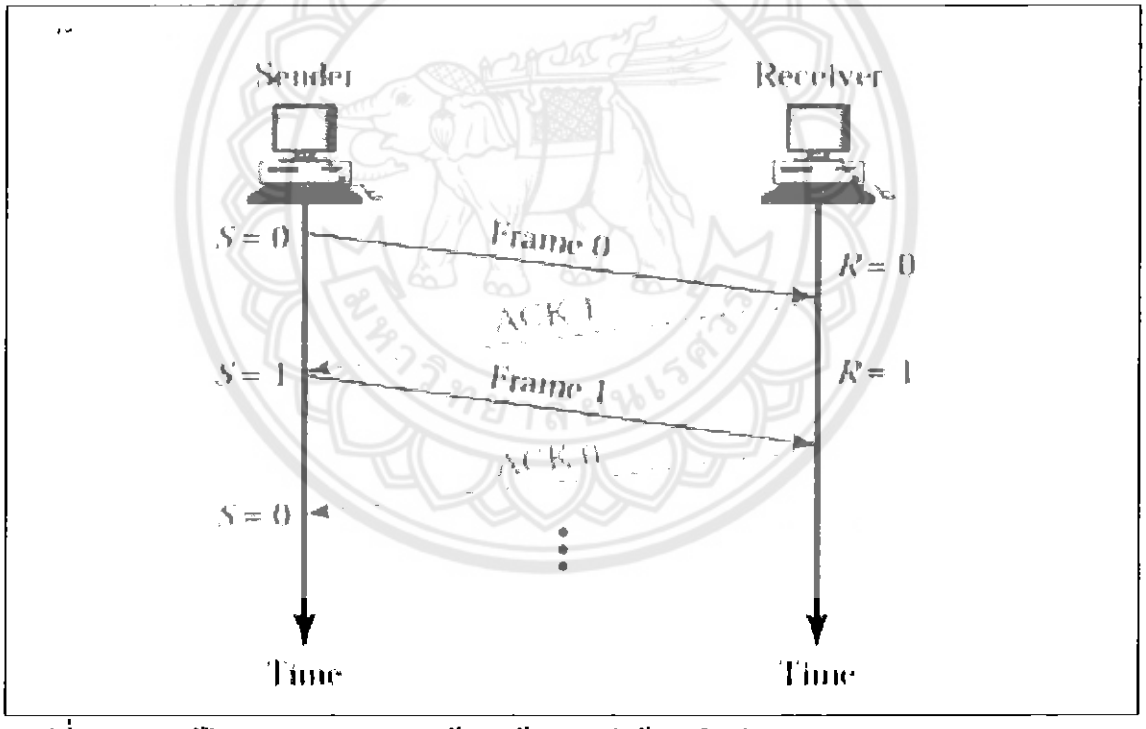
การควบคุมความผิดพลาดของข้อมูล(Error control) [2-6] จะหมายถึง การที่ผู้ส่งจะต้องส่งข้อมูลไปใหม่อีกครั้งหนึ่ง ถ้าผู้รับไม่สามารถรับข้อมูลหรือได้รับข้อมูลที่ไม่ถูกต้อง สาเหตุที่ต้องมีการควบคุมก็เนื่องจากว่าข้อมูลจะต้องเดินทางจากที่หนึ่งไปยังอีกที่หนึ่ง จึงมีความเป็นไปได้ที่ข้อมูลชุดนั้นจะเกิดการสูญหายหรือเสียหายในระหว่างการเดินทางได้ ดังนั้นผู้รับจะต้องมีกระบวนการในการตรวจสอบความผิดพลาดของข้อมูล นอกจากนั้นจะต้องมีการส่งข้อความไปบอกกับผู้ส่งว่าข้อมูลชุดไหนที่รับมามีความผิดพลาด ซึ่งจะต้องทำการส่งข้อมูลชุดนั้นๆไปใหม่อีกครั้งหนึ่งจนกว่าจะได้รับข้อมูลที่มีความถูกต้อง

กลไกที่ควบคุมอัตราการไหล และควบคุมความผิดพลาดของข้อมูลซึ่งมีอยู่ 3 วิธี คือ

1. Stop-and-Wait [11]
2. Go-back-N [11]
3. Selective-Repeat [11]



รูปที่ 2.6 วิธีการที่ใช้ควบคุมอัตราการไหล และควบคุมความผิดพลาดของข้อมูล



รูปที่ 2.7 การแก้ไขความผิดพลาดของข้อมูลด้วยการส่งข้อมูลใหม่(Automatic Repeat Request)[13]

2. FEC (Forward Error Correction)

Forward error correction หรือ FEC [1, 4-6] เป็นเทคนิคที่จะทำให้ผู้รับข้อมูลที่เกิดความผิดพลาดนั้นสามารถที่จะแก้ไขความผิดพลาดของข้อมูลได้อัตโนมัติ แต่อย่างไรก็ตามเทคนิคแบบนี้จะมีความยุ่งยากและต้องการจำนวนของบิตที่ใช้ตรวจสอบที่มากขึ้น ตัวอย่างการเข้ารหัสช่องสัญญาณที่ใช้ทฤษฎีนี้ เช่น แฮมมิงโค้ด

2.3 การเข้ารหัสช่องสัญญาณแบบแฮมมิง(Hamming code)

เทคนิคที่จะสามารถแก้ไขความผิดพลาดของข้อมูลโดยอัตโนมัติที่จะได้กล่าวถึงนี้ จะใช้ได้ในกรณีที่เกิดความผิดพลาดเกิดขึ้นเพียงบิตเดียวเท่านั้น โดยจะกำหนดให้ m เป็นบิตข้อมูล ส่วน r เป็นบิตตรวจสอบ หรือพาริตี้บิต ดังนั้นข้อมูลที่ต้องการส่งจริง(code word)คือ m + r

จากทฤษฎีของแฮมมิง(Hamming) [1-6, 9] เราสามารถเขียนความสัมพันธ์ระหว่างบิตข้อมูลและบิตตรวจสอบได้ดังนี้

$$2^r > m + r + 1 \qquad (2.1)$$

ตัวอย่าง รหัสแอสกี้(ASCII code) จะมีจำนวนบิตเท่ากับ 7 บิต ดังนั้นถ้าเราต้องการที่จะส่งตัวอักษร 7 ตัวอักษร จะต้องมีบิตที่ใช้ตรวจสอบ 4 บิต เนื่องจาก

$2^4 >= 7 + 4 + 1$

ซึ่งเป็นไปตามทฤษฎีของแฮมมิง(Hamming) [1-5, 9] ที่กล่าวถึงความสัมพันธ์ระหว่างบิตข้อมูลและบิตที่ใช้ตรวจสอบ

ในตารางที่ 2.2 จะแสดงถึงความสัมพันธ์ระหว่างบิตข้อมูล m บิต และบิตตรวจสอบ r บิต ที่สามารถใช้แก้ไขข้อมูลที่ผิดพลาดได้

ตารางที่ 2.2 ความสัมพันธ์ระหว่างบิตข้อมูลและบิตตรวจสอบ

จำนวนของบิตข้อมูล (m)	จำนวนของบิตตรวจสอบ (r)	จำนวนข้อมูลที่ส่งทั้งหมด (m + r)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10

ตารางที่ 2.2 (ต่อ) ความสัมพันธ์ระหว่างบิตข้อมูลและบิตตรวจสอบ

7	4	11

แฮมมิงโค้ด [1-5, 9] เป็นวิธีในการกำหนดว่าบิตตรวจสอบควรจะอยู่ตำแหน่งใดของข้อมูลบ้าง ซึ่งจะแตกต่างจากวิธีอื่นๆที่จะนำบิตตรวจสอบ หรือพาริตี้บิตจะอยู่ตรงส่วนท้ายของข้อมูล แฮมมิงโค้ดสามารถนำไปประยุกต์ใช้ได้กับข้อมูลที่มีความยาวเท่าใดก็ได้ เช่น รหัสแอสกี้ที่มีความยาวข้อมูล 7 บิต จะต้องมีบิตตรวจสอบ 4 บิต เพื่อให้สามารถตรวจสอบความผิดพลาดและแก้ไขความผิดพลาดขนาด 1 บิต ที่เกิดขึ้นได้

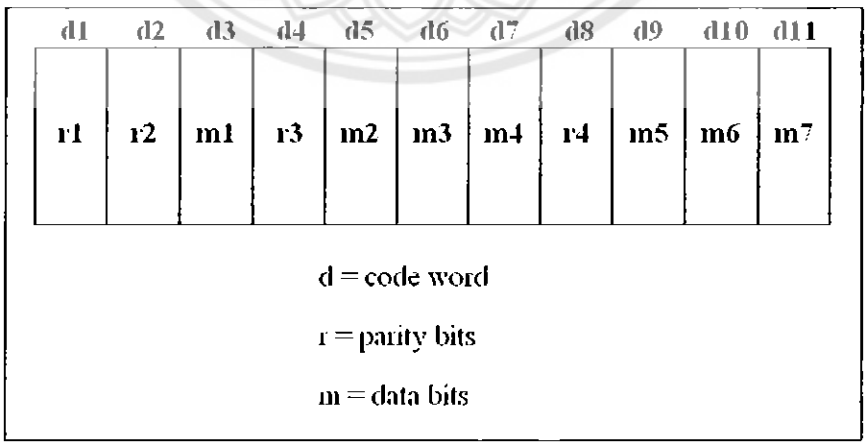
2.3.1 การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด

การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด [1-6, 9] นั้นจะต้องมีการคำนวณจำนวนของพาริตี้บิตหรือบิตที่จะต้องเพิ่มเข้าไปในส่วนของข้อมูลที่เรามีอยู่เพื่อใช้บิตที่เราเพิ่มเข้าไปเป็นตัวที่จะใช้ตรวจสอบว่าข้อมูลที่เราส่งออกไปยังปลายทางนั้นภาครับได้รับข้อมูลที่ถูกต้องหรือไม่ ถ้าข้อมูลที่ส่งเข้าไปยังภาครับมีความถูกต้อง ก็ให้ทำการตัดพาริตี้บิตออก หรือถ้าบิตข้อมูลที่ได้รับเกิดความผิดพลาดเกิดขึ้น 1 บิตก็จะสามารถตรวจสอบได้ว่าเกิดความผิดพลาดเกิดขึ้นแล้วเกิดขึ้นที่ตำแหน่งใดของข้อมูลที่ส่งออกไปแล้วทำการแก้ไขข้อมูลบิตที่ผิดพลาดตรงปลายทางเลย

ขั้นตอนการคำนวณเพื่อเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด

1. กำหนดตำแหน่งบิตที่จะใช้เป็นบิตตรวจสอบ

หลักในการคำนวณหาตำแหน่งการวางบิตตรวจสอบนั้นจะใช้เลขยกกำลังของ 2 คือ 2^n โดยที่ n เริ่มจาก 0, 1, 2, ... ก็จะได้ตำแหน่งที่ $2^0, 2^1, 2^2, \ldots, 2^{r-1}$ ก็คือตำแหน่งที่ 1, 2, 4, 8, 16, 32, 64, ..., 2^{r-1}



รูปที่ 2.8 ตำแหน่งของบิตตรวจสอบในแฮมมิงโค้ด [6]

จากรูปที่ 2.8 จะแสดงถึงการวางตำแหน่งของบิตตรวจสอบทั้ง 4 บิต ตำแหน่งของการวางบิตเหล่านี้ คือตำแหน่งที่ d1, d2, d4 และ d8 ซึ่งเราจะแทนด้วย r1, r2, r3 และ r4 บิตข้อมูลก่อนเข้ารหัสช่องสัญญาณ ตำแหน่งของการวางบิตเหล่านี้ คือตำแหน่งที่ d3, d5, d6, d7, d9, d10 และ d11 ซึ่งเราจะแทนด้วย m1, m2, m3, m4, m5, m6 และm7

2. ตำแหน่งของบิตข้อมูลที่จะส่งจะอยู่ในตำแหน่งอื่นๆที่ไม่อยู่บนตำแหน่งเดียวกับตำแหน่งของบิตตรวจสอบ ก็จะได้ว่าตำแหน่งของบิตข้อมูลจะอยู่ตำแหน่งที่ 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, ...

3. ค่าแต่ละตำแหน่งของบิตตรวจสอบก็จะต้องมีการคำนวณเพื่อหาค่าของบิตตรวจสอบที่ตำแหน่งนั้นๆ โดยนำแต่ละตำแหน่งดังวิธีการต่อไปนี้มาคำนวณ

บิตตรวจสอบตำแหน่งที่ 1 จะเช็ค 1 ตำแหน่งแล้วเว้นไป 1, เช็ค 1 ตำแหน่งแล้วเว้นไป 1 , ... ตำแหน่ง ก็จะได้ตำแหน่งที่จะนำมาคำนวณพาริตี้บิตตำแหน่งที่ 1 คือ 1, 3, 5, 7, 9, ...

บิตตรวจสอบตำแหน่งที่ 2 จะเช็ค 2 ตำแหน่งแล้วเว้นไป 2, เช็ค 2 ตำแหน่งแล้วเว้นไป 2, ... ตำแหน่ง ก็จะได้ตำแหน่งที่จะนำมาคำนวณพาริตี้บิตตำแหน่งที่ 2 คือ 2,3,6,7,10,11, ...

บิตตรวจสอบตำแหน่งที่ 4 จะเช็ค 4 ตำแหน่งแล้วเว้นไป 4, เช็ค 4 ตำแหน่งแล้วเว้นไป 4 , ... ตำแหน่ง ก็จะได้ตำแหน่งที่จะนำมาคำนวณพาริตี้บิตตำแหน่งที่ 4 คือ 4, 5, 6, 7, 12, 13, 14, 15, 20, ...

พาริตี้บิตตำแหน่งที่ 8 จะเช็ค 8 ตำแหน่งแล้วเว้นไป 8, เช็ค 8 ตำแหน่งแล้วเว้นไป 8 , ... ตำแหน่ง ก็จะได้ตำแหน่งที่จะนำมาคำนวณพาริตี้บิตตำแหน่งที่ 8 คือ 8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31, 40, 41, 42, 43, 44, 45, 46, 47, ...

บิตตรวจสอบตำแหน่งที่ 16 จะเช็ค 16 ตำแหน่งแล้วเว้นไป 16, เช็ค 16 ตำแหน่งแล้วเว้นไป 16 , ... ตำแหน่ง ก็จะได้ตำแหน่งที่จะนำมาคำนวณพาริตี้บิตตำแหน่งที่ 16 คือ 16-31, 48-63, 80-95, ...

บิตตรวจสอบตำแหน่งที่ 32 จะเช็ค 32 ตำแหน่งแล้วเว้นไป 32, เช็ค 32 ตำแหน่งแล้วเว้นไป 32 , ... ตำแหน่ง ก็จะได้ตำแหน่งที่จะนำมาคำนวณพาริตี้บิตตำแหน่งที่ 32 คือ 32-63,96-127,160-191, ...

การหาตำแหน่งของบิตตรวจสอบก็จะเป็นไปในลักษณะนี้ ทั้งนี้ขึ้นอยู่กับว่าจำนวนของบิตตรวจสอบที่เราต้องการนั้นมีจำนวนมากน้อยเพียงใด

4. ค่าของพาริตี้บิตจะนำบิตในตำแหน่งที่ได้กำหนดไว้ในข้อ 3 มาคำนวณโดยจะนำตำแหน่งแต่ละตำแหน่งมามอดูแลทกัน ถ้าได้ผลการมอดดูแลทออกมาเป็น 0 พาริตี้บิตก็จะมีค่าเป็น 0 ถ้าได้ผลการมอดดูแลทออกมาเป็น 1 พาริตี้บิตก็จะมีค่าเป็น 1

ตัวอย่าง การคำนวณหาค่าพาริตี้บิต

ให้บิตข้อมูลจำนวน 1 ไบต์ มีข้อมูลดังนี้ 10011010

create data word : _ _ 1 _ 0 0 1 _ 1 0 1 0

ช่องว่างที่เห็นใช้สำหรับใส่พาริตี้บิต

คำนวณพาริตี้บิตแต่ละตำแหน่ง

ตำแหน่งที่ 1 ตรวจสอบบิตที่ 1,3,5,7,9,11

? _ 1 _ 0 0 1 _ 1 0 1 0 จากการคำนวนได้ พาริตี้คู่ ให้เซตตำแหน่งที่ 1 มีค่า "0" จะได้บิตข้อมูลเป็น 0 _ 1 _ 0 0 1 _ 1 0 1 0

ตำแหน่งที่ 2 ตรวจสอบบิตที่ 2,3,6,7,10,11

0 ? 1 _ 0 0 1 _ 1 0 1 0 จากการคำนวนได้ พาริตี้คี่ ให้เซตตำแหน่งที่ 2 มีค่า "1" จะได้บิตข้อมูลเป็น 0 1 1 _ 0 0 1 _ 1 0 1 0

ตำแหน่งที่ 4 ตรวจสอบบิตที่ 4,5,6,7,12

0 1 1 ? 0 0 1 _ 1 0 1 0 จากการคำนวนได้ พาริตี้คี่ ให้เซตตำแหน่งที่ 4 มีค่า "1" จะได้บิตข้อมูลเป็น 0 1 1 1 0 0 1 _ 1 0 1 0

ตำแหน่งที่ 8 ตรวจสอบบิตที่ 8,9,10,11,12

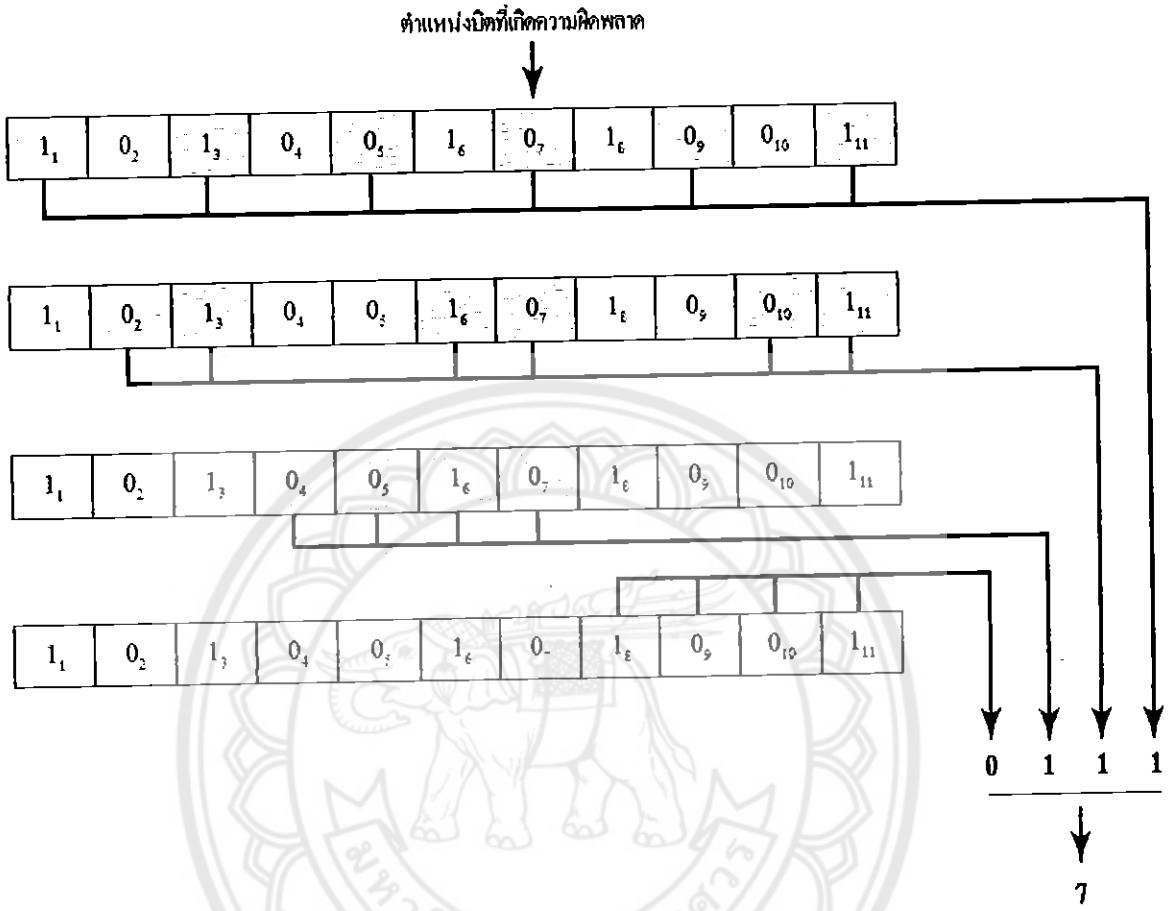
0 1 1 1 0 0 1 ? 1 0 1 0 จากการคำนวนได้ พาริตี้คู่ ให้เซตตำแหน่งที่ 1 มีค่า "0" จะได้บิตข้อมูลเป็น 0 1 1 1 0 0 1 0 1 0 1 0

ข้อมูลที่ทำการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดแล้ว(code word) คือ 0 1 1 1 0 0 1 0 1 0 1 0

ขั้นตอนการถอดรหัสแฮมมิง

1. ทางด้านรับคำนวณบิตตรวจสอบของแต่ละกลุ่มเหมือนทางด้านส่ง โดยแต่ละกลุ่มจะมีตำแหน่งบิตที่จะนำมาคำนวณเหมือนดังขั้นตอนในการเข้ารหัสแฮมมิงโค้ดในข้อ 3 ในขั้นตอนการคำนวณเพื่อเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด
2. เรียงค่าที่ได้จากการคำนวณของบิตตรวจสอบตามตำแหน่งดังนี้ ..., r8, r4, r2, r1
3. นำค่าที่ได้มาทำการคำนวณ วิธีการคำนวณคือนำค่าบิตตรวจสอบที่ได้แต่ละตำแหน่งมาทำการมอดูเลตกัน
4. ถ้าค่าที่ได้จากการคำนวณในข้อที่ 3 ในขั้นตอนการคำนวณเพื่อเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดมีค่าเป็น 0 แสดงว่าข้อมูลที่ได้รับมานั้นไม่มีความผิดพลาดเกิดขึ้น แต่ถ้าผลจากการคำนวณมีค่าที่ไม่ใช่ 0 แสดงว่าเกิดความผิดพลาดเกิดขึ้นและค่าที่ได้นั้นยังบ่งบอกได้อีกว่าตำแหน่งนั้นเป็นตำแหน่งที่มีความผิดพลาดเกิดขึ้น

5. เมื่อได้ทำการตรวจสอบและแก้ไขความผิดพลาดของข้อมูลแล้ว เราจะทำการตัดบิตที่ใช้ในการตรวจสอบออกให้เหลือแต่ข้อมูลต้นฉบับ กล่าวคือตัดตำแหน่งที่ r1, r2, r4, r8, ...



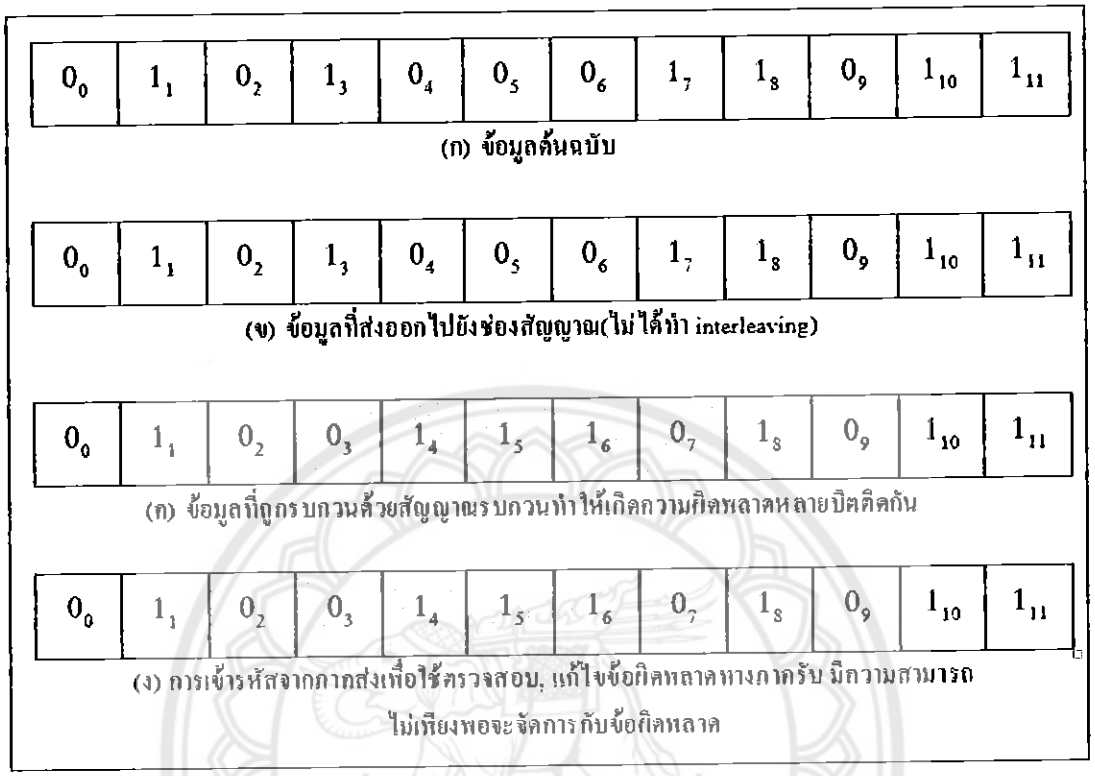
รูปที่ 2.9 การตรวจสอบความผิดพลาด [4]

จากรูปที่ 2.9 เราได้สมมติว่าตำแหน่งที่ 7 ได้เกิดความผิดพลาดขึ้น แล้วเราทำตามขั้นตอนที่ใช้ในการถอดรหัสแฮมมิงโค้ดก็คือ การนำตำแหน่งของข้อมูลของแต่ละกลุ่มของบิตตรวจสอบมาคำนวณซึ่งได้ออกมาเป็นเลขฐานสองคือ 0111 เมื่อทำการแปลงเป็นเลขฐานสิบ คือ 7 ซึ่งค่าที่ได้นี้ก็คือตำแหน่งที่ผิดพลาดนั่นเอง

2.4 อินเทอร์ลีฟวิง(interleaving)

อินเทอร์ลีฟวิง(Interleaving) [1, 6-8] เป็นวิธีการหนึ่งที่ใช้ในการแก้ไขความผิดพลาดของข้อมูลแบบหลายบิต(Burst error correction) [] ในระบบโทรศัพท์เคลื่อนที่ เนื่องจากการผิดพลาดแบบหลายบิตเป็นรูปแบบที่เกิดขึ้นอย่างบ่อยครั้งในการส่งผ่านสัญญาณในระบบโทรศัพท์เคลื่อนที่ เช่น เกิดจากการที่สัญญาณถูกบดบังด้วยวัตถุต่างๆที่อยู่รอบข้องไปชั่วขณะ ทำให้ข้อมูลที่ได้รับมีความผิดพลาดอย่างต่อเนื่อง ดังนั้นการทำอินเทอร์ลีฟวิงจึงมีประโยชน์มากในทางปฏิบัติ

2.4.1 ตัวอย่างแสดงให้เห็นถึงผลเสียเมื่อไม่สลับตำแหน่งข้อมูล(Interleaving) ก่อนที่จะส่งข้อมูลนั้นผ่านช่องสัญญาณ



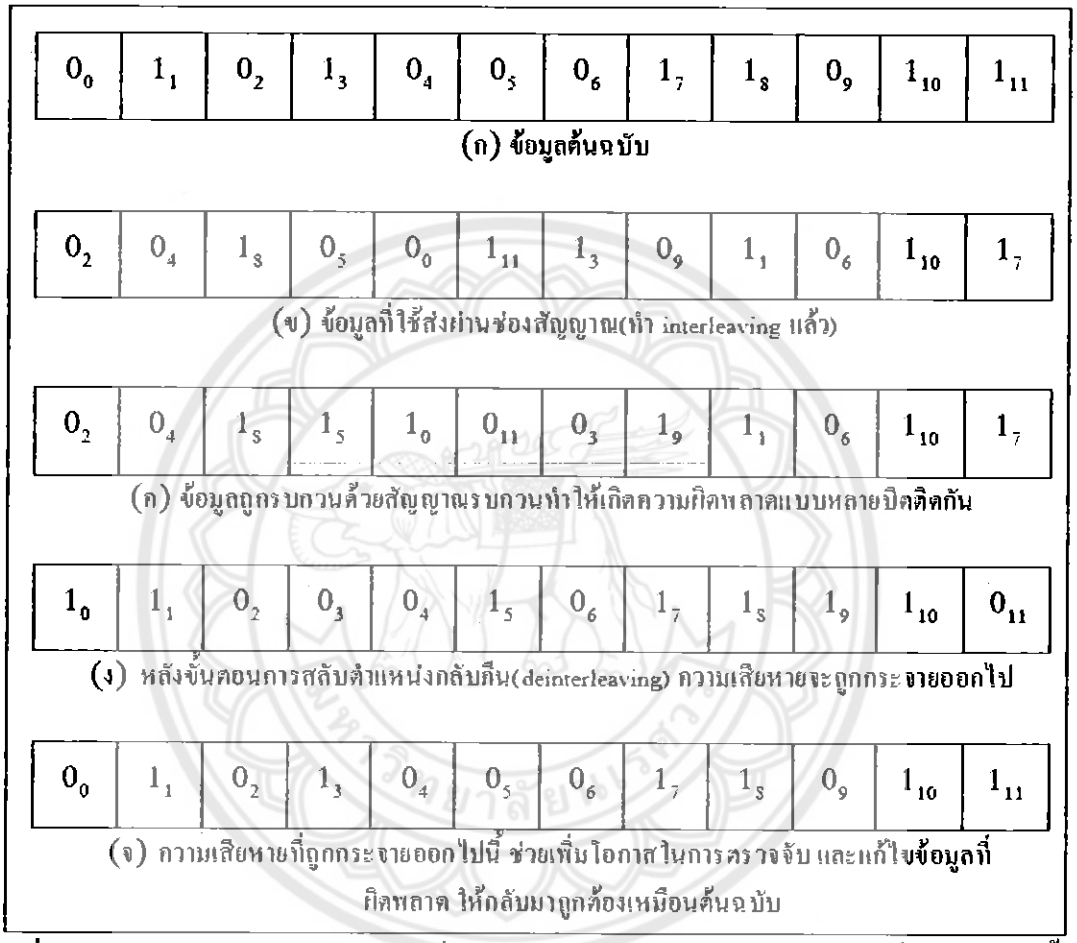
รูปที่ 2.10 ตัวอย่างแสดงให้เห็นถึงผลเสียเมื่อไม่สลับตำแหน่งข้อมูล(Interleaving) ก่อนที่จะส่งข้อมูลนั้นผ่านช่องสัญญาณ [12]

กรณีนี้ข้อมูลส่งออกอากาศ รูปที่ 2.10 จะเหมือนกันกับข้อมูลต้นฉบับทุกประการ รูปที่ 2.10 (ก) แต่อย่างไรก็ดีเมื่อข้อมูลชุดดังกล่าวถูกส่งออกไปแล้วก็จะมีสิ่งรบกวนบางประเภทในชั้นบรรยากาศซึ่งเข้ามาโจมตีขบวนข้อมูลในเวลาหนึ่ง ผลที่ตามมาคือทำให้เกิดข้อผิดพลาดบนข้อมูลดังกล่าวขึ้นทั้งนี้โดยทั่วไปความเสียหายจะกินบริเวณไปถึงข้อมูลที่อยู่ข้างเคียงกันในทางเวลาด้วย ดังรูปที่ 2.10 (ค) จากรูปบล็อคสีเทาคือข้อมูลส่วนที่ได้รับความเสียหาย (ค่าของมันจะเปลี่ยนไปจากค่าต้นฉบับ)

ด้วยลักษณะความเสียหายที่เกิดขึ้นเป็นบริเวณติดๆกันเช่นนี้เอง ที่ในหลายครั้งกลไกการตรวจจับและแก้ไขข้อผิดพลาดที่นำมาใช้แต่เพียงอย่างเดียวไม่สามารถจะจัดการกับมันได้ ส่งผลให้ด้านรับไม่อาจนำเอาสัญญาณที่ได้มาประมวลผลพร้อมแสดงออกทางอุปกรณ์ปลายทางเพราะฉะนั้นในระบบการส่งเพื่อใช้งานจริงจะต้องหาวิธีการเพื่อรองรับกับสถานการณ์เช่นนี้ด้วย อันถือเป็นการสร้างความทนทานให้แก่สัญญาณต่อสิ่งรบกวนเพิ่มเติมนอกเหนือจากกลไกที่มีอยู่แล้ว

2.4.2 ตัวอย่างแสดงให้เห็นถึงผลดีเมื่อสลับตำแหน่งข้อมูล(Interleaving)ก่อนที่จะส่งข้อมูลนั้นผ่านช่องสัญญาณ

คราวนี้ลองมาดูกันอีกกรณีหนึ่ง กล่าวคือ ข้อมูลชุดเดิมแต่มีการสลับตำแหน่งข้อมูลก่อนที่จะส่งไปในอากาศ



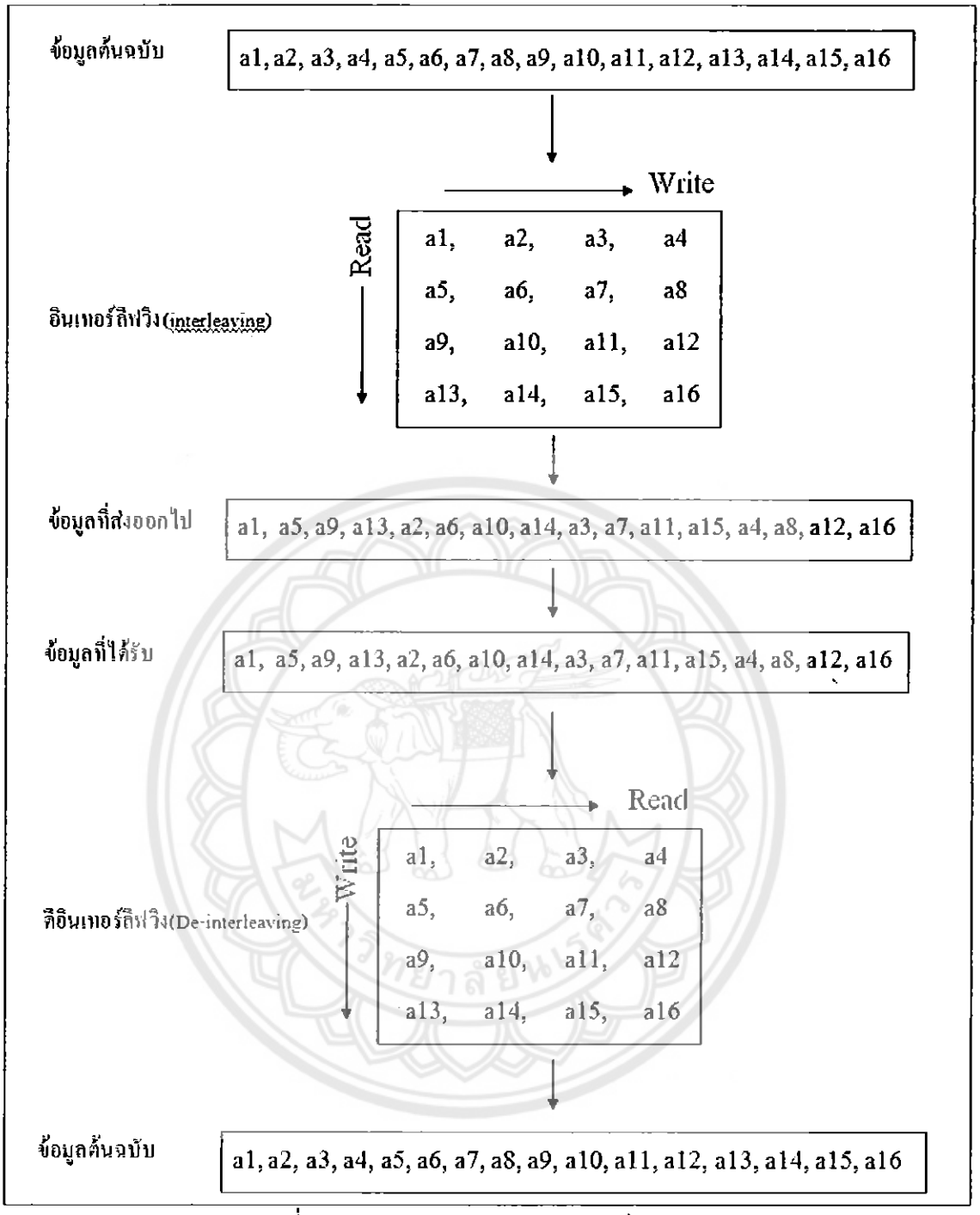
รูปที่ 2.11 ตัวอย่างแสดงให้เห็นถึงผลดีเมื่อสลับตำแหน่งข้อมูล(Interleaving)ก่อนที่จะส่งข้อมูลนั้นผ่านช่องสัญญาณ [12]

จะเห็นว่าเมื่อทำการสลับตำแหน่งข้อมูลก่อนที่จะส่งข้อมูลนั้นๆออกอากาศไป รูปที่ 2.11 (ข) ครั้นเมื่อข้อมูลถูกสิ่งรบกวนในอากาศโจมตีส่งผลให้ข้อมูลบริเวณหนึ่งได้รับความเสียหาย รูปที่ 2.11 (ค) แต่เนื่องจากข้อมูลเหล่านั้นในความเป็นจริงไม่ได้มีลำดับที่ติดต่อกันแต่อย่างใดจึงทำให้เมื่อขบวนการข้อมูลผ่านกระบวนการสลับตำแหน่งกลับคืนแล้วความเสียหายซึ่งเคยเกิดติดๆ กันก็จะกระจายออกไป รูปที่ 2.11 (ง) ด้วยสภาพที่ความผิดพลาดกระจายตัวเช่นนี้เองจึงเป็นงานง่ายขึ้นมากสำหรับการตรวจจับพร้อมแก้ไขข้อมูลให้กลับมาถูกต้องดังเดิม รูปที่ 2.11 (จ)

สรุปว่าวิธีการสลับตำแหน่งข้อมูลหรือ interleaving ถือเป็นเทคนิคอีกตัวหนึ่งที่ช่วยเพิ่มความแข็งแรงทนทานต่องสัญาณรบกวนต่างๆให้กับข้อมูล กล่าวคือมันไปลดโอกาสที่สัญญาณรบกวนจะทำให้เกิดความผิดพลาดแบบหลายบิตที่จนไม่สามารถนำข้อมูลกลับมาได้ตรงปลายทาง เพราะ interleaving ได้ไปช่วยกระจายข้อผิดพลาดไม่ให้กระจุกตัวเป็นกลุ่มก้อนหลังจากสลับข้อมูลกลับคืนมาแล้ว จึงเพิ่มโอกาสของการตรวจและแก้ไขข้อมูลให้ถูกต้องได้มากขึ้น

2.4.3 หลักการทำงานของอินเทอร์ลีฟวิง

หลักการทำอินเทอร์ลีฟก็คือ เราจะแบ่งบิตข้อมูลออกเป็นบล็อกๆ ขนาด r×c บิต โดย r คือจำนวนแถว และ c คือจำนวนคอลัมน์ จากนั้นก็จะทำการสลับตำแหน่งของแต่ละบิตกายในบล็อกเดียวกันก่อนการส่งออก โดยอาศัยวิธีการง่ายๆ คือ นำข้อมูลที่จะส่งมาเขียนลงในหน่วยความจำทีละแถวตามลำดับของข้อมูลที่เข้ามาจนครบหนึ่งบล็อก จากนั้นทำการอ่านข้อมูลเหล่านี้ในแนวตั้งเพื่อส่งออกทีละคอลัมน์จนหมด ดังที่แสดงในรูปที่ 2.12 และเมื่อบิตข้อมูลเหล่านี้ถึงที่ภาครับแล้วก็จะทำการสลับตำแหน่งของบิตให้กลับเป็นปกติโดยอาศัยกระบวนการที่กลับกัน กล่าวคือ นำข้อมูลที่ได้รับมาเขียนลงในหน่วยความจำทีละแถวตามลำดับของข้อมูลที่เข้ามาจนครบหนึ่งบล็อก จากนั้นทำการอ่านข้อมูลเหล่านี้ในแนวตั้งทีละคอลัมน์จนหมด



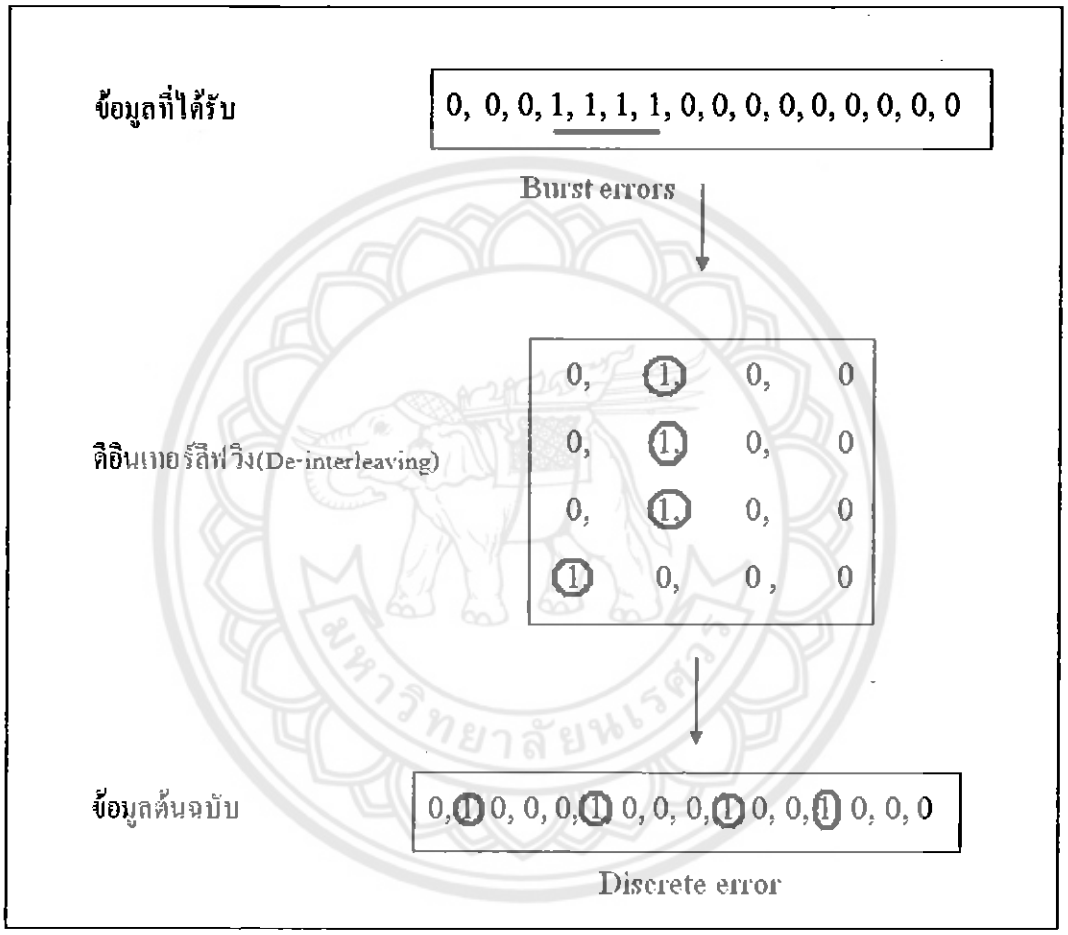
รูปที่ 2.12 หลักการทำงานของอินเทอร์ลีฟวิง [7]

2.5 การนำแฮมมิงโค้ดกับอินเตอร์ลีฟวิงมาประยุกต์

การนำแฮมมิงโค้ดกับอินเตอร์ลีฟวิงมาประยุกต์[1]นั้น เพื่อแก้ไขความผิดพลาดของข้อมูลที่เกิดขึ้นระหว่างการส่งข้อมูล ถึงแม้ว่าแฮมมิงโค้ดจะใช้ได้กับความผิดพลาดของข้อมูลแบบบิตเดียว แต่สามารถที่นำแฮมมิงโค้ดมาประยุกต์ใช้กับกรณีที่มีความผิดพลาดข้อมูลแบบหลายบิตได้ ซึ่งวิธีการจะคล้ายคลึงกันแต่แทนที่จะส่งข้อมูลทั้งหมดออกไปในคราวเดียวกัน ก็จะใช้วิธีการแบ่งข้อมูลที่ต้องการจะส่งออกเป็นบล็อกย่อยๆ n บล็อก จากนั้นจึงส่งข้อมูลบิตแรกของแต่ละบล็อกไป

ด้วยกัน เสร็จแล้วจึงส่งบิตที่สองของแต่ละบล็อกออกไปอีก เช่นนี้ไปจนกระทั่งหมดข้อมูลของทุกบล็อก จะเห็นได้ว่าการแบ่งข้อมูลแล้วส่งออกไปแบบนี้ถ้ามีสัญญาณรบกวนเกิดขึ้นในการส่ง จะส่งผลเพียงแค่บิตเดียวของแต่ละบล็อกเท่านั้น

เมื่อข้อมูลเดินทางถึงผู้รับแล้ว จะต้องนำมาเรียงข้อมูลของแต่ละคอลัมน์ที่ได้รับมา ซึ่งจากรูปที่ 2.13 จะเห็นได้ว่าในแต่ละบล็อกจะมีความผิดพลาดแบบบิตเดียวเท่านั้นดังนั้นจึงสามารถใช้แฮมมิงโค้ดในการแก้ไขความผิดพลาดของข้อมูลได้โดยอัตโนมัติ



รูปที่ 2.13 ตัวอย่างการกระจายความผิดพลาดวิธีอินเทอร์ลีฟ [7]

เทคนิคในการแบ่งข้อมูลเป็นบล็อกย่อยๆ แล้วส่งออกไปนั้น ถ้าจะให้สามารถตรวจสอบและแก้ไขข้อมูลที่ผิดพลาดได้โดยอัตโนมัติได้นั้น จะต้องอยู่บนเงื่อนไขที่ว่าข้อมูลของแต่ละบล็อกจะต้องมีความผิดพลาดของข้อมูลไม่เกิน 1 บิต

บทที่ 3
การออกแบบโปรแกรม

จากบทที่ผ่านมาเราได้ศึกษาในส่วนของทฤษฎีเกี่ยวกับวิธีการแก้ไขความผิดพลาด (errorcorrection) [2-5] ของข้อมูล ประเภทของความผิดพลาด(Type of error) [5] การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด [1-5, 9] และการแก้ไขความผิดพลาดแบบหลายบิตติดกัน (brust error) [6-7, 11] หรือ อินเทอร์ลีฟวิง(Interleaving) [1, 6-8] ทำให้มีความเข้าใจในหลักการทำงานรวมไปถึงขั้นตอนแต่ละขั้นในการเข้ารหัสช่องสัญญาณ ซึ่งสามารถนำข้อมูลที่ได้ศึกษามาใช้เป็นพื้นฐานในการออกแบบและประยุกต์ใช้วิธีการต่างๆรวมเข้าด้วยกัน

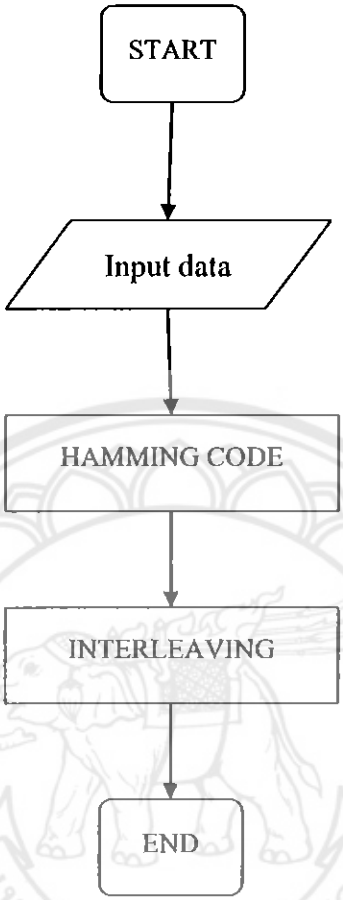
เราจะทำการออกแบบโปรแกรมโดยเราจะแบ่งส่วนของโปรแกรมออกเป็น 3 ส่วนหลักๆ โดยส่วนที่หนึ่งจะเป็นโปรแกรมที่อยู่ในส่วนของภาคส่งกล่าวคือทำการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดและทำอินเทอร์ลีฟวิง ส่วนที่สองก็จะเป็นส่วนของภาครับกล่าวคือจะทำการถอดรหัสข้อมูลที่ได้รับเพื่อให้ได้ข้อมูลที่แท้จริง ส่วนสุดท้ายจะเป็นส่วนที่จะทำให้ข้อมูลเกิดความผิดพลาดแบบหลายบิตขึ้นเพื่อใช้ทดสอบว่าโปรแกรมมีความสามารถที่จะแก้ไขความผิดพลาดที่เกิดขึ้นได้เหมือนกับที่ทฤษฎีได้กล่าวไว้หรือไม่

3.1 เป้าหมายการออกแบบโปรแกรม

เป้าหมายการออกแบบมีวัตถุประสงค์หลัก ดังนี้

1. เพื่อจัดลำดับขั้นตอนของแต่ละส่วนที่จะนำไปเขียนโปรแกรมเพื่อจะได้เกิดความสะดวก ไม่สับสนลำดับขั้นตอนในการเขียนโปรแกรม
2. เพื่อกำหนดให้การพัฒนาโปรแกรมเป็นไปตามแนวทางที่ได้กำหนดไว้
3. เพื่อให้เกิดความสะดวกในการใช้งาน
4. เพื่อให้ดูเข้าใจง่ายในส่วนของการแสดงผล
5. เพื่อให้ผู้ที่จะมาศึกษาส่วนของโค้ดโปรแกรมเข้าใจง่ายโดยส่วนของโปรแกรมจะมีการอธิบายไว้ด้วย

3.2 การออกแบบโปรแกรมของภาคส่ง

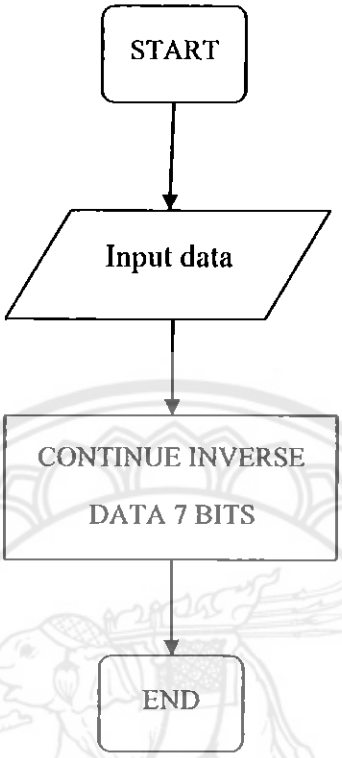


รูปที่ 3.1 โฟร์ชาร์จโปรแกรมของภาคส่ง

จากรูปที่ 3.1 เป็นโฟร์ชาร์จโปรแกรมของภาคส่งซึ่งจะมีลำดับขั้นตอนดังนี้

1. เริ่มต้นการทำงานของโปรแกรม
2. รับข้อมูลจากผู้ใช้
3. เข้ารหัสแฮมมิ่งโค้ด
4. ทำอินเตอร์ลีฟวิง
5. จบการทำงานของโปรแกรม

3.3 การออกแบบโปรแกรมที่ทำให้เกิดความผิดพลาดหลายบิตติดต่อกัน

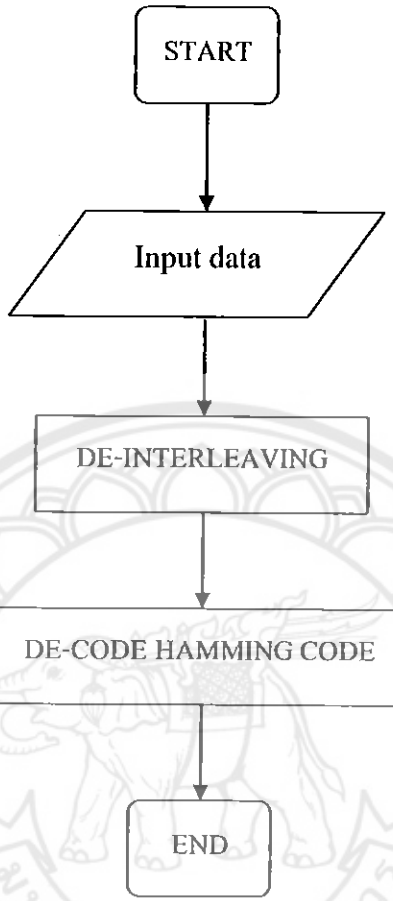


รูปที่ 3.2 โฟร์ชาร์จโปรแกรมที่ทำให้เกิดความผิดพลาดหลายบิตติดต่อกัน

จากรูปที่ 3.2 เป็นโฟร์ชาร์จโปรแกรมที่ทำให้เกิดความผิดพลาดหลายบิตติดต่อกันซึ่งจะมีลำดับขั้นตอนดังนี้

1. เริ่มต้นการทำงานของโปรแกรม
2. รับข้อมูล หรือ ข้อมูลที่รับมาจากภาคส่ง
3. เปลี่ยนข้อมูล 7 บิตติดต่อกัน เช่น เปลี่ยนข้อมูลบิต “0” เป็น “1” หรือ บิต “1” เป็น “0”
4. จบการทำงานของโปรแกรม

3.4 การออกแบบโปรแกรมของภาครับ

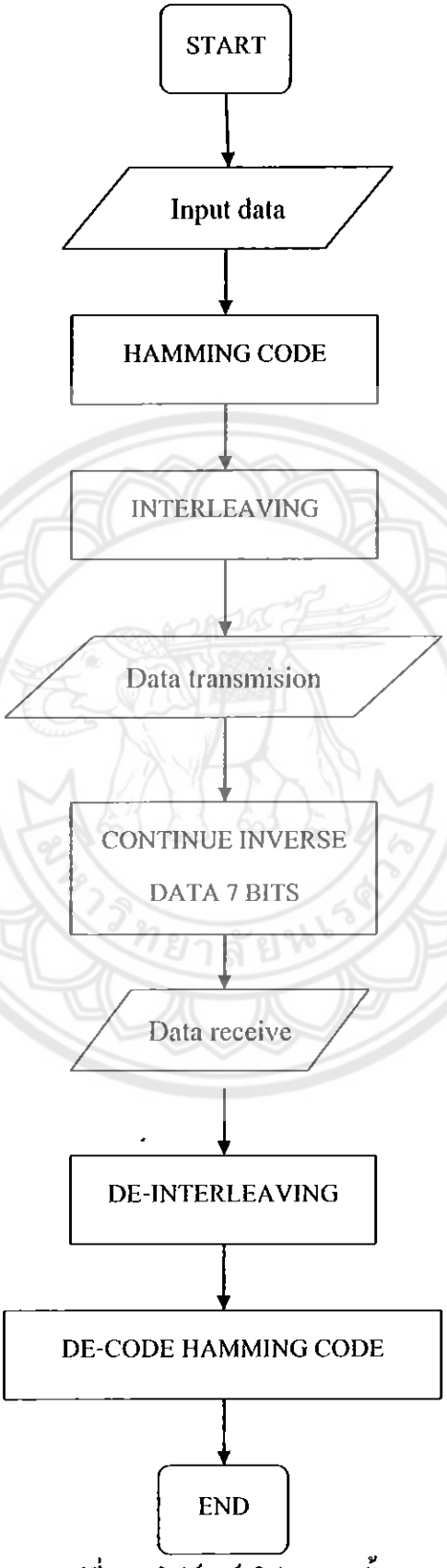


รูปที่ 3.3 โฟร์ชาร์จโปรแกรมของภาครับ

จากรูปที่ 3.3 เป็นโฟร์ชาร์จโปรแกรมของภาครับซึ่งจะมีลำดับขั้นตอนดังนี้

1. เริ่มต้นการทำงานของโปรแกรม
2. รับข้อมูล หรือ ข้อมูลที่รับมาจากโปรแกรมที่ทำให้เกิดความผิดพลาดหลายบิตติดต่อกัน
3. ทำอินเตอร์ลีฟวิงกลับ
4. ถอดรหัสแฮมมิงโค้ด
5. จบการทำงานของโปรแกรม

3.5 การออกแบบโปรแกรมทั้งหมด



รูปที่ 3.4 โฟร์ชาร์จโปรแกรมทั้งหมด

จากรูปที่ 3.4 เป็นโฟร์ชาร์จโปรแกรมทั้งหมดซึ่งจะมีลำดับขั้นตอนดังนี้

1. เริ่มต้นการทำงานของโปรแกรม
2. รับข้อมูลจากผู้ใช้
3. เข้ารหัสแฮมมิ่งโค้ด
4. ทำอินเตอร์ลีฟวิง
5. เปลี่ยนข้อมูล 7 บิตติดต่อกัน เช่น เปลี่ยนข้อมูลบิต "0" เป็น "1" หรือ บิต "1" เป็น "0"
6. ทำอินเตอร์ลีฟวิงกลับ
7. ถอดรหัสแฮมมิงโค้ด
8. จบการทำงานของโปรแกรม

บทที่ 4
การพัฒนาโปรแกรม

การพัฒนาโปรแกรมที่ใช้ในการจำลองการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดที่มีการนำเอาการทำอิเทอร์ลีฟวิงมาใช้ด้วยเพื่อเพิ่มสมรรถนะของการแก้ไขความผิดพลาดของข้อมูลที่อาจเกิดขึ้นแบบที่มีความผิดพลาดแบบหลายบิตติดต่อกัน

ในโครงงานนี้เราจะใช้โปรแกรมแมทแลบเวอร์ชั่น 7.0 (matlab V 7.0) [10] มาใช้พัฒนาเพื่อจำลองการทำงานในการเข้ารหัสช่องสัญญาณดังได้กล่าวไปข้างต้น

4.1 การพัฒนาโปรแกรมของภาคส่ง

4.1.1 การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด

ลำดับขั้นตอนการทำงานของโปรแกรมมีดังนี้

1. รับข้อมูลผ่านทางคอมมานด์ในโปรแกรมแมทแลบจำนวน 49 บิตในรูปของเมตริกซ์ โดยให้แบ่งเป็นเมตริกซ์ขนาด 7 × 7
2. นำแต่ละแถวมาทำการจัดตำแหน่งตามหลักการการเข้ารหัสแบบแฮมมิงโค้ดกล่าวคือนำข้อมูล ต้นฉบับมารวมกับบิตตรวจสอบโดยวางตำแหน่งตามหลักการการเข้ารหัสแบบแฮมมิงโค้ด
3. คำนวณค่าบิตตรวจสอบตามหลักการการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด(คำนวณทีละแถว)
4. นำค่าที่คำนวณได้ไปใส่ไว้ในตำแหน่งที่ได้จัดไว้ให้สำหรับวางบิตตรวจสอบเมื่อจบขั้นตอนนี้เราก็จะได้รหัสแฮมมิงโค้ด โดยรหัสแฮมมิงโค้ดจะแสดงอยู่ในรูปเมตริกซ์ขนาด 7×11

4.1.2 การทำอินเทอร์ลีฟวิง(Interleaving)

ลำดับขั้นตอนการทำงานของโปรแกรมมีดังนี้

นำชุดรหัสแฮมมิงโค้ดที่ได้จากหัวข้อ 3.2.1 มาทำการอินเทอร์ลีฟวิง(Interleaving) ตามขั้นตอนการทำอินเทอร์ลีฟวิง

1. นำข้อมูลที่เก็บอยู่ในรูปเมตริกซ์แถวที่ 1 มาทำการเปลี่ยนข้อมูลที่อยู่ในแถวที่ 1 ไปเก็บเป็นข้อมูลในหลักที่ 1 ในเมตริกซ์ตัวใหม่

2. กลับไปทำข้อที่ 1 จนกระทั่งทำข้อมูลครบ 7 แถว เมื่อครบ 7 แถวแล้วเราจะได้ข้อมูลที่มีการทำอินเทอร์ลีฟวิง ซึ่งจัดอยู่ในรูปเมตริกซ์ขนาด 11×7

3. ข้อมูลชุดที่ได้นี้จะทำการส่งต่อไปยังส่วนโปรแกรมของภาครับเพื่อให้ภาครับทำการถอดรหัสเพื่อนำข้อมูลต้นฉบับกลับมา

4.2 การพัฒนาโปรแกรมที่ทำให้เกิดความผิดพลาดหลายบิต

ลำดับขั้นตอนการทำงานของโปรแกรมมีดังนี้

1. ทำการเปลี่ยนข้อมูลในหัวข้อ 3.2 จำนวนตั้งแต่ 1 บิตขึ้นไปแต่ไม่เกิน 7 บิต ติดต่อกัน
2. นำข้อมูลที่ได้ทำการเปลี่ยนข้อมูลส่งต่อไปยังโปรแกรมส่วนของภาครับเพื่อนำข้อมูลนั้นไปทำการแก้ไขและถอดรหัส

4.3 การออกแบบโปรแกรมของภาครับ

4.3.1 การทำดีอินเทอร์ลีฟวิง(De-interleaving)

ลำดับขั้นตอนการทำงานของโปรแกรมมีดังนี้

1. นำข้อมูลซึ่งอยู่ในรูปเมตริกซ์ที่ได้จากการเข้ารหัสช่องสัญญาณในหัวข้อ 3.2 หรือ 3.3 มาทำการอินเทอร์ลีฟวิง(Interleaving) ตามขั้นตอนการทำอินเทอร์ลีฟวิง กล่าวคือมาทำการเปลี่ยนข้อมูลที่อยู่ในแถวที่ 1 ไปเก็บเป็นข้อมูลในหลักที่ 1 ในเมตริกซ์ตัวใหม่
2. กลับไปทำข้อที่ 1 จนกระทั่งทำข้อมูลครบ 11 แถว เมื่อครบ 11 แถวแล้วเราจะได้ข้อมูลที่มีการทำอินเทอร์ลีฟวิง ซึ่งจัดอยู่ในรูปเมตริกซ์ขนาด 7×11 ซึ่งในขั้นตอนนี้เราก็จะได้รหัสแฮมมิงโค้ดโดยรหัสแฮมมิงโค้ดจะแสดงอยู่ในรูปเมตริกซ์ขนาด 7×11
3. นำข้อมูลชุดนี้ส่งไปยังโปรแกรมถอดรหัสช่องสัญญาณแบบแฮมมิงโค้ด

4.3.2 การถอดรหัสช่องสัญญาณแบบแฮมมิงโค้ด

ลำดับขั้นตอนการทำงานของโปรแกรมมีดังนี้

1. รับข้อมูลจากขั้นตอนการทำอินเทอร์ลีฟวิง
2. นำแต่ละแถวของชุดข้อมูลมาทำการตามหลักการการถอดรหัสแบบแฮมมิงโค้ดกล่าวคือทำการคำนวณค่าบิตตรวจสอบว่าค่าที่ได้เป็นมีค่าเป็นเท่าไหร่
3. ถ้าค่าที่คำนวณได้มีค่าบิตตรวจสอบที่ได้มีค่าเป็น 0 แสดงว่าข้อมูลที่ได้รับมีความถูกต้องให้ทำการตัดบิตตรวจสอบออกก็จะได้ข้อมูลต้นฉบับมาแล้วให้แสดงผลข้อมูลและจบการทำงาน
4. ถ้าค่าที่ได้มานั้นมีค่าออกมาไม่ใช่ 0 แสดงว่าข้อมูลที่ได้รับมานั้นมีความผิดพลาดเกิดขึ้นให้คำนวณค่าจากบิตตรวจสอบออกมาว่าข้อมูลมีความผิดพลาดตำแหน่งใด
5. ให้โปรแกรมทำการแก้ไขข้อมูลในตำแหน่งที่มีความผิดพลาด เมื่อแก้ไขเสร็จแล้วให้แสดงผลข้อมูลและจบการทำงานของโปรแกรม

บทที่ 5
ตัวอย่างการทำงานของโปรแกรม

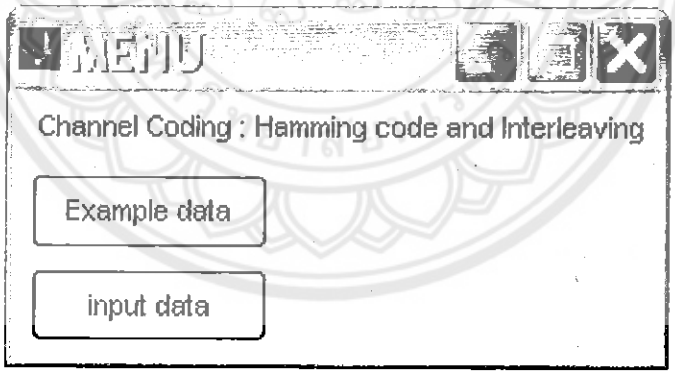
ในบทนี้จะแสดงตัวอย่างการทำงานของโปรแกรมเป็นลำดับขั้นตอนตามที่ได้ออกแบบโปรแกรมไว้ และจะอธิบายส่วนของการทำงานตามทฤษฎีไว้เพื่อให้ผู้ที่ศึกษาได้เข้าใจในที่มาของแต่ละส่วนของขั้นตอน

ข้อมูลชุดที่ต้องการส่งจะมีจำนวน 49 บิต ที่ใช้ 49 บิตเป็น 1 บล็อกข้อมูล เนื่องจากต้องการแก้ไขข้อมูลที่เกิดความผิดพลาด 7 บิตติดต่อกัน ซึ่งได้มาจากการคำนวณดังนี้

ข้อมูลที่จะเข้ารหัสแฮมมิงโค้ด 1 ชุด มี 7 บิต แฮมมิงโค้ด 1 ชุดจะมีความสามารถแก้ไขข้อมูลที่ผิดพลาดได้ 1 ตัวอักษร เพราะฉะนั้นจะแก้ไขข้อมูลผิดพลาด 7 บิตก็จะต้องมีข้อมูลทั้งหมด 7 ชุด ก็จะได้ข้อมูลทั้งหมด 49 บิต ก็จะให้ข้อมูล 49 บิต เป็น 1 บล็อก

เมื่อเราทำการเรียกฟังก์ชั่น HMIC(xXx) ในหน้า Command Window โดยที่ xXx คือตำแหน่งเริ่มต้นของบิตที่มีความผิดพลาดไป 7 บิต จะหน้าต่างขึ้นมาดังรูปที่ 5.1

ตัวอย่างนี้จะเรียกฟังก์ชั่น โดยให้บิตผิดพลาดเริ่มที่บิตที่ 3 ไป 7 บิต ทำการเรียกฟังก์ชั่นดังนี้ HMIC(3)



รูปที่ 5.1 เมนูเริ่มต้นเมื่อทำการเรียกฟังก์ชั่น

จากรูปที่ 5.1 เมื่อเรากดปุ่ม Example data โปรแกรมจะทำการรันโดยมีข้อมูลที่โปรแกรมได้เก็บเอาไว้เป็นตัวอย่างมีข้อมูลดังนี้

H = 1 0 0 1 0 0 0

a = 1 1 0 0 0 0 1

m = 1 1 0 1 1 0 1

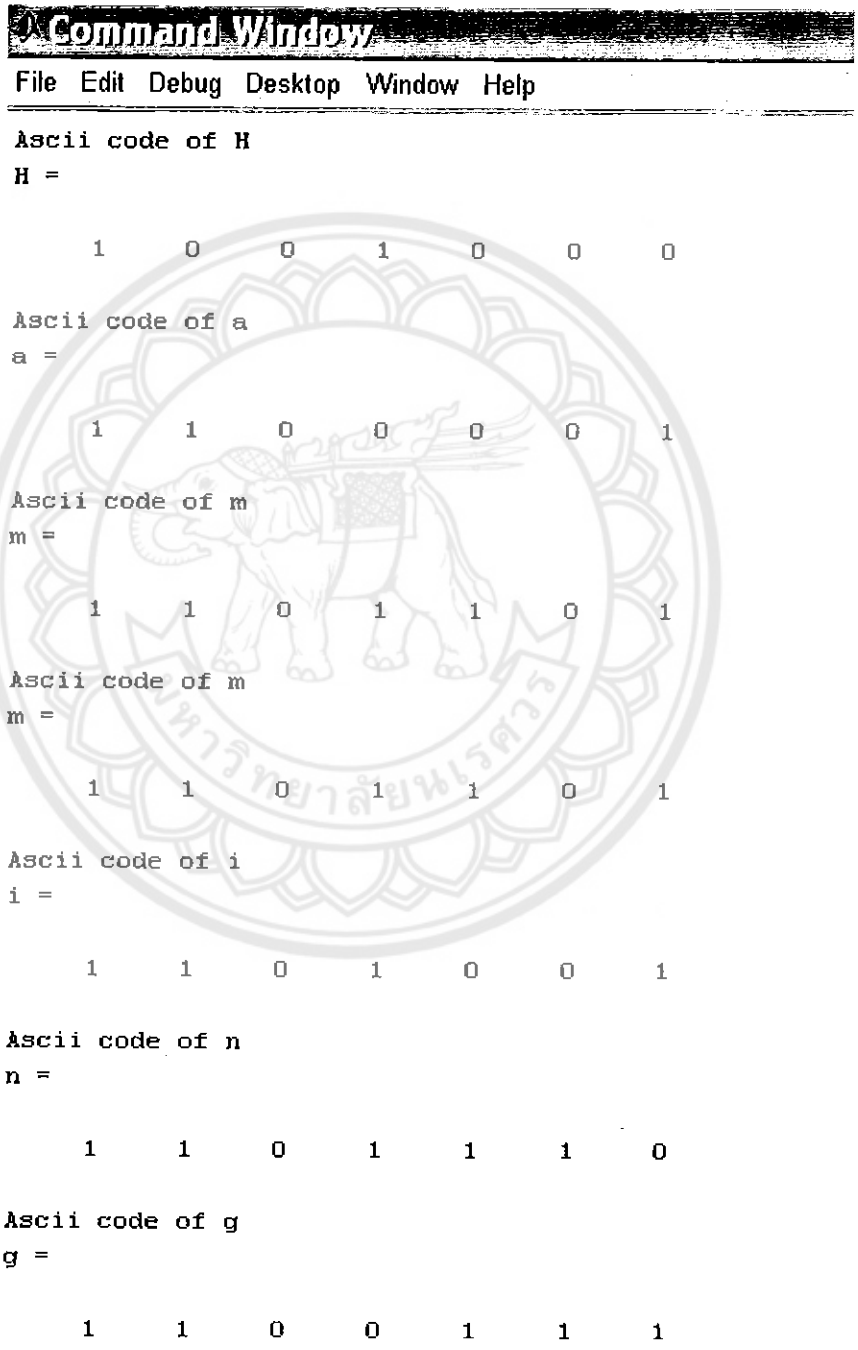
m2 = 1 1 0 1 1 0 1

i = 1 1 0 1 0 0 1

n = 1 1 0 1 1 1 0

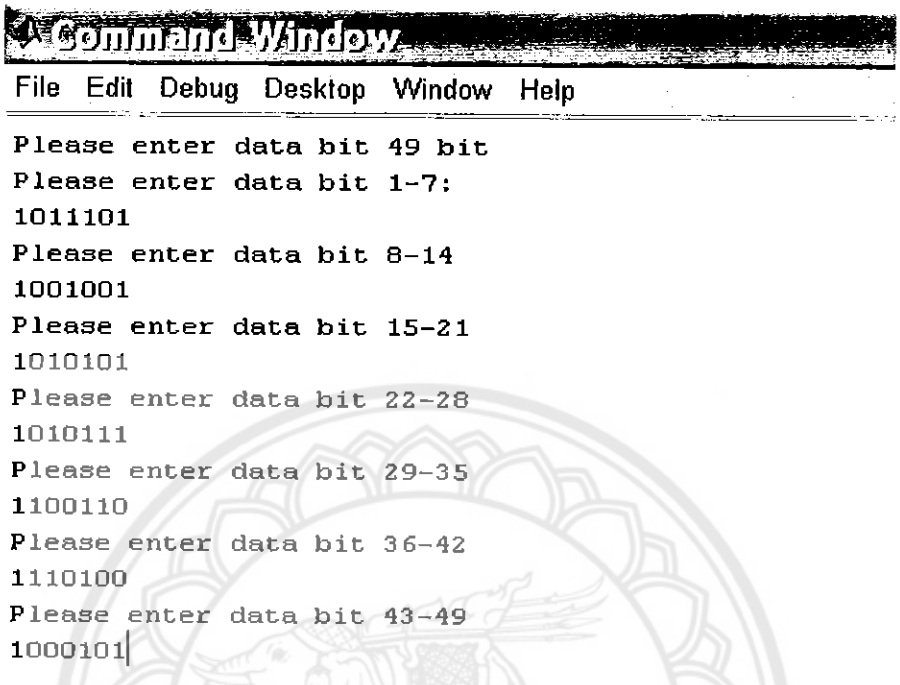
g = 1 1 0 0 1 1 1

แล้วโปรแกรมจะแสดงข้อมูลดังนี้



รูปที่ 5.2 โปรแกรมแสดงข้อมูลตัวอย่าง

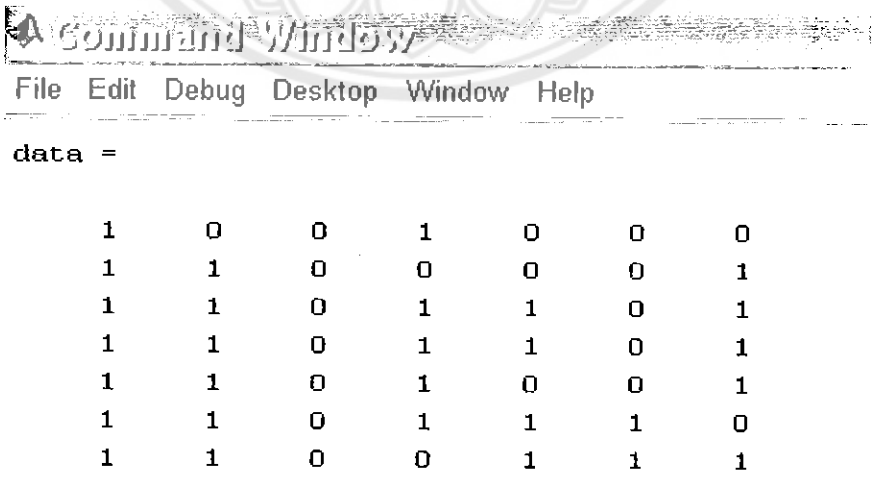
หรือ ถ้าเรากดปุ่ม input data โปรแกรมจะให้เราใส่ข้อมูลจำนวน 49 บิต โดยโปรแกรมจะให้ใส่ข้อมูลทีละ 7 บิต จำนวน 7 ชุด ดังรูปต่อไปนี้



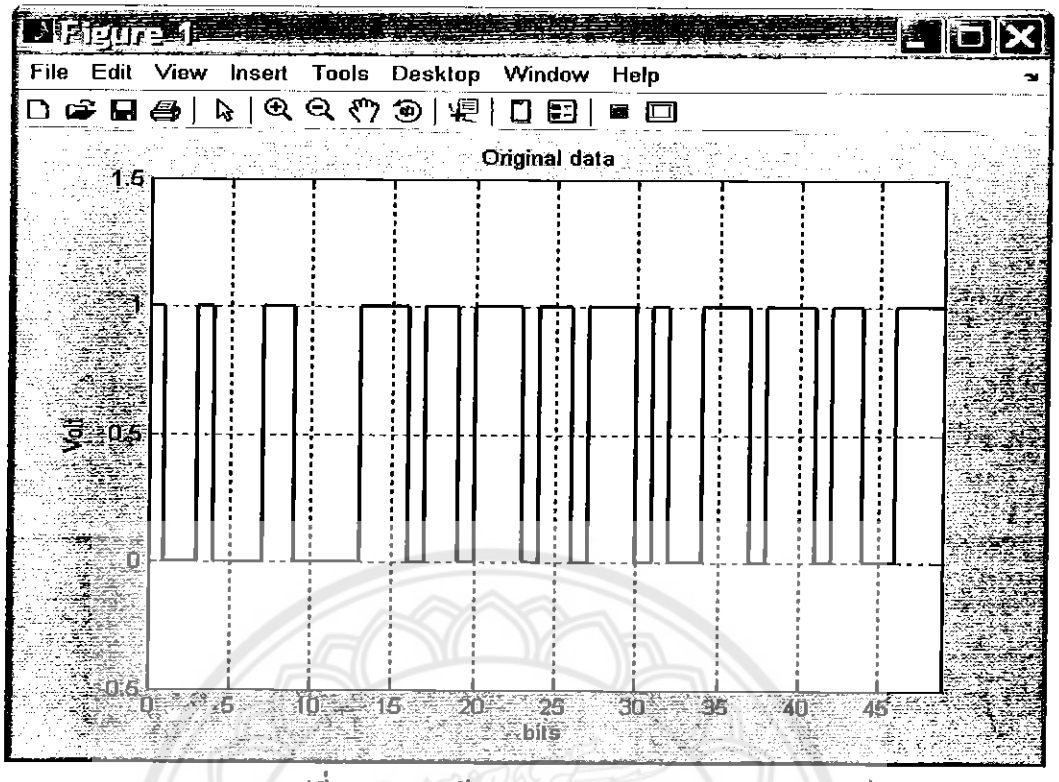
รูปที่ 5.3 โปรแกรมให้ใส่ข้อมูล 7 บิต 7 ชุด

5.1 ตัวอย่างโปรแกรมของภาคส่ง

ในตัวอย่างนี้เราจะทำการกดปุ่ม Example data เราจะได้ข้อมูลที่ได้เก็บไว้ในโปรแกรมเราจะนำมาเก็บให้อยู่ในรูปเมตริกซ์ขนาด 7 × 7 ดังรูป



รูปที่ 5.4 ข้อมูลที่ต้องการทำการเข้ารหัสช่องสัญญาณจัดอยู่ในรูปเมตริกซ์



รูปที่ 5.5 แสดงข้อมูลตัวอย่างในรูปของสัญญาณพัลส์

จากรูปที่ 5.4 ข้อมูลแต่ละชุดจะวางเป็นแถว เช่น แถวที่ 1 คือข้อมูลชุดที่ 1 แถวที่ 2 คือข้อมูลชุดที่ 2 แถวที่ 3 คือข้อมูลชุดที่ 3 ไปเรื่อยๆจนครบ 7 แถว ถ้าเราป้อนข้อมูลเองก็จะนำมาจัดเรียงในรูปแบบเดียวกัน

5.1.1 การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด

จากข้อมูลแถวที่ 1 โปรแกรมจะทำการจัดวางข้อมูลใหม่ดังนี้

create data word : _ _ 1 _ 0 0 1 _ 0 0 0 โดยช่องว่างที่เห็นใช้สำหรับใส่พาริตี้บิต

การคำนวณจะใช้ตำแหน่งต่างๆที่ใช้ในการตรวจสอบได้กล่าวไว้ในส่วนของทฤษฎีแล้ว กล่าวคือนำตำแน่งที่ว่าดังจะกล่าต่อไปนี้มาทำการมอดูเลตค่าที่ได้จะเป็นบิตพาริตี้

การคำนวณพาริตี้บิตแต่ละตำแหน่ง

ตำแหน่งที่ 1 ตรวจสอบบิตที่ 1,3,5,7,9,11

? _ 1 _ 0 0 1 _ 0 0 0 จากการคำนวนได้ พาริตี้คู่ ให้เซตตำแหน่งที่ 1 มีค่า "0" จะได้บิตข้อมูลเป็น **0 _ 1 _ 0 0 1 _ 0 0 0**

ตำแหน่งที่ 2 ตรวจสอบบิตที่ 2,3,6,7,10,11

0 ? 1 _ 0 0 1 _ 0 0 0 จากการคำนวนได้ พาริตี้คู่ ให้เซตตำแหน่งที่ 2 มีค่า "0" จะได้บิตข้อมูลเป็น **0 0 1 _ 0 0 1 _ 0 0 0**

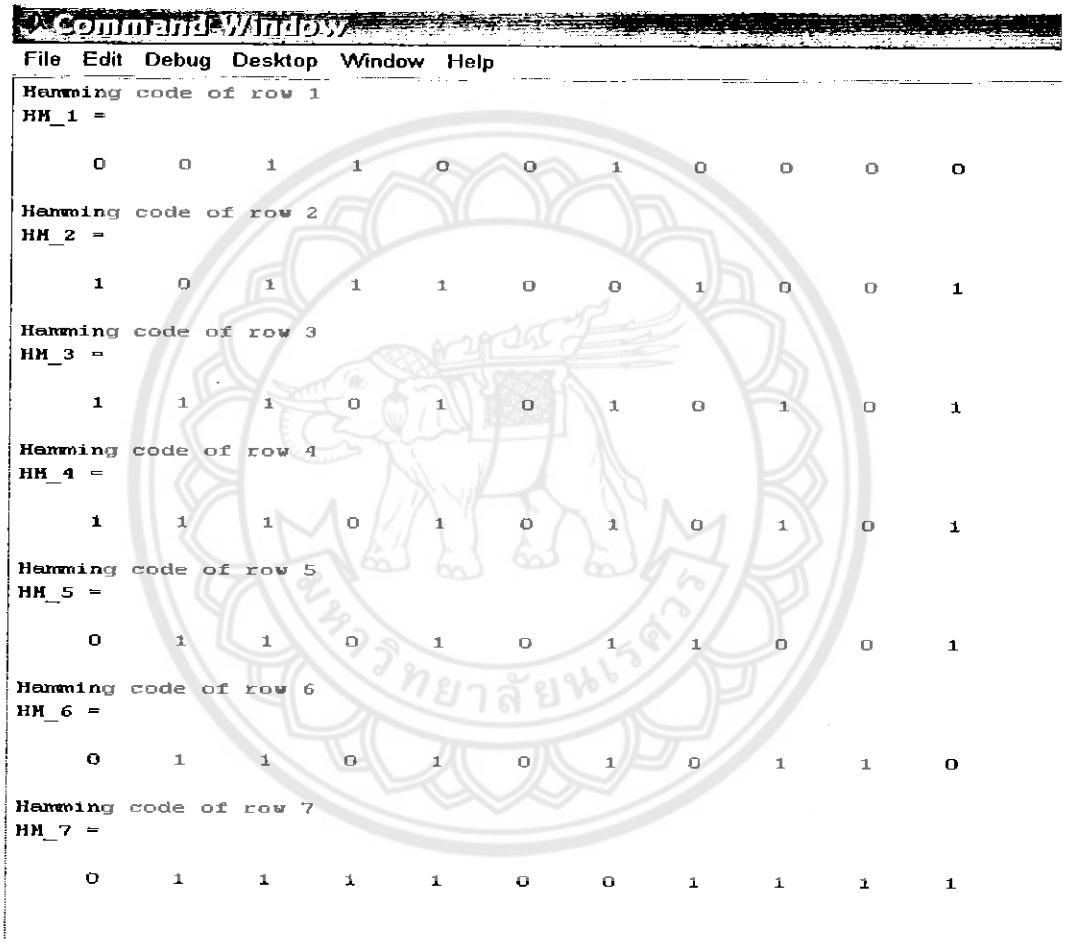
ตำแหน่งที่ 4 ตรวจสอบบิตที่ 4,5,6,7

0 0 1 ? 0 0 1 _ 0 0 0 จากการคำนวนได้ พาริตี้คี่ ให้เซตตำแหน่งที่ 4 มีค่า "1" จะได้บิตข้อมูลเป็น 0 0 1 1 0 0 1 _ 0 0 0

ตำแหน่งที่ 8 ตรวจสอบบิตที่ 8,9,10,11

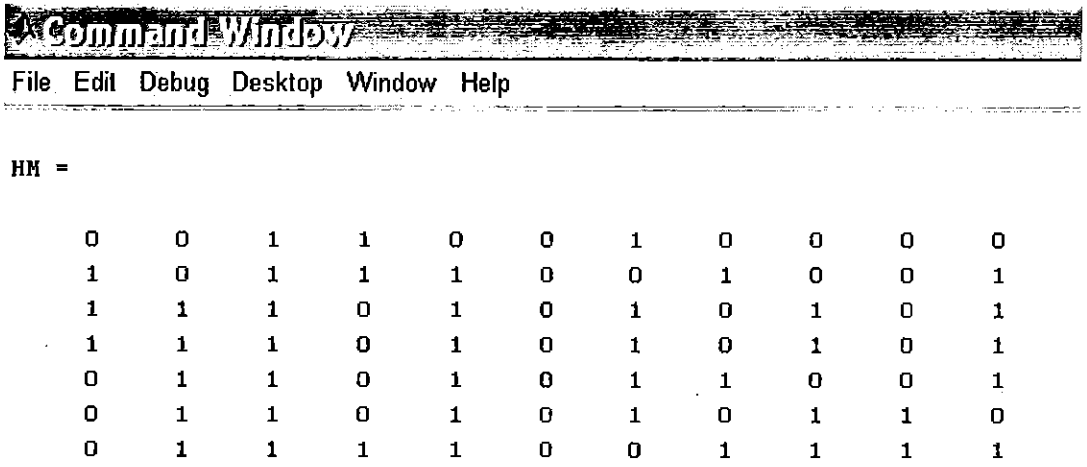
0 0 1 1 0 0 1 ? 0 0 0 จากการคำนวนได้ พาริตี้คู่ ให้เซตตำแหน่งที่ 1 มีค่า "0" จะได้บิตข้อมูลเป็น 0 0 1 1 0 0 1 0 0 0 0

ข้อมูลในแถวต่อๆไปก็ใช้หลักการเดียวกันจะได้ผลการเข้ารหัสแฮมมิงโค้ดดังรูปต่อไปนี้

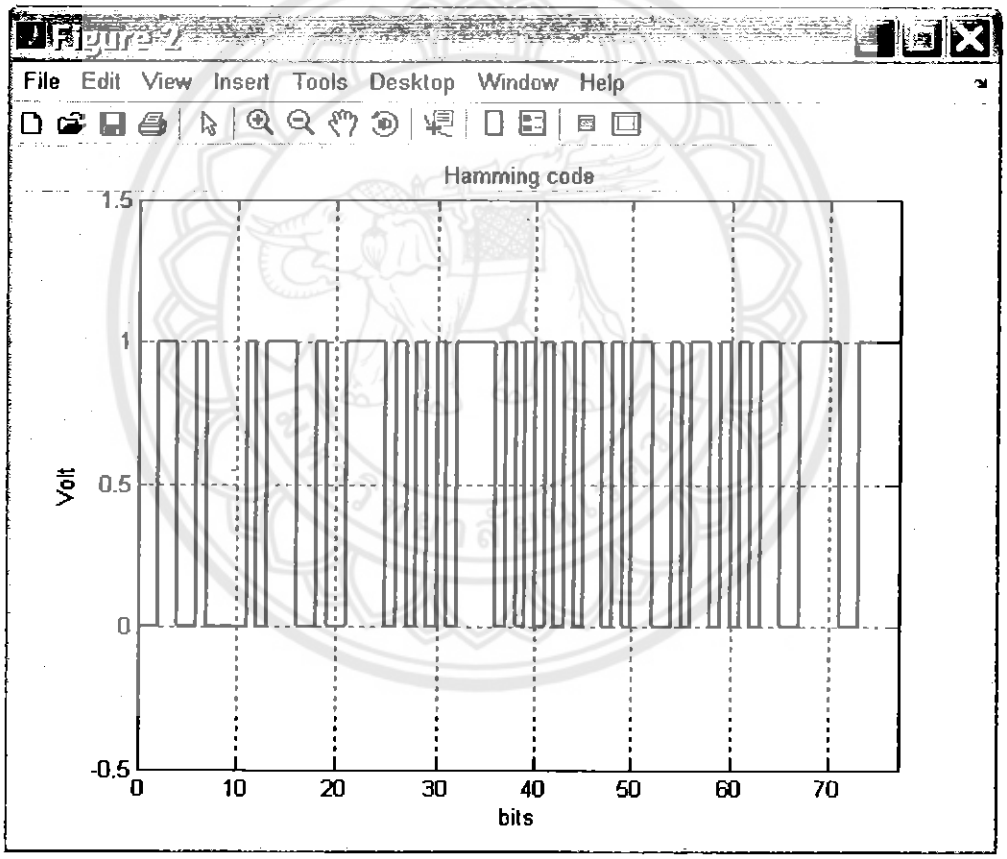


รูปที่ 5.6 ข้อมูลแต่ละชุดที่เข้ารหัสแฮมมิงโค้ดแล้ว

นำข้อมูลจากรูปที่ 5.6 นำมาเก็บให้อยู่ในรูปเมตริกซ์ขนาด 7 × 11 จะได้ดังรูปที่ 5.7



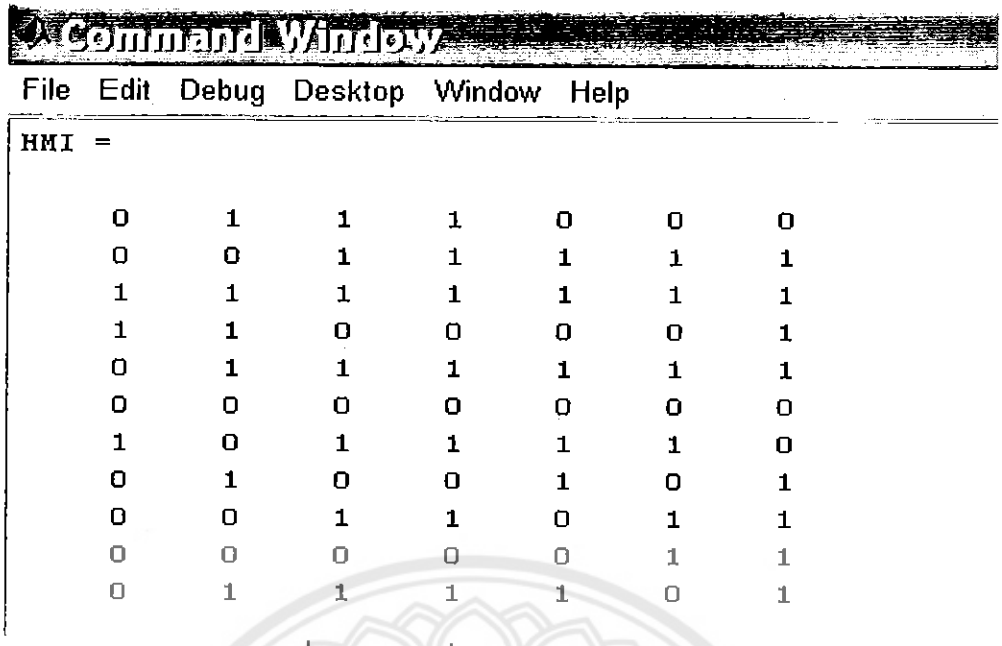
รูปที่ 5.7 ข้อมูลที่เข้ารหัสแฮมมิงโค้ดแล้วจัดอยู่ในรูปเมตริกซ์



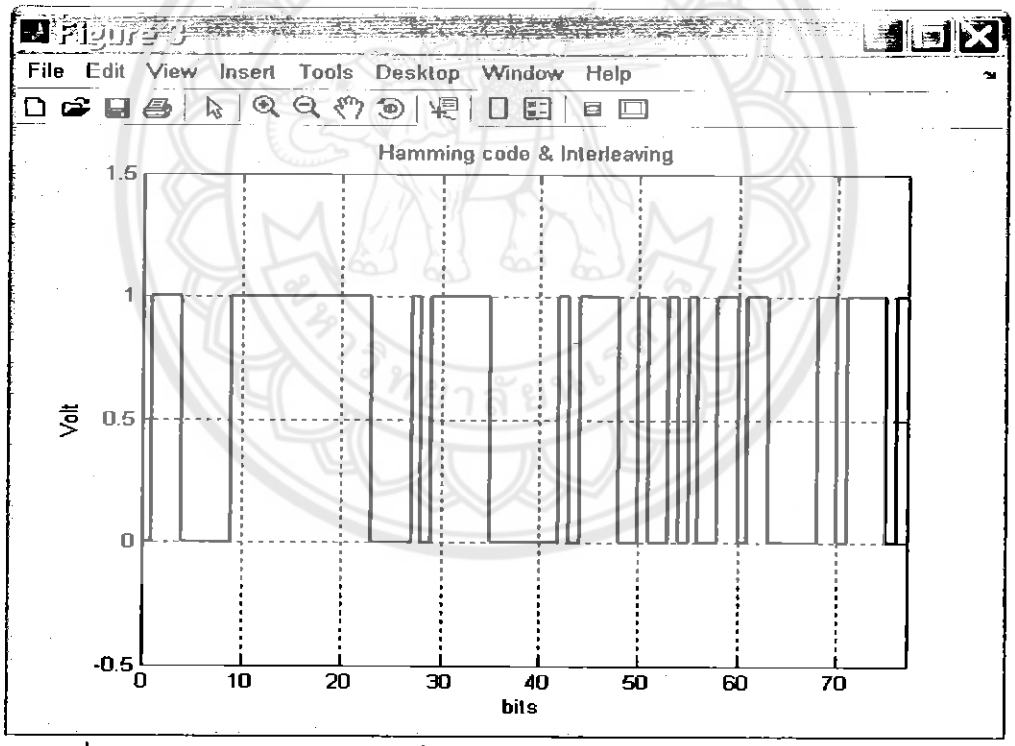
รูปที่ 5.8 แสดงข้อมูลที่เข้ารหัสแฮมมิงโค้ดในรูปของสัญญาณพัลส์

5.1.2 การทำอินเทอร์ลีฟวิง(Interleaving)

ทำการอินเทอร์ลีฟวิงข้อมูลที่ได้จากรูปที่ 5.6 กล่าวคือ อ่านข้อมูลในแถวที่ 1 มาเขียนใหม่ในหลักที่ 1 อ่านข้อมูลในแถวที่ 2 มาเขียนใหม่ในหลักที่ 2 อ่านข้อมูลในแถวที่ 3 มาเขียนใหม่ในหลักที่ 3 ทำตามนี้จนครบ 7 แถว ก็จะได้ข้อมูลที่อยู่ในรูปเมตริกซ์ขนาด 11 × 7 ดังรูป



รูปที่ 5.9 ข้อมูลที่ผ่านการทำอินเทอร์ลีฟวิง



รูปที่ 5.10 แสดงรหัสแฮมมิงโค้ดที่ผ่านการทำอินเทอร์ลีฟวิงในรูปของสัญญาณพัลส์

ข้อมูลที่ทำอินเทอร์ลีฟวิง หรือข้อมูลที่ส่งออกไป แล้วก็จะทำการส่งออกไปยังช่องสัญญาณทีละเริ่มจากแถวที่ 1 จะเริ่มส่งจากซ้ายไปขวา เมื่อหมดแถวที่ 1 ก็จะส่งแถวที่ 2 ต่อไปจนกระทั่งครบ 11 แถว

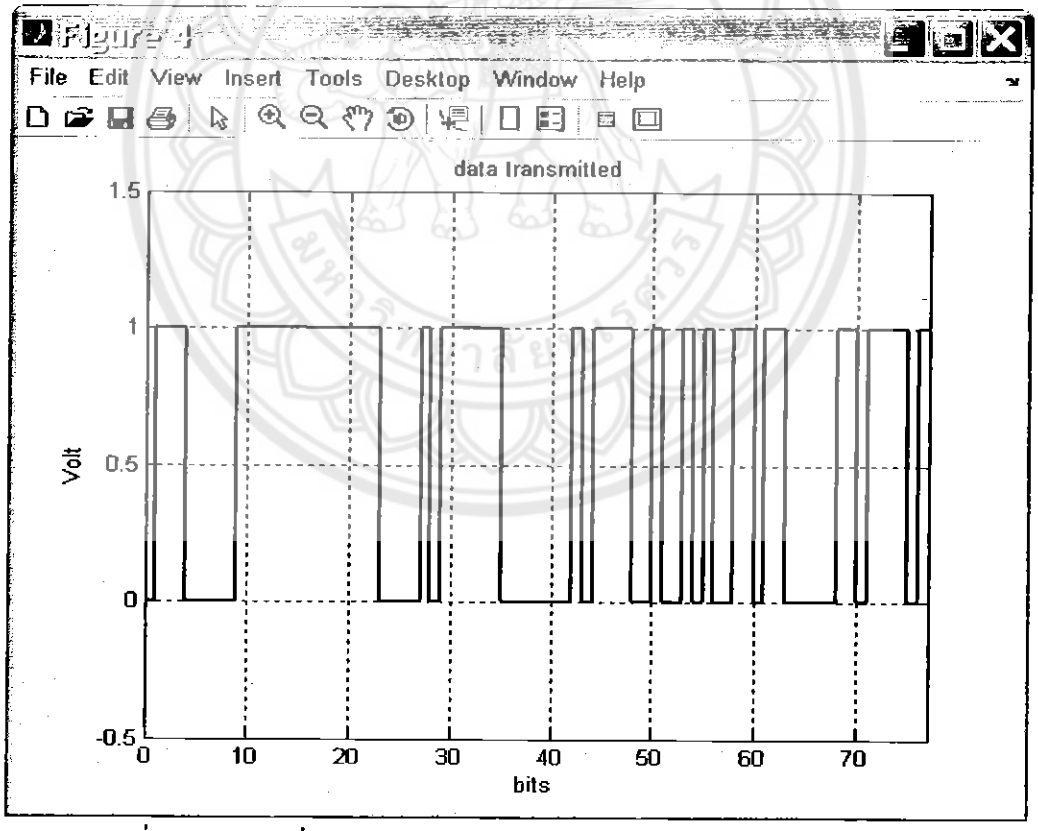
Command Window

File Edit Debug Desktop Window Help

```
Transmited data
TD =

     0     1     1     1     0     0     0
     0     0     1     1     1     1     1
     1     1     1     1     1     1     1
     1     1     0     0     0     0     1
     0     1     1     1     1     1     1
     0     0     0     0     0     0     0
     1     0     1     1     1     1     0
     0     1     0     0     1     0     1
     0     0     1     1     0     1     1
     0     0     0     0     0     1     1
     0     1     1     1     1     0     1
```

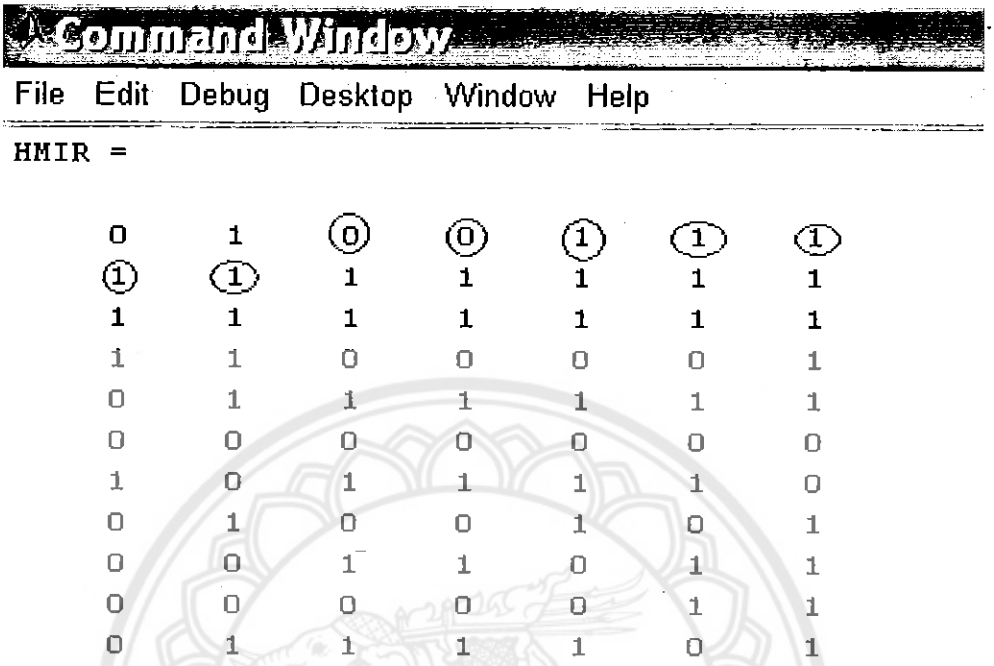
รูปที่ 5.11 ข้อมูลที่ทำการส่งออกไปยังช่องสัญญาณ



รูปที่ 5.12 ข้อมูลที่ทำการส่งออกไปยังช่องสัญญาณในรูปของสัญญาณพัลส์

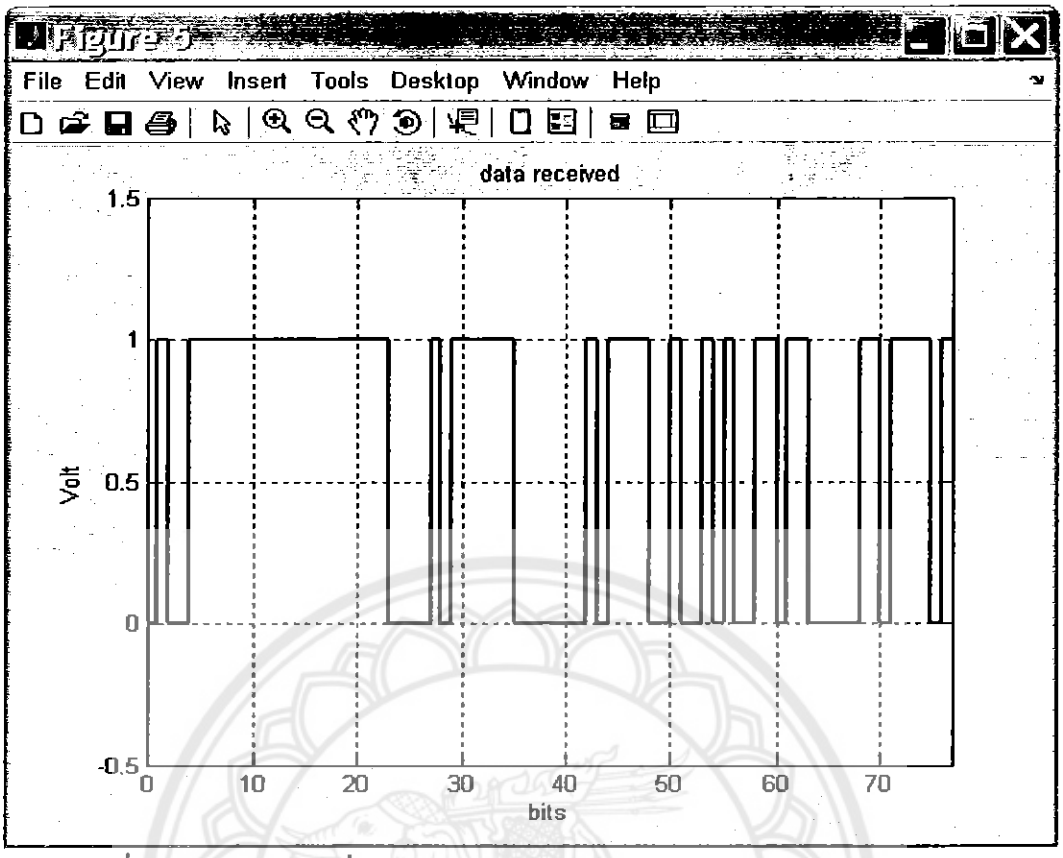
5.2 ตัวอย่างโปรแกรมที่ทำให้เกิดความผิดพลาดหลายบิต

โปรแกรมส่วนนี้จะทำการเปลี่ยนแปลงข้อมูลจากภาคส่งจำนวน 7 บิตติดต่อกัน

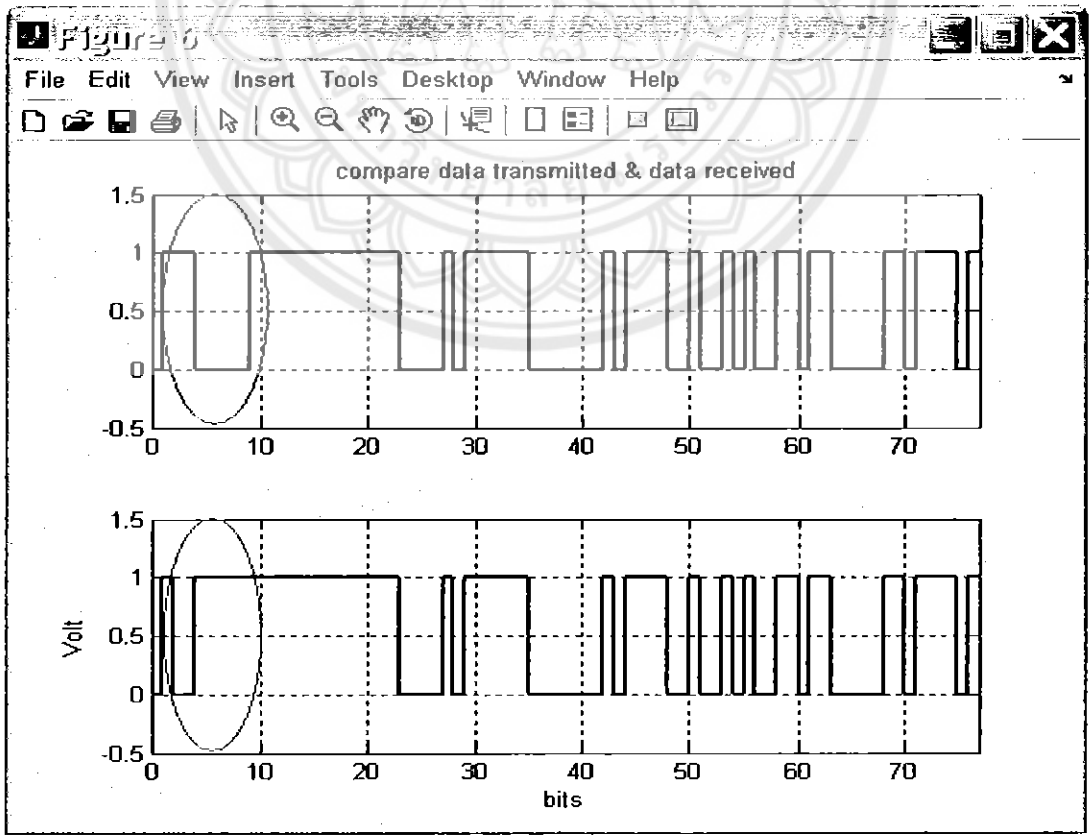


รูปที่ 5.13 แสดงข้อมูลที่เกิดความผิดพลาด 7 บิตติดต่อกัน

จากรูปที่ 5.13 ข้อมูลที่ได้จากภาคส่งได้ถูกโปรแกรมทำการเปลี่ยนข้อมูลเป็นจำนวน 7 บิตติดต่อกัน โดยตำแหน่งเริ่มต้นที่เกิดความผิดพลาดคือบิตที่ 3 เป็นต้นไปจนครบ 7 บิต(บิตที่ 3 – บิตที่ 9) เพื่อให้เห็นชัดเจนให้ดูเปรียบเทียบกับรูปที่ 5.11



รูปที่ 5.14 แสดงข้อมูลที่เกิดความผิดพลาด 7 บิตติดต่อกันในรูปของสัญญาณพัลส์



รูปที่ 5.15 สัญญาณเปรียบเทียบกันระหว่างสัญญาณที่ส่งและสัญญาณที่ถูกเปลี่ยนแปลง

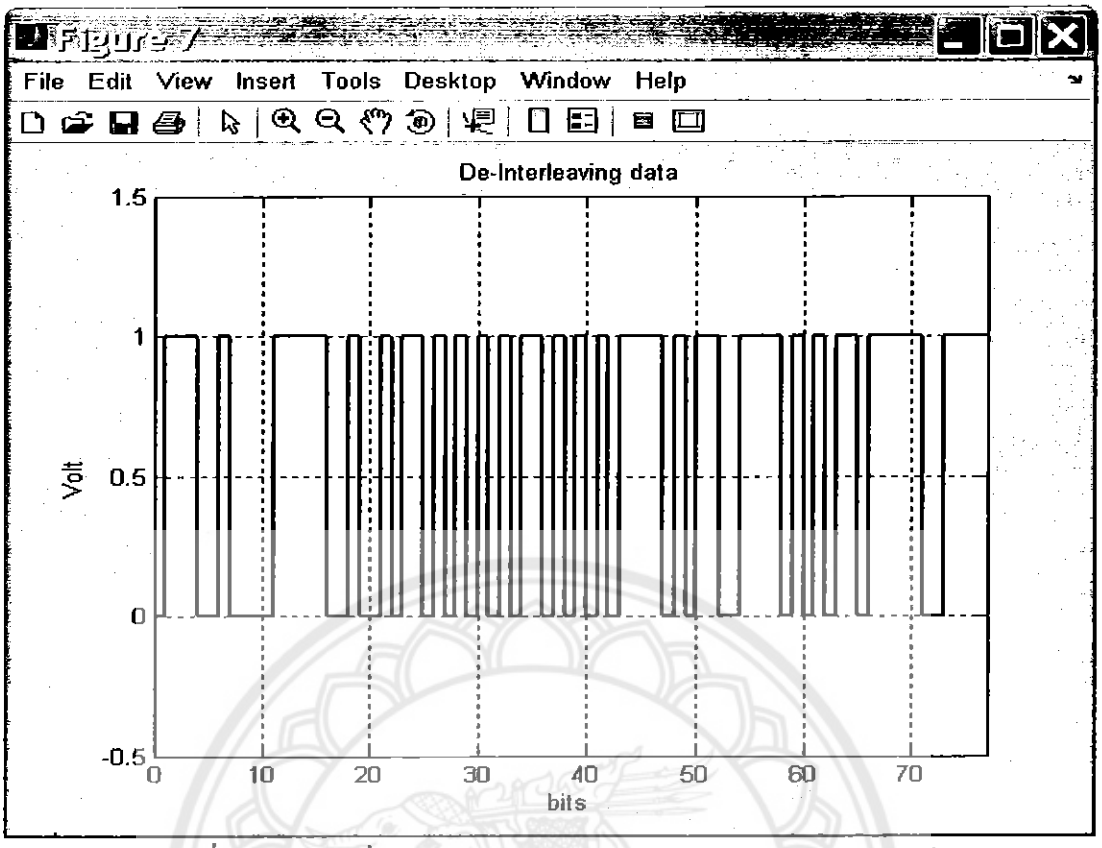
รูปที่ 5.15 เป็นการเปรียบเทียบสัญญาณพัลส์ที่ถูกส่งไปยังช่องสัญญาณและสัญญาณพัลส์ที่ได้เปลี่ยนแปลงไปในวงรีที่วงไว้นั่นหมายถึงตำแหน่งที่มีการเปลี่ยนแปลงไปของสัญญาณ หรือข้อมูลที่มีความผิดพลาด 7 บิตติดต่อกันนั่นเอง

5.3 การออกแบบโปรแกรมของภาครับ

5.3.1 การทำดีอินเทอร์ลีฟวิง(De-interleaving)

การทำดีอินเทอร์ลีฟวิงข้อมูลที่ได้จากรูปที่ 5.13 กล่าวคือ อ่านข้อมูลในแถวที่ 1 มาเขียนใหม่ในหลักที่ 1 อ่านข้อมูลในแถวที่ 2 มาเขียนใหม่ในหลักที่ 2 อ่านข้อมูลในแถวที่ 3 มาเขียนใหม่ในหลักที่ 3 ทำตามนี้จนครบ 11 แถว ก็จะได้ข้อมูลที่อยู่ในรูปเมตริกซ์ขนาด 7 × 11 ดังรูป

รูปที่ 5.16 ข้อมูลที่ผ่านการทำดีอินเทอร์ลีฟวิง



รูปที่ 5.17 ข้อมูลที่ผ่านการทำดีอินเทอร์ลีฟวิงในรูปของสัญญาณพัลส์

จากรูปที่ 5.14 ดูวงกลมที่วงไว้จะเห็นว่าข้อมูลที่ตอนแรกที่ได้รับมามีความผิดพลาดติดต่อกัน 7 บิต แต่เมื่อได้ผ่าการดีอินเทอร์ลีฟวิงบิตที่ผิดพลาดติดต่อกันนั้นได้กระจายออกจากกันทำให้แฮมมิงโค้ดสามารถทำการแก้ไขต่อไปได้

5.3.2 การถอดรหัสช่องสัญญาณแบบแฮมมิงโค้ด

จากข้อมูลแถวที่ 1 จากรูปที่ 5.14 โปรแกรมมีข้อมูลดังนี้

create data word : 0 1 1 1 0 0 1 0 0 0 0 โดยช่องว่างที่เห็นใช้สำหรับใส่พาริตี้บิต

การคำนวณจะใช้ตำแหน่งต่างๆที่ใช้ในการตรวจสอบหาความผิดพลาดได้ดังนี้

การคำนวณหาตำแหน่งความผิดพลาดที่เกิดขึ้นจะทำโดยนำตำแหน่งต่างๆดังจะกล่าวต่อไปนี้มามอดูเลตกันผลที่ได้จากการคำนวณเมื่อนำมาวางเรียงกันก็จะได้ตำแหน่งที่มีความผิดพลาด

บิตพาริตี้ที่ 1 ตรวจสอบบิตที่ 1,3,5,7,9,11

0 1 1 1 0 0 1 0 0 0 0 จากการคำนวนได้ พาริตี้คู่ ให้บิตพาริตี้ที่ 1 มีค่า "0"

บิตพาริตี้ที่ 2 ตรวจสอบบิตที่ 2,3,6,7,10,11

0 1 1 1 0 0 1 0 0 0 0 จากการคำนวนได้ พาริตี้คี่ ให้บิตพาริตี้ที่ 2 มีค่า "1"

บิตพาริตี้ที่ 4 ตรวจสอบบิตที่ 4,5,6,7

0 1 1 1 0 0 1 0 0 0 0 จากการคำนวนได้ พาริตี้คู่ ให้บิตพาริตี้ที่ 4 มีค่า "0"

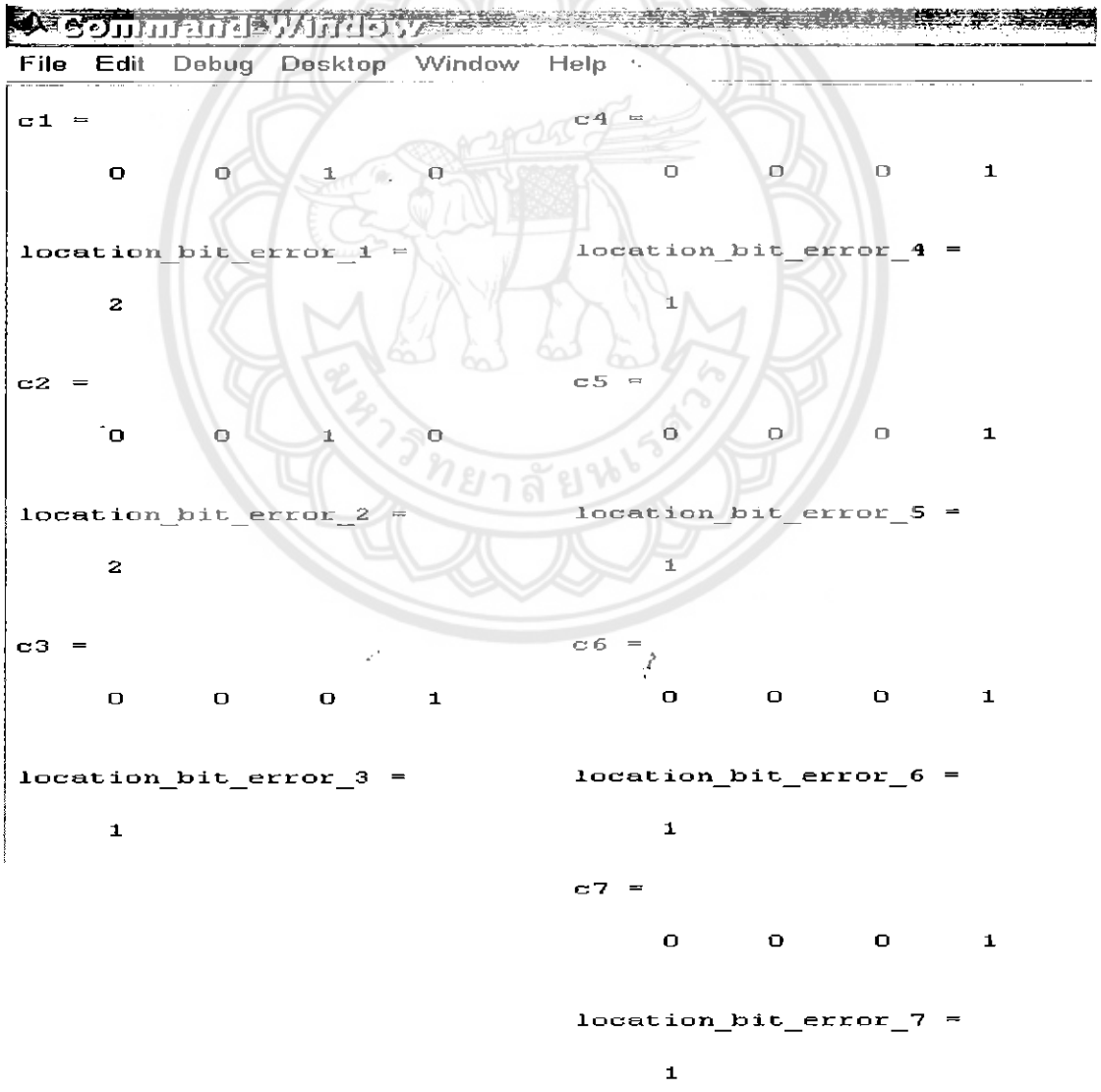
บิตพาริตี้ที่ 8 ตรวจสอบบิตที่ 8,9,10,11

0 1 1 1 0 0 1 **0 0 0 0** จากการคำนวนได้ พาริตี้คู่ ให้บิตพาริตี้ที่ 1 มีค่า "0"

นำข้อมูลที่ได้จากการคำนวณมาวางเรียงกันเริ่มจากบิตสูงไปหาบิตต่ำดังต่อไปนี้

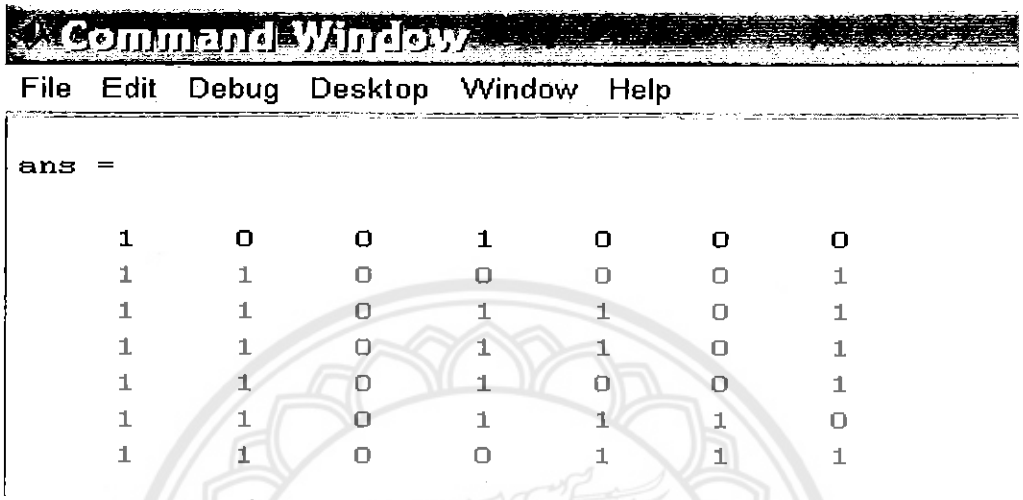
บิตพาริตี้ที่ 8, บิตพาริตี้ที่ 4, บิตพาริตี้ที่ 2, บิตพาริตี้ที่ 1 จะได้ 0 0 1 0 ซึ่งเมื่อแปลงเป็นเลขฐานสิบจะมีค่าเท่ากับ 2 ซึ่งค่าที่ได้นี้เองคือตำแหน่งที่ 2 เมื่อทราบตำแหน่งที่ผิดแล้วก็จะทำการแก้ไขข้อมูลในตำแหน่งที่ผิด หลังจานั้นให้ทำการตัดตำแหน่งของบิตพาริตี้ออกให้เหลือแต่ข้อมูล

ข้อมูลในแถวต่อๆไปที่จะถอดรหัสแฮมมิงโค้ดก็ใช้หลักการเดียวกันจะได้ผลการคำนวณตำแหน่งที่เกิดความผิดพลาดดังรูปต่อไปนี้

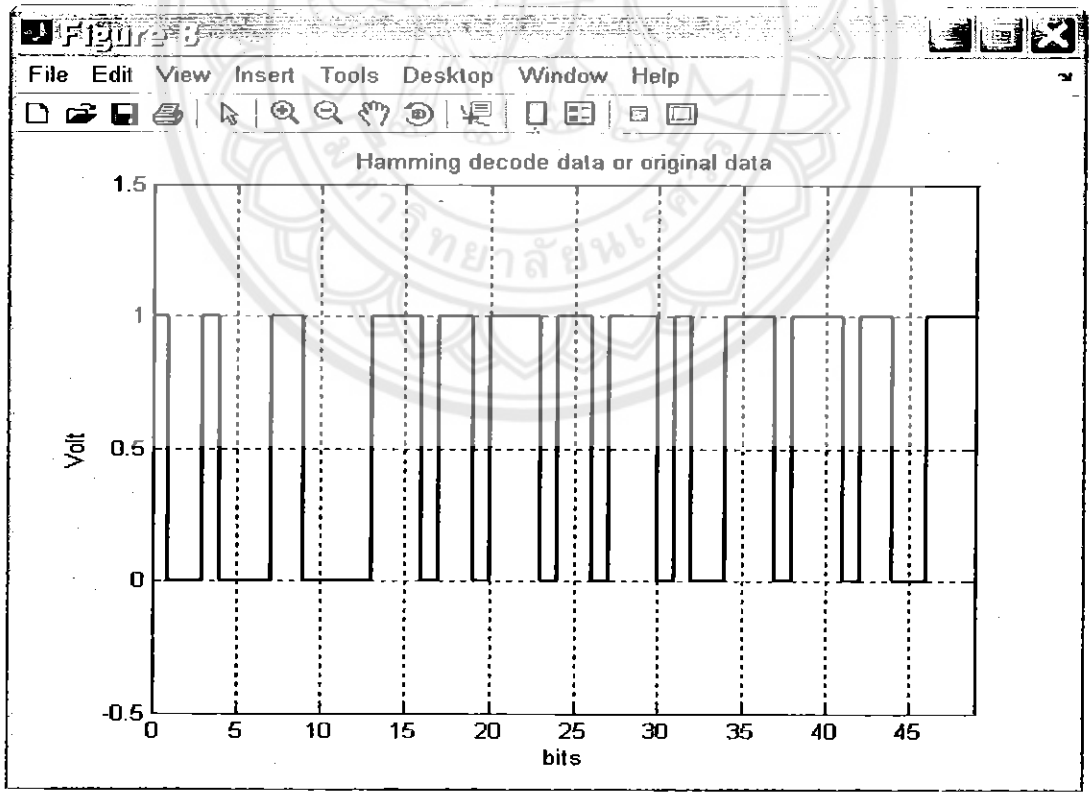


รูปที่ 5.18 ตำแหน่งที่เกิดความผิดพลาดที่เกิดขึ้น

จากรูปที่ 5.16 ตัวแปร c1, c2, c3, ..., c7 คือตำแหน่งที่ผิดพลาดในแถวที่ 1, 2, 3, ..., 7 ในรูปของเลขฐานสอง ส่วน location_bit_error_1, location_bit_error_2, location_bit_error_3, ..., location_bit_error_7 คือตำแหน่งที่ผิดพลาดในแถวที่ 1, 2, 3, ..., 7 ในรูปของเลขฐานสิบ เมื่อทำการถอดรหัสแฮมมิงโค้ดและแก้ไขข้อผิดพลาดแล้วจะได้ข้อมูลดังรูปต่อไปนี้



รูปที่ 5.19 ข้อมูลที่ได้หลังจากการถอดรหัสแฮมมิงโค้ด



รูปที่ 5.20 ข้อมูลที่ได้หลังจากการถอดรหัสแฮมมิงโค้ดในรูปของสัญญาณพัลส์

จากรูปที่ 5.17 ข้อมูลที่ได้นั้นมีข้อมูลเหมือนกับข้อมูลที่ได้กำหนดไว้ตอนเริ่มต้นก่อนที่จะทำการเข้ารหัสแฮมมิงโค้ดและทำอินเทอร์ลีฟวิง ดูเทียบกับรูปที่ 5.4 ผลที่ได้จากการรันโปรแกรมนี้ก็ได้บรรลุผลตามทฤษฎีที่ได้กล่าวไว้

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 บทสรุปในการดำเนินงาน

ในการพัฒนาโปรแกรมเพื่อจำลองการเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ดที่มีการนำเอาการทำอินเทอร์ลีฟวิงมาใช้ด้วยเพื่อเพิ่มสมรรถนะของการแก้ไขความผิดพลาดของข้อมูลที่อาจเกิดขึ้นแบบที่มีความผิดพลาดแบบหลายบิตติดต่อกันได้มีการทำงานเป็นขั้นตอน โดยขั้นตอนแรกเราได้ทำการศึกษาในส่วนของทฤษฎีเกี่ยวกับวิธีการแก้ไขความผิดพลาดของข้อมูล ประเภทของความผิดพลาด การเข้ารหัสช่องสัญญาณแบบแฮมมิงโค้ด และการแก้ไขความผิดพลาดแบบหลายบิตติดกัน หรือ อินเทอร์ลีฟวิง(Interleaving) ทำให้มีความเข้าใจในหลักการทำงานรวมไปถึงขั้นตอนแต่ละขั้นในการเข้ารหัสช่องสัญญาณ จากนั้นเราจึงออกแบบในส่วนของขั้นตอนการทำงานซึ่งได้นำเอาทฤษฎีที่ได้ศึกษามาใช้ในการออกแบบ เมื่อเราออกแบบโปรแกรมเสร็จแล้ว ขั้นต่อไปจึงเริ่มการพัฒนาโปรแกรมที่ใช้ในการจำลองการทำงาน โดยทำตามลำดับขั้นตอนที่เราได้ออกแบบมาก่อนหน้านี้มาเป็นแนวทาง เมื่อพัฒนาโปรแกรมเสร็จแล้วเราได้ทดสอบการทำงานของโปรแกรมว่ามีความถูกต้องหรือไม่ เมื่อโปรแกรมทำงานถูกต้องแล้วก็ได้ทำการปรับปรุงรูปแบบการป้อนค่าและการแสดงผลเพื่อให้มีการแสดงผลที่เข้าใจง่ายในการใช้งาน โดยส่วนของโปรแกรมได้มีการคอมเมนต์ไว้เพื่อที่ผู้ที่จะศึกษาหรือพัฒนาโปรแกรมต่อได้เกิดความเข้าใจง่ายขึ้น เราสามารถพัฒนาให้โปรแกรมสามารถเข้ารหัสช่องสัญญาณและทำการอินเทอร์ลีฟวิง จำลองการเกิดความผิดพลาดและสามารถตรวจสอบและแก้ไขข้อมูลที่ผิดพลาดติดต่อกัน 7 บิตได้ ซึ่งเป็นไปตามวัตถุประสงค์ที่วางไว้ทุกประการ

6.2 ปัญหาที่พบในการดำเนินงานปัญหาที่พบในการดำเนินงาน

1. การศึกษาตำราที่เป็นภาษาอังกฤษ ซึ่งมีความเข้าใจที่ยาก
2. การใช้งานโปรแกรมแมทแลบที่ต้องศึกษาในคู่มือนั้น การหาข้อมูลค่อนข้างยาก เนื่องจากบางคำที่เราค้นหานั้นไม่ตรงกับความต้องการของเรา
3. ด้านการพัฒนาโปรแกรมยังไม่สามารถพัฒนาให้มีความยืดหยุ่น และลดรูปในส่วนของโค้ดโปรแกรมให้มีขนาดสั้นลงได้
4. การจัดทำรูปเล่มโครงงานมีการใช้หลักภาษา และถ้อยคำสำนวนที่ทำให้ผู้อ่านเข้าใจง่าย นั้นเรียบเรียงถ้อยคำไม่ค่อยถูกต้องนัก

6.3 ข้อเสนอแนะ

การทำงานควรมีการวางแผนอย่างเป็นขั้นตอนและมีการวางตารางการทำงานไว้ การจัดตารางเวลาและแบ่งงานเป็นขั้นตอนนั้นจะทำให้เรารู้ว่า ขณะนี้เราทำงานช้าไป เร็วไป หรือว่าอยู่ในเวลาที่วางไว้อยู่ ทำให้สามารถแก้ไขข้อบกพร่องได้ทันท่วงที อีกทั้งการวางแผนงานควรมีการกำหนดการทำงานไว้อย่างดีและให้เสร็จก่อนกำหนดส่ง เนื่องจากว่าในความเป็นจริงนั้นงานส่วนมากเมื่อทำแล้วจะล่าช้า อันเป็นผลให้งานไม่เสร็จ

การศึกษาในเรื่องที่เกี่ยวข้องกับโครงงานช่วยให้มีความเข้าใจในโครงงานมากขึ้น และช่วยให้การทำงานสะดวก รวดเร็ว และเป็นไปในทางที่ถูกต้อง

ภาคผนวก

1. คู่มือการติดตั้งและการใช้งานโปรแกรม

โปรแกรมที่ใช้คือ Matlab 7.0

1.1 การติดตั้ง

คัดลอกไฟล์ HMIC.m ซึ่งเป็น M-file ลงใน directory ที่ต้องการเก็บไฟล์

1.2 การเรียกใช้

เรียกไฟล์ HMIC.m โดยพิมพ์ HMIC(xXx) ใน Command Window โดยที่ xXx คือ ตำแหน่งเริ่มต้นของบิตที่มีความผิดพลาดไป 7 บิต

1.3 ความช่วยเหลือ

พิมพ์ help HMIC ใน Command Window

2. โค้ดโปรแกรมที่จำลองการเข้ารหัสช่องสัญญาณ

```
function dbr=HMIC(xXx)
%xXx = Location start bit error (7 bit errors)
   %For example
   %Location errors bit 1-7 input xXx = 1
   %Location errors bit 2-8 input xXx = 2
%        .
%        .
%        .
   %Location errors bit 71-77 input xXx = 71

clc        %Clear Display command window

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      menu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MN=MENU('Channel Coding : Hamming code and Interleaving','Example data','input data');
switch(MN)
   case 1     %Example data
fprintf('Ascii code of H');
```

```
H=[1 0 0 1 0 0 0]
fprintf('Ascii code of a');
a=[1 1 0 0 0 0 1]
fprintf('Ascii code of m');
m=[1 1 0 1 1 0 1]
fprintf('Ascii code of m');
m2=[1 1 0 1 1 0 1]
fprintf('Ascii code of i');
i=[1 1 0 1 0 0 1]
fprintf('Ascii code of n');
n=[1 1 0 1 1 1 0]
fprintf('Ascii code of g');
g=[1 1 0 0 1 1 1]

   case 2    %Input data
fprintf('Please cnter data bit 49 bit\n');
fprintf('Please enter data bit 1-7: \n');
H=[];
input1=input('','s');
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
for ii=1:1:length(input1)
   if input1(ii)=='1'
      H(1,ii)=1;
   else
      H(1,ii)=0;
   end
end
H=H;
fprintf('Please cnter data bit 8-14 \n');
a=[];
input2=input('','s');
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
```

```
for ii=1:1:length(input2)
    if input2(ii)=='1'
       a(ii)=1;
    else a(ii)=0;
    end
end
a=a;
fprintf('Please enter data bit 15-21 \n');
m=[];
input3=input('','s');
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
for ii=1:1:length(input3)
    if input3(ii)=='1'
       m(ii)=1;
    else m(ii)=0;
    end
end
m=m;
fprintf('Please enter data bit 22-28 \n');
m2=[];
input4=input('','s');
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
for ii=1:1:length(input4)
    if input4(ii)=='1'
       m2(ii)=1;
    else m2(ii)=0;
    end
end
m2=m2;
fprintf('Please enter data bit 29-35 \n');
i=[];
input5=input('','s');
```

```
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
for ii=1:1:length(input5)
    if input5(ii)=='1'
       i(ii)=1;
    else i(ii)=0;
    end
end
i=i;
fprintf('Please enter data bit 36-42 \n');
n=[];
input6=input('','s');
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
for ii=1:1:length(input6)
    if input6(ii)=='1'
       n(ii)=1;
    else n(ii)=0;
    end
end
n=n;
fprintf('Please enter data bit 43-49 \n');
g=[];
input7=input('','s');
% นำข้อมูลที่ได้รับมา จัดให้อยู่ในรูปเมตริกซ์
for ii=1:1:length(input7)
    if input7(ii)=='1'
       g(ii)=1;
    else g(ii)=0;
    end
end
g=g;
end       %ออกจาก switch
```

```
% จัดตำแหน่งให้อยู่ในรูปเมตริกซ์ขนาด7*7
data=[H; a; m; m2; i; n; g]
datap=[H a m m2 i n g];    %data for plot

%****************** จัดตำแหน่งและคำนวนตามทฤษฎีแฮมมิ่งโค้ด  **************
data1=H;                %บิตข้อมูลที่ต้องการส่ง ชุดที่ 1
db1_1=data1(1,1);       %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_1=db1_1;
db2_1=data1(1,2);       %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
hm5_1=db2_1;
db3_1=data1(1,3);       %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_1=db3_1;
db4_1=data1(1,4);       %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_1=db4_1;
db5_1=data1(1,5);       %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
hm9_1=db5_1;
db6_1=data1(1,6);       %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_1=db6_1;
db7_1=data1(1,7);       %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_1=db7_1;

%  calculate parity bit(even)
p1_1=mod((db1_1+db2_1+db4_1+db5_1+db7_1),2);    %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_1=p1_1;
p2_1=mod((db1_1+db3_1+db4_1+db6_1+db7_1),2);    %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_1=p2_1;
p3_1=mod((db2_1+db3_1+db4_1),2);          %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_1=p3_1;
p4_1=mod((db5_1+db6_1+db7_1),2);          %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_1=p4_1;
```

```
%*************************************************************************

data2=a;                %บิตข้อมูลที่ต้องการส่ง ชุดที่ 2
db1_2=data2(1,1);       %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_2=db1_2;
db2_2=data2(1,2);       %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
hm5_2=db2_2;
db3_2=data2(1,3);       %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_2=db3_2;
db4_2=data2(1,4);       %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_2=db4_2;
db5_2=data2(1,5);       %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
hm9_2=db5_2;
db6_2=data2(1,6);       %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_2=db6_2;
db7_2=data2(1,7);       %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_2=db7_2;

%  calculate parity bit(even)
p1_2=mod((db1_2+db2_2+db4_2+db5_2+db7_2),2);    %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_2=p1_2;
p2_2=mod((db1_2+db3_2+db4_2+db6_2+db7_2),2);    %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_2=p2_2;
p3_2=mod((db2_2+db3_2+db4_2),2);        %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_2=p3_2;
p4_2=mod((db5_2+db6_2+db7_2),2);        %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_2=p4_2;
%*************************************************************************

data3=m;                %บิตข้อมูลที่ต้องการส่ง ชุดที่ 3
```

```
db1_3=data3(1,1);        %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_3=db1_3;
db2_3=data3(1,2);        %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
hm5_3=db2_3;
db3_3=data3(1,3);        %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_3=db3_3;
db4_3=data3(1,4);        %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_3=db4_3;
db5_3=data3(1,5);        %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
hm9_3=db5_3;
db6_3=data3(1,6);        %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_3=db6_3;
db7_3=data3(1,7);        %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_3=db7_3;

%  calculate parity bit(even)
p1_3=mod((db1_3+db2_3+db4_3+db5_3+db7_3),2);    %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_3=p1_3;
p2_3=mod((db1_3+db3_3+db4_3+db6_3+db7_3),2);    %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_3=p2_3;
p3_3=mod((db2_3+db3_3+db4_3),2);          %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_3=p3_3;
p4_3=mod((db5_3+db6_3+db7_3),2);          %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_3=p4_3;
%*************************************************************************

data4=m2;                %บิตข้อมูลที่ต้องการส่ง ชุดที่ 4
db1_4=data4(1,1);        %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_4=db1_4;
db2_4=data4(1,2);        %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
```

```
hm5_4=db2_4;
db3_4=data4(1,3);        %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_4=db3_4;
db4_4=data4(1,4);        %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_4=db4_4;
db5_4=data3(1,5);        %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
hm9_4=db5_4;
db6_4=data4(1,6);        %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_4=db6_4;
db7_4=data4(1,7);        %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_4=db7_4;

%  calculate parity bit(even)
p1_4=mod((db1_4+db2_4+db4_4+db5_4+db7_4),2);   %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_4=p1_3;
p2_4=mod((db1_4+db3_4+db4_4+db6_4+db7_4),2);   %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_4=p2_3;
p3_4=mod((db2_4+db3_4+db4_4),2);          %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_4=p3_3;
p4_4=mod((db5_4+db6_4+db7_4),2);          %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_4=p4_4;
%**************************************************************************

data5=i;              %บิตข้อมูลที่ต้องการส่ง ชุดที่ 5
db1_5=data5(1,1);        %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_5=db1_5;
db2_5=data5(1,2);        %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
hm5_5=db2_5;
db3_5=data5(1,3);        %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_5=db3_5;
```

```
db4_5=data5(1,4);       %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_5=db4_5;
db5_5=data5(1,5);       %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
hm9_5=db5_5;
db6_5=data5(1,6);       %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_5=db6_5;
db7_5=data5(1,7);       %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_5=db7_5;

%  calculate parity bit(even)
p1_5=mod((db1_5+db2_5+db4_5+db5_5+db7_5),2);   %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_5=p1_5;
p2_5=mod((db1_5+db3_5+db4_5+db6_5+db7_5),2);   %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_5=p2_5;
p3_5=mod((db2_5+db3_5+db4_5),2);          %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_5=p3_5;
p4_5=mod((db5_5+db6_5+db7_5),2);          %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_5=p4_5;
%*************************************************************************

data6=n;              %บิตข้อมูลที่ต้องการส่ง ชุดที่ 6
db1_6=data6(1,1);       %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_6=db1_6;
db2_6=data6(1,2);       %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
hm5_6=db2_6;
db3_6=data6(1,3);       %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_6=db3_6;
db4_6=data6(1,4);       %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_6=db4_6;
db5_6=data6(1,5);       %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
```

```
hm9_6=db5_6;
db6_6=data6(1,6);        %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_6=db6_6;
db7_6=data6(1,7);        %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_6=db7_6;

%  calculate parity bit(even)
p1_6=mod((db1_6+db2_6+db4_6+db5_6+db7_6),2);    %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_6=p1_6;
p2_6=mod((db1_6+db3_6+db4_6+db6_6+db7_6),2);    %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_6=p2_6;
p3_6=mod((db2_6+db3_6+db4_6),2);          %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_6=p3_6;
p4_6=mod((db5_6+db6_6+db7_6),2);          %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_6=p4_6;
%*************************************************************************

data7=g;              %บิตข้อมูลที่ต้องการส่ง ชุดที่ 7
db1_7=data7(1,1);        %บิตข้อมูลบิตที่ 1 หรือ hamming code บิตที่ 3
hm3_7=db1_7;
db2_7=data7(1,2);        %บิตข้อมูลบิตที่ 2 หรือ hamming code บิตที่ 5
hm5_7=db2_7;
db3_7=data7(1,3);        %บิตข้อมูลบิตที่ 3 หรือ hamming code บิตที่ 6
hm6_7=db3_7;
db4_7=data7(1,4);        %บิตข้อมูลบิตที่ 4 หรือ hamming code บิตที่ 7
hm7_7=db4_7;
db5_7=data7(1,5);        %บิตข้อมูลบิตที่ 5 หรือ hamming code บิตที่ 9
hm9_7=db5_7;
db6_7=data7(1,6);        %บิตข้อมูลบิตที่ 6 หรือ hamming code บิตที่ 10
hm10_7=db6_7;
```

```
db7_7=data7(1,7);        %บิตข้อมูลบิตที่ 7 หรือ hamming code บิตที่ 11
hm11_7=db7_7;

%  calculate parity bit(even)
p1_7=mod((db1_7+db2_7+db4_7+db5_7+db7_7),2);    %parity bit ที่ 1 หรือ hamming code บิตที่
1
hm1_7=p1_7;
p2_7=mod((db1_7+db3_7+db4_7+db6_7+db7_7),2);    %parity bit ที่ 2 หรือ hamming code บิตที่
2
hm2_7=p2_7;
p3_7=mod((db2_7+db3_7+db4_7),2);          %parity bit ที่ 3 หรือ hamming code บิตที่ 4
hm4_7=p3_7;
p4_7=mod((db5_7+db6_7+db7_7),2);          %parity bit ที่ 4 หรือ hamming code บิตที่ 8
hm8_7=p4_7;

%*********************  Hamming code ชุดที่ 1,2,3,4,5,6,7  ***************
fprintf('Hamming code of row 1');
HM_1=[hm1_1 hm2_1 hm3_1 hm4_1 hm5_1 hm6_1 hm7_1 hm8_1 hm9_1 hm10_1 hm11_1]
fprintf('Hamming code of row 2');
HM_2=[hm1_2 hm2_2 hm3_2 hm4_2 hm5_2 hm6_2 hm7_2 hm8_2 hm9_2 hm10_2 hm11_2]
fprintf('Hamming code of row 3');
HM_3=[hm1_3 hm2_3 hm3_3 hm4_3 hm5_3 hm6_3 hm7_3 hm8_3 hm9_3 hm10_3 hm11_3]
fprintf('Hamming code of row 4');
HM_4=[hm1_4 hm2_4 hm3_4 hm4_4 hm5_4 hm6_4 hm7_4 hm8_4 hm9_4 hm10_4 hm11_4]
fprintf('Hamming code of row 5');
HM_5=[hm1_5 hm2_5 hm3_5 hm4_5 hm5_5 hm6_5 hm7_5 hm8_5 hm9_5 hm10_5 hm11_5]
fprintf('Hamming code of row 6');
HM_6=[hm1_6 hm2_6 hm3_6 hm4_6 hm5_6 hm6_6 hm7_6 hm8_6 hm9_6 hm10_6 hm11_6]
fprintf('Hamming code of row 7');
HM_7=[hm1_7 hm2_7 hm3_7 hm4_7 hm5_7 hm6_7 hm7_7 hm8_7 hm9_7 hm10_7 hm11_7]
HM=[HM_1;HM_2;HM_3;HM_4;HM_5;HM_6;HM_7]
HMP=[HM_1 HM_2 HM_3 HM_4 HM_5 HM_6 HM_7];   %data for plot
```

```
%*********************  Interleaving ***************************
HMI1=HM(:,1)';              %ทำอินเตอร์ลิฟ
HMI2=HM(:,2)';
HMI3=HM(:,3)';
HMI4=HM(:,4)';
HMI5=HM(:,5)';
HMI6=HM(:,6)';
HMI7=HM(:,7)';
HMI8=HM(:,8)';
HMI9=HM(:,9)';
HMI10=HM(:,10)';
HMI11=HM(:,11)';
HMI=[HMI1;HMI2;HMI3;HMI4;HMI5;HMI6;HMI7;HMI8;HMI9;HMI10;HMI11] %ข้อมูลที่ทำ
อินเตอร์ลีฟ
HMIP=[HMI1 HMI2 HMI3 HMI4 HMI5 HMI6 HMI7 HMI8 HMI9 HMI10 HMI11];

%******************  ข้อมูลที่ถูกส่งออกไปยังช่องสัญญาณ  ****************
fprintf('Transmited data');
TD=[HMI1 ;HMI2 ;HMI3; HMI4; HMI5; HMI6; HMI7 ;HMI8 ;HMI9 ;HMI10; HMI11]
TDP=[HMI1 HMI2 HMI3 HMI4 HMI5 HMI6 HMI7 HMI8 HMI9 HMI10 HMI11];    %data for
plot

%*********************  plot data transmitted  *****************
Tmax4=length(TDP);
t4=linspace(0,Tmax4,Tmax4*2^10);
output4=[];
for ii=1:1:Tmax4
  if TDP(ii)==1
     output4=[output4 ones(1,2^10)];
  else
     output4=[output4 zeros(1,2^10)];
```

```
    end
end
figure(4)
plot(t4,output4,'Linewidth',2);grid on;
title('data transmitted');
ylabel('Volt');
xlabel('bits');
axis([0 Tmax4 -0.5 1.5])


%********************  เกิด burst errors ขึ้น  ********************
y=xXx;
yy=6+xXx;
for k=y:1:yy
   if TDP(k)==0
      TDP(k)=1;
   else TDP(k)=0;
   end

end
HMIRP=[TDP]    %data for plot
HMIR=[];


iii=1;
for i=1:1:11
for ii=1:1:7
    HMIR(i,ii)=HMIRP(iii);
    iii=iii+1;
end
end

HMIR=HMIR
```

```
%********************  plot data received   ******************
Tmax5=length(HMIRP);
t5=linspace(0,Tmax5,Tmax5*2^10);
output5=[];
for ii=1:1:Tmax5
   if HMIRP(ii)==1
      output5=[output5 ones(1,2^10)];
   else
      output5=[output5 zeros(1,2^10)];
   end
end
figure(5)
plot(t5,output5,'Linewidth',2);grid on;
title('data received');
ylabel('Volt');
xlabel('bits');
axis([0 Tmax5 -0.5 1.5])


%*******  plot compare data transmitted & data received  ***********
figure(6)
subplot(2,1,1);plot(t4,output4,'Linewidth',2);grid on;  axis([0 Tmax4 -0.5 1.5])
title('compare data transmitted & data received');
subplot(2,1,2);plot(t5,output5,'Linewidth',2);grid on;  axis([0 Tmax5 -0.5 1.5])
ylabel('Volt');
xlabel('bits');

%******************   interleaving (Receiver)   ***************
HMIR1=HMIR(:,1)';              %ทำอินเตอร์ลิฟกลับ
HMIR2=HMIR(:,2)';
HMIR3=HMIR(:,3)';
```

```
HMIR4=HMIR(:,4)';
HMIR5=HMIR(:,5)';
HMIR6=HMIR(:,6)';
HMIR7=HMIR(:,7)';
HMR=[HMIR1;HMIR2;HMIR3;HMIR4;HMIR5;HMIR6;HMIR7] %ข้อมูลที่ทำอินเตอร์ลีฟกลับ
HMRP=[HMIR1 HMIR2 HMIR3 HMIR4 HMIR5 HMIR6 HMIR7];  %data for plot

%************************  check bit error (even)  ***************
for l=1:1:7
c_1=mod((HMR(l,1)+HMR(l,3)+HMR(l,5)+HMR(l,7)+HMR(l,9)+HMR(l,11)),2);
c_2=mod((HMR(l,2)+HMR(l,3)+HMR(l,6)+HMR(l,7)+HMR(l,10)+HMR(l,11)),2);
c_3=mod((HMR(l,4)+HMR(l,5)+HMR(l,6)+HMR(l,7)),2);
c_4=mod((HMR(l,8)+HMR(l,9)+HMR(l,10)+HMR(l,11)),2);
if l==1
  c1=[c_4 c_3 c_2 c_1]   %  check bit error ข้อมูลชุดที่ 1z
  location_bit_error_1=(c1(1,1)*8+c1(1,2)*4+c1(1,3)*2+c1(1,4)*1)
else if l==2
    c2=[c_4 c_3 c_2 c_1]   %  check bit error ข้อมูลชุดที่ 2
  location_bit_error_2=(c2(1,1)*8+c2(1,2)*4+c2(1,3)*2+c2(1,4)*1)
 else if l==3
      c3=[c_4 c_3 c_2 c_1]   %  check bit error ข้อมูลชุดที่ 3
      location_bit_error_3=(c3(1,1)*8+c3(1,2)*4+c3(1,3)*2+c3(1,4)*1)
   else if l==4
        c4=[c_4 c_3 c_2 c_1]   %  check bit error ข้อมูลชุดที่ 4
        location_bit_error_4=(c4(1,1)*8+c4(1,2)*4+c4(1,3)*2+c4(1,4)*1)
      else if l==5
          c5=[c_4 c_3 c_2 c_1]   %  check bit error ข้อมูลชุดที่ 5
          location_bit_error_5=(c5(1,1)*8+c5(1,2)*4+c5(1,3)*2+c5(1,4)*1)
        else if l==6
          c6=[c_4 c_3 c_2 c_1]   %  check bit error ข้อมูลชุดที่ 6
          location_bit_error_6=(c6(1,1)*8+c6(1,2)*4+c6(1,3)*2+c6(1,4)*1)
          else
```

```
            c7=[c_4 c_3 c_2 c_1]    %  check bit error ข้อมูลชุดที่ 7
            location_bit_error_7=(c7(1,1)*8+c7(1,2)*4+c7(1,3)*2+c7(1,4)*1)
          end
        end
      end
    end
  end
end
end
location_bit_error=[location_bit_error_1;
          location_bit_error_2;
          location_bit_error_3;
          location_bit_error_4;
          location_bit_error_5;
          location_bit_error_6;
          location_bit_error_7;];

%******************** hamming decode (ตรวจสอบและแก้ไข) ***************
for n=1:1:7
z=location_bit_error(n,1);
switch (z)
  case 0  % ไม่เกิดข้อผิดพลาด

if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
    else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
        dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
      else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
          dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
          else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
```

```
                dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                    dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                    else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                        dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                    end
                end
            end
        end
    end
end
%**********************************************************************
case 1 %ผิดตำแหน่งบิตข้อมูลตัวที่ 1

if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  HMR(1,1)=mod(HMR(1,3)+1,2);
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
  else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
      HMR(2,1)=mod(HMR(2,3)+1,2);
        dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
       else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
           HMR(3,1)=mod(HMR(3,3)+1,2);
           dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
           else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
               HMR(4,1)=mod(HMIR(4,3)+1,2);
```

```
                dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                HMR(5,1)=mod(HMR(5,3)+1,2);
                dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                    HMR(6,1)=mod(HMR(6,3)+1,2);
                    dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                  else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                    HMR(7,1)=mod(HMR(7,3)+1,2);
                    dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                end
            end
        end
    end
end
%*************************************************************************
case 2 %ผิดตำแหน่งบิตข้อมูลตัวที่ 2

if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  HMR(1,2)=mod(HMR(1,5)+1,2);
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
  else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
      HMR(2,2)=mod(HMR(2,5)+1,2);
        dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
       else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
           HMR(3,2)=mod(HMR(3,5)+1,2);
```

```
            dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
                HMR(4,2)=mod(HMR(4,5)+1,2);
                dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
              else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                  HMR(5,2)=mod(HMR(5,5)+1,2);
                  dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                  else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                      HMR(6,2)=mod(HMR(6,5)+1,2);
                      dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                    else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                      HMR(7,2)=mod(HMR(7,5)+1,2);
                      dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                    end
                end
              end
            end
        end
end
%*************************************************************************
case 3 %ผิดตำแหน่งบิตข้อมูลตัวที่ 3

if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  HMR(1,3)=mod(HMR(1,6)+1,2);
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
  else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
      HMR(2,3)=mod(HMR(2,6)+1,2);
```

```
            dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
        else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
            HMR(3,3)=mod(HMR(3,6)+1,2);
            dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
                HMR(4,3)=mod(HMR(4,6)+1,2);
                dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                    HMR(5,3)=mod(HMR(5,6)+1,2);
                    dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                    else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                        HMR(6,3)=mod(HMR(6,6)+1,2);
                        dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                        else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                        HMR(7,3)=mod(HMR(7,6)+1,2);
                        dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                        end
                    end
                end
            end
        end
    end
end
%**************************************************************************
case 4 %ผิดตำแหน่งบิตข้อมูลตัวที่ 4
if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  HMR(1,4)=mod(HMR(1,7)+1,2);
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
```

```
  else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
        HMR(2,4)=mod(HMR(2,7)+1,2);
          dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
        else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
            HMR(3,4)=mod(HMR(3,7)+1,2);
            dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
                HMR(4,4)=mod(HMR(4,7)+1,2);
                dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                    HMR(5,4)=mod(HMR(5,7)+1,2);
                    dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                    else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                        HMR(6,4)=mod(HMR(6,7)+1,2);
                        dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                        else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                        HMR(7,4)=mod(HMR(7,7)+1,2);
                        dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                    end
                end
            end
        end
    end
end
%*************************************************************************
case 5 %ผิดตำแหน่งบิตข้อมูลตัวที่ 5
```

```
if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  HMR(1,5)=mod(HMR(1,9)+1,2);
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
  else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
       HMR(2,5)=mod(HMR(2,9)+1,2);
         dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
       else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
            HMR(3,5)=mod(HMR(3,9)+1,2);
            dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
                 HMR(4,5)=mod(HMR(4,9)+1,2);
                 dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                 else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                      HMR(5,5)=mod(HMR(5,9)+1,2);
                      dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                      else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                           HMR(6,5)=mod(HMR(6,9)+1,2);
                           dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                           else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                           HMR(7,5)=mod(HMR(7,9)+1,2);
                           dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                      end
                 end
            end
       end
  end
end
```

```
%**************************************************************************
case 6 %ผิดตำแหน่งบิตข้อมูลตัวที่ 6

if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
  HMR(1,6)=mod(HMR(1,10)+1,2);
  dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
  else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
       HMR(2,6)=mod(HMR(2,10)+1,2);
         dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
       else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
            HMR(3,6)=mod(HMR(3,10)+1,2);
            dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
            else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
                 HMR(4,6)=mod(HMR(4,10)+1,2);
                 dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                 else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                      HMR(5,6)=mod(HMR(5,10)+1,2);
                      dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
HMR(n,11)];
                      else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                           HMR(6,6)=mod(HMR(6,10)+1,2);
                           dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                      else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                           HMR(7,6)=mod(HMR(7,10)+1,2);
                           dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
HMR(n,10) HMR(n,11)];
                      end
                 end
            end
```

```
            end
        end
    end
    %*************************************************************************
    case 7 %ผิดตำแหน่งบิตข้อมูลตัวที่ 7

    if n==1 %ถอดรหัสแฮมมิ่ง ชุดที่ 1 (H)
      HMR(1,7)=mod(HMR(1,11)+1,2);
      dbr1=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
      else if n==2 %ถอดรหัสแฮมมิ่ง ชุดที่ 2 (a)
          HMR(2,7)=mod(HMR(2,11)+1,2);
            dbr2=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10) HMR(n,11)];
          else if n==3 %ถอดรหัสแฮมมิ่ง ชุดที่ 3 (m)
              HMR(3,7)=mod(HMR(3,11)+1,2);
              dbr3=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
    HMR(n,11)];
              else if n==4 %ถอดรหัสแฮมมิ่ง ชุดที่ 4 (m)
                  HMR(4,7)=mod(HMR(4,11)+1,2);
                  dbr4=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
    HMR(n,11)];
                else if n==5 %ถอดรหัสแฮมมิ่ง ชุดที่ 5 (i)
                    HMR(5,7)=mod(HMR(5,11)+1,2);
                    dbr5=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9) HMR(n,10)
    HMR(n,11)];
                  else if n==6 %ถอดรหัสแฮมมิ่ง ชุดที่ 6 (n)
                      HMR(6,7)=mod(HMR(6,11)+1,2);
                      dbr6=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
    HMR(n,10) HMR(n,11)];
                    else  %ถอดรหัสแฮมมิ่ง ชุดที่ 7 (g)
                      HMR(7,7)=mod(HMR(7,11)+1,2);
                      dbr7=[HMR(n,3) HMR(n,5) HMR(n,6) HMR(n,7) HMR(n,9)
    HMR(n,10) HMR(n,11)];
```

```
                    end
                end
            end
        end
    end
end

end
dbr=[dbr1;dbr2;dbr3;dbr4;dbr5;dbr6;dbr7];
dbrp=[dbr1 dbr2 dbr3 dbr4 dbr5 dbr6 dbr7];  %data for plot

%************************ plot Original data ******************************
Tmax1=length(datap);
t1=linspace(0,Tmax1,Tmax1*2^10);
output=[];
for ii=1:1:Tmax1
  if datap(ii)==1
    output=[output ones(1,2^10)];
  else
    output=[output zeros(1,2^10)];
  end
end
figure(1)
plot(t1,output,'Linewidth',2);grid on;
title('Original data');
ylabel('Volt');
xlabel('bits');
axis([0 Tmax1 -0.5 1.5])

%********************** plot Hamming code **********************
Tmax2=length(HMP);
```

```
t2=linspace(0,Tmax2,Tmax2*2^10);
output2=[];
for ii=1:1:Tmax2
   if HMP(ii)==1
      output2=[output2 ones(1,2^10)];
   else
      output2=[output2 zeros(1,2^10)];
   end
end
figure(2)
plot(t2,output2,'Linewidth',2);grid on;
title('Hamming code');
ylabel('Volt');
xlabel('bits');
axis([0 Tmax2 -0.5 1.5])

%******************  plot Hamming code & Interleaving  ************
Tmax3=length(HMIP);
t3=linspace(0,Tmax3,Tmax3*2^10);
output3=[];
for ii=1:1:Tmax3
   if HMIP(ii)==1
      output3=[output3 ones(1,2^10)];
   else
      output3=[output3 zeros(1,2^10)];
   end
end
figure(3)
plot(t3,output3,'Linewidth',2);grid on;
title('Hamming code & Interleaving');
ylabel('Volt');
xlabel('bits');
```

```
axis([0 Tmax3 -0.5 1.5])

%******************  plot De-Interleaving data  ******************
Tmax7=length(HMRP);
t7=linspace(0,Tmax7,Tmax7*2^10);
output7=[];
for ii=1:1:Tmax7
   if HMRP(ii)==1
      output7=[output7 ones(1,2^10)];
   else
      output7=[output7 zeros(1,2^10)];
   end
end
figure(7)
plot(t7,output7,'Linewidth',2);grid on;
title('De-Interleaving data');
ylabel('Volt');
xlabel('bits');
axis([0 Tmax7 -0.5 1.5])

%*******  plot Hamming decode data or original data  ************
Tmax8=length(dbrp);
t8=linspace(0,Tmax8,Tmax8*2^10);
output8=[];
for ii=1:1:Tmax8
   if dbrp(ii)==1
      output8=[output8 ones(1,2^10)];
   else
      output8=[output8 zeros(1,2^10)];
   end
end
figure(8)
```

```
plot(t8,output8,'Linewidth',2);grid on;
title('Hamming decode data or original data');
ylabel('Volt');
xlabel('bits');
axis([0 Tmax8 -0.5 1.5])
```

เอกสารอ้างอิง

[1] Http://gear.eng.ubu.ac.th/~atipong/DigitalCommu_2_2548

[2] Http://www.s-t.au.ac.th/~mam/ts3230/Lecture10_Error_Detection.pdf

[3] Http://webmailstaff.kmutt.ac.th/~iauaroen/EEE271/codes.ppt#293,11,Hamming%20code

[4] Http://math.sut.ac.th/~phorkaew/files/compcom/lecture08b.pdf

[5] Http://classroom.cs.mju.ac.th/triphop/cs432/09%20-%0Error%20Detection%20Correction_6.pdf

[6] Data communication

[7] Dharma Prakash Agrawal and Qing-An Zeng. **Introduction to Wireless and Mobile Systems.** Second Edition. Ontario : Nelson. 2006

[8] Http://math.sut.ac.th/~phorkaew/files/compcom/lecture05a.pdf

[9] Http://www.navy.mi.th/elecwww/document/magazine/1011.html

[10] รศ.ดร. มนัส สังวรศิลป์ . วรรัตน์ ภัทรอมรกุล. **คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์.** พิมพ์ครั้งที่ 1. นนทบุรี : อินโฟเพรส. พ.ศ. 2543

[11] cptd.chandra.ac.th/.../Error_Control.htm

[12] www.prd.go.th/engineer/Journal/Pusit/dvb/upload/DVB_Technology_p38_45.pdf

[13] http://www.cpe.ku.ac.th/~anan

ประวัติผู้เขียนโครงงาน

ชื่อ มโนรัตน์ จันทร์คำ

ภูมิลำเนา 215/124 ถ.สันนาลุง ต.วัดเกต อ.เมือง จ.เชียงใหม่ 50000

ประวัติการศึกษา

- จบระดับมัธยมศึกษาจากโรงเรียนกาวิละวิทยาลัย
- ปัจจุบันกำลังศึกษาในระดับปริญญาตรีชั้นปีที่ 4 สาขาวิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร

E-mail : cheer_no@hotmail.com